O‘ZBEKISTON RESPUBLIKASI VAZIRLAR MAHKAMASI
HUZURIDAGI TOSHKENT ISLOM UNIVERSITETI

N.Tursunov, R.V. Qobulov, A.I.Dadamuhamedov

«C++ TILIDA DASTURLASH»

FANIDAN O‘QUV QO‘LLANMA

Bakalavriat bosqichi talabalari uchun

Toshkent - 2016

UO`K:

KBK:

C++ tilida dasturlash / Mas'ul muharrir: Z.Z. Xodjimuratova.- Toshkent: TIU nashriyot-matbaa birlashmasi, 2016. -390 b.

MUALLIFLAR:
N.Tursunov, R. V. Qobulov, A.I.Dadamuhamedov

MAS'UL MUHARRIR:
Zuxra Xodjimuratova, *Ilmiy pedagogik kadrlar tayyorlash bo`limi boshlig`i, katta o`qituvchi*

TAQRIZCHILAR:
K.F.Kerimov, *Toshkent axborot texhologiyalari universiteti "Tizimli va amaliy dasturlashtirish" kafedrasi mudiri, t.f.n.,*
M.S. Xodjaeva, *texnika fanlari nomzodi, dotsent*

Qo'llanmada strukturali dasturlashtirishning asosiy tushunchalari, ya'ni o'zgaruvchilar, funksiyalar, strukturalar, dinamik xotira bilan ishlash asoslari bayon qilingan. Shu bilan birga obyektga yo`naltirilgan dasturlashtirishning asosiy tushunchlari bo`lgan sinflar, vorislik, amallarni qo'shimcha yuklash, istisnolar, polimorf sinflardan foydalanish orqali obyektli dasturlashtirish masalalari atroflicha yoritilgan.

Qo'llanma C++ tili bo'yicha nazariy bilimlarni oshirish hamda strukturali va obyektga yo'naltirilgan dasturlash bo`yicha amaliy ko'nikmalarni hosil qilish uchun talabalarga mo`ljallangan bo`lib, undan oliy o'quv yurtlari professor-o'qituvchilari ham foydalanishlari mumkin.

Toshkent islom universiteti Kengashining 2016 yil ____-_____ dagi qarori bilan nashrga tavsiya etilgan (___- son bayonnoma).

ISBN

MUQADDIMA

Har qanday mamlakatning kuchi uning intellectual salohiyati bilan belgilanadi. Salohiyatga esa turli yo`llar bilan erishish mumkin.

I. A. Karimov

Ma'lumki, C++ tili B.Straustrup tomonidan yaratilgan. Bern Straustrup (Bjarne Stroustrup), AT&T Bell Laboratories xodimi, 1980 yili C++ tili ustida ish boshlagan. C++ nomi rasmiy ravishda 1983 yili berildi. Tilning birinchi tijorat versiyasi 1985 yil oktyabrida taqdim etilib, xuddi shu yili Straustrup «C++ dasturlash tili» (The C++ Programming Language) kitobi nashr etildi.

Til standarti ustida 1990 yili ANSI (American National Standards Institute) komiteti ish boshladi. Bu standart oxirgi varianti 1997 yil noyabrida e'lon qilindi. Hozirgi davrda C++ tili eng ommaviy dasturlash tillaridan biriga aylandi.

Ushbu o‘quv qo‘llanma C++ tili nazariy asoslarini o‘rganish va amaliy dasturlash ko‘nikmalarini hosil qilishga qaratilgan.

O‘quv qo‘llanmada C++ tili asosiy tushunchalari, strukturali dasturlash va obyektli dasturlash usullari chuqur yoritilgan.

Qo‘llanmada asosiy e'tibor C++ tili asoslari va dasturlash usullariga qaratilgan bo‘lib, mavzular eng sodda tushunchalardan boshlab ketma-ketlik asosida yoritilgan. Qo‘llanma maqsadi o‘rganuvchi kompyuterda tez mustaqil holda dastur tuzish imkoniga ega bo‘lishi va zamonaviy obyektga yo‘naltirilgan dasturlash texnologiyalarini egallashidan iborat. Qo‘llanma to`rt qismdan iborat. Birinchi qismda Informatika va axborot texnologiyalarining asosiy tushunchalari va algoritmlash asoslari yoritilgan. Ikkinchi qismda C++ tilining tuzilishi yoritilgan bo‘lib, operatorlar, funksiyalar, murakkab turlardan foydalanish hamda dinamik xotira bilan ishlash yoritilgan. Shuningdek, bu qismda C tilidan o‘tgan standart funksiyalardan foydalanish ko‘rsatilgan. Uchinchi qism C++ tilida obyektli dasturlashga bag‘ishlangan. Har bir qism bir-biriga bog‘liqdir. Ushbu qismda

sinflar yaratish, vorislik, polimorf sinflar yaratish, oqimli sinflar kutubxonasi, hamda istisnolarni boshqarish, konteyner sinflar kutubxonasi, iteratorlar, algoritmlardan foydalanish xususiyatlari ko‘rsatilgan. Shuningdek satr sinfi usullari hamda standart algoritmlar batafsil yoritilgan.

Qo‘llanmani yaratishda mundarijada keltirilgan asosiy va qo`shimchaadabiyotlardan keng foydalanildi.

O‘ylaymizki, qo‘llanma bilan tanishgan har bir talaba, magistrant yoki professor-o`qituvchi C++ tilida dasturlash faoliyatida foydali jihatlarga ega bo‘ladi.

I QISM. INFORMATIKA VA AXBOROT TEXNOLOGIYALARINING ASOSIY TUSHUNCHALARI

INFORMATIKA ASOSLARI

Axborot tushunchasi. Informatika kompyuter texnikasini qo‘llashga asoslanib, inson faoliyatining turli sohalarida axborotlarni izlash, to‘plash, saqlash, qayta ishlash va undan foydalanish masalalari bilan shug‘ullanuvchi fandir.

“Informatsiya” so‘zi lotincha informatio so‘zidan olingan bo‘lib, “ma'lumot”,” tushintirish”, ”tasvirlash” degan ma'nolarni anglatadi. Informatika fanining tarixi qadim zamonlarga borib taqaladi. Borliq mavjudodlari bir-birlari orqali ma'lumot almashish bilan bog‘lanadilar.

Tarixiy asarlar, voqea-hodisalar noyob qo‘lyozmalar orqali bizlargacha yetib kelgan. Informatika fani kompyuterlar paydo bo‘lganidan keyin fan sifatida o‘rganila boshlandi. Aslida informatika fani fundamental fanlar qatoriga kiradi. Hozirgi davrda Informatika va axborot texnologiyalari fani metafan darajasiga chiqqan.

Informatika fanining asosiy elementi axborotdir. Axborotlar ikki turga ajratiladi:

1) Uzluksiz (ob-havo, yurak urishi).

2) Diskret, ya'ni uzlukli (baholash).

Axborot ishonchli bo‘lishi kerak. Aks holda uni qayta ishlashga zaruriyat tug‘iladi.

Har xil turdagi ma'lumotlar (matn, tasvir, tovush, video) kompyuterda ikkilik raqamlar (kodlar) ko‘rinishida saqlanadi.

Axborotni o‘lchash uchun unda ishtirok etgan harf, raqam va boshqa belgilar 0 va 1 raqamlaridan iborat bo‘lgan son bilan almashtiriladi. Axborotning eng kichik o‘lchov birligi sifatida *bit* qabul qilingan, u ingliz tilidagi *binary digit* so‘zidan olingan bo‘lib, ikkilik raqami degan ma'noni anglatadi. 1 bayt xotira hajmi maksimal 256 xil ikkilik raqamlarni aks ettirishi mumkin, ya’ni 1 bayt 8 ta 0 yoki 1 raqamlari kombinatsiyalaridan tashkil topishi mumkin: 1 bayt = 256 = 2^8 bit.

Baytdan katta o‘lchov birliklari ham bor:

1 kilobayt = 1024 bayt

1 megabayt = 1024 kilobayt

1 gegabayt = 1024 megabayt

. 1 terabayt = 1024 gegabayt

1 petabayt = 1024 terabayt

Axborotni o‘lchash uchun unda ishtirok etgan harf, raqam va boshqa belgilar 0 va 1 raqamlaridan iborat son bilan almashtiriladi. Masalan, 3 raqami - 11 kabi; 8 raqami - 1000 kabi; A harfi - 11 000 001 kabi ifodalanadi.

Misol. Bir kitob 250 sahifalik bo'lib, har bir sahifada 30 ta satr va har bir satrda 75 ta belgi bo'lsa, kitobdagi axborot hajmini hisoblang.

Yechish. Dastlab, bitta sahifada nechta belgi borligini hisoblaymiz:

$30 \times 75 = 2250$ ta.

Endi kitobdagi belgilarning jami sonini hisoblaymiz:

$2250 \times 250 = 562500$ ta.

Demak, kitobdagi axborot hajmi $562500 \times 8 = 450000$ bit, yoki 562500 bayt, yoki 562500 : 1024 = 525 Kbt, yoki 525 : 1024 = 0,5 Mbt, yoki 0,5 : 1024 = 0,0005 Gbt ekan.

Axborot texnologiyalari. Texnologiya bu mahsulotni qayta ishlash, yaratish usullari majmuidir.

Axborotlarni izlash, to‘plash, saqlash, qayta ishlash hamda foydalanish usullari va vositalariga axborot texnologiyalari deyiladi. Axborot texnologiyalari texnik ta’minotiga kompyuterlar, faks, telefon, telefizor, radio, pochta-telegraff kabi vositalar kiradi.

Zamonaviy informatsion texnologiyalar bu axborot ustida bajarish mumkin bo`lgan zamonaviy texnologiyalardir.

Zamonaviy informatsion texnologiyalarga kompyuter texnologiyalari, kompyuter tarmoqlari, internet texnologiyalari, multimedia texnologiyalari kiradi.

Dasturiy ta'minot. Informatika — bu axborotning nafaqat umumiy xususiyatlari, balki avtomatlashtirilgan ishlov berishning uslublari, jarayonlari va texnik vositalarini o`rganuvchi fandir. Avtomatlashtirilgan ishlov berish jarayonlarining asosini axborotni yig`ish, talqin qilish, saqlash, qayta ishlash va uzatish tashkil qiladi. Bu jarayonlar hisoblash texnikasi, jumladan, kompyuterlar yordamida amalga oshiriladi.

Kompyuter ikkita ajralmas qismdan tashkil topgan bo‘ladi: apparat ta’minoti (hardware) va dasturiy ta'minot (software).

Ular o‘zaro bog‘langan holda yagona uyg‘unlikda ishlaydi va muayyan vazifalarni bajaradi.

Kompyuterning imkoniyatlarini kengaytiradigan va turli vazifalar bajarishini ta'minlaydigan vosita bu albatta dasturiy ta'minotdir. Dasturiy ta'minot odatda kompyuterning qattiq diskida saqlanadi va kompyuter yoqilishi bilan maxsus dastur – operatsion tizim ishga tushadi.

Dasturiy ta'minot ikkita asosiy guruhga bo‘linadi:

1. Tizimli dasturiy ta’minoti.
2. Amaliy dasturiy ta'minot.
3. Instrumental dasturiy ta'minot.

1. Tizimli dasturiy ta'minot – bu barcha uchun yaratilgan va universal bo`lgan dasturlardir.

Tizimli dasturiy ta'minot bu kompyuter ishini boshqaruvchi va har xil yordamchi amallarni bajaruvchi dasturlar. Tizim dasturlar ichida 4 dasturlar guruhlari ajratiladi. Bular: operatsion tizimlar (tizimlar), utilita dasturlar, drayver dasturlar va dastur-qoplamalar.

Operatsion tizimlar kompyuter ishini boshqaradi, har xil dasturlarni kompyuter xotirasiga yuklaydi va bajarilshini nazorat qiladi, fayllar, kataloglar va disklar ustida har xil amallarni bajaradi.

Utilita dasturlar bu operatsion tizim imkoniyatlarini kuchaytiruvchi qo`shimcha dasturlardir.

Drayver dasturlar bu operatsion tizimga tashqi va ichki qurilmalar bilan ishlash imkonini beruvchi dasturlardir.

Dastur-qoplamalar bu operatsion tizimning imkoniyatlaridan chiroyli va foydalanuvchi uchun qulay holda foydalanishni ta'minlovchi dasturlardir.

2. **Amaliy dasturiy ta'minot** – bu amaliy dasturchilar tomonidan biror aniq masalani yechish uchun ishlab chiqligan dasturlardir.

Amaliy dasturiy ta'minot foydalanuvchining aniq bir vazifalari – ilovalarni ishlab chiqish va bajarish uchun mo‘ljallangan. Masalan, matnli ma'lumotlarni yaratish va tahrirlash, rasm va tasvir ma'lumotlarini yaratish yoki o‘zgartirish, ma'lumotlar ombori bilan ishlash, musiqa va video ma'lumotlarni ko‘rib chiqish va tahrirlash.

3. **Instrumental dasturiy ta'minot.** Instumental dasturlar bu yangi dasturlar yaratuvchi dasturlar tizimlaridir.

Instrumental dasturlarga maxsus dasturlash tillari bilan ishlaydigan dasturlar tizimlari kiradi. Ular dasturlash tiliga ko‘ra farqlanadi: C, Basic, C++, Delphi, va boshqalar. Bu dastur tizimlari o‘ziga bir nechta dasturni jamlagan bo‘lib bular: dasturlash tili muharriri, kompilyator, interpretator va boshqa yordamchi dasturlar.

.

Sanoq tizimlari va ulardagi amallar

Nopozitsion tizimlar. Sanoq tizimlari bu sonlarni maxsus belgilar – raqamlar yordamida ifodalash usulidir.

Ko‘pincha, sanoq tizimidagi hamma sonlar (yoki alfavit simvollari) yozuvdagi joylashgan joyiga bog‘liq bo‘ladi. Bunday sanoq tizimi pozitsion deyiladi, aks holda nopozitsion deyiladi.

Misol. Nopozitsion tizim – qadimgi rim alfavitini sonlardagi ifodasi, joylashgan joyidan qat'i nazar quyidagi ko‘rinishga ega bo‘ladi: X={I(1), V(5), X(10), L(50), C(100), D(500), M(1000)}.

Bu tizimdagi yozuv ikki yoqlama konkatenatsiya orqali hosil qilinadi, bunda o‘ng taraflama konkatenatsiya qo‘shilib borish orqali, chap taraflama konkatenatsiya olib tashlash orqalidir (masalan, IV va VI).

Rim sonlarining ko‘rinishi(qavsda – oddiy o‘nlik ekvivalentlari): III(3),IV(4),V(5),VI(6),IX(9),XI(11),DCL(650).

Arifmetik amallarni bajarish joylashuviga qaramaydi (masalan, XIV+IV=XVIII).

Pozitsion sanoq tizimlari. Ko‘pinchalik, sanoq tizimidagi hamma sonlar (yoki alfavit simvollari) yozuvdagi joylashgan joyiga bog‘liq bo‘ladi. Bunday sanoq tizimi pozitsion deyiladi.

Pozitsion sanoq tizimi R simvolli X alfavitdan iborat bo‘lib, har bir son shu alfavitdagi r simvollar orqali tasvirlanadi va qayta ishlanadi. Pozitsion sanoq tizimi r ga asoslangan X soni $(x)_r$ yoki x_r deb belgilanadi.

Hamma pozitsion sanoq tizimlari bir umumiy prinsipga asosan tuziladi: tizimning asosi r-kattaligi aniqlanadi, x ixtiyoriy soni esa r-og‘irlik darajasining kombinatsiyasi ko‘rinishida 0 dan n-darajaga quyidagicha tasvirlanadi:

$(x)_p=x_np^n+x_{n-1}p^{n-1}+\ldots+x_1p^1+x_0p^0$.

Informatikada eng ko‘p ishlatiladigan o‘nlik sanoq tizimidan tashqari sakkizlik va o‘n oltilik sanoq tizimlari ishlatiladi:

10 lik sanoq tizimida 10 ta raqam:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

8 lik sanoq tizimida 8 ta raqam:

0, 1, 2, 3, 4, 5, 6, 7;

16 lik sanoq tizimida 16 ta raqam:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (A=10, B=11, C=12, D=13, E=14, F=15).

Masalan:

$1100_2 = 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 8 + 4 = 12_{10}$

$157_8 = 1*8^2 + 5*8^1 + 7*8^0 = 64+40+7 = 111_{10}$

$A6F_{16} = 10 * 256 + 6 * 16 + 15 * 1 = 2671_{10}$.

Sanoq tizimlari

O‘nlik	Sakkizlik	Ikkilik	O‘n oltilik

0	0	000	0
1	1	001	1
2	2	010	2
3	3	011	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
10	12	1010	A
11	13	1011	V
12	14	1100	S
13	15	1101	D
14	16	1110	E
15	17	1111	F
16	20	10000	10

Butun sonlarni bir sanoq tizimidan ikkinchisiga o‘tkazish. Butun sonlarni bir sanoq tizimidan ikkinchisiga o‘tkazish uchun berilgan sonni o‘tkaziladigan sanoq tizimining asosiga bo‘lamiz. Agar hosil bo‘lgan son bu asosdan katta bo‘lsa, bo‘lishni bu asosdan kichik bo‘lgunga qadar davom ettiramiz. So‘ngra hosil bo‘lgan qoldiqlarni oxiridan boshlab yozib chiqamiz. Natijada hosil bo‘lgan son o‘tkaziladigan sanoq tizimidagi son bo‘ladi.

Misol. O‘nlik sanoq tizimidagi 54 sonini ikkilik sanoq tizimiga o‘tkazing.

Yechilishi:

```
54  | 2
54  27  | 2
 0  26  13  | 2
     1  12  6  | 2
        1   6  3  | 2
            0  2  1
```

Natija: 110110 bo`ladi.

Ikkilik sonlarni sakkizilik yoki o‘n oltilik tizimlarda ifodalash uchun razryadlar uchtalab yoki to‘rttalab ajratiladi va jadval asosida o‘tkaziladi. Agar kerak bo‘lsa, chapdan nol qo‘shish mumkin.

Ikkilik tizimga sakkizlik yoki o‘n oltilik tizimlardan teskari yo‘l bilan o‘tiladi. Quyida keltirilgan jadval ishlatiladi.

Masalan:

$1100_2 = 001100_2 = 14_8$
$157_8 = 001\ 101\ 111_2 = 1101111_2$
$A6F_{16} = 1010\ 0110\ 1111_2 = 1010011011112.$

Tizim asosi			
10	**2**	**8**	**16**
0	0	000	0000
1	1	001	0001
2	-	010	0010
3	-	011	0011
4	-	100	0100
5	-	101	0101
6	-	110	0110
7	-	111	0111
8	-	-	1000
9	-	-	1001
10	-	-	1010
11	-	-	1011
12	-	-	1100
13	-	-	1101
14	-	-	1110
15	-	-	1111

Kasr sonlar. O‘nlik sanoq tizimidagi kasr sonlarga o‘tishni ko`ramiz.

O‘nlik sanoq tizimidan r-sanoq tizimiga o‘tish yo‘llari:

Berilgan X sonining butun qismini ajratib olamiz, avval $[X]_{10}$ ni, ya'ni son butun qismini r ga bo‘lamiz, so‘ngra kasr qismini r-ga bo‘lamiz, to shu sondan kichik son qolguncha va hosil bo‘lgan qoldiqlari r-sanoq tizimidagi son butun qismini hosil qiladi.

Berilgan X sonining kasr qismini ajratib olamiz, sonning $\{X\}_{10}$ kasr qismi (mantissasini), toki nolga teng bo‘lgan mantissa hosil bo‘lguncha, yoki kerakli $\{X\}_r$ hosil bo‘lguncha yangi asos r ga ko‘paytiriladi. Son kasr qismi $\{X\}_r$ ning ko‘rinishi birinchi ko‘paytirishdan to oxirgi ko‘paytirishga hosil bo‘lgan sonlarning butun qismidan hosil qilinadi va natija quyidagi ko‘rinishga ega bo‘ladi: $(X)_r=[X]_p, \{X\}_p$.

Misol. Topilsin: $12,8=?_2$.

Yechimi:

Butun qismini olib o‘tamiz: $12_{10}=1100_2$;

Kasr qismini olib o‘tamiz: 0,8*2=1,6; 0,6*2=1,2; 0,2*2=0,4; 0,4*2=0,8; $0,8_{10}=0,1100110\ldots_2$;

Natija: $12,8=1100,1100110011\ldots_2$.

Ikkilik sonlarni sakkizlik yoki o‘n oltilik tizimlarda ifodalash uchun butun va kasr razryadlar uchtalab yoki to‘rttalab ajratiladi va jadval asosida o‘tkaziladi. Agar kerak bo‘lsa, chapdan butun qismiga va o‘ngdan kasr qismiga nol qo‘shish mumkin.

Ikkilik tizimga sakkizlik yoki o‘n oltilik tizimlardan teskari yo‘l bilan o‘tiladi.

Misol. Bir sanoq tizimidan ikkinchisiga o‘tish:

1. 2 likdan 8 likka o‘tish:

$$101,10111_2=\frac{101}{5_8},\ \frac{101}{5_8},\ \frac{110_2}{6_8}=5,56_8;$$

2. 8 likdan 2 likka o‘tish:

$$6,24_8=\frac{6}{110_2},\ \frac{2}{010_2},\ \frac{4_8}{100_2}=110,\ 0101_2;$$

3. 2 likdan 16 likka o‘tish:

$$101,10111_2=\frac{0101}{5_{16}},\ \frac{1011}{11(B)_{16}}\ \frac{1000_2}{8_{16}}=5,B8_{16};$$

Kompyuterda kasr sonlar, yarim logarifmik ko‘rinishda, aniqrog‘i suzuvchi vergulli ko‘rinishda tasvirlanadi:

$x=m*b^e$,

bunda m – mantissa, b – daraja, e – tartib. Odatda b, teng 2, mantissa normallashtiriladi, uning absolyut qiymati [1/2,2] intervalga joylashtiriladi.

IEEE (Institute of Electrical and Electronics Engineers) tashkiloti Floating Point Standard standartiga ko‘ra suzuvchi vergulli sonlar xotirada 32 razryaddan iborat bo‘lib, ulardan 24 bit mantissa va 8 bit darajaga ajratiladi. Bu taxminan 7 xonali o‘nli sonni saqlashga imkon beradi. O‘nlik sanoq tizimida berilgan ratsional sonlar ikkilik razryadga o‘tkazilganda buzilishi mumkin. Masalan 1/3, 0,1 sonlari aniq ikkilik ko‘rinishga ega emas.

Eng kichik 1,0 ga qo‘shilganda undan katta son hosil qiluvchi suzuvchi vergulli son *mashinali epsilon* deyiladi va quyidagicha belgilanadi e_{mash}. Mashinali epsilon kompyuter arifmetikasi xatoligini belgilaydi: agar x va y – ikki musbat son bo‘lsa va x >y, ular yig‘indisini quyidagicha yozish mumkin:

x+y=x(1+y/x).

Agar y/x< e_{mash} bo‘lsa, summa x ga teng bo‘ladi.

Bu xildagi sanoq tizimi formasidagi “qulaysizliklar” “xavfli” vaziyatlarni yuzaga keltirishi mumkin: suzuvchi vergulli sonlarni qo‘shishda (barcha amallar qo‘shish orqali amalga oshiriladi) navbatdagi mantissalarni qo‘shish uchun tartibni tenglashtirish yuzaga keladi, darajalarni tenglashtirishda esa kichik razryadlarni yo‘qotish vujudga keladi, masalan, bunday vaziyat bir “katta sonni” ikkinchi “eng kichik songa” qo‘shishda yuzaga keladi.

Algoritmlash asoslari

Algoritm tushunchasi. Algoritm so‘zi buyuk matematik-olim, vatandoshimiz Abu Muso al-Xorazmiyning nomi bilan bog‘liq bo‘lib, u birinchi bo‘lib arab raqamlaridan foydalangan holda arifmetik amallarni bajarish qoidasini bayon etgan. Har qanday qo‘yilgan masalani kompyuterda yechish uchun oldin uning yechish usulini tanlab, keyin uning algoritmini ishlab chiqish kerak bo‘ladi.

Algopitm — ijrochi uchun ma'lum bir masalani yechishga qaratilgan ko‘rsatmalarning aniq ketma-ketligi.

Al-Xorazmiyning nomini lotincha ifodasi — Algorithmi.

Algoritm — informatika va matematikaning asosiy tushunchalaridan hisoblanadi.

Algoritm ijrochisi — algoritmda ko‘rsatilgan buyruqlarni bajara oladigan abstrakt yoki real (texnik, biologik yoki biotexnik) tizim.

Odatda ijrochi algoritmning maqsadi haqida hech narsa bilmaydi.

Informatikada algoritmni universal ijrochisi – kompyuter.

Algoritmlarga xos xususiyatlar:

• Oddiy harakatlar;

• Buyruqlar tizimi. Har bir ijrochi faqatgina ushbu ijrochi tushunadigan buyruqlarni (ya'ni, ijrochi bajaradigan buyruqlar ro`yxatiga mansublarni) bajara oladi. Ijrochi buyruqlarni bajarish jarayonida oddiy harakatlarni bajaradi.

Ta'rif. Algoritm deb, qo‘yilgan masalani yechish uchun ma'lum qoidaga binoan bajariladigan amallarning chekli qadamlar ketma-ketligiga aytiladi. Har qanday algoritm ma'lum ko‘rsatmalarga binoan bajariladi va bu ko‘rsatmalarga buyruq deyiladi.

Algopitmning xossalari. Har qanday algoritm quyidagi xossalarga ega bo‘lishi lozim:

Diskpetlik. Jarayon bir nechta ketma-ket buyruqlar asosida rivojlanadi. Algopitm ijrochisi masalani yechish jarayonini alohida va sodda qadamlar ketma-ketligini bajarish deb tushunishi kerak.

Tushunarlilik. Algoritmni bajarish uchun qulay bo‘lgan matn ko‘rinishida tasvirlash. Algoritm ijrochisi buyruqlar ketma-ketligini qanday bajarishni aniq bilishi kerak.

Aniqlik. To‘g‘ri algoritm uchun har bir bajaruvchi tomonidan bir xil natijalarni olinishi. Algoritmning har bir qoidasi, undagi amallar va buyruqlar bir ma'noli bo‘lishi kerak. Shu xossaga asosan algoritm ijrochisi buyruqlar ketma-ketligini mexaniq bajarish imkoniyatiga ega bo‘ladi.

Ommaviylik. Algoritmni har xil ma'lumotlar to‘plami uchun haqiqiyligi. Masalani yechish algoritmi umumiy hollar uchun yaratiladi, ya'ni faqatgina

boshlang`ich qiymatlari bilan farqlanuvchi bir turdagi masalalar sinfi uchun tuziladi. Bunda boshlang‘ich qiymatlar algoritmning qiymatlar qabul qilishi mumkin bo‘lgan sohadan olinadi.

Natijaviylik. Ma'lum bir sondagi qadamlarni bajargandan so‘ng aniq natijani olishligi. Har qanday algoritmning ijrosi oxir-oqibat ma'lum bir yechimga kelishi kerak.

Tugallanganligi. Ketma ket bajariluvchi harakatlar soni cheksiz emas, ularni sanash mumkin. Ma'lum bir qadamdan algoritmning ijrosi tugashi lozim.

Algoritm turlari. Algoritmning asosiy uchta turi mavjud: chiziqli, tarmoqlanuvchi va takrorlanuvchi (siklik) algoritmlar.

Chiziqli algoritm deb, hech qanday shartsiz faqat ketma-ket bajariladigan jarayonlarga aytiladi.

Tarmoqlanuvchi algoritm deb, shartlarga muvofiq bajariladigan ko‘rsatmalardan tuzilgan algoritmga aytiladi.

Takrorlanuvchi algoritm deb, biron-bir shart tekshirilishi yoki biron parametrning har xil qiymatlari asosida algoritmda takrorlanish yuz beradigan jarayonlarga aytiladi.

Kompyuterda masalalar yechish bosqichlari. Kompyuterda masalalar yechish bosqichlari quyidagilardan iborat:

- Masalaning qo‘yilishi va maqsadlarni aniqlash;
- Matematik modelini yaratish;
- Tizimli tahlil qilish;
- Yechish usulini tanlash;
- Masalani yechish algoritmini ishlab chiqish;
- Dasturlashtirish;
- Dasturni kiritish va xatolarini tuzatish;
- Masalani bevosita kompyuterda yechish va olingan natijalarni tahlil qilish.

Algoritm tasvirlash usullari. Algoritm tasvirlash usullari quyidagilardan iborat:

• so‘zlar yordamida (og‘zaki nutqda ishlatiladigan so‘zlar yordamida, tabiiy tilda);

• grafik usulda (bloksxema yordamida);

• dastur ko‘rinishida (dasturlash tillariga oid umumlashtirlgan xizmatchi so‘zlar, operator va funksiyalar yordamida).

• formulalar yordamida (matematik formulalardan foydalangan holda, analitik ko‘rinishda).

• makrotildan foydalangan holda (dasturlovchi va kompyuterga tushunarli bo‘lgan makrokomandalar yordamida).

• jadval ko‘rinishida (mantiqiy algebra elementlaridan foydalangan holda).

Algoritmlarni *so‘zlar yordamida* tasvirlashda bajariladigan buyruqlar va ko‘rsatmalar ketma-ket og‘zaki nutqda ishlatiladigan so‘zlar orqali yoziladi.

Masalan, Ikki sonning eng katta umumiy bo‘luvchisini (EKUB) topish algoritmi quyidagicha yozilishi mumkin:

1. Ikkita sonni kiriting;
2. Agarda bu sonlar teng bo‘lsa, u holda ulardan birini javob sifatida oling va ishni to‘xtating, aks holda esa davom eting;
3. Ikkita son ichida kattasini aniqlang;
4. Katta va kichik sonlarning ayirmasini katta son bilan almashtiring;
5. Algoritmni 2-qadamdan boshlab qaytaring.

Psevdokod tushunchasi. Umumlashma operatorning soxta koddagi formal bo‘lmagan ifodasi bu ifoda mazmunini umumiy tarzda ochib beradigan ixtiyoriy gap bilan tabiiy tilda amalga oshiriladi. Bunday ifodani shakllantirishga qo‘yiladigan yagona formal talab quyidagichadir: bu gap bitta yoki bir nechta grafik (bosma) satrni to‘liq egallashi hamda nuqta (yoki buning uchun maxsus ajratilgan boshqa biron belgi bilan tugallanishi) lozim.

Har bir formal bo‘lmagan umumlashma operator uchun tuzilmaviy dasturlash konstruksiyasining kompozitsiyasi hamda boshqa umumlashma operatorlar yordamida uning ishi mantiqini ifodalab beradigan (uning mazmunini detallashtirib

beradigan) alohida tavsif yaratilishi kerak. Bunday tavsifning sarlavhasi sifatida detallashtirilayotgan umumlashma operatorning formal bo‘lmagan ifodasi kelishi kerak. Tuzilmaviy dasturlashning asosiy konstruksiyalari quyidagi ko‘rinishda taqdim etilishi mumkin:

Algoritmlarni blok-sxema ko‘rinishda tasvirlash. Algoritmlarni blok-sxema ko‘rinishda tasvirlash qulay va tushunarli bo‘lgani uchun eng ko‘p ishlatiladi. Algoritmlarni grafik usulda tasvirlashda har bir amal bir yoki bir nechta harakatni ifodalovchi o‘zaro bog‘liq funksional bloklar ketma-ketligi orqali tasvirlanadi.

Algoritmning bunday tasvirlash usuli algoritm sxemasi yoki blok-sxema deb ataladi.

Blok-sxemada har bir harakat turini (boshlang‘ich qiymatlarni kiritish, ifodalar qiymatlarini hisoblash, shartlarni tekshirish, amallarni takrorlashni boshqarish, qayta ishlashni tugatish va h.k.) ma'lum bir geometrik figura orqali ifodalanadi.

Bunda algoritmdagi har bir ko‘rsatma o‘z shakliga ega. Masalan: parallelogram ko‘rinishdagi belgi ma'lumotlarni kiritish va chiqarish; to‘g‘ri to‘rtburchak belgisi hisoblash jarayonini; romb belgisi shartlarning tekshirilishini bildiradi.

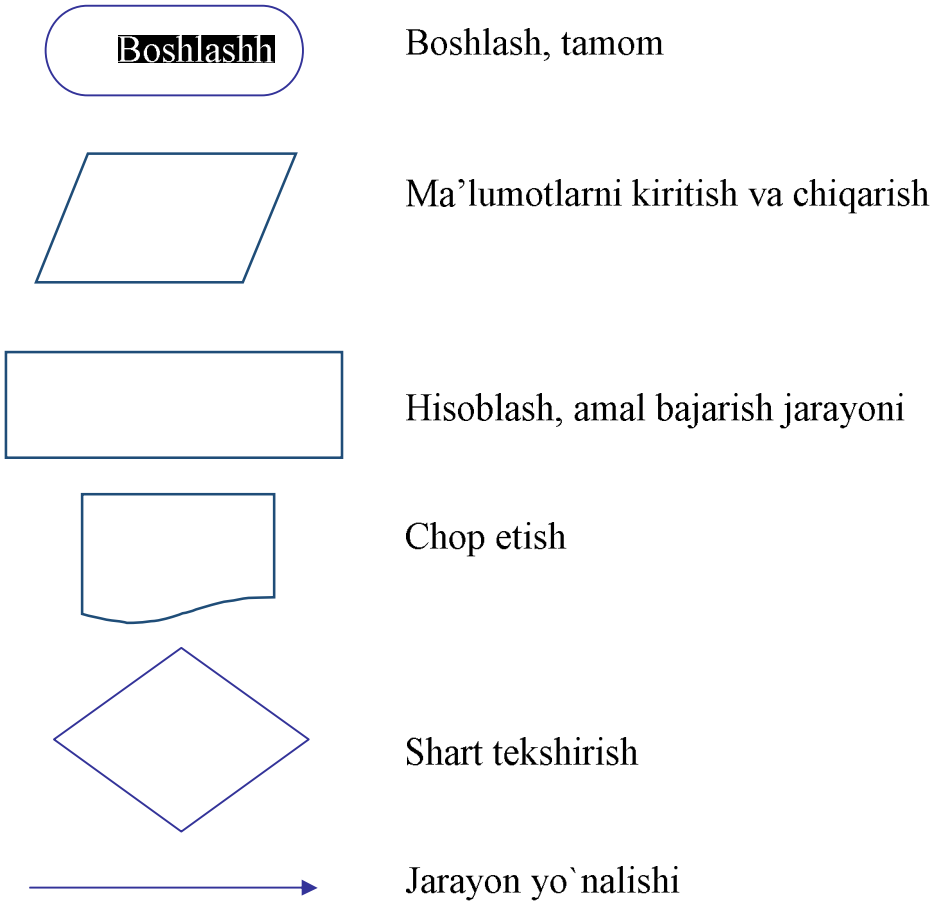
Blokli belgilar (geometrik figuralar) chiziqlar orqali bog‘lanadi (bunda qaysi amal oldin, qaysinisi keyin bajarilishi ko‘rsatiladi).

“Oddiy harakat” belgisi orqali formulalar, hisob-kitob, o‘zlashtirish amallari ifodalaniladi. Bir nechta amallarni alohida yoki bitta belgi orqali ifodalash mumkin.

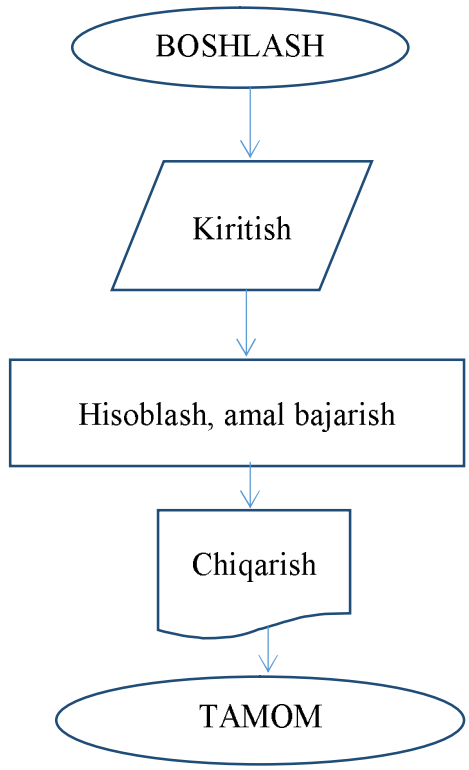
"Shart tekshirish" bloki orqali amallar bajarilish yo‘nalishi shart bajarilishi asosida ko‘rsatiladi. Bunday blokning har birida savol, shart yoki munosabat ko‘rsatiladi.

"Sikl" bloki amallarni takrorlash uchun ishlatiladi. Blok ichida siklning boshi va oxirini ko‘rsatuvchi parametr (masalan, i), parametrning o‘zgarish qadami ko‘rsatiladi.

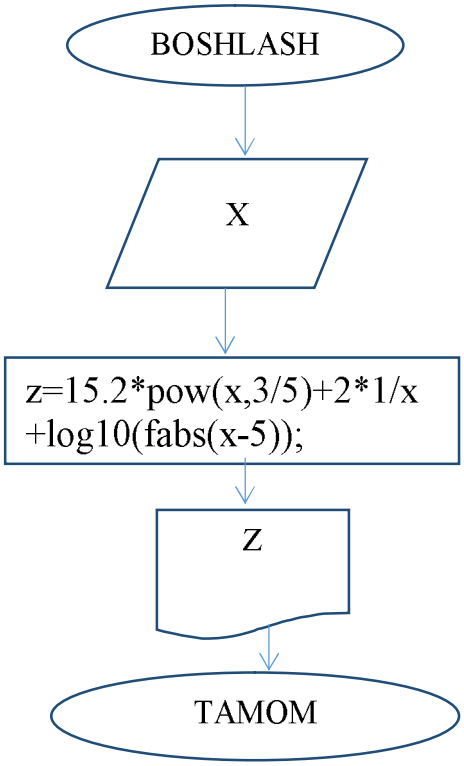
"Yordamchi algoritmga murojaat" bloki alohida va mustaqil ishlovchi qism dastur va yordamchi algoritmlarga murojaatni bildiradi.



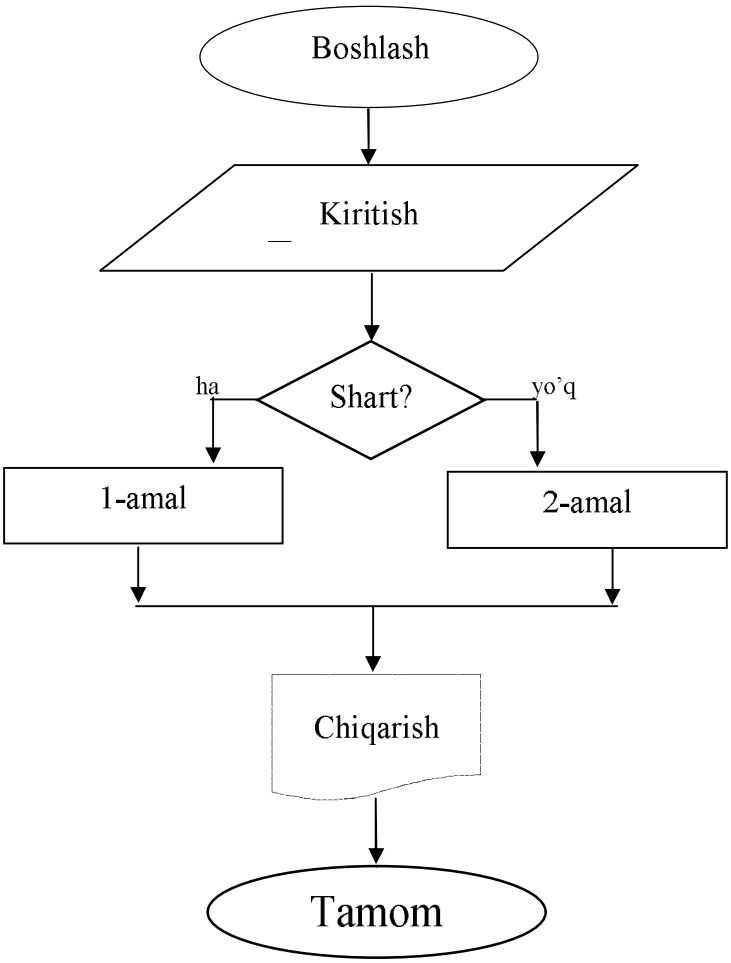
Chiziqli algoritm. Chiziqli algoritm umumiy sxemasi:



Misol. z=15.2$\sqrt[5]{x^3}+2x^{-1}+lg|x-5|$ hisoblash algoritmini tuzing

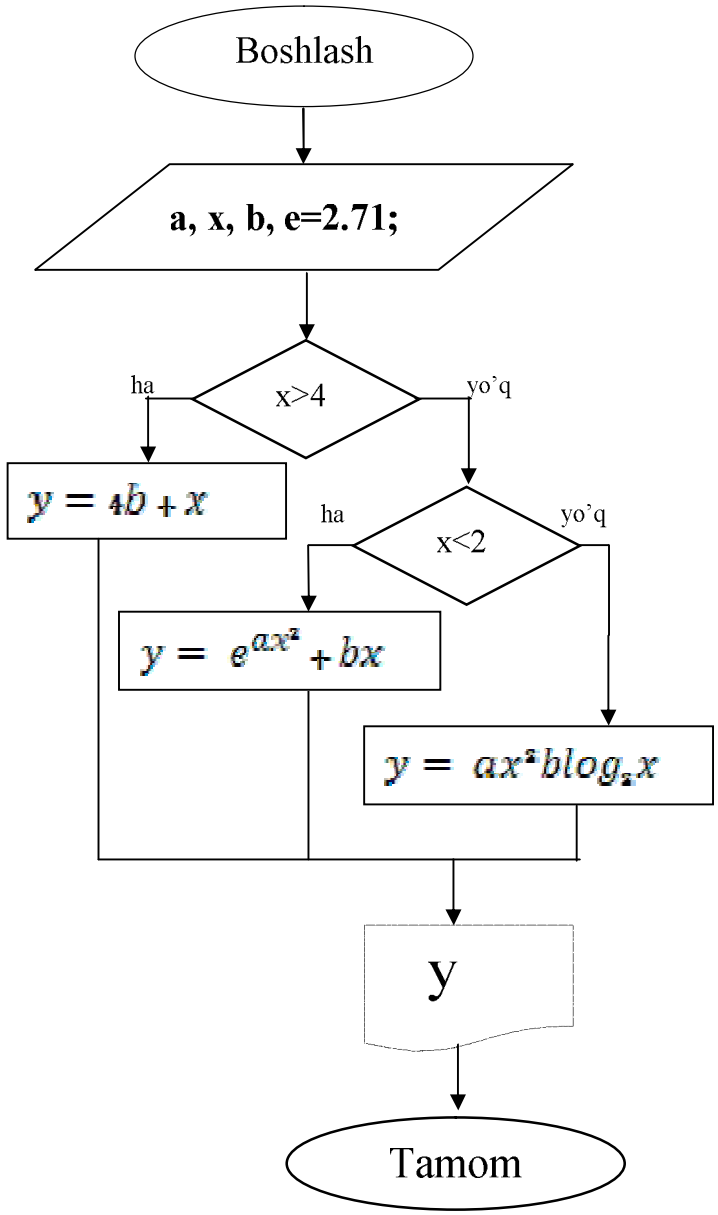


Tarmoqlanuvchi algoritm. Tarmoqlanuvchi algoritm umumiy sxemasi:

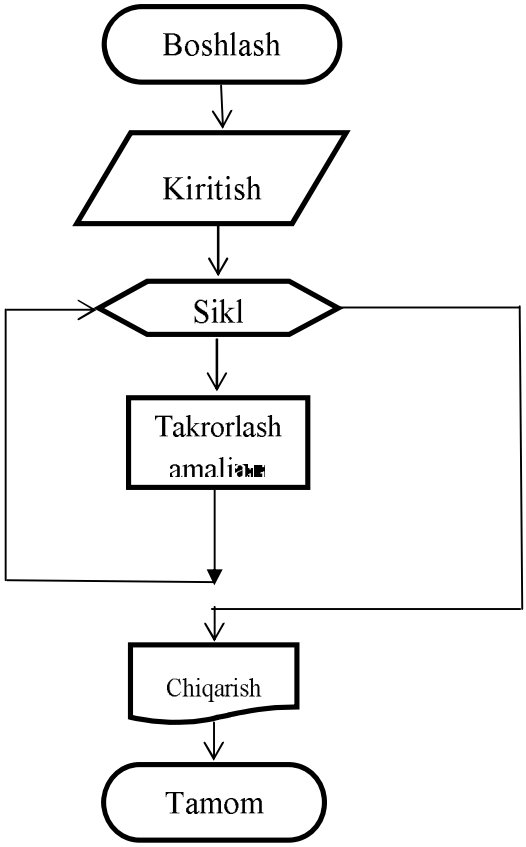


Misol. Funksiya algoritmini tuzing.

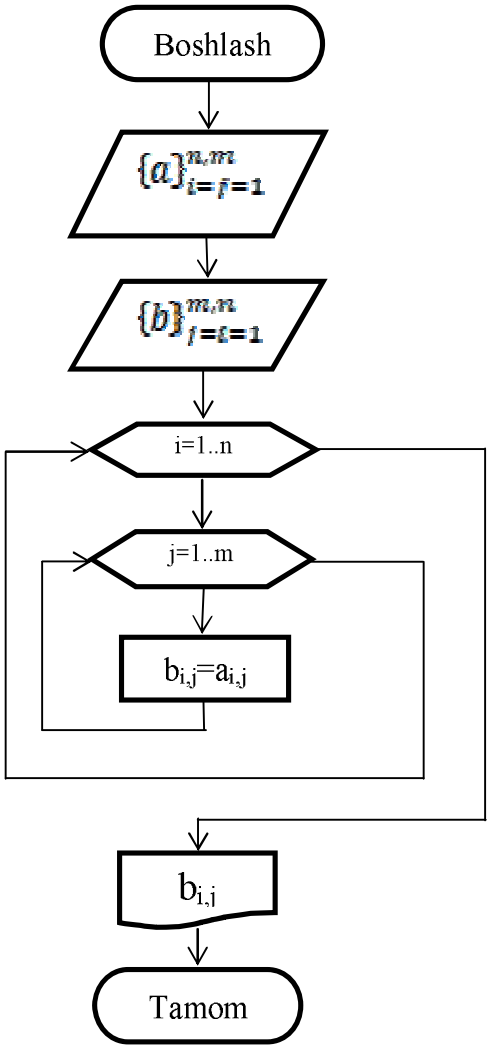
$$y = \begin{cases} ax^2 b\log_2 x & agar \quad 2 \le x \le 4 \\ 4b + x & agar \quad x > 4 \\ e^{ax^2} + bx & agar \quad x < 2 \end{cases}$$



Takrorlanuvchi algoritm. Takrorlanuvchi algoritm umumiy sxemasi:



Misol. a(nxm) matritsadan qator va ustun elementlarini almashtirib, yangi matritsa hosil qilish algoritmini tuzing:



Nazorat savollari:

1. Informatika fani nimani o‘rganadi?
2. Informatika fanining asosiy vazifalari.
3. Axborot tushunchasi.
4. Axborot qanday sifatlarga ega bo‘lishi lozim?
5. Axborot kompyuterda qanday ko‘rinishda tasvirlanadi?
6. Axborot o‘lchov birliklari.
7. Kompyuter dasturiy ta’minoti qanday turlari mavjud?
8. Sanoq tizimlari turlari.
9. Algoritm tushunchasi va xossalari.
10. Algoritmlarni tasvirlash usullari.

Misollar:

1. Kvadrat tenglamani yechish algoritm va bloksxemasini tuzing.

2. Quyidagi y=min(a,b,c) funksiyani hisoblash algoritm va blok sxemasini tuzing.

3. Berilgan eps aniqlikda umumiy xadi 1/n bo‘lgan ketma-ketlik yig‘indisini hisoblovchi algoritm va blok sxema tuzing.

4. Umumiy xadi x/n bo‘lgan ketma-ketlik n ta xadi yig‘indisini hisoblovchi algoritm va bloksxema tuzing.

II QISM. TIL ASOSLARI

Dastur strukturasi

C++ tilidagi dastur quyidagi strukturaga ega:

#< preprotsessor direktivasi>

<funksiyalar>

preprotsessor direktivasi – dasturdagi almashtirishlarni uning kompilyatsiyasigacha boshqaradi.

1) **#define** – matndagi almashtirishlar qonun-qoidasini aniqlaydi.

2) **#include <sarlavha fayl nomi>** - standart kutubxonalar bilan birga sarlavha faylidagi funksiyalarni ishlatishga mo‘ljallangan.

C++ tilida tuzilgan dastur obyektlar, funksiyalar, o‘zgaruvchilar va boshqa elementlardan tashkil topadi.

Misol:

```
#include <iostream.h>
int main( ){
 cout << "Salom!\n";
 return 0;
}
```

Natija:

Salom!

Preprotsessorning komandasi. Dasturda include – preprotsessor komandasi bo‘lib, u quyidagicha tarjima qilinadi: «Bu komandani ortidan fayl nomi keladi. Ushbu nomdagi faylni topish va fayldagi mazmunni dasturning joriy qismiga yozish lozim».

Burchakli qavs ichidagi faylni mos fayllar joylashtirilgan barcha papkalardan izlash lozimligini ko‘rsatadi. Agarda kompilyator to‘g‘ri sozlangan bo‘lsa, burchakli qavslar iostream faylini sizning kompilyatoringiz uchun fayllarni o‘zida saqlovchi papkadan izlashi kerakligini ko‘rsatadi. Iostream (input–output stream – kiritish–chiqarish oqimi) faylida ekranga ma'lumotlarni chiqarish jarayonini ta'minlaydigan cout obyekti aniqlangan. Birinchi qator bajarilgandan so‘ng

iostream fayli joriy dasturga biriktiriladi. Preprotsessor kompilyatordan keyin yuklanadi va funt (#) belgisi bilan boshlanuvchi barcha qatorlarni bajaradi, dastur kodlarini kompilyasiyaga tayyorlaydi.

Asosiy funksiya. Dasturning asosiy kodi main() funksiyasini chaqirish bilan boshlanadi. C++ tilidagi har bir dastur main() fuksiyasini o‘zida saqlaydi. Funksiya bu bir yoki bir necha amalni bajaruvchi dastur blokidir. Odatda funksiyalar boshqa funksiyalar orqali chaqiriladi, lekin main() funksiyasi alohida xususiyatga ega bo‘lib, u dastur ishga tushirilishi bilan avtomatik tarzda chaqiriladi.

Ayrim kompilyatorlar main() funksiyasini void turidagi qiymat qaytaradigan qilib e'lon qilish imkonini beradi. C++ da bundan foydalanmaslik kerak, chunki hozirda bunday uslub eskirgan. main() funksiyasini int turini qaytaradigan qilib aniqlash lozim va buning hisobiga funksiyaning oxiriga return 0 ifodasi yoziladi.

Barcha funksiyalar ochiluvchi figurali qavs ({) bilan boshlanadi va (}) yopiluvchi qavs bilan tugaydi. Figurali qavslarni ichida joylashgan barcha satrlar funksiya tanasi deb aytiladi.

Izoh operatori. Dasturni tushunarli bo‘lishi uchun izohlardan foydalanish lozim. Izohlar kompilyator tomonidan tushirib qoldiriladigan dasturning alohida satrida yoki butun bir blokida qo‘llaniladi. Izoh operatori bajarilmaydigan operator bo‘lib, ko‘p satrli yoki bir satrli bo‘lishi mumkin. Izohlar /* va */, belgilar bilan ajratilgan bo‘lsa, ichki joylashgan bo‘lolmaydi. Agar kod fragmentini /* va * / belgilar bilan ishlamaydigan qilib bo‘lmaydi, chunki uning o‘zi */ va */ simvollarni o‘z ichiga olishi mumkin.

Ifoda. C++ tilida ifodalar biror-bir hisoblash natijasini qaytaruvchi boshqa ifodalar ketma-ketligini boshqaradi yoki hech nima qilmaydi (nol ifodalar). C++ tilida barcha ifodalar nuqtali vergul bilan yakunlanadi. Qiymat berish amali (=) o‘zidan chap tomondagi operandga o‘ng tomondagi operandlar ustida bajarilgan amallar natijasini o‘zlashtiradi.

Bo‘sh joy (probel) belgisi. Bo‘sh joy belgilariga nafaqat probel, balki yangi satrga o‘tish va tabulyasiya belgilari ham kiradi. Bo‘sh joy belgilari dasturning o‘qilishliligini ta'minlaydi.

Blok va murakkab operator. har bir ifoda agar u nuqta vergul belgisi bilan tugasa operator hisoblanadi. Figurali qavsga olingan operatorlar va ta'riflar ketma-ketligi blok deyiladi. Agar blok faqat operatorlardan iborat bo‘lsa, murakkab opreator deyiladi.

Kiritish-chiqarish operatorlari. Ma'lumotni kiritish uchun *cin,* chiqarish uchun *cout* operatoridan foydalaniladi. Klaviaturadan ma'lumotni kiritish uchun *cin* so‘zini, undan so‘ng kiritish operatorini (>>) kiritish lozim. Ekranga ma'lumotni chiqarish uchun *cout* so‘zini, undan so‘ng chiqarish operatorini (<<) kiritish lozim. C++ kompilyatori (<<) belgisini bitta operator deb qaraydi. Kursorni keyingi qatorga o‘rnatish uchun *endl* (qator oxiri) simvoli yoki yangi qator simvoli (*\n*) dan foydalanish mumkin.

Dasturni to‘xtatib turish. Dastur oxirida system("pause") ekranda ma'lumotlarni ushlab turish uchun ishlatiladi. Ekranni tozalash uchun system("cls") komandasidan foydalanish mumkin.

Misol:

```
#include <iostream>
using namespace std;
int main( ) {
cout << "Salom!\n";
system("pause");
system("cls");
cout << "Hayr!\n";
system("pause");
return 0;
}
```

Standart nomlar fazosi – std. Dasturda ishlatiluvchi C++ standart kutubxonalariga kirgan hamma nomlar standart nomlar fazosiga jamlangan. Standart nomlar fazosi std deb nomlanadi. Direktiva using namespace std standart nomlar fazosini global nomlar fazosiga ulaydi. Standart va global nomlar fazosini ulamasdan dastur tuzish uchun ruxsat berish operatoridan foydalanish lozim.

Misol:

```
#include <iostream>
#include <string>
int main(void) {
std::string name;
std::cout<<"Name?"<<std::endl;
std::cin>>name;
std::cout<<"\n Hello " <<name.c_str()<< std::endl;
return 0;
}
```

O‘zgaruvchilar va konstantalar

Alfavit. C++ alfavitiga quyidagi simvollar kiradi:

- Katta va kichik lotin alfaviti harflari (A,B,..,Z,a,b,…,z)
- O`nlik raqamlar: 0,1,2,3,4,5,6,7,8,9
- Maxsus simvollar: “ , {} | [] () + - / % \ ; ‘ . : ? < = > _ ! & * # ~ ^
- Ko‘rinmaydigan simvollar ("umumlashgan bo‘shlik simvollari"). Leksemalarni o‘zaro ajratish uchun ishlatiladigan simvollar (misol uchun bo‘shlik, tabulyasiya, yangi qatorga o‘tish belgilari).

Izohlarda, satrlarda va simvolli konstantalarda boshqa literallar, masalan rus harflari ishlatilishi mumkin.

C++ tilida olti xil turdagi leksemalar ishlatiladi: erkin tanlanadigan va ishlatiladigan identifikatorlar, xizmatchi so‘zlar, konstantalar(konstanta satrlar), amallar(amallar belgilari), ajratuvchi belgilar.

Xizmatchi so‘zlar. Identifikatorlar lotin harflari,ostki chiziq belgisi va sonlar ketma-ketligidan iborat bo‘ladi. Identifikator lotin harfidan yoki ostki chizish belgisidan boshlanishi lozim. Katta va kichik harflar farqlanadi.

Tilda ishlatiluvchi, ya'ni dasturchi tomonidan o‘zgaruvchilar nomlari sifatida ishlatish mumkin bo‘lmagan identifikatorlar xizmatchi so‘zlar yoki kalit so‘zlar deyiladi. Kompilyatorning texnik dokumentatsiyasida barcha zahiralangan so‘zlarning ro‘yxati turadi.

O‘zgaruvchilar. O‘zgaruvchi nomi ostiga chizish belgisi yoki lotin harfidan boshlanuvchi lotin harflari, arab raqamlari va ostiga chizish belgilari ketma-ketligi,

ya'ni identifikatordir. O‘zgaruvchi nomi xizmatchi so‘zlardan farqli bo‘lishi kerak. Katta va kichik harflar farqlanadi.

C++ tilida o‘zgaruvchini aniqlash uchun kompyuterga uning turi (masalan, int, char yoki float) hamda ismihhaqida haqida ma'lumot beriladi. Bu axborot asosida kompilyatorga o‘zgaruvchi uchun qancha joy ajratish lozim va bu o‘zgaruvchida qanday turdagi qiymat saqlanishi mumkinligi haqida ma'lumot aniq bo‘ladi.

O‘zgaruvchilarning quyidagi turlari mavjuddir: **bool** (mantiqiy), **char (**belgi**), int (**butun**), float (**haqiqiy**), double (**ikkilangan haqiqiy**).**

Asosiy turlardan tashqari quyidagi modifikatorlardan foydalaniladi: **short (**qisqa), **long (**uzun), **unsigned (**ishorasiz).

Modifikatorlardan tur sifatida foydalanish mumkin, bu holda **short** turi **short int** turiga, **long** turi **long int** turiga, tur **unsigned** turiga **unsigned int** turiga mos keladi.

O‘zgaruvchi short turiga ega bo‘lsa, har doim 2 bayt, long turiga ega bo‘lsa, 4 bayt egallaydi. Agar int turiga ega bo‘lsa, hajmi kompilyatorga bog‘liq.

Butun sonlar ta'riflanganda ko‘rilgan turlar oldiga **unsigned** (ishorasiz) ta'rifi qo‘shilishi mumkin. Ishorali, ya'ni **signed** turidagi sonlarning eng katta razryadi son ishorasini ko‘rsatish uchun ishlatilsa **unsigned** (ishorasiz) turdagi sonlarda bu razryadli sonni tasvirlash uchun ishlatiladi.

Bu ta'rif qo‘shilgan butun sonlar ustida amallar mod 2^n arifmetikasiga asoslangandir. Bu yerda n soni int turi xotirada egallovchi razryadlar sonidir. Agar ishorasiz k soni uzunligi int soni razryadlar sonidan uzun bo‘lsa, bu son qiymati k mod 2^n ga teng bo‘ladi. Ishorasiz k son uchun –k amali 2^n – k formula asosida hisoblanadi.

O‘zgaruvchilar ta'rifining sodda shakli quyidagicha:

<tur> <o‘zgaruvchilar_nomlari_ro‘yxati>;

o‘zgaruvchilarni ta'riflashda boshlang`ich qiymatlarini ko‘rsatish mumkin:

<tur> <o‘zgaruvchilar_nomlari_ro‘yxati >= <initsializator>;

Bu usul initsializatsiya deyiladi.

Nomlangan konstantalar. O‘zgaruvchilar kabi nomlangan o‘zgarmaslar ham ma'lumotlarni saqlash uchun mo‘ljallangan xotira yacheykalarini o‘zida ifodalaydi. O‘zgaruvchilardan farqli ravishda ular dasturni bajarilishi jarayonida qiymati o‘zgarmaydi. O‘zgarmas e'lon qilinishi bilan unga qiymat berish lozim, keyinchalik bu qiymatni o‘zgartirib bo‘lmaydi.

Nomlangan konstantalar quyidagi shaklda kiritiladi:

const tur konstanta_nomi=konstanta_qiymati.

Initsializatsiyali ta'rif. O‘zgaruvchilarni ta'riflaganda avtomatik initsializatsiya qilishga imkon beruvchi quyidagi shakllardan foydalanish mumkin:

<tur> <o‘zgaruvchi_nomi>(<initsializator>);

<tur> < o‘zgaruvchi_nomi >(< o‘zgaruvchi_nomi >);

Masalan:

```
#include<iostream>
using namespace std;
int main() {
const int a=1;
const int aa(2);
cout<<a<<" "<<aa<<endl;
double b=0.5;
double bb(b);
cout<<b<<" "<<bb<<endl;
system("pause");
return 0;
}
```

Natija:

1 2

0.5 0.5

Konstantalar

Ma'lumotlarning butun son turi. Konstanta bu o‘zgartirish mumkin bo‘lmagan qiymatdir. Konstantalar butun, simvolli haqiqiy, mantiqiy turlari mavjud.

Butun sonlar o‘nlik, sakkizlik yoki o‘n oltilik sanoq tizimlarida berilishi mumkin.

- O‘nlik konstantalar o‘nlik raqamlari ketma-ketligidan iborat bo‘lib, birinchi raqami 0 bo‘lishi kerak emas.

- Sakkizlik konstantalar 0 bilan boshlanuvchi sakkizlik raqamlaridan iborat ketma-ketlikdir.

- O‘n oltilik konstantalar – 0x yoki 0X bilan boshlanuvchi o‘n oltilik raqamlar ketma-ketligidir.

Agar oxiriga l yoki L qo‘yilsa, uzun konstanta.

Agar oxiriga u yoki U qo‘yilsa, ishorasiz konstanta.

Ma'lumotlarning haqiqiy son turi. Haqiqiy konstanta ikki ko‘rinishda bo‘lishi mumkin: fiksirlangan nuqtali va suzuvchi nuqtali ko‘rinishda tasvirlanadi. Ma'lumotlarning haqiqiy son turi olti qismdan iborat bo‘lishi mumkin: butun qism, nuqta, kasr qism, e yoki E belgisi, o‘nlik daraja , F yoki f suffikslari.

Agar oxiriga f yoki F qo‘yilsa, **float** turiga ega.

Agar oxiriga l yoki L qo‘yilsa, **long double** turiga ega.

Mantiqiy konstantalar **true**(rost) va **false**(yolg‘on) qiymatlardan iborat. Ichki ko‘rinishi **false** – 0, ixtiyoriy boshqa qiymat **true** deb qaraladi.

Simvolli konstanta. Simvolli konstanta – apostrofga olingan bitta yoki bir nechta simvol. Slesh '\' simvolidan boshlangan simvollar eskeyp yoki boshqaruvchi simvollar deyiladi

Belgili o‘zgarmaslar odatda bir bayt joyni egallaydi va bu 256 xil belgini saqlash uchun yetarlidir. Char turi qiymatlarini 0..255 sonlar to‘plamiga yoki ASCII belgilar to‘plamiga interpretatsiya qilish mumkin.

ASCII belgilari deganda kompyuterlarda qo‘llaniladigan standart belgilar to‘plami tushuniladi. ASCII – bu American Standard Code for Information Interchange (Amerikaning axborot almashinishi uchun standart kodi) degan ma'noni anglatadi.

Maxsus belgilar. C++ kompilyatorida matnlarni formatlovchi bir nechta maxsus belgilardan foydalaniladi. (Ulardan eng ko‘p tarqalgani jadvalda keltirilgan). Bu belgilarni dasturda ishlatishda «teskari slesh»dan foydalanamiz. Teskari sleshdan keyin boshqaruvchi belgi yoziladi.

Maxsus belgilar axborotlarni ekranga, faylga va boshqa chiqarish qurilmalariga chiqarishda formatlash uchun qo‘llaniladi.

Maxsus '\' simvolidan boshlangan simvollar eskeyp simvollar deyiladi. Simvolli konstanta qiymati simvolning kompyuterda qabul qilingan sonli kodiga tengdir.

ESC (eskeyp) simvollar jadvali:

Yechilishi	Ichki kodi	Simvoli(nomi)	Ma'nosi
\a	0x07	bel (audible bell)	Tovush signali
\b	0x08	bs (bascspase)	Bir qadam qaytish
\f	0x0C	ff (form feed)	Sahifani o`tkazish
\n	0x0A	lf (line feed)	Qatorni o`tkazish
\r	0x0D	cr (carriage return)	Karetkani qaytarish
\t	0x09	ht (horizontal tab)	Gorizontal tabulyatsiya
\v	0x0B	vt (vertical tab)	Vertikal tabulyatsiya
\\	0x5C	\ (bacslash)	Teskari chiziq
\'	0x27	‘ (single out)	Apostrof (oddiy qavs)
\"	0x22	" (double quote)	Ikkilik qavs
\?	0x3F	? (question mark)	Savol belgisi
\000	000	ixtiyoriy (octal number)	Simvol sakkizlik kodi
\xhh	0xhh	ixtiyoriy (hex number)	Simvol o`n oltilik kodi

Satrli konstanta. Satrli konstantalar C++ tili konstantalariga kirmaydi, balki leksemalari alohida turi hisoblanadi. Shuning uchun adabiyotlarda satrli konstantalar satrli leksemalar deb ham ataladi.

Satrli konstanta bu ikkilik qavslarga olingan ixtiyoriy simvollar ketma-ketligidir.

Satrlar orasiga eskeyp simvollar ham kirishi mumkin. Bu simvollar oldiga \ belgisi qo‘yiladi.

Misol uchun :

```
#include <iostream>
using namespace std;
int main(){
cout<<"\n Bu satr \n uch qatorga \n joylashadi";
}
```

Natija:

Bu satr
uch qatorga
joylashadi

Satr simvollari xotirada ketma-ket joylashtiriladi va har bir satrli konstanta oxiriga avtomatik ravishda kompilyator tomonidan '\0' simvoli qo‘shiladi. Shunday satrning xotiradagi hajmi simvollar soni +1 baytga tengdir.

Ketma-ket kelgan va bo‘shlik, tabulyasiya yoki satr oxiri belgisi bilan ajratilgan satrlar kompilyasiya davrida bitta satrga aylantiriladi. Misol uchun:

```
#include <iostream>
using namespace std;
int main(){
cout<<"Salom" " Toshkent" ;
}
```

Natija:
Salom Toshkent
Bu qoidaga bir necha qatorga yozilgan satrlar ham bo‘ysunadi.

Misol uchun :

```
#include <iostream>
using namespace std;
int main(){
cout<<"Salom"
 " Toshkent" ;
}
```

Natija:
Salom Toshkent

Amallar

Arifmetik amallar. C++ tilida beshta asosiy matematik amallar qo‘llaniladi: qo‘shish (+), ayirish (-), ko‘paytirish (*), butun songa bo‘lish (/) va modul bo‘yicha bo‘lish (%)(qoldiqni olish). Arifmetik amallar binar va unar amallarga ajratiladi. Binar amallarga + qo‘shish, – ayirish, * ko‘paytirish, / bo‘lish va % modul olish amallari kiradi.

Butun sonlar bo‘linganda natija yaxlitlanadi. Agar modul olish amali musbat sonlarga qo‘llansa, natija musbat, manfiy sonlarga qo‘llanilsa, natija kompilyatorga bog‘liq.

Unar amallarga unar minus – va unar +; inkrement ++ va dekrement— amallari kiradi. Inkrement va dekrement amallari prefiks ya'ni ++i, postfiks ya'ni i++ ko‘rinishda ishlatishi mumkin.

Ifodalardagi farq shundan iboratki, prefiks shaklda oldin qiymat oshiriladi, so‘ngra ishlatiladi, postfiks shaklda oldin qiymat foydalaniladi, so‘ngra oshiriladi.

Murakkab ifodalarda qaysi amal birinchi navbatda bajarilishi amallar ustuvorligiga bog‘liq. Agarda ikkita matematik ifodaning prioriteti teng bo‘lsa, ular chapdan o‘ngga qarab ketma-ket bajariladi.

Additiv amallarining ustuvorligi multurlikativ amallarining ustuvorligidan pastroqdir.Unar amallarning ustuvorligi binar amallardan yuqoridir.

Nisbat amallari. Nisbat amallari ikkita qiymatni teng yoki teng emasligini aniqlash uchun ishlatilib katta >, kichik <, katta yoki teng >=, kichik yoki teng <=, teng ==, teng emas != amallaridan iborat. Solishtirish amallari qiymati – mantiqiy konstanta. Bunday amallar taqqoslash ifodasi doimo true (rost) yoki false (yolg‘on) qiymat qaytaradi. Munosabat amallari arifmetik turdagi operandlarga qo‘llanilsa qiymatlari 1 ga teng, agar nisbat bajarilsa va aksincha 0 ga tengdir.

Katta >, kichik <, katta yoki teng >=, kichik yoki teng <= amallarining ustuvorligi bir xildir.

Teng == va teng emas != amallarining ustuvorligi o‘zaro teng va kolgan nisbat amallaridan pastdir.

Mantiqiy amallar. Mantiqiy amallar || (dizyunksiya), && (konyunksiya), !(inkor) amallaridan iborat. Inkor ! amali unar qolganlari binar amallardir.

Operandlar qiymati mantiqiy konstantalardan tashqari arifmetik konstantalar va ifodalar bo‘lishi mumkin. Bu holda ifoda qiymati nolga teng bo‘lsa yolg‘on, aks holda rost hisoblanadi.

Ustuvorlik oshib borishi tatibi: dizyunksiya, konyunksiya, inkor.

Razryadli amallar. Razryadli amallar | (razryadli dizyunksiya); & (razryadli konyunksiya); ^(razryadli XOR); !(razryadli inkor); chapga surish<<; o‘ngga surish >> amallaridan iborat.

Razryadli amallar natijasi butun sonlarni ikkilik ko‘rinishlarining har bir razryadiga mos mantiqiy amallarni qo‘llashdan hosil bo‘ladi. Chapga N pozitsiyaga surish bu operand qiymatini ikkining N chi darajasiga ko‘paytirishga mos keladi. Agar operand musbat bo‘lsa, N pozitsiyaga o‘ngga surish chap operandni ikkining N-darajasiga bo‘lib, kasr qismini tashlab yuborishga mosdir. Agarda operand qiymati manfiy bo‘lsa, ikki variant mavjuddir: arifmetik siljitishda bo‘shatilayotgan razryadlar ishora razryadi qiymati bilan to‘ldiriladi, mantiqiy siljitishda bo‘shatilayotgan razryadlar no`llar bilan to‘ldiriladi.

Razryadli inkor amali unar qolgan amallar binar amallarga kiradi.

Razryadli surish amallarining ustuvorligi o‘zaro teng, razryadli inkor amalidan past, qolgan razryadli amallardan yuqoridir.

Qiymat berish amali. Oddiy qiymat berish amali binar amal bo‘lib, chap operandi, odatda, o‘zgaruvchi o‘ng operandi ifodaga teng bo‘ladi:

O‘zgaruvchi_nomi = ifoda;

O‘zlashtirish amali (=) o‘zidan chap tomonda turgan operand qiymatini tenglik belgisidan o‘ng tomondagilarni hisoblangan qiymatiga almashtiradi.

Murakkab qiymat berish amali unar amal bo‘lib quyidagi ko‘rinishga ega: **O‘zgaruvchi_nomi amal= ifoda;**

Bu yerda amal quyidagi amallardan biri * , / , % , + , - , & ,^ , | , << , >>.

Shartli amal. Shartli amal ternar amal deyiladi va uchta operanddan iborat bo‘ladi: <1-ifoda>? <2-ifoda> : <3-ifoda>.

Shartli amal bajarilganda avval 1-ifoda hisoblanadi. Agar 1-ifoda qiymati 0 dan farqli bo‘lsa, 2- ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi, aks holda 3-ifoda hisoblanadi va qiymati natija sifatida qabul qilinadi.

Shuni aytish lozimki, shartli ifodadan har qanday ifoda sifatida foydalanish mumkin. Agar F FLOAT turga,a N – INT turga tegishli bo‘lsa, (N > 0) ? F : N ifoda N musbat yoki manfiyligidan qat'i nazar DOUBLE turiga tegishli bo‘ladi. Shartli ifodada birinchi ifodani qavsga olish shart emas.

Imlo belgilari amal sifatida. C++ tilida ba'zi bir imlo belgilari ham amal sifatida ishlatish mumkin. Bu belgilardan oddiy **()** va kvadrat **[]** qavslardir. Oddiy

qavslar binar amal deb qaralib, ifodalarda yoki funksiyaga murojaat qilishda foydalaniladi. Murakkab ifodalarni tuzishda ichki qavslardan foydalaniladi. Kvadrat qavslardan massivlarga murojaat qilishda foydalaniladi. Vergul simvolini ajratuvchi belgi deb ham qarash mumkin, amal sifatida ham qarash mumkin. Vergul bilan ajratilgan amallar ketma-ketligi bir amal deb qaralib, chapdan o‘ngga hisoblanadi va oxirgi ifoda qiymati natija deb qaraladi.

Turlar bilan ishlash

typedef kalitli so‘zi. unsigned short int kabi kalit so‘zlarni ko‘p martalab dasturda yozilishi zerikarli va diqqatvozlik talab qilganligi uchun C++ tilida bunday turlarni **typedef** kalitli so‘zi yordamida psevdonimini (tahallusini) tuzish imkoniyati berilgan. **typedef** so‘zi turni aniqlash ma'nosini bildiradi.

Psevdonim tuzishda turning nomi yangi tuziladigan tur nomidan farqli bo‘lishi lozim. Bunda birinchi typedef kalitli so‘zi, keyin mavjud tur nomi, undan so‘ng esa yangi nom yoziladi.

Misol:

```
#include <iostream>
using namespace std;
typedef  unsigned short int ushort;
int main(){
ushort a = 5;
ushort b = 10;
ushort s = a* b;
cout << "Yuza:" << s << endl;
}
```

Natija:
Yuza: 50

Turlar bilan ishlovchi amallar. Turlarni o‘zgartirish amali quyidagi ko‘rinishga ega:

(tur_nomi) operand;

Bu amal operandlar qiymatini ko‘rsatilgan turga keltirish uchun ishlatiladi. Operand sifatida kostanta, o‘zgaruvchi yoki qavslarga olinga ifoda kelishi mumkin. Misol uchun **(long)6** amali konstanta qiymatini o‘zgartirmagan holda

operativ xotirada egallagan baytlar sonini oshiradi. Bu misolda konstanta turi o‘zgarmagan bo'lsa, **(double)6** yoki **(float)6** amali konstanta ichki ko‘rinishini ham o‘zgartiradi. Katta butun sonlar haqiqiy turga keltirilganda sonning aniqligi yo‘qolishi mumkin.

Yuqorida ko‘rilgan turlarni o‘zgartirish amali shakli kanonik deb ataladi. Bundan tashqari funksional shakli qo‘llaniladi. Turlarni keltirish kanonik shakli:

tur_nomi (operand);

Masalan:

```
#include <iostream>
using namespace std;
int main(){
int x=1.7+1.8;
int y=int(1.7)+int(1.8);
cout << x <<" "<< y << endl;
}
```

Natija:

3 2

Turlarni keltirish funksional shaklini murakkab bir necha so‘zdan iborat turlarga qo‘llab bo‘lmaydi.

sizeof amali operand sifatida ko‘rsatilgan obyektning baytlarda xotiradagi hajmini hisoblash uchun ishlatiladi. Amali sizeof preprotsessor qayta ishlash jarayonida bajariladi, shuning uchun bu amali protsessor vaqtini olmaydi.

Bu amalning ikki ko‘rinishi mavjud:

sizeof ifoda

sizeof (tur)

Misol uchun:

```
#include <iostream>
using namespace std;
int main(){
cout<<sizeof 3.14<<" ";
cout<<sizeof 3.14f<<" ";
cout<<sizeof 3.14L<<" ";
cout<<sizeof(char)<<" ";
cout<<sizeof(double)<<endl;
}
```

Natija:
8 4 12 1 8

Turlarni keltirish. Turlarni keltirish (**type casting**) ma'lum turdagi o‘zgaruvchi boshqa turdagi qiymat qabul qilganda foydalaniladi. Ba'zi turlar uchun keltirish avtomatik ravishda bajariladi. Avtomatik turlarni keltirish o‘zgaruvchi turi hajmi qiymatni saqlashga yetarli bo‘lganda bajariladi. Bu jarayon kengaytirish *(**widening**)* yoki yuksaltirish *(**promotion**)* deb ataladi, chunki, kichik razryadli tur katta razryadli turga kengaytiriladi. Bu holda turlarni avtomatik keltirish xavfsiz deb ataladi. Masalan, int turi char turidagi qiymatni saqlashga yetarli, shuning uchun turlarni keltirish talab qilinmaydi. Teskari jarayon toraytirish *(**narrowing**)* deb ataladi, chunki qiymatni o‘zgartirish talab etiladi. Bu holda turlarni avtomatik keltirish xavfli hisoblanadi. Masalan, haqiqiy turni butun turga keltirilganda, kasr qism tashlab yuboriladi.

Quyidagi turlarni keltirish ketma-ketligi xavfsiz hisoblanadi.

signed char -> short -> int -> long

unsigned char -> unsigned short -> unsigned int -> unsigned long

float -> double -> long double

Agar turlarni avtomatik keltirish xavfli bo‘lsa, kompilyasiya jarayonida ma'lumot chiqariladi.

Amallarda turlarni avtomatik keltirish. Binar arifmetik amallar bajarilganda turlarni keltirish quyidagi qoidalar asosida amalga oshiriladi:

short va char turlari int turiga keltiriladi;

Agar operandlar biri long turiga tegishli bo‘lsa, ikkinchi operand ham long turiga keltiriladi va natija ham long turiga tegishli bo‘ladi;

Agar operandlar biri float turiga tegishli bo‘lsa, ikkinchi operand ham float turiga keltiriladi va natija ham float turiga tegishli bo‘ladi;

Agar operandlar biri double turiga tegishli bo‘lsa, ikkinchi operand ham double turiga keltiriladi va natija ham double turiga tegishli bo‘ladi;

Agar operandlar biri long double turiga tegishli bo‘lsa, ikkinchi operand ham long double turiga keltiriladi va natija ham long double turiga tegishli bo‘ladi.

Ifodalarda turlarni avtomatik keltirish. Agar ifodada short va int turidagi o‘zgaruvchilar ishlatilsa, butun ifoda turi int ga ko‘tariladi. Agar ifodada biror o‘zgaruvchi turi long bo‘lsa, butun ifoda turi long turga ko‘tariladi. Jimlik bo‘yicha hamma butun konstantalar int turiga ega deb qaraladi. Hamma butun konstantalar oxirida L yoki l simvoli turgan bo‘lsa, long turiga ega.

Agar ifoda float turidagi operandga ega bo‘lsa, butun ifoda float turiga ko‘tariladi. Agar biror operand double turiga ega bo‘lsa, butun ifoda turi double turiga ko‘tariladi.

Amallar ustuvorligi. Murakkab ifodalarda qaysi amal birinchi navbatda bajarilishi amal ustuvorligiga bog‘liq.

Agarda ikkita matematik ifodaning ustuvorligi teng bo‘lsa, ular chapdan o‘ngga qarab ketma-ket bajariladi.

Amallar ustuvorligi oshib borishi tartibida: vergul, qiymat olish, shartli, mantiqiy, razryadli, munosabat, surish, arifmetik, unar, dumaloq va kvadrat qavslar. Multurlikativ amallar ustuvorligi additivlarnikidan yuqori.

Amallar ustuvorligi jadvali

Rang	Amallar	Yo‘nalish
1	() [] -> :: .	Chapdan o‘ngga
2	! ~ + - ++ -- & * (tur) sizeof new delete tur()	O‘ngdan chapga
3	.* ->*	Chapdan o‘ngga
4	* / % (multurlikativ binar amallar)	Chapdan o‘ngga
5	+ - (additiv binar amallar)	Chapdan o‘ngga
6	<< >>	Chapdan o‘ngga
7	< <= >= >	Chapdan o‘ngga
8	= !=	Chapdan o‘ngga
9	&	Chapdan o‘ngga
10	^	Chapdan o‘ngga
11	|	Chapdan o‘ngga
12	&&	Chapdan o‘ngga
13	||	Chapdan o‘ngga
14	?:(shartli amal)	O‘ngdan chapga
15	= *= /= %= += -= &= ^= |= <<= >>=	O‘ngdan chapga
16	, (vergul amali)	Chapdan o‘ngga

Chap tomonli ifoda. Ifoda chap tomonli **lvalue** deyiladi, agar uni qiymat olish amali chap tomonida ishlatish mumkin bo‘lsa. Aks holda u o‘ng tomonli

rvalue ifoda deyiladi. Masalan, oddiy o‘zgaruvchi nomi chap tomonli, nomlangan konstanta o‘ng tomonli ifodadir. Aniqrog‘i l-qiymat deb, quyidagi shartlarga mos keluvchi obyekt tushuniladi:

- obyekt adresini olish mumkin;
- obyekt qiymatini olish mumkin;
- bu qiymatni o‘zgartirish mumkin (agar obyekt e'lonida const spetsifikatoridan foydalanilmasa).

Aksincha, r-qiymat – bu qiymati hisblanadigan ifoda yoki vaqtinchalik obyekt bo‘lib, uning adresini olib bo‘lmaydi.

Masalan:

```
#include<iostream>
using namespace std;
int main() {
int a=3,b=2;
cout<<(a&&b)<<" "<<(a&b)<<endl;
cout<<(a||b)<<" "<<(a|b)<<endl;
cout<<(!a)<<" "<<(~a)<<endl;
system("pause");
return 0;
}
```

Natija:

1 2
1 3
0 -4

Masalan:

```
#include<iostream>
#define zero 0
using namespace std;
int main() {
const int maska=2;
int a=zero,b=maska;
bool c=true;float d;
a+=2*3-5/2-5%2;cout<<"a="<<a<<endl; //a=3
b&=(3^2)|!2;cout<<"b="<<b<<endl;       //b=0
c=true||!true;cout<<"c="<<c<<endl;       //c=1
d=a>b ? 1.5:2.5;cout<<"d="<<d<<endl; //d=1.5
return 0;
}
```

Turlar maksimal va minimal qiymatlari. Turli turlar masimal va minimal qiymatlari **<limits.h>** faylidagi konstantalarda saqlanadi.

CHAR_BIT CHAR_MIN,CHAR_MAX konstantalar char turidagi bitlar soni va char turi maksimal va minimal qiymatini bildiradi.

SHRT_MIN,SHRT_MAX,INT_MIN, INT_MAX konstantalar **short**, **int**, **long** turlari maksimal va minimal qiymatini bildiradi.

Quyidagi dasturda **unsigned char**, **unsigned short**, **unsigned int**, **unsigned long** turlari maksimal va minimal qiymati ekranga chiqariladi:

```
#include <iostream>
using namespace std;
int main(){
cout<<"UCHAR_MAX="<<UCHAR_MAX<<endl;
cout<<"USHRT_MAX="<<USHRT_MAX<<endl;
cout<<"UINT_MAX="<<INT_MAX<<endl;
cout<<"ULONG_MAX="<<ULONG_MAX<<endl;
system("pause");
return 0;
}
```

Nazorat savollari:

1. Asosiy turlarni ko‘rsating.
2. O‘zgaruvchi ta'rifini keltiring.
3. Razryadli amallarni keltiring
4. Butun sonli va haqiqiy turlarni qanday farqi bor?
5. Ishorasiz unsigned turining xossalarini ko‘rsating.
6. Ishorasiz unsigned short int va long int turlarining o‘zaro farqi nimada?
7. Birinchi qaysi funksiya bajariladi?
8. Shartli amalning umumiy ko‘rinishi.
9. Turlarni keltirish qoidalari.
10. Izohlar bir necha qatorda yozilishi mumkinmi?

Misollar:

1. Matematik amallardan foydalanishni ko‘rsatuvchi dastur tuzing.
2. Mantiqiy amallardan foydalanishni ko‘rsatuvchi dastur tuzing.

3. Nisbat amallardan foydalanishni ko‘rsatuvchi dastur tuzing.

4. Munosabat amallardan foydalanishni ko‘rsatuvchi dastur tuzing.

5. Son absolyut qiymatini shartli amal yordamida hisoblovchi dastur tuzing.

OPERATORLAR

Tanlash operatorlari

Shartli operator. Shartli tanlash operatorida avval shart tekshiriladi. Agar shart rost bo‘lsa, birinchi, aks holda ikkinchi operator (agar u bo‘lsa) bajariladi.

if (ifoda) 1- operator else 2- operator eki **if (ifoda) 1-operator**

Misol. Simvol ikkilik ekanligini tekshiruvchi dastur:

```
#include<iostream>
using namespace std;
int main() {
char ch='1';
if (ch=='1' || ch == '0') cout<< "Simvol binary";
else cout<<"Simvol no binary";
system("pause");
return 0;
}
```

Natija:

Simvol binary

if operatori orqali murakkab konstruksiyalarni hosil qilish. if–else konstruksiyasida ifodalar blokida ixtiyoriy operatorlarni ishlatishda hech qanday chegara yo‘q. Shu jumladan, ifodalar bloki ichida yana if–else operatorlarini ishlatish mumkin. Bu holda bir nechta **if** operatoridan iborat ichma-ich konstruksiya hosil bo‘ladi.

```
if (1–shart)
{
 if (2–shart)
  1–ifoda
 else
 {
  if (3–shart)
   2–ifoda
  else
   3–ifoda
```

```
  }
}
 else
 4–ifoda;
```

Ushbu bir nechta if operatoridan tashkil topgan konstruksiya quyidagi tartibda ishlaydi: agarda 1–shart va 2–shart rost bo‘lsa, 1–ifoda bajariladi. Agarda 1–shart rost va 2–shart yolg‘on natija qaytarsa, u holda 3–shart tekshiriladi va agarda bu shart rost bo‘lsa, 2–ifoda, yolg‘on bo‘lsa esa 3–ifoda bajariladi. Va eng oxiri, agarda 1–shart yolg‘on bo‘lsa, 4–ifoda bajariladi. Bunday murakkab konstruksiyaga misol quyida keltirilgan.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int ball=57,max_ball=100,baho;
float d=(float)ball/max_ball;
if (d>0.85) baho=5; else{
if (d>0.71) baho=4; else{
if (d>0.55) baho=3; else baho=2;
}
}
cout<<"baho="<<baho<<endl;
system("pause");
return 0;
}
```

Natija:

baho=3

Kalit bo‘yicha tanlash operatori. C++ tilida switch operatori if operatoridan asosiy farqi shuki, unda bir yo‘la bir nechta shart tekshiriladi.

Kalit bo‘yicha tanlash operatori quyidagi shaklga ega:

```
switch(<ifoda>) {
      case <1-qiymat>:<1-operator>
       ...
      default: <operator>
       … }
```

Kalit bo‘yicha tanlash operatorida berilgan ifoda qiymati biror **case** qiymatiga mos kelsa, shundan keyingi hamma operatorlar bajariladi, aks holda **default** (agar u bo‘lsa) so‘zidan keyingi operator bajariladi.

Agarda bu operator mavjud bo‘lmasa boshqaruv **switch** bloki tanasidan chiqadi va keyingi dastur satrlariga beriladi.

Operator identifikatori o‘rnida ham ixtiyoriy operator yoki ifoda hamda operator va ifodalarning ketma-ketligini ishlatish mumkin. Lekin bu yerda mantiqiy amallar yoki taqqoslash ifodalarini ishlatish mumkin emas.

Operator yoki ifodalardan keyin **break** operatori qo‘llanilmasa, joriy **case** operatoridan keyingi **case** blokidagi barcha ifodalar bajariladi. Ko‘p hollarda bunday vaziyatda xatolik ro‘y beradi.

Misol. Simvol ikkilik ekanligini tekshiruvchi dastur:

```
#include<iostream>
using namespace std;
int main() {
char ch='1';
switch (ch) {
case '0':
case '1': cout<< "Simvol binary";break;
default: cout<<"Simvol no binary";
}
return 0;
}
```

Natija:
Simvol binary

Misol tariqasida bahoni son miqdoriga qarab aniqlash dasturini ko‘ramiz:

```
#include <iostream>
using namespace std;
int main(){
int baho=3;
switch(baho){
case 2:cout<<"\n yomon";break;
case 3:cout<<"\n o'rta";break;
case 4:cout<<"\n yahshi";break;
case 5:cout<<"\n alo";break;
default:cout<<"\n noto'g'ri kiritilgan";
};
```

```
cout<<endl;
system("pause");
return 0;
}
```
Natija:
o'rta

Sikl operatorlari

Sikllarni tashkil etish. Har qanday dasturning strukturasi tarmoqlanish va takrorlashlar to‘plamining kombinatsiyasidan iborat bo‘ladi. Qator masalalarni yechish uchun ko‘pincha bitta amalni bir necha marotaba bajarish talab qilinadi. Amaliyotda bu rekursiyalar va iterativ algoritmlar yordamida amalga oshiriladi. Iterativ jarayonlar – bu amallar ketma-ketligini zaruriy sonda takrorlanishidir.

Old shartli takrorlash quyidagi ko‘rinishga ega:

while (<shart>) <sikl tanasi> ;

Old shartli takrorlashda oldin shart tekshiriladi, keyin to shart yolg‘on bo‘lguncha sikl tanasi bajariladi.

Misol. Sikl yordamida 10 gacha bo`lgan sonlar yig`indisini hisoblash dasturi:

```
#include<iostream>
using namespace std;
int main() {
int i= 1,s=0;
while(i<=10) s+=i++;
cout<<s<<endl;
system("pause");
return 0;
}
```
Natija:
55

Misol: Quyidagi dasturda counter o‘zgaruvchisi qiymati toki 5 ga teng bo‘lgunga qadar oshib boradi:

```
#include <iostream>
using namespace std;
int main(){
int counter=0;
while(counter<5){
```

```
counter ++;
cout << "counter :"<< counter << ". \n" ;
}
cout<<"sikl tugadi. Counter:"<<counter<<".\n";
system("pause");
return 0;
}
```

Natija:

counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
sikl tugadi. Counter: 5.

while operatori orqali murakkab konstruksiyalarni tuzish. while operatori shartida murakkab mantiqiy ifodalarni ham qo‘llash mumkin. Bunday ifodalarni qo‘llashda && (mantiqiy ko‘paytirish), || (mantiqiy qo‘shish) hamda ! (mantiqiy INKOR) kabi amallardan foydalaniladi. Quyida while operatori konstruksiyasida murakkabroq shartlarni qo‘yilishiga misol keltirilgan.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int n=25;
while (n>5&&n<50){
cout<<n<<endl;
cin>>n;
}
system("pause");
return 0;
}
```

So`ng shartli takrorlash. Ayrim hollarda while operatori yordamida sikllarni tashkil etishda uning tanasidagi amallar umuman bajarilmasligi mumkin. Chunki siklni davom etish sharti har bir iteratsiyadan oldin tekshiriladi. Agarda boshlang‘ich berilgan shart to‘g‘ri bo‘lmasa, sikl tanasining birorta operatori ham bajarilmaydi. Bu hollarda so`ng shartli takrorlashdan foydalanish lozim.

So`ng shartli takrorlash quyidagi ko‘rinishga ega:

```
do
<sikl tanasi>;
```

while (<shart>);

Oldin sikl tanasi bajarilib, keyin shart tekshiriladi. Takrorlash to shart yolg‘on bo‘lguncha davom etadi.

Misol. So`ng shartli takrorlash yordamida 10 gacha bo`lgan sonlar yig`indisini hisoblash dasturi:

```
#include<iostream>
using namespace std;
int main() {
int i=1,s=0;
do s+=i++; while(i<=10);
cout<<s<<endl;
system("pause");
return 0;
}
```

Natija:
55

Misol. do…while konstruksiyasi qo`llanilgan boshqa bir dastur:

```
#include <iostream>
using namespace std;
int main(){
int counter;
cout<<"How many hellos ?";
cin >>counter;
do{
cout << "hello \n";
counter--;
}
while(counter>0);
cout << "Counter is :" << counter <<endl;
system("pause");
return 0;
}
```

Natija:
How many hellos ? 2
hello
hello
Sounter is : 0

Parametrli takrorlash operotori quyidagi ko‘rinishga ega:

```
for( 1-ifoda; shart; 2-ifoda)
sikl tanasi;
```

Avval 1-ifoda bajariladi va to shart yolg‘on bo‘lguncha sikl tanasi va 2- ifoda bajariladi. Ixtiyoriy ifoda bo‘sh bo‘lishi mumkin, lekin ularni ajratuvchi qavs « ; » bo‘lishi shart.

Bu operator quyidagi operatorga mosdir.

```
1-ifoda;
while(2-ifoda) {
operator
3-ifoda
}
```

Misol. Sikl yordamida 10 gacha bo`lgan sonlar yig`indisini hisoblash dasturi:

```
#include<iostream>
using namespace std;
int main() {
int s=0;
for(int i=1; i<=10; i++)  s+=i;
cout<<s<<endl;
system("pause");
return 0;
}
```

Natija:

55

for operatori uchun murakkab ifodalarni berilishi. For operatori dasturlashning kuchli va qulay instrumentidir. for operatorida siklni o‘zaro bog‘liq bo‘lmagan parametrlar (boshlang‘ich qiymat o‘zlashtirish, bajarilish sharti va qadam) ni qo‘llanilishi sikl ishini boshqarishda juda yaxshi imkoniyatlarni ochib beradi.

For operatori quyidagi ketma–ketlikda ishlaydi:

1. Sikl hisoblagichida boshlang‘ich qiymat o‘zlashtiriladi.
2. Siklni davom etish shartidagi ifoda qiymati hisoblanadi.
3. Agarda shart ifodasi true qiymat qaytarsa, oldin sikl tanasi bajariladi, keyin esa sikl hisoblagichi ustida berilgan amallar bajariladi.

Har bir iteratsiyada 2- va 3-qadamlar takrorlanadi.

Siklda bir nechta hisoblagichni qo‘llanilishi. For operatorining sintaksisi unda bir nechta o‘zgaruvchi-hisoblagichni qo‘llanilishiga, siklni davom etishini murakkab shartlarini tekshirishga va sikl hisoblagichlari ustida ketma-ket bir nechta amalini bajarilishiga imkon beradi.

Agarda bir nechta hisoblagichga qiymat o‘zlashtirilsa, yoki ular o‘rtasida bir nechta amallar bajarilsa, bu ifodalar vergul bilan ajratilgan holda ketma-ket yoziladi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
for (int i=0, j=0; i<3; i++, j++)
cout<< "i:" <<i<< " j:" <<j<< endl;
system("pause");
return 0;
}
```

Natija:

```
i: 0        j: 0
i: 1        j: 1
i: 2        j: 2
```

**for operatorida no`l parametrlarni ishlatilishi.** For operatorining ixtiyoriy parametri tushirib qoldirilishi mumkin. No`l parametrlar for operatorining boshqa parametrlaridan nuqtali vergul (;) bilan ajratiladi. Agarda for operatorining 1- va 3-parametrlarini tushirib qoldirsak, u xuddi while operatoridek qo‘llaniladi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int counter=0;
for ( ; counter<5 ; ){
counter++;
cout <<" Looping!";
}
cout<< "\n Counter:" << counter<< ".\n";
system("pause");
return 0;
}
```

Natija:
Looping! Looping! Looping! Looping! Looping!
Counter: 5

Bu siklni bajarilishi while siklini bajarilishiga o‘xshash tarzda amalga oshiriladi. Avval counter o‘zgaruvchisiga qiymat o‘zlashtirilayapti. **For** operatorida esa parametr sifatida faqatgina siklni davom etish shartini tekshirish ifodasi ishlatilgan. Sikl o‘zgaruvchisi ustida amali ham tushirib qoldirilgan. Bu holda ushbu Siklni quyidagicha ifodalash mumkin:

while (counter<5) ...

for operatorining tanasi bo‘sh bo‘lgan holda qo‘llanilishi. Siklda for operatori orqali uning tanasiga hech qanday operator yozmasdan turib ham biror bir amalni bemalol bajarish mumkin. Bunda sikl tanasi bo‘sh satrdan iborat bo‘ladi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
for (int i=0; i<5; cout<< "i " <<i++<<endl);
system("pause");
return 0;
}
```

Natija:
i: 0
i: 1
i: 2
i: 3
i: 4

Ichki sikllar. Boshqa siklning ichida tashkil etilgan sikl ichki sikl deb aytiladi. Bu holda ichki sikl tashqi siklni har bir iteratsiyasida to‘liq bajariladi. Quyida matritsa elementlarini ichki sikl orqali to‘ldirilishi namoyish qilingan.

Misol:

```
#include <iostream>
using namespace std;
```

```
int main(){
int rows, columns;
char theChar;
cout << "How many rovs?";
cin >> rows;
cout << "How many columns?";
cin >> columns;
cout << "What character?";
cin>>theChar;
for ( int i=0;  i<rows;  i++){
for (int j=0; j<columns; j++)
cout << theChar;
cout<< "\n";
}
system("pause");
return 0;
}
```

Natija:

How many rows? 4
How many columns? 12
What character? x
x x x x x x x x x x x x
x x x x x x x x x x x x
x x x x x x x x x x x x
x x x x x x x x x x x x

for operatori hisobchisining ko‘rinish sohasi. ANSI ning yangi standarti bo‘yicha siklda e'lon qilingan o‘zgaruvchining ko‘rinish sohasi faqat sikl ichidangina iborat. Lekin ko‘pgina kompilyatorlar eski standartlarni ham qo‘llab-quvvatlaydilar. Quyida keltirilgan dastur kodini kiritib o‘zingizning kompilyatoringiz yangi standartga mos kelish-kelmasligini tekshirishingiz mumkin:

```
#include <iostream>
using namespace std;
int main(){
for( int i = 0; i<5; i++ ){
cout << " i: " << i <<  endl;
}
i=7; // i ko‘rinish sohasi chegarasidan tashqarida
system("pause");
```

```
return 0;
}
```

Agarda bu dastur xatoliksiz kompilyasiya qilinsa, demak u ANSI ning yangi standartini qo‘llab-quvvatlamaydi. Yangi standartga muvofiq, kompilyatorlar i=7 ifoda uchun xatolik haqida xabar berishi kerak. Dastur kodiga ozgina o‘zgartirish kiritilgandan so‘ng, dastur barcha kompilyatorlar uchun xatoliksiz ishlaydi.

```
#include <iostream>
using namespace std;
int main(){
int i;
for(i = 0; i<5; i++ ){
cout << " i: " << i <<  endl;
}
i=7; // i ko‘rinish sohasi chegarasidan tashqarida
system("pause");
return 0;
}
```

O‘tish operatorlari

O‘tish operatorlari boshqarishni shartsiz uzatishni amalga oshiradi.

Blokdan yoki sikldan chiqish uchun **break** operatoridan foydalaniladi.

Agar siklni davom ettirish shartini sikl o‘rtasida tekshirish zarur bo‘lsa, break operatoridan foydalanish qulaydir.

Masalan, cheksiz sikldan **break** operatori yordamida chiqish:

```
#include<iostream>
using namespace std;
int main() {
int i= 1,s=0;
while(1){
if(i>10) break; s+=i++;
};
cout<<s<<endl;
system("pause");
return 0;
}
```

Natija:

55

Ko‘pincha siklning navbatdagi iteratsiyasiga sikl tanasidagi boshqa operatorlar (navbatdagi operatorlar) bajarilmasdan turib o‘tish zaruriyati tug‘iladi. Sikl keyingi iteratsiyasiga o‘tish uchun **continue** – operatoridan foydalaniladi.

Masalan, 3 ga va 5 ga karrali bo‘lmagan sonlar yig`indisini hisoblash:

```
#include<iostream>
using namespace std;
int main() {
int s=0;
for(int i=1; i<=10; i++){
if((i%3==0)||(i%5==0)) continue;
s+=i;
}
cout<<s<<endl;
system("pause");
return 0;
}
```

Natija:

22

While (true) konstruksiyasini qo‘llanilishi. Takrorlashning navbatdagi iteratsiyasiga o‘tishda shart sifatida C++ tilida sintaksisi bo‘yicha to‘g‘ri bo‘lgan ixtiyoriy ifoda qatnashishi mumkin. Bunda shart «to‘g‘ri» bo‘lsa, sikl bajarilaveradi. Cheksiz sikllarni tashkil etish uchun shart sifatida true mantiqiy o‘zgarmas qo‘llaniladi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int counter = 0;
while(true){
counter++;
if(counter>10)
break;
}
cout<<"counter:"<<counter << "\n ";
system("pause");
return 0;
}
```

Natija:
counter: 11

For operatorini parametrsiz qo‘llanishi. Yuqorida ko`rgan misollarimiz yana bir marta C++ tilida bir xil masalani bir necha usul bilan hal qilish imkoniyati borligini ko‘rsatadi. Bundan tashqari, for operatorining 3 ta parametrini ham tushirib qoldirish va siklni break va continue operatorlarini qo‘llash orqali boshqarish mumkin. For konstruksiyasini parametrlarsiz qo‘llanilishiga misol quyida ko‘rsatilgan.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int counter=0;
int max;
cout<< "How many hellos?";
cin>> max;
for( ; ; ){
if (counter <max){
cout << "Hello! \n";
counter++;
}
else break;
}
system("pause");
return 0;
}
```

Natija:
How many hellos? 3
Hello!
Hello!
Hello!

Shartsiz o‘tish operatori

Goto operatori. Dasturlashni ilk davrlarida kichikroq hajmdagi va yetarlicha sodda dasturlar ishlatilar edi. Bunday dasturlarda sikllar nishonlardan, operatorlar va buyruqlar ketma-ketligidan hamda o‘tish operatoridan iborat edi.

C++ tilida nishon deb orqasidan ikki nuqta (:) yoziladigan identifikatorga aytiladi. Nishon doimo boshqaruv o‘tishi lozim bo‘lgan operatordan oldin o‘rnatiladi. Kerakli nishonga o‘tish uchun goto operatori qo‘llaniladi.

Bunda kalit so‘zdan keyin nishon nomi yoziladi. O‘tish operatorining ko‘rinishi:

goto <identifikator>;

Bu operator identifikator bilan belgilangan operatorga o‘tish kerakligini ko‘rsatadi. Masalan,

goto A1;…; A1:y=5;

Misol:

```
#include <iostream>
using namespace std;
int main(){
int counter=0; // hisobchini initsializatsiya qilish
loop:
counter ++;   // siklni boshlanishi
cout<<"counter: " <<counter<< "\n";
if(counter <5) //qiymatni tekshirish
goto loop;  //sikl boshiga qaytish
cout<<"sikl tugadi.counter:"<<counter<<endl;
system("pause");
return 0;
}
```

Natija:

counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
sikl tugadi.counter: 5.

Nima uchun goto operatorini ishlatmaslik kerak. **goto** operatori orqali dasturning ixtiyoriy nuqtasiga borish mumkin. Lekin **goto** operatorining tartibsiz qo‘llanilishi bu dasturni umuman tushunarsiz bo‘lishiga olib keladi. Shuning uchun oxirgi 20 yillikda butun jahon bo‘yicha dasturlashni o‘rganuvchilarga quyidagi fikr ta'kidlanib kelinmoqda: “Hech qachon goto operatorini ishlatmang’’.

goto operatorining o‘rnini bir muncha mukammalroq strukturaga ega bo‘lgan konstruksiyalar egalladi. Bular **for**, **while** va **do…while** operatorlari bo‘lib, ular goto operatoriga nisbatan ko‘proq imkoniyatlarga egadir. Lekin dasturlashda har qanday instrument to‘g‘ri qo‘llanilgandagina foydali bo‘lishi hisobga olinib, ANSI komiteti C++ tilida goto operatorini qoldirishga qaror qildi.

Ba'zi hollarda o‘tish operatoridan foydalanish dasturlashni osonlashtiradi. Misol uchun bir necha sikldan birdan chiqish kerak bo‘lib qolganda, to‘g‘ridan-to‘g‘ri **break** operatorini qo‘llab bo‘lmaydi, chunki u faqat eng ichki sikldan chiqishga imkon beradi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int n=16,s=0;
int i,j;
for(i=1;i<5;i++)
for(j=1;j<5;j++){
if(i*j>n) goto A;
C++;
};
A:cout<<"Sikl tugadi s="<<s<<endl;
system("pause");
return 0;
}
```

Natija:
Sikl tugadi s=16

Operator **goto** belgi bo‘yicha shartsiz o‘tish uchun ishlatiladi. Barcha hollarda **goto** operatori break yoki return operatori bilan almashtirilishi mumkin.

Nazorat savollari:

1. Shartli operator umumiy ko‘rinishi.
2. Takrorlash operatorlarini ko‘rsating.
3. while operatorining umumiy ko‘rinishi.
4. do while operatori while operatoridan qanday farq qiladi?

5. for operatorining umumiy ko‘rinishi.

6. Qaytarish continue operatoridan nima uchun foydalaniladi?

7. break operatoridan nima uchun foydalaniladi?

8. Cheksiz sikl qanday tashkil etiladi?

9. goto operatori boshqarishni qayerga uzatadi?

Misollar:

1. Berilgan eps aniqlikda umumiy xadi x/n! bo‘lgan ketma-ketlik yig‘indisini hisoblovchi dastur tuzing.

2. Umumiy xadi 1/n! bo‘lgan ketma-ketlik n ta xadi yig‘indisini hisoblovchi dastur tuzing.

3. Kiritilgan n ta son qat'iy o‘suvchi ekanligini tekshiruvchi dastur yarating.

4. Kiritilgan n simvoldan nechtasi o‘nli harf ekanligini switch operatori yordamida hisoblovchi dastur tuzing.

5. Berilgan son tub son ekanligini tekshiruvchi dastur tuzing.

FUNKSIYALAR

Funksiya ta'rifi. Funksiyani C++ tilida quyidagi ikki sifatda qarash mumkin:

- hosil a turlardan biri;
- dastur bajariluvchi minimal moduli.

Funksiyalar umumiy nomga ega qismdasturlardir. Ular tuzilishi bo‘yicha oddiy dasturlarga o‘xshaydi.

Funksiya ta'rifida ishlatiluvchi parametrlar *formal parametrlar* deyiladi.

Funksiya chaqirig‘ida uzatiluvchi parametrlar *haqiqiy parametrlar* deyiladi.

Funksiya yozuvi umumiy ko‘rinishi quyidagicha:

<tur> <funksiya nomi>(<formal_parametrlar_ta'rifi>)

Agar funksiya qiymat qaytarmasa turi void deb yoziladi. Formal parametrlarga ta'rif berilganda ularga boshlang‘ich qiymatlari ham ko‘rsatilishi mumkin. Funksiya qaytaruvchi ifoda qiymati funksiya tanasida

return <ifoda>;

operatori orqali ko‘rsatiladi.

Funksiyaga murojaat qilish quyidagicha amalga oshiriladi:

<funksiya nomi> (<haqiqiy parametrlar ro`yxati>);

Har bir funksiya o‘zining nomiga egadir. Qachonki, dasturda bu nom uchrasa, boshqaruv shu funksiya tanasiga o‘tadi. Bu jarayon funksiyani chaqirilishi (yoki funksiyaga murojaat qilish) deb aytiladi. Funksiya ishini tugatgandan so‘ng, dastur o‘z ishini funksiya chaqirilgan qatorning keyingisidan boshlab davom ettiradi.

Funksiya sarlavha va tanadan iboratdir. Funksiya sarlavhasida uning qaytaradigan turi, nomi va parametrlari aniqlanadi. Funksiya chaqirilganda unga uzatiladigan qiymat argument deb aytiladi. Funksiya chaqirilganda unda ko‘rsatilgan amallar ochiluvchi figurali qavsdan ({) keyingi birinchi ifodadan boshlab bajariladi. Funksiya o‘z tanasida boshqa funksiyalarni va hatto o‘z-o‘zini ham chaqirishi mumkin.

Misol:

```
#include<iostream>
using namespace std;
float  min(float a, float b)
{
if (a<b) return a;
return b;
}
int main()
{
float y=-6.0, z;
z=min(6,y);
cout<<z<<endl;
system("pause");
return 0;
}
```

Natija:

-6

Funksiyaning qaytaradigan qiymatlari. Funksiya yo biror-bir real qiymatni, yo kompilyatorga hech qanday qiymat qaytarilmasligi haqida xabar beruvchi void turidagi qiymatni qaytaradi.

Funksiyani qiymat qaytarishi uchun return kalitli so‘zidan foydalaniladi. Bunda oldin return kalitli so‘zi, keyin esa qaytariladigan qiymat yoziladi. Qiymat sifatida esa o‘zgarmaslar kabi butun bir ifodalarni ham berish mumkin.

Agarda funksiyada return kalit so‘zi uchrasa, undan keyingi ifoda bajariladi va uning natijasi funksiya chaqirilgan joyga uzatiladi. **return** operatori bajarilgandan keyin dastur funksiya chaqirilgan satrdan keyingi ifodaga o‘tadi. **return** kalitli so‘zidan keyingi funksiya tanasidagi operatorlar bajarilmaydi.

Funksiya bir nechta **return** operatorlarini o‘zida saqlashi mumkin.

Misol:

```
#include<iostream>
using namespace std;
void Print()
{
cout<< "Salom!"<<endl;
return;
cout<< "Hayr!"<<endl;
}
int main()
{
Print();
system("pause");
return 0;
}
```

Natija:

Salom!

Foydalanuvchi funksiyalari

Parametrlar initsializatsiyasi. Funksiya ta'rifida formal parametrlar initsializatsiya qilinishi, ya'ni boshlang‘ich qiymatlar ko‘rsatilishi mumkin. Funksiyaga murojaat qilinganda biror haqiqiy parametr ko‘rsatilmasa, uning o‘rniga mos formal parametr ta'rifida ko‘rsatilgan boshlangich qiymat ishlatiladi. Initsializatsiya qilingan formal parametrlar initsializatsiya qilinmagan formal parametrlardan so‘ng kelishi lozim.

Misol:

```
#include<iostream>
using namespace std;
float  min(float a, float b=0.0)
{
```

```
if (a<b) return a;
return b;
}
int main()
{
float y=-6.0, z;
z=min(y);
cout<<z<<endl;
system("pause");
return 0;
}
```
Natija:
-6

Inlayn funksiyalar. Dasturda funksiya ta'riflanganda kompilyator funksiya kodini bir marta mashina kodiga o‘tkazadi va dasturga ma'lumotlarni stekka joylovchi instruksiyalar qo‘shadi. Argumentlarni stekka qo‘shish, funksiyaga o‘tish va undan qaytish mashina vaqtini oladi. C++ kompilyatorida inlayn funksiya dasturga joylashtirilishi uchun rekursiv bo`lmasligi kerak va **for**, **while**, **do**, **switch**, **goto** operatorlari ishlatilmagan bo‘lishi kerak.

Misol:
```
#include<iostream>
#include<cmath>
using namespace std;
inline float Line(float x1,float y1,float x2=0, float y2=0)
{
return sqrt(pow(x1-x2,2)+pow(y1-y2,2));
}
int main()
{
float a=1.0,b=2.0;
float z=Line(a,b);
cout<<"z="<<z<<endl;
system("pause");
return 0;
}
```
Natija:
z=2.23607

C++ kompilyatori **inline** so‘zini uchratsa, bajariluvchi faylga har bir funksiyaga murojaat o‘rniga funksiya operatorlarini qo‘yadi. Shunday qilib dastur effektivligini oshirish mumkin.

Funksiya e'loni va ta'rifi

Prototip. Agar dasturda funksiya ta'rifi murojaatdan keyin berilsa, yoki funksiya boshqa faylda joylashgan bo‘lsa, murojaatdan oldin shu funksiyaning prototipi joylashgan bo‘lishi kerak. Prototip funksiya nomi va formal parametrlar turlaridan iborat bo‘ladi. Formal parametrlar nomlarini berish shart emas. Funksiyaning prototipi nuqtali vergul orqali tugaydigan funksiyani qaytaradigan qiymati va signaturasidan iboratdir. Funksiyani signaturasi deb uning nomi va parametrlar ro‘yxati tushiniladi.

Funksiyaning prototipi hamda aniqlanishidagi uning qaytaradigan qiymati turi va signaturasi mos bo‘lishi lozim. Agarda bunday mutanosiblik bo‘lmasa kompilyator xatolik haqida xabar beradi.

Funksiyani e'lon qilish va aniqlanishi. Dasturda funksiyani qo‘llash uchun, oldin uni e'lon qilish, keyin esa aniqlash lozim. Funksiyani e'lon qilishda kompilyatorga uning nomi, qaytaradigan qiymatlari va parametrlari haqida xabar beriladi. Funksiyani aniqlanishidan kompilyator uning qanday ishlashi haqida ma'lumot oladi. Dasturdagi biror funksiyani oldindan e'lon qilmasdan turib chaqirish mumkin emas. Funksiyani e'lon qilinishi uning prototipini (timsolini) hosil qilish deb ataladi.

Misol:

```
#include<iostream>
using namespace std;
int main() {
float  min(float , float);
float y=6.0,z;
z=min(3.3,y);
cout<<z<<endl;
system("pause");
return 0;
};
float  min(float a, float b) {
```

```
if (a<b) return a;
return b;
}
```
Natija:
3.3

Funksiyalar va o‘zgaruvchilar

Lokal o‘zgaruvchilar. Funksiyaga qiymatlar uzatish bilan birga uning tanasida o‘zgaruvchilarni e'lon qilish ham mumkin. Bu lokal o‘zgaruvchilar orqali amalga oshiriladi. Qachonki dasturni bajarilishi funksiyadan asosiy qismga qaytsa, bu funksiyadagi lokal o‘zgaruvchilar xotiradan o‘chiriladi.

Lokal o‘zgaruvchilar xuddi boshqa o‘zgaruvchilar kabi aniqlanadi. Funksiyaga beriladigan parametrlarni ham lokal o‘zgaruvchilar deb atash mumkin va ularni funksiya tanasida aniqlangan o‘zgaruvchilar kabi ishlatish mumkin.

Misol:
```
#include <iostream>
using namespace std;
float Almashtirish(float);
int main(){
float TempFer;
float TempCel;
cout <<" Feringait bo`yicha temperaturani";
cout<< "kiriting:";
cin >> TempFer;
TempCel = Almashtirish(TempFer);
cout << "\n Bu temperatura selziy shkalasi";
cout<< "bo`yicha: ";
cout << TempCel << endl;
return 0 ;
}
float Almashtirish(float TempFer)
{
float TempCel;
TempCel=((TempFer-32)*5)/9;
return TempCel;
};
```
Natija:
Feringait bo`yicha temperaturani kiriting: 50
Bu temperatura selziy shkalasi bo`yicha: 10

Global o‘zgaruvchilar. main() funksiyasida aniqlangan o‘zgaruvchilar dasturdagi barcha funksiyalar uchun murojaat qilishga imkonli va ko‘rinish sohasiga ega hisoblanadi. Bunday o‘zgaruvchilar dasturdagi funksiyalar uchun global o‘zgaruvchilar deyiladi.

Global o‘zgaruvchi nomi bilan funksiya ichida nomlari ustma-ust tushadigan lokal o‘zgaruvchilar faqatgina joriy funksiyaning ichidagina global o‘zgaruvchining qiymatini o‘zgartiradi. Lekin global o‘zgaruvchi funksiya o‘z ishini tugatgach, u chaqirilishidan oldingi qiymatini saqlab qoladi, ya'ni funksiya tanasida e'lon qilingan lokal o‘zgaruvchi funksiyaning ichida global o‘zgaruvchini yashiradi xolos.

Misol:

```
#include <iostream>
using namespace std;
void MyFunk() ; // prototip
int x = 5, y = 7; // global o‘zgaruvchilar
int main(){
cout << "global x:"<<x<<"\n";
cout<<"global y:"<<y<<"\n";
MyFunk();
cout << "global x:"<<x<<"\n";
cout<<"global y:"<<y<<"\n";
return 0;
}
void MyFunk(){
int y = 10;
cout<<"lokal "<<"x:" << x <<"\n";
cout<<"lokal "<<"y:"<<y<<"\n";
}
```

Natija:

```
global x: 5
global y: 7
local x: 5
local y: 10
global x: 5
global y: 7
```

Lokal o‘zgaruvchilar haqida batafsilroq ma'lumot. Funksiyani ichida aniqlangan o‘zgaruvchilar lokal ko‘rinish sohasiga ega deyiladi. Yuqorida aytib o‘tilganidek, bu o‘zgaruvchilarni faqatgina funksiyaning ichidagina qo‘llash mumkinligini anglatadi. C++ da o‘zgaruvchilarni nafaqat dasturning boshida, balki ixtiyoriy joyda aniqlash mumkin. Agarda o‘zgaruvchi funksiya tanasidagi biror-bir blok ichida aniqlangan bo‘lsa, bu o‘zgaruvchi faqatgina shu blok ichidagina ta'sirga ega bo‘lib, butun funksiyaning ichida ko‘rinish sohasiga ega bo‘lmaydi.

Misol:

```
#include <iostream>
using namespace std;
void MyFunk();
 int main(){
int x = 5, y = 7;
cout<< "global x:"<<x<<"\n";
cout<< "global y:"<<y<<"\n";
MyFunk();
cout << "global x:"<<x<<"\n";
cout<<"global y:"<<y<<"\n";
return 0;
}
void MyFunk(){
int x = 3,y = 10;
cout<<"lokal "<<"x:" << x <<"\n";
cout<<"lokal "<<"y:"<<y<<"\n";
}
```

Hatija:

Natija:

global x: 5
global y: 7
local x: 3
local y: 10
global x: 5
global y: 7

Funksiyadan foydalanish xususiyatlari

Funksiyada ishlatiladigan operatorlar. Funksiyada ishlatiladigan operatorlar soni va turiga hech qanday chegara yo‘q. Funksiya ichida istalgancha

boshqa funksiyalarni chaqirish mumkin bo‘lsa-da, lekin funksiyalarni aniqlash mumkin emas.

C++ tilida funksiyaning hajmiga hech qanday talab qo‘yilmasa ham, uni kichikroq tarzda tuzgan ma'quldir. Har bir funksiya tushunish oson bo‘ladigan bitta masalani bajarishi lozim. Agarda funksiya hajmi kattalashayotganligini sezsangiz, bu funksiyani bir nechta kichikroq funksiyalarga ajratish haqida o‘ylashingiz kerak.

Funksiya argumentlari. Funksiyaning argumentlari turli turda bo‘lishi mumkin. Shuningdek, argument sifatida C++ tilidagi biror-bir qiymat qaytaradigan o‘zgarmaslar, matematik va mantiqiy ifodalar va boshqa ixtiyoriy funksiyalarni berish mumkin.

Parametrlar bu lokal o‘zgaruvchilardir. Funksiyaga uzatilgan har bir argument, bu funksiyaga nisbatan lokal munosabatda bo‘ladi. Funksiyani bajarilishi jarayonida argumentlar ustida bajarilgan o‘zgartirishlar funksiyaga qiymat sifatida berilgan o‘zgaruvchilarga ta'sir qilmaydi.

Misol:

```
#include <iostream>
using namespace std;
void change(int x, int y)
{
int temp;
temp = x;
x = y;
y = temp;
cout<<"x:"<<x<<" y:"<<y<< "\n";
}
int main()
{
int x = 5,  y =10;
cout<<"x:"<<x <<"  y:"  << y <<"\n";
change(x,y);
cout<<"x:"<<x<< "y: " <<y<< "\n";
return 0;
}
```

Hatija:

x: 5 y:10
x:10 y:5
x:5 y:10 x:

Parametrlarni qiymat va adresga ko‘ra jo‘natish. Havolalar

Qiymat bo‘yicha uzatish. Parametrlar qiymatini funksiyaga uzatishning ikkita usuli mavjud: adres bo‘yicha va qiymati bo‘yicha.

C++ tilida chaqirilgan funksiya chaqiruvchi funksiyadagi o‘zgaruvchi qiymatini o‘zgartira olmaydi. U faqat o‘zining vaqtinchalik nusxasini o‘zgartirishi mumkin xolos. Chaqirayotgan va chaqirilayotgan funksiyalar parametrlar orqali axborot almashadi.

Qiymati bo‘yicha uzatishda quyidagi amallar bajariladi:

1. faktik parametrlar o‘rnida turgan ifodalar qiymatlari hisoblanadi;
2. funksiyaning formal parametrlari uchun stekda xotira ajratiladi;
3. hisoblangan qiymatlar formal parametr uchun ajratilgan adres bo‘yicha yoziladi, bunda turlarning o‘zaro muvofiqligi tekshiriladi hamda, zarurat tug‘ilganda, ular qayta o‘zgartiriladi.

Shunday qilib, stekka faktik parametrlarning nusxalari kiritiladi va funksiya operatorlari ushbu nusxalar bilan ish olib boradi. Faktik parametrlarning o‘ziga funksiyaning kirish huquqi yo‘q, demak ularni o‘zgartirish imkoni ham yo‘q.

Qiymat bo‘yicha chaqirish qulaylik tug‘diradi. Chunki funksiyalarda kamroq o‘zgaruvchilarni ishlatishga imkon beradi. Misol uchun shu xususiyatni aks ettiruvchi POWER funksiyasi variantini keltiramiz:

```
#include <iostream>
using namespace std;
int power(int x,int n)
{
int p;
for (p = 1; n > 0; --n)
p = p * x;
return (p);
}
int main()
{
```

```
cout<<power(3,2);
return 0;
}
```
Hatija:
9

Argument N vaqtinchalik o‘zgaruvchi sifatida ishlatiladi. Undan to qiymati 0 bo`lmaguncha bir ayriladi. N funksiya ichida o‘zgarishi funksiyaga murojaat qilingan boshlang`ich qiymatiga ta'sir qilmaydi.

Konstanta. Agar formal parametrlar qiymatlarini funksiya tanasida o‘zgartirishni ta’qiqlash lozim bo‘lsa, **const** modifikatoridan foydalanish lozim. Bu modifikator funksiya sarlavhasidan qaysi parametrlarni o‘zgartirish mumkin, qaysi parametrni o‘zgartirish mumkin emasligini aniqlashga imkon beradi.

Qiymat parametrlar

Adres bo‘yicha uzatish. Adres bo‘yicha uzatishda stekka parametrlar adreslarining nusxalari kiritiladi, demak, funksiyada faktik parametr joylashtirilgan xotira qismiga kirish huquqi paydo bo‘ladi va funksiya bu parametrni o‘zgartirishi mumkin.

Adres bo‘yicha uzatish uchun ilovalar qo‘llanishi mumkin. Ilova bo‘yicha uzatishda funksiyaga chaqirish paytida ko‘rsatilgan parametr adresi uzatiladi, funksiya ichida esa parametrga barcha murojaatlarning sezilmagan holda nomlari bekor qilinadi.

Misol uchun to`rtburchak yuzi va perimetrini berilgan tomonlari bo‘yicha hisoblash funksiyasini quyidagicha tasvirlash mumkin:

```
#include <iostream>
using namespace std;
void pr(const float a, const float b, float& s, float& p){
p=2*(a+b);
s= a*b;
}
int main(){
float s,p;
pr(2,3,s,p);
cout<<"s="<<s<<" p="<<p;
return 0;
```

```
}
```
Hatija:
s=6 p=10

Bu funksiyaga quyidagicha murojaat qilish mumkin: **pr(a,b,&p,&s)**. Funksiyaga p va s o‘zgaruvchilarning adreslari uzatiladi. Funksiya tanasida shu adreslar bo‘yicha 2*(a+b) va a*b qiymatlar yoziladi.

Keyingi misolda berilgan ikki o‘zgaruvchining qiymatlarini o‘zaro almashtirish funksiyasidan foydalaniladi:

Misol:
```
#include<iostream>
using namespace std;
void change (int &a, int &b){
int r;
r = a;  a = b; b = r;
}
int main(){
int a=1,b=2;
change(a,b);
cout<<"a="<<a<<" b="<<b<<endl;
system("pause");
return 0;
}
```
Natija:
a=2 b=1

Funksiyalarni qo‘shimcha yuklash

Funksiyalarni qo‘shimcha yuklash. Funksiyalarni qo‘shimcha yuklashdan maqsad bir xil nomli funksiyaga har xil turli o‘zgaruvchilar bilan murojaat qilib qiymat olishdir. Kompilyator haqiqiy parametrlar ro‘yxati va funksiya chaqirig‘i asosida qaysi funksiyani chaqirish kerakligini o‘zi aniqlaydi. Qayta yuklanuvchi funksiyalar bir-birlari bilan parametrlari ro‘yxati bilan farq qilishi lozim. Bunda parametrlar yoki turli sonda aniqlanishi, yoki turlari bilan farqlanishi kerak.

Misol:
```
#include<iostream>
using namespace std;
float  min(float a, float b) {
if (a<b) return a; return b;
```

```
}
int  min(int a, int b) {
if (a<b) return a; return b;
}
int main() {
int a=1;float x=4.5;
int m=min(6, a);float c=min(5.0, x);
cout<<"m="<<m<<" c="<<c<<endl;
system("pause");
return 0;
}
```

Natija:
m=1 c=4.5

Funksiyalarni bir xil nom bilan aniqlanishi ularning polimorfizmi deb ham ataladi. Polimorfizm so‘zi poli – ko‘p, morfe – shakl so‘zidan olingan bo‘lib, ko‘pshakllilik degan ma'noni anglatadi.

Funksiyaning polimorfizmi deganda, dasturda bir nechta turli vazifalarni bajaruvchi bir xil nomdagi funksiyalar bo‘lishi tushuniladi. Parametrlari soni va turini o‘zgartirish orqali bir nechta bir xil nomdagi funksiyalarni aniqlash mumkin. Bunday holda funksiyani chaqirishda hech qanday noaniqlik bo‘lmaydi, kerakli funksiya parametriga muvofiq tarzda aniqlanadi.

Funksiyalarni qo‘shimcha yuklash qoidalari.

1) Funksiyalar bitta ko‘rinish sohasiga tegishli bo‘lishi lozim.

2) Funksiyalar parametrlari ko‘zda tutilgan qiymatlarga ega bo‘lishi mumkin, lekin har xil funksiyalardagi bir xil parametrlar bir xil qiymatga ega bo‘lishi kerak.

3) Agar funksiyalar parametrlari ta'rifi faqat **const** modifikatori yoki ilova mavjudligi bilan farq qilsa, bu fuknksiyalarni qo‘shimcha yuklash mumkin emas.

Masalan, kompilyator **int&f1(int&,const int&){. . . }** va **int f1(int,int){. . . }** funksiyalarni ajrata olmaydi.

Rekursiv funksiyalar

Rekursiv funksiyalar. Rekursiv funksiya deb, o‘ziga-o‘zi murojaat qiluvchi funksiyaga aytiladi. U ikki xil: to‘g‘ri rekursiya va bilvosita rekursiya bo‘lishi mumkin. Agarda funksiya o‘zini-o‘zi chaqirsa, bu to‘g‘ri rekursiya deyiladi.

Agarda funksiya boshqa bir funksiyani chaqirsa va u funksiya o‘z navbatida birinchisini chaqirishi esa bilvosita rekursiya deyiladi.

Ayrim muammolar rekursiya yordamida osonroq yechiladi. Agarda ma'lumotlar ustida biror protsedura ishlatilsa va uning natijasi ustida ham xuddi shu protsedura bajarilsa, bunday hollarda rekursiyani ishlatish lozim. Rekursiya ikki xil natija bilan yakunlanadi: u yoki oxir-oqibat albatta tugaydi va biror natija qaytaradi, ikkinchisi hech qachon tugallanmaydi va xatolik qaytaradi.

Misol uchun faktorialni hisoblash funksiyasini keltiramiz:

```
#include<iostream>
using namespace std;
int factorial(int n) {
if (n!=1) return n * factorial(n-1);
else return 1;
}
int main() {
cout<<factorial(3)<<endl;
system("pause");
return 0;
}
```

Natija:

6

Manfiy argument uchun funksiya 0 qiymat qaytaradi. Parametr 0 ga teng bo‘lsa, funksiya 1 qiymat qaytaradi. Aks holda parametr qiymat birga kamaytirilgan holda funksiyaning o‘zi chaqiriladi va uzatilgan parametrga ko‘paytiriladi. Funksiyaning o‘z-o‘zini chaqirish formal parametr qiymati 0 ga teng bo‘lganda to‘xtatiladi.

Ayrim kompilyatorlar **cout** obyekti qatnashgan ifodalarda operatorlarni qo‘llashda ba'zi qiyinchiliklar paydo bo‘ladi. Agarda kompilyator chiqarish operatoridagi ifoda haqida ogohlantirish bersa, ayirish amalini qavs ichiga oling.

Rekursiyaga misol sifatida sonni satr shaklida chiqarish masalasini ko‘rib chiqamiz. Son raqamlari teskari tartibda hosil bo‘ladi. Birinchi usulda raqamlarni massivda saqlab, so‘ngra teskari tartibda chiqarishdir.

Rekursiv usulda funksiya har bir chaqiriqda bosh raqamlardan nusxa olsh uchun o‘z-o‘ziga murojaat qiladi, so‘ngra oxirgi raqamni bosib chiqaradi:

```
#include <iostream>
using namespace std;
void printd( int n){
int i;
if (n < 0){
cout<< '-';
n = -n;
}
if ((i = n/10) != 0)
printd(i);
cout<< n % 10<<" ";
}
int main(){
printd(132);
}
```

Natija:
1 3 2

Dastur bajarilishini boshqarish funksiyalari

Dasturni to‘xtatish. Jarayonni to‘xtatish uchun **abort** funksiyasidan foydalaniladi.

void abort(void);

Funksiyadan foydalanish uchun **<cstdlib >** sarlavhali faylni ulash lozim. Ko‘p kompilyatorlarda bu fayl avtomatik ulanadi.

Funksiya **abort stderr** oqimga ("**Abnormal program termination**") ma'lumot yozadi va 3 chiqish kodini hosil qiladi:

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]) {
cout<<"Calling abort()\n";
abort();
return 0;
}
```

Dasturdan chiqish. Dasturdan chiqish uchun **exit** funksiyasidan foydalaniladi.

void exit(int status);
Funksiyadan foydalanish uchun **<cstdlib>** ulash lozim.

Normal holda 0 kodi bilan chiqiladi. Aks holda nolga teng bo‘lmagan kod bilan chiqiladi. Chiqishdan oldin hamma fayllar berkitiladi. Funksiya qiymat qaytarmaydi.

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]) {
char status;
do { cout<<"Enter 1 or 2\n";
status = getchar();
}
while((status<'1')||(status>'2'));
exit(status - '0');
}
```

Dastur bajarilishini tekshirish. Dastur bajarilishini tekshirish uchun **assert** makrosidan foydalaniladi. Makros deb tanasi chaqirig‘i o‘rniga joylashtiriluvchi funksiyaga aytiladi.

Prototipi: **void assert(int test);**

Funksiyadan foydalanish uchun **< cassert >** ulash lozim.

Agar tekshirish 0 qaytarsa, **assert stderr** oqimiga quyidagi ma'lumot chiqariladi: **Assertion failed: test, fayl nomi, qator nomeri**.

Shundan so‘ng, **abort** funksiyasi chaqirilib, dastur to‘xtatiladi.

Agar **#include < cassert >** direktivasi oldidan **#define NDEBUG directive ("no debugging")** direktivasi qo‘yilsa, keyingi funksiyalar bajarilmaydi. Funksiya qiymat qaytarmaydi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
assert(1==0);
return 0;
}
```

Kutilmagan hodisalar. Dastur bajarilishi jarayoni kutilmagan hodisalar, uzilishlar haqida ma'lumot berish uchun **raise** funksiyasidan foydalaniladi:

int raise(int sig);

Funksiyadan foydalanish uchun **<SIGNAL.H>** sarlavhali faylni ulash lozim.

Dastur bajarilishi jarayonida hosil bo‘lgan kutilmagan hodisalar uzilishlar haqida ma'lumot berish uchun foydalaniladi.

Agar xatolik bo‘lmasa, 0 qaytaradi. Aks holda 0 ga teng bo‘lmagan son qaytaradi.

Misol:

```
#include <iostream>
#include <csignal>
using namespace std;
int main(){
int a, b;
a = 10;
b = 0;
if (b == 0)
raise(SIGFPE);
else  cout<<a / b;
return 0;
}
```

SIGFPE arifmetik xatolikni bildirib **exit(1)** funksiyani chaqiradi.

Nazorat savollari:

1. Funksiya tushunchasi?
2. Funksiya prototipini e'lon qilish va funksiyani aniqlash o‘rtasida qanday farq bor?
3. Agarda funksiya hech qanday qiymat qaytarmasa, uni qanday e'lon qilish kerak?
4. Rekursiya nima?
5. Inlayn funksiya qanday xususiyatga ega?
6. Funksiya prototipi nima uchun ishlatiladi?
7. Funksiyalar xossalarini ko‘rsating.
8. Parametrlarlarga qiymat uzatish xususiyatlari.
9. Funksiyalarni qo‘shimcha yuklash qoidalari.
10. Rekursiv funksiya deb qanday funksiyaga aytiladi?

Misollar:

1. Berilgan eps aniqlikda umumiy xadi 1/n! bo‘lgan ketma-ketlik yig‘indisini hisoblovchi funksiya tuzing va dasturda foydalaning.
2. Umumiy xadi x/n! bo‘lgan ketma-ketlik n ta xadi yig‘indisini hisoblovchi funksiya tuzing va dasturda foydalaning.
3. Kiritilgan son tub sonligini tekshiruvchi funksiya tuzing va dasturda foydalaning.
4. Kiritilgan ikki son eng katta umumiy bo‘luvchisini hisoblovchi funksiya tuzing va dasturda foydalaning.
5. Rekursiya yordamida Paskal uchburchagini hisoblovchi funksiya tuzing va dasturda foydalaning.

Chiziqli va binar qidiruv algoritmlari. Massivlarga ishlov berishda ushbu algoritmlarni qo‘llash

Massivlarni ta'riflash. Massiv bu indeksli o‘zgaruvchi bo`lib, bir turli nomerlangan ma'lumotlar jamlanmasidir. Massiv ta'riflanganda turi, nomi va indekslar chegarasi ko‘rsatiladi.

Massiv sodda ta'rifi:

<tur> <o‘zgaruvchi_nomi>>[<konstanta_ifoda>] = <initsializator>;

Massiv indekslar qiymati har doim 0 dan boshlanadi. Indeks sifatida butun tur qiymatini qaytaradigan har qanday ifoda qo‘llanishi mumkin: char, short, int, long.

Massiv chegarasidan tashqariga chiqish (ya'ni mavjud bo‘lmagan elementni o‘qish/yozishga urinish) dastur bajarilishida kutilmagan natijalarga olib kelishi mumkin. SHuni ta'kidlab o‘tamizki, bu eng ko‘p tarqalgan xatolardan biridir.

Agar massiv initsializatsiya qilinganda elementlar chegarasi ko‘rsatilgan bo‘lsa , ro‘yxatdagi elementlar soni bu chegaradan kam bo‘lishi mumkin, lekin ortiq bo‘lishi mumkin emas. Agar massiv uzunligiga qaraganda kamroq element berilgan bo‘lsa, qolgan elementlar 0 hisoblanadi.

Agar nomlangan massivning tavsifida uning o‘lchamlari ko‘rsatilmagan bo‘lsa, kompilyator tomonidan massiv chegarasi avtomatik aniqlanadi.

Massivda musbat elemenlar soni va summasini hisoblash.

```
#include <iostream>
using namespace std;
int main(){
int s=0,k=0;
int  x[]={-1,2,5,-4,8,9};
for(int i=0; i<6; i++){
if (x[i]<=0) continue;
k++;
s+=x[i];
};
cout<<k<<endl;
cout<<s;
system("pause");
return 0;
}
```

Natija:

4

24

Massivning eng katta, eng kichik elementi va o‘rta qiymatini aniqlash:

```
#include <iostream>
using namespace std;
int main(){
int i,j,n;
int min,max,s;
float a,b,d,x[100];
while(1){
cout<<("\n n=");
cin>>n;
if ( n>0 && n<=100 ) break;
cout<<"\n Hato  0<n<101 bo'lishi kerak";
}
cout<<"\n elementlar qiymatlarini kiriting:\n";
for (i=0;i<n;i++){
cout<<"x["<<i<<"]=";
cin>>x[i];
}
max=x[0];
min=x[0];
for (s=0,i=0;i<n;i++){
s+=x[i];
if (max<x[i]) max=x[i];
if (min>x[i]) min=x[i];
};
```

```
s/=n;
cout<<"\n max="<<max;
cout<<"\n min="<<min;
cout<<"\n o'rta qiymat="<<s;
system("pause");
return 0;
}
```

Bir o‘lchamli massivlarni funksiya parametrlari sifatida uzatish. Massivdan funksiya parametri sifatida foylalanganda, funksiyaning birinchi elementiga ko‘rsatkich uzatiladi, ya'ni massiv hamma vaqt adres bo‘yicha uzatiladi. Bunda massivdagi elementlarning miqdori haqidagi axborot yo‘qotiladi, shuning uchun massivning o‘lchamlari haqidagi ma'lumotni alohida parametr sifatida uzatish kerak.

Misol. Massiv barcha elementlari chiqarilsin:

```
#include <iostream>
using namespace std;
int form(int a[100]){
int n;
cout<<"\nEnter n:";
cin>>n;
for(int i=0;i<n;i++)
a[i]=rand()%100;
return n;
}
void print(int a[100],int n){
for(int i=0;i<n;i++)
cout<<a[i]<<" ";
cout<<"\n";
}
int main(){
int a[100];
int n;
n=form(a);
print(a,n);
system("pause");
return 0;
}
```

Funksiyaga massiv boshlanishi uchun ko‘rsatkich uzatilgani tufayli (adres bo‘yicha uzatish), funksiya tanasining operatorlari hisobiga massiv o‘zgarishi mumkin.

Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish shart emas.

Misol. Massivdan barcha juft elementlar chiqarib tashalsin:

```
#include <iostream>
using namespace std;
void form(int a[],int n){
for(int i=0;i<n;i++)
a[i]=rand()%100;
}
void print(int a[],int n){
for(int i=0;i<n;i++)
cout<<a[i]<<" ";
cout<<"\n";
}
int Dell(int a[],int n){
int j=0,i,b[100];
for(i=0;i<n;i++)
if(a[i]%2!=0){
b[j]=a[i];j++;
}
for(i=0;i<n;i++)a[i]=b[i];
return j;
}
int main(){
int a[100];
int n;
cout<<"\nEnter n:";
cin>>n;
form(a,n);
print(a,n);
n=Dell(a,n);
print(a,n);
system("pause");
return 0;
}
```

Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish shart emas. Misol tariqasida n o‘lchovli vektorlar bilan bog‘liq funksiyalarni ta'riflashni ko‘rib chiqamiz.

Vektorning uzunligini aniqlash funksiyasi:

```
#include <iostream>
#include <cmath>
using namespace std;
float mod_vec(int n,float x[]){
float a=0;
for (int i=0; i<n; i++)
a+=x[i]*x[i];
return sqrt(double(a));
}
int main(){
float a[]={1,-1.5,-2};
cout<<mod_vec(3,a)<<endl;
system("pause");
return 0;
}
```

Natija:
2.69258

Funksiyalarda bir o‘lchovli massivlar qaytariluvchi qiymatlar sifatida ham kelishi mumkin. Misol uchun ikki vektor summasini hisoblovchi funksiya protsedurani ko‘ramiz:

```
#include <iostream>
using namespace std;
void sum_vec(int n, float a[],float b[], float c[]){
for(int i=0;i<n;i++) c[i]=a[i]+b[i];
}
void print(int n,float a[]){
for(int i=0;i<n;i++)
cout<<a[i]<<" ";
cout<<"\n";
}
int main(){
float a[]={1,-1.5,-2};
float b[]={-5.2,1.3,-4};
float c[3];
sum_vec(3,a,b,c);
print(3,c);
```

```
system("pause");
return 0;
}
```
Natija:
-4.2 -0.2 -6

Ko‘p o‘lchovli massiv

Ko‘p o‘lchovli massiv ta'rifi. Ko‘p o‘lchovli massiv initsializatsiya qilinganda massivning birinchi indeksi chegarasi ko‘rsatilishi shart emas, lekin qolgan indekslar chegaralari ko‘rsatilishi shart. Ikki o‘lchovli massivlar matematikada matritsa yoki jadval tushunchasiga mos keladi. Jadvallarning insializatsiya qilish qoidasi, ikki o‘lchovli massivning elementlari massivlardan iborat bo‘lgan bir o‘lchovli massiv ta'rifiga asoslangandir

Quyidagi ko‘radigan misolimizda jadval kiritilib har bir qatorning maksimal elementi aniqlanadi:

```
#include <iostream>
using namespace std;
int main(){
double a[4][3];
double max;
int i,j;
for(i=0;i<4;i++){
for(j=0;j<3;j++){
cout<<"a["<<i<<"]["<<j<<"]=";
cin>>a[i][j];
}
cout<<'\n';
};
for(i=0;i<4;i++){
max=a[i][0];
for(j=0;j<3;j++)
if (max<a[i][j]) max=a[i][j];
cout<<max<<endl;
}
system("pause");
return 0;
}
```

Funksiyaga ko‘p o‘lchamli massivlarni uzatish. Ko‘p o‘lchamli massivlarni funksiyaga uzatishda barcha o‘lchamlar parametrlar sifatida uzatilishi kerak. C++ da ko‘p o‘lchamli massivlar aniqlanishi bo‘yicha mavjud emas. Har xil chegarali jadvallar bilan funksiyalardan foydalanishning bir yo`li bu oldindan kiritiluvchi konstantalardan foydalanishdir.

```
#include <iostream>
using namespace std;
const int N=4;//global o‘zgaruvchi
void input_array(int a[][N],int n){
int i,j;
for(i=0;i<n;i++){
for(j=0;j<n;j++){
cout<<"a["<<i<<"]["<<j<<"]=";
cin>>a[i][j];
}
cout<<'\n';
}
}
void print_array(int a[][N],int n){
int i,j;
for(i=0;i<n;i++){
for(j=0;j<n;j++){
cout<<"a["<<i<<"]["<<j<<"]=";
cout<<a[i][j]<<" ";
}
cout<<'\n';
}
}
void transp(int a[][N],int n){
int r;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
if(i<j){
r=a[i][j];a[i][j]=a[j][i];a[j][i]=r;
}
}
int main(){
int mas[N][N];
int n;
cout<<'\n'<<" n="; cin>>n;
input_array(mas,n);
transp(mas,n);
```

```
print_array(mas,n);return 0;
system("pause");
return 0;
}
```

Satrlar. Satrli konstanta ikkilik qavslarga olingan simvollar ketma-ketligidir. Satrli konstanta oxiriga avtomatik ravishda satr oxiri '\0' simvoli qo‘shiladi. Satr qiymati simvolli konstanta bo‘lgan simvolli massiv sifatida ta'riflanadi.

Qiymat berish operatori yordamida satrga qiymat berish mumkin emas. Satrni massivga yoki kiritish paytida yoki nomlantirish yordamida joylashtirish mumkin.

Nul-terminatorning holatiga qarab satrning amaldagi uzunligi aniqlanadi. Bunday massivdagi elementlar soni, satr tasviriga qaraganda, bittaga ko‘p.

Simvolli massivlar quyidagicha satr yordamida initsializatsiya qilinadi. Bu holda avtomatik ravishda massiv elementlari soni aniqlanadi va massiv oxiriga satr ko‘chirish '\0' simvoli qo‘shiladi.

Initsializatsiyani harfma-harf amalga oshirish mumkin. Bu holda so‘z oxirida '\0' simvoli aniq ko‘rsatilishi shart.

Qiymat berish operatori yordamida satrga qiymat berish mumkin emas. Satrni massivga yoki kiritish paytida yoki nomlantirish yordamida joylashtirish mumkin.

Misol:

```
#include <iostream>
using namespace std;
int main(){
char s1[10]="string1";
int k=sizeof(s1);
cout<<s1<<"\t"<<k<<endl;
char s2[]="string2";
k=sizeof(s2);
cout<<s2<<"\t"<<k<<endl;
char s3[]={'s','t','r','i','n','g','3','\0'};
k=sizeof(s3);
cout<<s3<<"\t"<<k<<endl;
char *s4="string4";//satr ko‘rsatkichi, uni o‘zgartirib bo‘lmaydi
k=sizeof(s4);
cout<<s4<<"\t"<<k<<endl;
return 0;
}
```

Natija:
string1 10
string2 8
string3 8
string4 4

Keyingi misolimizda kiritilgan so‘zdan berilgan harf olib tashlash dasturi berilgan:

```
#include <iostream>
using namespace std;
int main(){
char s[100];
cin>> s;
int i, j;
for ( i = j = 0; s[i] != '\0'; i++)
if ( s[i] != 'c' )
s[j++] = s[i];
s[j] = '\0';
cout<< s;
system("pause");
return 0;
}
```

Har gal 's' dan farqli simvol uchraganda, u j pozitsiyaga yoziladi va faqat shundan so‘ng j qiymati 1 ga oshadi.

Funksiyalar va satrlar. Funksiyalarda satrlar ishlatilganda ularning chegarasini ko‘rsatish shart emas. Satrlarning uzunligini hisoblash **len** funksiyasi ta'rifi keltirilgan dasturni keltiramiz:

```
#include <iostream>
using namespace std;
int len(char c[]){
int m=0;
while(c[m++]);
return m-1;
};
int main(){
char e[]="Pro Tempore!";
cout<<len(e)<<endl;
system("pause");
return 0;
}
```

Natija:
12

Bu funksiyaning standart varianti **strlen** deb ataladi va bu funksiyadan foydalanish uchun **string.h** sarlavhali faylidan foydalanish lozim.

Satrdan nusxa olish **strcpy** funksiyasini C++ tilida quyidagicha ta'riflash mumkin:

```
#include <iostream>
using namespace std;
void strlen(char s1[], char s2[]){
int i=0;
while(s2[i]!='\0') s1[i++]=s2[i];
s1[i]=s2[i];
}
int main(){
char s1[]="aaa";
char s2[]="ddd";
strcpy(s1,s2);
cout<< s1<<endl;
system("pause");
return 0;
}
```

Natija:
ddd

Berilgan satrni teskariga aylantiruvchi funksiya:

```
#include <iostream>
using namespace std;
void reverse(char s[]){
int c, i, j;
for(i = 0, j = strlen(s) - 1; i < j; i++, j--){
c = s[i];
s[i] = s[j];
s[j] = c;
}
}
int main(){
char s1[]="acd";
reverse(s1);
cout<< s1<<endl;
system("pause");
return 0;
```

```
}
```
Natija:
dca

So‘zlar massivlari

So‘zlar massivini kiritish. C++ tilida so‘zlar massivlari ikki o‘lchovli simvolli massivlar sifatida ta'riflanadi.

So‘zlar massivlari initsializatsiya qilinganda so‘zlar soni ko‘rsatilmasligi mumkin. Bu holda so‘zlar soni avtomatik aniqlanadi.

Quyidagi dasturda berilgan harf bilan boshlanuvchi so‘zlar ro‘yxati bosib chiqariladi:

```
#include <iostream>
using namespace std;
int main(){
char a[10][10];
char c;
for (int i=0;i<10;i++) cin>>a[i];
cin>>c;
for (int i=0;i<10;i++)
if (a[i][0]==c) cout<<a[i]<<endl;
system("pause");
return 0;
}
```

Quyidagi dasturda fan nomi, talabalar ro‘yxati va ularning baholari kiritiladi. Dastur bajarilganda ikki olgan talabalar ro‘yxati bosib chiqariladi:

```
#include <iostream>
using namespace std;
int main(){
char a[10][10];
char s[10];
int k[10];
cin>>s;
for (int i=0;i<10;i++) {
cin>>a[i];
cin>>k[i];
};
for (int i=0;i<10;i++)
```

```
if (k[i]==2) cout<<a[i]<<endl;
system("pause");
return 0;
}
```

Funksiyalar va so‘zlar massivlar. Satrli massivlar funksiya argumenti sifatida ishlatilganda satrlarning umumiy uzunligi aniq ko‘rsatilishi shartdir.

Misol:

```
#include<iostream>
using namespace std;
int count_family (int n, char a[][100], char c[] ) {
int s=0;
for( int i=0; i<n; i++ ) if (strcmp(a[i],c)==0) C++;
return s;
}
int main() {
char c[]="Tashkent";
char a[][100] = { "Tashkent", "Termez", "Hiva","Tashkent",};
cout<< count_family (4,a,c)<<endl;
system("pause");
return 0;
}
```

Natija:

2

Izlash va tartiblash

Massivlarni navlarga ajratish. Navlarga ajratish – bu berilgan ko‘plab obyektlarni biron-bir belgilangan tartibda qaytadan guruhlash jarayoni.

Massivlarning navlarga ajratilishi tez bajarilishiga ko‘ra farqlanadi. Navlarga ajratishning n*n ta qiyoslashni talab qilgan oddiy usuli va n*log(n) ta qiyoslashni talab qilgan tez usuli mavjud. Oddiy usullar navlarga ajratish tamoyillarini tushuntirishda qulay hisoblanadi, chunki sodda va kalta algoritmlarga ega. Murakkablashtirilgan usullar kamroq sonli amallarni talab qiladi, biroq amallarning o‘zi murakkabroq, shuning uchun uncha katta bo‘lmagan massivlar uchun oddiy usullar ko‘proq samara beradi.

Oddiy usullar uchta asosiy kategoriyaga bo‘linadi:

- oddiy kiritish usuli bilan navlarga ajratish;

- oddiy ajratish usuli bilan navlarga ajratish;

- oddiy almashtirish usuli bilan navlarga ajratish

Oddiy kiritish usuli bilan navlarga ajratish. Massiv elementlari avvaldan tayyor berilgan va dastlabki ketma-ketliklarga bo‘linadi. i=2 dan boshlab, har bir qadamda dastlabki ketma-ketlikdan i-element chiqarib olinadi hamda tayyor ketma-ketlikning kerakli o‘rniga kiritib qo‘yiladi. Keyin i bittaga ko‘payadi va h.k.

Kerakli joyni izlash jarayonida, ko‘proq o‘ngdan bitta pozitsiyadan tanlab olingan elementni uzatish amalga oshiriladi, ya'ni tanlab olingan element, j:=i-1 dan boshlab, navlarga ajratib bo‘lingan qismning navbatdagi elementi bilan qiyoslanadi. Agar tanlab olingan element a[i] dan katta bo‘lsa, uni navlarga ajratish qismiga qo‘shadilar, aks holda a[j] bitta pozitsiyaga suriladi, tanlab olingan elementni esa navlarga ajratilgan ketma-ketlikning navbatdagi elementi bilan qiyoslaydilar. To‘g‘ri keladigan joyni qidirish jarayoni ikkita turlicha shart bilan tugallanadi:

- agar a[j]>a[i] elementi topilgan bo‘lsa;

- agar tayyor ketma-ketlikning chap uchiga etilgan bo‘lsa.

```
#include <iostream>
using namespace std;
void sort(int n,int a[]){
int i, j, x;
for(i=1; i<n; i++){
x=a[i];
j=i-1;
while(x<a[j]&&j>=0){
a[j+1]=a[j];
j--;
}
a[j+1]=x;
}
}
int main(){
int a[]={4, 5, 3};
sort(3,a);
for(int i=0;i<3;i++) cout<<a[i]<<" ";
```

```
cout<<endl;
system("pause");
return 0;
}
```
Natija:
3 4 5

Oddiy tanlash usuli bilan navlarga ajratish. Massivning minimal elementi tanlanadi hamda massivning birinchi elementi bilan joy almashtiriladi. Keyin jarayon qolgan elementlar bilan takrorlanadi va h.k.

```
#include <iostream>
using namespace std;
void sort(int n,int a[]){
int i,min,n_min,j;
for(i=0;i<n-1;i++){
min=a[i];n_min=i;
for(j=i+1;j<n;j++)
if(a[j]<min){min=a[j];n_min=j;}
a[n_min]=a[i];
a[i]=min;
}
}
int main(){
int a[]={4, 5, 3};
sort(3,a);
for(int i=0;i<3;i++) cout<<a[i]<<" ";
cout<<endl;
system("pause");
return 0;
}
```
Natija:
3 4 5

Oddiy almashtirish usuli bilan navlarga ajratish. Elementlar juftlari oxirgisidan boshlab qiyoslanadi va o‘rin almashinadi. Natijada massivning eng kichik elementi uning eng chapki elementiga aylanadi. Jarayon massivning qolgan elementlari bilan davom ettiriladi.

```
#include <iostream>
using namespace std;
void sort(int n,int a[]){
for(int i=1;i<n;i++)
```

```
for(int j=n-1;j>=i;j--)
if(a[j]<a[j-1])
{int r=a[j];a[j]=a[j-1];a[j-1]=r;}
}
int main(){
int a[]={4, 5, 3};
sort(3,a);
for(int i=0;i<3;i++) cout<<a[i]<<" ";
cout<<endl;
system("pause");
return 0;
}
```

Natija:

3 4 5

Tezkor tartiblash. Quyidagi dasturda tezkor tartiblash algoritmiga asoslangan funksiyadan foydalanilgan. Algoritm mohiyati shundan iboratki, avval yetakchi element tanlanadi. Funksiyada yetakchi element sifatida boshlang‘ich element tanlanadi. Shundan so‘ng massiv ikki qismga ajratiladi. Yetakchi elementdan kichik elementlar past qismga, katta elementlar yuqori qismga to‘planadi. Shundan so‘ng rekursiya asosida algoritm ikkala qismga alohida qo‘llanadi.

```
#include <iostream>
using namespace std;
int a[]={5,4};
void sqsort(int k1,int k2){
if(k1<k2){
int i,j,k;
i=k1;j=k2;
while(i<j){
if (a[k1]>a[i]) {i++;continue;}
if (a[k1]<a[j]) {j--;continue;}
k=a[j];a[j]=a[i];a[i]=k;
}
k=a[k1];a[k1]=a[i];a[i]=k;
sqsort(k1,i);
sqsort(i+1,k2);
};
}
int main(){
int i;
```

```
sqsort(0,1);
for(i=0;i<2;i++) cout<<a[i]<<" ";
cout<<endl;
system("pause");
return 0;
}
```

Natija:

4 5

Nazorat savollari:

1. Satr simvolli massivdan qanday farq qiladi?
2. Bir o‘lchovli massivlarni initsializatsiya qilish usullarini ko‘rsating.
3. Ko‘p o‘lchovli massiv ta'rifi xususiyatlarini keltiring.
4. Ko‘p o‘lchovli masivlar initsializatsiyasi xususiyatlari.
5. Satrlarni initsializatsiya qilish usullarini ko‘rsating.
6. So‘zlar massivi qanday kiritiladi?
7. Qanday qilib bir o‘lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?
8. Qanday qilib ko‘p o‘lchovli massivlar formal parametrlar sifatida ishlatilishi mumkin?
9. Satr ta'riflash usullari.
10. Satrlar funksiya parametri sifatida.

Misollar:

1. Berilgan massiv qat'iy kamayuvchi ekanligini tekshiruvchi funksiya tuzing va dasturda foydalaning.

2. Matritsani vektorga ko‘paytirish funksiyasini tuzing va dasturda foydalaning.

3. Berilgan satr telefon raqami ekanligini aniqlovchi funksiya tuzing va dasturda foydalaning.

4. Berilgan satr o‘zgaruvchi ekanligi tekshiruvchi funksiya tuzing va dasturda foydalaning.

5. Berilgan jumladan eng qisqa so‘z ajratib oluvchi funksiya tuzing va dasturda foydalaning.

UNIVERSAL FUNKSIYALAR VA ALGORITMLAR

Funksiyalar shablonlari

Funksiya shabloni (parametrlangan turlar) bog‘langan funksiyalar oilasini tuzish imkonini beradi. Shablon kiritilishi uchun, hosil qilingan funksiya avtomatlashtirish, har xil turli ma'lumotlarni qayta ishlashdan iborat. Masalan, algoritm tartiblash uchun har qaysi funksiyani o‘zini aniqlovchi turi qo‘llaniladi. Funksiya shabloni bir marta aniqlanadi, lekin parametrli aniqlashda va hokazo, ma'lumotlar turi shablon parametrlari orqali beriladi. Shablon formati:

template <class tur_nomi [,class tur_nomi]>
<funksiya_sarlavhasi>
{ <funksiya tanasi>}

Shablon funksiyasining asosiy parametrlarining xossasi.

Funksiyalar shablonlari parametrlarining asosiy xususiyatlari:

1. Parametrlar nomlari shablonning butun ta'rifi bo‘ylab unikal bo‘lmog‘i lozim.

2. Shablon parametrlarining ro‘yxati bo‘sh bo‘la olmaydi.

3. Shablon parametrlari ro‘yxatida har biri **class** so‘zidan boshlanadigan bir nechta parametr bo‘lishi mumkin.

Shablon parametri nomi funksiyada hamma nom huquqlariga ega.

Shablon yordamida aniqlangan funksiya ixtiyoriy sondagi parametrlashtirilmagan formal parametrlarga ega bo‘lishi mumkin. Funksiya qaytaruvchi qiymat ham parametrlashmagan bo‘lishi mumkin.

Parametrlar ro‘yxatidagi parametrlar nomlari shablon ta'rifidagi parametrlar nomlari bidlan bir xil bo‘lishi shart emas.

Parametrlashgan funksiya konkretlashtirilganda bir xil parametrlashgan formal parametrlarga mos haqiqiy turlar parametrlari bir xil bo‘lishi lozim.

4. Funksiya shablonini qo‘shimcha yuklash mumkin.

Funksiya shablonini qo‘shimcha yuklash. Funksiyalarni qo‘shimcha yuklash shablon parametrlari konkretlashgandan so‘ng amalga oshiriladi va u hamma tur o‘zgartirishlarni hisobga oladi.

Agar hamma qo‘shimcha yuklash xususiyatlari bir xil bo‘lsa, oddiy funksiya shablonli funksiyaga emas, oddiy funksiyaga murojaat qilinadi.

Agar ikki shablondan birini tanlash lozim bo‘lsa, ko‘proq mutaxassislashgan funksiya tanlanadi.

Misol:

```
#include <iostream>
using namespace std;
template<class T>
T maximum( T x, T y){
if(x>y) return x; else return y;
};
int maximum( int x, int y){
if(x>y) return x; else return y;
};
int main(){
float a=0.1,b=0.2;
float d=maximum (a,b);
cout<<d<<endl;
int c=maximum(4,5);
cout<<c;
return 0;
}
```

Natija:

```
0.2
5
```

Bunda **<class T>** shablonining argumenti tomonidan taqdim etilgan ma'lumotlar turi har qanday bo‘lishi mumkin. Undan dasturda foydalanishda kompilyator funksiya kodini bu funksiyaga uzatilayotgan parametrlarning faktik turiga muvofiq generatsiya qiladi.

Faktik turlar kompilyasiya paytida ma'lum bo‘lishlari kerak. Shablonlarsiz **max** funksiyasini ko‘p martalab ortiqcha yuklashga to‘g‘ri kelar edi, ya'ni garchi barcha funksiya versiyalarining kodlari bir xil bo‘lsa ham, har bir qo‘llanayotgan tur uchun alohida ortiqcha yuklash kerak bo‘lar edi.

Tur bo‘lmagan funksiya shablonlari parametrlari. Funksiya shablonida tur bo‘lmagan parametrdan foydalanish mumkin.

Masalan quyidagi dasturda biror qiymatni qo‘shish uchun mo‘ljallangan funksiyalar guruhini aniqlaydi.

```
#include <iostream>
using namespace std;
template<typename T, int val>
T add( T x){
return x+val;
};
int main(){
float a=0.1;
float d=add<float,2>(a);
cout<<d<<endl;
int c=add<int,3>(4);
cout<<c<<endl;
system("pause");
return 0;
}
```

Natija:

2.1

7

Tur bo‘lmagan shablon parametrlari ba'zi cheklanishlarga ega. Bunday parametrlar faqat butun konstanta sanovchi tur yoki tashqi bog‘lanishli obyektlarga ko‘rsatkich bo‘lishi mumkin. Suzuvchi nuqtali sonlar va sinf turidagi obyektlardan foydalanish mumkin emas.

Tarixiy sabablarga ko‘ra tur parametrini aniqlash uchun **typename** o‘rniga **class** kalit so‘zidan foydalanish mumkin. Kalit so‘z **typename** C++ tili evolyusiyasi davomida yaqinda paydo bo‘ldi. Undan oldin tur parametrini berish uchun faqat **class** so‘zi mavjud edi. Semantik jihatdan bu ikki so‘z o‘rtasida farq yo‘q. Hatto **class** so‘zi ishlatilganda ham ixtiyoriy turdan foydalanish mumkin. Lekin **class** so‘zi chalg‘itishi mumkin (T o‘rniga sinf emas ixtiyoriy turni qo‘yish mumkin), sinf bo‘lmagan turlar uchun **typename** kalit so‘zidan foydalanish ma’qulroqdir.

Algoritm

Ikkiga bo‘lib izlash. Quyidagi dasturda tartiblangan massivda ikkiga bo‘lib kalit sonni izlash algoritmi asosida tuzilgan funksiyadan foydalanish keltirilgan. Dasturda shu funksiya satrlar uchun varianti keltirilgan:

```
#include <iostream>
#define size 5
using namespace std;
template<class T>
int bsearch(T a[],T key,int n){
int m1,m2,m;
m1=0;m2=n-1;
while(m1<=m2){
m=(m2+m1)/2;
if (a[m]==key) return m;
if (a[m]>key) m2=--m;
if (a[m]<key) m1=++m;
}
return -1;
};
//
int main(){
int m;
int a[]={5,6,9,11};
m=bsearch(a,9,4);
cout<<m<<endl;
system("pause");
return 0;
}
```

Natija:

2

Standart algoritmlar

Iteratorlar. Algoritmni ixtiyoriy tuzilmalarga umumlashtirish uchun iteratorlardan foydalaniladi. Iterator qiymati elementning ketma-ketlikda masalan massivda pozitsiyasiga teng obyektdir. Iteratorlarga misol massivlar nomlari. Iteratorlar bilan quyidagi amallar bog‘langan:

pos+n n qadam oldindagi element pozitsiyasi

pos+=n ekvivalent **pos=pos+n**

pos-n n qadam orqadagi element pozitsiyasi

pos-=n ekvivalent **pos=pos-n**

pos1-pos2 po1 va pos2 orasidagi elementlar soni

poC++, ++pos keyingi pozitsiyaga o‘tish

pos—, --pos oldingi pozitsiyaga o‘tish

***pos** pos ko‘rsatgan pozitsiyadagi element

pos==pos1 iteratorlar qiymati teng

pos!=pos1 iteratorlar qiymati teng emas

Iteratorlarning quyidagi turlari mavjud:

1. Bitta yo‘nalishdagi iteratorlar (forward_iterator) kiritish/chiqarish amallarning barchasini qo‘llaydi, bundan tashqari chegarasiz o‘zlashtirishning imkonini beradi.

==, !=, =, *i, ++i, i++, *i++

2. Ikki yo‘nalishdagi iteratorlar (biderectional_iterator) forward-iteratorlarning barcha xususiyatlariga ega, bundan tashqari, konteynerni ikkita yo‘nalishi bo‘yicha o‘tish imkonini beradigan qo‘shimcha dekrementa (--i, i--, *i--) amaliga ega.

3. Ixtiyoriy ruxsatga ega bo‘lgan iteratorlar (random_access_iterator) biderectional-iteratorlarning barcha xususiyatlariga ega, bundan tashqari solishtirish va manzil arifmetikasi amallarni qo‘llaydi.

i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2

Algoritmlar

Har bir algoritm funksiyalar shabloni yoki funksiyalar shabloni to‘plami yordamida ifodalanadi. Shunday qilib, algoritm har xil turdagi qiymatlarga ega bo‘lgan har xil tuzilmalar bilan ishlay oladi. Barcha algoritmlarni argumentlari (**beg, end**) yarim oraliqlar bo‘ladi.

Misol tariqasida interval hamma elementlari berilgan shartga javob berishini tekshiruvchi algoritm ko‘ramiz.

Misol:

#include <iostream>

```
using namespace std;
template<typename FI, typename CompFunk>
bool all_if(FI beg, FI end, CompFunk ff) {
while(beg!=end){
if (!ff(*beg)) return false;
beg++;
}
return true;
};
bool poz(int x){
if(x>0) return true; else return false;
};
int main(){
int a[]={1,2,-3};
cout<<all_if(a,a+3,poz);
cout<<endl;
system("pause");
return 0;
}
```

Natija:

0

Algoritm **for_each()** yordamida har xil ko‘rinishidagi qayta ishlashni va har bir elementni modifikatsiyasini ko‘rish mumkin.

for_each() algoritmining qo‘llanishi:

```
#include <iostream>
using namespace std;
void print(int x){cout<<x<<" ";};
class Print{
public:
void operator()(int x){cout<< x<<" ";}
};
template<class Iterator, class Operation>
Operation for_each(Iterator beg, Iterator end, Operation op){
while (beg!=end) {
op(*beg);
++beg;
};
return op;
};
int main(){
Print p;
```

```
int a[]={1,2,3};
for_each(a,a+3,print);cout<<endl;
for_each(a,a+3,p);cout<<endl;
for_each(&a[0],&a[3],Print());cout<<endl;
return 0;
}
```

Ikkiga bo‘lib izlash. Quyidagi dasturda tartiblangan massivda ikkiga bo‘lib kalit sonini izlash algoritmi asosida iteratorlar yordamida tuzilgan funksiyadan foydalanish keltirilgan. Bu funksiyada ixtiyoriy murojaat qilinuvchi iteratorlardan foydalanilgan.

```
#include <iostream>
using namespace std;
template<class iter, class T>
int bsearch(iter beg,iter end, T key){
int m1,m2,m;
int n;
n=(end-beg);
m1=0;m2=n-1;
while(m1<=m2){
m=(m2+m1)/2;
if (*(beg+m)==key) return m;
if ((*beg+m)>key) m2=--m;
if (*(beg+m)<key) m1=++m;
}
return -1;
};
int main(){
int m;
int a[]={5,6,9,11};
m=bsearch(a,a+4,11);
cout<<m<<endl;
system("pause");
return 0;
}
```

Funksiyaga ilovalar

Ilovalar. Ilova biror obyektning o‘zgacha nomidir. Ilovalar quyidagicha ta'riflanishi mumkin:

<tur>& <Ilova_nomi>=<ifoda>
Misol:

```
#include <iostream>
using namespace std;
int main() {
int i=123;
int &p=i;
cout<<"\n i="<<i<<" p="<<p;
return 0;
}
```
Natija:
I=123 p=123

Ilovalar bilan ishlash qoidalari. Ilova o‘zgaruvchi emasdir. Ilovaga bir marta qiymat bergandan so‘ng uni o‘zgartirish mumkin emas. Bundan tashqari ilovalar ustida quyidagi amallarni bajarish mumkin emasdir:

- C++ adres olish operatori yordamida ilovaning adresini olish mumkin emas.
- Ilovaga ko‘rsatkich qiymatini berish mumkin emas.
- Ilovalarni solishtirish mumkin emas.
- Ilovalar ustida arifmetik amallar bajarish mumkin emas.
- Ilovani o‘zgartirish mumkin emas.

Funksiya va ilova. Funksiya ilova qaytarishi mumkin. Odatda funksiya o‘ng qiymatli hisoblanadi, lekin funksiya ilova qaytarsa chap qiymatli obyekt bo‘ladi.

Misol:

```
#include <iostream>
using namespace std;
int a=5;
int &fa() {
int&b=a;
return b;
}
int main() {
fa()=6;
cout<<a;
return 0;
}
```
Natija:
6

Funksiyalarga ilovalar. Funkiyaga ilova quyidagicha shunday aniqlanadi:

funksiya_turi(&ilova_nomi)(parametrlar)nomlantiruvchi_ifoda;

Funksiya nomini parametrlarsiz va qavslarsiz qo‘llash funksiya adresi sifatida qabul qilinadi. Funksiyaga ilova funksiya nomining sinonimi bo‘ladi. Funksiyaga ilovaning qiymatini o‘zgartirib bo‘lmaydi.

Funksiyaga ilova funksiya parametri sifatida kelishi mumkin. Bu funksiyani parametr sifatida uzatishga imkon beradi.

Misol:

```
#include<iostream>
using namespace std;
double kub(double x){
return x*x*x;
}
double rec(double (&pf)(double x),double a){
return pf(a);
}
int main(){
float a=-2;
double c=rec(kub,a);
cout<<c<<endl;
return 0;
}
```

Natija:

-8

Razryadli arifmetika

Maska. Maska deb tanlangan bitlari 1 ga teng bo‘lib, qolgan bitlari 0 ga teng bo‘lgan songa aytiladi. Sodda maska deb bitta tanlangan bit 1 ga teng bo‘lib, qolgan bitlari 0 ga teng bo‘lgan songa aytiladi. Ta'rifdan ko‘rinib turibdiki, sodda maska ikkining darajalaridan iboratdir.

Masalan, uchinchi biti noldan farqli maska hosil qilish:

```
unsigned mask=1;
mask<<=2;
```

Maskalar ko‘pincha & amali bilan qo‘llanadi:

```
unsigned mask=2;
val&=mask
```

Bu misolda val o‘zgaruvchining oxiridan ikkinchi bitidan tashqari hamma bitlar 0 ga teng bo‘ladi. Oxiridan ikkinchi biti o‘zgarmaydi.

Yana bir misol:

ch &= 0xff; /* yoki ch &= 0377; */

Ma'lumki 0xff, qiymati binar ko‘rinishi 11111111 ya'ni 0377. Bu maska o‘zgaruvchi oxirgi 8 bitini o‘zgartirmasdan qolganlarini 0 ga o‘rnatadi.

Bitni 1 ga o‘rnatish:

unsigned mask=2;

flags |= MASK;

Bu misolda oxiridan ikkinchi bit 1 ga o‘rnatiladi. Qolgan bitlar qiymatlari o‘zgarmaydi.

Bitni 0 ga o‘rnatish:

unsigned mask=2;

flags &= ~MASK;

Bu misolda oxiridan ikkinchi bit 0 ga o‘rnatiladi. Qolgan bitlar qiymatlari o‘zgarmaydi.

Bitni teskariga o‘rnatish:

```
unsigned mask=2;
flags^= mask;
```

Bu misolda oxiridan ikkinchi bit 0 bo‘lsa 1 ga o‘rnatiladi va 1 bo‘lsa 0 ga o‘rnatiladi. Qolgan bitlar qiymatlari o‘zgarmaydi.

Hamma sodda maskalarni chiqarish:

```
#include <iostream>
using namespace std;
int main(){
unsigned mask=1;
int i;
for(i=1;i<33;i++){
cout<<i<<" "<<mask<<endl;
mask<<=1;
}
system("pause");
return 0;
}
```

Hamma bitlarni chiqarish. Butun ishorali sonlar tasvirlanganda yetakchi bit ishorani ko‘rsatish uchun ishlatiladi. Musbat butun sonlar to‘g‘ri kodda tasvirlanadi, ya'ni bosh kodi nolga teng bo‘lib, qolgan bitlar son ikkilik ko‘rinishi asosida tasvirlanadi.

Manfiy sonlar qo‘shimcha kodda tasvirlanadi, ya'ni musbat soni tasviridagi hamma kodlar teskarisiga aylantiriladi. So‘ngra 1 qo‘shiladi. Sonni hamma bitlarini chiqarish uchun siklda 1 soni bilan & amali natijasi ekranga chiqariladi va songa surish >> amali qo‘llanadi.

Dastur:

```
#include <iostream>
using namespace std;
void printbits(int c){
int i;
unsigned k=8*sizeof(int);
for(i=0;i<k;i++){
cout<<(c&1);
c>>=1;
}
}
int main(){
printbits(5);
cout<<endl;
system("pause");
return 0;
}
```

Bu dastur natijasida son bitlari teskari tartibda ekranga chiqadi. To‘g‘ri tartibda chiqarish uchun sonni o‘zini avval aylantirib olinadi. Quyidagi dasturda sonni aylantirish va bitlarni ekranga chiqarish funksiyalaridan foydalanilgan:

```
#include <iostream>
using namespace std;
int reversebits(int c){
int i;
int m=0;
int k;
unsigned l=8*sizeof(int);
for(i=0;i<l;i++){
k=c&1;
m<<=1;
```

```
m|=k;
c>>=1;
}
return m;
}
void printbits(int c){
int i;
unsigned k=8*sizeof(int);
for(i=0;i<k;i++){
cout<<(c&1);
c>>=1;
}
}
int main(){
int m;
m=reversebits(-3);
cout<<m<<endl;
printbits(m);
cout<<endl;
system("pause");
return 0;
}
```

Keyingi misolimizda berilgan sonni n bitini o‘zgartirish funksiyasi yaratilgan. Ma'lumki operator ~ hamma bitlarn teskariga aylantiradi. Ma'lum bitlarni tanlashga imkon bermaydi. Quyida maskadan foydalanib n bitni teskariga aylantirish ko‘rsatilgan:

```
#include <iostream>
using namespace std;
int invert_end(int num, int bits){
int mask = 0;
int bitval = 1;
while (bits-- > 0){
mask |= bitval;
bitval <<= 1;
}
return num ^ mask;
}
void printbits(int c){
int i;
unsigned k=8*sizeof(int);
for(i=0;i<k;i++){
printf(" %d",c&1);
```

```
c>>=1;
}
}
int main(){
int m;
printbits(5);
cout<<"\n";
m=invert_end(5,3);
printbits(m);
return 0;
}
```

Nomlar fazosi

An'anaviy tarzda qabul qilinishicha, biron-bir lokal soha (funksiya, sinf tanasi yoki translyasiya moduli) ga kiritilmagan barcha nomlar umumiy global ismlarni bo‘lib olishadi. Shuning uchun, agar ayrim modullarni yig‘ish jarayonida nomlar takroran aniqlangani ayon bo‘lib qolsa, bu holda har bir nomni qandaydir yo‘l bilan farqlash zarurligini talab qiladi. C++da bu muammoning yechilishi nomlar fazosi (**namespace**) mexanizmi zimmasiga yuklatilgan.

Bu mexanizm ilovani bir necha tarmoq tizimlar (tizimchalar) ga bo‘lib tashlash imkonini beradi, bunda har bir tarmoq tizim nomlarni tanlashda erkin ish tutadi. Har bir tarmoq tizim global nomlar umumiy fazosida o‘zining paydo bo‘lganini **namespace** kalit-so‘zdan keyin kelgan unikal identifikator yordamida identifikatsiya qiladi:

namespace<identifikator> {[<e'lon qilish >]}

Identifikatsiya qilingan nomlar fazosi elementlariga kirishning uchta usuli mavjud:

Konkret elementga ochiq-oydin kirish kvalifikatsiyasi:

<fazo nomi > :: < element nomi>;

Barcha elementlarga qirish:

using namespace <fazo nomi >;

Nomlarning lokal fazosida yangi identifikatorning e'lon qilinishi:

using :: <identifikator>;

Misol:

```
#include <iostream>
using namespace std;
namespace dragon{int gold = 50;}
namespace king{int gold = 250;}
int main(void){
using dragon::gold;
cout << gold<<endl;
cout<<king::gold<<endl;
system("pause");
}
```

Natija:
50
250

Nomlar fazosini qayta ta'riflash. Agar ikki bir xil nomli nomlar fazosi e'lon qilinsa ikkinchisi birinchisiga qo‘shiladi.

Misol:

```
#include <iostream>
using namespace std;
namespace x{
void func1(void) {cout <<"::func1" << endl; }
}
namespace x{
void func2(void) {cout <<"::func2" << endl; }
}
int main(void) {
x::func1();
x::func2();
system("pause");
}
```

Natija:
::func1
::func2

C++dastur kompilyasiyasida ikki nomlar fazosini bitta ko‘inish sohasiga joylaydi. Shuning uchun nomlar fazolarida bir xil nomli komponentalar mavjud bo‘lish mumkin emas.

Global nomlar fazosi. Global nomlar fazosi – eng yuqori ko‘rinish sohasi ismi (bu sohada global o‘zgaruvchilar mavjuddir). Global nomlar fazosi komponentasiga murojaat qilish uchun ba'zida ruxsat berish operatoridan (::)

foydalanishga to‘g‘ri keladi. Bu lokal va global nomlar fazolarida bir xil identifikatorlar mavjud bo‘lganda yuz beradi, chunki jimlik bo‘yicha eng kichik ko‘rinish sohasiga ega o‘zgaruvchi tanlanadi. Global nomlar fazosi komponentasiga murojaat sintaksisi:

::globalKomponent;

Ismsiz nomlar fazolari yaratish. Ismsiz nomlar fazosi yaratilganda C++ avtomatik ravishda unikal ism beradi. Bu holda C++ using namespace e'lon qo‘shib ismsiz nomlar fazosi komponentalariga murojaat qilishga imkon beradi.

Misol:

```
#include <iostream>
using namespace std;
namespace {
void func(void) {cout <<"::func" << endl; }
}
int main(void) {
::func();
system("pause");
}
```

Natija:

::func

Standart nomlar fazosi std. Standart nomlar fazosi std deb nomlanadi va C++ standart kutubxonalariga kirgan hamma nomlarni o‘z ichiga oladi, shu qatorda **iostream** va satrlar bilan ishlash kutubxonasini. Direktiva **using namespace std** standart nomlar fazosini global nomlar fazosiga ulaydi. Standart va global nomlar fazosini ulamasdan dastur tuzish uchun ruxsat berish operatoridan foydalanish lozim.

Misol:

```
#include <iostream>
#include <string>
int main(void) {
std::string name;
std::cout<<"Name?"<<std::endl;
std::cin>>name;
std::cout<<"\n Hello " <<name.c_str()<< std::endl;
system("pause");
}
```

Turlarni keltirish operatorlari

Operator const_cast. Operator konstant turni konstant bo‘lmagan turga hamda harakatchan **(volatile)** turni harakatsiz turga keltirish uchun ishlatiladi. Qolgan hollarda ishlatishga urinish kompilyasiya xatosiga olib keladi.

Misol:

```
#include<iostream>
using namespace std;
int main() {
const k=5;
int i=const_cast<int>(k);
cout<<i;
return 0;
}
```

Bu misolda butun konstantani haqiqiy konstantaga keltirishga urinish xatolik keltirib chiqaradi.

Operator static_cast. Operator turlarni ko‘zda tutilgan qoidalar asosida keltirish uchun ishlatiladi. Avtomatik keltirishdan farqi shundaki, xavfli keltirish haqida kompilyator ma'lumot bermaydi.

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]){
float h=5.5;
int k=static_cast<int>(h);
cout<<k<<endl;
system("pause");
return 0;
}
```

Natija:

5

Operator reinterpret_cast. Operator obyektlarni ichki ko‘rinishi bilan ishlaydi. O‘zgartirish to‘g‘riligi dasturchiga bog‘liq.

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]) {
int k=5;
```

```
int c=3;
k=reinterpret_cast<int>(c);
cout<<k<<endl;
system("pause");
return 0;
}
```

Natija:

3

Nazorat savollari:

1. Shablonlarlardan nima maqsadda foydalaniladi?
2. Funksiya shabloni asosiy xossalarini ko‘rsating.
3. Parametrlashtirilgan funksiyalar xossalarini ko‘rsating.
4. Shablon parametrlari ro‘yxati bo‘sh bo‘lishi mumkinmi?
5. Parametrlashtirilgan funksiya qanday chaqiriladi?
6. Iterator tushunchasi.
7. Iteratorlar amallari.
8. Universal algoritmlar yaratish usullarini ko‘rsating.
9. Nomlar fazosi mexanizmidan foydalanish.
10. Turlarni keltirish operatorlari.

Misollar:

1. Bir ketma-ketlikdan ikkinchisiga nusxa oluvchi universal funksiya yarating.

2. Qiymati berilganga teng elementlar sonini hisoblovchi universal funksiya tuzing.

3. Berilgan qiymatdagi elementlarni o‘chiruvchi universal funksiya tuzing.

4. Maksimal elementni topish universal funksiyasini yarating.

5. Berilgan qiymatdagi element bilan ketma-ketlikni to‘ldiruvchi universal funksiya yarating.

MURAKKAB TURLAR

Sanovchi tur

Agar har xil qiymatlarga ega bir nechta nomlangan konstanta kiritish kerak bo‘lsa, sanovchi turdan foydalanish mumkin:

enum <tur_nomi > {<konstantalar ro‘yxati >};

Konstantalar butun bo‘lishi kerak va initsializatsiya qilinishi mumkin, agar initsializator mavjud bo‘lmasa, birinchi konstanta nol qiymat oladi, qolganlari bo‘lsa oldingisidan bitta ko`p bo‘ladi.

Misol:

```
#include <iostream>
using namespace std;
enum {bir, ikki, uch};
int main(){
cout<<bir<<" "<<ikki<<endl;
system("pause");
return 0;
}
```

Natija:
0 1

Nomlangan sanovchi tur kiritib, shu turdagi o‘zgaruvchilar, ko‘rsatkichlar va ilovalardan foydalanish mumkin.

Misol:

```
#include <iostream>
using namespace std;
enum color{black, green=3, yellow, blue, red, white};
int main(){
cout<<green<<" "<<blue<<endl;
system("pause");
return 0;
}
```

Natija:
3 5

Operator **static_cast** arifmetik turni enum turiga keltirishga imkon beradi.

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]) {
enum num {first = 1, second, third};
```

```
int i=1;
num nk=static_cast<num>(i);
cout<<nk<<endl;
cout<<first<<" "<<second<<" "<<third<<endl;
system("pause");
return 0;
}
```

Natija:

1

1 2 3

Bu misolda o‘zgaruvchi 1, 2 yoki 3 qiymatga teng bo‘lishi kerak.

Strukturalar

Struktura – bu ma'lumotlarni bir butun nomlangan elementlar to‘plamiga birlashtirish. Struktura elementlari (maydonlar) har xil turda bo‘lishi mumkin va ular har xil nomlarga ega bo‘lishi kerak.

Strukturali tur quyidagicha aniqlanadi:

struct { <ta'riflar ro‘yxati > }

Strukturada albatta bitta komponenta bo‘lishi kerak. Struktura turidagi o‘zgaruvchi quyidagicha ta'riflanadi:

<struktura_nomi > <o‘zgaruvchi>;

Struktura turidagi o‘zgaruvchi ta'riflanganda initsializatsiya qilinishi mumkin:

<struktura_nomi > <o‘zgaruvchi>=<initsializator>;

Strukturani initsializatsiyalash uchun uning elementlar qiymatlarini figurali qavslarda tavsiflanadi.

Strukturalarni o‘zlashtirish. Bitta tuzilma turdagi o‘zgaruvchilar uchun o‘zlashtirish amali aniqlangan. Bunda har bir elementdan nusxa olinadi.

Struktura elementlariga murojaat. Struktura elementlariga murojaat aniqlangan ismlar yordamida bajariladi:

<Struktura_nomi>.<element_nomi>

Bir xil turdagi strukturalarga qiymat berish amalini qo‘llash mumkin. Lekin strukturalar uchun solishtirish amallari aniqlanmagan.

Konkret strukturalar ta'riflanganda massivlar kabi initsializatsiya qilinishi mumkin. Nomsiz struktura kiritish mumkin. Strukturalar ta'riflanganda konkret strukturalar ro‘yxatini kiritish mumkin:

Misol:

```
#include <iostream>
using namespace std;
struct {
char name[20];
char title[30];
float rate;
}emp={"Smit", "director",10000};
int main(){
cout<<emp.name<<endl;
cout<<emp.title<<endl;
cout<<emp.rate<<endl;
system("pause");
return 0;
}
```

Natija:
Smit
director
10000

Konkret strukturalar elementlari dasturda alohida kiritilishi va chiqarilishi zarurdir. Quyidagi misolda xizmatchi struktura kiritiladi:

```
#include <iostream>
using namespace std;
struct employee{
char name [64];
long employee_id;
float salary;
char phone[10];
int office_number;
} worker;
void show_emploee(employee worker)
{
cout << "Ismi: " << worker.name << endl;
cout << "Telefon: " << worker.phone << endl;
cout << "Nomer: " << worker.employee_id << endl;
cout << "Oylik: " << worker.salary << endl;
cout << "Ofis: " << worker.office_number << endl;
};
```

```
int main(){
worker.employee_id = 12345;
worker.salary = 25000.00;
worker.office_number = 102;
cout<<"\n ismi:"; cin>> worker.name;
cout<<"\n telefon:"; cin>>worker.phone;
cout<<endl;
show_emploee(worker);
return 0;
}
```

Struktura o‘zgaruvchisi maydon sifatida. Murakkab strukturalarni hosil qilishda oldin uni tashkil etuvchi oddiyroq strukturalarni e'lon qilib, keyin esa ularni birlashtirish orqali strukturani hosil qilish maqsadga muvofiqdir. Masalan, g‘ildirak strukturasi, motor strukturasi, uzatish korobkasi strukturasi va boshqa strukturalarni hosil qilib, keyin esa ularni birlashtirish orqali avtomobil strukturasini qurish mumkin.

Quyidagi misolda oddiy nuqta strukturasi yaratilgan. So‘ngra moddiy nuqta yaratilib, uning ichida nuqta sinfiga tegishli o‘zgaruvchiga ta'rif berilgan:

```
#include <iostream>
using namespace std;
struct Point{
int x;
int y;
};
struct FPoint{
Point p;
float w;
};
void Print(FPoint X){
cout<<"coord x="<<X.p.x<<endl;
cout<<"coord y="<<X.p.y<<endl;
cout<<"weight w="<<X.w<<endl;
};
int main(){
FPoint X={10,20,1.5};
Print(X);
system("pause");
return 0;
}
```

Natija:

coord x=10
coord y=20
weight w=1.5

Masalan, to‘g‘ri to‘rtburchak chiziqlardan tashkil topgan. Chiziq esa ikki nuqta orqali aniqlanadi. Har bir nuqta x va u koordinatalar yordamida aniqlanadi. Quyidagi dasturda to‘rtburchak strukturasi ko‘rsatilgan. To‘g‘ri to‘rtburchak diagonal bo‘yicha ikki nuqta va ikki tomon yordamida aniqlanadi. Shuning uchun oldin har bir nuqtaning x, u koordinatalarini saqlash uchun nuqta strukturasi e'lon qilingan.

Misol:

```
#include <iostream>
using namespace std;
struct Point{
int x;
int y;
};
Point KPoint(int x1,int y1){
Point p;
p.x=x1;
p.y=y1;
return p;
};
struct Rectangle{
Point p1,p2;
int a,b;
};
Rectangle KRectangle(int x1,int y1,int x2,int y2){
Rectangle r;
r.p1=KPoint(x1,y1);
r.p2=KPoint(x2,y2);
r.a=x2-x1;
r.b=y2-y1;
return r;
};
int Per(Rectangle p){
return 2*(p.a+p.b);
}
int Sq(Rectangle p){
return p.a*p.b;
}
```

```
int main(){
Rectangle X;
X=KRectangle(10, 20, 50, 80);
cout<<"Perimetr="<<Per(X)<<endl;
cout<<"Yuza="<<Sq(X)<<endl;
system("pause");
return 0;
}
```
Natija:
Perimetr=200
Yuza=2400

Strukturalar va massivlar

Massivlar strukturalar elementlari sifatida. Massivlarni strukturalar elementi sifatida ishlatilishi hech qanday qiyinchilik tug‘dirmaydi. Biz yuqorida simvolli massivlardan foydalanishni ko‘rdik.

Quyidagi misolda fazoda berilgan nuqtaviy jismni tasvirlovchi komponentalari jism massasi va koordinatalaridan iborat struktura kiritilgan bo‘lib, nuqtaning koordinatalar markazigacha bo‘lgan masofasi hisoblangan.

```
#include <iostream>
#include <cmath>
using namespace std;
struct {
double mass;
float coord[3];
} point={12.3,{1.0,2.0,-3.0}};
int main(){
int i;
float s=0.0;
for (i=0;i<3; i++)
s+=point.coord[i]*point.coord[i];
cout<<"\n masofa="<<sqrt(s);
return 0;
}
```
Natija:
masofa=3.74166

Bu misolda point strukturasi nomsiz strukturali tur orqali aniqlangan bo‘lib, qiymatlari initsializatsiya yordamida aniqlanadi.

Strukturalar massivlari. Strukturalar massivlari oddiy massivlar kabi tasvirlanadi. Quyidagi misolda nuqtaviy jismlarni tasvirlovchi strukturalar massivi kiritiladi va bu nuqtalar tizimi uchun og‘irlik markazi koordinatalari (x_c,y_c,z_c) hisoblanadi. Bu koordinatalar quyidagi formulalar asosida hisoblanadi:

$m=\sum m_i$; $x_c=(\sum x_i m_i)/m$; $y_c=(\sum y_i m_i)/m$; $z_c=(\sum z_i m_i)/m$;

Misol:

```
#include <iostream>
using namespace std;
struct particle{
double mass;
double coord [3];
};
int main(){
struct particle mass_point[]={ 20.0,
{2.0,4.0,6.0},40.0, {6.0,-2.0,6.0}, 10.0,
{1.0,3.0,2.0}};
int N;
struct particle center ={ 0.0, {0.0,0.0,0.0}};
int I;
N=sizeof(mass_point)/sizeof(mass_point[0]);
for (I=0;I<N;I++){
center.mass+=mass_point[I].mass;
center.coord[0]+= mass_point[I].coord[0]*
mass_point[I].mass;
center.coord[1]+= mass_point[I].coord[1]*
mass_point[I].mass;
center.coord[2]+= mass_point[I].coord[2]*
mass_point[I].mass;
}
cout<<"\n Massa markazi koordinatalari:";
for (I=0;I<3;I++){
center.coord[I]/=center.mass;
cout<<"\n Koordinata "<<(I+1)<<center.coord[I];
}
return 0;
}
```

Struktura xossalari

Strukturaviy tur kiritish. Strukturaviy turni kiritishda yana bir imkoniyatni yordamchi so‘z **typedef** yaratadi.

Strukturaviy tur kiritilgan va qayta nomlangan hol uchun ta'rif quyidagi ko‘rinishga ega bo‘ladi:

typedef struct{elementlar tarifi} struktura turi

Keltirilgan qoida strukturaviy tur va unga belgilash kiritadi. Bu belgilash orqali struktura turidagi o‘zgaruvchilarni xuddi shunday oddiy nomlangan strukturaviy tur kabi kiritish mumkin.

Strukturaviy turga dasturchi **typedef** yordamida nom beradi shu bilan birga u ikkinchi nomga ham **struct** yordamchi so`z orqali ega bo‘la oladi.

Strukturalar va funksiyalar. Strukturalar funksiyalar argumentlari sifatida yoki funksiya qaytaruvchi qiymat sifatida kelishi mumkin. Bundan tashqari funksiya argumenti sifatida struktura turidagi massiv kelishi mumkin.

Misol:

```
#include <iostream>
using namespace std;
typedef struct{
char name[20];
int year;
} person;
person old_person(person a[], int n){
int i;
person s=a[0];
for(i=1;i<n;i++)
if(a[i].year>s.year) s=a[i];
return s;
}
void print_person(person s){
cout<<"name "<<s.name<<endl;
cout<<"year "<<s.year<<endl;
}
int main(){
person a[]={{"smit",34},{"bobbi",45},{"pit",56}};
person s=old_person(a,3);
print_person(s);
system("pause");
return  0;
}
```

Natija:
name pit
year 56

Funksiyada bitta struktura turidagi o‘zgaruvchi qiymatini o‘zgartirish mumkin emas, lekin massiv elementlari qiymatini o‘zgartirish mumkin:

```
#include <iostream>
using namespace std;
struct goods {
char name[20];
long price;
float percent;
};
void change_percent(struct goods a[], int n, float percent){
int i;
for(i=1;i<n;i++) a[i].percent=percent;
}
void print_goods(struct goods s){
cout<<s.name<<" "<<s.price<<" "<<s.percent<<endl;
}
void all_print(struct goods a[], int n){
int i;
for(i=0;i<n;i++) print_goods(a[i]);
};
int main(){
struct goods a[]={
{"smit",34,0.5},{"bobbi",45,0.7},{"pit",56,0.8}
};
all_print(a,3);
change_percent(a,3,0.5);
all_print(a,3);
system("pause");
return  0;
}
```

Abstrakt turlarni tasvirlash

Amaliy masalalarni yechishda, shu soha uchun mos bo‘lgan ma'lumotlar turlarini aniqlab olish qulaydir. Dasturda bu turlar strukturali turlar sifatida tasvirlanadi. So‘ngra shu tur bilan bog‘liq hamma funksiyalarni ta'riflab, kutubxona hosil qilinadi. Misol tariqasida kasrlar bilan bog‘liq abstrakt tur kiritamiz. Kasrlar ustida quyidagi funksiyalarni kiritamiz.

input() kasr sonni kiritish;

out() kasr soni ekranga chiqarish;

add() kasrlarni qo‘shish;

sub() kasrlarni ayirish;

mult() kasrlarni ko‘paytirish;

divide() kasrlarni bo‘lish;

Quyidagi dastur kiritilgan funksiyalar yordamida kasrlar ishlashga misol bo‘ladi

```
#include <iostream>
using namespace std;
typedef struct rational_fraction{
int numerator;
int denominator;
} fraction;

fraction input(){
int N;
fraction dr;
cout<<"Sutat:";
cin>>dr.numerator;
cout<<"Mahraj:";
cin>>N;
if (N==0){
cout<<"\nXato! No'l mahraj";
exit(0);
}
dr.denominator=N;
return dr;
}
void out( fraction dr){
cout<<"Kasr ";
cout<<dr.numerator<<"\ "<<dr.denominator;
```

```
}
fraction add ( fraction dr1, fraction dr2){
fraction dr;
dr.numerator=dr1.numerator * dr2.denominator+ dr1.denominator *
dr2.numerator;
dr.denominator=dr1.denominator * dr2.denominator;
return dr;
}
fraction sub ( fraction dr1, fraction dr2, fraction * pdr){
fraction dr;
dr. numerator=dr1.numerator * dr2.numerator- dr2.numerator *
dr1.denominator;
dr. denominator= dr1.denominator* dr2.denominator;
return dr;
}
fraction mult ( fraction dr1, fraction dr2 ){
fraction dr;
dr.numerator=dr1.numerator * dr2.numerator;
dr. denominator= dr1.denominator * dr2.denominator;
return dr;
}
fraction divide ( fraction d1,  fraction d2){
fraction dr;
dr.numerator= d1. numerator * d2. denominator;
dr.denominator=d1. denominator * d2. numerator;
return dr;
}
int main(){
fraction a,b,c,d;
fraction p;
cout<<"Kasr kiriting"<<endl;
a=input();
b=input();
c=add(a,b);
out(c);cout<<endl;
c=mult(a,b);
out(c);cout<<endl;
c=divide(a,b);
out(c);
return 0;
}
```

Bitli maydonlar

Bitli maydonlar strukturalarning xususiy holidir. Bitli maydon ta'riflanganda uning uzunligi bitlarda ko‘rsatiladi (butun musbat konstanta).

Misol:

```
#include <iostream>
using namespace std;
struct {
int a:8;
int b:6;
} xx={64,64};
int main() {
cout<<xx.a<<endl;
cout<<xx.b<<endl;
system("pause");
return 0;
}
```

Natija:

64

0

Bitli maydonlar ixtiyoriy butun turga tegishli bo‘lishi mumkin. Bitli maydonlar adresini olish mumkin emas. Xotirada bitli maydonlarni joylashtirish kompilyator va apparaturaga bog‘liq.

Razryadli maydonlar yordamida razryadli massivlar hosil qilish mumkin. Yuqorida ko‘rilgan son hamma bitlarini chiqarish dasturini quyidagicha yozish mumkin:

```
#include <iostream>
using namespace std;
struct bit{
unsigned int i:1;
};
unsigned printbits(int c, struct bit pp[]){
unsigned int i;
unsigned k=8*sizeof(int);
for(i=0;i<k;i++){
pp[i].i=c&1;
cout<<(c&1);
c>>=1;
}
return k;
}
```

```
int main(){
unsigned int k,i;
struct bit pp[100];
k=printbits(5,pp);
cout<<"\n";
for(i=k-1;i>0;i--)
cout<<pp[i].i;
cout<<pp[0].i;
return 0;
}
```

Birlashmalar

Strukturalarga yaqin tushuncha bu birlashma tushunchasidir. Birlashmalar **union** xizmatchi so‘zi yordamida kiritiladi.

Birlashmalarning asosiy xususiyati shundaki, uning hamma elementlari bir xil boshlang‘ich adresga ega bo‘ladi. Birlashmalarning asosiy avfzalliklaridan biri xotira biror qismi qiymatini har xil turdagi qiymat shaklida qarash mumkindir.

Misol:

```
#include <iostream>
using namespace std;
enum paytype{CARD,CHECK};
struct{
paytype ptype;
union{
char card[4];
long check;
};
}info;
int main() {
info.ptype=CHECK;
info.check=77;
switch (info.ptype) {
case CARD:cout<<"\n CARD:"<<info.card;break;
case CHECK:cout<<"\nCHECK:"<<info.check;break;
}
system("pause");
return 0;
}
```

Natija:

CHECK:77

Strukturalar uchun xotira ajratish

Strukturalar uchun xotiradan joy ajratish. Struktura uchun ajratilgan joy hajmini quyidagi amallar yordamida aniqlash mumkin:

Sizeof (strukturali_tur_nomi);

Sizeof (struktura_nomi);

Sizeof struktura_nomi.

Oxirgi holda struktura nomi ifoda deb karaladi. Ifodaning turi aniqlanib, hajmi hisoblanadi.

Misol uchun:

Sizeof (struct goods)

Sizeof (tea)

Sizeof coat

Murakkab turlar, ya'ni massivlar va strukturali turlar uchun xotiraga talab ularning ta'rifiga bog‘liqdir. Masalan, **double array[10]** ta'rif xotiradan **10*sizeof bayt** joy ajratilishiga olib keladi.

```
Struct mixture{
int ii;
long ll;
char cc[8];
};
```

Bu ta'rif har bir **struct mixture** turidagi obyekt xotirada **sizeof(int)+sizeof(long)+8*sizeof(char)** bayt joy egallashini ko‘rsatadi. Oby'ekt aniq hajmini quyidagi amal hisoblaydi:

Sizeof(struct mixture)

Misol:

```
#include <iostream>
using namespace std;
int main(){
double array[10];
cout<<sizeof(array);
system("pause");
return 0;
}
```

Natija:

80

Misol:

```
#include <iostream>
using namespace std;
struct mixture{
int ii;
long ll;
char cc[8];
};
int main(){
cout<<sizeof(mixture);
system("pause");
return 0;
}
```

Xotira ajratilishi. Xotira struktura uchun emas, strukturali obyekt uchun ajratiladi. Har gal obyekt ta'riflanganda, maydonlar uchun xotira ajratiladi. Xotira bir marta ajratiladi. Odatda obyektga ajratilgan xotira hajmi maydonlar hajmlari summasi sifatida aniqlanadi. Lekin xotira zich ajratilmaydi. Odatda xotira ikki mashina so‘zi chegarasi bo‘yicha tekislanadi. Bu xotiraga murojaatni tezlashtiradi. Ba'zi kompilyatorlarda xotira ajratishni **#pragma** direktivasi yordamida boshqarish mumkin. Bu direktiva quyidagi shaklda ishlatiladi:

Pragma pack(n)

Bunda n qiymati 1,2 yoki 4 ga teng bo‘lishi mumkin.

Pack(1) – Baytli tekislash;

Pack(2) – Mashina so‘zi chegarasiga tekislash;

Pack(4) – Ikkilik mashina so‘zi chegarasiga tekislash;

Misol:

```
#include <iostream>
using namespace std;
#pragma pack(1)
struct Alpha{
char c;
int i;
char s;
};
int main (){
cout<<"sizeof(Alpha)="<<sizeof(Alpha)<<endl;
```

```
system("pause");
return 0;
}
```
Natija:
sizeof(Alpha) =6

Nazorat savollari:

1. Struktura umumiy ko‘rinishi.
2. Strukturalar qanday initsializatsiya qilinadi?
3. Struktura elementlariga qanday murojaat qilinadi?
4. Strukturaga ko‘rsatkich ta'rifi.
5. Struktura hajmi qanday o‘lchanadi?
6. Strukturalar tarkibidagi massivlar elementlariga qanday murojaat qilinadi?
7. Strukturalar massivi qanday initsializatsiya qilinadi?
8. Birlashmalar xossalarini ko‘rsating.
9. Razryadli maydon deb qanday maydonlarga aytiladi?
10. Strukturalar orasidagi munosabatlar.

Misollar:

1. Abiturient (ismi, tug‘ilgan yili, yig‘ilgan ball, attestat o‘rta bali) strukturasini yarating. Struktura turidagi massiv yarating. Ko‘rsatilgan raqamli elementni olib tashlang va ko‘rsatilgan familiyali elementdan so‘ng element qo‘shing.

2. Xodim (ismi, lavozimi, tug‘ilgan yili, oyligi) strukturasini yarating. Struktura turidagi massiv yarating. Ko‘rsatilgan familiyali elementni olib tashlang va ko‘rsatilgan raqamli elementdan so‘ng element qo‘shing.

3. Mamlakat (nomi, poytaxt, aholi soni, egallagan maydoni) strukturasini yarating. Struktura turidagi massiv yarating. Ko‘rsatilgan aholi sonidan kichik bo‘lgan elementni olib tashlang va ko‘rsatilgan nomga ega bo‘lgan elementdan so‘ng element qo‘shing.

4. Davlat (nomi, davlat tili, pul birligi, valyuta kursi) strukturasini yarating. Struktura turidagi massiv yarating. Ko‘rsatilgan nomga ega bo‘lgan elementni o‘chiring va fayl oxiriga ikkita element qo‘shing.

5. Inson (ismi, yashash manzili, telefon raqami, yoshi) strukturasini yarating. Struktura turidagi massiv yarating. Ko‘rsatilgan yoshga ega bo‘lgan elementni o‘chiring va berilgan telefon raqamidagi elementdan oldin element qo‘shing.

KO‘RSATKICHLAR

Ko‘rsatkichlar ta'rifi. Ko‘rsatkichlar qiymati konkret turdagi obyektlar uchun xotirada ajratilgan adreslarga tengdir. SHuning uchun ko‘rsatkichlar ta'riflanganda ularning adreslarini ko‘rsatish shart. O‘zgaruvchi ko‘rsatkichlar quyidagicha ta'riflanadi:

<tur> * <ko‘rsatkich nomi>

Ko‘rsatkichlarni ta'riflaganda insializatsiya qilish mumkindir. Initsializatsiya quyidagi shaklda amalga oshiriladi:

<tur> * <ko‘rsatkich nomi>=<konstanta ifoda>

Konstanta ifoda sifatida **&** simvoli yordamida aniqlangan obyekt adresi keladi.

Ko‘rsatkichlar turiga qaramasdan xotirada bir xil joy odatda 4 bayt joy egallaydi.

Ko‘rsatkichlar ustida amallar. Ko‘rsatkichlar ustida quyidagi amallar aniqlangan:

x+n ifoda x+sieof(<>)*n ifodaga ekvivalent

x-n ifoda x-sieof(<>)*n ifodaga ekvivalent

x-y ifoda =((long)x-(long)y)/sizeof(<tur>) ifodaga ekvivalent

x++ ifoda x+=sieof(<>) ifodaga ekvivalent

x-- ifoda x-=sieof(<>) ifodaga ekvivalent

Bu misollarda x,y bir turli ko‘rsatkichlar, n butun son.

Bir xil turdagi ko‘rsatkichlar qiymatlarini ==, !=, <, <=",">, >= amallari orqali solishtirish mumkin. Ko‘rsatkich qiymatlari butun deb qaralib, solishtirish natijasi 0 (yolg‘on) yoki 1 (rost) ga teng.

Misol:

```
#include <iostream>
using namespace std;
int main() {
int i=123;
int *p=&i;
cout<<"\n i="<<i<<" p="<<p;
int j=456;
p=&j;
cout<<"\n j="<<i<<" p="<<p;
return 0;
}
```

Natija:
i=123 p=0012FF60
j=456 p=0012FF48

Misol:

```
#include <iostream>
using namespace std;
int main() {
int a=1,b=2,c=3;
int *p=&b;
cout<<*p<<" "<<*--p<<" "<<*(p+2);
return 0;
}
```

Natija:
2 3 1

Ko‘rsatkichlar xossalari

Konstanta ko‘rsatkich va konstantaga ko‘rsatkichlar. Konstanta ko‘rsatkich quyidagicha ta'riflanadi:

<tur>* const<ko‘rsatkich nomi>=<konstanta ifoda>

Konstanta ko‘rsatkich qiymatini o‘zgartirish mumkin emas, lekin * amali yordamida xotiradagi ma'lumot qiymatini o‘zgartirish mumkin. Konstantaga ko‘rsatkich quyidagicha ta'riflanadi:

<tur>const*<ko‘rsatkich nomi>=<konstanta ifoda>

Bu ko‘rsatkichga * amalini qo‘llash mumkin emas, lekin ko‘rsatkichning qiymatini o‘zgartirish mumkin. Qiymati o‘zgarmaydigan konstantaga ko‘rsatkichlar quyidagicha kiritiladi:

<tur>const* const<ko‘rsatkich nomi>=<konstanta ifoda>

Misol:

```
#include <iostream>
using namespace std;
int main() {
int a=0,b=1;
int* const p1=&a;
int const* p2=&a;
int  const* const p3=&a;
cout<<*p1<<" "<<*p2<<" "<<*p3<<endl;
*p1=5;p2--;
cout<<*p1<<" "<<*p2<<" "<<*p3;
return 0;
}
```

Natija:

0 0 0
5 1 5

Turlashtirilmagan ko‘rsatkich. Turlashtirilmagan (turlashtirilmagan) ko‘rsatkich **void** turiga ega bo‘lib, ixtiyoriy turdagi o‘zgaruvchi adresi qiymat sifatida berilishi mumkin.

Maxsus **void** turidagi ko‘rsatkichlar ajdodiy ko‘rsatkichlar deb atalib, har xil turdagi obyektlar bilan bog‘lanish uchun ishlatiladi. Operator **static_cast** yordamida void turidagi ko‘rsatkichni biror turdagi ko‘rsatkichga keltirish mumkin.

Misol uchun:

```
#include <iostream>
using namespace std;
int main() {
int i=77;
float euler=2.18282;
void *vp;
vp=&i;
cout<<*(int*)vp<<endl;
vp=&euler;
cout<<*(float*)vp;
return 0;
```

```
}
```
Natija:
77
2.18282

Ko‘rsatkichlar funksiya parametri sifatida. Ko‘rsatkichlar yordamida parametr qiymatini o‘zgartirish mumkin.

Keyingi misolda berilgan ikki o‘zgaruvchining qiymatlarini o‘zaro almashtirish funksiyasidan foydalaniladi:

```
#include <iostream>
using namespace std;
void change(int*a,int*b){
int r=*a; *a=*b;*b=r;
}
int main() {
int x=1,y=5;
change(&x,&y);
cout<<x<<" "<<y;
return 0;
}
```
Natija
x=5 y=1

Agar funksiya ichida parametrning o‘zgarishini ta’qiqlash lozim bo‘lib qolsa, bu holda **const** modifikatori qo‘llanadi. Bu modifikatorni funksiyada o‘zgarishi ko‘zda tutilmagan barcha parametrlar oldidan qo‘yish tafsiya qilinadi (undagi qaysi parametrlar o‘zgaradiyu, qaysilari o‘zgarmasligi sarlavhadan ko‘rinib turadi).

Ko‘rsatkichlar va massivlar

Ko‘rsatkich va massiv nomi. Massivlar nomi dasturda konstanta ko‘rsatkichdir. Massivlar bilan ishlanganda qavslarsiz ishlash mumkin.

Quyidagi misolda har bir harf alohida bosib chiqariladi:

Misol:

```
#include <iostream>
using namespace std;
int main() {
char x[]="DIXI";
int i=0;
```

```
while (*(x+i)!='\0')
cout<<*(x+i++)<<" ";
return 0;
}
```
Natija
D I X I

Kompilyator **x[i]** qiymatni hisoblaganda **(x+i)*sizeof(<>)** formula bo‘yicha adresni hisoblab, shu adresdagi qiymatni oladi.

Agar massiv **<tur> x[n][k][m]** shaklda ta'riflangan bo‘lsa, **x[i][j][l]** qiymatni hisoblanganda **(x+k*j+l) *sizeof(<>)** formula bo‘yicha adresni hisoblab shu adresdagi qiymatni oladi.

Massivlarni funksiyalar parametrlari sifatida. Massivlar funksiyaga turidagi bir o‘lchamli massivlar sifatida yoki ko‘rsatkichlar sifatida uzatilishi mumkin. Masalan, satrlar funksiyaga **char** turidagi bir o‘lchamli massivlar sifatida yoki **char*** turidagi ko‘rsatkichlar sifatida uzatilishi mumkin. Oddiy massivlardan farqli o‘laroq, funksiyada satr uzunligi ko‘rsatilmaydi, chunki satr oxirida satr oxiri **‘\0’** belgisi bor.

Misol: Berilgan belgini satrda qidirish funksiyasi

```
#include <iostream>
using namespace std;
int find(char *s,char c){
for (int i=0;i<strlen(s);i++)
if(s[i]==c) return i;
return -1;
}
int main() {
char c[]="DIXI";
cout<<find(c,'I');
return 0;
}
```
Natija:
1

Ko‘p o‘lchamli massivlar va ko‘rsatkichlar. C++ da massivning eng umumiy tushunchasi bu ko‘rsatkichdir, bunda har xil turdagi ko‘rstakich bo‘lishi mumkin, ya'ni massiv har qanday turdagi elementlarga, shu jumladan, massiv

bo‘lishi mumkin bo‘lgan ko‘rsatkichlarga ham ega bo‘lishi mumkin. O‘z tarkibida boshqa massivlarga ham ega bo‘lgan massiv ko‘p o‘lchamli hisoblanadi.

Bunday massivlarni e'lon qilishda kompyuter xotirasida bir nechta turli xildagi obyekt yaratiladi. Masalan, int arr[4][3]

Arr
↓
arr[0]→ arr[0][0] arr[0][1] arr[0][2]
arr[1]→ arr[1][0] arr[1][1] arr[1][2]
arr[2]→ arr[2][0] arr[2][1] arr[2][2]
arr[3]→ arr[3][0] arr[3][1] arr[3][2]

Shunday qilib, **arr[4][3]** ning e'lon qilinishi dasturda uchta turli xildagi obyektlarni yuzaga keltiradi: **arr** identifikatorli ko‘rsatkichni, to‘rtta ko‘rsatkichdan iborat nomsiz massivni va **int** turidagi o‘n ikkita sondan iborat nomsiz massivni. Nomsiz massivlarga kirish huquqiga ega bo‘lish uchun **arr** ko‘rsatkichli adresli ifodalar qo‘llanadi. Ko‘rsatkichlar massivi elementlariga kirish huquqi **arr[2]** yoki ***(arr+2)** shaklidagi indeksli ifodaning bittasini ko‘rsatish orqali amalga oshiriladi. Butun **int** turidagi ikki o‘lchamli sonlar massiviga kirish uchun **arr[1][2]** shaklidagi ikkita indeksli ifoda yoki unga ekvivalent bo‘lgan ***(*(arr+1)+2)** va **(*(arr+1))[2]** shaklidagi ifodalar qo‘llanishi kerak.

Ko‘rsatkichlar massivlari. Bosh funksiya parametrlari

Ko‘rsatkichlar massivlari. Ko‘rsatkichlar massivlari quyidagicha ta'riflanadi:

<tur> *<nom>[<son>]

Misol uchun familiyalar ro‘yxatini kiritish uchun ikki o‘lchovli massivdan foydalanish kerak.

Har xil chegarali jadvallar bilan funksiyalardan foydalanishning bir yo‘li bu oldindan kiritiluvchi konstantalardan foydalanishdir. Lekin asosiy yo‘li ko‘rsatkichlar massivlaridan foydalanish hisoblanadi.

Quyidagi misolda ko‘rsatkichlar massivi yordamida so‘zlar massivi tartiblanadi:

```
#include <iostream>
using namespace std;
void sort(int n, char* a[]){
char* c;
int i,j;
for (i=0;i<n;i++)
for (j=i+1;j<n;j++)
if  (a[i][0]<a[j][0] ){
c=a[i];a[i]=a[j];a[j]=c;
};
};
int main() {
char* pa[10]={"Alimov","Dadashev","Boboev"};
sort(3,pa);
for(int i=0;i<3;i++) cout<<pa[i]<<endl;
return 0;
}
```

Natija:
Dadashev
Boboev
Alimov

Bosh funksiya parametrlari. Har qanday dastur quyidagicha sarlavhaga ega bo‘lishi lozimdir:

int main (int argc, char*argv[ ])

argv – satrlarga ko‘rsatkichlar massivi.

argc– int turidagi parametr argv massividagi elementlar sonini belgilaydi.

Asosiy **main()** funksiyasi parametrlari vazifasi bajarilayotgan dastur bilan operatsion tizim, aniqrog‘i dasturni ishga tushirgan komanda qatori bilan aloqani ta'minlashdan iboratdir.

Misol uchun komanda qatoridan hamma ma'lumotni ekranga so‘zma-so‘z chiqaruvchi dasturni ko‘ramiz.

```
#include<iostream>
using namespace std;
int main (int argc, char* argv[ ] ){
int i;
for (i=0; i<argc ; i++) cout<<  argv [i];
```

```
return 0;
}
```

Odatda ko‘rsatkichlar massivlari **NULL** qiymat bilan tugaydi. Bu xossadan foydalanib, massivni birinchi parametrsiz chiqarish mumkin:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[ ]) {
int i;
for (i=0; argv[i]!=NULL; i++) cout<<"\n"<<argv[i];
return 0;
}
```

Umumiy holda har qanday dastur quyidagicha sarlavhaga ega bo‘lishi lozimdir:

```
int main (int argc, char*argv[ ] ,char*envp[ ])
```

Uchinchi parametr **envp** vazifasi dasturga muhit haqidagi ma'lumotni uzatish uchun ishlatiladi. Bu parametr imkoniyatlarini quyidagi dasturda ko‘ramiz:

```
#include <iostream>
using namespace std;
int main(int argc, char* argv[ ], char*envp[ ]){
int n;
cout<<"\n dastur parametrlar soni"<<argc-1;
for (n=0; n<argc; n++)
cout<<"\n"<<n<<"chi parametr"<<argv[n];
for (n=0; envp[n]; n++)
cout<<"\n"<<envp[n];
return 0;
}
```

Dinamik massivlar

Xotira turlari. C++ tilida o‘zgaruvchilar yo statik tarzda, ya’ni kompilyasiya paytida, yoki standart kutubxonadan funksiyalarni chaqirib olish yo‘li bilan dinamik tarzda, ya’ni dasturni bajarish paytida joylashtirilishi mumkin. Asosiy farq ushbu usullarni qo‘llashda ko‘rinadi – ularning samaradorligi va moslashuvchanligida. Statik joylashtirish samaraliroq, chunki bunda xotirani ajratish dastur bajarilishidan oldin sodir bo‘ladi. Biroq bu usulning moslashuvchanligi ancha past, chunki bunda biz joylashtirilayotgan obyektning

turi va o‘lchamlarini avvaldan bilishimiz kerak bo‘ladi. Masalan, matnli faylning ichidagisini satrlarning statik massivida joylashtirish qiyin: avvaldan uning o‘lchamlarini bilish kerak bo‘ladi. Noma'lum sonli elementlarni oldindan saqlash va ishlov berish kerak bo‘lgan masalalar odatda xotiraning dinamik ajratilishini talab qiladi.

Xotirani dinamik va statik ajratish o‘rtasidagi asosiy farqlar quyidagicha:

- statik obyektlar nomlangan o‘zgaruvchilar bilan belgilanadi hamda ushbu obyektlar o‘rtasidagi amallar to‘g‘ridan-to‘g‘ri, ularning nomlaridan foydalangan holda amalga oshiriladi. Dinamik obyektlar o‘z shaxsiy nomlariga ega bo‘lmaydi va ular ustidagi amallar bilvosita, ko‘rsatkichlar yordamida, amalga oshiriladi;
- statik obyektlar uchun xotirani ajratish va bo‘shatish kompilyator tomonidan avtomatik tarzda amalga oshiriladi. Dasturchi bu haqda o‘zi qayg‘urishi kerak emas. Statik obyektlar uchun xotirani ajratish va bo‘shatish to‘laligicha dasturchi zimmasiga yuklatiladi. Bu anchayin qiyin masala va uni yechishda xatoga yo‘l qo‘yish oson.

Xotirani dinamik taqsimlash. Xotirani dinamik taqsimlash uchun **new** va **delete** amallardan foydalaniladi. Amali **new** tur ismi orqali belgilangan ma'lumotlar turiga mos keluvchi o‘lchamli bo‘sh xotira qismini ajratish va unga murojaat etish imkonini beradi. Dinamik xotira **new** operatori orqali ajratilgandan so‘ng, **delete** operatori tomonidan bo`shatilmaguncha saqlanadi. Agar dinamik xotira dastur bajarilishi tugaguncha bo‘shatilmagan bo‘lsa, avtomatik ravishda dastur tugaganda bo‘shatiladi. Shunga qaramay ajratilgan xotirani dasturda maxsus bo‘shatish dasturlashning sifatini oshiradi.

Ajratilgan xotira qismiga initsializator orqali aniqlangan qiymat kiritiladi.

<tur_nomi> *<nom> =new <tur_nomi> (<initsializator>)

Xotira ajratish new amali orqali oldindan ajratilgan xotira qismi **delete** amali yordamida bo‘shatiladi.

< tur_nomi > *< nom > =new < tur_nomi > (<initsializator>)

C++ tilida **delete** operatori **delete()** funksiyani nol operand uchun chaqirmaydi deyilgan. Shuning uchun 0 ga tekshirish shart emas.

Dinamik massivlar. O‘zgaruvchi o‘lchamli massivlarni shakllantirish ko‘rsatkichlar va xotirani dinamik taqsimlash vositalari yordamida tashkil etiladi.

< tur_nomi > *< nom > =new < tur_nomi > [<initsializator>]

Oldin **new** operatori tomonidan ajratilgan xotirani bo‘shatish uchun **delete** operatoridan foydalaniladi**:**

delete[] <ko‘rsatkich nomi >

Shuni aytish lozimki, qavslar majburiy emas. Agar qavs bo‘lmasa massiv birinchi elementi o‘chiriladi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int *mas=new int[5];
for(int i=0;i<5;i++) cout<<mas[i]<<" ";
cout<<endl;
delete[] mas;
system("pause");
return 0;
}
```

Matritsani shakllantirishda oldin bir o‘lchovli massivlarga ko‘rsatuvchi ko‘rsatkichlar massivi uchun xotira ajratiladi, keyin esa parametrli siklda bir o‘lchovli massivlarga xotira ajratiladi.

Xotirani bo‘shatish uchun bir o‘lchovli massivlarni bo‘shattiruvchi siklni bajarish zarur.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int i,j;
// Dinamik o‘zgaruvchi
int *pi;  pi=new int(10);
cout<<*pi<<endl;delete pi;
// Dinamik massiv
```

```
int *mas=new int[5];
for(i=0;i<5;i++) cout<<mas[i]<<" ";
cout<<endl;
delete[] mas;
// Dinamik matritsa
int n=3;
int **matr=new int*[n];
for (int i=0;i<n;i++)
matr[i]=new int[n];
for(i=0;i<n;i++){
cout<<endl;
for(j=0;j<n;j++) cout<<matr[i][j]<<" ";
};
for(i=0;i<n;i++)
delete [] matr[i];
delete [] matr;
return 0;
}
```

Funksiyaga ko‘rsatkich

Funksiyalarni chaqirishda foydalanish. C++ tili sintaksisiga ko‘ra, funksiyaga ko‘rsatkich funksiya adresini aks ettiruvchi o‘zgaruvchi yoki ifodadir. Funksiyaga ko‘rsatkich bajariluvchi qiymati funksiya kodining birinchi bayti adresidir. Funksiyaga ko‘rsatkichlar ustida arifmetik amallar bajarish mumkin emas. Eng keng qo‘llanuvchi funksiyaga konstanta ko‘rsatkich funksiyaning nomidir. Funksiyaga o‘zgaruvchi ko‘rsatkich funksiya ta'rifi va prototipidan alohida kiritiladi. Funksiyaga o‘zgaruvchi ko‘rsatkich quyidagi ko`rinishda keltiriladi:

<funksiya turi> (* ko‘rsatkich nomi)(parametrlar spetsifikatsiyasi)

Bu ta'rifda qavslar muhim ahamiyatga ega, chunki qavslar yozilmasa, bu ta'rif parametrsiz funksiya prototipi deb qaraladi. Funksiyaga o‘zgaruvchi ko‘rsatkich qiymatlari sifatida, bir xil turga ega bo‘lgan har xil funksiyalar adreslarini berish mumkin.

Qiymati biror funksiya adresiga teng bo‘lgan funksiyaga o‘zgaruvchi ko‘rsatkich shu funksiyaga murojaat qilish uchun ishlatilishi mumkin:

```
#include <iostream>
using namespace std;
```

```
void f1(){
cout<<"\nf1 funkciya bajarildi";
};
void f2(){
cout<<"\nf2 funkciya bajarildi";
};
int main(){
void (*ptr) (void);
ptr=f1;
(*ptr)();
ptr=f2;
(*ptr)();
return 0;
}
```

Natija:
f1 funkciya bajarildi
f2 funkciya bajarildi

Dasturda funksiyaga konstanta ko‘rsatkich, ya'ni nomlari orqali va o‘zgaruvchi ko‘rsatkichlar yordamida murojaat qilishning hamma usullari ko‘rsatilgandir. Shuni ta'kidlash lozimki, adres olish * amali qo‘llanilganda qavslar ishlatish shartdir.

Funksiyaga o‘zgaruvchi ko‘rsatkich ta'riflanganda insializatsiya qilish, ya'ni boshlang‘ich qiymat sifatida o‘zgaruvchi ko‘rsatkich bilan bir xil turga ega bo‘lgan funksiya adresini ko‘rsatish mumkin.

Funksiyaga ko‘rsatkich parametr sifatida. Funksiyaga ko‘rsatkich funksiya parametri sifatida kelishi mumkin. Bu funksiyani parametr sifatida uzatishga imkon beradi.

Misol:

```
#include <math.h>
#include<iostream>
using namespace std;
double kub(double x){
return x*x*x;
}
double rec(double (*pf)(double x),double a){
return pf(a);
}
int main(){
float a=-2;
```

```
double c=rec(kub,a);
cout<<c<<endl;
system("pause");
return 0;
}
```
Natija:
-8

Funksiyaga ko‘rsatkichlar massivlari. C tilida funksiyalar massivlarni yaratish mumkin emas, lekin funksiyaga ko‘rsatkichlar masivlarini kiritish mumkin. Bunday massivlar quyidagicha ta'riflanadi:

<tur>(*massiv_nomi[hajm])(parametrlar spetsifikatsiyasi)

Tur – funksiyalar qaytaradigan qiymatlar turlari;

Massiv - nomi ixtiyoriy identifikator;

Hajm – massiv elemenlari soni;

Parametrlar spetsifikatsiyasi- funksiyalar parametrlari nomlari va turlarini aniqlaydi.

Bu massiv elementlari qiymatlari quyidagi prototiplarga ega bo‘lgan funksiyalar adreslaridir: **int funksiya_nomi (char)**.

Funksiyaga ko‘rsatkichlar massivlari initsializatsiya qilinishi mumkin.

Indeks konkret qiymati bo‘yicha massiv elementlariga va konkret funksiyalarga murojaat quyidagi shaklda amalga oshiriladi:

Massiv_nomi[indeks] (haqiqiy parametrlar_ro‘yxati);

(* Massiv_nomi[indeks]) (haqiqiy parametrlar_ro‘yxati);

Funksiyaga ko‘rsatkichlar massivlari menyular yaratishda qulaydir. Quyida shunday menyu misolini ko‘ramiz:

```
#include <iostream>
#define n 2
using namespace std;
void act0(char* name ){
cout<<"Ish tugadi!\n"<<name;
}
void act1(char* name){
cout<<"Ish 1\n"<<name;
}
void act2(char* name ){
```

```
cout<<"Ish 2\n"<<name;
}
int main(){
void (*pact[])(char*)={act0,act1,act2};
char string[12]="Bajaruldi";
int number;
cout<<"\n Ish nomerini kiriting 0 dan "<<n<<" gacha";
while(1){
cin>>number;
pact[number](string);
if (number==0) break;
}
return 0;
}
```

Funksiyaga ko‘rsatkich funksiya qiymati sifatida. Menyular tashkil qilganda funksiyaga ko‘rsatkichlarni qiymat sifatida qaytaruvchi funksiyalardan foydalanish qulaydir. Shu usulda menyu tashkil qilishni ko‘rsatuvchi dasturni ko‘rib chiqamiz. Dasturda uchta funksiya kiritilgan: **int f(void)** prototipiga ega bo‘lgan **f1()** va **f2()** funksiyalari va **int (*menu(void)) (void)** prototipiga ega bo‘lgan **menu()** funksiyasi. **Menu()** funksiyasi bajarilganda, **f1()** va **f2()** funksiyalarga mos keluvchi menyu punktlaridan birini tanlash imkoni beriladi. Shu punktlardan biri tanlanganda, mos funksiya adresi qaytariladi, agar punktlar noto‘g‘ri tanlangan bo'lsa, **NULL** qiymati qaytariladi. Bu qiymatlar asosiy dasturda **r** ko‘rsatkichga uzatiladi.

Agar qiymat **NULL** bo‘lsa, **“The End”** ma'lumoti chiqarilib, dastur bajarilishi to‘xtatiladi, aks holda **t=(*r)()** murojaat adresi qaytarilgan funksiya bajarilishiga olib keladi.

```
#include <iostream>
using namespace std;
int f1(void){
cout<<"the first actions: ";
return 1;
}
int f2(void){
cout<<"the second actions: ";
return 2;
}
```

```
int (*menu(void))(void){
int choice;
int (*menu_items[])()={f1,f2};
cout<<"\n Pick the menu item (1 or 2): ";
cin>>choice;
if (choice<3&&choice>0)
return menu_items[choice-1];
else
return NULL;
}
int main(){
int (*r)(void);
int t;
while(1){
r=menu();
if (r==NULL){
cout<<"\nThe End!";
return 0;
}
t=(*r)();
cout<<"|tt="<<t;
}
return 0;
}
```

Strukturaga ko‘rsatkichlar

Strukturaga ko‘rsatkichlar. Strukturaga ko‘rsatkichlar oddiy ko‘rsatkichlar kabi tasvirlanadi.

Strukturaga ko‘rsatkich ta'riflanganda initsializatsiya qilinishi mumkin.

Ko‘rsatkich orqali struktura elementlariga ikki usulda murojaat qilish mumkin. Birinchi usul adres bo‘yicha qiymat olish amaliga asoslangan bo‘lib, quyidagi shaklda qo‘llaniladi:

(* strukturaga ko‘rsatkich).element nomi;

Ikkinchi usul maxsus strelka (->) amaliga asoslangan bo‘lib, quyidagi ko‘rinishga ega:

strukturaga ko‘rsatkich->element nomi

Misol:

```
#include <iostream>
using namespace std;
```

```
struct Person{
char name[20];
char title[30];
float rate;
}emp={"Smit", "director",10000};
int main(){
Person *p=&emp;
cout<<p->name<<endl;
cout<<p->title<<endl;
cout<<(*p).rate<<endl;
system("pause");
return 0;
}
```

Natija:
Smit
director
10000

Strukturaga ko‘rsatkich parametr sifatida. Funksiyada struktura qiymatini o‘zgartirish uchun adres orqali uzatish lozim.

```
#include <iostream>
using namespace std;
struct point{
int  x,y;
};
void print_point(point a){
cout<<a.x<<" "<<a.y;
}
void input_point(point *pa){
cin>>pa->x>>pa->y;
}
int main(){
struct point a;
input_point(&a);
print_point(a);
return 0;
}
```

Strukturalarga ko‘rsatkichlar ustida amallar. Strukturalarga ko‘rsatkichlar ustida amallar oddiy ko‘rsatkichlar ustida amallardan farq qilmaydi. Agar ko‘rsatkichga strukturalar massivining biror elementi adresi qiymat sifatida berilsa, massiv bo‘yicha uzluksiz siljish mumkin bo‘ladi.

Misol tariqasida kompleks sonlar massivi summasini hisoblash masalasini ko‘rib chiqamiz:

```
#include <iostream>
using namespace std;
struct complex {
float x;
float y;
};
int main(){
struct complex array[]={1.0,2.0,3.0,-4.0,-5.0,-6.0};
struct complex summa={0.0,0.0};
struct complex *point=&array[0];
int k,i;
k=sizeof(array)/sizeof(array[0]);
for(i=0;i<k;i++){
summa.x+=point->x;
summa.y+=point->y;
point++;
}
cout<<"\n Summa: real="<<summa.x;
cout<<" imag="<<summa.y;
return 0;
}
```

Dastur bajarilishi natijasi:
Summa: real=-1 imag=-8

Nazorat savollari:

1. Ko‘rsatkich ta'rifini keltiring.
2. Ilova ko‘rsatkichdan qanday farq qiladi?
3. Ko‘rsatkichlar bilan bog‘liq amallarni keltiring.
4. Ko‘rsatkich va massiv nomi orasida qanday farq bor?
5. Ko‘rsatkichlar massivi qanday ta'rif qilinadi?
6. Dinamik xotira ajratish va bo‘shatish funksiyalari.
7. Dinamik massivlar oddiy massivlardan qanday farq qiladi?
8. Dinamik massivlarni hosil qilish usullarini ko‘rsating.
9. Funksiyaga ko‘rsatkich ta'rifi umumiy ko‘rinishi.
10. Funksiyaga ko‘rsatkich massivlari.

Misollar:

1. Berilgan satr o‘zgaruvchi ekanligini aniqlovchi funksiya tuzing va dasturda foydalaning. Funksiya tanasida faqat ko‘rsatkichlar ustida amallardan foydalaning.

2. Matritsani vektorga ko‘paytirish funksiyasini yaratib dasturda foydalaning. Matritsa ko‘rsatkichlar massivi sifatida kiritilsin.

3. Dinamik ravishda o‘quvchilar familiyalari va baholari massivlarini hosil qiluvchi funksiyalar yarating. Hamma a'lochilar familiyalarini chiqaruvchi funksiya tuzib dasturda foydalaning.

4. Uchburchak dinamik massiv yordamida Paskal uchburchagini hisoblovchi funksiya tuzing. Bu funksiyadan tashqari uchburchakni ekranga chiqapuvchi funksiya tuzib dasturda foydalaning.

5. Dixotomiya usuli yordamida f(x)=0 tenglamani yechish uchun funksiya tuzing. Funksiyaga ko‘rsatkich parametr sifatida uzatilsin.

Preprotsessor vositalari

Preprotsessor tushunchasi

Dastur matni va preprotsessor. C++ tilida matnli fayl shaklida tayyorlangan dastur uchta qayta ishlash bosqichlaridan o‘tadi:

1. Matnni preprotsessor direktivalari asosida o‘zgartilishi. Bu jarayon natijasi yana matnli fayl bo‘lib, preprotsessor tomonidan bajariladi.
2. Kompilyasiya. Bu jarayon natijasi mashina kodiga o‘tkazilgan obyektli fayl bo‘lib, kompilyator tomonidan bajariladi.
3. Bog‘lash. Bu jarayon natijasi to‘la mashina kodiga o‘tkazilgan bajariluvchi fayl bo‘lib, bog‘lagich (komponovshik) tomonidan bajariladi.

Preprotsessor vazifasi dastur matnini preprotsessor direktivalari asosida o‘zgartirishdir. Masalan **define** direktivasi dasturda bir jumlani ikkinchi jumla

bilan almashtirish uchun ishlatiladi. Bu direktivaning umumiy ko‘rinishi quyidagicha:

#define <almashtiruvchi ifoda> <almashinuvchi ifoda>

Bu direktiva bajarilganda dastur matnidagi almashtiruvchi ifodalar almashinuvchi ifodalarga almashtiriladi.

Misol:

```
#include <iostream>
using namespace std;
#define eyler 2.718282
int main() {
double d=2*eyler;
cout<<d;
return 0;
}
```

Natija:
5.43656

include direktivasi ikki ko‘rinishda ishlatilishi mumkin:

1. **#include “*fayl nomi*”** direktivasi dasturning shu direktiva o‘rniga qaysi matnli fayllarni qo‘shish kerakligini ko‘rsatadi.
2. **#include <*fayl nomi*>** direktivasi dasturga kompilyator standart kutubxonalariga mos keluvchi sarlavhali fayllar matnlarini qo‘shish uchun mo‘ljallangandir. Bu fayllarda funksiya prototipi, turlar, o‘zgaruvchilar, konstantalar ta'riflari yozilgan bo‘ladi. Shuni ta'kidlash lozim-ki, bu direktiva dasturga standart kutubxona qo‘shilishiga olib kelmaydi. Standart funksiyalarning kodlari bog‘lash, ya'ni aloqalarni tahrirlash bosqichida, kompilyatsiya bosqichidan so‘ng amalga oshiriladi.

Kompilyasiya bosqichida sintaksis xatolar tekshiriladi va dasturda bunday xatolar mavjud bo‘lmasa, standart funksiyalar kodlarisiz mashina kodiga o`tkaziladi.

Sarlavhali fayllarni dasturning ixtiyoriy joyida ulash mumkin bo‘lsa ham, bu fayllar odatda dastur boshida qo‘shish lozimdir. Shuning uchun bu fayllarga sarlavhali fayl (**header file**) nomi berilgandir.

Dastur matnini kompilyasiya qilish. Dastur kodini bajariluvchi faylga o‘tkazish uchun kompilyatorlar qo‘llaniladi. Kompilyator qanday chaqiriladi va unga dastur kodi joylashgan joyi haqida qanday xabar qilinadi, bu konkret kompilyatorga bog‘liqdir. Bu ma'lumotlar kompilyatorning dokumentatsiyasida berilgan bo‘ladi.

Dastur kodi kompilyasiya qilinishi natijasida obyektli fayl hosil qilinadi. Bu fayl odatda **.obj** kengaytmali bo‘ladi. Lekin bu hali bajariluvchi fayl degani emas. Obyektli faylni bajariluvchi faylga o‘girish uchun yig‘uvchi dastur qo‘llaniladi.

Yig‘uvchi dastur yordamida bajariluvchi faylni hosil qilish. C++ tilida dasturlar odatda bir yoki bir nechta obyektli fayllar yoki kutubxonalarni komponovka qilish yordamida hosil qilinadi. Kutubxona deb, bir yoki bir nechta komponovka qilinuvchi fayllar to‘plamiga aytiladi. C++ ning barcha kompilyatorlari dasturga qo‘shish mumkin bo‘lgan funksiyalar (yoki protseduralar) va sinflardan iborat kutubxona hosil qila oladi. Funksiya – bu ayrim xizmatchi amallarni, masalan, ikki sonni qo‘shib, natijasini ekranga chiqarishni bajaruvchi dastur blokidir. Sinf sifatida ma'lumotlar to‘plami va ularga bog‘langan funksiyalarni qarash mumkin.

Demak, bajariluvchi faylni hosil qilish uchun quyida keltirilgan amallarni bajarish lozim:

- **.cpp** kengaytmali dastur kodi hosil qilinadi;
- Dastur kodini kompilyasiya qilish orqali **.obj** kengaytmali obyektli fayl tuziladi;
- Bajariluvchi faylni hosil qilish maqsadida **.obj** kengaytmali fayli zaruriy kutubxonalar orqali komponovka qilinadi.

Fayllardan matnlar qo‘shish. Fayldan matn qo‘shish uchun uch shaklga ega bo`lgan **#include** operatori qo`llaniladi:

#include <fayl nomi>

#include "fayl nomi"

#include makros nomi

Makros nomi **#define** direktivasi orqali kiritilgan preprotsessor identifikatori yoki makros bo‘lishi mumkin.

Agar birinchi shakl qo‘llanilsa, preprotsessor qo‘shilayotgan faylni standart kutubxonalardan izlaydi.

Agar ikkinchi shakl qo‘llanilsa, preprotsessor foydalanuvchining joriy katalogini ko‘rib chiqadi va bu katalogda fayl topilmasa, standart tizimli kataloglarga murojaat qiladi.

Agar dasturda bir necha funksiyalardan foydalanilsa, funksiyalar ta'rifi, tanasi bilan birga alohida fayllarda saqlash qulaydir. Hamma funksiyalar tanasiga va **main()** funksiyasi tanasiga chaqirilayotgan funksiyalar prototiplari joylashtirilsa, dastur tanasida funksiyalarni ixtiyoriy joylashtirish mumkin. Bu holda dastur faqat protsessor komandalaridan ham iborat bo‘lishi mumkin.

C va C++ standarti bo‘yicha **.h** suffiksi kutubxonaga tegishli funksiyalarning prototiplari hamda, turlar va konstantalar ta'rifi joylashgan fayllarni ko‘rsatadi. Bunday fayllarni sarlavhali fayllar deb ataladi. Kompilyator kutubxonalari bilan ishlashga mo‘ljallangan sarlavhali fayllar ro‘yxati til standartida ko‘rsatilgan bo‘lib, bu fayllar nomlari tilning xizmatchi so‘zlari hisoblanadi.

Quyida shu standart fayllar nomlari keltirilgan:

Assert.h – dastur diagnostikasi.

Type.h – simvollarni o‘zgartirish va tekshirish.

Error – xatolarni tekshirish.

Float.h – haqiqiy sonlar bilan ishlash.

Limits.h – butun sonlarning chegaralari.

Locate.h – milliy muhitga moslash.

Match.h – matematik hisoblashlar.

Setjump.h – nolokal o‘tishlar imkoniyatlari.

Signal.h – g‘ayrioddiy holatlar bilan ishlash.

Stdarg.h – o`zgaruvchi sonli parametrlarni qo‘llash.

Stddef.h – qo‘shimcha ta'riflar.

Stdlib.h – xotira bilan ishlash.

String,h – simvolli qatorlar bilan ishlash.

Time.h – sana va vaqtni aniqlash.

Turbo C va Borland C++ kompilyatorlarida grafik kutubxona bilan bog‘lanish uchun **graphic.h** – sarlavhali fayl qo‘llaniladi.

Shartli direktivalar. Shartli direktiva quyidagi ko‘rinishga egadir:

#if ***butun_sonli_ifoda***

matn_1

#else

matn_2

#endif

#else ***matn_2*** qismi ishlatilishi shart emas.

Direktiva bajarilganda **#if** dan so‘ng yozilgan butun sonli ifoda qiymati hisoblanadi. Agar bu qiymat 0 dan katta bo‘lsa, **matn_1** kompilyatsiya qilinayotgan matnga qo‘shiladi, aksincha **matn_2** qo‘shiladi. Agar **#else** direktivasi va **matn_2** mavjud bo‘lmasa, bu direktiva o‘tkazib yuboriladi.

#ifdef ***identifikator***

direktivasida **#define** direktivasi yordamida identifikator aniqlanganligi tekshiriladi. Agar identifikator aniqlangan bo‘lsa, matn_1 bajariladi.

#ifndef ***identifikator***

direktivasida aksincha, shart rost hisoblanadi, agar identifikator aniqlanmagan bo‘lsa.

Dasturga ulash mo‘ljallangan fayllarning har biriga bitta fayl ulanish mo‘ljallangan bo‘lsa, bu fayl bir necha marta dasturga ulanib qoladi. Bu qayta ulanishni oldini olish uchun standart fayllar yuqorida ko‘rilgan direktivalar yordamida himoya qilingandir.

Tarmoqlanuvchi shartli direktivalar yaratish uchun quyidagi direktiva kiritilgan:

#elif ***butun_sonli_ifoda***

Bu direktiva ishlatilgan matn strukturasi:

#if shart

```
matn
#elif 1_ifoda
1_matn
#elif 2_ifoda
2_matn
...
#else
matn
#endif
```

Preprotsesssor avval **#if** direktivasidagi shartni tekshiradi. Agar shart 0 ga teng bo‘lsa, **1_ifoda** hisoblanadi, agar u ham 0 bo‘lsa, **2_ifoda**ni hisoblaydi va hokazo.

Agar hamma ifodalar 0 bo‘lsa, else uchun ko‘rsatilgan matn ulanadi. Agar biror ifoda 0 dan katta bo‘lsa, shu direktivada ko‘rsatilgan matn ulanadi.

Yordamchi direktivalar. Quyidagi qo‘shimcha direktivalar mavjud:

#line ***konstanta*** satrlarni nomerlash uchun ishlatiladi.

#line ***konstanta “fayl nomi”*** fayl nomini o‘zgartirish uchun ishlatiladi.

#error ***leksemalar ketma-ketligi*** diagnostik ma'lumotlarni chiqarish uchun ishlatiladi.

bo‘sh direktiva.

Direktivaga misol. Misol tariqasida nuqta tushunchasini aniqlovchi **point** strukturasini yaratib, **point.h** fayliga yozib qo‘yamiz:

```
#ifndef POINTH
#define POINTH 1
struct point{
int x,y;
};
void show(point a){
cout<<a.x<<" "<<a.y;
};
#endif
```

Kelgusida **point** strukturasini boshqa sinflarga qo‘shish mumkin bo‘lgani uchun shartli protsessor direktivasi **#ifndef point.h** ishlatilgan. Protsessorli identifikator **point.h** **#define point.h** direktivasi orqali kiritilgan.

Shuning uchun **#include "point.h"** direktivasi bir necha marta ishlatilganda ham, **point** strukturasi ta'rifi matni faqat bir marta kompilyatsiya qilinayotgan faylda paydo bo‘ladi.

Kiritilgan **point** strukturasi va funksiyalari bilan ishlovchi dasturni keltiramiz:

```
#include <iostream>
using namespace std;
#include "point.h"
#include "point.h"
int main(){
point A={200,50};
show(A);
return 0;
}
```

Oldindan kiritilgan makronomlar. Protsessorga qayta ishlash jarayonida ma'lumot beruvchi quyidagi oldindan kiritilgan makronomlar mavjuddir:

__LINE__ – qiymati o‘nlik konstanta o‘qilayotgan satr nomeri. Birinchi satr nomeri 1 ga teng.

__FILE__ – fayl nomi simvollar qatori sifatida. Preprotsessor har gal boshqa fayl nomi ko‘rsatilgan **#include** direktivasini uchratganda nom o‘zgaradi. **#include** direktivasi bajarilib bo‘lgandan so‘ng, nom qayta tiklanadi.

__DATE__ – “Oy, sana, yil” formatidagi simvollar satri. Fayl bilan ishlash boshlangan sanani ko‘rsatadi.

__TIME__ – “Soatlar : minutlar : soniyalar” formatidagi simvollar satri. Preprotsessor tomonidan faylni o‘qish boshlangan vaqtni ko‘rsatadi.

__STDC__ – Agar kompilyator ANSI-standart bo‘yicha ishlayotgan bo’lsa, qiymati 1 ga teng konstanta, aks holda konstanta qiymati aniqlanmagan.

Misol:

```
#include <iostream>
using namespace std;
int main(){
cout<<__LINE__<<endl;
cout<<__STDC__<<endl;
}
```

Natija:

4

1

Misol:

```
#include <iostream>
using namespace std;
int main(){
cout<<__DATE__<<endl;
cout<<__TIME__<<endl;
cout<<__FILE__<<endl;
}
```

Xotira sinflari

Avtomatik obyektlar. Avtomatik xotira obyektlari faqat o‘zi aniqlangan blok ichida mavjud bo‘ladi. Blokdan chiqishda obyektlar uchun ajratilgan xotira qismi bo‘shatiladi, ya'ni obyektlar yo`qoladi. Shunday qilib, avtomatik xotira har doim ichki xotiradir, ya'ni bu xotiraga o‘zi aniqlangan blokda murojaat qilish mumkin. Avtomatik xotira obyektlari **auto** so‘zi yordamida ta'riflanadi.

Agar maxsus ko‘rsatilmagan bo‘lsa, o‘zgaruvchi har doim avtomatik xotira turiga tegishli deb hisoblanadi.

Registrli obyektlar. Registrli obyektlar avtomatik obyektlardan farq qilmaydi, faqat ular uchun registrlarda joy ajratiladi. Registrli xotira obyektlari **register** suzi yordamida ta'riflanadi.

Statik obyektlar. Statik xotira obyektlari blokdan chiqilgandan so‘ng ham mavjud bo‘lib qolaveradi. Statik xotira obyektlari **static** xizmatchi so‘zi yordamida ta'riflanadi. Statik o‘zgaruvchilarga jimlik bo‘yicha nol boshlang‘ich qiymat beriladi.

Misol:

```
#include<iostream>
using namespace std;
void autofunc(void) {
static int K=1;
cout<<" K="<<K;
K++;
return;
}
int main(){
int i;
```

```
for (i=0;i<5;i++)
autofunc();
cin.get();
return 0;
}
```
Natija:
K=1 K=2 K=3 K=4 K=5

Tashqi obyektlar. Dastur bir necha matnli fayllarda joylashgan bo‘lishi mumkin. Dastur o‘z navbatida funksiyalardan iboratdir. Hamma funksiyalar tashqi hisoblanadi. Hatto har xil fayllarda joylashgan funksiyalar ham har xil nomlarga ega bo‘lishi lozimdir. Funksiyalardan tashqari dasturlarda tashqi obyektlar o‘zgaruvchilar, ko‘rsatkichlar, massivlar ishlatilishi mumkin.

Tashqi obyektlar hamma funksiyalarda ham ko‘rinmasligi mumkin. Agar obyekt fayl boshida ta'riflangan bo‘lsa, u shu fayldagi hamma funksiyalarda ko‘rinadi.

Agar tashqi obyektga shu obyekt ta'riflangan blokdan yuqorida yoki boshqa faylda joylashgan blokdan murojaat qilinishi kerak bo‘lsa, bu obyekt **extern** xizmatchi so‘zi yordamida ta'riflanshi lozimdir. Shuni aytish lozim-ki, **extern** xizmatchi so‘zi yordamida ta'riflanganda initsializatsiya qilish yoki chegaralarni ko‘rsatish mumkin emas.

Misol:

1-faylda:

```
int sp=10;
```

2-faylda:

```
#include <iostream>
#include "Untitled2.cpp"
using namespace std;
void print(int k){
cout<<k<<endl;
}
int main() {
extern int sp;
extern void print(int );
print(sp);
system("pause");
```

```
return 0;
}
```

O‘zgaruvchan e'lonlar

Operatsion tizim, apparat qurilmalari yoki boshqa oqim boshqarilishi oqibatida qiymati o‘zgartirilishi mumkin bo‘lgan o‘zgaruvchini e'lon qilishda **volatile** modifikatori qo‘llanadi:

volatile <tur><obyekt nomi>;

C++da **volatile** kalit so‘zning qo‘llanishi sinflar va a'zo-funksiyalarga ham tegishlidir. Bu kalit so‘z ko‘rsatilgan obyekt qiymatiga nisbatan taxminlar qilishni kompilyatorga ta'qiqlaydi, chunki bunday qilinsa, ushbu obyektni o‘z ichiga olgan ifodalarni hisoblashda, uning qiymati har bir daqiqada o‘zgarib ketishi mumkin. Bundan tashqari o‘zgarib turadigan o‘zgaruvchi registr modifikatori bilan e'lon qilinishi mumkin emas. Qiymat o`zgarishi mumkin bo`lganligi sababli, kompilyator har gal uni xotiradan yuklaydi.

Dinamik xotira. Dastur bajarilish davomida ajratilgan xotira qismiga murojaat imkoniyati shu qismga adreslovchi ko‘rsatkichga bog‘liqdir. Shunday qilib, biror blokda ajratilayotgan dinamik xotira bilan ishlashning quyidagi uchta varianti mavjuddir:

- Ko‘rsatkich – avtomatik xotira turiga kiruvchi lokal obyekt. Bu holda ajratildgan xotiraga blokdan tashqarida murojaat qilib bo‘lmaydi, shuning uchun blokdan chiqishda bu xotirani bo‘shatish lozimdir.
- Ko‘rsatkich – avtomatik xotira turiga kiruvchi lokal statik obyekt. Blokda bir marta ajratilgan dinamik xotiraga, blokka har bir qayta kirilganda ko‘rsatkich orqali murojaat qilish mumkindir. Xotirani blokdan foydalanib bo‘lgandan so‘ng bo‘shatish lozimdir.
- Ko‘rsatkich blokka nisbatan global obyektdir. Dinamik xotiraga ko‘rsatkich ko‘rinuvchi hamma bloklardan murojaat qilish mumkin. Xotirani faqat undan foydalanib bo‘lgandan so‘ng bo‘shatish lozimdir.

Ikkinchi variantga misol keltiramiz, bu misolda dinamik xotira obyekti ichki statik ko‘rsatkich bilan bog‘liqdir:

```
#include <iostream>
using namespace std;
void dynamo(void){
static char *uc=NULL;
if (uc==NULL){
uc=new char(1);
*uc='A';
}
cout<<*uc<<" ";
(*uc)++;
return;
};
int main(){
int i;
for (i=0;i<5;i++)
dynamo();
return 0;
}
```

Dastur bajarilishi natijasi:

A B C D E

Bu dasturning kamchiligi ajratilgan xotira bo‘shatilmasligidir.

Keyingi dasturda dinamik xotiraga ko‘rsatkich global obyektdir:

```
#include <iostream>
using namespace std;
char *uk=NULL;
void dynam1(void){
cout<<*uk<<" ";
(*uk)++;
return;
};
int main(){
int i;
uk=new char(1);
*uk='A';
for (i=0;i<5;i++){
dynam1();
(*uk)++;
}
delete uk;
return 0;
}
```

Dastur bajarilishi natijasi:

A C E G I

Dinamik obyekt asosiy dasturda yaratilib, **uk** ko‘rsatkich bilan bog‘liqdir. Dasturda boshlang‘ich 'A' qiymatga ega bo‘ladi. Ko‘rsatkich global bo‘lgani uchun dinamik obyektga **main()** va **dynam1()** funksiyalarida murojaat qilish mumkin.

Dinamik xotiraga ajratilgandan so‘ng, shu obyekt bilan bog‘liq ko‘rsatkich tashqi obyekt sifatida ta'riflangan ixtiyoriy blokda murojaat qilish mumkin.

Xotira ajratish funksiyalari

Dinamik xotira ajratish. Dinamik xotira ajratish uchun C tilida **malloc** funksiyasidan foydalanish lozim. Bu funksiya argumenti ajratilishi lozim bo‘lgan baytlar soni bo‘lib, ***void** turidagi ajratilgan xotiraga ko‘rsatkich qaytaradi.

void *malloc(size_t size);

Funksiya uyumli xotiradan (**size**) baytga teng blok ajratadi. Agar ajratish muvaffaqiyatli bo‘lsa, ajratilgan xotiraga ko‘rsatkich qaytaradi, aks holda (blok ajratish mumkin bo‘lmasa yoki hajmi 0 bo‘lsa), **null** qaytaradi.

Agar ajratilgan xotirani bo‘shatilmasa, bu xotira foydalanilmay qoladi, agar uning ko‘rsatkichi o‘z qiymatini o‘zgartirgan bo‘lsa, uni tizimga qaytarish mumkin emas. Bu hodisa xotiraning yo‘qotilishi, degan maxsus nom bilan ataladi. Pirovard natijada dastur xotira yetishmagani tufayli avariya holatida tugallanadi (agar u ancha vaqt ishlayversa).

Dinamik ajratilgan xotirani bo‘shatish uchun **free** funksiyasidan foydalanish lozim:

void free(void *block);

Shuni hisobga olish lozim-ki, bu funksiya yordamida bo‘shatilgan xotirani yana bo‘shatish mumkin emas.

Misol:

```
#include <iostream>
using namespace std;
int main(){
char *str;
if ((str = (char *) malloc(10)) == NULL){
cout<<"Not enough memory to allocate buffer\n";
```

```
exit(1);
}
strcpy(str, "Hello");
cout<<"String is "<<str;
free(str);
return 0;
}
```
Natija:
String is Hello

Xotira ajratish funksiyalari. Xotira ajratib, initsiallash uchun **calloc** funksiyasidan foydalaniladi. Bu funksiyadan foydalanish uchun **<STDLIB.H>** sarlavhali faylni ulash lozim.

void *calloc(size_t nitems, size_t size);

Funksiya **calloc** uyumli xotiradan (**nitems * size**) baytga teng blok ajratadi va 0 qiymat bilan to‘ldiradi. Blok 64K dan oshmasligi lozim. Agar ajratish muvaffaqiyatli bo‘lsa, ajratilgan xotiraga ko‘rsatkich qaytaradi, aks holda (blok ajratish mumkin bo‘lmasa yoki hajmi 0 bo‘lsa), **null** qaytaradi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
char *str = NULL;
str = (char *) calloc(10, sizeof(char));
strcpy(str, "Hello");
cout<<"String is "<<str;
free(str);
return 0;
}
```
Natija:
String is Hello

Xotirani qayta ajratish. Xotirani qayta ajratish uchun **realloc** funksiyasidan foydalaniladi. Bu funksiyadan foydalanish uchun **<STDLIB.H>** sarlavhali faylni ulash lozim.

void *realloc(void *block, size_t size);

Funksiya **realloc** ajratilgan xotira hajmini o‘zgartiradi. Kerak bo‘lsa, yangi joyga nusxa oladi.

Agar qayta ajratish muvaffaqiyatli bo‘lsa, qayta ajratilgan xotiraga ko‘rsatkich qaytaradi, aks holda (blok ajratish mumkin bo‘lmasa yoki hajmi 0 bo‘lsa), **null** qaytaradi.

```
#include <iostream>
using namespace std;
int main(){
char *str = (char *) malloc(10);
strcpy(str, "Hello");
cout<<"String is "<<str<<endl;
str = (char *) realloc(str, 20);
cout<<"String is "<<str;
free(str);
return 0;
}
```

Makroslar

Makros tushunchasi. Makros ta'rifiga ko‘ra, bir jumlani ikkinchi jumla bilan almashtirishdir. Makroslar ko‘pincha makrota'rif deb ham ataladi. Eng sodda makrota'rif

#define *identifikator_almashtiruvchi_satr*.

Parametrli makrota'riflardan foydalanish yanada kengroq imkoniyatlar yaratadi:

 # define nom (parametrlar_ro‘yxati) almashtiriluvchi_qator

Bu yerda **nom** – makros nomi.

Makroslar **#undef** komandasi orqali rad etiladi.

Funksional turdagi makroslarni qo‘shimcha yuklash mumkin emas.

Misol:

```
#include<iostream>
using namespace std;
#define max(a,b) (a<b ? b:a)
int main(){
int m,k=1,n=5;
float c,a=1.5,b=1.2;
m=max(6*k,n);
cout<<m<<endl;
c=max(a,b);
cout<<c;
```

```
cin.get();
return 0;
}
```
Natija:
6
1.5

Makroslarning funksiyadan farqi. Funksiya dasturda bitta nusxada bo‘lsa, makros hosil qiluvchi matnlar makros har gal chaqirilganda dasturga joylashtiriladi. Funksiya parametrlar spetsifikatsiyasida ko‘rsatilgan turlar uchun ishlatiladi va aniq turdagi qiymat qaytaradi. Makros har qanday turdagi parametrlar bilan ishlaydi. Hosil qilinayotgan qiymat turi faqat parametrlar turlari va ifodalarga bog‘liq.

Makrojoylashlardan to‘g‘ri foydalanish uchun almashtiriluvchi satrni qavsga olish foydalidir.

Funksiyaning haqiqiy parametrlari bu ifodalardir, makros argumentlari bo‘lsa, vergul bilan ajratilgan leksemalardir. Argumentlarga makro-kengaytirishlar qo‘llanmaydi.

Shuni ko‘rsatish lozim-ki, C++ tilida makroslar juda kam ishlatiladi.

Parametrlar soni o‘zgaruvchi bo‘lgan funksiyalar.

Ixtiyoriy sonli parametrlar. C++ tilida parametrlar soni belgilanmagan funkiyalar yaratish mumkindir. Parametrlar soni va turi funksiyaga murojaat qilinganda, haqiqiy parametrlar soni va turi asosida aniqlanadi. Parametrlar soni o‘zgaruvchan bo‘lgan funksiya juda bo‘lmasa bitta parametrga ega bo‘lishi kerak.

Parametrlar soni o‘zgaruchan funksiya ta'rifi:

<tur><ism>(<oshkor_parametrlar>, . . .)

Verguldan so‘ng, uch nuqta keyingi parametrlar soni va turi ixtiyoriy bo‘lishi mumkinligini ko‘rsatadi. Ro‘yxat boshi va oxirini belgilash uchun ikki usul mavjud:

1) ro‘yxat tugashi belgisi ma'lum;

2) uzatilayotgan parametrlar soni ma'lum.

Misol:

```
#include <iostream>
using namespace std;
int sum(int n, ...){
int total = 0;
int arg;
int*pa=&n;
for(int i=0;i<n;i++){
pa++;
arg=*pa;
total += arg;
};
return total;
};
int main(){
int k=9;
int m=sum(4,1,k,k,1);
cout<<m;
return 0;
}
```

Natija:
20

Ixtiyoriy sonli parametrlar uchun makrovositalar. Parametrlar soni o‘zgaruvchi funksiyalar yaratish qulay usuli **stdarg.h** faylida saqlanuvchi makrokota'riflardan foydalanishdir. Bu makrota'riflar ixtiyoriy sonli parametrlarga ega bo‘lgan funksiyalar yaratishning qulay va sodda vositalari bo‘lib quyidagi ko‘rinishga egadirlar:

void va_start (va_list param, oxirgi_aniq_parametr)-ro‘yxatning birinchi elementi bilan boglanish ;

<tur> va_arg (va_list param, <tur>) – ro‘yxatning keyingi elementini ukish.

void va_end(va_list param) – ro‘yxat tugagandan so‘ng chaqiriladigan komanda.

Bu makroslarda birinchi parametrlar **stdarg.h** faylida aniqlangan maxsus **va_list** turiga tegishli bo‘lishi lozimdir. Bu tur orqali ko‘rsatkich xossasiga ega bo‘lgan obyekt e'lon qilinadi. Funksiya tanasida albatta **va_list** turidagi obyekt ta'riflangan bo‘lishi lozim. Bu obyekt **va_start()** funksiyasi yordamida noma'lum

uzunlikdagi ro‘yxatning birinchi elementi bilan bog‘lanadi. Buning uchun makrosning ikkinchi parametri sifatida oxirgi ta'riflangan parametr ishlatiladi.

Noma'lum ro‘yxat birinchi parametr turini funksiyaga uzatish va ko‘rsatkichni shu parametrga to‘g‘rilash uchun **va_arg** makrosiga murojaatdan foydalanish lozim.

Keyingi parametr turini funksiyaga uzatish va ko‘rsatkichni shu parametrga to‘g‘rilash uchun yana **va_arg()** makrosiga murojaat qilinadi.

Ixtiyoriy sonli parametrli funksiyadan to‘g‘ri qaytish uchun **va_end** makrosidan foydalaniladi.

Bu makros parametrlar ro‘yxati tugagandan chaqiriladi. Shuni ko‘rsatish lozimki **va_start()** makrosi chaqirilmasdan oldin **va_end()** makrosi qayta chaqirilishi mumkin emas.

Misol:

```
#include <iostream>
#include <cstdarg>
using namespace std;
void miniprint(char *format,...){
va_list ap;
char *p;
int ii;
double dd;
va_start(ap,format);
for (p=format;*p;p++){
if (*p!='%'){
cout<<*p;
continue;
}
switch(*++p){
case 'd':ii=va_arg(ap,int);
cout<<ii;
break;
case 'f':dd=va_arg(ap,double);
cout<<dd;
break;
default:cout<<*p;
}
}
va_end(ap);
```

```
}
int main(){
int k=154;
double e=2.718282;
miniprint("\n k=%d,\t e=%f",k,e);
return 0;
}
```

Nazorat savollari:

1. Kompilyator va preprotsessorning farqi nimadan iborat?
2. Shartli direktivalar nima uchun ishlatiladi?
3. Xotira sinflarini ko‘rsating.
4. Tashqi obyektlar qanday e'lon qilinadi?
5. Ta'rif bilan e'lon orasida qanday farq bor?
6. Dinamik xotira bilan ishlash qanday variantlari mavjud?
7. Dinamik o‘zgaruvchilar oddiy o‘zgaruvchilardan qanday farq qiladi?
8. Bosh funksiya parametrlari.
9. Nima uchun parametrlari soni o‘zgaruvchan funksiya juda bo‘lmasa bitta parametrga ega bo‘lishi kerak
10. Parametrlar soni o‘zgaruvchi funksiyalar yaratish uchun qanday makrota'riflardan foydalanish qulay?

Misollar:

1. Ikkita funksiya yaratib ikki faylga yozing. Ikkala faylni dasturga ulab funksiyalarni ishlating.
2. Ikki son minimumini hisoblovchi makros yarating va dasturda foydalaning.
3. Berilgan ketma-ketlik qat'iy kamayuvchi ekanligini tekshiruvchi parametrlar soni o‘zgaruvchan funksiya tuzing va dasturda foydalaning.
4. Parametrlari soni o‘zgaruvchan funksiya yordamida Matritsani vektorga ko‘paytirig, funksiyasini yaratib dasturda foydalaning. Matritsa ko‘rsatkichlar massivi sifatida kiritilsin.
5. Hamma xotira sinflariga tegishli o‘zgaruvchilar qatnashgan dastur tuzing.

STANDART FUNKSIYALAR

Matematik funksiyalar

Quyida ko‘riladigan matematik funksiyalardan foydalanish uchun dasturga **<math.h>** sarlavhali faylni ulash lozim.

Trigonometrik funksiyalar: cos(), sin(), tan().

double cos(double x);

double sin(double x);

double tan(double x);

Argumentlar qiymatlari radianda beriladi.

Sinus va kosinus -1 dan 1 gacha qiymat qaytaradi.

Tangens **sin(x)/cos(x)** formula bo‘yicha hisoblanadi.

Misol:

```
#include <iostream>
#include <math>
using namespace std;
int main(){
cout<<sin(M_PI_2)<<" "<<cos(M_PI_2)<<endl;
cout<<tan(M_PI_4)<<endl;
return 0;
}
```

Bu misolda **M_PI_2** va **M_PI_4** pi yarmi va to‘rtdan bir qismini bildiruvchi konstantalardir.

Teskari trigonometrik funksiyalar. Teskari trigonometrik funksiyalar **acos**, **asin**, **atan**, ikki son bo‘linmasi arktangensi – **atan2** funksiyalari yordamida hisoblanadi:

double acos(double x);

double asin(double x);

double atan(double x);

double atan2(double y, double x);

Arkkosinus va arksinus argumentlari qiymati -1 va 1 oralig‘ida yotishi kerak.

Funksiyalar radianda qiymat qaytaradi.

Arkkosinus qiymati 0 va **pi** oralig‘ida.

Arksinus qiymati **-pi/2** va **pi/2** oralig‘ida.

Arktangens qiymati **-pi/2** va **pi/2** oralig‘ida

Bo‘linma arktangensi qiymati **-pi** va **pi** oralig‘ida.

Misol:

```
#include <iostream>
#include <math>
using namespace std;
int main(){
cout<<asin(1)<<" "<<acos(0)<<endl;
cout<<atan(1)<<" "<<atan2(2,2)<<endl;
return 0;
}
```

Absolyut qiymat. Haqiqiy son absolyut qiymati **fabs** funksiyasi yordamida hisoblanadi.

```
double fabs(double x);
```

Misol:

```
#include <iostream>
#include <math>
using namespace std;
int main(){
double x=-1.0;
cout<<abs(x)<<endl;
return 0;
}
```

Eksponenta. Eksponenta hisoblash uchun **exp** funksiyasidan foydalaniladi.

```
double exp(double x);
```

Logarifm. Natural logarifmni hisoblash uchun **log**, o‘nlik logarifmni hisoblash uchun **log10** funksiyalaridan foydalaniladi:

```
double log(double x);
double log10(double x);
```

Misol:

```
#include <iostream>
#include <math>
using namespace std;
```

```
int main(){
double x=1.0;
cout<<exp(x)<<endl;
x=2.718282;
cout<<log(x)<<endl;
x=10.0;
cout<<log10(x);
return 0;
}
```

Daraja. Darajani hisoblash uchun **pow** funksiyasidan foydalanish lozim:

double pow(double x, double y);

Funksiya x^y qiymatni qaytaradi.

Agar x va y ikkalasi 0 bo‘lsa, natijasi 1.

Agar $x < 0$ va y musbat bo‘lsa, xatolik yuzaga keladi.

Ildiz. Kvadrat ildiz olish uchun **sqrt** funksiyasidan foydalaniladi:

double sqrt(double x);

Agar parametr qiymati musbat bo‘lsa, natija ham musbat bo`ladi, agar parametr qiymati manfiy bo‘lsa, xatolik kelib chiqadi.

Misol:

```
#include <iostream>
#include <math>
using namespace std;
int main(){
double x = 4.0;
double y=sqrt(x);
double z=pow(x,0.5);
cout<<y<<" "<<z<<endl;
return 0;
}
```

Yaxlitlash funksiyalari. Sonni yuqoriga yaxlitlash uchun **ceil** va pastga yaxlitlash uchun **floor** funksiyasidan foydalaniladi.

Prototiplari:

double ceil(double x);

double floor(double x);

Birinchi funksiya x dan kichik bo‘lmagan eng kichik butun sonni qaytaradi.

Ikkinchi funksiya x dan katta bo‘lmagan eng katta butun sonni qaytaradi.

Ikkala funksiya butun sonni **double** turda qaytaradi.

Misol (**ceil** va **floor** uchun):

```
#include <iostream>
#include <math>
using namespace std;
int main(){
double number = 123.54;
double down, up;
down = floor(number);
up = ceil(number);
cout<<down<<" "<<up;
return 0;
}
```

Modul. Modul ya'ni ikki son bo‘linmasi x/y qoldig‘ini hisoblash uchun fmod funksiyasidan foydalaniladi.

double fmod(double x, double y);

Funksiya qiymati f quyidagi shartlarga mos keladi:

x = (ay + f) bunda a butun son va 0 < f < y.

Agar y = 0,bo‘lsa fmod qiymati 0 ga teng bo‘ladi.

Kasr ajratish. Haqiqiy sonni butun va kasr qismini ajratish uchun modf funksiyasidan foydalaniladi.

double modf(double x, double *ipart);

Funksiya double x sonni qismini ipart parametrga yozib, kasr qismini qiymat sifatida qaytaradi.

Misol:

```
#include<iostream>
#include<math>
using namespace std;
int main() {
cout<<fmod(4.5,2)<<endl;
double x,y;
x=modf(4.5,&y);
cout<<x<<" "<<y<<endl;
system("pause");
return 0;
```

}

Simvolli funksiyalar

Simvolni tekshirish. Simvolni tekshirish uchun quyidagi makroslardan foydalaniladi: **isalnum**, **isalpha**, **iscntrl**, **isdigit**, **isgraph**, **islower**, **isprint**, **ispunct**, **isspace**, **isupper**, **isxdigit**.

Bu makroslardan foydalanish uchun dasturga **<ctype.h>** sarlavhali faylni ulash lozim.

Makroslar argumentlari simvol ASCII kodini bildiruvchi butun son.

Makroslar qiymatlari nolga teng bo‘lmagan son agar shart bajarilsa va 0 aks holda.

Har bir makrosni **#undef** direktivasi yordamida rad etish mumkin.

int isalnum(int c); c harf (A dan Z gacha yoki a dan z gacha) yoki c raqam (0 dan 9 gacha)

int isalpha(int c); c harf (A dan Z gacha yoki a dan z gacha)

int iscntrl(int c); c delete simvoli yoki control simvol (0x7F yoki 0x00 yoki 0x1F)

int isdigit(int c); c raqam (0 dan 9 gacha)

isgraph(int c); c bosiluvchi simvol isprint dan farqli, bo‘shlik simvoli ham kiradi.

int islower(int c); c kichik harf (a dan z gacha)

int isprint(int c); c bosiluvchi simvol (0x20 dan 0x7E gacha)

int ispunct(int c); c punktuatsiya belgisi (iscntrl yoki isspace)

int isspace(int c); c yoki bo‘shlik, tab, karetkani o‘tkazish return, yangi satr, vertikal tab, yoki formfeed (0x09 to 0x0D, 0x20)

int isupper(int c); c katta harf (A to Z)

int isxdigit(int c); c o‘n oltilik raqam (0 dan 9 gacha, A dan F gacha, a dan f gacha)

Simvollarni o‘zgartirish. Katta harflarni kichik harflarga aylantirish uchun **tolower** funksiyasidan foydalaniladi.

int tolower(int ch);

Bu funksiya o`zgaruvchinning qiymati katta harf kodiga teng bo‘lsa, mos kichik harf kodini qaytaradi (A dan Z gacha; natija a dan z gacha). Qolgan hollarda simvol o‘zgarmaydi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int i;
char str1[20]= "THIS IS A STRING";
for (i=0; str1[i]!='\0'; i++){
str1[i] = tolower(str1[i]);
}
cout<<str1;
return 0;
}
```

Kichik harflarni katta harflarga aylantirish uchun **toupper** funksiyasidan foydalaniladi.

int toupper(int ch);

Bu funksiya o`zgaruvchining qiymati kichik harf kodiga teng bo‘lsa, mos katta harf kodini qaytaradi (a dan z gacha; natija A dano Z gacha). Qolgan hollarda simvol o‘zgarmaydi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
int i;
char str1[20]= "this is a string";
for (i=0; str1[i]!='\0'; i++){
str1[i] = toupper(str1[i]);
}
cout<<str1;
return 0;
}
```

Satrli funksiyalar

Satrli funksiyalardan foydalanish uchun dasturga **<string.h>** sarlavhali fayl ulash lozim.

Satrda simvollar sonini hisoblash uchun **strlen** funksiyasidan foydalaniladi.

size_t strlen(const char *s);

Satrdagi simvollar sonini qaytaradi. Satr oxirini bildiruvchi **null** simvol hisobga kirmaydi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
cout<<strlen("Borland International");
return 0;
}
```

Satrdan nusxa olish uchun **strcpy** funksiyasidan foydalaniladi.

char *strcpy(char *dest, const char *src);

Bu funksiya satr simvollarini **dest** satrga nusxa oladi va **dest** satr qaytaradi.

Satrdan berilgan sondagi simvollardan nusxa olish uchun **strncpy** funksiyasidan foydalaniladi.

char *strncpy(char *dest, const char *src, size_t maxlen);

Bu funksiya satr **maxlen** dan oshmagan simvollarini **dest** satrga nusxa oladi va **dest** satr qaytaradi.

Qaytarilayotgan **dest** satri satr oxirini bildiruvchi **null**-simvolga ega bo‘lmasligi mumkin, agar **src** uzunligi **maxlen** ga teng yoki katta bo‘lsa.

Misol:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
char dest1[25],dest2[25];
strcpy(dest1, "United");
strncpy(dest2, "United", 2);
cout<<dest1<<endl;
cout<<dest2;
return 0;
}
```

Satrlarni ulash uchun **strcat** funksiyasidan foydalaniladi.

char *strcat(char *dest, const char *src);

Birinchi **dest** satr oxiriga **src** satr simvollarini ulaydi.

Natija uzunligi **strlen(dest) + strlen(src)** bo`ladi.

Bu funksiya satrlar konkatenatsiyasiga ko‘rsatkich qaytaradi.

Berilgan sondagi simvollarni ulash uchun **strncat** funksiyasidan foydalanish lozim.

char *strncat(char *dest, const char *src, size_t maxlen);

Berilgan **dest** satr oxiriga **src** satr **maxlen** dan oshmagan simvollarini ulaydi va satr oxiriga **null** simvol qo‘shadi.

Natija uzunligi **strlen(dest) + maxlen** bo`ladi.

Bu funksiya **dest** satr qaytaradi.

Misol:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
char dest1[25];
char source1[] = "States";
char source2[] = "St";
strcat(source1, " United");
strncat(source2, " United",3);
cout<<source1<<endl;
cout<<source2;
return 0;
}
```

Satrlarni solishtirish uchun **strcmp** funksiyasidan foydalaniladi.

int strcmp(const char *s1, const char*s2);

Bu funksiya **s1** va **s2** satrlarni leksikografik solishtiradi.

Qaytaradigan qiymati:

natija < 0, agar s1 < s2;

natija == 0, agar s1 == s2;

natija > 0, agar s1 > s2.

Satrlarni berilgan sondagi simvollarni solishtirish uchun **strncmp** funksiyasidan foydalaniladi.

Prototipi:

int strncmp (const char *s1, const char *s2, size_t maxlen);

Funksiya **maxlen** dan ko‘p bo‘lmagan **s1**va s2 satr simvollarini leksikografik solishtiradi.

Qaytaradigan qiymati:

natija < 0, agar s1 < s2;

natija == 0, agar s1 == s2;

natija > 0 , agar s1 > s2.

Misol:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
char *buf1 = "aaa", *buf2 = "aab";
cout<<strcmp(buf2, buf1)<<endl;
cout<<strncmp(buf2, buf1,2);
return 0;
}
```

Satrda simvolni birinchi kirishini qidirish uchun **strchr** funksiyasidan foydalaniladi:

char *strchr(char *s, int c);

Bu funksiya **s** satrda **c** simvol birinchi kirishini qidiradi.

Satr oxirini bildiruvchi **null**-simvolni quyidagicha qidirish mumkin: **strchr(strs, 0).**

Funksiya simvol birinchi kirishiga ko‘rsatkich qaytaradi. Xatolik yuz bersa **null** qaytaradi.

Satrda simvolni oxirgi kirishini qidirish uchun **strrchr** funksiyasidan foydalaniladi:

char *strrchr(char *s1, int c);

Bu funksiya **s1** satrda **c** simvol oxirgi kirishini qidiradi.

Satr oxirini bildiruvchi **null**-simvolni quyidagicha qidirish mumkin: **strrchr(strs, 0).**

Funksiya simvol oxirgi kirishiga ko‘rsatkich qaytaradi. Xatolik yuz bersa **null** qaytaradi.

Satrda satr bo`lagini izlash uchun strstr funksiyasidan foydalaniladi:

char *strstr(const char *s1, const char *s2);

Bu funksiya **s1** satrda **s2** satr bo`lagi birinchi kirishini izlaydi.

Agar satr bo`lagi mavjud bo‘lsa, birinchi kirishiga ko‘rsatkich, aks holda **null** qaytaradi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
char *str1 = "Borland International", *str2 = "nation", *ptr;
ptr = strstr(str1, str2);
if (ptr==NULL) cout<<"NO";
else cout<<"YES";
return 0;
}
```

Birinchi satrga birinchi kiruvchi ikkinchi satrdagi simvolni topish uchun **strpbrk** funksiyasidan foydalaniladi:

char *strpbrk(const char *s1, const char *s2);

Bu funksiya **s1** satrga birinchi kiruvchi **s2** satrdagi simvolni qidiradi. Agar simvol mavjud bo‘lsa, shu simvolga ko‘rsatkich qaytaradi, aks holda **null** qaytaradi.

Ikki satr orasidagi farqni aniqlash uchun **strtok** funksiyasidan foydalaniladi:

char *strtok(char *s1, const char *s2);

Bu funksiya **s1** satrda **s2** satrdan farqli bir yoki bir necha simvollarni aniqlaydi.

Birinchi chaqirishda **s1** satrdagi farqli birinchi satr bo`lagiga ko‘rsatkich qaytaradi. Bunday satr bo`lagi mavjud bo‘lmasa, **null** qaytaradi. Agar qayta chaqirishda birinchi parametri birinchi **null** bo‘lsa, ikkinchi kirishidan oldingi farq qiluvchi satr bo`lagiga ko‘rsatkich qaytaradi.

Bir satr boshida ikkinchi satrdan farqli qismini izlash uchun **strcspn** funksiyasidan foydalaniladi.

Prototipi:

size_t strcspn(const char *s1, const char *s2);

Birinchi **strcspn** funksiyasi **s1** satr boshidan shunday segment izlaydi-ki, biror simvoli **s2** satrga kirmaydi.

Bu funksiya boshlang‘ich satr uzunligini qaytaradi.

Bir satr boshida ikkinchi satrni izlash uchun **strspn** funksiyasidan foydalaniladi:

size_t strspn(const char *s1, const char *s2);

strspn funksiyasi **s1** satr boshidan shunday segment izlaydi-ki, hamma simvollari **s2** satrga kirsin.

Bu funksiya boshlang‘ich satr uzunligini qaytaradi.

Misol:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
char input[16] = "adbcetbc";
char *p;
p = strpbrk(input, "bc");
if (p) cout<<p<<endl;
p = strtok(input, "bc");
if (p)   cout<<p;
return 0;
}
```

O‘zgartirish funksiyalari

O‘zgartirish funksiyalari. Satr shaklida berilgan butun sonni songa aylantirish uchun **atoi** makrosidan foydalaniladi. Prototipi:

int atoi(const char *s);

Satr quyidagi shaklda berilgan bo‘lishi kerak:

[ws] [sn] [ddd]

Bu yerda:

[ws] - tabulyasiya yoki bo‘shlik belgisi, **[sn]** - ishora belgisi, **[ddd]** – raqamlar satri.

Birinchi aylantirib bo‘lmaydigan simvolgacha bajariladi.

Satr shaklida berilgan uzun sonni **long** turidagi son sifatida qaytarish uchun **atol** funksiyasidan foydalaniladi:

long atol(const char *s);

Satr quyidagi shaklda bo‘lishi kerak:

[ws] [sn] [ddd]

Birinchi aylantirib bo‘lmaydigan simvolgacha bajariladi. Aylantirish mumkin bo‘lmasa 0 qaytariladi.

Satr shaklida berilgan haqiqiy sonni son sifatida qaytaruvchi funksiya **atof** deb nomlanadi:

double atof(const char *s);

Satr quyidagi shaklda berilgan bo‘lishi kerak:

[whitespace] [sign] [ddd] [.] [ddd] [e|E[sign]ddd]

Satrni ikkilangan haqiqiy songa aylantirish uchun **strtod** funksiyasidan foydalaniladi:

double strtod(const char *s, char **endptr);

Satr quyidagi ko‘rinishda bo‘lishi lozim:

strtod: [ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]

Bu yerda:

[ws] – bo‘shlik, **[sn]** – ishora (+ yoki -), **[ddd]** – raqamlar, **[fmt]** – e yoki E, **[.]** – nuqta, **[0]** – nol (0), **[x]** – x yoki X.

Bundan tashqari **+INF** va **-INF** plyus yoki minus cheksizlik qaytaradi hamda **+NAN** va **-NAN** (Not-A-Number) son bo`lmagan qiymat uchun qaytaradi.

Funksiya to aylantirib bo‘lmaydigan birinchi simvolgacha ishlaydi.

Misol:

```
#include <iostream>
using namespace std;
int main(){
char input[80], *endptr;
```

```
double value;
cout<<"Enter a floating point number:";
cin>>input;
value = strtod(input, &endptr);
cout<<value;
return 0;
}
```

Tasodifiy sonlar

Tasodifiy sonlar. Tasodifiy sonlar generatori **rand** deb ataladi.

int rand(void);

Funksiyadan foydalanish uchun **<stdlib.h>** sarlavhali faylni ulash lozim. Funksiya **rand** davri 232 bo‘lgan tasodifiy sonlar generatoridan foydalangan holda 0 dan to **rand_max** oraliqdagi tasodifiy son qaytaradi.

Misol: 0 va 99 oraliqdagi tasodifiy sonlar generatsiyasini hosil qilish:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]){
for(int i=0; i<10; i++)
cout<<rand()%100<<" "; cout<<endl;
system("pause");
}
```

Tasodifiy sonlar generatorini initsializatsiya qilish uchun **srand** funksiyasidan foydalaniladi:

void srand(unsigned seed);

Tasodifiy sonlar generatori **srand** funksiyasi 1 qiymat bilan chaqirilganda qayta initsializatsiya qilinadi. Bu funksiya qiymat qaytarmaydi.

Misol:

```
#include<iostream>
using namespace std;
int main(int argc, char* argv[]){
int i;
for(i=0; i<10; i++) cout<<rand()%100<<" ";
cout<<"\n";
srand(1);
for(i=0; i<10; i++) cout<<rand()%100<<" ";
cout<<"\n";
```

```
system("pause");
}
```

Vaqt. Dasturda joriy vaqtni ishlatish uchun dasturga **<time.h>** sarlavhali faylni ulash lozim.

Funksiyalarda **time_t** va **clock_t** turlardan foydalaniladi.

Konstanta **clk_tck** bir soniyada soat millari qancha yurganini, ya'ni tiklar sonini aniqlaydi.

Asosiy funksiyalar:

clock_t clock(void) dastur ishga tushgan vaqtdan boshlab, protsessor vaqti intervalini qaytaradi. Xato bo'lsa -1 qaytaradi.

time_t time(time_t *timer) joriy vaqtni soniyalarda 00:00:00 GMT, January 1, 1970 dan boshlab oladi. Funksiya vaqtni soniyalarda qaytaradi. Xatolik yuz bersa, **null** qaytaradi.

double difftime(time_t time2, time_t time1) farqni soniyalarda hisoblaydi.

char *asctime(const struct tm *tblock) sana va vaqtni ASCII ga o'tkazadi.

char *ctime(const time_t *time) sana va vaqtni satrga aylantiradi.

Satr 26 simvoldan iborat:

DDD MMM dd hh:mm:ss YYYY

Misol: **Mon Oct 20 11:31:54 1952\n\0**

struct tm *gmtime(const time_t *timer) vaqt va sanani GMT ga aylantiradi.

struct tm *localtime(const time_t *timer) vaqt va sanani strukturaga aylantiradi.

time_t mktime(struct tm *t) kalendar formatga aylantiradi.

size_t _cdecl strftime(char *s, size_t maxsize, const char *fmt, const struct tm *t) format spetsifikatorlari asosida vaqtni massivda shakllantiradi. Simvollar sonini qaytaradi. Agar xato bo'lsa. 0 qaytaradi. Formatlar **strftime** funksiyasi uchun razryadli amallar yordamida quyidagi formatlar asosida shakllantiriladi:

%a	Hafta qisqa nomi	Mon
%A	Hafta to'liq nomi	Monday

%b	Oy qisqa nomi	Jul
%B	Oy to‘liq nomi	July
%c	Sana va vaqt	Jul 12 21:53:22 1998
%d	Oy kuni 2 raqamli	12
%H	Soat (24 shkalada) 2 raqamli	21
%I	Soat (12 shkalada) s 2 raqamli	12
%j	Yil kuni 3 raqamli	193
%m	Oy tartibi 2 raqamli	7
%M	Minutlar 2 raqamli	53
%p	Sutka yarmi AM yoki PM	PM
%S	Soniyalar 2 raqamli (00--59)	22
%U	Hafta nomeri yakshanbadan	28
%W	Hafta nomeri dushanbadan	28
%w	Hafta kuni (yakshanba=0)	0
%x	Sana	Jul 12 1998
%X	Vaqt	21:53:22
%y	Qisqa yil 2 raqamli	98
%Y	To`liq yil	1998
%Z	Soat poyasi	MEST

tm strukturasi vaqt oralig‘ini tasvirlaydi:

```
struct tm {
    int tm_sec;     /* Soniyalar */
    int tm_min;     /* Minutlar */
    int tm_hour;    /* Soat (0--23) */
    int tm_mday;    /* Oy kuni (1--31) */
    int tm_mon;     /* Oy (0--11) */
    int tm_year;    /* Yil (Kalendar minus 1900) */
    int tm_wday;    /* Hafta kuni (0--6; YAkshanba = 0) */
    int tm_yday;    /* Yil kuni (0--365) */
    int tm_isdst;   /* 0 agar xato */
```

};

Misol:

```
#include<iostream>
#include<ctime>
using namespace std;
int main(){
int i;
time_t t1,t2;
t1 = time(NULL);
cout<<"\n"<<t1;
for(i=0;i<1000000000;i++);
t2 = time(NULL);
cout<<"\n"<<t2;
cout<<"\n"<<t2-t1<<endl;
system("pause");
}
```

Lokallashtirish. Lokallashtirish, ya'ni mahalliy tilga o‘tish uchun uchun C tilida **setlocale** va **localeconv** funksiyasidan foydalaniladi. Bu funksiyalardan foydalanish uchun **locale** sarlavhali faylni ulash lozim.

setlocale funksiyasi prototipi:

char *setlocale(int category, char *locale);

Birinchi parametr quyidagi qiymatlarga ega bo‘lishi mumkin: **lc_all**, **lc_collate**, **lc_ctype**, **lc_monetary**, **lc_numeric**, **lc_time.**

localeconv funksiyasi prototipi:

struct lconv *localeconv(void);

Misol:

```
#include <iostream>
using namespace std;
int main(){
setlocale(LC_ALL,"RUSSIAN");
cout<<"PRIVET"<<endl;
system("pause");
return 0;
}
```

Keyingi misolda mahalliy til xususiyatlarini ekranga chiqarish ko‘rsatilgan. Misolda C tilida ishlatiluvchi chiqarish printf funksiyasidan foydalanilgan:

```
#include <iostream>
using namespace std;
void get(char* r){
char *old_locale;
/* The only locale supported in Turbo C++ is "C" */
old_locale = setlocale(LC_ALL,r);
struct lconv ll;
struct lconv *conv = &ll;
/* read the locality conversion structure */
conv = localeconv();
/* display the structure */
printf("Decimal Point              : %s\n", conv->decimal_point);
printf("Thousands Separator        : %s\n", conv->thousands_sep);
printf("Grouping                   : %s\n", conv->grouping);
printf("International Currency symbol  : %s\n",
conv->int_curr_symbol);
printf("$ thousands separator      : %s\n", conv->mon_thousands_sep);
printf("$ grouping                 : %s\n", conv->mon_grouping);
printf("Positive sign              : %s\n", conv->positive_sign);
printf("Negative sign              : %s\n", conv->negative_sign);
printf("International fraction digits  : %d\n", conv->int_frac_digits);
printf("Fraction digits            : %d\n", conv->frac_digits);
printf("Positive $ symbol precedes     : %d\n", conv->p_cs_precedes);
printf("Positive sign space separation : %d\n", conv->p_sep_by_space);
printf("Negative $ symbol precedes     : %d\n", conv->n_cs_precedes);
printf("Negative sign space separation : %d\n", conv->n_sep_by_space);
printf("Positive sign position     : %d\n", conv->p_sign_posn);
printf("Negative sign position     : %d\n", conv->n_sign_posn);
}
int main() {
get("RUSSIAN");
system("pause");
return 0;
}
```

Nazorat savollari:

1. Trigonometrik funksiyalarni ko‘rsating.
2. Teskari trigonometrik funksiyalarni ko‘rsating.
3. Qanday yaxlitlash funksiyalari mavjud?
4. Ixtiyoriy asosdagi logarifm qanday hisoblanadi?

5. Ixtiyoriy ildiz qanday hisoblanadi?
6. Ikki satr orasidagi farqni aniqlash uchun qaysi funksiyalardan foydalaniladi?
7. Satrdan nusxa olish uchun qaysi funksiyalardan foydalaniladi?
8. Satrlarni solishtirish uchun qanday funksiyalardan foydalaniladi?
9. Simvolli funksiyalardan foydalanish uchun qaysi sarlavhali faylni ulash lozim?
10. Satrni songa aylantirish funksiyalarini ko‘rsating.

Misollar:

1. Trigonometrik funksiyalarga dastur yarating.
2. Daraja, eksponenta va logarifm funksiyalariga dastur yarating.
3. Simvolli tekshiruvchi funksiyalarga dastur yarating.
4. Simvolli almashtiruvchi funksiyalarga dastur yarating.
5. Satrni songa aylantiruvchi funksiyalarga dastur yarating.

III QISM. SINFLAR VA OBYEKTLAR

Sinf

Sintaksis bo‘yicha, C++ da sinf – bu mavjud bo‘lgan turlar asosida yangi yaratilgan strukturlangan tur.

Sinfning umumiy sodda shakli quyidagicha:

<sinf_turi> <sinf_nomi>{<sinf_komponentlari_ro‘yxati>};

Bu yerda:

<sinf_turi> – **class, struct, union** xizmatchi so‘zlaridan biri;

<sinf_nomi> – identifikator;

<sinf_komponentlari_ro‘yxati> – sinfga tegishli ma'lumotlar va funksiyalar ta'rifi.

Sinf komponentalari sodda holda turlangan ma'lumotlar va funksiyalardan iborat bo‘ladi. Figurali qavslarga olingan komponentalar ro‘yxati sinf tanasi deb ataladi.

Funksiya – bu obyektlar ustida bajariladigan amallarni aniqlovchi sinf usuli.

Ma'lumotlar – bu obyekt strukturasini hosil qiluvchi maydon.

Usullar sinfdan tashqarida aniqlanganda, ularning nomlarini xususiylashtirish (kvalifikatsiyalash) kerak. Usulning ko‘rimlilik sohasini aniqlaydigan uning bunday kvalifikatsiya sintaksisi quyidagi ko‘rinishga ega:

<sinf nomi>::<usul nomi>

Sinf ichida aniqlangan usullar jimlik bo‘yicha joylashtiriluvchi (**inline**) funksiya hisoblanadi. Sinf tashqarisida aniqlangan usullarni oshkor ravishda joylashtiriluvchi deb ta'riflanishi lozim.

Sinf obyekti (sinf nusxasi) ni ta'riflash uchun quyidagi konstruksiyadan foydalaniladi:

<sinf_nomi> <obyekt_nomi>;

Obyekt orqali maydonlarga va usullarga quyidagicha murojaat qilish mumkin:

< obyekt_nomi >. <maydon_nomi>

< obyekt_nomi >. <usul_nomi>

Dasturda obyekt komponentasiga quyidagicha murojaat qilish mumkin:

Sinf-nomi.obyekt-nomi :: komponenta-nomi

Komponenta o‘zgaruvchilar va komponenta funksiyalar. Sinf komponenta o‘zgaruvchilari sifatida o‘zgaruvchilar, massivlar, ko‘rsatkichlar ishlatilishi mumkin. Elementlar ta'riflanganda initsializatsiya qilish mumkin emas. Buning sababi shu-ki, sinf uchun xotiradan joy ajratilmaydi. Komponenta elementlariga komponenta funksiyalar orqali murojaat qilinganda, faqat nomlari ishlatiladi. Sinfdan tashqarida sinf elementlariga emas, obyekt elementlariga murojaat qilish mumkin. Bu murojaat ikki xil bo‘lishi mumkin:

1. **Obyekt_nomi.Element_nomi.** Sinf elementlari sinfga tegishli funksiyalarda ishlatilishidan oldin ta'riflangan bo‘lishi shart emas. Xuddi shunday bir funksiyadan hali ta'rifi berilmagan ikkinchi funksiyaga murojaat qilish mumkin.

2. **Komponenta funksiya ta'rifi.** Komponenta funksiya albatta sinf tanasida ta'riflangan bo‘lishi lozim. Global funksiyalardan farqli komponenta funksiya sinfning hamma komponentalariga murojaat qilishi mumkin. Funksiyaning faqat prototipi emas, to‘la ta'rifi sinf tanasida joylashgan bo‘lsa, bu funksiya joylashtiruvchi (**inline**) funksiya hisoblanadi. Ma'lum-ki, **inline** funksiyalarda sikllar, kalit bo‘yicha o‘tish operatori ishlatilishi mumkin emas. Bundan tashqari bunday funksiyalar rekursiv funksiya bo‘lolmaydi. Bu chegaralarni yengish uchun sinf tanasiga faqat funksiya prototipi joylashtirilib, funksiyaning to‘la ta'rifi sinf tashqarisida dasturga kiruvchi boshqa funksiyalar bilan birga beriladi. Komponenta funksiyani sinf tashqarisida ta'riflanganda, qaysi sinfga tegishli ekanligini quyidagi shaklda ko‘rsatiladi:

Sinf_nomi :: Komponenta_funksiya_nomi

Sinf tanasiga komponenta funksiya prototipi quyidagi shaklda joylashtiriladi:

Tur funksiya_nomi(formal_parametrlar_ta'rifi)

Sinf tashqarisida funksiya quyidagi shaklda ta'riflanadi:

Tur sinf_nomi::funksiya_nomi(formal_parametrlar_ta’rifi){funksiya_tanasi}

Komponentalarga murojaat huquqlari

Komponentalarga murojaat huquqi murojaat spetsifikatorlari yordamida boshqariladi: **public, private, protected**.

Umumiy (**public**) komponentalar dasturni ixtiyoriy qismida murojaat huquqiga ega. Ulardan, ixtiyoriy funksiya ushbu sinf ichida va sinf tashqarisida foydalansa ham bo‘ladi.

Xususiy (**private**) komponentalar sinf ichida murojaat huquqiga ega, lekin sinf tashqarisidan esa murojaat qilish mumkin emas. Komponentalardan ushbu ular tavsiflangan sinfdagi funksiya-a'zolari yoki "do‘stona"-funksiyalar orqali foydalanish mumkin.

Himoyalangan (**protected**) komponentalar sinf ichida va hosila sinflarda murojaat huquqiga ega.

Agar sinf ta'rifida **class** so‘zi ishlatilgan bo‘lsa, hamma komponentalari xususiy hisoblanadi, agar **struct** so‘zi ishlatilgan bo‘lsa, hamma komponentalar umumiy hisoblanadi.

Misol:

```
#include<iostream>
using namespace std;
struct complex{
public:
void show (){ cout << "re=" << re<<" im=" << im<<endl;};
void ini (double re1, double im1){re = re1; im = im1;}
private:
double re,im;
};
int main() {
complex ss;
ss.ini (5.9,0.15);
ss.show();
system("pause");
return 0;
}
```

Natija:

re=5.9 im=0.15

Konstruktor

Konstruktor - bu sinf obyektlarini avtomatik initsializatsiya qilish uchun ishlatiladigan maxsus komponentali funksiya.

Konstruktorlar ko‘rinishi quyidagicha bo‘lishi mumkin:

<Sinf nomi> (<formal parametrlar ruyxati>) {<konstruktor tanasi>}

Bu komponenta funksiya nomi sinf nomi bilan bir xil bo‘lishi lozim.

Dasturchi tomonidan ko‘rsatilmagan holda ham **new** operator yordamida sinf obyekti yaratilganda yoki xotirada joylashganda konstruktor avtomatik ravishda chaqiriladi.

Konstruktor obyekt uchun xotirada joy ajratadi va ma'lumotlar – sinf a'zolarini initsializatsiyalaydi.

Konstruktor bir nechta xususiyatlarga ega:

- Konstruktorlar uchun qaytariluvchi turlar, xatto **void** turi ham ko‘rsatilmaydi;
- Konstruktor adresini hisoblash mumkin emas. Konstruktor parametri sifatida o`z sinfining nomini ishlatish mumkin emas, lekin bu nomga ko‘rsatkichdan foydalanish mumkin;
- Konstruktorlar vorislikka ega emas.

Konstruktorlar ixtiyoriy sinflar uchun doimo mavjud, lekin agarda u ko‘rsatilgan holda tavsiflanmagan bo‘lsa, u avtomatik ravishda yaratiladi. Ko‘rsatilmagan holda parametrsiz konstruktor va nusxa olish konstruktori yaratiladi. Agarda konstruktor ochiq holda tavsiflangan bo‘lsa, unda jimlik bo‘yicha konstruktor yaratilmaydi. Ko‘rsatilmagan holda umumiy (**public**) konstruktorlar yaratiladi.

Konstruktorni oddiy komponenta funksiya sifatida chaqirib bo‘lmaydi. Konstruktorni ikki xil shaklda chaqirish mumkin.

Sinf_nomi. Obyekt_nomi (konstruktor_haqiqiy_parametlari)

Sinf_nomi (konstruktor_haqiqiy_parametlari)

Birinchi shakl ishlatilganda hakikiy parametrlar ro‘yxati bo‘sh bo‘lmasligi lozim. Bu shakldan yangi obyekt ta'riflanganda foydalaniladi.

Konstruktorni ikkinchi shaklda chaqirish nomsiz obyekt yaratilishiga olib keladi. Bu nomsiz obyektdan ifodalarda foydalanish mumkin.

Misol:

```
#include<iostream>
using namespace std;
class complex {
double re, im;
public:
void show (){
cout << "re=" << re<<" im=" << im<<endl;
};
complex(double re1,double im1){re = re1; im = im1;}
};
int main() {
complex ss (5.9,0.15);
```

```
complex aa = complex (5.9,0.15);
aa.show();
system("pause");
return 0;
}
```
Natija:
re=5.9 im=0.15

Konstruktorlarda ko‘zda tutilgan qiymatlardan ham foydalanish mumkin.

Misol:

```
#include<iostream>
using namespace std;
class complex {
double re, im;
public:
void show (){
cout << "re=" << re<<" im=" << im<<endl;
};
complex(double re1=0.0, double im1=0.0){re = re1; im = im1;}
};
int main() {
complex a;
a.show();
complex aa(5.9,0.15);
aa.show();
system("pause");
return 0;
}
```
Natija:
re=0 im=0
re=5.9 im=0.15

C++ tilida konstruktorlarni ham qo‘shimcha yuklash mumkin. Quyidagi dasturda konstruktor **employee** qo‘shimcha yuklangan. Birinchi konstruktor dasturda ishchi, uning raqami va oyligi ko‘rsatilishini talab qiladi. Ikkinchi konstruktor oylikni kiritilishini so‘raydi. Sinf ta'rifi ichida ikkala konstruktor prototipi ko‘rsatilishi lozim:

```
#include <iostream>
using namespace std;
class employee{
public:
```

```
employee(long, float);
employee(long);
void show_employee(void);
private:
long employee_id;
float salary;
};
employee::employee(long employee_id, float salary){
employee::employee_id = employee_id;
if (salary < 5000000.0) employee::salary = salary;
else
employee::salary = 0.0;
}
employee::employee(long employee_id){
employee::employee_id = employee_id;
do{
cout << "Maosh kiriting 5000000 dan kichik: ";
cin >> employee::salary;
}
while (salary >= 5000000.0);
}
void employee::show_employee(void){
cout << "Raqam: " << employee_id << endl;
cout << "Maosh: " << salary << endl;
}
int main(){
cout<<"ishchi"<<endl;
employee worker(101, 10101.0);
worker.show_employee();
cout<<"manager"<<endl;
employee manager(102);
manager.show_employee();
system("pause");
return 0;
}
```

Initsializatorlar ro‘yxati. Konstruktor yordamida obyekt ma'lumotlarni initsiyalizatsiyalashni ikkita usuli mavjud:

Birinchi usulda parametrlar qiymatlari konstruktor tanasiga uzatiladi.

Ikkinchi usulda esa ushbu sinfdagi initsializatorlar ro‘yxatidan foydalanish nazarda tutilgan: bu ro‘yxat parametrlar ro‘yxati va konstruktor tanasi orasiga

joylashadi. Ro‘yxatdagi har bir initsializator konkret aniq komponentaga bog‘liq va quyidagi ko‘rinishga ega:

<nom> (<ifoda>)

Konstantalar va ilovalar uchun initsializatorlar ro‘yxatidan foydalanish majburiy, aks holda kompilyator xato chiqaradi.

Misol:

```
#include<iostream>
using namespace std;
class alpha {
int i; float e; char c;
public:
alpha(int ii, float ee, char cc) : i(8), e( i * ee + ii ),c(cc){}
void show(){cout<<"i="<<i<<" e="<<e<<" c="<<c<<endl;}
};
int main(){
alpha ss (5,5.9,'a');
ss.show();
system("pause");
return 0;
}
```

Natija:
i=8 e=52.2 c=a

Ochiq-oydin e'lonlar

Odatda bitta parametrli konstruktor e'lon qilingan sinf obyektlariga turlari avtomatik tarzda (yashirish holda) o‘z sinfi turiga qayta o‘zgaradigan qiymatlarni berish mumkin. Konstruktorni e'lon qilishda **explicit** konstruktoridan foydalanish mumkin:

explicit<konstruktorni_e'lon_qilish >

Bu holda berilgan sinf konstruktorlarini **explicit** kalit-so‘z bilan e'lon qilishda sinfning barcha obyektlariga faqat shunday qiymatlarni berish mumkin-ki, bu qiymatlar turlari o‘z sinfi turiga ochiq-oydin qayta o‘zgaradigan bo‘lishi kerak.

Misol:

```
#include <iostream>
using namespace std;
class Alpha{
string name;
```

```
int val;
public:
explicit Alpha(string s){name=s;val=0;}
Alpha(string s,int v){name=s;val=v;};
void show(){cout<<name.c_str()<<" "<<val;}
};
int main() {
Alpha a("tea",15);
a=Alpha("kofe");
a.show();
return 0;
}
```

Konstruktorlarning ochiq-oydin e'lonlari shuni talab qiladi-ki, nom berish operatorlaridagi qiymatlar qaysi sinfiy tur obyektlariga berilgan bo‘lsa, ular xuddi shu sinfiy turga qayta o‘zgartirilishini talab qiladi.

Konstanta usullar

Sinf usulini shunday ta'riflash mumkin-ki, murojaat qilayotgan obyektni (*this orqali) faqat o‘qishi mumkin. Buning uchun parametrlar ro‘yxati oxiriga **const** xizmatchi so‘zini qo‘shish lozim. Konstanta usulni, ya'ni **const** spetsifikatsiyasiga ega usulni doimiy obyektlar uchun chaqirish mumkin, bu spetsifikatsiyaga ega bo‘lmagan usullarni chaqirish mumkin emas.

Misol:

```
#include<iostream>
using namespace std;
class MyX {
int m;
public:
MyX(int k){m=k;}
int readme() const { return m; }
void writeme(int i) { m = i; }
};
int main(){
MyX mut(1);
const MyX con =mut;
mut.readme();
mut.writeme(7);
cout<<con.readme()<<endl;
```

```
cout<<mut.readme()<<endl;
system("pause");
return 0;
}
```
Natija:
1
7

Hatto konstantali ifodaga kirganiga qaramay o‘zgartirilishi mumkin bo‘lgan o‘zgaruvchan o‘zgaruvchining boshqa bir turi **mutable** modifikatori yordamida e'lon qilinadi:

mutable<o‘zgaruvchining nomi>;

mutable kalit-so‘zning vazifasi shundan iborat-ki, u biron-bir sinf ma'lumotlari a'zolarini spetsifikatsiya qiladi, bunda ushbu ma'lumotlar a'zolari mana shu sinfning konstantali funksiyalari tomonidan modifikatsiya qilinishi mumkin bo‘lishi kerak. Ma'lumotlar a'zosi **count** ni F1 konstantali funksiya modifikatsiya qiladigan misol:

```
#include <iostream>
using namespace std;
class ALPHA {
public:
mutable int count;
int F1 (int p=0)const {
count=++p;
return count;
}
};
int main() {
ALPHA a;
cout<<a.F1(3)<<endl;
return 0;
}
```

Qiymat olish va initsializatsiya

Nusxa olish konstruktori

Ko‘rsatilmagan holda **T::T(const T&)** ko‘rinishdagi nusxa konstruktori yaratiladi, bu erda T – sinf ismi. Har gal sinfga tegishli obyektlarni nusxasi olinayotganda nusxa konstruktori chaqiriladi. Xususan:

a) obyekt funksiyaga qiymati bo‘yicha uzatilsa;

b) funksiya qiymatlarini qaytaruvchi vaqtinchalik obyektlarni yaratishda;

v) boshqa obyektni initsializatsiyalash uchun obyektdan foydalanishda.

Agarda sinfda aniq bir nusxalash konstruktori ochiq ko‘rsatilgan bo‘lmasa, unda yuqorida aytib otilgan uch holatdan bittasi mavjud bo‘lsa, obyektni nusxalash bit bo‘yicha amalga oshiriladi, lekin bit bo‘yicha nusxalash har doim ham adekvat deb hisoblanmaydi. Xuddi shu hollarda xususiy nusxalash konstruktorini belgilamoq lozim.

Qiymat berish amali. Qiymat berish va initsializatsiya har xil amallardir. Qiymat berish amalini oshkor ravishda quyidagicha e'lon qilish mumkin **T::T&operator=(const T&).**

Bitta obyekt ikkinchisiga qiymat sifatida berilsa quyidagi qadamlar bajariladi:

1. Sinfda oshkor nusxa olish konstruktori mavjudligi aniqlanadi.
2. Agar mavjud bo‘lsa, murojaat huquqi tekshiriladi.
3. Operator qiymat berish uchun chaqiriladi, agar chaqirish mumkin bo‘lmasa, kompilyator xato haqida ma'lumot chiqaradi.
4. Agar oshkor operator mavjud bo‘lmasa, jimlik bo‘yicha har bir elementga qiymat beriladi.
5. Jimlik bo‘yicha chap tomondagi obyekt har bir elementiga o‘ng tomondagi obyekt har bir elementi qiymat sifatida beriladi.
6. Agar element obyekt bo‘lsa rekursiv 1-6 qadamlar qo‘llanadi.

Nusxalashni man etish. Ba'zi hollarda obyektlardan nusxa olishni va qiymat berishni mann etishga to`g`ri keladi. Masalan, maydonlar qiymatlari unikal bo‘lishi uchun. Buning uchun nusxa olish konstruktorini va qiymat berish amalini xususiy sifatida qo‘shimcha yuklash lozim.

Masalan:

#include<iostream>

```
using namespace std;
class TPoint{
double x,y;
TPoint(TPoint&);
TPoint& operator=(const TPoint& st);
public:
TPoint():x(0),y(0){};
TPoint(double x1,double y1 ){
x = x1; y = y1;
}
void show(){
cout<<x<<" "<<y<<endl;
}
};
int main() {
TPoint a(2,3);
TPoint b;
a.show();
b.show();
system("pause");
return 0;
}
```

Natija:

2 3

0 0

Quyida ma'lumotdan nusxa olishda xatolikka olib keluvchi holatlar ko‘rsatilgan:

TPoint A(1,2); // xatolikka olib kelmaydi

TPoint B; // xatolikka olib kelmaydi

TPoint D=A; // xato, nusxa olish konstruktori xususiy.

TPoint C(A); // xato, nusxa olish konstruktori xususiy.

B=A // xato, qiymat berish amali xususiy.

Destruktor

Sinfning biror obyekti uchun ajratilgan xotira obyekt yo‘qotilgandan so‘ng bo‘shatilishi lozimdir. Sinflarning maxsus komponentalari destruktorlar, bu vazifani avtomatik bajarish imkonini yaratadi. Destruktorni standart shakli quyidagicha:

~ <sinf_nomi> ( ) {<destruktor tanasi>}

Destruktor parametri yoki qaytariluvchi qiymatga ega bo‘lishi mumkin emas (xatto void turidagi). Dastur obyektni o‘chirganda destruktor avtomatik chaqiriladi.

Agarda sinfda destruktor ochiq ko‘rsatilmagan bo‘lsa, unda kompilyator ko‘rsatilgan obyekt egallagan xotirani bo‘shatuvchi destruktorni generatsiyalaydi. Boshqa obyektlar egallagan xotirani bo‘shatish kerak bo‘lsa, destruktorni ochiq aniqlash lozim.

Misol:

```
#include <iostream>
using namespace std;
class Person{
public:
Person (){
cout<<"Yaratidi"<<endl;
}
~Person (){
cout<<"O'chirldi"<<endl;
}
};
int main() {
{
Person work;
}
system("pause");
return 0;
}
```

Natija
Yaratidi
O'chirldi

Obyektlarga ko‘rsatkichlar.This ko‘rsatkichi

Obyektlar aniqlangandan so‘ngб shu obyektlarga ko‘rsatkichlar belgilash mumkin. Obyektning umumiy elementlariga murojaat uchun -> amalini yoki ism almashtirish va nuqta amalidan foydalanish mumkin.

Agarda konkret obyektga ishlov berish uchun sinf a'zosi, ya’ni funksiya chaqirilsa, unda shu funksiyaga obyektga belgilangan ko‘rsatkich avtomatik va

ko‘rsatilmagan holda uzatiladi. Bu ko‘rsatkich **this** ismiga ega va **x* this** kabi har bir funksiya uchun ko‘rsatilmagan holda belgilanadi.

Sinfni quyidagi ko‘rinishda tavsiflash mumkin:

```
#include<iostream>
using namespace std;
class Pair{
int number;
float persent;
public:
Pair( int number,float persent) {
this->number=number;
Pair::persent=persent;
}
void show() {
cout<<this->number<<" "<<Pair::persent<<endl;
}
};
int main(){
Pair a(5,1.5);
Pair*pa=&a;
pa->show();
(*pa).show();
return 0;
}
```

A'zolarga murojaat etishda **this** dan foydalanish ortiqcha. Asosan **this** bevosita ko‘rsatkichlar bilan manipulyasiya qilish uchun a'zo funksiyalarini yaratilishida foydalaniladi. Bu misolda elementga kvalifikatsion nomi orqali murojaat qilish ko‘rsatilgan.

Bu nom sinf nomi kvalifikatsiya operatori va element nomidan iborat.

Sinfning statik komponentalari

Sinf komponentasi yagona bo‘lib va hamma yaratilgan obyektlar uchun umumiy bo‘lishi uchun uni statik element sifatida ta'riflash, ya'ni **static** atributi orqali ta'riflash lozimdir. Obyektlarni yaratishda sinf statik ma'lumotlari takrorlanmaydi, ya'ni har bir statik komponentlar birdan-bir ko‘rinishda mavjud.

Statik elementlarga murojaat qilish uchun oldin initsializatsiya qilinishi lozim. Initsializatsiya quyidagicha amalga oshiriladi:

<Sinf-nomi>:: <kompleks-nomi> <initsializator>

Bu taklifni sinfni aniqlashdan so‘ng global sohada joylashtirish lozim. Faqatgina sinf statistik ma'lumotlarini initsializatsiyalashda u xotiraga ega bo‘ladi va unga murojaat etish mumkin. Sinf statik ma'lumotlarga faqatgina obyekt ismi orqali murojaat etishi mumkin: **<obyekt_nomi>.<komponenta_nomi>**

Lekin, statik komponetlarga sinf obyekti aniqlanmagan holda ham murojaat etish mumkin. Statistik komponentlarga nafaqat obyekt ismi, balki sinf ismi orqali ham murojaat etish mumkin.

<sinf_nomi> : : <komponenta_nomi>

Lekin shunday murojaat faqatgina ***public*** komponentlarga tegishli.

private statik komponentlarga tashqaridan murojaat etishda **funksiya – statik komponentlardan** foydalaniladi.

Bu funksiyalarni sinf ismi orqali chaqirish mumkin.

<sinf_nomi> : : <statik_funksiya_nomi>

Misol:

```
#include<iostream>
using namespace std;
class T
Point {
double x,y;
static int N;
public:
TPoint(double x1 = 0.0,double y1 = 0.0){N++; x = x1; y = y1;}
~TPoint(){N--;};
static int& count(){return N;}
};
int TPoint::N = 0;
int main() {
TPoint A(1.0,2.0);
TPoint B(4.0,5.0);
TPoint C(7.0,8.0);
cout<< "count="<<TPoint::count()<<endl;
system("pause");
return 0;
```

```
}
```

Natija:
count=3

Navbatdagi dastur **book_series** sinfini aniqlaydi. Bu sinf (seriya)ning barcha obyektlari (kitoblari) uchun bir xilda bo‘lgan **page_count** elementidan birgalikda foydalanadi. Agar dastur ushbu element qiymatini o‘zgartirsa, bu o‘zgarish shu ondayoq barcha sinf obyektlarida o‘z aksini topadi:

```
#include <iostream>
using namespace std;
class book_series{
public:
book_series(float);
void show_book(void);
void set_pages(int) ;
private:
static int page_count;
float price;
};
int book_series::page_count;
void book_series::set_pages(int pages){
page_count = pages;
}
book_series::book_series(float price){
book_series::price = price;
}
void book_series:: show_book (void){
cout << "Narx: " << price << endl;
cout << "Betlar: " << page_count << endl;
}
int main() {
book_series programming(213.95);
book_series word(19.95);
word.set_pages(256);
programming.show_book ();
word.show_book() ;
cout << endl << "page_count ning o'zgarishi " << endl;
programming.set_pages(512);
programming.show_book();
word.show_book();
system("pause");
return 0;
```

```
}
```

Ko‘rinib turganidek, sinf **page_count** ni **static int** sifatida e'lon qiladi. Sinfni aniqlagandan so‘ng, dastur shu vaqtning o‘zida **page_count** elementini global o‘zgaruvchi sifatida e'lon qiladi. Dastur **page_count** elementini o‘zgartirganda, o‘zgarish shu vaqtning o‘zidayoq **book_series** sinfining barcha obyektlarida namoyon bo‘ladi.

Statik funksiya-elementlardan foydalanish. Avvalgi dastur ma'lumotlar ***statik*** elementlarining qo‘llanishini ko‘rsatib bergan edi. C++ xuddi shunday usul bilan *statik* funksiya-elementlar (usullar)ni aniqlash imkonini beradi. Agar *statik* usul yaratilayotgan bo‘lsa, dastur bunday usulni, hatto uning obyektlari yaratilmagan holda ham, chaqirib olishi mumkin. Masalan, agar sinf sinfdan tashqari ma'lumotlar uchun qo‘llanishi mumkin bo‘lgan usulga ega bo‘lsa, siz bu usulni statik qila olishingiz mumkin bo‘lardi. Funksiyadan foydalanish uchun dastur uni **public** va **static** sifatida e'lon qilishi kerak. Masalan, quyidagi dasturda, xatto **book_series** sinfidagi obyektlar mavjud bo‘lmasa ham, bu sinfning **show_count()** usulidan foydalaniladi:

```
#include <iostream>
using namespace std;
class book_series{
public:
static int show_count() { return page_count;};
private:
float price;
static int page_count;
};
int book_series::page_count=256;
int main(){
cout << "page_count ning joriy qiymati ";
cout << book_series::show_count() <<"ga teng"<< endl;
system("pause");
return 0;
}
```

Nazorat savollari:

1. C++ tilida **class, struct** va **union** orasida qanday farq bor?

2. Sinf usullarini qo‘shimcha yuklash mumkinmi?
3. Konstruktorlar va destruktorlar vazifasini ko‘rsating.
4. Konstruktorlar xossalarini ko‘rsating.
5. Ochiq-oydin e'lonlar.
6. Konstanta usullar nima uchun ishlatiladi?
7. Obyektlar massivi yaratilganda qanday konstruktorlar chaqiriladi?
8. Destruktorlar qanday chaqiriladi?
9. Sinf usuli deb qanday usulga aytiladi?
10. Statik komponentalar xususiy bo‘lishi mumkinmi?

Misollar:

1. Talaba sinfini yarating. Sinfda parametrsiz va parametrli konstruktor, kiritish va chiqarish usullari yaratilsin.

2. Uchta tomoni bilan berilgan Uchburchak sinfini yarating va dasturda qo‘llang. Sinfda yuza va perimetrni hisoblash usullari bo‘lsin. Konstruktorda berilgan uch tomon haqiqatan uchburchak tashkil qilishi tekshirilsin.

3. Kriptografik sinf yarating. Sinfda Sezar usuli asosida shifrlash va deshifrlash usullari mavjud bo‘lsin. Kalit konstruktorda kiritilsin.

4. Futbolchi sinfini va dasturda qo‘llang. Sinfda parametrsiz va parametrli konstruktor, kiritish va chiqarish usullari yaratilsin.

5. Futbolchi (ism, yosh, amplua, gollar soni) sinfini yarating. Sinfda konstruktor va destruktor yarating.

Sinflar orasida munosabatlar

Sinflar orasidagi munosabatlar turlari

Har bir sinfni ikkita muhim jihati mavjud: u arxitektura birligi moduli hisoblanadi va bir necha ma'lumotlar turlarini aniqlagan holda sermazmun tushunchaga ega. Dastur tizimi sinflari hammasi bir-biri bilan o‘zaro aniq bog‘lanishda bo‘ladi.

Obyektli dasturlash tizimida ikkita asosiy tur sinflarni o‘zaro bog‘lanishi aniqlangan. Birinchi bog‘lanish “Mijoz va yetkazuvchi”, odatda mijoz bog‘lanishi deb ataladi yoki ichma-ich bog‘lanishda bo‘ladi. Ikkinchi bog‘lanish “Ota-onalar va vorislar” bu odatda vorislik deb nomlanadi.

Ta'rif 1. A va V sinflar “mijoz va yetkazuvchi” bog‘lanishida bo‘lsa, agar V sinfning maydoni A sinf obyekti bo‘lsa, A sinf V sinfning etkazuvchisi deb nomlanadi, V sinfi A sinfning mijozi deb ataladi.

Ta'rif 2. A va V sinflar "Ota-onalar va vorislar" bog‘lanishida deyiladi, agar V sinf e'lon qilishda A sinf ota sinf sifatida ko‘rsatilgan bo‘lsa, V sinf A sinfning vorisi deb ataladi.

Ikkala bog‘lanish – vorislik va ichma-ich joylashganlik tranzitivlik xossasiga ega. Agar V sinf A sinf mijozi, va S sinf V sinfning mijozi, bo‘lsa S sinf A sinfning mijozi bo‘ladi. Agar V sinf A sinf vorisi bo‘lsa, S sinf V sinfning vorisi bo‘lsa S sinf A sinfning vorisi bo‘ladi.

Yuqoridagi 1 va 2 ta'riflar to‘g‘ridan-to‘g‘ri mijoz va yetkazuvchi, vorislik munosabatlarini aniqlaydi (1-bosqich mijoz, 1-bosqich yetkazuvchi va hokazo). Rekursiv tarzda shuni aniqlash mumkin-ki: to‘g‘ridan-to‘g‘ri k-bosqichdagi mijoz k+1 bosqichdagi mijoz bilan bog‘lanishda bo‘ladi. Vorislik bog‘lanishida, tabiiy tildagi tushunchalar qo‘llanadi. To‘g‘ridan-to‘g‘ri bo‘lmagan vorislik ajdodlar va avlodlar bog‘lanishi deyiladi.

"has" (egalik) va "is a" (bir xillik) munosabatlari

Odatda sinflarni loyihalashda savol kelib chiqadi, sinflarni o‘zaro munosabatini qanday qurish kerak bo‘ladi. Ikkita oddiy sinflarga misol ko‘ramiz – Square va Rectangle, ular kvadrat va to‘g‘ri to‘rtburchakdir. Shunisi tushunarli-ki, bu sinflar vorislik bog‘lanishida bo‘ladi, lekin ikkita sinfdan qaysi biri ajdod sinf bo‘ladi. Yana ikkita sinfga misol – Car va Person, ya'ni mashina va inson. Bu sinflar bilan **Person_of_Car**, ya'ni mashina egasi sinfi qanday aloqada bo‘lishi mumkin? Bu ikki sinf bilan vorislik bog‘lanishida bo‘lishi mumkinmi? Sinflarni loyihalash bilan bog‘liq bu savollarga javob topish uchun shuni nazarda tutish

kerak-ki, "mijoz-yetkazuvchi" bog‘lanishi "ega" ("**has**") bog‘lanishini, vorislik bog‘lanishi esa "bir xil" ("**is a**") bog‘lanishi tushunchalarini ifodalaydi. **Square** va **Rectangle** sinflari misoli tushunarli, har bir obyekt kvadrat to‘g‘ri to‘rtburchakdir, shuning uchun bu sinflar o‘rtasida vorislik bog‘lanishi ifodalanadi, va **Rectangle** sinfi vorislik sinfini ifodalaydi. **Square** sinfi uning vorisidir. Mashina egasi mashinaga ega va insondir. Shuning uchun **Person_of_Car** sinfi **Car** sinfning mijozi bo‘lib hisoblanadi va Person sinfning vorisidir.

Sinflar va obyektlar orasida munosabatlar.

Munosabatlar turlari. Til qoidalariga ko‘ra quyidagi munosabatlarni ajratish mumkin:

Bir sinf ta'rifi boshqa sinf obyektini yoki obyektga ko‘rsatkich yoki ilovani o‘z ichiga oladi.

Bir sinf ta'rifi ikkinchi sinf ichida yotadi.

Bir sinf obyekti ikkinchi sinf obyekti usuli parametridir.

Bundan tashqari egalik va foydalanish munosabatlarini ajratish mumkin.

Agar bir sinf obyekti yaratilishi yoki o‘chirilishi ikkinchi sinf obyektini yaratilishi yoki o‘chirilishiga olib kelsa, egalik munosabati mavjud.

Boshqa hollarda foydalanish munosabati mavjud.

Munosabatlar alohida turi bu do‘stona munosabatlardir.

Obyektlar sinf a'zosi sifatida

Agar sinfda boshqa sinf obyektlari mavjud bo‘lsa, unda, sinf a'zosi - konstruktor uchun parametri sinf konstruktori ta'rifida (lekin tavsifida emas) ko‘rsatiladi. A'zosi uchun konstruktor uning uchun parametrlar ro‘yxatini belgilovchi konstruktor bajarilgandan so‘ng chaqiriladi.

Boshqa a'zolar (agarda ular mavjud bo‘lsa) konstruktorlari uchun shuning singari parametrlarni belgilash mumkin.

A'zolar uchun parametrlar ro‘yxati bir-biridan vergul (ikkita nuqta emas) yordamida ajratiladi, ammo a'zolar uchun initsializatorlar ro‘yxatini ixtiyoriy tartibda belgilash mumkin.

Konstruktorlar sinf tavsifida belgilangan tartibda chaqiriladi.

Misol:

```
#include<iostream>
using namespace std;
class alpha {
int i;
public:
alpha(int ii) {i=ii;}
void show(){
cout<<i<<endl;
}
};
class betta {
int k;
alpha member;
alpha fri;
public:
betta(int ii, int kk): fri(ii),member(ii){k=kk;}
void show(){
cout<<k<<endl;
member.show();
fri.show();
}
};
int main() {
alpha a(1);
betta b(1,2);
a.show();
b.show();
system("pause");
return 0;
}
```

Natija:

```
1
2
1
1
```

Obyekt maydon sifatida. Murakkab sinflarni hosil qilishda oldin uni tashkil etuvchi oddiyroq sinflarni e'lon qilib, keyin esa ularni birlashtirish orqali sinfni hosil qilish maqsadga muvofiqdir. Masalan, g‘ildirak sinfi, motor sinfi, uzatish korobkasi sinfi va boshqa sinflarni hosil qilib, keyin esa ularni birlashtirish orqali avtomobil sinfini qurish oldimizga turgan masalani yechishni ancha osonlashtiradi.

Ikkinchi misolni ko‘rib chiqamiz. To‘g‘ri to‘rtburchak chiziqlardan tashkil topgan. Chiziq esa ikki nuqta orqali aniqlanadi. Har bir nuqta x va y koordinatalar yordamida aniqlanadi. Quyidagi dasturda to‘rtburchak sinfi ko‘rsatilgan. To‘g‘ri to‘rtburchak diagonal bo‘yicha ikki nuqta va ikki tomon yordamida aniqlanadi. Shuning uchun oldin har bir nuqtaning x, y koordinatalarini saqlash uchun **Point** sinfi e'lon qilingan.

Misol:

```
#include <iostream>
using namespace std;
class Point{
public:
Point(int x1=0,int y1=0){
x=x1;y=y1;
}
int  GetX() const {return x;}
int  GetY() const {return y;}
private:
int x;
int y;
};
class Rectangle{
public:
Rectangle(int x1,int y1,int x2,int y2):
p1(x1,y1),p2(x2,y2)
{
a=x2-x1;
b=y2-y1;
};
Rectangle(Point a1,Point a2):p1(a1),p2(a2){
a=a2.GetX() - a1.GetX();
b=a2.GetY() - a1.GetY();
};
int Per() { return 2*(a+b); }
```

```
int Sq()  {return a*b; }
private:
Point p1,p2;
int a,b;
};
int main(){
Rectangle X(10, 20, 50, 80);
cout << "Perimetr="<<X.Per()<<endl;
cout << "Yuza="<<X.Sq()<<endl;
cout<<endl;
Point a(2,4);Point b(5,6);
Rectangle Y(a,b);
cout << "Perimetr="<<Y.Per()<<endl;
cout<<"Yuza="<<Y.Sq();
return 0;
}
```

Natija:
Perimetr=200
Yuza=2400

Perimetr=10
Yuza=6

Foydalanish. Sinf boshqa sinf umumiy yoki statik elementlaridan boshqa sinf obyektini yaratmasdan va o‘chirmasdan foydalanishi mumkin. Bu holda bir sinf obyekti ikkinchi sinf usuli argumenti bo‘lib keladi. Shuni hisobga olish lozim-ki, agar konstruktor standart turni obyektga o‘zgartirishi mumkin bo‘lsa, haqiqiy parametr sifatida standart turdagi o‘zgaruvchini uzatish mumkin. Buni oldini olish uchun konstruktor ta'rifida **explicit** modifikatori ko‘rsatiladi.

Sinf boshqa sinf obyektiga ilovani o‘z ichiga olishi mumkin.

Misol:

```
using namespace std;
#include<iostream>
using namespace std;
class Account {
unsigned number;
int value;
public:
explicit Account(unsigned number1,int value1=0){
number=number1;
value=value1;
```

```
}
unsigned GetNum() const {return number;}
int GetVal()const {return value;};
};
class PrintAccount{
public:
static void Print(const Account& a){
cout<<"number="<<a.GetNum()<<" value="<<a.GetVal()<<endl;
}
};
class ShowAccount{
const Account& a;
public:
ShowAccount(Account pa):a(pa){};
void Print(){
cout<<"number="<<a.GetNum()<<" value="<<a.GetVal()<<endl;
}
};
int main() {
Account mut(1,5);
PrintAccount::Pri
nt(mut);
ShowAccount P(mut);
P.Print();
return 0;
}
```

Natija:

number=1 value=5
number=1 value=5

Sinf do‘stlari ta'rifi

C++ da aniq sinf do‘stlariga bu sinfning xususiy elementlariga murojaat etish imkonini beradi. C++ da funksiya ikkinchi sinfga do‘stona sinfligini ko‘rsatish uchun **friend** kalit so‘zidan foydalanish.

Do‘stona funksiya ta'riflash:

friend <funksiya prototipi >

Misol:

```
#include<iostream>
using namespace std;
class alpha {
int i;
```

```
public:
alpha(int ii) {i=ii;}
friend void show(alpha a);
};
void show(alpha a){cout<<a.i<<endl;}
int main() {
alpha d(4);
show(d);
return 0;
}
```

C++ da bitta sinf ikkinchi sinfga do‘stona sinfligini ko‘rsatish uchun **friend** kalit so‘zidan foydalanish va do‘stona sinf ismini ikkinchi sinf tavsifiga kiritish lozim.

Do‘stona sinflarni ta'riflash:

friend <sinf nomi >

Do‘stona sinf oldin e'lon qilingan bo‘lishi kerak.

Masalan, quyidagi **book** sinfi **librarian** sinfini o‘ziga do‘stona sinf deb belgilagan.

Shuning uchun **librarian** sinf obyektlari **book** sinfning xususiy elementlariga, nuqta operatoridan foydalangan holda, to‘g‘ridan-to‘g‘ri murojaat etishi mumkin:

```
#include <iostream>
using namespace std;
class librarian;
class book{
char title[20];
char author[10];
char catalog[6];
public:
friend librarian;
book(char title[], char author[], char catalog[]){
strcpy(book::title,title);
strcpy(book::author,author);
strcpy(book::catalog,catalog);
}
void show_book(void){
cout << "Title: " << title << endl;
cout << "Author: " << author << endl;
cout << "Catalog: " << catalog << endl;
}
```

```
};
class librarian{
public:
void change_catalog(book& this_book, char new_catalog[]){
strcpy(this_book.catalog,new_catalog);
}
void show_catalog(book this_book){
cout<<this_book.catalog<<endl;
}
};
int main() {
book programming( "Learning C++ ", "Jamsa", "P101");
librarian library;
programming.show_book();
library.change_catalog(programming, " C++ 101");
programming.show_book();
return 0;
}
```

Ko‘rib turganimizdek, dastur **librarian** sinfining **change_catalog** funksiyasiga **book** obyektini adres orqali bermoqda. Bu funksiya sinfning **book** elementini o‘zgartirgani uchun, dastur parametrni adres orqali uzatishi lozim. **Book** sinfi aniqlanishidan **friend** operatori o‘chirib yuborilsa, C++ kompilyatori har gal book sinfi xususiy ma'lumotlariga murojaatda sintaksik xato haqida xabar chiqaradi.

Sinf e'loni. Sinfni tavsiflamasdan e'lon qilish mumkin. E'lon qilingan, lekin ta'rifi berilmagan sinfdan foydalanish cheklangandir. Bunday sinf hajmini aniqlab bo‘lmaydi.

Lekin bunday sinfga ko‘rsatkich yoki ilova e'lon qilish mumkin, chunki ko‘rsatkich yoki ilova hajmi sinf hajmiga bog‘liq emas. Lekin bunday ko‘rsatkichga qiymat olish (*) amalini qo‘llash mumkin emas. Bundan tashqari sinf elementiga murojaat qilish ham mumkin emas.

Do‘stlar sonini chegaralash

Agarda bir nechta sinf funksiyalariga boshqa sinfning xususiy ma'lumotlariga murojaat qilish kerak bo‘lsa, u holda C++ do‘stona sinfning faqatgina belgilangan funksiyalari xususiy elementlarga murojaat etishiga imkoniyat beradi.

Masalan, faqatgina **change_catalog** va **get_catalog** funksiyalarga **book** sinfning xususiy elementlariga murojaat qilish kerak bo`lsin. Quyida ko‘rsatilgandek, **book** sinfning ichida faqatgina shu funksiyalarda xususiy funksiyalarga murojaat chegarasini qo‘yishi lozim:

Misol:

```
#include <iostream>
using namespace std;
class book;
class librarian{
public:
void change_catalog(book& this_book, char new_catalog[]){
strcpy(this_book.catalog,new_catalog);
}
void show_catalog(book this_book){
cout<<this_book.catalog<<endl;
}
};
class book{
char title[20];
char author[10];
char catalog[6];
public:
friend void change_catalog(book& this_book, char new_catalog[]);
friend void show_catalog(book this_book);
book(char title[], char author[], char catalog[]){
strcpy(book::title,title);
strcpy(book::author,author);
strcpy(book::catalog,catalog);
}
void show_book(void){
cout << "Title: " << title << endl;
cout << "Author: " << author << endl;
cout << "Catalog: " << catalog << endl;
}
};
```

```
int main() {
book programming( " C++", "Jamsa", "P101");
librarian library;
programming.show_book();
library.change_catalog(programming, "C++ 101");
programming.show_book();
return 0;
}
```

Ko‘rib turganimizdek, **friend** operatorlari xususiy elementlarga murojaat qiluvchi hamma do‘stona funksiyalarini to‘liq prototiplarini o‘z ichiga oladi.

Agar dastur bir sinfdan boshqasiga murojaat qilsa va sinflar aniqlanish tartibi noto‘g‘ri bo‘lsa, sintaksik xatoga duch kelish mumkin. Bizning holda **book** sinfi **librarian** sinfida e'lon qilingan funksiyalar prototiplariga murojaat qilmoqda. Shuning uchun **librarian** sinfi aniqlanishi **book** sinfi aniqlanishidan oldin kelishi kerak, biroq **librarian** sinfi **book** sinfiga murojaat qilmoqda.

Dasturda **book** sinfi ta'rifini **librarian** sinfi ta'rifidan oldin qo‘yib bo‘lmagani uchun C++ **book** sinfini e'lon qilish imkonini beradi va shu bilan u kompilyatorga bunday sinf borligi haqida xabar beradi va keyinroq o‘zi ham ta'riflanadi. Quyida buni qanday amalga oshirish keltirilgan:

class book; // sinf elon qilinishi

Lokal sinflar

Sinf blok ichida, masalan funksiyada tariflanishi mumkin. Bunday sinf model sinf deb ataladi. Lokal sinf komponentalariga shu sinf tariflangan blok yoki funksiya tashqarisida murojaat qilish mumkin emas. Lokal sinf statik komponentlarga ega bo‘lishi mumkin emas. Lokal sinf ichida shu sinf aniqlangan soniga tegishli nomlari; statik (**static**) o‘zgaruvchilar; tashqi (**extern**) o‘zgaruvchilar va tashqi funksiyalardan foydalanish mumkin. Avtomatik xotira turiga tegishli o‘zgaruvchilardan foydalanish mumkin emas. Lokal sinflar komponent funksiyalari faqat joylashinuvchi (**inline**) funksiya bo‘lishi mumkin.

Lokal sinfning o‘zi joylashgan nomlar fazosidagi nomlarga murojaati cheklangandir. U faqat turlar nomlari, statik o‘zgaruvchilarga va sanovchi elementlariga murojaat qilishi mumkin.

Misol:

```
#include <iostream>
using namespace std;
void BFunk(){
static int si=5;
enum Loc { a = 1024, b };
class Bar{
public:
Loc locVal;
int barVal;
void SetBar ( Loc l = a ) {
barVal = si;
locVal = b;
};
void show(){
cout<<barVal<<" "<<locVal<<endl;
};
};
Bar ba;
ba.SetBar();
ba.show();
};
int main(){
BFunk();
system("pause");
return 0;
}
```

Natija:

5 1025

Joylashtirilgan sinflar

Joylashtirilgan sinf deb, boshqa sinf global sohasida ta'riflangan sinfga aytiladi.

Joylashtirilgan sinf ismi tashqi sinfda ko‘rinadi. Tashqi sinf joylashtirilgan sinf xususiy elementlariga murojaat qilish huquqiga ega emas. Joylashtirilgan sinf ta'rifi tashqi sinf tashqarisida berilishi mumkin.

Kompilyator joylashtirilgan sinf ta'rifini ko‘rmaguncha faqat ko‘rsatkich va ilova e'lon qilish mumkin. Joylashtirilgan sinf tashqi sinf nostatik elementlariga murojaat qilolmaydi, hatto ular umumiy murojaat huquqiga ega bo‘lsa ham. Statik elementlarga, tur nomlariga va sanovchi elementlariga ko‘rsatkich yoki ilova orqali murojaat qilish mumkin, albatta ular umumiy bo‘lsa.

Quyidagi misolda moddiy nuqta sinfi yaratilib, uning ichida nuqta sinfiga ta'rif berilgan va nuqta (**point**) sinfi obyekti maydon sifatida kelgan:

Misol:

```
#include <iostream>
using namespace std;
class FPoint{
public:
//Class Point
class Point{
public:
Point(int x1=0,int y1=0){
x=x1;y=y1;
}
int  GetX() const {return x;}
int  GetY() const {return y;}
private:
int x;
int y;
};
//
FPoint(int x1,int y1,double w1):p(x1,y1),w(w1){};
void show(){
cout<<"coord x="<<p.GetX()<<endl;
cout<<"coord y="<<p.GetY()<<endl;
cout<<"massa w="<<w;
}
private:
Point p;
double w;
};
int main() {
cout<<"fizik point "<<endl;
FPoint X(1, 2, 5.5);
X.show();
cout<<"\n\nsimple point"<<endl;
FPoint::Point Y(2,3);
```

```
cout<<"coord x="<<Y.GetX()<<endl;
cout<<"coord y="<<Y.GetY()<<endl;
system("pause");
return 0;
}
```

Natija:

```
fizik point:
coord x=1
coord y=2
massa w=5.5

simple point:
coord x=2
coord y=3
```

Oddiy nuqta sinfi moddiy nuqta sinfining umumiy seksiyasiga joylashtirilgan.

Dasturda moddiy nuqta sinfi obyektidan tashqari nuqta sinfi obyekti ham yaratilgan. Lekin bu sinf nomi oldida moddiy nuqta nomi va kvalifikatsiya operatori joylashtirilgan.

Keyingi misolimizda nuqta sinfi ta'rifi moddiy nuqta sinfining xususiy elementlari seksiyasiga joylashtirilgan:

Misol:

```
#include <iostream>
using namespace std;
class FPoint {
public:
FPoint(int x1,int y1,double w1):p(x1,y1),w(w1){};
void show() {
cout<<"coord x="<<p.GetX()<<endl;
cout<<"coord y="<<p.GetY()<<endl;
cout<<"massa w="<<w<<endl;
}
private:
//Class Point
class Point {
public:
Point(int x1=0,int y1=0) {
x=x1;y=y1;
}
int  GetX() const {return x;}
int  GetY() const {return y;}
```

```
private:
int x;
int y;
};
//
Point p;
double w;
};
int main() {
cout<<"fizik point "<<endl;
FPoint X(1, 2, 5.5);
X.show();
system("pause");
return 0;
}
```

Natija:
fizik point:
coord x=1
coord y=2
massa w=5.5

Bu misolda nuqta sinfi hamma elementlari umumiy, lekin dasturda bu sinfdan foydalanib bo‘lmaydi.

Obyektlar massivi

Obyektlar massivini ta'riflash uchun sinf ko‘zda tutilgan (parametrsiz) konstruktorga ega bo‘lishi kerak.

Obyektlar massivi ko‘zda tutilgan konstruktor tomonidan yoki har bir element uchun konstruktor chaqirish yo‘li bilan initsializatsiya qilinishi mumkin.

Misol:

```
#include<iostream>
using namespace std;
class complex{
public:
void show (){ cout << "re=" << re<<" im=" <<im<<endl;};
complex(double re1 = 0.0,double im1 = 0.0){re = re1; im = im1;}
private:
double re, im;
};
void show_array(int n,complex a[]){
```

```
for(int i=0;i<n;i++) a[i].show();
};
int main(){
complex  b[2]={complex (10),complex (100,5)};
show_array(2,b);
return 0;
}
```

Quyidagi misolda **player** sinfi kiritiladi. Dasturda sinf funksiyasi **show_player** va konstruktor tashqarisida ta'riflanadi. So‘ngra **player** turidagi ikki massiv yaratilib, har biri haqidagi ma'lumot ekranga chiqariladi:

```
#include <iostream>
using namespace std;
class player {
public:
player();
player (int namber,int weight, int age);
void show_player (void);
private:
int namber;
int weight;
int age;
};
player::player(){
namber=0;
weight = 0;
age = 0;
};
player::player(int namber,int weight, int age){
player::namber=namber;
player::weight = weight;
player::age = age;
};
void player::show_player (void){
cout<<"Nomer: " << namber << endl;
cout<<"Vazn: " << weight << endl;
cout<<"Yosh: " << age << endl;
}
class array_player{ public:
void show_array(player a[],int n){
for(int i=0;i<n;i++){
a[i].show_player();cout<<endl;
}
```

```
}
void input_array(player a[],int n){
int namber;
int weight,age;
for(int i=0;i<n;i++){
cin>>namber>>weight>>age;
a[i]=player(namber,weight,age);
}
}
};
int main(){
array_player arr;
player happy[]={player(1,58,24),player(2,72,35)};
arr.show_array(happy,2);
player matt[2];
arr.input_array(matt,2);
arr.show_array(matt,2);
system("pause");
return 0;
}
```

Natija:
Nomer: 1
Vazn: 58
Yosh: 24

Nomer: 2
Vazn: 72
Yosh: 35

Nazorat savollari:

1. Qachon sinflar orasida egalik munosabati mavjud bo‘ladi?
2. Do‘stona sinflar aniqlanish shaklini ko‘rsating.
3. Sinf do‘stlaridan nima uchun foydalaniladi?
4. Do‘stona sinfni e'lon qilish sinfning qaysi seksiyasida ekanligi ahamiyatlimi?
5. Sinflar qanday qilib boshqa sinflardan tashkil topishi mumkin?
6. Agar bir sinf obyekti boshqa sinf maydoni bo‘lsa, birinchi sinf maydonlariga qanday murojaat qilinadi?

7. Agar bir sinf obyekti boshqa sinf maydoni bo‘lsa, birinchi sinf konstruktori qanday chaqiriladi?
8. Lokal sinf deb qanday sinfga aytiladi?
9. Lokal sinflar obyektlari qanday aniqlanadi?
10. Lokal sinf komponentalariga shu sinf tariflangan blok yoki funksiya tashqarisida murojaat qilish mumkinmi?

Misollar:

1. REKORD sinfini yarating. Bu sinfga do‘stona sinf yarating. Do‘stona sinf usullari ma'lum shartga mos obyektlarni chiqarsin. Dasturda statik obyektlar massivi yarating. Do‘stona sinf obyekti yordamida shartga mos massiv elementlarini chiqaring.

2. AVTOBUS sinfini yarating. Bu sinfga do‘stona sinf yarating. Do‘stona sinf usullari ma'lum shartga mos obyektlarni chiqarsin. Dasturda statik obyektlar massivi yarating. Do‘stona sinf obyekti yordamida shartga mos massiv elementlarini chiqaring.

3. SANA sinfini yarating. Bu sinf obyektidan maydon sifatida foydalanib MASHG‘ULOT sinfini yarating.

4. Nuqta sinfini yarating. Bu sinf asosida uchburchak sinfini yarating va dsturda qo‘llang. Bu sinfda perimetr, yuzani hisoblash va uchburchakni chizish usullari mavjud bo‘lsin.

5. Yuqorida ko‘rsatilgan misolda nuqta sinfini uchburchak sinfi ichida lokal sinf sifatida joylashtiring.

Sinflarda vorislik

Vorislikda murojaat huquqlarini boshqarish

Vorislik o‘zining barcha ajdodlarining xususiyatlari, ma'lumotlari, usullari va voqealarini meros qilib oladigan hosil **a** sinfini e'lon qilish imkoniyatini beradi. Shuningdek yangi tavsiflarni e'lon qilishi hamda meros sifatida olinayotgan ayrim

funksiyalarni qo‘shimcha yuklashi mumkin. Bazaviy sinfning ko‘rsatib o‘tilgan tavsiflarini meros qilib olib, yangi yaratilgan sinfda ushbu tavsiflarni kengaytirish, toraytirish, o‘zgartirish, yo‘q qilish yoki o‘zgarishsiz qoldirish mumkin.

Hosila sinfni e'lon qilishning umumlashgan sintaksisi:

class <sinf_nomi>:[<kirish_huquqini_beruvchi_spetsifikator>|
<ajdod_sinf_nomi> {…}

Agar murojaat huquqini boshqaruvchi spetsifikator ko‘rsatilmagan bo‘lsa, hosila sinfdagi meros komponentalar **private,** ya'ni xususiy murojaat huquqiga ega bo‘ladi. Murojaat huquqini boshqaruvchi spetsifikatorlar ko‘rsatilsa quyidagicha o‘zgaradi:

- **private.** Meros bo‘lib o‘tayotgan (ya'ni himoyalangan va umumiy) elementlar **private,** ya'ni xususiy murojaat huquqiga ega bo‘ladi.
- **protected**. Meros bo‘lib o‘tayotgan (ya'ni himoyalangan va umumiy) elementlar **protected,** ya'ni himoyalangan murojaat huquqiga ega bo‘ladi.
- **public**. Meros bo‘lib o‘tayotgan (ya'ni himoyalangan va umumiy) elementlar murojaat huquqlari o‘zgarmaydi.

Agarda hosila sinf **struct** kalitli so‘z yordamida ta'riflangan bo‘lsa va murojaat huquqini boshqaruvchi spetsifikator ko‘rsatilmagan bo‘lsa, meros komponentalar **public,** ya'ni umumiy murojaat huquqiga ega bo‘ladi.

Asos yoki hosila sinfni **union** so‘zi yordamida ta'riflab bo‘lmaydi, ya'ni jamlanmalardan vorislikda foydalanish mumkin emas.

Quyidagi misolda tavsiflarni kengaytirish va cheklash usullarining qo‘llanishini ko‘rib chiqamiz:

Misol:

```
#include<iostream>
using namespace std;
class Base {
public:
void display(){ cout << "Hello!  \n"; }
};
class Derived : public Base {
public:
void display(){
Base::display();
```

```
cout <<"GoodBy! \n";
}
};
int main() {
Derived d;
d.display();
system("pause");
return 0;
}
```
Natija:
Hello!
GoodBy!

Bu misolning **Base::display()** qatorida asos sinf **display()** usuliga oshkor murojaat qilinadi. Lekin bu yo‘l bilan xususiy elementlarga murojaat qilib bo‘lmaydi.

Konstruktor va destruktorlarda vorislik

Konstruktorlar meros bo‘lmagani uchun, hosila sinfni yaratishda undan meros bo‘lgan ma'lumot-a'zolari asosiy (bazaviy) sinf konstruktori orqali initsializatsiyalanishi lozim. Asosiy sinf konstruktori avtomatik ravishda chaqiriladi va hosila sinfni konstruktoridan oldin bajariladi. Asosiy (bazaviy) sinfni konstruktorining parametrlari hosila sinf konstruktorini aniqlashda ko‘rsatiladi. Shunday qilib, argumentlarni hosila sinf konstruktoridan asosiy (bazaviy) sinf konstruktoriga uzatish vazifasi bajariladi.

Masalan:

```
#include<iostream>
using namespace std;
class Basis{
int a,b;
public:
Basis(int x,int y){a=x;b=y;}
void show(){cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;}
};
class Inherit:public Basis{
int sum;
public:
```

```
Inherit(int x,int y, int s):Basis(x,y){sum=s;}
void show(){
Basis::show();
cout<<"sum="<<sum<<endl;
}
};
int main(){
Inherit a(1,2,3);
a.show();
system("pause");
return 0;
}
```

Natija:
a=1
b=2
sum=3

Sinf obyektlari pastdan tepaga qarab yaratiladi: avvalo asosiy (bazaviy), keyin esa komponent – obyektlar (agarda ular mavjud bo‘lsa), undan keyin esa hosila sinfning o‘zi. Shunday qilib, hosila sinfning obyekti quyi obyekt sifatida asosiy (bazaviy) sinf obyektini o‘z ichiga oladi.

Obyektlar teskari tartibda o‘chiriladi: avvalo hosila, keyin uning komponent – obyektlari, undan keyin esa asosiy (bazaviy) obyekt.

Shunday qilib obyektni o‘chirish tartibi uning konstruktorlash tartibiga nisbatan teskari bo‘ladi.

Quyida keltirilgan dasturda ikkita oddiy geometrik obyekt - doira va silindrning sinflar tabaqalanishi e'lon qilingan.

Dastur shunday tuzilgan-ki, bunda doiraning r-radiusi va silindrning h-balandligi o‘zgaruvchilarining ichki qiymatlari yaratilayotgan obyektlar parametrlarini aniqlashi kerak. **Circle** bazaviy sinfi doirani modellashtiradi, **Cylinder** hosila sinfi esa silindrni modellashtiradi.

```
#include <iostream>
#include <cmath>
using namespace std;
const double pi = 4 * atan(1);
class Circle{
protected:
double r;
```

```
public:
Circle (double rVal =0) : r(rVal) {}
void setRadius(double rVal) { r = rVal; };
double getRadius() { return r; };
double Area() { return pi*r*r; };
void showData();
};
class Cylinder : public Circle {
protected:
double h;
public:
Cylinder(double hVal = 0, double rVal = 0): h(hVal), Circle(rVal) { };
void setHeight(double hVal) { h = hVal; }
double getHeight() { return h; };
double Area() { return 2*Circle::Area()+2*pi*r*h; };
void showData();
};
void Circle::showData(){
cout <<"Doira radiusi = "<<getRadius()<<endl;
cout<<"Aylana maydoni = "<<Area()<<endl;
};
void Cylinder::showData(){
cout << "Asos radiusi = "<< getRadius()<<endl;
cout<< "Silindr balandligi = "<< getHeight()<< endl;
cout<<"Yuza maydoni = "<<Area ()<< endl;
};
int main(){
Circle circle(2) ;
Cylinder cylinder(10, 1);
circle.showData ();
cylinder.showData();
return 0;
}
```

Circle sinfining e'loni **r** ma'lumotlarining yagona a'zosi, konstruktor va qator usullardan iborat. Obyektni yaratishda konstruktor **r** ma'lumotlar a'zosini doira radiusining boshlang‘ich qiymati bilan nomlaydi (initsiallashtiradi). Konstruktorning yangi sintaksisini ko‘rsatib o‘tamiz: chaqirishda u bazaviy konstruktor sinfiga, shuningdek, ikki nuqtadan keyin ko‘rsatilgan har qanday ma'lumotlar a'zosiga murojaat qilishi mumkin. Bizning misolimizda **r** ma'lumotlari

a'zosi unga **rVal** parametri bilan murojaat qilish orqali «yaratiladi» va nulli qiymat bilan nomlanadi (initsiallashtiriladi).

setRadius usuli **r** ma'lumotlar a'zosi qiymatini belgilaydi, **getRadius** usuli esa uni qaytaradi. **Area** usuli doira maydonini qaytaradi. **showData** usuli aylana radiusi va doira maydonining qiymatlarini chiqarib beradi.

Circle sinfining hosilasi deb e'lon qilingan **Cylinder** sinfi **h** – yagona ma'lumotlar a'zosi, konstruktor va qator usullardan iborat. Bu sinf **r** ma'lumotlar a'zosini silindr asosi radiusini saqlash uchun hamda **setRadius** va **getRadius** usullarini meros qilib oladi. Obyektni yaratishda konstruktor **r** va **h** ma'lumotlar a'zolarini boshlang‘ich qiymatlar bilan nomlaydi. Konstruktorning yangi sintaksisini ko‘rsatib o‘tamiz: bizning holatda **h** ma'lumotlar a'zosi **hVal** argumentining qiymati bilan nomlanadi (initsiallashtiriladi), **r** ma'lumotlar a'zosi esa **rVal** argumentiga ega bo‘lgan bazaviy sinf konstruktorini chaqirish bilan nomlanadi.

setHeight funksiyasi **h** ma'lumotlar a'zosi qiymatini belgilaydi, **getHeight** esa qaytaradi. **Circle::Area** funksiyasi bazaviy sinfdan meros olingan funksiyani, silindr yuzasi maydonini qaytarish uchun, ortiqcha yuklaydi. **showData** funksiyasi esa silindr asosining radiusi, balandligi va yuzasining maydoni qiymatlarini chiqarib beradi.

main funksiyasi **Circle** sinfiga mansub 2 radiusli **Circle** aylanasini hamda balandligi 10, asosining radiusi 1 bo‘lgan **Cylinder** sinfiga mansub **cylinder** silindrni yaratadi, keyin yaratilgan obyektlarning parametrlarini chiqarish uchun **showData** ga murojaat qiladi.

Aylana radiusi=2 Doira maydoni=12.566

Asos radiusi=1 silindr balandligi=10 YUzasining maydoni=69.115

Ko‘plikdagi vorislik va virtual sinflar

Bir necha to‘g‘ridan-to‘g‘ri bazaviy sinf mavjudligi ko‘plik vorislik deb ataladi. Bir sinf ikkinchi sinfning to‘g‘ridan to‘g‘ri faqat bir marta ajdodi bo‘lishi mumkin. Lekin bilvosita bir necha marta ajdodi bo‘lishi mumkin. Agar sinf bir xil

komponentalarga ega ikki sinf avlodi bo‘lsa, komponentaga murojaat qilish uchun kvalifikatsion ismdan foydalanish lozim.

Misol:

```
#include<iostream>
using namespace std;
class X {
public:
void  show(){cout<<"Hello!"<<endl;};
};
class Y: public X {};
class Z: public X {};
class D: public Y, public Z {
public:
void show(){
Y::show();
};
};
int main(){
D a;
a.show();
system("pause");
return 0;
}
```

Natija:
Hello!

Virtual sinflar. Bir xil nomdagi obyektlarni bartaraf qilishda to‘g‘ridan-to‘g‘ri ajdod sinflarni ko‘plik vorislikda virtual deb e'lon qilinadi. Buning uchun ajdod sinf sinf nomidan oldin virtual kalit so‘zini joylash lozim.

Bu holda asos sinf qancha uchrashiga qaramasdan faqat bitta asos sinfi obyekti yaratiladi. Vorislar shu obyektga ilovani o‘z ichiga oladi. Vorislikning bu usuli virtual vorislik deb ataladi.

Hosila sinf hajmi agar virtual sinflar mavjud bo‘lmasa, ularning komponentalari va vorislikka o‘tgan komponentalar hajmiga teng.

Virtual sinflar obyektlari novirtual sinflar obyektlaridan oldin yaratiladi.

Konstruktorlar chaqirilishi tartibi ham shunday. Destruktorlar konstruktorlarga teskari tartibda chaqiriladi.

Misol:

```
#include<iostream>
using namespace std;
class X {
long double a;
public:
void setx(int a1) {a=a1;}
void show(){cout<<a;};
};
class Y: virtual public X{};
class Z: virtual public X{};
class D: public Y,  public Z {};
int main() {
cout<<"sizeof(X)="<<sizeof(X)<<endl;
cout<<"sizeof(Y)="<<sizeof(Y)<<endl;
cout<<"sizeof(Z)="<<sizeof(Z)<<endl;
cout<<"sizeof(D)="<<sizeof(D)<<endl;
D d;
d.setx(1.0);
d.show();
system("pause");
return 0;
}
```

Natija:

sizeof(X)=12
sizeof(Y)=16
sizeof(Z)=16
sizeof(D)=20

Quyidagi dasturda computer sinfini yaratish uchun, **computer_screen** va **mother_board** asos sinflaridan foydalaniladi:

```
#include <iostream>
using namespace std;
class computer_screen{
public:
computer_screen(char type[], long, int, int);
void show_screen(void);
private:
char type[10];
long colors;
int x_resolution;
int y_resolution;
};
computer_screen::computer_screen(char type[], long colors, int x_res, int y_res){
```

```
strcpy(computer_screen::type,type);
computer_screen::colors = colors;
computer_screen::x_resolution = x_res;
computer_screen::y_resolution = y_res;
}
void computer_screen::show_screen(void){
cout << "Ekran turi: " << type << endl;
cout << "Rang: " << colors << endl;
cout << "Kattaligi: " << x_resolution << " ga " << y_resolution << endl;
}
class mother_board{
public:
mother_board(int, int, int);
void show_mother_board(void);
private:
int processor;
int speed;
int RAM;
};
mother_board::mother_board(int processor, int speed, int RAM){
mother_board::processor = processor;
mother_board::speed = speed;
mother_board::RAM = RAM;
}
void mother_board::show_mother_board(void){
cout << "Processor: " << processor << endl;
cout << "Chastota: " << speed << "MGC" << endl;
cout << "Operativ xotira: " << RAM << " MBayt" << endl;
}
class computer : public computer_screen, public mother_board{
public:
computer(char name[], int, float, char screen[], long, int, int, int, int, int);
void show_computer(void);
private:
char name[10];
int hard_disk;
float floppy;
};
computer::computer(char name[], int hard_disk, float floppy, char
screen[], long colors, int x_res, int y_res, int processor, int speed, int RAM):
computer_screen(screen, colors, x_res, y_res), mother_board(processor,
speed, RAM){
strcpy(computer::name,name);
computer::hard_disk = hard_disk;
computer::floppy = floppy;
```

```
}
void computer::show_computer(void){
cout << "Tur: " << name << endl;
cout << "Qattiq disk: " << hard_disk << "MBayt" << endl;
cout << "Yumshoq disk: " << floppy << "MBayt" << endl;
show_mother_board();
show_screen();
}
int main(){
computer my_pc("Compaq", 212, 1.44, "SVGA",
16000000, 640, 480, 486, 66, 8);
my_pc.show_computer();
return 0;
}
```

Bu misolda **computer** sinfi konstruktori **mother_board** va **computer_screen** konstruktorlarini chaqiriladi.

Virtual funksiyalar

Virtual funksiyalar mexanizmiga biror komponent funksiyaning har bir hosilaviy sinfda alohida varianti mavjud bo‘lish lozim bo‘lganda murojaat qilinadi. Bunday funksiyalarga ega sinflar **polimorf** sinflar deb ataladi va obyektli dasturlashda alohida o‘ringa ega.

Virtual funksiyalar kechki yoki dinamik bog‘lanish mexanizmiga asoslangandir. Asos sinf har qanday nostatik komponent funksiyasi **virtual** kalit so‘zi yordamida virtual deb e'lon qilinishi mumkin.

Kechki bog‘lanishda erta bog‘lanishga o‘xshab adreslar statik ravishda kompilyasiya jarayonida emas, balki dinamik dastur bajarilishi jarayonida aniqlanadi. Bog‘lash jarayoni virtual funksiyalarni adreslar bilan almashtirishdan iborat. Virtual funksiyalar adreslar haqida ma'lumot saqlanuvchi jadvaldan foydalanadi.

Virtuallik vorislikka o‘tadi. Funksiya virtual deb e'lon qilingandan so‘ng hosila sinfda qayta ta'rifi (shu prototip bilan) bu sinfda yangi virtual funksiyani yaratadi, bu holda virtual spetsifikatori talab qilinmaydi. Konstruktorlar virtual bo‘lolmaydi.

Misol:

```
#include <iostream>
using namespace std;
class Parent {
public:
virtual void F1() { cout<<"I am Parent"<<endl; };
void F2(int n) {
for(int i=0;i<n;i++) F1();
};
};
class Child : public Parent {
public:
void F1() { cout<<"I am Child"<<endl; }
};
int main() {
Child child;
child.F2(3);
system("pause");
return 0;
}
```

Natija:
I am Child
I am Child
I am Child

Bu misolda **Parent** sinfi F1 va F2 a'zo-funksiyalarga ega, bunda F2 ni F1 chaqiradi. **Parent** sinfining hosilasi bo‘lgan **Child** sinfiga F2 funksiyasi vorislikka o‘tadi, biroq F1 funksiyasi qayta ta'riflanadi. Agar F1 funksiya virtual bo‘lmasa, kompilyator vorislikka o‘tgan F2 funksiyani **Parent::F1** funksiyasi bilan bog‘lab translyasiya qilib yuboradi. Natijada uch marta I am Parent bosilib chiqadi. Lekin misolda F1 funksiya virtual bo‘lgani uchun uch marta I am Child bosilib chiqadi.

Virtual funksiyalar va polimorfizm

Funksiya va usullarda asos sinfga ilova ko‘rsatilib, voris obyekt adresini uzatish mumkin. Teskarisi mumkin emas.

Misol:

```
#include<iostream>
using namespace std;
class base{
```

```
public:
virtual void print(){
cout<<"base";
}
};
class dir : public base{
public:
void print(){
cout<<"\ndir"<<endl;
}
};
class sell{
public:
void print(base& a){a.print();}
};
int main(){
base B;
dir D;
sell U;
U.print(B); // base
U.print(D); // dir
system("pause");
return 0;
}
```

Natija:
base
dir

Bu misolda avlod sinfi obyektiga adres qiymat sifatida berilgan ajdod sinf turidagi ilova orqali avlodda qo‘shimcha yuklangan usulni chaqirishga e'tibor berish lozim. Agar funksiya novirtual bo‘lsa ajdod sinf usuli, virtual bo‘lsa avlod sinf usuli chaqiriladi.

Polimorf obyekt-telefonning yaratilishi. Aytaylik, sizning boshliqlaringiz sizga obyekt-telefoningiz diskli, tugmachali yoki to‘lovli telefonlardan birini tanlab olib, emulyasiya qila olishi kerak dedi. Boshqacha qilib aytganda, obyekt-telefon bitta qo‘ng‘iroq uchun tugmachali apparat sifatida, boshqa qo‘ng‘iroq uchun to‘lovli telefon sifatida va h.k. ishlashi mumkin edi. Ya'ni bir qo‘ng‘iroqdan ikkinchisiga sizning obyekt-telefoningiz o‘z shaklini o‘zgartirishi lozim bo‘ladi.

Turli sinflarga mansub bu telefonlarda faqat bitta farqlanuvchi funksiya mavjud – bu dial usuli. Polimorf obyektni yaratish uchun, siz avval bazaviy sinf funksiyalarini, ularning prototiplari oldidan virtual kalit-so‘zni qo‘ygan holda, aniqlaysiz. Bu bazaviy sinf funksiyalari hosila sinflar funksiyalaridan shuning bilan farqlanadiki, ular *virtual*dirlar.

Keyin dasturda parametri bazaviy sinf obyektiga ilova global funksiya tuziladi. Funksiya tanasda dial usuliga murojaat qilinadi.

Obyekt shaklini o‘zgartirish uchun, siz, quyida ko‘rsatilganidek, ushbu funksiyaga hosila sinf obyektini parametr sifatida uzatasiz.

Funksiyada kelgan (**phone&**) belgisi turlarga keltirishga imkon berib, bir turdagi o‘zgaruvchi (**touch_tone**) adresini boshqa turdagi o‘zgaruvchi (**phone**)ga berish zarurligini ma'lum qiladi. Dastur **dial_phone** funksiyasiga turli obyektlar adresini taqdim qilishi mumkin ekan, demak, funksiya polimorf bo‘lishi mumkin. Navbatdagi dasturda bu usuldan obyekt-telefon yaratish uchun foydalanadi. Dastur ishga tushirilgach, **poly_phone** obyekti o‘z shaklini diskli telefondan tugmachalisiga, keyin esa to‘lovlisiga o‘zgartiradi:

```
#include <iostream>
using namespace std;
class phone{
public:
virtual void dial(char number[]) {
cout << "Raqam to'plami " << number << endl;
}
void answer(void) { cout << "Javobni kutish" << endl; }
void hangup(void) { cout << "Qo'ng'iroq bajarildi-trubkani qo'yish"
<< endl;}
void ring(void) { cout << "Qo'ng'iroq, qo'ng'iroq, qo'ng'iroq" << endl;
}
phone(char number[]) { strcpy(phone::number,number); };
protected:
char number[20];
};
class touch_tone : public phone{
public:
void dial(char number[]) { cout << "Pik Pik Raqam to'plami " <<
 number << endl; }
touch_tone(char number[]) : phone(number) { }
```

```
};
class pay_phone: public phone{
public:
void dial(char number[]) { cout << "Iltimos to'lang " << amount << "
sent" << endl;
cout << "Raqam to'plami " << number << endl; }
pay_phone(char number[], int amount) : phone(number) {
pay_phone::amount = amount; }
private:
int amount;
};
void dial_phone(phone& this_phone,char this_number[]){
this_phone.dial(this_number);
};
int main(){
pay_phone city_phone("702-555-1212", 25);
touch_tone home_phone("555-1212");
phone rotary("201-555-1212") ;
dial_phone(rotary,"818-555-1212");
dial_phone(home_phone,"303-555-1212");
dial_phone(city_phone,"212-555-1212");
return 0;
}
```

Agar ushbu dastur kompilyasiya qilinib ishga tushirilsa, displey ekranida quyidagi yozuv paydo bo‘ladi:

Raqam to‘plami 818-555-1212
Pik Pik Raqam to‘plami 303-555-1212
Iltimos to‘lang 25 sent
Raqam to‘plami 212-555-1212

Abstrakt sinflar

Hech bo‘lmasa bitta sof (bo‘sh) virtual funksiyaga ega bo‘lgan sinf abstrakt sinf deyiladi. Quyidagi tavsifga ega bo‘lgan komponentali funksiya sof virtual funksiya deyiladi:

virtual <tur> <funksiya_nomi>(<formal_parametrlar_ro‘yxati>) = 0;

Abstrakt sinf hosila sinf uchun asosiy (bazaviy) sinf sifatida ishlatilishi mumkin. Abstrakt sinflarning mexanizmi keyinchalik aniqlanadigan umumiy tushunchalarni tavsiflash uchun ishlab chiqilgan. Bu holda, sinflar ierarxiyasini yaratish quyidagi sxema bo‘yicha bajariladi: ierarxiya asosida abstrakt bazaviy

sinf turadi. U interfeysni meros qilib olish uchun foydalaniladi. Hosila sinflar bu interfeysni aniqlashtiradi va amalga oshiradi. Abstrakt sinfda sof virtual funksiyalar e'lon qilingan, ular aslida **abstrakt usullar.**

Ba'zi sinflar masalan **shape** sinfi, abstrakt tushunchalarni ifodalaydi va ular uchun obyekt yaratib bo'lmaydi.

Agar sof virtual funksiya hosila sinfda to'liq ta'riflanmasa, u hosila sinfda ham sof virtual bo'lib qoladi, natijada hosila sinf ham abstrakt sinf bo'ladi.

Masalan, operatsion tizimda qurilma drayveri abstrakt sinf sifatida berilishi mumkin. Drayverlar **character_device** abstrakt sinfining ajdodlari sifatida kiritilishi mumkin.

Misol:

```
#include<iostream>
using namespace std;
class Abstract {
public:
virtual void draw() = 0;
};
class Derived : public Abstract {
public:
void draw() { cout << "Hello!"<<endl;}
};
int main( ) {
Derived d;
d.draw();
system("pause");
return 0;
}
```

Natija:
Hello!

Abstrakt sinflar va ko'rsatkichlar

Abstrakt sinf turidagi o'zgaruvchi yaratib bo'lmaydi, lekin abstrakt sinf turidagi ko'rsatkich yoki ilova yaratish mumkin. Bu ko'rsatkich yoki ilovaga abstrakt sinf abstrakt bo'lmagan turli avlodlari adresini qiymat sifatida berib, abstrakt usulga mos turli usullarni chaqirish mumkin. Agar abstrakt sinf turidagi

ko‘rsatkichlar massividan foydalanilsa universal funksiyalar yoki usullar yaratish mumkin.

Masalan, yuqoridagi polimorf telefon quyidagicha ta'riflanish mumkin:

```
#include <iostream>
using namespace std;
class phone{
public:
virtual void dial(char number[])=0;
void answer(void) { cout << "Javobni kutish" << endl; }
void hangup(void) { cout << "Qo'ng'iroq bajarildi-trubkani qo'yish" <<
endl; }
void ring(void) { cout << "Qo'ng'iroq, qo'ng'iroq, qo'ng'iroq" << endl; }
phone(char number[]) { strcpy(phone::number,number); };
protected:
char number[20];
};
class touch_tone : public phone{
public:
void dial(char number[]) { cout << "Pik Pik Raqam to'plami " <<
number << endl; }
touch_tone(char number[]) : phone(number) { }
};
class pay_phone: public phone{
public:
void dial(char number[]) { cout << "Iltimos to'lang " << amount << "
sent" << endl;
cout << "Raqam to'plami " << number << endl; }
pay_phone(char number[, int amount) : phone(number) {
pay_phone::amount = amount; }
private:
int amount;
};
void dial_phone(phone& this_phone,char this_number[]){
this_phone.dial(this_number);
};
int main(){
pay_phone city_phone("702-555-1212", 25);
touch_tone home_phone("555-1212");
dial_phone(home_phone,"303-555-1212");
dial_phone(city_phone,"212-555-1212");
return 0;
}
```

Nazorat savollari:

1. Vorislik nima uchun kerak?
2. Himoyalangan (protected) va xususiy (private) huquqlari orasida qanday farq bor?
3. Nima uchun avval ajdod sinf konstruktorlari chaqirilib, so‘ngra avlod sinf konstruktori chaqiriladi?
4. Nima uchun destruktorlar konstruktorlarga nisbatan teskari tartibda chaqiriladi?
5. Vorislikda ajdod sinf spetsifikatori sifatida protected ko‘rsatilishi mumkinmi?
6. Sinflar kutubxonasini qurishda vorislikdan qanday foydalaniladi?
7. Xususiy deb e'lon qilingan komponentalarga boshqa sinf usullari orqali murojaat qilish mumkinmi?
8. Polimorfizm deb nimaga aytiladi?
9. Usulni virtual deb e'lon qilish qanday imkon yaratadi?
10. Abstrakt sinf deb qanday sinfga aytiladi?

Misollar:

1. Vorislikdan foydalanib Artist va Sirk_artisti sinfini yarating.

2. Vorislikdan foydalanib Nuqta, To‘rtburchak va To‘g‘ri_to‘rtburchak sinflarini yarating.

3. Vorislik va abstrakt sinfdan foydalanib detal, radiodetal, kondensator sinfini yarating.

4. Vorislikdan va abstrakt sinfdan foydalanib sinov, imtihon, bitiruv imtihoni sinflarini yarating.

5. Vorislikdan va abstrakt sinfdan foydalanib nashr, kitob, o‘quv qo‘llanma sinflarini yarating.

Polimorf sinflar

Qo‘shimcha yuklash ta'rifi

Standart amallarni (masalan +) qo‘shimcha yuklash orqali biror sinf bilan birga qo‘llashda mazmunini o‘zgartirishdan iboratdir.

Standart amallarni qo‘shimcha yuklash maxsus funksiya-komponenta kiritish yo‘li bilan amalga oshiriladi. Qo‘shimcha yuklash til standartiga asosan amalga oshiriladi, amallar belgisi va operandlar soni o‘zgarmaydi.

Amallarni qo‘shimcha yuklash uchun quyidagi ta'rifdan foydalaniladi:

<operator amal> (<operandlar ro‘yxati>)

quyidagi amallarni qo‘shimcha yuklash mumkin:

\+ - * / % ^ & | ~ !

= < > += -= *= /= %= ^= &=

|= << >> >>= <<= == != <= >= &&

|| ++ -- [] () new delete

Bu amallar ustuvorligi va ifodalar sintaksisini o‘zgartirish mumkin emas. Masalan, unar amal % yoki binar ! amalni kiritish mumkin emas. Funksiya amal har qanday funksiya kabi ta'riflanadi va chaqiriladi.

Standart turlar uchun to‘rt amal ("+", "-", "*" i "&") ham unar ham binar amal sifatida ishlatiladi va qo‘shimcha yuklanadi.

Hamma qo‘shimcha yuklangan amallar uchun operator() amalidan tashqari, ko‘zda tutilgan argumentlardan foydalanish mumkin emas.

Amallar xossalaridan ba'zilaridan foydalaniladi. Xususan, operator=, operator[], operator() va operator-> nostatik komponenta – funksiya bo‘lishi lozim.

Operator – funksiya yoki sinf komponentasi bo‘lishi kerak yoki juda bo‘lmasa, bitta parametri sinf obyekti bo‘lishi kerak (**new** va **delete** amallarini qo‘shimcha yuklovchi funksiyalar uchun bu shart emas).

Operator - funksiya, birinchi parametri asosiy turga tegishli bo‘lsa, funksiya-komponenta bo‘lolmaydi.

C++ tilida quyidagi amallarni qo‘shimcha yuklash mumkin emas:

. sinf obyekti a'zosiga murojaat;

.* ko‘rsatkich orqali murojaat;

?: shartli amal;

:: ko‘rinish soxasini ko‘rsatuvchi amal;

sizeof hajmni hisoblash amali;

preprotsessor amali.

Binar amallarni qo‘shimcha yuklash

Ixtiyoriy ⊕ binar amali ikkita usulda tavsiflanishi mumkin: yoki bitta parametrli funksiya komponentasi sifatida, yoki ikkita parametrli global (do‘stona bo‘lishi mumkin) funksiya sifatida.

Birinchi holatda **x⊕y** ifoda **x.operator⊕(y),** ikkinchi holda esa **operator ⊕(x,y)** chaqirilishini bildiradi.

Aniq sinf doirasida qayta yuklangan amallar faqat parametrli nostatik komponentli funksiya orqali qayta yuklanadi. Sinfdagi chaqiriladigan obyekt avtomatik ravishda birinchi operand sifatida qabul qilinadi.

Sinfdan tashqari yuklanadigan amallar ikkita operandga ega bo‘lishi kerak. Ulardan biri sinf turiga ega bo‘lishi lozim.

Qiymat qaytarish uchun vaqtinchalik o‘zgaruvchidan foydalaniladi. Bu maqsadda konstruktordan va **this** ko‘rsatkichidan foydalanish mumkin.

Bitta amalni bir necha marta qo‘shimcha yuklashdan qutilish uchun, bitta parametrli konstruktordan foydalanish mumkin.

Ikkala konstruktor o‘rniga ko‘zda tutilgan qiymatga ega bitta konstruktordan foydalanish mumkin.

Unar amallarni qo‘shimcha yuklash

Ixtiyoriy ⊕ unar amali ikkita usulda tavsiflanishi mumkin: yoki paramatrsizli funksiya komponentasi sifatida, yoki bitta parametrli global (do‘stona bo‘lishi mumkin) funksiya sifatida. Birinchi holatda **⊕ Z** ifoda **Z.operator ⊕ ()**, ikkinchi holatda esa operator **⊕(Z)** chaqirilishini bildiradi.

Aniq sinf doirasida qayta yuklangan unar amallar faqat parametrsizli nostatik komponentli funksiya orqali qayta yuklanadi. Sinfdagi chaqiriladigan obyekt avtomatik ravishda operand sifatida qabul qilinadi.

Sinf doirasidan tashqarida qayta yuklangan unar amallar (global funksiya kabi) sinf turdagi bitta parametrga ega bo‘lishi lozim. Shu parametr orqali uzatiladigan obyekt operand sifatida qabul qilinadi.

Sintaksis:

a) birinchi holda (sinf doirasida tavsiflash):

<qaytariluvchi_qiymat_turi> operator <amal_belgisi>

b) ikkinchi holda (sinf doirasidan tashqari tavsiflash):

<qaytariluvchi_qiymat_turi> operator <amal_belgisi>

(<tur_identifikatori>)

Misol:

```
#include<iostream>
using namespace std;
class complex {
double re, im;
public:
complex(){};
complex(double r, double i=0) { re=r; im=i; }
//
double real(){return re;}
double imag(){return im;}
//
complex operator+();
complex operator-();
complex operator+(complex);
complex operator-(complex);
complex operator*(complex);
complex operator/(complex);
//
complex& operator+=(complex);
complex& operator-=(complex);
complex& operator*=(complex);
complex& operator/=(complex);
//
bool operator==(complex);
bool operator!=(complex);
```

```
};
//
inline complex complex::operator+(){
return *this;
}
inline complex complex::operator-(){
return complex(-re, -im);
}
inline complex complex::operator+(complex b){
return complex(re+b.re, im+b.im);
}
inline complex complex::operator-(complex b){
return complex(re+b.re, im+b.im);
}
inline complex complex::operator*(complex b){
return complex(re*b.re-im*b.im, re*b.im+im*b.re);
}
inline complex complex::operator/(complex b){
double t=b.re*b.re+b.im*b.im;
return complex((re*b.re+im*b.im)/t,(im*b.re-re*b.im)/t);
}
inline complex& complex::operator+=(complex b){
re+=b.re; im+=b.im;
return *this;
}
inline complex& complex::operator-=(complex b){
re-=b.re; im-=b.im;
return *this;
}
inline complex& complex::operator*=(complex b){
*this=*this*b;
return *this;
}
inline complex& complex::operator/=(complex b){
*this=*this/b;
return *this;
}
inline bool complex::operator==(complex b){
return (re==b.re)&&(im==b.im);
}
inline bool complex::operator!=(complex b){
return (re!=b.re)||(im!=b.im);
}
int main() {
complex a(1,1);
```

```
complex b(2,2);
complex c = a + b;
cout<<"re="<<c.real()<<" im="<<c.imag()<<endl;
complex d = a.operator+(b);
cout<<"re="<<d.real()<<" im="<<d.imag()<<endl;
d+=b;
cout<<"re="<<d.real()<<" im="<<d.imag()<<endl;
system("pause");
return 0;
}
```

Natija:

re=3 im=3
re=3 im=3
re=5 im=5

Global funksiya sifatida qo‘shimcha yuklash. Global funksiya sifatida qo‘shimcha yuklanuvchi amallar ikki operandga ega bo‘lishi va operandlardan biri sinf turiga tegishli bo‘lishi kerak.

Amallarni do‘stona funksiya sifatida qo‘shimcha yuklash ko‘pincha chap va o‘ng operandlar simmetriyasini ta'minlash uchun ishlatiladi.

Agar argumentlar ilova bo‘yicha uzatilsa, amallar sinf a'zolarni sifatida qo‘shimcha yuklanishi lozim. Aks holda murakkab ifodalardan foydalanish mumkin bo‘lmay qoladi.

Misol:

```
#include<iostream>
using namespace std;
class complex {
double re,im;
public:
complex(){};
complex(double r, double i=0) { re=r; im=i; }
//
double real(){return re;}
double imag(){return im;}
//
friend complex operator+(complex b){
return b;
};
friend complex operator-(complex b){
return complex(-b.re, -b.im);
```

```
};
friend complex operator+(complex a, complex b){
return complex(a.re+b.re, a.im+b.im);
}
friend complex operator-(complex a, complex b){
return complex(a.re-b.re, a.im-b.im);
}
friend complex operator*(complex a,complex b){
return complex(a.re*b.re-a.im*b.im, a.re*b.im+a.im*b.re);
}
friend complex operator/(complex a,complex b){
double t=b.re*b.re+b.im*b.im;
return complex((a.re*b.re+a.im*b.im)/t,(a.im*b.re-a.re*b.im)/t);
}
//
friend complex& operator+=(complex a,complex b){
a.re+=b.re; a.im+=b.im;
return a;
}
friend complex& operator-=(complex a,complex b){
a.re-=b.re; a.im-=b.im;
return a;
};
friend complex& operator*=(complex a,complex b){
a=a*b;
return a;
}
friend complex& operator/=(complex a,complex b){
a=a/b;
return a;
};
//
friend bool operator==(complex a,complex b){
return (a.re==b.re)&&(a.im==b.im);
};
friend bool operator!=(complex a,complex b){
return (a.re!=b.re)||(a.im!=b.im);
};
};
int main() {
complex a(1,1);
complex b(2,2);
complex c = a + b;
cout<<"re="<<c.real()<<" im="<<c.imag()<<endl;
complex d = operator+(a,b);
```

```
cout<<"re="<<d.real()<<" im="<<d.imag()<<endl;
d+=b;
cout<<"re="<<d.real()<<" im="<<d.imag()<<endl;
system("pause");
return 0;
}
```

Natija:

re=3 im=3
re=3 im=3
re=5 im=5

Inkrement va dekrement amallarini qo‘shimcha yuklash

C++ tilining zamonaviy versiyalarida prefiks ++ va -- amallarni qo‘shimcha yuklash boshqa amallarni yuklashdan farq qilmaydi. Postfiks shakldagi ++ va -- amallarini qayta yuklaganda yana bir **int** turidagi parametr kiritilishi kerak. Agar qo‘shimcha yuklash uchun global funksiya ishlatilsa, uning birinchi parametri sinf turiga, ikkkinchi parametri **int** turiga ega bo‘lishi kerak.

Dasturda postfiks ifoda ishlatilganda butun parametr ham qiymatga ega bo‘ladi.

Quyida postfiks va prefiks ++ va – amallari uchun funksiya-amallarga misollar keltirilgan:

```
#include<iostream>
using namespace std;
class Mini{
int val;
public:
Mini(int i=0) { val=i; }
//
int value(){return val;}
//
Mini& operator++(){
val++;
return *this;
};
Mini& operator++(int k){
++val;
return *this;
};
//
friend Mini& operator--(Mini a){
```

```
a.val--;
return a;
};
friend Mini& operator--(Mini a,int k){
--a.val;
return a;
};
//
};
//
int main() {
system("pause");
return 0;
}
```

Turlarni o‘zgartirish operatori. Yuqorida konstruktor yordamida standart turdagi, ya'ni haqiqiy o‘zgaruvchini sinf turidagi (kompleks turidagi o‘zgaruvchiga) keltirishni ko‘rdik. Teskarisini amalga oshirish uchun turni o‘zgartirish operatoridan foydalanish mumkin. Operator umumiy ko‘rinishi quyidagicha:

operator type()

Bu yerda **type** standart tur, sinf nomi yoki **typedef** orqali kiritilgan nom bo‘lishi mumkin. Lekin **type** massiv yoki funksiya bo‘lishi mumkin emas. Tur operatori funksiya-a'zo bo‘lishi lozim. Uning e'lonida qaytaruvchi qiymat turi va parametrlar ro‘yxati ko‘rsatilmaydi.

Operator turlar oshkor o‘zgartirilganda chaqiriladi. Agar o‘zgartiralayotgan qiymat sinf obyekti bo‘lsa va sinfda mos tur o‘zgartirish operatori mavjud bo‘lsa, shu operator chaqiriladi.

Agar maqsadli tur operator turiga aniq kelmasa ham, maqsadli turdan operator turiga keltiruvchi standart tur o‘zgartirishlari mavjud bo‘lsa operator chaqiriladi. Agar operator turiga keltirish uchun juda bo‘lmasa bitta foydalanuvchi kiritgan tur o‘zgartirish lozim bo‘lsa, operator chaqirilmaydi.

C++ standart kutubxonassiga kiruvchi sinflarda tur o‘zgartirish operatori ishlatilmaydi.

Quyidagi misolda ratsional sinfi kiritilgan va bu sinfda ratsional sonni haqiqiy songa keltirish operatori aniqlangan:

Misol:

```
#include <iostream>
using namespace std;
class Fraction{
int s,m;
public:
Fraction(int s1,int m1=1){s=s1;m=m1;};
void show(){cout<<s<<'/'<<m<<endl;};
operator double(){
return (double)s/m;
}
};
int main ( ){
Fraction D(1,2);
D.show();
double c=D;
cout<<c<<endl;
D=1;
D.show();
return 0;
}
```

Sinflar shablonlari

Sinf shablonlari (o‘zgacha parametrlangan sinf) avlodga oid sinfni tuzish uchun ishlatiladi. Tuzish qoidalarni va ayrim obyektlarni formatini aniqlovchi sinf kabi, sinf shablonlari ayrim sinflarni tuzish usullarini aniqlaydi. Shablondagi sinf ta'rifida sinf ismi ayrim sinflarning emas, oilaviy sinflarning parametrlangan ismi bo‘ladi.

Parametrlangan sinfni e'lon qilishning umumiy shakli:

template <class tur_dannqx> class imya_sinfa { . . . };

Sinf shablonlarning asosiy xossalari

*Parametrlangan sinf funksiyalarining komponentalari avtomatik ravishda parametrlangan bo‘ladi. Ularni **template** yordamida parametrlangan sifatida e’lon qilish shart emas.

*Parametrlangan sinfda tavsiflangan do‘stona funksiyalar avtomatik ravishda parametrlangan funksiyalar bo‘lmaydi, ya'ni ko‘rsatilmagan holda shunday funksiyalar berilgan shablon bo‘yicha tashkil etilgan barcha sinflar uchun do‘stona bo‘ladi.

*Agarda **friend**-funksiya o‘z tavsifida parametrlangan sinf turdagi parametrga ega bo‘lsa, unda berilgan shablon bo‘yicha yaratilgan barcha sinflar uchun xususiy **friend**–funksiyasi mavjud.

*Parametrlangan sinf doirasida **friend**-shablonlarni (do‘stona parametrlangan sinflar) tavsiflash mumkin emas.

*Bir tarafdan, shablonlar shablonlardan hosil (meros) bo‘lgandek, oddiy sinflardan ham hosil (meros) bo‘lishi mumkin. Ikkinchi tarafdan esa ulardan boshqa shablonlar va sinflar uchun bazoviy sifatida foydalanishi mumkin.

*Sinf a'zosi bo‘lgan funksiyalar shablonlarini **virtual** sifatida tavsiflash mumkin emas.

*Lokal sinflar o‘z elementlari sifatida shablonlarni o‘z ichiga olishi mumkin emas.

Parametrlangan sinflarning komponent funksiyalari

Sinf shablonining tavsifidan tashqarida joylashgan sinf shablonining komponentli funksiyasini amalda oshirishda quyidagi ikkita elementni qo‘shimcha kiritish lozim:

*Tavsiflash **template** kalitli so‘zdan boshlanishi lozim, undan so‘ng burchakli qavslarda sinf shablonni tavsifida ko‘rsatilgan **turlar_parametrlarining_ro`yxati** keladi.

* Ko‘rish sohasi amalidan (**::**) oldinda bo‘lgan sinf ismidan so‘ng shablonning **parametrlar_ismlari_ro‘yxati** kelishi lozim.

template <turlar_ro‘yxati> <qaytariluvchi_qiymat_turi> <sinf_nomi>
<parametrlar_nomlari_ro‘yxati > : : <funksiya_nomi>
(<parametrlar_ro‘yxati>) { . . . }

Sinf obyektlari bilan ishlash uchun **vector** qo‘shimcha yuklangan shablon sinfi:

```
#include <iostream>
using namespace std;
template<class T > class vector {
T data[200];
int size;
public:
vector (int k) {size =k; };
T&  operator[](int i){return data[i];};
int getsize() { return size; };
void input();
void show();
};
template<class T > void vector <T >:: input() {
for (int i = 0; i < size; i++) {cin>>data[i] ; cout << ' ';}
}
template<class T > void vector <T >:: show () {
for (int i = 0; i < size; i++) cout << data[i] << ' ';
}
int main(){
vector<int> a(5);
a.input();
a.show();
return 0;
}
```

Instansionirlash va spetsializatsiya. Shablonlardan ularning parametrlari o‘rniga haqiqiy qiymatlarni qo‘yish yo‘li bilan oddiy sinflar, funksiyalar va funksiya-a'zolarni hosil qilish jarayoni shablonni instansionirlash deb ataladi. Shablonni instansionirlash natijasida olingan mohiyat (sinf, funksiya, funksiya-a'zo) umumiy holda spetsializatsiya deb ataladi.

Lekin C++ tilida instansionirlash jarayoni spetsializatsiya olish yagona usuli emas.

Dasturchiga shablon parametrlarga konkret turlar qo‘yish oshkor ko‘rsatish imkoni mavjuddir. Bunday spetsializatsiya **template<>** konstruksiyasi yordamida kiritiladi.

Bu oshkor spetsializatsiyadir (**explicit specialization**) (instansionirlanuvchi yoki generatsiya qilinuvchi spetsializatsiyadan farqli (**instantiated specialization, generated specialization**)).

Shablon parametrlari saqlanib qolgan spetsializatsiya, qisman spetsializatsiya deyiladi (**partial specialization**).

Agar spetsializatsiya (oshkor yoki xususiy) haqida so‘z borsa, umumiy shablon birlamchi yoki asosiy shablon (**primary template**) deyiladi.

Misol:

```
template <typename T>
class Alpha{
T k;
public:
Alpha(T k1){k=k1;}
T show() {
cout<<k<<endl;
}
};
template<>
class Alpha<int>{
int k;
public:
Alpha(int k1){k=k1;}
int show() {
cout<<k<<endl;
}
};
int main(){
Alpha<int> a(5);
Alpha<double> b(2.2);
a.show();
b.show();
system("pause");
return 0;
}
```

Natija:

5

2.2

Yuqori darajali funksiyalar

Funksiya obyektlari – bu «kichik qavs» () amali aniqlangan sinf nusxasi. Ba'zi bir holatlarda funksiyani obyekt – funksiyalarga almashtirish qulaydir. Obyekt - funksiya funksiya sifatida ishlatilsa, uni chaqirish uchun **operator ()** dan foydalaniladi.

Yuqori darajadagi algoritm bu shunday algoritm-ki, bitta yoki bir nechta argumentlar funksional turga tegishlidir.

Dixotomiya usuli yordamida ixtiyoriy funksiya uchun [a,b] oraliqda f(x)=0 tenglamani yechish masalasi misolida yuqori darajali funksiyani ko‘rib o‘tamiz. Bu maqsadda masalani yechadigan metodni tavsiflovchi sinf yaratamiz. Sinfni tavsiflovchi dastur kodini ko‘ramiz:

```
#include <iostream>
using namespace std;
template <class T>
class Funk{
public:
virtual T operator()(T a)=0;
};
template <class T>
class FunctionZero{
public:
static T dihotom(T a, T b, T eps, Funk<T>& f);
};
template <class T>
T FunctionZero<T>::dihotom(T a, T b, T eps, Funk<T>& f) {
T x, x1=a, x2=b;
while ((x2-x1)>eps) {
x=(x1+x2)/2;
if (f(x)==0) return x;
if (f(x)>0) x2=x; else x1=x;
};
return x;
};
class Lin:public Funk<double>{
public:
double operator()(double x){return 3*x;}
};
int main(){
```

```
Lin f;
double z=FunctionZero<double>::dihotom(-1,2,0.1,f);
cout<<z;
return 0;
}
```

Usulni umumiy turga umumlashtirishda turni funksiya sinfiga uzatish muammosi tug‘iladi. Bu muammoni hal qilish yo‘llaridan biri – funksiya uchun abstrakt sinf shablonidan foydalanish.

Ko‘rsatkich this. Ajdod sinfga ega sinf shablonlari uchun x nomidan foydalanish, hatto x vorislikka o‘tgan bo‘lsa ham, **this->x** murojaatga ekvivalent emas.

Umuman olganda, ajdod sinfda ta'rifi keltirilgan va shablon parametriga bog‘liq nomga murojaat qilish uchun to‘liq kvalifikatsion nom ko‘rsatilishi lozim. Buning uchun **this->** yoki **Base<T>::** konstruksiyadan foydalanish lozim.

Shablonlarda har qanday noaniqlik oldini olish uchun, a'zolarga murojaat qilganda to‘liq nomdan foydalanish ma'quldir.

Sinf shabloni statik elementlari. Sinf shablonida statik elementlar-a'zolar ta'riflanishi mumkin. Har bir konkretlashgan nusxa o‘z a'zolariga ega bo‘ladi. Statik elementlar faqat foydalanilganda konkretlashadi. Bunday a'zo o‘zi ham shablondir. Uning uchun shablon ta'rifi xotira ajratilishiga olib kelmaydi: xotira faqat konkretlashgan statik element nusxasi uchun ajratiladi. Shunday qilib, statik elementga murojaat konkretlashgan sinf orqali bajariladi.

Misol:

```
#include <iostream>
using namespace std;
template <typename T>
class Base {
public:
void print(){
cout<<"Base"<<endl;
};
};
template <typename T>
class Derived : Base<T> {
public:
```

```
void show() {
this->print();
}
};
int main(){
Derived<int> D;
D.show();
system("pause");
return 0;
}
```
Natija:
Base

Sinflar shabloni xossalari. Sinflar shabloni quyidagi qo‘shimcha xossalarga ega:

Shablon uchun **typedef** operatori yordamida yangi tur kiritish mumkin.

Bu yangi tur yordamida o‘zgaruvchilar ta'rifi berilishi mumkin.

Shablon parametri ko‘zda tutilgan qiymatga ega bo‘lishi mumkin (bu parametr-turlar uchun va parametr-konstantalar uchun o‘rinli). Bunday argument sifatida ko‘pchilik konkretlash uchun mos tur yoki qiymat tanlash lozim. Xuddi funksiyalar kabi biror parametr uchun shunday argument berilgan bo‘lsa, undan keyingi hamma parametrlar uchun shu kabi argumentlar berilishi lozim (hatto shablon boshqa e'lonida).

Shablon parametri standart turdagi o‘zgaruvchi bo‘lishi mumkin. U shuni ko‘rsatadi-ki, undan keyingi nom – shablon ta'rifida konstanta sifatida ishlatiluvchi potensial qiymatdir.

Bu o‘zgaruvchi konstruktorda massiv o‘lchamini berish uchun ishlatilishi mumkin.

Sinf shabloni parametri tur bo‘lishi shart emas. Bu parametrlar o‘rniga qo‘yiladigan argumentlarga cheklanishlar qo‘yiladi. Parametr bog`langan ifoda konstantali, ya'ni kompilyasiya jarayonida hisoblanadigan bo‘lishi lozim.

Bu uchchala holat quyidagi misolda berilgan:

```
#include <iostream>
using namespace std;
template<class T, class T1=int, int c=0>
```

```
class Trio {
public:
Trio(T a1,T1 b1){a=a1;b=b1;};
void show(){cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;}
private:
T a;
T1 b;
};
int main(){
Trio<int,float,2> a(1,2.5);
a.show();
Trio<int> b(1,2);
b.show();
typedef Trio<int> IntTrio;
IntTrio c(1,1);
c.show();
system("pause");
return 0;
}
```

Natija:

a=1 b=2.5 c=2
a=1 b=2 c=0
a=1 b=1 c=0

Nazorat savollari:

1. Postfiks va prefiks amallar orasida qanday farq bor?
2. Qo‘shimcha yuklangan amallar qanday ikki usulda aniqlanadi?
3. Global do‘stona funksiya yordamida hamma amallarni qo‘shimcha yuklash mumkinmi?
4. Qaysi holatda amalni global funksiya yordamida qo‘shimcha yuklash mumkin?
5. Funksiya operatorda "sinf" yoki "sinfga ilova" turidagi parametr ishlatish shartmi?
6. Unar va binar amal-funksiyalar sintaksisi farqi nimadan iborat?
7. Shablonlarlardan nima maqsadda foydalaniladi?
8. Parametrlashtirilgan sinflar xossalarini ko‘rsating.
9. Parametrlashtirilgan sinflar hamma komponenta funksiyalari parametrlashganmi?

10.Sinf shabloni tashqarisida komponenta funksiyalar qanday aniqlanadi?

Misollar:

1. Qo‘shimcha yuklangan amallarga ega vaqt intervali sinfini yarating va dasturda qo‘llang.

2. Qo‘shimcha yuklangan amallarga ega ratsional son sinfini yarating va dasturda qo‘llang.

3. Dekart koordinatalarda berilgan vektor sinfi shablonini yarating. Vektorlar ustida amallar ta'rifini kiriting va dasturda qo‘llang.

4. Stek sinfi shablonini yarating.

5. Navbat sinfi shablonini yarating.

Oqimli sinflar

Oqimli sinflar ierarxiyasi

C++da oqimli sinflar kutubxonasi ikkita bazaviy **ios** va **streambuf** sinflar asosida tuzilgan. **streambuf** sinfi kiritish-chiqarish fizik qurilmalari bilan xotirada joylashgan kiritish-chiqarish buferlarni o‘zaro bo‘g‘lanishini va tashqilini ta'minlaydi. **streambuf** sinfining metodlarini va ma'lumotlarini dasturchi ochiq ishlatmaydi. Mavjud bo‘lgan sinflar asosida yangi sinflarni yaratishda dasturchiga ham sinfga murojaat etish ruxat etilgan.

ios sinfi – formal kiritish chiqarish va xatolarni tekshirish vositalarga ega.

Standart oqimlar (**istream**, **ostream**, **iostream**) terminal bilan ishlash uchun xizmat qiladi.

Satrli oqimlar (**istrstream**, **ostrstream**, **strstream**) xotirada joylashtirilgan satrli buferlardan kiritish-chiqarish uchun xizmat qiladi.

Faylli oqimlar (**ifstream**, **ofstream**, **fstream**) fayllar bilan ishlash uchun xizmat qiladi.

Oqimli sinflar, ularning metodlari va ma'lumotlari dasturda murojaat etish ruxsatiga ega bo‘ladi, qachon-ki, unga kerakli bosh fayl kiritilgan bo‘lsa.

iostream.h – **ios**, **ostream**, **istream** uchun.

strstream.h – **strstream**, **istrstream**, **ostrstream** uchun

fstream.h – **fstream**, **ifstream**, **ofstream** uchun

Quyidagi obyekt-oqimlar dasturda **main** funksiyasini chaqirish oldidan avvaldan aniqlangan va ochilgan bo‘ladi:

extern istream cin; //Klaviaturadan kiritish standart oqimi

extern ostream cout; //Ekranga chiqarish standart oqimi

extern ostream cerr; //Xatolar haqidagi xabar chiqarish standart oqimi

Oqimli sinflar usullari

Oqimdan qiritish uchun **istream** sinfdagi obyektlar ishlatiladi, oqimga chiqarish uchun – **ostream** sinfdagi obyektlar.

istream sinfda quyidagi funksiyalar tavsiflangan:

- **istream get (char& S);**

istream dan S ga simvolni o‘qiydi. Xato holatida **S 0XFF** qiymatini oladi.

- **int get();**

istream dan keyingi simvolni chiqaradi. Faylni oxirini aniqlagach, **EOF**ni qaytaradi.

- **istream& get(char* buffer,int size,char delimiter='\n');**

Bu funksiya **istream**dan simvollarni chiqaradi va ularni buferga nusxalaydi. Amal yoki faylning oxiriga yetganda, yoki **size** fayllardan nusxa olgan jarayonda, yoki ko‘rsatilgan ajratuvchini aniqlaganda to‘xtaydi. Ajratuvchi esa nusxalanmaydi va **streambuf** qoladi. O‘qib bo`lingan simvollar ketma-ketligi har doim **null** simvol bilan tugatiladi.

- **istream& getline(char* buffer,int size, char delimiter='\n');**

Ajratuvchi oqimdan chiqariladi, lekin, buferga kiritilmaydi. Bu esa satrlarni oqimdan chiqaruvchi asosiy funksiya. O‘qib chiqilgan simvollar **null** simvoli bilan tamomlanadi.

- **istream& read(char* buffer,int size);**

Ajratuvchilar qo‘llanmaydi va buferga o‘qilgan simvollar **null** simvoli bilan tugamaydi.

- **int peek();**

istream dan simvolni chiqarmasdan **istream**ga qaytaradi.

- **int gcount();**

Formatlanmagan oxirgi kiritish amali vaqtida o‘qilgan simvollar sonini qaytaradi.

* **Istream& putback(S);**

Agar **get** doirasidagi **streambuf** obyektida bo‘sh fazo mavjud bo‘lsa, unda o‘sha yerga **S** simvoli joylashtiriladi.

* **istream& ignore(int count=1,int target=EOF);**

Quyidagilar bajarilmaguncha, **istream** dan simvol chiqarilaveradi:

– funksiya **count** simvollarni chiqarmaguncha;

– **target** simvoli aniqlanmaguncha;

– faylni oxiriga yetmaguncha.

ostream sinfida quyidagi funksiyalar taviflangan:

*** ostream& put(char C);**

ostream ga **S** simvolni joylashtiradi.

*** ostream& write(const char* buffer,int size);**

Buferda mavjudlarni ostreamga yozadi. Xatoga duch kelmaguncha yoki **size** simvollarni nusxasi olmaguncha simvollarni nusxasi olinadi. **Bufer** formatlanmasdan yoziladi. Nol simvollarga ishlov berish boshqa ishlov berishlardan farq qilmaydi. Quyidagi funksiya ishlov berilmagan (binar va matnli) ma'lumotlarni **ostream**ga uzatadi.

*** ostream& flush();**

streambuf buferni olib tashlaydi.

Bu funksiyalardan tashqari **istream** sinfda >>, ostream sinfda esa << amallar qayta yuklangan. << va >> amallar ikkita operandga ega. Chap operandi – bu **istream** (**ostream**) sinfning obyekti, o‘ng operandi esa – bu dasturlash tilida ko‘rsatilgan turdagi ma'lumot.

Misol:

```
#include <iostream>
#include <cstring>
using namespace std;
int  main(){
char a[250];
cin.getline(a,250);
cout.write(a,strlen(a));
cout.put('\n');
cin.get(a,250);
cout.write(a,strlen(a));
return 0;
}
```

Formatlash

Ushbu ma'lumotlar uchun **cout**, **cin**, **cerr**, **clog** standart oqimlarga kiritish << va chiqarish >> amallarni to‘g‘ridan-to‘g‘ri qo‘llash qayta uzatish qiymatlarni tashqi tavsiflash aytib o‘tilmagan formatlardan foydalanishga olib keladi.

Chiqaruvchi axborotni tavsiflash formatlari va ma'lumotlarni kiritishda qabul qilish qoidalari dasturlovchi orqali formatlash bayroqlari yordamida o‘zgartiriladi. Bu bayroqlar **ios** bazaviy sinfdagi hamma oqimlardan meros bo‘lgan.

Formatlash bayroqlardan tashqari **ios** sinfning quyidagi **protected** komponentalari ishlatiladi:

int x_width – chiqarish maydonning minimal eni.

int x_precision – qiritishda haqiqiy sonlarning tavsiflash aniqligi (kasr qisimning raqamlar soni);

int x_fill – chiqarishda to‘ldiruvchi simvol, bo‘shlik – ko‘rsatilmagan holda.

Ushbu maydonlarni qiymatlarini olish (o‘rnatish) uchun quyidagi funksiyalar komponentalari ishlatiladi:

```
int width();
int width(int);
int precision();
int precision(int);
char fill();
char fill(char);
```

Agar bir marta **cout.fill**, yordamida to‘ldiruvchi simvol tanlansa **cout.fill** qayta chaqirilmaguncha o‘zgarmaydi.

Misol:

```
#include <iostream>
using namespace std;
class Test{
double a;
public:
Test(double a1=0){a=a1;}
void show(){
cout.width(11);
cout.precision(9);
cout.fill('_');
cout<<a<<endl;
}
void flag_print() {
cout<<"width="<<cout.width();
cout<<" precision="<<cout.precision();
cout<<" fill="<<cout.fill()<<endl;
};
};
int   main() {
double a=3.11111101;
cout<<a<<endl;
Test t(a);
t.flag_print();
t.show();
system("pause");
return 0;
}
```

Natija:

```
width=0 precision=6 fill=
3.11111
_3.11111101
```

Manipulyatorlar

Manipulyatorlar - oqim ishini modifikatsiyalashini imkon etuvchi maxsus funksiyalar. Manipulyatorlarning xususiyati shunda-ki, ularni >> yoki << amallarning o‘ng operand sifatida foydalanish mumkin. Chap operand sifatida esa har doimgidek oqim (oqimga ilova) ishlatiladi.

boolalpha satr sifatida true va false tasvirlash;

***noboolalpha** mantiqiy true va false qiymatlarni 1 va 0 tasvirlash;

showbase sanoq asosini ko‘rsatuvchi prefiksni chiqarish;

***noshowbase** sanoq asosini ko‘rsatuvchi prefiksni chiqarmaslik;

showpoint har doim o‘nlik nuqtani chiqarish;

***noshowpoint** o‘nlik nuqtani kasr qism nol bo‘lmasa chiqarish;

showpos manfiy bo‘lmagan sonlar uchun + chiqarish;

***noshowpos** manfiy bo‘lmagan sonlar uchun + chiqarmaslik;

***skipws** kiritish operatorida bo‘shlik simvolini o‘tkazish;

noskipws kiritish operatorida bo‘shlik simvolini o‘tkazmaslik;

uppercase o‘n oltilik sanoq tizimida 0X; ilmiy notatsiyada E chiqarish;

***nouppercase** o‘n oltilik sanoq tizimida 0x; ilmiy notatsiyada e chiqarish;

***dec** o‘nlik sanoq tizimida chiqarish;

hex o‘nlik oltilik sanoq tizimida chiqarish;

oct sakkizlik sanoq tizimida chiqarish;

left qiymat o‘ng tomoniga to‘ldirish belgisini qo‘shish;

right qiymat chap tomoniga to‘ldirish belgisini qo‘shish;

internal belgi va qiymat orasiga to‘ldirish belgisini qo‘shish;

***fixed** suzuvchi nuqtali sonni o‘nlik notatsiyada chiqarish;

scientific suzuvchi nuqtali sonni ilmiy notatsiyada chiqarish;

flush bufer ostream tozalash;

ends nol belgi qo‘shish va bufer ostream tozalash;

endl yangi qator belgisini qo‘shish va bufer ostream tozalash;

ws bo‘shlik simvolini o‘tkazish.

Quyidagi manipulyatorlar uchun **#include <iomanip>** ulash lozim:

setfill(ch) bo‘sh joyni ch simvoli bilan to‘ldirish;

setprecision(n) suzuvchi nuqtali son chiqarish aniqligini n ga o‘rnatish;

setw(w) kiritish yoki chiqarish maydoni kengligini w ga o‘rnatish;

setbase(b) butun sonni b asosda chiqarish.

Misol:

```
#include <iostream>
#include <iomanip>
using namespace std;
class Test{
double a;
public:
Test(double a1=0){a=a1;}
void show(){
cout<<setw(11);
cout<<setprecision(9);
cout<<setfill('_');
cout<<a<<endl;
}
};
int   main() {
double a=3.11111101;
cout<<a<<endl;
Test t(a);
t.show();
system("pause");
return 0;
}
```

Natija:

3.11111

_3.11111101

Oqimni holati

Har bir oqim u bilan bog‘liq holatga ega. Oqimni holati **enum** o‘tkazish ko‘rinishida **ios** sinfida tavsiflanadi.

```
public:
enum io_state{
goodbit, //0X00 xatosi yo‘q
eofbit,  //0X01 faylni oxiri
failbit, //0X02 oxirgi amali bajarilmagan
badbit,//0X04 mumkin bo‘lmagan amalini ishlatishni xarakat qilish
hardfail        //0X08 taqdiriy xato
};
```

ios obyekti bilan oxirgi bajarilgan amalini natijalarini aniqlovchi bayroqlar **state** o‘zgaruvchisida mavjud. Shu o‘zgaruvchining qiymatlarini **int rdstate()** funksiyalari yordamida olish mumkin.

Bundan tashqari, oqimlar holatini quyidagi funksiyalar orqali tekshirish mumkin:

int bad(); 1, agar badbit yoki hardfail

int eof(); 1, agar eofbit

int fail(); 1, agar failbit, badbit yoki hardfail

int good(); 1, agar goodbit

Agarda >> amali ma'lumotlarni yangi turlari uchun ishlatilsa, unda uni qayta yuklashda tegishli tekshirishlarni ko‘zda tutmoq lozim.

Formatlash bayroqlari

Formatlash bayroqlari alohida qayd etilgan bitlar ko‘rinishida amalga oshirilgan va **long x_flags** sinfning **protected** komponentasida saqlanadi. Ularga murojaat etish uchun tegishli **public** funksiyalar mavjud.

Asosiy formatlash bayroqlari.

skipws – o‘qishda bo‘shlik simvollarini o‘tkazib yuborish;

left – qiymatni chap chegaraga tekislash;

right – qiymatni o‘ng chegaraga tekislash;

internal – son qiymati va ishorasi orasiga qo`yiladigan simvol.

dec– o‘nlik sanoq tizimi;

oct – sakkizlik sanoq tizimi;

hex – o‘n oltilik sanoq tizimi;

boolalpha – 0 va 1 ni mos holda "false" va "true" ga o‘girish;

showbase – chikarishda sanoq tizimi belgisini chop etish;

showpoint – haqiqiy sonlarni chiqarishda albatta o‘nlik nuqta va undan keyingi nollarni chiqarish;

uppercase - yuqori registrdagi harflardan foydalanish;

showpos – musbat sonlar ishorasini chikarish;

scientific – haqiqiy sonlar uchun nuqtali daraja va mantissani ko‘rsatish;

fixed – haqiqiy sonlarni chiqarishda fiksirlangan nuqtali formatdan foydalanish;

unitbuf – hamma oqimlarni tozalash;

stdio – hamma **stdout** , **stderr** oqimlarni tozalash;

Bayroqlar bilan ishlash uchun **setf()** va **unsetf()** usullaridan foydalanish mumkin.

Bundan tashqari formatlash bayroqlari bilan ishlash uchun **setiosflags()** va **resetiosflags()** parametrli manipulyatorlardan foydalanish mumkin. Buning uchun **#include <iomanip>** ulash lozim.

Misol:

```
#include <iomanip>
using namespace std;
class Test{
double a;
public:
Test(double a1=0){a=a1;}
void show(){
cout<<setiosflags(ios::fixed);
cout<<setiosflags(ios::showpoint);
cout<<a<<endl;
}
};
int  main() {
double a=0.19;
cout<<a<<endl;
Test t(a);
t.show();
system("pause");
return 0;
}
```

Natija:

0.19

0.190000

Chiqarish operatorini qo‘shimcha yuklash

Chiqarish operatori **ostream** sinfi obyektiga ilova qaytaruvchi binar operatordir. Umumiy holda qo‘shimcha yuklangan chiqarish operatori ta'rifi quyidagi ko‘rinishga ega:

```
ostream& operator <<( ostream& os, const ClassType &object ){
…..
Return os;
```

}

Bu ta'rif birinchi argumenti **ostream** obyektiga ilova, ikkinchisi odatda konstant biror sinf obyektiga ilovadir. Qaytariluvchi qiymat **ostream** obyektiga ilovadir.

Birinchi argument ilova bo‘lgani uchun, chiqarish operatori sinf a'zosi sifatida emas oddiy funksiya sifatida ta'riflanishi zarur. Agar funksiya yopiq sinf a'zolariga murojaat qilishi zarur bo‘lsa, do‘stona deb e'lon qilinishi zarur.

Chiqarish operatorini qo‘shimcha yuklash:

```
#include <iostream>
#include <string>
using namespace std;
class Person {
friend ostream&
operator<<( ostream&, const Person& );
public:
Person( string word, int cnt):name(word),year(cnt){};
private:
string name;
int year;
};
ostream& operator <<( ostream& os, const Person& wd ){
os <<wd.name<<" "<<wd.year;
return os;
};
int  main(){
Person wd("Aman",25);
cout << "wd:\n" << wd << endl;
return 0;
}
```

Qo‘shimcha yuklangan chiqarish operatorini **ofstream** sinfi obyektlariga ham qo‘llash mumkin.

Fayllarni ochish va yopish

C++ da fayllar bilan ishlash **ftsream** kutubxonasidagi biron-bir sinflar yordamida amalga oshiriladi.

Ftsream kutubxonasi fayllarni o‘qib olish uchun javob beradigan **ifstream** sinfiga hamda faylga axobotning yozib olinishiga javob beradigan **oftsream** sinfiga ega.

Biron-bir faylni yozish yoki o‘qish uchun ochish, **oftsream** turdagi yoki mos holda **iftsream** turdagi o‘zgaruvchini yaratish kerak. Bunday o‘zgaruvchini initsiallashda fayl nomi o‘zgaruvchi nomidan keyin qavs ichida berilgan belgilar massivi ko‘rinishida uzatiladi.

Agar fayl ham dasturning bajarilayotgan fayli joylashtirilgan papkada bo‘lsa, u holda faylning nomi to‘liq ko‘rsatilmasligi mumkin (faqat fayl nomi, unga borish yo‘lisiz). Bundan tashqari fayl nomini to‘g‘ridan-to‘ri ko‘rsatish o‘rniga, uning nomidan iborat belgilar massivlarini ko‘rsatish mumkin.

Fayllar bilan ishlash uchun quyidagi usullardan foydalaniladi:

bool iosbase::eof() bu funksiya 0 qiymatini qaytarib beradi agar fayl oxiri hali uchramagan bo‘lsa, agar fayl oxiri uchrasa, 1 qimatini qaytaradi.

bool iosbase::fail() funksiya yolg‘on (0) ni qaytaradi, agar fayl amali jarayonida xatolar bo‘lmagan bo‘lsa. Biroq, agar xato uchrasa, funksiya haqiqatni qaytaradi.

void ofstream::close() funksiya faylni berkitadi. Faylni yopayotganda, dastur ushbu faylga yozib olgan barcha ma'lumotlar diskka tashlanadi va ushbu fayl uchun katalogdagi yozuv yangilanadi.

Faylli kiritish-chiqarish

Qo‘shish tuzimida faylni ochish uchun quyida ko‘rsatilgan ikkinchi parametrni ko‘rsatish lozim:

fstream output_file("FILENAME.EXT", ios::app);

Bu holda **ios::app** parametri *faylni ochish tuzimiga* ko‘rsatadi:

Ochish tuzimi	***Vazifasi***
ios::app	Fayl ko‘rsatkichni faylni oxirida joylashtirib ko‘shish tuzimida faylni ochadi.
ios::ate	Fayl ko‘rsatkichini faylni oxiriga joylashtiradi. O‘qish mumkin emas, chiqaruvchi ma'lumotlar faylni oxiriga

	yoziladi.
ios::in	Kiritish uchun faylni ochilishini ko‘rsatadi.
ios::nocreate	Agarda ko‘rsatilgan fayl mavjud bo‘lmasa, fayl yaratilmaydi va xato qaytariladi.
ios::noreplace	Agarda fayl mavjud bo‘lsa, ochish amali to‘xtash va xatolikni qaytarish lozim.
ios::out	Chiqarish uchun faylni ochilishini ko‘rsatadi
ios::trunc	Mavjud bo‘lgan faylni ichidagini olib tashlaydi (ko‘chiradi).

Quyidagi faylni ochish amali faylni ochishda, mavjud bo‘lgan faylni ko‘chirishini oldini olish uchun **ios::noreplace** tuzimidan foydalanadi.

Komponent-funksiya **open** quyidagi ko‘rinishga ega:

void open (const char *filename , int mode=ko‘zda tutilgan qiymat

int protection = ko‘zda tutilgan qiymat)

Birinchi parametr **filename** mavjud yoki yaratilayotgan fayl nomi, ikkinchi parametr **mode** – fayl ochish rejimlari parametrlari diz'yunksiyasi, uchinchi parametr **protection** (himoya) – kam ishlatiladi. Aniqrog‘i dastur uchun ko‘zda tutilgan qiymat yetarli.

Usul **open** quyidagicha chaqiriladi:

Oqim_ ismi open(fayl nomi, rejim, ximoya)

Agar oqim **ofstream** sinfiga tegishli bo‘lsa, ikkinchi parametr **ios:out** qiymatga ega.

Funksiya **open()** to‘g‘ri bajarilganligini tekshirish uchun qo‘shimcha yuklangan **!** amalidan foydalaniladi. Agar xato mavjud bo‘lsa, qiymat 0 dan farqli.

Agar oqim **fstream** sinfiga tegishli bo‘lsa, ikkinchi parametrni ko‘rsatish shart.

Misol:

```
#include<iostream>
#include <fstream>
using namespace std;
class TestFile{
ifstream input_file;
char line[255];
public:
```

```
TestFile(string st){
input_file.open(st.c_str(),ios::in);
if (input_file.fail()) cerr << "Error Open file " << endl;
}
void show_all(){
while((!input_file.eof())&&(!input_file.fail())) {
input_file.getline(line, sizeof(line));
if (! input_file.fail()) cout << line << endl;
}
}
void close(){
input_file.close ();
}
};
int main() {
TestFile f("BOOKINFO.DAT");
f.show_all();
f.close();
return 0;
}
```

O‘qish va yozish amallarining bajarilishi

Hozirga qadar ko`rilgan dasturlar belgili satrlar ustida amallar bajarar edi. Dasturlar murakkablashgani sari, massivlar va tuzilmalarni o‘qish va yozish kerak bo‘lib qoladi. Buning uchun dasturlar **read** va **write** funksiyalaridan foydalanishlari mumkin. Bu funksiyalaridan foydalanishda ma'lumotlar o‘qiladigan yoki yozib olinadigan ma'lumotlar buferini, shuningdek buferning baytlarda o‘lchanadigan uzunligini ko‘rsatish lozim. Bu quyida ko‘rsatilganidek amalga oshiriladi:

```
input_file.read(buffer, sizeof(buffer));
output_file. write(buffer, sizeof(buffer));
```

Misol:

```
#include<iostream>
#include <fstream>
using namespace std;
class employee {
char name[64];
int age;
float salary;
public:
employee(){};
```

```
employee(char* s, int a, float f){
strcpy(name,s);age=a;salary=f;
}
void show(){
cout<<name<<" "<<age<<" "<<salary;
}
};
class TestFile{
public:
void write_file(){
employee worker;
ifstream emp_file("EMPLOYEE.DAT");
emp_file.read((char *)&worker,sizeof(employee));
worker.show();
emp_file.close ();
}
void read_file(){
employee worker;
ofstream emp_file("EMPLOYEE.DAT");
emp_file.write((char *) &worker, sizeof(employee));
emp_file.close();
}
};
int main() {
TestFile f;
f.read_file();
f.write_file();
return 0;
}
```

Oqim ko‘rsatkichlari

Oqim o‘qish yoki yozish pozitsiyasini aniqlash uchun ixtiyoriy oqim sinfida **get** yoki **put** ko‘rsatkichlaridan foydalaniladi. Bulardan:

• **ifstream** oqim **get** ko‘rsatkichga ega.

• **ofstream** oqim **put** ko‘rsatkichga ega.

• **fstream** oqim ikkala ko‘rsatkichga ega.

Ko‘rsatkichlarni boshqarish uchun quyidagi usullardan foydalaniladi:

• **istream::tellg()** fayl boshidan **get** ko‘rsatkich ko‘rsatayotgan pozitsiyagacha baytlar sonini qaytaradi;

• **ostream::tellp()** fayl boshidan **put** ko‘rsatkich ko‘rsatayotgan pozitsiyagacha baytlar sonini qaytaradi;

• **istream:: seekg ()** oqimda **get** ko‘rsatkich holatini o‘rnatadi;

• **ostream: :seekp()** oqimda **put** ko‘rsatkich holatini o‘rnatadi;

• **seekg() i seekp()** ko‘rsatkich siljishi yo‘nalishini va kattaligini o‘rnatadi.

Siljish yo‘nalishlari:

ios:beg siljish oqim boshidan hisoblanadi;

ios:cur siljish oqim joriy pozitsiyadan hisoblanadi;

ios:end siljish oqim oxiridan hisoblanadi.

Quyidagi misolda usullar fayl hajmini aniqlash uchun ishlatiladi:

```
#include<iostream>
#include <fstream>
using namespace std;
int main() {
int n1,n2;
ifstream drag("drag.txt", ios::in|ios::binary);
n1=drag.tellg();
drag.seekg(0, ios::end);
n2=drag.tellg();
cout<<"Size files"<<n2-n1<<endl;
return 0;
}
```

Binar fayllar

Binar fayllardan o‘qish va binar faylga yozish uchun **istream::read()** va **ostream: :write()** usullaridan foydalaniladi.

Quyidagi dasturda shu usullardan foydalanib "drag.txt" faylidan, "drag2.txt" fayliga nusxa olinadi.

```
#include <iostream>
#include <fstream>
using namespace std;
class TestBin{
fstream file1;//source file
fstream file2;//destination file
public:
TestBin(char* filename1, char* filename2){
```

```
file1.open(filename1, ios::in | ios::out | ios::binary);
file2.open(filename2, ios::out|ios::binary);
}
//copy the file
void Copy(){
char buffer;
int index = 0;
//sets the pointers to the beginning of the file
file1.seekg (0, ios::beg);
file2.seekp (0, ios::beg);
//reads the first value
file1.read(&buffer, 1);
while(file1.good() && file2.good()){
file2.write(&buffer, 1);
index++;
file1.seekp(index);
file2.seekg(index);
file1.read(&buffer, 1);
}
}
//size files
void sizefiles(){
int n1,n2;
n1=file1.tellg();
file1.seekg(0, ios::end);
n2=file1.tellg();
cout<<"Size files"<<n2-n1<<endl;
}
//closes both of the files
void close(void){
file1.close();
file2.close();
}
};
int main(){
TestBin delta("dragons.txt", "output1.txt");
delta.sizefiles();
delta.Copy();
delta.close();
return 0;
}
```

Satrli oqimlar

Satrli oqimlar bilan ishlash uchun **istringstream**, **ostringstream** va **stringstream** sinflaridan foydalaniladi. Bu sinflar mos ravishda **istream**, **ostream** va **iostream** sinflari vorislaridir. Satrli oqimlardan foydalanish uchun dasturga **<sstream>** sarlavhali faylni ulash lozim.

Satrli oqimlarni qo‘llash faylli oqimlarni qo‘llashga o‘xshashdir.

Farqi shunda-ki, oqim fizik ma'lumoti diskdagi faylda emas, operativ xotirada saqlanadi. Bundan tashqari satrli oqimlar sinflari **str()** usuliga ega bo‘lib, bu usul **string** turidagi satr nusxasini qaytaradi yoki oqimga shunday satr qiymatini beradi:

```
string str() const;
void str(const string& s);
```

Misol:

```
#include <iostream>
#include <fstream>
#include <sstream>
using namespace std;
class TestString{
fstream file;
public:
TestString(int argc, char* argv[]){
if(argc!=2) cerr<<"Error.";
file.open(argv[1],ios::in);
if(!file) cerr<<"file not found:";
}
//read the file
void read(){
int n = 0;
char buf[1024];
while (!file.eof()) {
n++;
file.getline(buf,sizeof(buf));
ostringstream line;
line<<"Line "<<n<<": "<<buf<<endl;
cout<<line.str();
}
};
void close(void){
file.close();
}
};
```

```
int main(int argc, char* argv[]){
TestString delta(argc,argv);
delta.read();
delta.close();
return 0;
}
```

Nazorat savollari:

1. Qaysi sinflar asosida oqimlar kutubxonasi qurilgan?
2. Satrli oqimlarni va ularrning vazifalarini ko‘rsating.
3. Oqimlar usullarini ko‘rsating.
4. Formatlash uchun qanday komponenta funksiyalardan foydalanadi?
5. Fayllar bilan ishlashda qaysi kutubxonadan foydalaniladi?
6. Fayl oxirini aniqlash uchun qaysi funksiyadan foydalaniladi?
7. Xatolikni aniqlash uchun qaysi funksiyadan foydalaniladi?
8. Fayllar bilan ishlash **read** va **write** funksiyalari vazifasini ko‘rsating.
9. Faylni ochish rejimlarini ko‘rsating.
10. Manipulyatorlar vazifasini ko‘rsating.

Misollar:

1. F1 fayldan F2 faylga juft sonlardan nusxa oling.

2. F1 fayldan F2 faylga juft satrlardan nusxa oling. F1 va F2 (baytlarda) fayllar hajmini hisoblang.

3. F1 fayldan F2 faylga 1 raqamdan boshlanuvchi sonlardan nusxa oling.

4. F1 fayldan F2 faylga «A» harfdan boshlanuvchi so‘zlardan nusxa oling. F2 faylda so‘zlar sonini hisoblang.

5. FILM strukturasi yaratilsin. Struktura kiritish va chiqarish funksiyasi yaratilsin. Dasturda struktura turidagi massiv kiritilib faylga yozilsin. Faylga yozilgan ma'lumotlar massivga o‘qilsin. Massiv chop etilsin.

Istisnolarni boshqarish

Istisno holatlar

C++ tili OMD doirasida istisnolarga xizmat ko‘rsatish standartini belgilab beradi. Istisno holatlar (exception) dasturda xatoni – kutilmagan hodisani ifodalaydi. Dastur o‘zining ishlab chiqilishida ko‘zda tutilmagan normal bo‘lmagan vaziyatga duch kelganda, boshqaruvni ushbu muammoni hal qilishga qodir bo‘lgan dasturning boshqa qismiga berishi mumkin hamda yo dasturni bajarishni davom ettirish yoki ishni tugallash kerak. Istisnolarni joydan joyga tashlab berish (*excpletion throwing*) dasturning normal bajarilishiga to‘sqinlik qiladigan sabablarning tashxisi uchun foydali bo‘lishi mumkin bo‘lgan axborotni tashlab berish nuqtasida to‘plash imkonini beradi. Siz dastur tugallanishi oldidan zarur hatti-harakatlarni bajaradigan istisnolarga ishlov bergich (*exception handler*) ni aniqlashingiz mumkin. Dastur ichida yuzaga keladigan sinxron istisnolar deb nomlanuvchi istisnolarga xizmat ko‘rsatiladi. Ctrl+C klavishalarini bosish kabi tashqi holatlar istisno hisoblanmaydi.

Dasturda har bir istisno holat sinf sifatida aniqlanadi. Istisno holatlar o‘zgaruvchilarni va sinf funksiyalarini ishlatishi mumkin.

Istisnolarni qayta ishlash

Dastur istisno holatni ko‘rishdan va unga javob berishdan oldin istisno holatni aniqlovchi C++ dagi **try** operatorini ishlatish lozim. Istisnolarni generatsiya qila oladigan kod bloki **try** kalit-so‘z bilan boshlanadi va shakldor qavslar ichiga olinadi. Agar **try** blok ichida istisnoni topib olsa, dasturiy uzilish sodir bo‘ladi hamda quyidagi hatti-harakatlar ketma-ketligi bajariladi:

1.Dastur istisnoga ishlov bergichning to‘g‘ri keladiganini qidiradi.

2.Agar ishlov bergich topilsa, stek tozalanadi va boshqaruv istisnolarga ishlov bergichga uzatiladi.

3.Agar ishlov bergich topilmagan bo‘lsa, ilovani tugatish uchun **terminate** funksiyasi chaqiriladi.

Yuzaga kelgan istisnoga ishlov beruvchi kod bloki **catch** kalit-so‘z bilan boshlanadi va shakldor qavs ichiga olinadi. Istisnoga ishlov bergichning kamida bitta kod bloki bevosita **try** blokining ortidan kelishi kerak. Dastur generatsiya qilishi mumkin bo‘lgan har bir istisno uchun o‘z ishlov bergichi ko‘zda tutilgan bo‘lishi kerak. Istisnolarga ishlov bergichlar navbatma-navbat ko‘rib chiqiladi hamda turi bo‘yicha catch operatoridagi argument (dalil) turiga to‘g‘ri keladigan istisnoga ishlov bergich tanlab olinadi. Ishlov bergich tanasida **goto** operatorlari bo‘lmagan taqdirda, berilgan **try** bloki istisnolariga ishlov bergichning oxirgisidan keyin kelgan nuqtadan boshlab dasturning bajarilishi yana davom etadi.

Qanday istisno holat ro‘y berganini aniqlash uchun **try** operatoridan so‘ng dastur bitta yoki bir nechta **catch** operatorlarni joylashtirish lozim.

Agarda funksiyaning chaqiruvi xatosiz bajarilgan va istisno xatolar aniqlanmagan bo‘lsa, C++ **catch** operatorini shunchaki e’tiborga olmaydi.

Qayta ishlovchilar tartibi muhimdir. Ixtiyoriy istisnolarni qayta ishlash uchun **catch(...)** operatoridan foydalaniladi.

Istisnolarni generatsiya qilish

C++ o‘zi istisno holatlarni yuzaga keltirmaydi. Ularni C++ ning **throw** operatoridan foydalangan dasturlar yuzaga keltiradi. Istisno yuzaga kelganda, **throw** operatoridagi **<ifoda>** nom berish ifodasi muvaqqat obyektni nomlaydi (initsiallashtiradi), Bunda muvaqqat obyektning turi ifoda argumenti (dalili) ning turiga mos keladi. Ushbu obyektning boshqa nusxlari, masalan, istisno obyektidan nusxa ko‘chirish konstruktori yordamida generatsiya qilinishi mumkin.

Quyidagi misolda chetlanish funksiya tanasidan tashqarida qayta ishlanadi.

```
#include <iostream>
using namespace std;
void ff(int x){
if (x == 10)
throw x;
};
int main ( ){
try{
ff(10);
```

```
}
catch(int x) {
cerr <<"O yo'q! x teng " << x << "!!!!";
}
catch(float f){
cerr << "Nima bo'ldi?";
}
return 0;
}
```

Natija:
O yo'q! x teng 10!!!!

Quyidagi dastur ixtiyoriy istisnoni qayta ishlashga misol bo‘ladi:

```
#include <iostream>
using namespace std;
int main (void){
try{
throw 6;
}
catch(...){
cerr << "There was an exception, but it was caught!";
}
return 0;
}
```

Natija:
There was an exception, but it was caught!

Kutilmagan istisnolarni qayta ishlash

Agar dasturda ko`zda tutilmagan istisno hodisa yuz bersa, standart istisnolarni qayta ishlovchi ishlatiladi. Ko`p hollarda, bu standart qayta ishlovchi dastur bajarilishini to`xtatib qo`yadi.

Avval **unexpected** funksiyasi chaqirilib, undan so‘ng jimlik bo‘yicha **terminate** funksiyasi ishga tushadi. Bu funksiya dasturni to‘xtatish uchun **abort** funksiyasini chaqiradi.

Dasturda maxsus qayta ishlovchidan foydalanish uchun **set_unexpected** va **set_terminate** funksiyalaridan foydalanish lozim. Bu funksiyalar prototiplari **except.h** sarlavhali faylda aniqlangan. Bu funksiyalar **void** turiga ega bo‘lib, parametrlarga ega bo‘lmasligi kerak.

```
#include <exception>
```

```
#include <iostream>
using namespace std;
void error_handler();
void error_handler(){
cerr << "There was an uncaught error.";
}
int main (){
set_terminate(error_handler);
throw 5;
return 0;
}
```

G‘ayrioddiy holatlar sinf sifatida

Yuqorida keltirilgan kamchilikdan xalos bo‘lish uchun cheklanishni maxsus sinov obyekti sifatida hosil qilish mumkindir.

Chetlanishlarni sinf sifatida dasturlashni Evklid algoritm misolida ko‘rib chiqamiz. Bu algoritm ikki butun manfiy bo‘lmagan sonlarning EKUBini topishga mo‘ljallangandir. Algoritm har bir qadamida quyidagi amallar bajariladi:

Agar x>=y bo‘lsa, javob topilgan

Agar x<y bo‘lsa, y=y-x

Agar x>y bo‘lsa, x=x-y

Quyidagi misolda DATA sinfi kiritilgandir.

```
#include <iostream>
#include <string>
using namespace std;
struct DATA{
int n,m;
string s;
DATA (int x, int y, string c){
n=x; m=y; s=c;
}
};
struct EVKLID{
int GCM_ONE (int x, int y){
if (x==0||y==0) throw DATA ( x, y,"\n ZERO!");
if(x<0) throw DATA (x, y, "\n Negative parametr1");
if(y<0) throw DATA (x, y, "\n Negative parametr2");
while (x!=y)
{
if(x>y) x=x-y;
```

```
else y=y-x;
}
return x;
}
};
int main(){
EVKLID E;
try{
cout<<"\n GCM(66,44)="<<E.GCM_ONE(66,44);
cout<<"\n GCM_ONE(0,7)="<<E.GCM_ONE(0,7);
cout<<"\n GCM_ONE(-12,8)="<<E.GCM_ONE(-12,8);
}
catch (DATA d){
cerr<<d.s<<" x="<<d.n<<" y="<<d.m;
}
return 0;
}
```

Natija

GCM_ONE(66,44)=22

ZERO! x=0 y=7

DATA obyekti bu misolda funksiya tanasida sinf konstruktori bajarilganda yaratiladi.

Bu obyekt cheklanish bo‘lmaganida chaqirish nuqtasida bo‘lar edi.

Shunday qilib chegaranishlar sinf obyekti bo‘lishi lozimdir. Bu sinflarni global ta'riflash shart emas. Asosiy talab – tanlanish nuqtasida ma'lum bo‘lishi.

Istisno holatning ma'lumotlar elementlaridan foydalanish

Yuqorida ko‘rib o‘tilgan misollarda dastur **catch** operatoridan foydalanib, qanday istisno holat ro‘y berganini va ularga tegishli holda javob berish imkonini beradi. Istisno holatga tegishli shunday ma'lumotni saqlash uchun dastur istisno holat sinfiga ma'lumotlar elementlarini qo‘shishi, agar keyinchalik dastur istisno holatni yuzaga keltirsa, u ushbu ma'lumotni, istisno holatiga ishlov beruvchi funksiyaga o‘zgaruvchi sifatida uzatadi.

Istisno holatga ishlov berishda bu parametrlar sinfga tegishli o‘zgaruvchilarga o‘zlashtirilishi mumkin (konstruktorga o‘xshaydi).

Istisnolar spetsifikatsiyasi. Funksiya ta'rifida **throw** kalit so‘zi yordamida shu funksiya yaratishi mumkin bo‘lgan istisno holatlar ko‘rsatilishi mumkin. Funksiya e'lonida istisnolar spetsifikatsiyasi uning ta'rifidagi spetsifikatsiya bilan aniq mos kelishi lozim.

Qavslar ichida vergul bilan ajratilgan ixtiyoriy sondagi istisnolar turlari ko‘rsatilishi mumkin.

Agar funksiya istisnolar spetsifikatsiyasiga ega bo‘lmasa, u ixtiyoriy istisno yaratishi mumkin. Masalan, pastdagi funksiya ixtiyoriy istisno yaratishi mumkin.

Agar spetsifikatsiyada turlar ro‘yxati bo‘sh bo‘lsa, funksiya hech qanday istisno generatsiya qilmaydi.

Agar funksiya signaturasidan so‘ng birorta istisno turi ko‘rsatilmagan bo‘lsa, funksiya o‘zi istisno generatsiya qilmaydi, lekin chaqirilgan funksiyalar yaratgan istisnoni uzatishi mumkin.

Istisnolar spetsifikatsiyasi funksiya signaturasi qismi hisoblanmaydi. Ya'ni faqat istisnolar spetsifikatsiyasi bilan farq qiladigan ikki funksiya yaratib bo‘lmaydi.

Misol:

```
#include <iostream>
using namespace std;
class nuke_meltdown {
int k;
public:
nuke_meltdown(int m){k=m;}
void print(){cerr <<"parametr "<<k<<" not normal"<< endl; }
};
void add_u232(int amount) throw(nuke_meltdown) {
if (amount < 255) cout << "Parametr add_u232 normal!" << endl;
else throw nuke_meltdown(amount);
}
int main() {
try {
add_u232(255);
}
catch (nuke_meltdown a) {
a.print();
}
```

```
system("pause");
return 0;
}
```
Natija:
parametr 255 not normal

Misol:
```
#include <iostream>
using namespace std;
const int maxi=1000;
class DivByZero{};
class Over{};
int divide(int a, int b) throw(DivByZero,Over){
if ((a>maxi)||(b>maxi)){
throw Over();
};
if (b == 0){
throw DivByZero();
}
return a/b;
};
int main (){
try{
divide(5,0);
}
catch(DivByZero){
cerr << "DivideByZero!";
}
catch (Over){
cerr << "Overflow!";
}
return 0;
}
```

Stekni orqaga qaytarish va destruktorlarni chaqirish. Istisno holat yaratilganda **catch-**qayta ishlovchisini qidirish – stekni orqaga qaytarish – istisno yaratgan funksiyadan boshlanadi va joylashtirilgan chaqiriqlar ko‘rib chiqiladi.

Bu jarayonda ko‘rilgan funksiyalardan anomal chiqish yuz beradi. Agar biror resursni egallagan bo‘lsa, u bo‘shatilmaydi.Qayta izlash jarayonida biror lokal obyekt aniqlangan blokdan chiqilsa, shu obyekt destruktori avtomatik chaqiriladi. Agar resurs sinf sifatida ta'riflangan bo‘lsa va resursga egalik konstruktorda

amalga oshirilib, destruktorda bo‘shatilsa, qayta ishlanmagan istisno natijasida funksiyadan chiqilganda shu funksiyada egallangan resurs avtomatik bo‘shatiladi.

Misol:

```
#include <iostream>
#include<exception>
using namespace std;
class PTR{
int* ptr;
public:
PTR(int n){
ptr=new int[n];
};
~PTR(){
delete[]ptr;
cout<<"killed"<<endl;
}
};
int divide(int a, int b) throw (exception) {
PTR d(3);
if (b==0) throw exception();
return a/b;
};
int main () {
try {
divide(5,0);
}
catch (exception a) {
cout<<"error";
a.what();
}
system("pause");
return 0;
}
```

Natija:

killed

error

Vorislikdan foydalanish. G‘ayrioddiy holatlar sinf bo‘lgani uchun vorislik asosida ierarxiya yaratish mumkin. Ikkita suzuvchi sonlarning bo‘linishidan hosil bo‘lgan bo‘linmaning interaktiv hisoblagichidan iborat dasturni ko‘rib chiqamiz. Bu dasturda istisnoni taqdim etayotgan sinflar ierarxiyasi yaratilgan.

G‘ayrioddiy holatlar yaratilganda vorislikka o‘tuvchi usul qayta ta'riflanganda, har bir g‘ayrioddiy holat uchun alohida **catch** bo‘limi yaratish talab etiladi. Agar bu usul virtual, deb e'lon qilinsa, faqat bitta **catch** bo‘limida avlod hamda ajdod cheklanishlar obyektlarini qayta ishlash mumkin bo‘ladi. Buning uchun ajdod sinf obyektiga ilovadan foydalanish lozimdir. Ajdod sinf abstrakt sinf ham bo‘lishi mumkin.

```
#include <iostream>
using namespace std;
const int max_num=100;
class MathError {
public:
virtual void print()=0;
};
class DivideByZero : public MathError {
public:
void print(){
cerr << " DivideByZero!";
}
};
class Overflow : public MathError  {
public:
void print(){
cerr << " Overflow!";
}
};
class Calc{
public:
static int divide(int a, int b) {
if (b == 0) {
throw DivideByZero();
return false;
}
if (a>max_num||b>max_num) {
throw DivideByZero();
return false;
}
return a/b;
};
};
int main () {
try {
Calc::divide(5,0);
```

```
}
catch (MathError &a) {
a.print();
}
return 0;
}
```

Istisno holatlar va sinflar

Sinf yaratishda shu sinfga tegishli istisno holatlarni aniqlash mumkin. Aniq sinfga tegishli istisno holatni yaratish uchun ushbu istisno holatni sinfning umumiy (**public**) elementlari sifatida kiritish zarur.

Istisnolar konstruktordagi xatolar haqida ma'lumot berishga imkon beradi. Konstruktor chaqiruvchi funksiya tekshirib ko‘rishi mumkin bo‘lgan qiymat qaytarmagani uchun istisnolarsiz quyidagicha xatolik haqida ma'lumot berish mumkin:

1. Obyektni xatolik bilan yaratish.
2. Obyekt yaratilmagan.

Istisnolar bu ma'lumotni tashqariga uzatishga imkon beradi.

Misol:

```
#include <iostream>
using namespace std;
const int maxsize=100;
class sstring {
char  p[maxsize];
int size;
public:
class Range {
int index;
public:
Range(int i) : index(i){};
int get_index(){return index;}
};
class Size {};
sstring(int sz);
char& operator[](int i);
};
sstring::sstring (int sz) {
if (sz<0 || maxsize<sz) throw Size();
size=sz;
```

```
};
char& sstring::operator[](int i) {
if (0<=i && i <size) return p[i];
throw Range(i);
}
int main() {
try {
sstring v(100);
cout<<v[200];
}
catch (sstring::Range r ) {
cerr << "index error " << r.get_index() << '\n';
}
catch (sstring::Size) {
cerr << "size error "<< '\n';
}
system("pause");
return 0;
}
```

Natija:

index error 200

Standart istisnolar ierarxiyasi

C++ tili standart kutubxonasida shu kutubxonaga kiruvchi funksiyalar yaratuvchi g`ayrioddiy holatlar haqida ma'lumot beruvchi sinflar ierarxiyasi mavjud. Bu sinflar asosida maxsus g‘ayrioddiy holatlar sinflarini yaratish va ishlatish mumkin. Ierarxiya tuguni **exception** sinfidir. Bu sinf **<exception>** sarlavhali faylda aniqlangan bo‘lib, qolgan sinflar uchun ajdod sinf bo‘lib xizmat qiladi.

Sinf interfeysi:

```
namespace std {
class exception
public:
exception() throw();
exception( const exception & ) throw();
exception& operator=( const exception & ) throw();
virtual ~exception() throw();
virtual const char* what() const throw();
}
```

Har qanday standart sinf kabi **exception** sinfi **std** nomlar fazosiga joylashtirilgan. Sinf hamma elementlari umumiy (**public**) sifatida ta'riflangan. Destruktor virtual deb e'lon qilingan. Asosy funksiya bu virtual funksiya **what()** bo‘lib, g‘ayrioddiy holat haqidagi ma'lumotni C-satr sifatida qaytaradi. Ajdod sinflar bu funksiyani qo‘shimcha yuklashi mumkin. Hamma funksiyalar bo‘sh **throw()** spetsifikatsiyana ega, ya'ni hech qanday g‘ayrioddiy holat yaratmaydi.

Dastur istisno obyektlar bilan amallar bajarganda (masalan, **exception** turidagi **catch**-qayta ishlovchilar ichida), obyektlarni yaratish, nusxalash va o‘chirish funksiyalari istisno generatsiya qilishidan qo‘rqmasa bo‘ladi.

Misol:

```
#include <iostream>
#include<exception>
using namespace std;
class GreatError:public exception{};
class LessError:public exception{};
class Test{
public:
static void funk(int a, int maxi, int mini) throw (GreatError,LessError) {
if (a< mini) throw LessError();
if (a> maxi) throw GreatError();
cout<<a;
};
};
int main () {
try {
Test::funk(5,0,4);
}
catch(exception& a) {
cerr <<a.what()<<endl;
}
system("pause");
return 0;
}
```

Natija:

10GreatError

Mantiqiy va bajarilish vaqti xatolari. Standart kutubxonadagi hamma sinflar **Exception** dan tashqari odatda ikki kategoriyaga ajratiladi: mantiqiy xatolar va bajarish vaqti xatolari.

Mantiqiy xatolarni dastur bajarilmasdan oldin tutib olish mumkin hisoblanadi. Standart kutubxonada quyidagi mantiqiy xatolar kiritilgan:

```
namespace std {
  class logic_error : public exception {  // logicheskaya oshibka
  public:
    explicit logic_error( const string &what_arg );
  };
  class invalid_argument : public logic_error { // nevernqy argument
  public:
    explicit invalid_argument( const string &what_arg );
  };
  class out_of_range : public logic_error { // vne diapazona
  public:
    explicit out_of_range( const string &what_arg );
  };
  class length_error : public logic_error { // nevernaya dlina
  public:
    explicit length_error( const string &what_arg );
  };
  class domain_error : public logic_error { // vne dopustimoy oblasti
  public:
    explicit domain_error( const string &what_arg );
}
}
```

Agar argument qiymati noto‘g‘ri berilgan bo‘lsa, funksiya **invalid_argument** qaytarishi mumkin. Agar diapazonda yotmasa, **out_of_range** qaytarishi mumkin. Agar mumkin bo`lgan maksimal uzunlikdan katta uzunlikdagi obyekt yaratishga urinilsa, **length_error** qaytarishi mumkin.

Bajarilish vaqti xatolarini dasturga bog‘liq bo‘lmagan hodisa keltirib chiqaradi. Dastur bajarilmasdan oldin ularni ajratish mumkin emas. Standart kutubxonada quyidagi sinflar aniqlangan:

```
namespace std {
  class runtime_error : public exception {  // oshibka vremeni vqpolneniya
  public:
    explicit runtime_error( const string &what_arg );
  };
  class range_error : public runtime_error { // oshibka diapazona
  public:
    explicit range_error( const string &what_arg );
```

```
};
class overflow_error : public runtime_error { // perepolnenie
public:
  explicit overflow_error( const string &what_arg );
};
class underflow_error : public runtime_error { // poterya znachimosti
public:
  explicit underflow_error( const string &what_arg );
```

Ichki hisoblashda xatolik kelib chiqsa, funksiya **range_error** qaytarishi mumkin, arifmetik to‘lib ketish yuz bersa, **overflow_error** qaytarishi mumkin va **underflow_error** – qiymat yo‘qolsa.

exception sinfi quyidagi sinflar uchun ham ajdod sinfdir: **new()** operatori xotira ajratolmasa, yaratadigan **bad_alloc** sinfi, shuningdek, **dynamic_cast** operatori ko‘rsatkichli varianti bajarib bo‘lmaganda yaratiladigan **bad_cast** sinfi.

Oldindan kiritilgan sinflardan foydalanish uchun dasturga **<stdexcept>** sarlavhali faylini ulash lozim. G‘ayrioddiy holat haqidagi ma'lumot **string** turidagi **eObj** obyektida saqlanadi. Bu ma'lumotni olish uchun **what()** funksiyasidan foydalanish lozim.

Misol:

```
#include <iostream>
#include <stdexcept>
using namespace std;
class calc{
public:
static int divide(int a, int b) {
if (b == 0) {
string eObj ="DivideByZero!";
throw runtime_error(eObj);
return false;
}
return a/b;
};
};
int main (){
try {
calc::divide(5,0);
}
catch (runtime_error &excep) {
```

```
cerr << excep.what()  << '\n';
}
system("pause");
return 0;
}
```
Natija:
DivideByZero!

Quyida “Kim oxirgi?” o‘yini dasturi keltirilgan. O‘yin mazmuni shu-ki, M buyumlardan K dan ortmagan buyumlar tanlanadi. Kim oxirgi buyumni olsa, yutadi.

```
#include <iostream>
#include<iostream>
using namespace std;
class Win{
int n;
public:
Win(int k) {n=k;};
int get_number(){return n;}
};
class Error{
int n;
int k;
public:
Error(int k1,int k2) {n=k1;k=k2;};
int get_number(){return n;}
int get_value(){return k;}
};
class Play{
int max;
int step;
int number;
public:
Play(int max1,int step1){
max=max1;step=step1;number=0;
}
void Step (int k){
if((k<1) ||(k>step)) throw(Error(number,k));
if(k>=max) throw(Win(number));
max-=k;
number=1-number;
}
};
```

```
int main(){
int k;
int m;
Play pa(10,5);
try{
while(1){
cin>>m;
pa.Step (m);
}
}
catch(Win a){
cout<<"Ura! "<<a.get_number()<<" o'yinchi yutdi";
}
catch(Error a){
cout<<a.get_number()<<" o'yinchi "<<a.get_value()<<" hato yurdi";
}
return 0;
}
```

Nazorat savollari:

1. Istisnolarni nima uchun generatsiya va qayta ishlash kerak?
2. Istisno generatsiyasi sintaksisini keltiring.
3. Istisnoni qayta ishlash sintaksisini keltiring.
4. Istisnolar bilan qanday operatorlar bog‘liq?
5. Kutilmagan istisnolarni qayta ishlash usullari.
6. Istisnoni generatsiya qiluvchi funksiya sintaksisini keltiring.
7. Stekni orqaga qaytarish tushunchasi.
8. Standart istisnolar ierarxiyasi.
9. Mantiqiy xatolar.
10. Bajarilish vaqti xatolari.

Misollar:

1. Istisnolar ko‘zda tutilgan chiziqli tengamani yechish funksiyasini tuzing.

2. Istisnolar ko‘zda tutilgan kvadrat tengamani yechish funksiyasini tuzing.

3. Massiv sinfi shabloni yaratilsin. Indeks chegaradan chiqqanda istisno generatsiya qilinsin.

4. Stek sinfi shablonini yarating. Bo‘sh joy o‘chirilganda yoki to‘la stekka element qo‘shishga urinilganda. istisno generatsiya qilinsin.

5. Navbat sinfi shablonini yarating. Bo‘sh joy o‘chirilganda yoki to‘la navbatga element qo‘shishga urinilganda istisno generatsiya qilinsin.

KO‘RSATKICHLAR VA SINFLAR

Obyektlarga ko‘rsatkichlar

Obyektlarga ko‘rsatkichlar ta'rifi. Strukturalar kabi obyektlar aniqlangandan so‘ng. shu obyektlarga ko‘rsatkichlar belgilash mumkin.

Obyektning umumiy elementlariga murojaat uchun -> amalini yoki ism almashtirish va nuqta amalidan foydalanish mumkin.

Misol:

```
#include <iostream>
using namespace std;
class complex{
float x;
float y;
public:
complex(float x1=0,float y1=0):x(x1),y(y1){};
void show(){
cout<<"x="<<x<<" y="<<y<<endl;
}
};
int main(){
complex a(1.0,2.0);
complex *pa=&a;
pa->show();
complex b;
complex *pb=&b;
(*pb).show();
return 0;
}
```

Dastur bajarilishi natijasi:

x=1.0 y=2.0
y=0.0 y=0.0

Virtual usullar va ko‘rsatkichlar

Virtual usullar va ko‘rsatkichlar. Asos turdagi ko‘rsatkichga hosila sinfi turidagi obyekt adresini qiymat sifatida berish mumkin, lekin teskarisini bajarib bo‘lmaydi.

Avlod sinfi obyektiga adres qiymat sifatida berilgan ajdod sinf turidagi ko‘rsatkich orqali avlodda qo‘shimcha yuklangan usulni chaqirishga e'tibor berish lozim. Agar funksiya novirtual bo‘lsa ajdod sinf usuli, virtual bo‘lsa avlod sinf usuli chaqiriladi.

Shunday qilib, ajdod sinf turidagi ko‘rsatkich orqali virtual usul chaqirish natijasi shu ko‘rsatkich qiymati ya'ni chaqiriq bajarilayotgan obyekt turi bilan aniqlanadi.

Qaysi virtual funksiyani chaqirish ko‘rsatkich turiga emas, shu ko‘rsatkich qaratilgan (dastur bajarilish jarayonida) obyekt turiga bog‘liq.

Quyidagi misolda ajdod sinf va avlod sinf bir xil usulga ega bo‘lgan hol ko‘rilgan:

```
#include <iostream>
using namespace std;
class base{
public:
virtual void print(){cout<<"\nbase";}
};
class dir:public base{
public:
void print(){cout<<"\ndir";}
};
int main(){
base B;
dir D;
base *bp=&B;
bp->print(); // base
bp=&D;
bp->print(); // dir
return 0;
}
```

Natija:

base

dir

Bu misolda avlod sinfi obyektiga adres qiymat sifatida berilgan ajdod sinf turidagi ko‘rsatkich yoki ilova orqali avlodda qo‘shimcha yuklangan usulni chaqirishga e'tibor berish lozim. Agar funksiya novirtual bo‘lsa ajdod sinf usuli, virtual bo‘lsa avlod sinf usuli chaqiriladi.

Shunday qilib, ajdod sinf turidagi ko‘rsatkich orqali virtual usul chaqirish natijasi shu ko‘rsatkich qiymati, ya'ni chaqiriq bajarilayotgan obyekt turi bilan aniqlanadi.

Qaysi virtual funksiyani chaqirish ko‘rsatkich turiga emas, shu ko‘rsatkich qaratilgan (dastur bajarilish jarayonida) obyekt turiga bog‘liq.

Virtual destruktorlar. Asos sinf obyekti ko‘rsatkichiga avlod sinf obyekti qiymat sifatida berilgan bo‘lsin. Agar asos sinf virtual usulga ega bo‘lsa va ko‘rsatkichga **delete** amali qo‘llansa, faqat ajdod sinf obyekti o‘chiriladi, buni oldini olish uchun asos sinf destruktori virtual bo‘lishi lozim.

Misol:

```
#include<iostream>
using namespace std;
class base{
public:
virtual void print(){cout<<"\nbase";}
virtual ~base (){ cout<<"\nbase delete";}
};
class dir : public base{
public:
virtual void print(){cout<<"\nbase";}
~dir (){cout<<"\ndir delete";}
};
int main(){
dir a;
base*pa=&a;
pa->print();
base *pb = new dir;
delete pb;
cin.get();
return 0;
}
```

Natija:

base
dir delete

base delete

Turlar haqida dinamik ma'lumot

RTTI mexanizmi. RTTI (Run-Time Type Information) mexanizmi dasturni bajarish jarayonida obyektlar turini aniqlash va hatto o‘zgartirishga imkon beradi. Buning uchun asosan **dinamic_cast** va **typeid** operatorlaridan foydalaniladi. Bu imkoniyatlardan foydalanish uchun asos sinf va uning bir necha vorislari mavjud bo‘lishi lozim. Asos sinf polimorf bo‘lishi kerak, ya'ni juda bo‘lmasa, bitta virtual funksiyaga ega bo‘lishi kerak.

Operator dinamic_cast. Operator **dinamic_cast** sinflari umumiy asos sinfga ega obyektlar turlarini tekshirishga imkon beradi:

```
#include <iostream>
using namespace std;
class base {
public:
virtual void print(){
cout<<"\nbase";
}
};
class dir1: public base {
public:
void print(){
cout<<"\ndir1";
}
};
class dir2: public base {
public:
void print(){
cout<<"\ndir2";
}
};
bool isDir1(base*pr){
dir1*p1;
if (p1=dynamic_cast<dir1*>(pr)) return true;else return false;
}
int main(){
dir1* D1=new dir1;
dir2*D2=new dir2;
if(isDir1(D1)) cout<<"komponent dir1"<<endl;
else cout<<"not komponent dir1"<<endl;
```

```
if(isDir1(D2)) cout<<"komponent dir2";
else cout<<"not komponent dir2";
return 0;
}
```

Bu misolda **dynamic_cast** operatori **pr** ko‘rsatkich turini **dir1** turiga keltirishga urinadi.

typeid operatori. typeid operatori obyekt haqida to‘laroq ma'lumot olish, masalan, obyekt turi nomini aniqlashga imkon beradi:

```
#include <iostream>
using namespace std;
class base {
public:
virtual void print(){
cout<<"\nbase";
}
};
class dir1: public base {
public:
void print(){
cout<<"\ndir1";
}
};
class dir2: public base {
public:
void print(){
cout<<"\ndir2";
}
};
void displayName (base*pr){
cout<<"pointer on object ";
cout<<typeid(pr).name()<<endl;
};
int main(){
dir1* D1=new dir1;
displayName(D1);
dir2*D2=new dir2;
displayName(D2);
return 0;
}
```

Bu misolda funksiya displayName uzatilgan obyekt ismini aks ettiradi.

Konstruktor va Destruktor

Destruktor. Sinfning biror obyekti uchun ajratilgan xotira obyekt yo‘qotilgandan so‘ng bo‘shatilishi lozimdir. Sinflarning maxsus komponentalari **destruktorlar**, bu vazifani avtomatik bajarish imkonini yaratadi. Destruktorni standart shakli quyidagicha:

~ <sinf_nomi> ( ) {<destruktor tanasi>}

Destruktor parametri yoki qaytariluvchi qiymatga ega bo‘lishi mumkin emas (hatto **void** turidagi). Dastur obyektni o‘chirganda destruktor avtomatik chaqiriladi.

Boshqa obyektlar egallagan xotirani bo‘shatmoqchi bo‘lsak, destruktorni ochiq aniqlash lozim.

Nusxa olish konstruktori. Ko‘rsatilmagan holda **T::T(const T&)** ko‘rinishdagi nusxa konstruktori yaratiladi, bu yerda T – sinf ismi. Har gal sinfga tegishli obyektlarni nusxasi olinayotganda nusxa konstruktori chaqiriladi. Xususan:

a) obyekt funksiyaga qiymati bo‘yicha uzatilsa;

b) funksiya qiymatlarini qaytaruvchi vaqtinchalik obyektlarni yaratishda;

v) boshqa obyektni initsializatsiyalash uchun obyektdan foydalanishda.

Agarda sinfda aniq bir nusxalash konstruktori ochiq ko‘rsatilgan bo‘lmasa, unda yuqorida aytib otilgan uch holatdan bittasi mavjud bo‘lsa, obyektni nusxalash bit bo‘yicha amalga oshiriladi. Lekin bit bo‘yicha nusxalash har doim ham adekvat bo`lmaydi. Xuddi shu hollarda, xususiy nusxalash konstruktorini belgilamoq lozim.

Initsializatsiya masalasini hal qilish uchun nusxa olish konstruktorini kiritish kerak.

Qiymat berish va initsializatsiya. Qiymat berish va initsializatsiya turli amallardir. Ayniqsa, destruktor aniqlanganda bu muhimdir. Biror sinf turidagi obyektni insializatsiya qilish nusxa olish konstruktori yordamida amalga oshiriladi.

Dinamik xotira konstruktor tomonidan yaratilib, destruktor bilan o‘chirilganda, muammo tug‘ilishi mumkin. Bir obyekt ikkinchisiga qiymat berish natijasida biri o‘chirilib, ikkinchisi nusxasi bilan almashtiriladi. Funksiyadan

chiqishda ikki obyekt uchun ham destruktor chaqiriladi va bitta satr ikki marta o‘chiriladi. Bu muammoni hal qilish uchun qiymat berish amalini qo‘shimcha yuklash lozim.

Natijada sinf quyidagi ko‘rinishga keladi:

```
#include <iostream>
using namespace std;
class Person{
char * name;
int  age;
public:
// Parametrik konstruktor
Person (char* st,int N):age(N){
int k;
k=strlen(st);
name= new char[k+1];
strcpy(name,st);
}
Person(){
name= NULL;age=0;
}
//Nusxa olish konstruktori
Person(const Person& st){
int len=strlen(st.name);
name=new char[len+1];
strcpy(name,st.name);
age=st.age;
}
// Qiymat berish amali qo‘shimcha yuklash
Person& operator=(const Person& a){
if (this !=&a) {   // s=s bo‘lsa xavfli
delete name;
int len=strlen(a.name);
name = new char[len];
strcpy(name,a.name);
age=a.age;
}
return *this;
}
int GetAge (void){
return age;
}
char * GetName (){
return name;
```

```
}
// Destruktor
~Person(){
delete[] name;
}
};
int main(){
Person worker("Happy Jamsa", 51);
cout << "Ism: " << worker.GetName()<< endl;
cout << "Yosh: " << worker.GetAge()<< endl;
return 0;
}
```

Natija:
Ism: Happy Jamsa
Yosh: 51

Vektor shabloni. Sinf obyektlari bilan ishlash uchun **vector** qo‘shimcha yuklangan shablon sinfi sodda ko‘rinishi berilgan. Bu shablonda paramerli va parametrsiz konstruktor, nusxa olish konstruktori hamda qo‘shimcha yuklangan qiymat berish amali ta'rifi keltirilgan. Bundan tashqari indeks bo‘yicha tekshirishsiz murojaat **[]** amali, tekshirishli **at()** usuli hamda hajmni hisoblash **size()** usullari kiritilgan.

Dastur:

```
#include <iostream>
using namespace std;
template<class T>
class vector{
T* data;
int len;
public:
// Parametrik konstruktor
vector(int k){
len =k;
data = new T[len];
}
//Nusxa olish konstruktori
vector(const vector& st);
// Qiymat berish amali qo‘shimcha yuklash
vector& operator=(const vector& a);
// Indeks bo‘yicha murojaat
T& operator[](int i){
```

```
return data[i];
}
// Tekshiruvchi indeks bo‘yicha murojaat
T& at(int i){
if (i<len) return data[i];
}
// Vektor hajmini qaytarish
int size(){
return len;
}
// Destruktor
~vector() {
delete []data;
}
};
//Nusxa olish konstruktori
template<class T >
vector <T >:: vector(const vector& st){
len=st.len;
data=new T[len];
for (int i = 0; i < len; i++) data[i]=st.data[i];
}
// Qiymat berish amali qo‘shimcha yuklash
template<class T >
vector<T>& vector<T>:: operator=(const vector& a){
if (this !=&a) {   // s=s bo‘lsa xavfli
delete[] data;
len=a.len;
data = new T[len];
for (int i = 0; i < len; i++) data[i]=a.data[i];
}
return *this;
};
int main(){
vector<int> B(3);
for (int i = 0; i < B.size(); i++) B[i]=i;
for (int i = 0; i < B.size(); i++) cout<<B[i]<<" ";
return 0;
}
```

Natija:

0 1 2

Universal stek va daraxt

Stek. Stek deb, shunday strukturaga aytiladi-ki, stekka kelib tushgan oxirgi elementga birinchi bo‘lib xizmat ko‘rsatiladi va stekdan chiqariladi. Mazkur ko‘rinishdagi xizmat ko‘rsatishni **LIFO** (*Last input-First output*, ya'ni oxirgi kelgan – birinchi ketadi) nomlash qabul qilingan. Stek bir tomondan ochiq bo‘ladi.

Universal stek har bir tuguni axborot qismdan va keyingi elementga ko‘rsatkichdan iborat strukturadir:

```
template<class T>
struct link{
T info;
link* next;
link(T a): info(a),next(NULL){}
};
```

Stek o‘zi alohida struktura sifatida kiritilgan.

Asosiy funksiyalar:

void pop() – stek oxiridagi elementni o‘chirish

void push(T val) – stek oxiriga element qo‘shish

T top() – stek oxiridagi axborot qismini qaytarish

int empty() – stek bo‘shligini tekshirish

int size (struct stack p) – stek elementlari soni

Misol:

```
#include <iostream>
using namespace std;
template<class T>
class stack{
//
struct link{
T info;
link* next;
link(T a):
info(a),next(NULL){}
};
//
void copy(const stack& st){
if(st.n==0){
n=0;beg=NULL;return;
}
link* rex=st.beg;
```

```
beg=new link(rex->info);
link* temp=beg;
for (int i = 1; i < st.n; i++){
rex=rex->next;
temp->next=new link(rex->info);
temp=temp->next;
}
n=st.n;
};
//
void dell(){
link* rex=beg;
link* temp;
while(rex!=NULL){
temp=rex;
rex=temp->next;
delete temp;
}
}
//
link* beg;
int n;
//
public:
stack(){
beg=NULL;
n=0;
};
void push(T a){
link* rex;
rex=new link(a);
if (n==0)
beg=rex;
else{
rex->next=beg;
beg=rex;
}
n++;
};
void pop(){
link * rex;
rex=beg;
if (n>0) {beg=beg->next;n--;}
if (rex!=NULL) delete rex;
};
```

```
T top(){
return beg->info;
};
bool empty(){
if(n==0) return false;
else return true;
}
int size(){
return n;
}
//
~stack(){
dell();
}
//Nusxa olish konstruktori
stack(const stack& st){
beg==NULL;n=0;link* rex=st.beg;
copy(st);
}
// Qiymat berish amali qo‘shimcha yuklash
stack& operator=(const stack& st){
if (this!=&st) {   // s=s bo‘lsa xavfli
dell();
copy(st);
}
return *this;
};
};
int main(){
stack<int> s1;
s1.push(1);s1.push(2);
stack<int> s(s1);
cout<<s.top()<<" ";s.pop();cout<<s.top();
return 0;
}
```

Natija:

2 1

Universal daraxt. Daraxt bu ajdod avlod munosabatlari bilan bog‘langan tugun deb ataluvchi elementlar ierarxik strukturasidir. Hech qanday ajdodga ega bo‘lmagan yagona tugun ildiz deb ataladi. Hech qanday avlodga ega bo‘lmagan tugunlar barglar deyiladi.

Ikkilik binar daraxtda har bir tugun bir yoki ikki nasl-tugun bilan bog‘liq bo‘ladi. Agar binar daraxt bo‘sh bo‘lmasa, bitta ildiz va chap hamda o‘ng ostki daraxtga ega bo‘ladi. Binar izlash daraxti bo‘sh bo‘lmasa quyidagi xossaga ega: har bir tugun shu tugun ildiz bo‘lgan chap ostki daraxt hamma elementlaridan katta va o‘ng ostki daraxt hamma elementlaridan kichik. Bunday daraxtlarda izlash tezligi: $O(n) = C \bullet \log_2 n$ (eng yomon holda $O(n) = n$).

Asosiy funksiyalar:

void insert (T val) – daraxtga element qo‘shish

int find(T val) – berilgan qiymatli element mavjudligini aniqlash

int empty() – daraxt bo‘shligini tekshirish

int size (struct stack p) – daraxt elementlari soni

Dastur matni:

```
#include <iostream>
using namespace std;
template<class T>
class sbtree{
struct sbtree_node{
T info;
struct sbtree_node* lchild;
struct sbtree_node* rchild;
};
//
sbtree_node* creat_sbtree_node(T val){
sbtree_node* pn;
pn=new sbtree_node();
pn->info=val;
pn->lchild=NULL;
pn->rchild=NULL;
return pn;
};
//
void dell(sbtree_node* beg){
if (beg==NULL) return;
dell(beg->lchild);
dell(beg->rchild);
delete beg;
}
//
sbtree_node* copy(sbtree_node* t){
```

```
if (t==NULL) return t;
sbtree_node* pn=new sbtree_node();
pn->info=t->info;
pn->lchild=copy(t->lchild);
pn->rchild=copy(t->rchild);
return pn;
}
//
sbtree_node* beg;
int len;
//
public:
//
void insert(T val){
char c;
sbtree_node* p1;
sbtree_node* p2;
if(len==0){
beg=creat_sbtree_node(val);
len=1;
return;
}
p1=beg;
while(p1!=NULL){
if (p1->info==val) return;
p2=p1;
if(p1->info>val){
c='l';
p1=p1->lchild;
continue;
}
if(p1->info<val){
c='r';
p1=p1->rchild;
continue;
}
}
p1=creat_sbtree_node(val);
if (c=='l') p2->lchild=p1;else p2->rchild=p1;
len++;
}
int find(T val){
sbtree_node* p1;
if(len==0) return 0;
p1=beg;
```

```
while(p1!=NULL){
if(p1->info==val) return 1;
if(p1->info>val) {p1=p1->lchild;continue;}
if(p1->info<val) {p1=p1->rchild;continue;}
}
return 0;
}
//
int size(){
return len;
}
int empty(){
if (size>0) return 0;else return 1;
}
//
sbtree(){
beg=NULL;
len=0;
}
//Nusxa olish konstruktori
sbtree(const sbtree& st){
beg=NULL;len=0;sbtree_node* rex=st.beg;
copy(st);
}
// Qiymat berish amali qo‘shimcha yuklash
sbtree& operator=(const sbtree& st){
if (this!=&st) {   // s=s bo‘lsa xavfli
dell();
beg=copy(st);
}
return *this;
};
};
int main(){
int i,k;
int a[]={3,15,2,4,15};
sbtree<int> sp;
for(i=0;i<5;i++)
sp.insert(a[i]);
k=sp.find(0);
cout<<k<<endl;
for(i=0;i<5;i++){
k=sp.find(a[i]);
cout<<k<<" ";
}
```

```
return 0;
}
```
Natija:
0
1 1 1 1 1

Nazorat savollari:

1. Obyektlarga ko‘rsatkichlar ta'rifi.
2. Ko‘rsatkich orqali sinf komponentlariga murojaat usullari.
3. Qanday hollarda ko‘rsatkichlardan foydalaniladi.
4. Virtual usullar va ko‘rsatkichlar.
5. Destruktor vazifasi.
6. Destruktorni nima uchun qo‘shimcha yuklab bo‘lmaydi?
7. Nusxa olish konstruktorini qanday holda oshkor ta'riflash lozim?
8. Qiymat berish amalini qanday holda oshkor ta'riflash lozim?
9. Dinamik informatsion struktura deb qanday strukturaga aytiladi?
10. Qo‘shimcha **link** strukturasi nima uchun kiritilgan?

Misollar:

1. Abiturient (ismi, tug‘ilgan yili, yig‘ilgan ball, attestat o‘rta bali) sinfini yarating. Sinfga konstruktor, destruktor va nusxa olish konstruktori va qo‘shimcha yuklangan qiymat berish amali kiritilsin.

2. Xodim (ismi, lavozimi, tug‘ilgan yili, oyligi) sinfini yarating. Sinfga konstruktor, destruktor va nusxa olish konstruktori va qo‘shimcha yuklangan qiymat berish amali kiritilsin.

3. Mamlakat (nomi, poytaxt, axoli soni, egallagan maydoni) sinfini yarating. Sinfga konstruktor, destruktor va nusxa olish konstruktori va qo‘shimcha yuklangan qiymat berish amali kiritilsin.

4. Navbat sinfi shabloni yarating. Navbatga kelib tushgan birinchi elementga birinchi bo‘lib xizmat ko‘rsatiladi va navbatdan chiqariladi. Mazkur ko‘rinishdagi

xizmat ko‘rsatishni **FIFO** (*First input-First output*, ya'ni birinchi kelgan – birinchi ketadi) nomlash qabul qilingan.

5. Daraxt sinfi shabloniga elementni o‘chirish usulini qo‘shing.

Dinamik sinflar

Vektor konteyner sinfi

Konteynerlar (**containers**) – bu boshqa elementlarni saqlovchi obyektlardir. Standart kutubxonada **vector** – vektor konteyneri dinamik massiv sifatida aniqlanadi. Bu konteynerdan foydalanish uchun **<vector.h>** sarlavhali faylni ulash lozim. Massiv elementlariga indeks orqali ruxsat beriladi.

Vektorda saqlanadigan ixtiyoriy obyekt uchun ko‘zda tutilgan konstruktor aniqlash zarur. Bundan tashqari, obyekt uchun < va == operatorlar aniqlanishi lozim.

Turlar. Usullar ta'rifida quyidagi turlar ishlatiladi:

value_type – element turi.

reference – elementga ilova.

const_reference – elementga konstanta ilova.

iterator – iterator turi.

const_iterator – konstanta iterator turi.

size_type – ishorasiz butun turga mos keladi.

Konstruktorlar

Ixtiyoriy konteyner-sinfi ko‘zda tutilgan konstruktor va destruktor hamda nusxalovchi konstruktorga ega.

Masalan, vektor konteyner-sinfining konstruktori va destruktori:

vector() bitta ham elementga ega bo‘lmagan bo‘sh vektorni yaratadi;

explicit vector (const vector & c) nusxalovchi konstruktorni yaratadi;

explicit vector (size_type n) ko‘zda tutilgan konstruktor orqali inizializatsiya qilinuvchi n elementli vektorni yaratadi;

explicit vector (size_type n, const T& value) value elementning n nusxalari yordamida initsializatsiya etilgan vektorni yaratadi;

~ vector () destruktor;

vector (InputIterator first, InputIterator last) interval hamma elementlari orqali initsializatsiya qilinuvchi vektorni yaratadi.

Amallar

Qiymat berish amali qayta yozilayotgan elementlar uchun qiymat berish operatorini, qo‘shilayotgan elementlar uchun nusxalash konstruktorini va o‘chirilayotgan elementlar uchun destruktorni chaqiradi.

konteyner& operator=(const konteyner& c)

Solishtirish amali ikki konteyner solishtirish natijasini qaytaradi. Ikki konteyner teng hisoblanadi, agar elementlari soni teng bo‘sa va ular bir xil pozitsiyalarda joylashgan bo‘lsa. Kichik munosabati leksikografik kriteriy bo‘yicha tekshiriladi. Konteyner obyektlari uchun tegishli amallar aniqlangan bo‘lsa, hamma konteynerlar uchun aniqlangandir.

bool konteyner operator op (konteyner &c1, konteyner& c2)

Bu erda **op** quyidagi amallardan biri ==,!=,<,<=,>,>=.

Amali [] tekshirmasdan indeks bo‘yicha murojaat uchun xizmat qiladi.

reference konteyner::operator[](size_type idx)

const_reference konteyner::operator[](size_type idx) const

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
vector<int> v1(2,1);
int a[]={1,2};
vector<int> v2(a,a+2);
for(int i=0;i<2;i++) cout<<v2[i]<<" ";
cout<<endl<<(v1<v2);
return 0;
}
```

Natija:

1 2
1

Kattalik va hajm

Kattalik bu foydalanuvchi ishlatishi mumkin bo‘lgan ichki xotiradagi elementlar sonidir. Hajm ichki xotirada joylashtirish mumkin bo‘lgan elementlar sonidir.

Kattalik usullari:

size_type size() elementlar sonini qaytaradi.

bool empty() elementlar yo‘qligini tekshiradi. Ekvivalent size()==0.

size_type max_size() konteynerda saqlanishi mumkin bo‘lgan elementlar maksimal sonini qaytaradi. Qiymat realizatsiyaga bog‘liq. Umumiy holda indeks maksimal qiymatiga teng.

Kattalikni o‘zgartirish usullari:

void resize(size_type num) konteyner kengayganda yangi elementlar ko‘zda tutilgan konstruktor bilan initsializatsiya qilinadi.

void resize(size_type num, T value) konteyner kengayganda yangi elementlar value qiymati bilan initsializatsiya qilinadi.

Hajm bilan bog‘liq amallar:

size_type capacity() const xotira qayta taqsimlamasdan saqlash mumkin bo‘lgan elementlar sonini qaytaradi.

void reserve(size_type num) xotira num element uchun ajratiladi. Asosan bo‘sh vektor uchun elementlarni initsializatsiya qilmasdan xotira ajratish uchun ishlatiladi.

Misol. Xotira ajratish va hisoblash funksiyalaridan foydalanish:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
vector<int> a(4,2);cout<<a.size()<<" "<<a.capacity()<<endl;
vector<int> b; cout<<b.size()<<" "<<b.capacity()<<endl;
b.reserve(5); cout<<b.size()<<" "<<b.capacity()<<endl;
return 0;
}
```

Natija:
4 4
0 0
0 5
Oxirgi natija **size** va **capacity** funksiyalari orasidagi farqni ko‘rsatadi.

Vektor usullari

Elementlarga murojaat:

reference back() oxirgi elementga ilova qaytaradi.

const_reference back() const konstanta usul.

reference at(size_type id) indeks bo‘yicha tekshirishli murojaat. Agar indeks haqiqiy bo‘lmasa, **length_error** istisno yaratiladi.

const_reference at(size_type id) const konstanta usul.

Element vektor oxiriga qo‘shish va o‘chirish:

void push_back(const T& value) elementni oxiriga qo‘shish.

void pop_back()oxirgi elementni o‘chirish.

Qiymat berish usullari, shablon funksiyalar:

void assign(size_type num, const T& value) konteynerga value elementini num nusxasini qiymat sifatida berish.

void assign(Iterator first, Iterator last) diapazon elementlarini qiymat sifatida berish.

Almashtirish va tozalash usullari:

void swap(konteyner& x) almashtirish usuli.

void clear() hamma elementlarni o‘chiradi.

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main() {
vector<int> a,b;
for(int i=1;i<6;i++) a.push_back(i);
b.assign(a.begin(),a.begin()+3);
for(int i=0;i<b.size();i++) cout<<b.at(i)<<" ";
cout<<endl;
system("pause");
```

```
return 0;
}
```
Natija:
1 2 3

Misol:
```
#include<iostream>
#include<vector>
using namespace std;
int main(){
unsigned i;
vector<int> a(4,2);
vector<int> b(5,1);
for(i=0;i<a.size();i++)cout<<a[i]<<" ";
cout<<endl;
a.swap(b);
for(i=0;i<a.size();i++)cout<<a[i]<<" ";
return 0;
}
```
Natija:
2 2 2 2
1 1 1 1 1

Iterator qaytaruvchi usullar:

iterator begin() birinchi pozitsiyaga iterator qaytaradi.

iterator end() oxirgi pozitsiyaga iterator qaytaradi.

const_iterator begin() konstanta usul.

const_iterator end()konstanta usul.

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
unsigned i;
const vector<int> a(4,2);
vector<int>:: const_iterator p;
for(p=a.begin();p!=a.end();p++)cout<<*p<<" ";
return 0;
}
```
Natija:
2 2 2 2

Joylash va o‘chirish usullari: Joylash funksiyasi funksiya shablonidan iborat.

Iterator insert(iterator pos, const T& value) elementni berilgan pozitsiyaga joylash. Joylash pozitsiyasini qaytaradi.

void insert(Iterator pos, size_type num, const T& value) element berilgan sondagi nusxasini berilgan pozitsiyaga joylash.

void insert(Iterator pos, InputIterator first, InputIterator last) intervalni berilgan pozitsiyaga joylash.

Joylash usullaridan foydalanish:

a.insert(a.begin(), v); boshiga joylash

a.insert(a.begin()+i, num,v); berilgan indeksli pozitsiyaga joylash

a.insert(a.begin(),b.begin(),b.end()); boshiga hamma elementlarni joylash

O‘chirish usullari. Element uchun destruktor chaqiradi.

Iterator erase(Iterator pos) berilgan pozitsiyadagi elementni o‘chirish. O‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

Iterator erase(Iterator first, Iterator last) interval elementlarini o‘chirish. O‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

a.erase (a.begin()); bosh elementni o‘chirish.

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
vector<int> v;
for(int i=0;i<5;i++) v.insert(v.begin(),i);
v.insert(v.begin(),v.begin(),v.end());
for(int i=0;i<v.size();i++) cout<<v[i]<<" ";
v.erase(v.begin()+5,v.end());
cout<<endl;
for(int i=0;i<v.size();i++) cout<<v[i]<<" ";
return 0;
}
```

Natija:

```
4 3 2 1 0 4 3 2 1 0
4 3 2 1 0
```

Ikki o‘lchovli dinamik massiv yaratishga misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
unsigned i,j,k,m;
typedef  vector<int> dint;
vector<dint> a(2);
a[0]=dint(2,1);
a[1]=dint(3,2);
m=a.size();
for(i=0;i<m;i++){
k=a[i].size();
cout<<endl;
for(j=0;j<k;j++)cout<<a[i][j]<<" ";
};
return 0;
}
```

Vektor funksiya parametri va qaytaruvchi qiymati sifatida:

```
#include<iostream>
#include<vector>
using namespace std;
vector<int> sum(vector<int> a,vector<int> b){
vector<int> c;
for(unsigned i=0;i<a.size();i++)
c[i]=a[i]+b[i];
return c;
}
int main(){
unsigned i;
vector<int> a(4,2);
vector<int> b(4,3);
vector<int> c=sum(a,b);
for(i=0;i<c.size();i++)cout<<c[i]<<" ";
return 0;
}
```

Vektor va funksiya shabloni:

```
#include<iostream>
#include<vector>
using namespace std;
template<class T>
void sum(vector<T> a,vector<T> b, vector<T> &c){
for(unsigned i=0;i<a.size();i++)
```

```
c[i]=a[i]+b[i];
}
int main(){
unsigned i;
vector<int> a(4,2);
vector<int> b(4,3);
vector<int> c(4);
sum(a,b,c);
for(i=0;i<c.size();i++)cout<<c[i]<<" ";
return 0;
}
```

Satr murakkab tur sifatida

String turi. Satrlar bilan ishlash uchun standart kutubxonaga kiruvchi **string** sinfidan foydalanish qulaydir.

Bu sinfdan foydalanish uchun quyidagi sarlavhali faylni ulash lozim:

#include <string>

Usullarni qo‘shimcha yuklash. Amaliy jihatdan hamma usullar satr **string&** turi va C satr **char*** turi uchun qo‘shimcha yuklangan. Shuning uchun hamma usullarda argument sifatida C satrni uzatish mumkin. Oxirgi simvol ishlatilmaydi. Bundan tashqari standart kutubxonada satrli konstanta **const char*** sifatida aniqlangan, shuning uchun hamma usullarda argument sifatida satrli konstantani uzatish mumkin.

Konstruktorlar. Sinf quyidagi konstruktorlarga va destruktorga ega.

string() ko‘zda tutilgan konstruktor. Bo‘sh satrni yaratadi.

string (const string & str) nusxalovchi konstruktor. Satrni boshqa satr asosida yaratadi. Bundan tashqari C satr yoki satr konstanta asosida yaratishi mumkin.

string(const string & str, size_type pos, size_t n=0) berilgan satr n simvollari bilan **pos** pozitsiyadan boshlab initsializatsiya qilish.

string(size_type n, char c) berilgan simvol n nusxasi bilan initsializatsiya qilish. Simvollar soni ko‘rsatilishi shart.

string (InputIterator first, InputIterator last) interval hamma elementlari orqali initsializatsiya qilinuvchi satrni yaratadi.

~ **string()** destruktor.

Amallar

Kiritish va chiqarish amallari:

istream& operator >>(istream& strm, const string& str) simvollarni **strm** oqimdan o‘qiydi.

ostream& operator << (ostream& strm, const string& str) simvollarni **strm** oqimga yozadi.

Oqimli sinflarda ma'lumotlarni satrma-satr o‘qish uchun **getline** usuli aniqlangan.

istream& getline(istream& strm, string st)

istream& getline(istream& strm, string st, char delim)

Solishtirish amallari:

bool operator *op*(const string& str1, const string& str2) satr yoki C satr bilan solishtirish. Parametr **op** quyidagi amallardan biri ==,!=,<,>,<=,>=. Solishtirish leksikografik tartibda amalga oshiriladi.

Murojaat amallari:

char& operator[] (size_type id) indeks bo‘yicha tekshirishsiz murojaat. Oxirgi simvoldan keyingi pozitsiya haqiqiy hisoblanmaydi.

char operator[] (size_type id) const konstanta usul. Oxirgi simvoldan keyingi pozitsiya haqiqiy hisoblanadi. Bu holda ko‘zda tutilgan konstruktor generatsiya qilgan qiymat qaytariladi.

Qiymat berish amallari:

string& operator= (const string& str) satrni qiymat sifatida berish.

string& operator= (char c) simvolni qiymat sifatida berish.

string& operator+= (const string& str) satrni yoki C satrni qiymat ulash. Oxirgi simvol hisobga olinmaydi.

string& operator+= (char c) simvolni ulash.

Konkatenatsiya:

string operator+(const string& str1, const string& str2) satr yoki C satr bilan konkatenatsiya.

string operator+(const string& str, char c) satr va simvol konkatenatsiyasi.

string operator+(char c, const string& str) simvol va satr konkatenatsiyasi.

Misol:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s1="ABC";
cout<<s1[0]<<endl;
string s2="DEF";
string s3;
s1+=s2;
cout<<s1<<endl;
if(s1<=s2) s3="Yes";else s3="No";
cout<<s3;
return 0;
}
```

Natija:

A
ABCDEF
Yes

Usullar: Usullarda **npos** natijasiz qidiruv yoki hamma qolgan simvollarni bildiruvchi maxsus qiymatdir. Ishorasiz butun bo‘lib, boshlang‘ich qiymati **-1**.

Kattalik usullari:

size_type size() const satrda simvollar sonini qaytaradi.

bool empty() satrni bo‘shlikka tekshirish.

size_type length() const satrda simvollar sonini qaytaradi.

size_type max_size() satrda saqlanishi mumkin bo‘lgan elementlar maksimal sonini qaytaradi.

Misol:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s1="Hello";
```

```
string s2;
cout<<s1.size()<<endl;
cout<<s2.size()<<endl;
if(s2.empty()) cout<<"Yes";else cout<<"No";
return 0;
}
```
Natija:
5
0
Yes

Joylash usullari:

string& insert(size_type posl, const string& str, size_type pos=0,

size_type n=0) joriy satrga **posl** pozitsiyadan boshlab satr **pos** pozitsiyasidan n ta simvolini joylashtiradi.

string& insert(size_type posl, size_type n, char c) joriy satrga **posl** pozitsiyadan boshlab n simvol nusxasini joylashtiradi.

Almashtirish usullari:

string& replace (size_type posl, size_type nl, const string& str,

size_type pos=0, size_type n=0) joriy satr **posl** pozitsiyasidan **nl** elementini satr **pos** pozitsiyasidan, **n** simvoli bilan almashtiradi.

string& replace(size_type posl, size_type n, char c) joriy satr **posl** pozitsiyasidan **nl** elementini **n** ta **c** simvoli bilan almashtiradi.

Misol:

```
#include<iostream>
using namespace std;
int main(){
string a;
cout<<a.insert(0,2,'b')<<endl;
string b("abca");
cout<<a.insert(0,b,2,2)<<endl;
cout<<a.replace(0,2,b,0,2)<<endl;
return 0;
}
```
Natija:
bb
cabb
abbb

Ostki satr, o‘chirish va almashtirish usullari:

string substr(size_type pos=0, size_type n=npos) const uzunligi **n** ga teng ostki satrni **pos** pozitsiyasidan qaytaradi.

string& erase(size_type pos=0, size_type n=npos) berilgan **n** elementni, **pos** pozitsiyadan boshlab o‘chiradi. Parametrsiz chaqirish satrni tozalashga olib keladi.

void swap(string& x) almashtirish usuli.

void clear() hamma simvollarni o‘chiradi.

Misol:

```
#include<iostream>
using namespace std;
int main(){
string b("abdefg");
string a=b.substr(2,2);
cout<<a<<endl;
cout<<b.erase(0,2)<<endl;
return 0;
}
```

Natija:

de

defg

O‘zgartirish usullari. C++ tilida C satrga satrni qiymat sifatida berish mumkin emas. Buning o‘rniga C satrga aylantirish usullari aniqlangan. Shuni hisobga olish lozim-ki, **null** ko‘rsatkich uzatish mumkin emas.

const char* s_str() satr nusxasini o‘z ichiga oluvchi C satrga ko‘rsatkich qaytaradi. S satrni o‘zgartirib bo‘lmaydi.

size_type copy(char* s, size_type n, size_type pos=0) const simvolli massiv **s** ga chaqirayotgan satr **n** elementini **pos** pozitsiyadan boshlab nusxalaydi. Nol-simvol massivga yozilmaydi. Usul nusxalangan elementlar sonini qaytaradi.

Misol:

```
#include<iostream>
using namespace std;
int main() {
string str( "fa.disney.com" );
int size = str.size();
```

```
for ( int ix = 0; ix < size; ++ix )
if ( str[ ix ] == '.' ) str[ ix ] = '_';
cout<<str.c_str()<<endl;
system("pause");
return 0;
}
```
Natija:
fa.disney.com

Izlash usullari. Izlash usullari qaytarayotgan qiymatni tekshirish uchun **size_type** turdan foydalanish lozim. Bu tur ishorasiz butun deb aniqlangan bo‘lsa ham, realizatsiyasi turli kompilyatorlarda farqlanishi mumkin.

size_type find(const string& str, size_type pos=0) const Chaqirayotgan satrga **str** satrni eng chap kirishini **pos** pozitsiyadan boshlab qidiradi. Agar satr kirmasa, **npos** qaytaradi.

size_type find(char c, size_type pos=0) const Chaqirayotgan satrga **s** simvol kirishini, **pos** pozitsiyadan boshlab qidiradi. Agar simvol kirmasa, **npos** qaytaradi.

size_type rfind(const string& str, size_type pos=0) const Chaqirayotgan satrga **str** satr kirishini, **pos** pozitsiyadan boshlab qidiradi.

size_type rfind(char c, size_type pos=0) const Chaqirayotgan satrga **S** simvol o‘ng tomondan birinchi kirishini, **pos** pozitsiyadan boshlab qidiradi.

size_type find_first_of(const string& str, size_type pos=0) const Chaqirayotgan satrga **str** satr ixtiyoriy simvolini chap tomondan birinchi kirishini, **pos** pozitsiyadan boshlab qidiradi.

size_type find_last_of(const string& str, size_type pos=0) const Chaqirayotgan satrga **str** satr ixtiyoriy simvolini o‘ng tomondan birinchi kirishini, **pos** pozitsiyadan boshlab qidiradi.

Misol:
```
#include<iostream>
using namespace std;
int main(){
string a("abca");
string b("bc");
cout<<a.find('a')<<" "<<a.rfind('a')<<endl;
cout<<a.find(b)<<" "<<a.rfind(b);
return 0;
```

```
}
```
Natija:
0 3
1 1

Solishtirish usullari. Solishtirish usullari mantiqiy qiymat emas, son qiymat qaytaradi.

int compare(size_type id, size_type n, const string& str, size_type sd=0,

size_type sn=0) const joriy satrni argument sifatida berilgan satr yoki satr qismi bilan solishtiradi.

Agar satr va argument teng bo‘lsa, nol qaytaradi.

Agar satr leksikografik kichik bo‘lsa, manfiy son qaytaradi.

Agar satr leksikografik katta bo‘lsa, musbat son qaytaradi.

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
string a("abc");
string b("bcd");
cout<<b.compare(a);
return 0;
}
```
Natija:
1

Qiymat berish usullari shablon funksiyalar.

string& assign(size_type num, const char& c) konteynerga **c** simvolni **num** nusxasini qiymat sifatida berish.

string& assign(const string& str, size_type pos=0, size_type n=0) joriy satrga satr qismini qiymat sifatida berish.

string& append(const string& str, size_type pos=0, size_type n=0) joriy satrga satr qismini ulash.

Misol.

```
#include<iostream>
using namespace std;
```

```
int main(){
string a;
cout<<a.assign(3,'b')<<endl;
string b("abca");
cout<<a.assign(b,0,2)<<endl;
cout<<a.append(b,0,2)<<endl;
return 0;
}
```
Natija:
bbb
ab
abab

Kattalikni o‘zgartirish usullari:

void resize(size_type num) satr kengayganda yangi elementlar ko‘zda tutilgan konstruktor bilan initsializatsiya qilinadi.

void resize(size_type num, char c) satr kengayganda yangi elementlar num ta c qiymati bilan initsializatsiya qilinadi.

Hajm bilan bog‘liq amallar:

size_type capacity() const xotira qayta taqsimlamasdan saqlash mumkin bo‘lgan elementlar sonini qaytaradi.

void reserve(size_type num) xotira **num** simvol uchun ajratiladi.

Misol:

```
#include<iostream>
#include<string>
using namespace std;
int main() {
string a("Hello");
cout<<a.size()<<" "<<a.length()<<endl;
cout<<a.capacity()<<" "<<a.max_size()<<endl;
a.reserve(7);
cout<<a.capacity()<<" "<<a.max_size();
return 0;
}
```
Natija:
5 5
5 1073741820
7 1073741820

Simvollarga murojaat:

reference back() oxirgi elementga ilova qaytaradi.

const_reference back() const konstanta usul.

reference at(size_type id) indeks bo‘yicha tekshirishli murojaat. Agar indeks haqiqiy bo‘lmasa, **length_error** istisno yaratiladi.

const_reference at(size_type id) const konstanta usul.

Element satr oxiriga qo‘shish va o‘chirish:

void push_back(const char& c) elementni oxiriga qo‘shish.

void pop_back() oxirgi elementni o‘chirish.

Iterator qaytaruvchi usullar:

iterator begin() birinchi pozitsiyaga iterator qaytaradi.

iterator end() oxirgi pozitsiyaga iterator qaytaradi.

const_iterator begin() konstanta usul.

const_iterator end() konstanta usul.

Iteratorlar bilan bog‘liq usullar:

string& insert(iterator pos, size_type n, char c) pos oldiga simvol **n** nusxasini joylash.

string& insert(iterator pos, char c) pos oldiga simvol joylash.

string& insert(iterator pos, InputIterator beg, InputIterator end) pos oldiga interval hamma elementlarini joylash.

string& replace(iterator beg,iterator end, const string& str,

size_type n=0) interval hamma simvollarini satr n simvollari bilan almashtirish

string& replace (iterator beg, iterator end, size_type n, char c) interval hamma simvollarini simvol n nusxasi bilan almashtirish

string& replace (iterator beg, iterator end, InputIterator newbeg, InputIterator newend) interval simvollari bilan almashtirish

string& append(InputIterator beg, InputIterator end) joriy satrga interval hamma simvollarini ulash.

string& assign(Iterator first, Iterator last) diapazon elementlarini qiymat sifatida berish.

Misol:

```
#include<iostream>
using namespace std;
int main(){
string b(“abdefg”);
string a;
cout<<a.assign(b.begin(),b.end()-1)<<endl;
cout<<a<<endl;
return 0;
}
```

Natija:
abdef
abdef

Misol:

```
#include<iostream>
using namespace std;
int main(){
string b("abdefg");
b.insert(b.begin(),'c');
cout<<b<<endl;
string a("klm");
a.insert(a.end(),b.begin(),b.begin()+3);
cout<<a<<endl;
return 0;
}
```

Natija:
cabdefg
klmcab

Nazorat savollari:

1. Vektor sinfidan nima maqsadda foydalaniladi?
2. Vektor sinfi amallarini.
3. Vektor sinfi usullarini ko‘rsating.
4. Vektor sinfi obyekti bo‘sh bo‘lishi mumkinmi?
5. Vektor sinfida iteratorlar bilan bog‘liq usullar.
6. Satr sinfi.
7. Satr sinfi usullari?
8. Satr sinfida iteratorlar bilan bog‘liq usullar.
9. Kattalik va hajm qanday farqlanadi?

10. Satrda izlash usullari xususiyatlari.

Misollar:

1. Berilgan vektor qat'iy o‘suvchi ekanligini tekshiruvchi funksiya tuzing va dasturda foydalaning.

2. Matritsani vektorga ko‘paytirish funksiyasini tuzing va dasturda foydalaning.

3. Berilgan satr telefon raqami ekanligini aniqlovchi funksiya tuzing va dasturda foydalaning.

4. Berilgan satr o‘zgaruvchi ekanligi tekshiruvchi funksiya tuzing va dasturda foydalaning.

5. Berilgan jumladan eng uzun so‘z ajratib oluvchi funksiya tuzing va dasturda foydalaning.

KONTEYNERLAR STANDART KUTUBXONASI

STL tarkibi

Kutubxona yadrosi uchta elementdan iborat: **konteynerlar**, **algoritmlar** va **iteratorlar**.

Konteynerlar (**containers**) – bu boshqa elementlarni saqlovchi obyektlar. Masalan, vektor, chiziqli ro‘yxat, to‘plam.

Assotsiativ konteynerlar (**associative containers**) kalitlar yordamida ularda saqlanadigan qiymatlarni tezkor olish imkonini yaratadi.

Har bir sinf – konteynerida ular bilan ishlash uchun mo‘ljallangan funksiyalar to‘plami aniqlangan. Masalan, ro`yxat elementlarni kiritish, chiqarish, va qo‘shish funksiyalarni o‘z ichiga oladi.

Algoritmlar (algorithms) konteyner ichidagilar ustidan amallar bajaradi. Konteyner ichidagilarni initsializatsiyalash, qidirish, saralash va almashtirish uchun algoritmlar mavjud. Ko‘p algoritmlar konteyner ichidagi elementlarni

chiziqli ro‘yxatini ifodalovchi ketma-ketlik (sequence) bilan ishlash uchun mo‘ljallangan.

Iteratorlar (iterators) – bu konteynerga nisbatan ko‘rsatkich sifatida bo‘lgan obyektlar. Ular massiv elementlariga ruxsat oluvchi ko‘rsatkichlar kabi, konteyner ichidagiga ruxsat olish imkoni beradi.

Iteratorlar

Iteratorlar bilan ko‘rsatkichlar kabi ishlash mumkin. Ularga *, inkrement, dekrement operatorlarni qo‘llash mumkin. Iterator turi sifatida har xil konteynerlarda aniqlangan iterator tur elon qilinadi.

Iteratorlarning beshta turi mavjud:

1. **Kiritish iteratorlar** (**input_iterator**) tenglik, nomini o‘zgartirish va inkrementa amallarni qo‘llaydi.

==, !=, *i, ++i, i++, *i++

Kiritish iteratsiyasining maxsus holati **istream_iterator** iborat.

2. Chiqarish iteratorlar (**output_iterator**) o‘zlashtirish operatorning chap tarafidan imkon bo‘lgan ismning o‘zgartirish va inkrement amallar qo‘llanadi.

++i, i++, *i=t, *i++=t

Chiqarish iteratsiyasining maxsus holati **ostream_iterator**.

3. **Bitta yo‘nalishdagi iteratorlar** (**forward_iterator**) kiritish/chiqarish amallarning barchasini qo‘llaydi, bundan tashqari chegarasiz o‘zlashtirishning imkonini beradi.

==, !=, =, *i, ++i, i++, *i++

4. Ikki yo‘nalishdagi iteratorlar (**biderectional_iterator**) **forward-**iteratorlarning barcha xususiyatlariga ega, bundan tashqari, konteynerni ikkita yo‘nalishi bo‘yicha o‘tish imkonini beradigan qo‘shimcha dekrementa (--i, i--, *i--) amaliga ega.

5. Ixtiyoriy ruxsatga ega bo‘lgan iteratorlar (**random_access_iterator**) biderectional-iteratorlarning barcha xususiyatlariga ega, bundan tashqari solishtirish va manzil arifmetikasi amallarni qo‘llaydi.

i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2

Shuningdek, STLda teskari iteratorlar (**reverse iterators**) qo‘llaniladi. Ketma-ketlikni teskari yo‘nalishda o‘tuvchi ikki yo‘nalishli yoki ixtiyoriy ruxsatga ega bo‘lgan iteratorlar teskari iteratoralar bo‘lishi mumkin.

Xotirani taqsimlovchilar, predikatlar va solishtirish funksiyalari

Konteynerlarga, algoritmlarga va STLdagi iteratorlarga qo‘shimcha bir nechta standart komponentalar ham qo‘llaniladi. Ulardan asoslari esa xotira taqsimlovchilar, predikatlar va solishtirish funksiyalaridir.

Har bir konteynerda uning uchun aniqlangan va konteyner uchun xotirani belgilash jarayonini boshqaradigan xotira taqsimlovchisi (**allocator**) mavjud.

Ko‘rsatilmagan holda esa xotira taqsimlovchisi **allocator** sinf obyektidir. Xususiy taqsimlovchini tavsiflash mumkin.

Ba'zi bir algoritmlar va konteynerlarda muxim turdagi predikat ataluvchi funksiyalar ishlatiladi. Predikatlar unar va binar bo‘lishi mumkin. U yoki bu qiymatni olish aniq shartlari dasturchi orqali aniqlanadi. Unar predikatlarning turi – **UnPred**, binar predikatlarning esa – **BinPred**. Argumentlar turi konteynerda saqlanuvchi obyektlar turiga mos.

Ikkita elementlarni solishtirish uchun binar predikatlarning maxsus turi aniqlangan. U **solishtirish funksiya** (**comparison function**) deyiladi. Agarda birinchi element ikinchidan kichik bo‘lsa, unda funksiya rost qiymatni qaytaradi. **Comp** tur funksiya turidir.

STL da obyekt-funksiyalar o‘ziga xos ahamiyatga ega.

Obyekt-funksiyalar – bu sinfda «kichik qavslar» **()** amali aniqlangan sinf nusxalari. Ba'zi bir hollarda funksiyalarni obyekt-funksiyalarga almashtirish qulay deb hisoblanadi. Obyekt-funksiya funksiya sifatida ishlatilsa, unda uni chaqirish uchun operator **()** operator ishlatiladi.

Konteynerlar

Konteyner ta'rifi. Konteynerlar deb, turli turdagi obyektlarni saqlash uchun mo‘ljallangan sinf shablonlariga aytiladi. Konteynerlar dinamik xotirani avtomatik boshqarishga imkon beradi. Konteyner xotirasi stekda yaratiluvchi tashqi va uyumda yaratiluvchi ichki qismga ega. Obyekt o‘chirilganda ichki xotira ham avtomatik o‘chiriladi. Bo‘sh konteynerdan elementni o‘chirish natijasi oldindan aniqlanmagan.

Hamma konteynerlar uchun leksikografik solishtirish amallari ==,!=,<,<=,>,>= va qiymat berish amali = aniqlangan.

Konteynerda saqlanuvchi har qanday obyekt uchun, ko‘zda tutilgan konstruktor, nusxa olish konstruktori va destruktor aniqlangan bo‘lishi kerak.

Turlar. Usullar ta'rifida ishorasiz turlar ishlatiladi.

value_type element turi.

reference elementga ilova.

const_reference elementga konstanta ilova.

iterator iterator turi.

const_iterator konstanta iterator turi.

size_type ishorasiz butun turga mos keladi.

Amallar. Qiymat berish amali qayta yozilayotgan elementlar uchun qiymat berish operatorini, qo‘shilayotgan elementlar uchun nusxalash konstruktorini va o‘chirilayotgan elementlar uchun destruktorni chaqiradi.

konteyner& operator=(const konteyner& c)

Solishtirish amali ikki konteyner solishtirish natijasini qaytaradi. Ikki konteyner teng hisoblanadi agar elementlari soni teng bo‘sa va ular bir xil pozitsiyalarda joylashgan bo‘lsa. Kichik munosabati leksikografik kriteriy bo‘yicha tekshiriladi. Konteyner obyektlari uchun tegishli amallar aniqlangan bo‘lsa hamma konteynerlar uchun aniqlangandir.

bool konteyner operator op (konteyner &c1, konteyner& c2)

Bu erda **op** quyidagi amallardan biri ==,!=,<,<=,>,>=.

[] amali tekshirmasdan indeks bo‘yicha murojaat uchun xizmat qiladi.

reference konteyner::operator[](size_type idx)

const_reference konteyner::operator[](size_type idx) const

Kattalik. Kattalik bu foydalanuvchi ishlatishi mumkin bo‘lgan ichki xotiradagi elementlar sonidir. Kattalik usullari:

size_type size() const elementlar sonini qaytaradi;

bool empty() const elementlar yo‘qligini tekshiradi. Ekvivalent size()==0.

size_type max_size() const konteynerda saqlanishi mumkin bo‘lgan elementlar maksimal sonini qaytaradi. Qiymat realizatsiyaga bog‘liq. Umumiy holda indeks maksimal qiymatiga teng.

Almashtirish va o‘chirish usullari.

void swap(konteyner& x) almashtirish usuli;

void clear() elementlarni destruktor chaqirish yo‘li bilan o‘chiradi.

Iterator usullari. Har bir konteyner elementlarni qayta ishlashga imkon beruvchi o‘z iterator turiga ega. Obyekt iteratorlar quyidagicha yaratiladi:

Konteyner<tur>::iterator pos;

Iterator qaytaruvchi usullar:

iterator begin() birinchi pozitsiyaga iterator qaytaradi.

iterator end() oxirgi pozitsiyaga iterator qaytaradi.

const_iterator begin() konstanta usul.

const_iterator end()konstanta usul.

Misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main() {
vector<int> a(3,1);
vector<int> b(4,2);
a.swap(b);b.clear();
cout<<a.size()<<" "<<b.size();
cout<<endl;
system("pause");
return 0;
}
```

Natija:

Chiziqli konteynerlar

Vektor (**vector** sarlavhali fayl <**vector.h**>) - STL da dinamik massiv sifatida aniqlanadi. Massiv elementlariga indeks orqali ruxsat beriladi.

Vektor va satrning ichki xotirasi odatda elementlar uzluksiz ketma-ketligi ko‘rinishida bo‘ladi. Vektor va satrdan foydalanganda kattalik va hajm orasidagi farqni bilish lozim. Hajm bu obyekt uchun ajratilgan ichki xotirada saqlanishi mumkin bo‘lgan elementlar soni. Vektor va satr uchun hajm va kattalik mos kelmasligi mumkin, chunki dasturchi ishlatishi mumkin bo‘lgandan ko‘proq xotira ajratiladi.

Ikki yo‘nalishli tartib (**deque** sarlavhali fayl < **deque.h>**)– vektor kabi, ixtiyoriy ruxsat iteratorlarni qo‘llovchi ketma-ketlik ko‘rinishi. Bundan tashqari, u o‘zgarmas vaqtda boshida yoki oxirida kiritish va ochirish amallarni qo‘llaydi. O‘rtada kiritish va o‘chirish chiziqli vaqtni egallaydi. Xotirani boshqarishiga ishlov berish esa vektorlar kabi avtomatik ravishda bajariladi. Dek doimiy kattalikdagi bloklardan iborat.

Ro‘yxat (**list** sarlavhali fayl **<list.h>**) – ikki yo‘nalishli iteratorlarni qo‘llaydigan hamda kiritish va o‘chirish amallarni o‘zgarmas vaqtda ketma-ketlikni ixtiyoriy joyida bajaradigan, shuningdek, xotirani boshqarishiga avtomatik ravishda ishlov beruvchi ketma-ketlik ko‘rinishi. Vektorlar va ikki tarafli tartiblardan farqi shunda-ki, elementlar ro‘yxatiga tez va ixtiyoriy ro‘xsat qo‘llanmaydi, lekin ko‘pgina algoritmlarga esa ketma-ketlik ruxsat zarur.

Ichki xotira ixtiyoriy joylashgan elementlar majmuasidan iborat bo‘lib, har bir element oldingi va keyingi elementga ko‘rsatkichga ega.

Konstruktorlar. Hamma chiziqli konteynerlar quyidagi konstruktorlar va destruktorga ega:

konteyner () bitta ham elementga ega bo‘lmagan bo‘sh konteynerni yaratadi.

explicit konteyner (const konteyner & c) nusxalovchi konstruktor

explicit konteyner (size_type n) ko‘zda tutilgan konstruktor orqali inizializatsiya qilinuvchi n elementli vektorni yaratadi.

explicit konteyner (size_type n, const T& value) **value** elementning n nusxalari yordamida initsializatsiya etilgan vektorni yaratadi.

~ konteyner () destruktor. elementlarni o‘chiradi va xotirani bo‘shatadi

konteyner (InputIterator first, InputIterator last) interval hamma elementlari orqali initsializatsiya qilinuvchi vektorni yaratadi.

Konteynerda saqlanuvchi har qanday obyekt uchun ko‘zda tutilgan, konstruktor aniqlangan bo‘lishi kerak.

Umumiy usullar. Chiziqli konteynerlar umumiy usullari:

void push_back(const T& value) elementni oxiriga qo‘shish.

void pop_back() elementni oxiridan o‘chirish.

reference back() oxirgi elementga ilova qaytaradi

const_reference back()(size_type id) const konstanta usul.

reference at(size_type id) indeks bo‘yicha tekshirishli murojaat. Agar indeks haqiqiy bo‘lmasa **length_error** istisno generatsiya qilinadi.

const_reference at(size_type id) const konstanta usul.

void assign(size_type num, const T& value) berilgan value element num nusxasini konteynerga qiymat sifatida beradi

void assign(Iterator first, Iterator last) diapazonni qiymat sifatida beradi.

Kattalik va hajm. Kattalik – bu foydalanuvchi ishlatishi mumkin bo‘lgan ichki xotiradagi elementlar soni. Hajm – ichki xotirada joylashtirish mumkin bo‘lgan elementlar soni. Vektor va satr qo‘shimcha usullarga ega:

size_type capacity() const ichki xotira hajmini qaytaradi.

void reserve(size_type num) xotira **num** element uchun ajratiladi.

Amallar. Vektor, dek va satr uchun **[]** amali tekshirmasdan indeks bo‘yicha murojaat uchun xizmat qiladi.

reference konteyner::operator[](size_type idx)

const_reference konteyner::operator[](size_type idx) const

Qo‘shish, o‘chirish va murojjaat. Dek va ro‘yxat qo‘shimcha usullarga ega:

reference front() birinchi elementga ilova qaytaradi.

const_reference front(size_type id) const konstanta usul.

void push_front (const T& value) elementni boshiga qo‘shish.

void pop_ front () elementni boshidan o‘chirish.

void push_front(const T& value) boshiga element qo‘shish

void pop_front(const T& value) birinchi elementni o‘chirish.

Umumiy joylash va o‘chirish usullari. Joylash funksiyasi funksiya shablonidan iborat. O‘chirish usullari element uchun destruktor chaqiradi.

Iterator insert(iterator pos, const T& value) elementni berilgan pozitsiyaga joylash. Joylash pozitsiyasini qaytaradi.

void insert(Iterator pos, size_type num, const T& value) element berilgan sondagi nusxasini berilgan pozitsiyaga joylash.

void insert(Iterator pos, InputIterator first, InputIterator last) intervalni berilgan pozitsiyaga joylash.

Iterator erase(Iterator pos) berilgan pozitsiyadagi elementni o‘chirish. O‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

Iterator erase(Iterator first, Iterator last) interval elementlarini o‘chirish. O‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

Standart turli vektorga misol:

```
#include<iostream>
#include<vector>
using namespace std;
int main() {
vector<int> v;
unsigned i;
for(i=0;i<5;i++)v.push_back(i);
cout<<"size="<<v.size()<<endl;
for(i=0;i<5;i++)cout<<v[i]<<" ";
for(i=0;i<5;i++) v.pop_back();
cout<<"\nsize="<<v.size()<<endl;
system("pause");
return 0;
}
```

Natija:

```
size=5
0 1 2 3 4
size=0
```

Ro‘yxatga misol:

```
#include<iostream>
#include<list>
using namespace std;
int main(){
list<int> ll;
for(int i=0;i<6; i++) {
ll.push_back(i);
ll.push_front(i);
}
list<int>::iterator p;
for(p=ll.begin();p!=ll.end();p++) cout<<*p<<" ";
cout<<endl;
system("pause");
return 0;
}
```

Natija:
5 4 3 2 1 0 0 1 2 3 4 5

Ikki tomonlama ro‘yxatga misol:

```
#include<iostream>
#include<deque>
using namespace std;
int main(){
deque<int> ll;
for(int i=0;i<6; i++) {
ll.push_back(i);
ll.push_front(i);
}
for(int i=0;i<ll.size();i++) cout<<ll[i]<<" ";
cout<<endl;
system("pause");
return 0;
}
```

Natija:
5 4 3 2 1 0 0 1 2 3 4 5

Ro‘yxat usullari. Ro‘yxat ixtiyoriy murojaatni qo‘llamagani uchun **merge()**, **remove()**, **reverse()**, **sort()** va **unique()** algoritmlarini ro‘yxatga qo‘llamagan ma'qul. Buning o‘rniga shu algoritmlarga mos keluvchi usullar va **splice()** amali aniqlangan:

* **void list::merge(list& source)** ikki tartiblangan ro‘yxatni birlashtiradi;

* **void list::merge(list& source, CompFunc op)** ikki tartiblangan ro‘yxatni birlashtiradi;

* **void list::remove(const T& value)** berilgan qiymatdagi elementlarni o‘chiradi;

* **void list::remove_if(UnaryPredicate op)** berilgan shartga mos keluvchi elementlarni o‘chiradi. Ikkala usul obyektlar destruktorlarini chaqiradi;

* **void list::reverse()** elementlarni teskari tartibda joylashtiradi;

* **void list::sort()** ro‘yxat elementlarini tartiblaydi;

* **void list::sort(CompFunc op)** ro‘yxat elementlarini tartiblaydi;

* **void list::splice(iterator pos, list& source)** elementlarni bir ro‘yxatdan ikkinchisiga o‘tkazadi(1-shakl);

* **void list::splice(iterator pos, list& source, iterator sourcePos)** elementlarni bir ro‘yxatdan ikkinchisiga o‘tkazadi(2-shakl);

* **void list::splice(iterator pos, list& source, iterator sourceBeg, iterator sourceEnd)** elementlarni bir ro‘yxatdan ikkinchisiga o‘tkazadi(3-shakl);

* **void list::unique()** biketma-ket kelgan bir xil elementlardan bittasini qoldiradi;

* **void list::unique(BinaryPredicate op)** biketma-ket kelgan bir xil elementlardan bittasini qoldiradi.

Elementlarni bitta ro‘yxatdan ikkinchisiga ko‘chirish amali **splice()** uch shaklga ega: bitta elementni ko‘chirish, hamma elementlarni va diapazonni har gal joylash, pozitsiyasini ko‘rsatuvchi iterator uzatiladi, ko‘chirilayotgan elementlar uning oldiga joylashadi.

Ikki tomonlama ro‘yxatga misol:

```
#include<iostream>
#include<deque>
using namespace std;
int main(){
int ar1[]={3,1,2,5,3,7};
list<int> lst1(ar1,ar1+6);
lst1.sort();
int ar2[]={1,4,6,7};
list<int> lst2(ar2,ar2+4);
lst2.merge(lst1);
lst1.unique(EvenPair());
lst1.remove_if(pred);
```

```
system("pause");
return 0;
}
```
Natija:
5 4 3 2 1 0 0 1 2 3 4 5

Assotsiativ konteynerlar

Assotsiativ konteynerlar elementlarni logarifmik murakkablik bilan tez izlash uchun mo‘ljallangan. Assotsiativ konteynerlarda, elementlar tartiblangan holda keladi. Tartiblanganlik elementlar joylanganda va o‘chirilganda saqlanadi.

Assotsiativ konteynerlar odatda balansirlangan izlash daraxti shaklida realizatsiya qilinadi. Hamma assotsiativ konteynerlar ikki tomonlama iteratorlarni qo‘llaydi. Lekin iteratorlar o‘zlari ilova qilayotgan element qiymatini o‘zgartira olmaydi.

To‘plam (**set** sarlavhali fayl **< set.h>**) berilgan kriteriy bo‘yicha tartiblangan elementlar majmuasidir. To‘plam elementlari uchun “kichik”(<) amali aniqlangan bo‘lishi lozim.To‘plam qaytariluvchi qiymatlarga ega bo‘la olmaydi.

Multito‘plam (**multiset** sarlavhali fayl **<set.h>**) qaytariluvchi qiymatlarni saqlashga imkon beradi.

Karta (**map** sarlavhali fayl **<map.h>**) – (kalit, qiymat) juftliklaridan iborat bo‘lib, kalit bo‘yicha tez qiymat olishga imkon beradi. Assotsiativ konteyner map kalit turi uchun “<” amali majud bo‘lishini talab qiladi.

Kartada **map** kalit/qiymat juftliklari **pair** turidagi obyektlar ko‘rinishida saqlanadi.

Kartadagi kalit/qiymat juftliklarini **pair** sinfi konstruktorlari yordamida va **make_pair** funksiyasi yordamida yaratish mumkin.

Multikarta (**multimap** sarlavhali fayl **<map.h>**) – qaytariluvchi qiymatli kalitlarni saqlashga imkon beradi. Multikarta uchun indeks bo‘yicha murojaat amali aniqlanmagan. Karta va multikarta balansirlangan kalitli izlash daraxti shaklida realizatsiya qilinadi.

Iterator orqali kalitga va qiymatga murojaat qilish uchun **pos->first** va **pos-> second** ifodalaridan foydalaniladi. Bu ifodalarga mos ifodalar **(*pos).first** va **(*pos).second** ifodalaridir.

Konstruktorlar. Hamma assotsiativ konteynerlar quyidagi konstruktorlarga va destruktorga ega:

konteyner() bo‘sh konteyner yaratadi;

explicit konteyner (const konteyner & c) shu turdagi konteyner nusxasini yaratadi;

konteyner(InputIterator first, InputIterator last) interval hamma elementlari bilan initsializatsiya qilinuvchi konteyner yaratadi;

explicit konteyner (const CompFunk& op) tartiblash kriteriysi op bo‘lgan yangi bo‘sh konteyner yaratadi;

konteyner(InputIterator first, InputIterator last, const CompFunk& op) tartiblash kriteriysi **op** bo‘lgan va interval xama elementlari bilan initsializatsiya qilinuvchi konteyner yaratadi;

~ konteyner () hamma elementlarni o‘chiradi va xotirani tozalaydi.

Iterator bilan assotsiativ izlash usullari. Hamma usullar logarifmik murakkablikka ega. Karta va multikarta uchun T – kalit turi, to‘plam va multito‘plam uchun – qiymat turi.

size_type count(const T& value) const berilgan **value** qiymatli elementlar sonini qaytaradi. Murakkabligi – chiziqli. Karta va multikarta uchun T – kalit turi, to‘plam va multito‘plam uchun – qiymat turi.

iterator find(const T& value) berilgan **value** qiymatli birinchi element pozitsiyasini qidirish.

Karta uchun **cout<<a[elem]** shaklda izlab chiqish noqulay, chunki element mavjud bo‘lmasa, u avtomatik yaratiladi.

if(count(elem)) cout<<a[elem] samarasiz, chunki izlash ikki marta amalga oshiriladi.

Eng ma'qul usul:

```
map<string,int>::iterator p=find(elem);
cout<<(p->first)<<(p->second);
```

Elementlarni joylash usullari.

iterator insert(const T& value) konteynerga element joylash. Yangi element pozitsiyasini qaytaradi. Multito'plam va multikarta uchun. Elementlarga qaytarilishi mumkin bo'lgani uchun har doim muvaffaqiyatli bajariladi.

pair<iterator,bool> insert(const T& value) elementni konteynerga joylash. Yangi element pozitsiyasini qaytaradi va **true** qaytaradi, agar element joylangan bo'lsa, aks holda to'plam va karta uchun mavjud element pozitsiyasini va **false** qaytaradi.

Iterator insert(iterator pos, const T& value) elementni berilgan pozitsiyaga joylash. Joylash pozitsiyasini qaytaradi.

void insert(InputIterator first, InputIterator last) interval elementlarini joylash.

O'chirish usullari. Hamma shakllari elementlar uchun destruktor chaqiradi.

size_type erase(const T& value) berilgan qiymatli elementlarnir o'chiradi. O'chirilgan elementlar sonini qaytaradi. Multito'plam va multikarta uchun faqat birinchi elementni o'chirish uchun avval pozitsiyasini topib, topilgan pozitsiyadagi elementni o'chirish lozim.

void erase(Iterator pos) berilgan pozitsiyadagi elementni o'chiradi.

void erase(Iterator first, Iterator last) interval hamma elementlarini o'chiradi.

Joylash pozitsiyasini izlash

iterator lower_bound(const T& value) joylash birinchi pozitsiyasi;

const_iterator lower_bound(const T& value) const konstanta usul;

iterator upper_bound(const T& value) joylash oxirgi pozitsiyasi;

const_iterator upper_bound(const T& value) const konstanta usul;

pair<iterator, iterator> equal_range(const T& value) element juftlik sifatida joylanishi mumkin bo‘lgan birinchi va oxirgi pozitsiyani topadi.

Misol:

```
#include<iostream>
#include<set>
using namespace std;
int main() {
int b[]={1,4,2,4};
set<int> a(b,b+4);
//insert
set<int>::iterator p;
pair<set<int>::iterator,bool> pa=a.insert(3);
if(pa.second) cout<<*pa.first<<" insert!"<<endl;
else cout<<*pa.first<<" not insert!"<<endl;
//erase
a.erase(4);
//print
for(p=a.begin();p!=a.end();p++) cout<<*p<<" ";
cout<<endl;
system("pause");
return 0;
}
```

Natija:

3 insert!

1 2 3

Karta yordamida elementlar sonini hisoblashga misol:

```
#include <iostream>
#include<map>
using namespace std;
int main(){
string a[]={"al","bet","al"};
map<string,int> mt;
//count
for(int i=0;i<3;i++) mt[a[i]]++;
//print
map<string,int>::iterator p;
for(p=mt.begin();p!=mt.end();p++)
cout<<p->first<<" "<<p->second<<endl;
system("pause");
return 0;
}
```

Natija:

al 2
bet 1

Misol:

```
#include <iostream>
#include<map>
using namespace std;
int main(){
multimap<string,string> st;
st.insert(make_pair("alpha","1"));
st.insert(make_pair("betta","2"));
st.insert(make_pair("gamma","3"));
st.insert(make_pair("gamma","32"));
//find
multimap<string,string>::iterator where=st.find("alpha");
if (where!=st.end())
cout<<where->first<<" "<<where->second<<endl;
else cout<<"No";
//
multimap<string,string>::iterator p1,p2,p;
p1=st.lower_bound("gamma");
p2=st.upper_bound("gamma");
for(p=p1;p!=p2;p++)
cout<<p->first<<" "<<p->second<<endl;
system("pause");
return 0;
}
```

Natija:
alpha 1
gamma 3
gamma 32

Hosila konteynerlar

Hosila konteynerlar iteratorlar usullariga va **max_size** usuliga ega emas.

Konstruktorlar. Hamma hosila konteynerlar quyidagi konstruktorlarga va destruktorga ega:

konteyner() ko‘zda tutilgan konstruktor;

explicit konteyner (const konteyner & cont) nusxalovchi konstruktor;

~konteyner () destruktor.

Amallar:

konteyner& operator=(const konteyner& c) qiymat berish;

bool konteyner operator op (konteyner &c1, konteyner& c2) solishtirish. Bu yerda **op** amal ==,!=,<,<=,>,>=.

Kattalik usullari:

size_type size() const elementlar soni;

bool empty() const konteyner bo‘shligini tekshirish;

stek (stack sarlavhali fayl **<stack.h>**) birinchi kelgan – oxirgi ketadi, prinsipi bo‘yicha ishlaydi;

Sinf stack usullari;

void push(const value_type& elem) elementni stek boshiga joylash;

value_type& top() stek boshidagi elementga ilova qaytarish;

const value_type& top() const konstanta usul;

void pop() stek boshidagi elementni o‘chirish;

Navbat (queue sarlavhali fayl <queue.h>) ma'lumotlarni bir tomondan yozib ikkinchi tomondan o‘chirish uchun ishlatiladi.

Sinf queue usullari;

void push(const value_type& elem) elementni joylash;

value_type& front () birinchi elementni qaytarish;

const value_type& front() const konstanta usul;

value_type& back () oxirgi elementni qaytarish;

const value_type& back() const konstanta usul;

void pop() elementni o‘chirish.

Prioritetli navbat (priority_queue sarlavhali fayl **<queue.h>**) oddiy navbatdan farqi shu-ki, o‘qishda maksimal element tanlanadi. Shuning uchun har gal konteyner o‘zgarganda maksimal element konteyner boshiga keltiriladi. Agar prioritetli navbat dasturchi sinfi obyektlari uchun tashkil etilsa, bu sinfda < amali aniqlangan bo‘lishi kerak.

Konstruktorlar. Sinf quyidagi qo‘shimcha konstruktorlarga ega:

priority_queue(InputIterator first, InputIterator last) interval hamma elementlari bilan initsializatsiya qilinuvchi konteyner yaratadi;

explicit priority_queue (const CompFunk& op) berilgan op tartiblash kriteriysi bilan bo‘sh navbat yaratadi;

priority_queue(InputIterator first, InputIterator last,

const CompFunk& op) interval hamma elementlari bilan initsializatsiya qilinuvchi berilgan **op** tartiblash kriteriysi bilan konteyner yaratadi.

Sinf priority_queue usullari:

void push(const value_type& elem) elementni joylash;

const value_type& top() const maksimal elementni qaytarish;

void pop() elementni o‘chirish.

Misol:

```
#include <iostream>
#include<stack>
using namespace std;
bool checking(string a){
stack<char> st;
char c;
for(int i=0;i<a.size();i++){
c=a[i];
if (c!='('&&c!=')') return false;
if (st.size()==0||c=='(') {
st.push(c);continue;
}
if (st.top()=='(') st.pop();
else st.push(c);
};
if(st.size()==0) return true; else return false;
};
int main(){
string a("(())");
cout<<checking(a)<<endl;
system("pause");
return 0;
}
```

Natija:

1

Misol:

```
#include <iostream>
#include<queue>
using namespace std;
int main() {
queue<int > qt;
for(int i=1;i<6;i++) qt.push(i);
while(!qt.empty()){cout<<qt.front()<<" ";qt.pop();}
cout<<endl;
system("pause");
return 0;
}
```

Natija:
1 2 3 4 5

Misol:

```
#include <iostream>
#include<queue>
using namespace std;
int main() {
int a[]={1,3,2,7,5};
priority_queue<int > qt(a,a+5);
//print
while (!qt.empty()) {
cout<<qt.top()<<" ";
qt.pop();
}
cout<<endl;
system("pause");
return 0;
}
```

Natija:
7 5 3 2 1

Nazorat savollari:

1. Kutubxona yadrosi qanday elementlardan iborat?
2. Har qanday konteyner qanday konstruktorlarga ega?
3. Iteratorlar turlarini ko‘rsating.
4. Chiziqli konteynerlarga qaysi konteynerlar kiradi?
5. Nima uchun ro‘yxat konteyneriga qo‘shimcha usullar kiritilgan?
6. Assotsiativ massivlar qanday xususiyatlarga ega?
7. Elementlarga murojaat usullarini ko‘rsating.

8. Elementlarni o‘chirish usullarini ko‘rsating.

9. Konteyner hajmini o‘zgartirish uchun qanday usuldan foydalaniladi?

10. Hosila konteynerlar xususiyatlari.

Misollar:

1. Ro‘yxat yordamida har bir berilgan raqamdagi elementni o‘chirib oxirgi elementni aniqlash dasturini tuzing.

2. To‘plam yordamida xizmatchi bo‘lmagan so‘zlarni aniflash dasturini tuzing.

3. Xarita yordamida matnda so‘zlar chasotasini hisoblash dasturini tuzing.

4. Prioritetli navbat yordamida savatni to‘ldirish masalasini hal qiluvchi dastur tuzing.

5. Stek yordamida qavslar to‘g‘riligini tekshiruvchi dastur tuzing.

STANDART ALGORITMLAR

Algoritmlar

har bir algoritm funksiyalar shabloni yoki funksiyalar shabloni to‘plami yordamida ifodalanadi. SHunday qilib, algoritm har xil turdagi qiymatlarga ega bo‘lgan har xil konteynerlar bilan ishlay oladi. Barcha algoritmlarni argumentlari **[beg, end)** yarim oraliqlar bo‘ladi.

Algoritmlar **<algorithm.h>** sarlavha faylda tavsiflangan.

Quyida ko‘p ishlatiladigan STL funksiya-algoritmlari keltirilgan.

Belgilashlar. Algoritmlarni yozish uchun quyidagi belgilashlardan foydalanilgan.

Iteratorlar uchun:

II InputIterator kiritish iteratori;

OI OutputIterator chiqarish iteratori;

FI ForwardIterator bir tomonlama iterator;

BI BidirectionalIterator ikki tomonlama iterator;

RAI RandomAccessIterator ixtiyoriy murojaatli iterator;

Funksiya obyektlar uchun:

Unpred unar predikat;

UnProc unar amal;

UnFunk unar funksiya;

CompFunk solishtirish funksiyasi rostga teng, agar birinchi element ikkinchisidan kichik bo'lsa;

Funk qiymatlarni generatsiya qiluvchi funksiya;

RandFunk tasodifiy qiymatlarni generatsiya qiluvchi funksiya;

BinFunk arifmetik hisoblashlar uchun binar funksiya;

BinPred binar predikat rostga teng, agar ikki parametri teng bo'lsa.

Kvadrat qavslardagi ifoda majburiy bo'lmagan ifodadir.

Bunday parametrli yozuv algoritmlar ikki ta'rifiga mos keladi. Masalan:

II min_element(II *beg*, II*end* [,CompFunk*op*])

Juft funksiyalar shabloniga mos keladi:

template<class InputIterator beg, InputIterator end)

InputIterator min_element(InputIterator beg, InputIterator end)

template<class InputIterator, class InputIterator, class CompFunk)

InputIterator min_element(InputIterator beg, InputIterator end, CompFunk op)

O'zgartirmaydigan algoritmlar

1. Oraliqdagi elementlarni o'zgartirmaydigan algoritm. Izlash, maksimum va minimum algoritmlari topilgan element pozitsiyasini qaytaradi.

UnProc for_earch(II *beg*, II *end*, UnProc *op*) oraliq har bir elementi uchun amallarni bajaradi. Oxirgi qo'llangan amalini qaytaradi;

II find(II *beg*, II *end*, const T& *value*) qiymatni oraliqdagi birinchi kirishini topadi;

II find_if(II *beg*, II *end*, UnPred *op*) oraliqda predikatga birinchi moslashuvni topadi;

difference_type count(II *beg*, II *end*, const T& *value*) qiymatni oraliqqa kirishini hisoblaydi;

difference_type count_if(II *beg*, II *end*, UnPred *op*) oraliqda predikatni bajarilishini hisoblaydi;

II min_element(II *beg*, II *end* [,CompFunk*op*]) oraliqdagi eng kichik qiymat;

II max_element(II *beg*, II *end* [,CompFunk*op*]) oraliqdagi eng katta qiymat.

2. Oraliqni boshqa oraliqqa nusxasini olib o‘tish algoritmlari. Hamma algoritmlar qabul qiluvchi oraliqdagi oxirgi ko‘chirilgan elementdan keyingi pozitsiyani qaytaradi**.**

OI copy(II *sourcebeg*, II *sourceend*, OI *destbeg*) birinchi elementdan boshlab oraliqni nusxasini oladi.

BI1 copy_backward (BI1 *sourcebeg*, BI1 *sourceend*, BI2 *destbeg*) oxirgi elementdan boshlab oraliqni nusxasini oladi;

OI replace_copy(II *sourcebeg*, II *sourceend*, OI *destbeg*, const T& *oldvalue*, const T& *newvalue*) ko‘rsatilgan elementlarni almashtirib nusxasini oladi;

OI replace_copy_if(II *sourcebeg*, II *sourceend*, OI *destbeg*, UnPred *op*, const T& *newvalue*) predikatni bajargan elementlarni almashtirgan holda oraliqni nusxasini oladi;

OI remove_copy(II *sourcebeg*, II *sourceend*, OI *destbeg*, const T& *value*) ko‘rsatilgan qiymatga ega bo‘lgan elementlarni o‘chirgan holda oraliqni nusxasini oladi;

OI remove_copy_if(II *sourcebeg*, II *sourceend*, OI *destbeg*, UnPred *op*) (*sourcebeg*, *sourceend*, *destbeg*, *op*) predikatni bajargan elementlarni o‘chirgan holda oraliqni nusxasini oladi;

OI unique_copy(II *beg*, II *end*, OI *destbeg* [,BinPred *op*]) teng bo‘lgan qo‘shni elementlarni o‘chirgan holda oraliqni nusxasini oladi;

OI rotate_copy(FI *beg*, FI *newbeg*, FI *end*, OI *destbeg*) nusxalash jarayonida elementlarni sikl bo‘yicha suradi.

3. Izlash va ikki oraliqni solishtirish algoritmlari. Topilgan element pozitsiyasini qaytaradi.

FI1 search(FI1 *beg*, FI1 *end*, FI2 *searchbeg*, FI2 *searchend* [,BinPred op]) oraliqni birinchi kiritilishini topadi;

FI1 find_end(FI1 *beg*, FI1 *end*, FI2 *searchbeg*, FI2 *searchend* [,BinPred op]) oraliqni oxirgi kiritilishini topadi;

pair<II1,II2 > mismatch(II1 *beg*, II1 *end*, II2 *cmpbeg* [,BinPred *op*]) farqlanadigan birinchi elementni qaytaradi.

Ikki oraliqni solishtirish algoritmlari shart bajarilganda rost qaytaradi:

bool equal(II1 *beg*, II1 *end*, II2 *cmpbeg* [,BinPred *op*]) ikkita oraliqni tengligini tekshiradi;

bool lexicographical_compare(II1 *beg1*, II1 *end2*, II2 *beg2*, II2 *end2*

[,CompFunk *op*]) leksikografik solishtirilish.

4. Saralangan oraliqda izlash algoritmlari. Topilgan element pozitsiyasini qaytaradi.

FI lower_bound(FI *beg*, FI *end*, const T& *value* [,BinPred *op*]) qiymatni birinchi kirishini topadi;

FI upper_bound(FI *beg*, FI *end*, const T& *value* [,BinPred *op*]) qiymatdan katta bo‘lgan birinchi elementni topadi;

bool binary_search(FI *beg*, FI *end*, const T& *value* [,BinPred *op*])

ko‘rsatilgan element mavjudligini aniqlaydi. Mantiqiy qiymat qaytaradi.

5. Ikkita saralangan oraliq ustidagi algoritmlar. Natija oraliqdagi oxirgi ko‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

bool includes(II1 *beg*, II1 *end*, II2 *searchbeg*, II2 *searchend* [BinPred *op*]) bitta oraliqni ikkinchi oraliqqa kirishini tekshirish. Mantiqiy qiymat qaytaradi;

OI set_union(II *sourcebeg1*, II *sourceend1*, II *sourcebeg2*, II *sourceend2*,

OI *destbeg* [,BinPred *op*]) oraliqlarni birlashtirish;

OI set_intersection(II *sourcebeg1*, II *sourceend1*, II *sourcebeg2*, II *sourceend2*, OI *destbeg* [,BinPred *op*]) oraliqlarni o‘zaro kesishishi;

OI set_difference(II *sourcebeg1*, II *sourceend1*, II *sourcebeg2*, II *sourceend2*, OI *destbeg* [,BinPred *op*]) oraliqlarni ayirmasi;

OI set_symmetric_difference(II *sourcebeg1*, II *sourceend1*, II *sourcebeg2*, II *sourceend2*, OI *destbeg* [,BinPred *op*]) oraliqlarni simmetrik ayirmasi;

OI merge(II *sourcebeg1*, II *sourceend1*, II *sourcebeg2*, II *sourceend2*,

OI *destbeg* [,BinPred *op*]) ikkita oraliqni birlashtirish.

O‘zgartiruvchi algoritmlar

1. Oraliqda elementlarni almashtirish algoritmlari. Qiymat qaytarmaydi.

void fill(FI beg, FI end, const T& newvalue) ko‘rsatilgan qiymatdagi barcha elementlarni almashtiradi;

void replace(FI beg, FI end, const T& oldvalue, const T& newvalue) ko‘rsatilgan qiymatli elementlarni almashtiradi;

void replace_if(FI beg, FI end, UnPred op, const T& newvalue) predikat bajarilganda elementlarni almashtiradi.

2. Oraliqda elementlarni o‘chirish algoritmlari. Mantiqy o‘chirishni amalga oshiradi va mantiqiy oxiri pozitsiyasini qaytaradi. Konteyner hajmi o‘zgarmaydi.

FI remove(FI beg, FI end, const T& value) ko‘rsatilgan qiymatdagi barcha elementlarni o‘chiradi;

FI remove_if(FI beg, FI end, UnPred op) predikat bajarilganda elementlarni o‘chiradi;

FI unique(FI beg, FI end [,BinPred op]) teng bo‘lgan qo‘shni elementlarni o‘chiradi.

3. Oraliqda elementlarni joyini almashtirish algoritmlari. Qiymat qaytarmaydi.

void reverse(BI beg, BI end) elementlar tartibini teskarisiga almashtiradi;

void random_shuffle(RAI beg, RAI end [,RandFunk& op]) elementlarni tasodifiy tekis taqsimot asosida joylab chiqadi;

void rotate(FI beg, FI newbeg, FI end) sikl bo‘yicha elementlarni siljitadi**;**

Kriteriy asosida almashtirish algoritmlari kriteriyga mos kelmaydigan birinchi element pozitsiyasini qaytaradi:

BI partition(BI beg, BI end, UnPred op) elementlar tartibini o‘zgartiradi, bunda kriteriyaga javob beradigan elementlar oldida bo‘ladi;

BI stable_partition (BI beg, BI end, UnPred op) partition() ga o‘xshash, lekin kriteriyga javob beradigan va javob bermaydigan elementlar tartibini saqlaydi.

Leksikografik almashtirish algoritmlari. Agar natija o‘sish (**next**) bo‘yicha yoki kamayish bo‘yicha (**prev**) tartiblangan bo‘lsa, **false** qaytaradi:

next_permutation(beg,end) leksikografik tartibdagi keyingi almashuv;

prev_permutation(beg,end) leksikografik tartibdagi oldingi almashuv.

4. Oraliqdagi elementlarni saralash algoritmlari. Qiymat qaytarmaydi.

void sort(RAI beg, RAI end [,CompFunk op]) oraliqni saralaydi. Tezkor tartiblash mexanizmiga asoslangan;

void partial_sort(RAI beg, RAI sortend, RAI end [,CompFunk op]) oraliqning qismini saralaydi. Qo‘shib tartiblash mexanizmiga asoslangan;

void stable_sort(RAI beg, RAI end [,CompFunk op]) teng elementlarni ketma-ketlik tartibini saqlagan holda oraliqni saralaydi. Uyumli tartiblash mexanizmiga asoslangan.

5. Ikkita oraliqlar uchun o‘zgarituvchi algoritmlar. Qabul qiluvchi oraliqdagi oxirgi o‘zgartirilgan elementdan keyingi element pozitsiyasini qaytaradi.

OI transform(II source beg, II sourceend, OI destbeg, UnFunk op) elementlarni modifikatsiyalaydi (va nusxasini oladi) hamda ikkita oraliqdagi elementlarni birlashtiradi;

OI transform(II1 sourcebeg, II1 sourceend, II2 sourcebeg2,OI destbeg, UnFunk op) ikki intervalni birlashtiradi;

FI2 swap_ranges(FI1 beg1, FI1 end1, FI2 beg2) ikkita oraliqni joyini almashtiradi.

Sonli algoritmlar

1. Oraliq uchun sonli algoritmlar. Oraliqni o‘zgartirmaydiganlari jamlovchi qiymat qaytaradi:

T accumulate(II *beg*, II *end*, T *initvalue* [,BinFunk op]) elementlarning barcha qiymatlarini birlashtiradi (yig‘indini, ko‘paytmani va h.k. hisoblaydi);

T inner_product(II1 beg1, II1 end1, II12 beg2 T initvalue [,BinFunk op1, BinFunk op2]) ikkita oraliqdagi barcha elementlarni birlashtirish (skalyar ko‘paytmasini va h.k.larni hisoblaydi).

Oraliqni o‘zgartiruvchilari qabul qiluvchi oraliqdagi oxirgi elementdan keyingi pozitsiyani qaytaradi;

OI adjacent_difference(II sourcebeg, II sourceend, OI destbeg [,BinFunk op]) har bir elementni uni oldingi qiymati bilan birlashtirish (ayirmasi va h.k.larni hisoblaydi);

OI partial_sum(II sourcebeg, II sourceend, OI destbeg, [BinFunk op]) har bir elementni ularning oldingi qiymatlari bilan birlashtiradi (barcha xususiy yig‘indilarni va h.k. hisoblaydi).

Algoritmlardan foydalanish

Izlash, maksimum va minimum algoritmlari. Izlash, maksimum va minimum algoritmlari topilgan element pozitsiyasini qaytaradi. Agar element mavjud bo‘lmasa, birinchi oraliq **end** pozitsiyasini qaytaradi.

Misol:

```
#include <iostream>
#include<list>
#include<string>
#include<algorithm>
using namespace std;
bool cmp(string elem){return (elem.size()==4);};
int main(){
```

```
string s1[] = {"Bill","Jessica","Bill", "Bill","Benni"};
list <string> l1(s1,s1+5);
cout<<*max_element(l1.begin(), l1.end())<<endl;
cout<<*min_element(l1.begin(), l1.end())<<endl;
system("pause");
return 0;
}
```

Natija:
Jessica
Benni

Misol:

```
#include <iostream>
#include<list>
#include<string>
#include<algorithm>
using namespace std;
bool cmp(string elem){return (elem.size()==4);};
int main(){
string s1[] = {"Bill","Jessica","Bill", "Bill","Benni"};
list <string> l1(s1,s1+5);
int m; m=count(l1.begin(), l1.end(),"Bill"); cout<<m<<" ";
m=count_if(l1.begin(), l1.end(),cmp);  cout<<m<<endl;
system("pause");
return 0;
}
```

Natija:
3 3

Elementni izlash algoritmlari topilgan element pozitsiyasini qaytaradi. Agar element mavjud bo‘lmasa, birinchi oraliq **end** pozitsiyasini qaytaradi.

Funksiya **distance iterator** orasidagi ayirmani hisoblaydi:

```
#include <iostream>
#include<list>
#include<string>
#include<algorithm>
using namespace std;
template<class T>
void PosPrint(T beg,T end, T pos){
if (pos==end) cout<<"Not Found"<<endl; else
cout<<"Found position "<<distance(beg,pos)<<endl;
}
bool cmp(string elem){return (elem.size()==4);};
```

```
int main(){
string s1[] = {"Bob","Jessica","Bill", "Bill","Mary"};
list <string> l1(s1,s1+5); list<string>::iterator pp;
pp=find(l1.begin(), l1.end(), "Mary");  PosPrint(l1.begin(),l1.end(),pp);
pp=find_if(l1.begin(),l1.end(),cmp);
PosPrint(l1.begin(),l1.end(),pp);
string s2[]={"Bill","Benni"};
list <string> sl1(s2,s2+2);
pp=search(l1.begin(), l1.end(),sl1.begin(),sl1.end());
PosPrint(l1.begin(),l1.end(),pp);
system("pause");
return 0;
}
```

Natija:
Found position 4
Found position 2
Not Found

Ikki oraliqni solishtirish algoritmlari shart bajarilganda, rost qaytaradi. **mismath** algoritmi farqlanadigan birinchi juftlikni qaytaradi.

Misol:

```
#include <iostream>
#include<list>
#include<string>
#include<algorithm>
using namespace std;
int main(){
string s1[] = {"Bob","Jessica","Bill", "Bill","Benni"};
list <string> l1(s1,s1+5);
string s2[]={"Bill","Benni"}; list <string> sl1(s2,s2+2);
bool bp;  bp=equal(l1.begin(), l1.end(),sl1.begin());
if(bp)cout << "equal" << endl;else cout << "no equal" << endl;
bp=lexicographical_compare(sl1.begin(), sl1.end(),l1.begin(),l1.end());
if(bp)cout << "lexi_less" << endl; else cout << "no lexi_less" << endl;
system("pause");
return 0;
}
```

Natija:
no equal
lexi_less

Misol:

```
#include <iostream>
```

```
#include<list>
#include<string>
#include<algorithm>
using namespace std;
int main(){
string s1[] = {"Bob","Jessica","Bill", "Bill","Benni"};
list <string> l1(s1,s1+5);
string s2[]={"Bill","Benni"}; list <string> sl1(s2,s2+2);
pair<string*,string*> pair_ia = mismatch( s1, s1+5, s2 );
cout<< *pair_ia.first <<" "<< *pair_ia.second << endl;
system("pause");
return 0;
}
```

Natija:
Bob Bill

Almashtirish algoritmlari qiymat qaytarmaydi:

```
#include <iostream>
#include<list>
#include<string>
#include<algorithm>
using namespace std;
bool cmp(string elem){return (elem.size()==4);};
template <class  type>
void CPrint(type& cont) {
typename type::const_iterator p = cont.begin();
if (cont.empty())
cout << "Container is empty.";
for (p; p != cont.end(); ++p)
cout << *p << ' ';
cout << endl;
};
int main(){
list<string> l1;
fill (l1.begin(), l1.end(),"again"); CPrint(l1);
string s1[] = {"Bill","Jessica","Bill", "Mary","Ben"};
list <string> ls1(s1,s1+5);  list <string> ls2(s1,s1+5);;
replace(ls1.begin(), ls1.end(),"Bill","Smit");CPrint(ls1);
replace_if(ls2.begin(), ls2.end(),cmp,"Smit"); CPrint(ls2);
system("pause");
return 0;
}
```

Natija:
Container is empty.

Smit Jessica Smit Mary Ben
Smit Jessica Smit Smit Ben

Oraliqda elementlarni o‘chirish algoritmlari mantiqy o‘chirishni amalga oshiradi va mantiqiy tugash pozitsiyasini qaytaradi. Konteyner hajmi o‘zgarmaydi.

```
#include <iostream>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
bool cmp(string elem){return (elem.size()==4);};
template <class  type>
void CPrint(type& cont) {
typename type::const_iterator p = cont.begin();
if (cont.empty()) cout << "Container is empty.";
for (p; p != cont.end(); ++p) cout << *p << ' ';
cout << endl;
};
int main(){
string s1[] = {"Bill","Jessica","Bill", "Bill","Ben"};
vector<string> d1(s1,s1+5); vector<string> d2(s1,s1+5);
vector<string> d3(s1,s1+5); vector<string>::iterator end;
end=remove(d1.begin(), d1.end(),"Bill");CPrint(d1);
d1.erase(end,d1.end()); CPrint(d1);
end=unique(d2.begin(), d2.end());CPrint(d2);
d2.erase(end,d2.end()); CPrint(d2);
end=remove_if(d3.begin(), d3.end(),cmp); CPrint(d3);
d3.erase(end,d3.end()); CPrint(d3);
system("pause");
return 0;
}
```

Natija:
Jessica Ben Bill Bill Ben
Jessica Ben
Bill Jessica Bill Ben Ben
Bill Jessica Bill Ben
Jessica Ben Bill Bill Ben
Jessica Ben

Oraliqdagi elementlarni joyini almashtirish algoritmlari. Qiymat qaytarmaydi. Leksikografik almashtirish algoritmlari agar natija o‘sish (**next**) bo‘yicha yoki kamayish bo‘yicha (**prev**) tartiblangan bo‘lsa, **false** qaytaradi.

```
#include <iostream>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
bool cmp(string s) { return s[0]=='B';};
template <class  type>
void CPrint(type& cont) {
typename type::const_iterator p = cont.begin();
if (cont.empty()) cout << "Container is empty.";
for (p; p != cont.end(); ++p) cout << *p << ' ';
cout << endl;
};
int main(){
string s1[] = {"Bob","Bil","Jes","Ann", "Sem"};
vector<string> cl1(s1,s1+5); vector<string> cl2(s1,s1+5);
CPrint(cl1);
reverse(cl1.begin(),cl1.end()); CPrint(cl1);
rotate(cl1.begin(),cl1.begin()+2,cl1.end());CPrint(cl1);
random_shuffle(cl1.begin(),cl1.end()); CPrint(cl1);
partition(cl1.begin(),cl1.end(),cmp);CPrint(cl1);
sort(cl1.begin(),cl1.end()); CPrint(cl1);
next_permutation(cl1.begin(),cl1.end()); CPrint(cl1);
prev_permutation(cl1.begin(),cl1.end()); CPrint(cl1);
stable_partition(cl1.begin(),cl1.end(),cmp); CPrint(cl1);
system("pause");
return 0;
}
```

Natija:

```
Bob Bill Jes Ann Sem
Sem Ann Jes Bil Bob
Jes Bil Bob Sem Ann
Ann Bil Sem Bob Jes
Bob Bill Sem Ann Jes
Ann Bil Bob Jes Sem
Ann Bil Bob Sem Jes
Ann Bil Bob Jes Sem
Bil Bob Ann Jes Sem
```

Nusxalash algoritmlari qabul qiluvchi oraliqdagi oxirgi ko‘chirilgan elementdan keyingi pozitsiyani qaytaradi.

```
#include <iostream>
#include<vector>
#include<string>
```

```
#include<algorithm>
using namespace std;
bool cmp(string s) { return s[0]=='B';};
template <class  type>
void CPrint(type& cont) {
typename type::const_iterator p = cont.begin();
if (cont.empty()) cout << "Container is empty.";
for (p; p != cont.end(); ++p) cout << *p << ' ';
cout << endl;
};
int main(){
string s1[] = {"Jessica","Bill","Bill", "Mary","Ben"};
vector<string> ls1(s1,s1+5);
vector<string> ds1(5),ds2(5),ds3(5),ds4(5);
vector<string> ds5(5),ds6(5),ds7(5),ds8(5);
replace_copy(ls1.begin(), ls1.end(),ds1.begin(),"Bill","Smit");
CPrint(ds1);
remove_copy(ls1.begin(), ls1.end(), ds2.begin(),"Bill"); CPrint(ds2);
unique_copy(ls1.begin(), ls1.end(),ds3.begin()); CPrint(ds3);
rotate_copy(ls1.begin(),ls1.begin()+3, ls1.end(),ds4.begin()); CPrint(ds4);
copy_backward (ls1.begin(),ls1.end(), ds5.end()); CPrint(ds5);
replace_copy_if(ls1.begin(), ls1.end(),ds6.begin(),cmp,"Smit");
CPrint(ds6);
remove_copy_if(ls1.begin(), ls1.end(),ds7.begin(),cmp);CPrint(ds7);
copy(ls1.begin(), ls1.end(),ds8.begin());CPrint(ds8);
system("pause");
return 0;
}
```

Natija:

```
Jessica Smit Smit Mary Ben
Jessica Mary Ben
Jessica Bill Mary Ben
Mary Ben Jessica Bill Bill
Jessica Bill Bill Mary Ben
Jessica Smit Smit Mary Smit
Jessica Mary
Jessica Bill Bill Mary Ben
```

Transformatsiya va almashtirish qabul qiluvchi oraliqdagi oxirgi o‘zgartirilgan elementdan keyingi pozitsiyani qaytaradi:

```
#include <iostream>
#include<list>
#include<algorithm>
using namespace std;
```

```
int double_val( int val ) { return val + val; }
int difference( int val1, int val2 ) {
return abs( val1 - val2 ); };
int main(){
int a1[]={1, 1, 2, 3, 5, 6}; int a2[]={2, 2, 3, 4, 6, 7};
list<int> l1(a1,a1+6);list<int> lc1(6);
list<int> l2(a2,a2+6);list<int> lc2(6);
list<int>::iterator p;
transform( l1.begin(),l1.end(),lc1.begin(),double_val);
for(p=lc1.begin();p!=lc1.end();p++) cout<<*p<<" "; cout<<endl;
transform( l1.begin(),l1.end(),l2.begin(),lc2.begin(),difference);
for(p=lc2.begin();p!=lc2.end();p++) cout<<*p<<" ";cout<<endl;
swap_ranges(l1.begin(),l1.end(),l2.begin());
for(p=l1.begin();p!=l1.end();p++) cout<<*p<<" "; cout<<endl;
for(p=l2.begin();p!=l2.end();p++) cout<<*p<<" ";cout<<endl;
system("pause");
return 0;
}
```

Natija:

```
2 2 4 6 10 12
1 1 1 1 1 1
2 2 3 4 6 7
1 1 2 3 5 6
```

Saralangan oraliqda izlash algoritmlari topilgan element pozitsiyasini qaytaradi. Agar element topilmasa, oraliq oxirgi **end** pozitsiyasini qaytaradi. Algoritm **binary_search** mantiqiy qiymat qaytaradi:

```
#include <iostream>
#include<set>
#include<string>
#include<algorithm>
using namespace std;
int main(){
string s1[5] = {"Bill", "Jessica", "Ben", "Mary", "Monica"};
set<string> bt(s1, s1 + 5);
if (binary_search( bt.begin(),bt.end(),"Mary")) cout<<"YES";
else cout<<"NO"; cout<<endl;
set<string>::iterator pos;
pos=lower_bound( bt.begin(),bt.end(),"Ben" );
cout<<distance(bt.begin(),pos)<<" "<<*pos<<endl;
pos=upper_bound( bt.begin(),bt.end(),"Bill" );
cout<<distance(bt.begin(),pos)<<" "<<*pos<<endl;
system("pause");
return 0;
```

```
}
```
Natija:
Yes
0 Ben
2 Jessica

Ikkita saralangan oraliq ustidagi algoritmlar qabul qiluvchi oraliqdagi oxirgi ko‘chirilgan elementdan keyingi pozitsiyani qaytaradi:

```
#include <iostream>
#include<set>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
template <class  type>
void CPrint(type& cont) {
typename type::const_iterator p = cont.begin();
if (cont.empty()) cout << "Container is empty.";
for (p; p != cont.end(); ++p) cout << *p << ' ';
cout << endl;
};
int main(){
const int N = 5;
string s1[N] = {"Bill", "Jessica", "Ben", "Mary", "Monica"};
string s2[N] = {"Sju","Monica","John","Bill","Sju"};
typedef set<string> Sets;
Sets a(s1, s1 + N);Sets b(s2, s2 + N);
CPrint(a); CPrint(b);
if (includes(a.begin(), a.end(), b.begin(), b.end())) cout << "Yes" << endl;
else cout << "No" << endl;
Sets sum,prod, dif,sdif;
set_union(a.begin(), a.end(), b.begin(), b.end(),
inserter(sum,sum.begin())); CPrint(sum);
set_intersection(a.begin(), a.end(), b.begin(), b.end(),
inserter(prod, prod.begin())); CPrint(prod);
set_difference(a.begin(), a.end(), b.begin(), b.end(),
inserter(dif, dif.begin())); CPrint(dif);
set_symmetric_difference(a.begin(), a.end(), b.begin(), b.end(),
inserter(sdif, sdif.begin()));  CPrint(sdif);
vector<string> bt;
merge(a.begin(),a.end(),b.begin(), b.end(),back_inserter(bt));
CPrint(bt);
system("pause");
return 0;
```

```
}
```

Sonli algoritmlar. Oraliqni o‘zgartirmaydiganlari jamlovchi qiymat qaytaradi. Oraliqni o‘zgartiruvchilari qabul qiluvchi oraliqdagi oxirgi elementdan keyingi pozitsiyani qaytaradi:

Misol:

```
#include <iostream>
#include <list>
#include <algorithm>
#include <numeric>
using namespace std;
int main(){
int a1[] =  { 2, 3, 5, 8 };int a2[] = { 1, 2, 3, 4 };
list <int> c1(a1,a1+4);list <int> c2(a2,a2+4); list<int> c3(4);
int s1=accumulate (c1.begin(), c1.end(), 0);
cout<<s1<<endl;
int s2 = inner_product( c1.begin(), c1.end(),c2.begin(), 0 );
cout<<s2<<endl;
system("pause");
return 0;
}
```

Natija:

18

55

Misol:

```
#include <iostream>
#include <list>
#include <algorithm>
#include <numeric>
using namespace std;
int main(){
int a1[] =  { 2, 3, 5, 8 };int a2[] = { 1, 2, 3, 4 };
list <int> c1(a1,a1+4);list <int> c2(a2,a2+4); list<int> c3(4);
list<int>::iterator p;
adjacent_difference( c1.begin(), c1.end(), c3.begin());
for (p=c3.begin(); p != c3.end(); ++p) cout << *p << ' ';
cout<<endl;
partial_sum( c2.begin(), c2.end(), c2.begin());
for (p=c2.begin(); p != c2.end(); ++p) cout << *p << ' ';
cout<<endl;
system("pause");
```

```
return 0;
}
```

Natija:
2 1 2 3
1 3 6 10

Nazorat savollari:

1. Iteratorlar funksiyalari.
2. Hamma algoritmlar argumenti nimadan iborat?
3. Interval elementini o‘zgartirmaydigan algoritmlar.
4. Ikki tartiblangan intervallar ustida algoritmlarni ko‘rsating.
5. Interval elementlarini o‘chirish uchun qaysi algoritmlardan foydalaniladi?
6. Nusxa olish algoritmlarini ko‘rsating.
7. Oraliqda elementlarni joyini almashtirish algoritmlari xossalari.
8. Tartiblash algoritmlari qaysi usullarga asoslangan.
9. Intervallarni solishtirish algoritmlari.
10. Sonli algoritmlarni ko‘rsating.

Misollar:

1. Izlash algoritmlariga dastur tuzing.
2. Nusxalash algoritmlariga dastur tuzing.
3. O‘rin almashtirish algoritmlariga dastur tuzing.
4. Ikki oraliqni solishtirish algoritmlariga dastur tuzing.
5. Sonli algoritmlarga dastur tuzing.

Algoritmlar va iteratorlar

Iteratorlar xossalari

Iteratorlar bilan bog‘liq funksiyalar. Funksiya **advance** iteratorni to‘g‘ri yoki teskari tartibda berilgan sondagi qadamga ko‘chiradi. Ikki tomonlama va

ixtiyoriy murojaatli iteratorlar uchun qadam soni manfiy bo‘lishi mumkin. Funksiya ketma-ketlikdan chiqish imkoniyatini tekshirmaydi.

Funksiya **distance** ikki iterator orasidagi farqni hisoblaydi. Ikki iterator bitta konteyner elementlariga murojaat qilishi kerak.

Funksiya **iter_swap** iteratorlar murojaat qilgan elementlarni almashtiradi. Iteratorlar har xil turli bo‘lishi mumkin lekin qiymat berish bo‘yicha mos kelishi kerak.

Misol:

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;
int main(){
int ar1[]={3,2,4,5,3,8};
list<int> lst1(ar1,ar1+6);
list<int>::iterator pos1=lst1.begin();
list<int>::iterator pos2=lst1.begin();
advance(pos2,3);
cout<<distance(pos1,pos2)<<endl;
cout<<*pos1<<" "<<*pos2<<endl;
iter_swap(pos1,pos2);
cout<<*pos1<<" "<<*pos2;
return 0;
}
```

Natija:

```
3
3 5
5 3
```

Misol:

```
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
class Array{
vector<int> vec;
public:
Array(vector<int> v):vec(v){};
public:
int size(){
```

```
return distance(vec.begin(),vec.end());
}
void print(int k) {
vector< int >::iterator iter=vec.begin();
while(iter!=vec.end()){
cout<<*iter<<" ";
advance(iter,k);
}
cout<<endl;
}
void reverse(){
vector< int >::iterator iter1=vec.begin();
vector< int >::iterator iter2=vec.end()-1;
while(iter1!=iter2) iter_swap(iter1++,iter2--);
}
};
int main(){
int a[] = {2,1,3};
vector<int> v(a,a+3);
Array P(v);
P.print(1);
cout<<P.size()<<endl;
P.reverse();
P.print(1);
system("pause");
return 0;
}
```

Natija:

2 1 3

3

3 1 2

Konstanta iteratorlar. Til talabi bo‘yicha const-konteyner bilan bog‘liq iterator konstanta bo‘lishi zarur. Bu const-konteyner o‘zgarmasligini ta'minlaydi.

Amali **begin()** va **end()** bo‘lib, **const** spetsifikatori mavjud bo‘lsa, konstanta iterator qaytaradi.

Konstanta iteratorga konstanta bo‘lmagan iteratorni qiymat sifatida berish mumkin.

Teskari iteratorlar. Amallar **begin()** va **end()** mos holda birinchi va oxirgidan keyingi elementga ko‘rsatuvchi iteratorlar qaytaradi. Bundan tashqari konteynerni oxiridan boshiga o‘quvchi iterator qaytarish mumkin. Hamma konteynerlarda bu

imkoniyatdan foydalanish uchun **rbegin()** va **rend()** amallari qo‘llanadi.. Teskari iteratorlar konstanta va konstanta bo‘lmagan versiyalari mavjud.

Teskari iterator to‘g‘risi kabi qo‘llanadi. Farqi – keyingi va oldinga o‘tish operatorlarida. To‘g‘ri iterator uchun operator ++ keyingi elementga o‘tish, teskari iterator uchun -- keyingi elementga o‘tish.

Inkrement va dekrement operatorlari semantikasini almashtirish chalkashtirishi mumkin, lekin dasturchiga to‘g‘ri iteratorlar o‘rniga teskari iteratorlarni uzatish imkonini beradi. Masalan, vektorni kamayish tartibida tartiblash uchun **sort()** algoritmiga teskari iteratorlar juftligini uzatish mumkin.

Teskari iteratorni to‘g‘ri iteratorga aylantirish uchun **base** funksiyasidan foydalaniladi.

Misol:

```
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
class Array{
vector<int> vec;
public:
Array(vector<int> v):vec(v){};
public:
void sorted(){
sort( vec.rbegin(), vec.rend() );
}
void print() {
vector< int >::const_iterator iter;
for(iter=vec.begin();iter!=vec.end();iter++) cout<<*iter<<" ";
cout<<endl;
}
void reverse_print() {
vector< int >::reverse_iterator riter;
for(riter=vec.rbegin();riter!=vec.rend();riter++) cout<<*riter<<" ";
cout<<endl;
}
};
int main(){
int a[] = {2,1,3};
vector<int> v(a,a+3);
Array P(v);
```

```
P.print();
P.reverse_print();
P.sorted();
P.print();
system("pause");
return 0;
}
```
Natija:
2 1 3
3 1 2
3 2 1

Joylash iteratorlari. Nusxalash algoritmlari qiymat berish amalidan foydalanadi. Agar qabul qiluvchi konteynerda joy ajratilmagan bo‘lsa, bu amal muvaffaqiyatsiz bajariladi.

Bu masalani hal qilish uchun standart kutubxonada maxsus joylash iteratorlarini qaytaruvchi uch adapterdan foydalaniladi:

* **back_inserter()** qiymat amali o‘rniga **push_back()** joylash amalini chaqiradi. Konteyner o‘zi **back_inserter()** argumentidir.

* **front_inserter()** qiymat amali o‘rniga **push_front()** joylash amalini chaqiradi. Konteyner o‘zi **push_front()** argumentidir. Vektor uchun bu adapterni qo‘llab bo‘lmaydi.

* **inserter()** qiymat amali o‘rniga **insert()** joylash amalini chaqiradi. Adapter **inserter()** ikki argumentga ega: konteyner va joylash pozitsiyasini ko‘rsatuvchi iterator. Iterator har bir joylashdan so‘ng oldinga suriladi.

Misol:

```
class Test{
deque<int> dr;
public:
template<typename iter>
void copy(iter begin,iter end){
unique_copy( begin, end, back_inserter( dr) );
unique_copy( begin, end, front_inserter( dr ) );
unique_copy( begin, end, inserter( dr, dr.begin()));
}
void print(){
for(int i=0;i<dr.size();i++) cout<<dr[i]<<" ";
cout<<endl;
```

```
}
};
int main(){
int i,n;
int ia[] = { 0, 1, 1, 2 };
vector< int > ivec( ia, ia+4 );
Test P;
P.copy(ivec.begin(), ivec.end());
P.print();
system("pause");
return 0;
}
```

Natija:

0 1 2 2 1 0 0 1 2

Oqimli iteratorlar

Oqimli iteratorlar. Standart kutubxona standart konteynerlar va umumlashgan iteratorlar bilan birga ishlovchi o‘qish va yozish oqimli iteratorlaridan foydalanish imkonini beradi. **istream_iterator** sinf iterator amallarini **istream** sinfi yoki fayldan o‘qish uchun hosila **ifstream** sinfi bilan qo‘llashga imkon beradi. Xuddi shunday **ostream_iterator** iterator amallarini **ostream** sinfi yoki faylga yozish uchun hosila **ofstream** sinfi bilan qo‘llashga imkon beradi. Bu iteratorlardan foydalanish uchun quyidagi sarlavhali faylni ulash lozim:

```
#include <iterator>
```

Misol:

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>
#include <functional>
using namespace std;
/*
* input:
* 23 109 45 89 6 34 12 90 34 23 56 23 8 89 23
*
output:
* 109 90 89 56 45 34 23 12 8 6
*/
int main(){
istream_iterator< int > input( cin );
```

```
istream_iterator< int > end_of_stream;
vector<int> vec;
copy ( input, end_of_stream, inserter( vec, vec.begin() ));
sort( vec.begin(), vec.end(), greater<int>() );
ostream_iterator< int > output( cout, " " );
unique_copy( vec.begin(), vec.end(), output );
int ii;cin>>ii;cin>>ii;
return 0;
}
```

Iterator istream_iterator. Umumiy holda o‘qish oqim iteratori **istream_iterator** e'loni quyidagi ko‘rinishga ega:

istream_iterator<Type> identifier(istream&)

bu yerda **Type** – kiritish amali aniqlangan sinf turi. Konstruktor argumenti **istream** sinfi obyekti, masalan **cin** yoki uning vorisi masalan – **ifstream**.

Har bir **istream_iterator** obyektiga inkrement operatori qo‘llanganda, kirish oqimidan element o‘qiladi, buning uchun **operator>>()** operatori ishlatiladi. Umumlashgan algoritmlarda fayl boshini va oxirini belgilovchi ikki iterator uzatish lozim. Boshlang‘ich pozitsiyani **istream** obyekti bilan initsializatsiya qilingan **istream_iterator** beradi. Oxirgi pozitsiyani olish uchun **istream_iterator** sinfi maxsus ko‘zda tutilgan konstruktori beradi.

Iterator ostream_iterator. Yozish oqimli iterator ta'rifi ikki shaklda berilishi mumkin:

ostream_iterator<Type> identifier(ostream&)

ostream_iterator<Type> identifier(ostream&, char * delimiter)

bu yerda **Type** – chiqarish operatori (**operator<<**) aniqlangan ixtiyoriy sinf. Ikkinchi shaklda **delimiter** – bu ajratuvchi, ya'ni har bir elementdan keyin faylga chiqariluvchi C-satr. Bunday satr ikkilik nol bilan tugashi lozim. Argument **ostream** sifatida **ostream** sinfi obyekti, masalan **cout** yoki uning vorisi, masalan, **ofstream** kelishi mumkin.

Misol:

```
#include <iterator>
#include <algorithm>
#include <iostream>
```

```
using namespace std;
class Print{
public:
static void print(){
copy( istream_iterator< int >( cin ),
istream_iterator< int >(),
ostream_iterator< int >( cout, " " ));
}
};
int main(){
Print::print();
system("pause");
}
```

Misol:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <fstream>
#include <iterator>
using namespace std;
class CopyFile{
string file_name;
public:
CopyFile(string name):file_name(name){};
int copy_file() const{
if ( file_name.empty() || !cin ) {
cerr << "unable to read file name\n"; return -1;
}
ifstream infile( file_name.c_str());
if ( !infile ) {
cerr << "unable to open " << file_name << endl;
return -2;
}
istream_iterator< string > ins( infile ), eos;
ostream_iterator< string > outs( cout, " " );
copy( ins, eos, outs );
return 0;
}
};
int main() {
string file_name;
cout << "please enter a file to open: ";
cin >> file_name;
```

```
CopyFile P(file_name);
P.copy_file();
system("pause");
return 0;
}
```

Obyekt funksiyalar

Obyekt funksiyalar. Ko‘p hollarda funksiyani obyekt-funksiyaga almashtirish qulaydir. Obyekt-funksiya yoki funktor – bu «dumaloq qavs» **()** amali aniqlangan sinf nusxasidir. Obyekt-funksiya funksiya sifatida ishlatilganda, uni chaqirish uchun operator **()** ishlatiladi.

Dumaloq qavs **()** amalini qo‘shimcha yuklash sinf obyektiga funksiya chaqirish sintaksisini qo‘llashga imkon beradi. Qavslarda operandlar soni ixtiyoriy. Avfzalliklari:

Obyekt funksiya ichki holatga ega bo‘lishi mumkin;

Obyekt funksiya turga ega;

Obyekt funksiya oddiy funksiyadan tezroq bajariladi.

Daraja hisoblash uchun obyekt-funksiya:

```
#include<iostream>
using namespace std;
class POW{
double s,x;
public:
POW(double x1=0):x(x1),s(1.0){};
double operator()(double x,int n){
s=1.0;
for(int i=1;i<=n;i++) s*=x;
return s;
}
double operator()(){
s*=x;
return s;
}
};
int main(){
POW spow;
cout<<spow(3.0,2)<<endl;
POW npow(3);
for(int i=1;i<=3;i++) cout<<npow()<<" ";
```

```
cin.get();
return 0;
}
```
Naija:
9
3 9 27

Olddindan kiritilgan obyekt-funksiyalar. Oldindan kiritilgan obyekt-funksiyalar arifmetik, mantiqiy va solishtiruvchi turlarga ajratiladi. Har bir obyekt – bu sinf shablonidir. Ulardan foydalanish uchun quyidagi sarlavhali faylni ulash lozim:

#include <functional>

Olddindan kiritilgan obyekt-funksiyalar umumlashgan algoritmlar argumentlari sifatida ishlatiladi.

Arifmetik obyekt-funksiyalar:

* Qo‘shish: **plus<Type>**
* Ayirish: **minus<Type>**
* Ko‘paytirish: **multurlies<Type>**
* Bo‘lish: **divides<Type>**
* Qoldiq olish: **modulus<Type>**
* Teskari qiymat: **negate<Type>**

Solishtiruvchi obyekt-funksiyalar:

* Teng: **equal_to<Type>**
* Noteng: **not_equal_to<Type>**
* Katta: **greater<Type>**
* Katta yoki teng: **greater_equal<Type>**
* Kichik: **less<Type>**
* Kichik yoki teng: **less_equal<Type>**

Mantiqiy obyekt-funksiyalar:

* Mantiqiy VA: **logical_and<Type>**
* Mantiqiy YOKI: **logical_or<Type>**
* Mantiqiy INKOR: **logical_not<Type>**

Misol:

```
#include <iostream>
#include<vector>
using namespace std;
int main(){
plus< int > intAdd;
int ival1 = 10, ival2 = 20;
int sum = intAdd( ival1, ival2 );
cout<<sum<<endl;//30
divides< int > intDiv;
int d=intDiv(20,10);
cout<<d<<endl;//2
int ii;cin>>ii;
return 0;
}
```

Natija:
30
2

Misol:

```
#include <iostream>
#include<vector>
#include<algorithm>
#include <numeric>
using namespace std;
class Array{
vector<int> vec;
public:
Array(vector<int> v):vec(v){};
public:
int mult(){
multurlies< int > intMult;
int k=accumulate (vec.begin(), vec.end(),1 , intMult);
}
void sorted(){
sort( vec.begin(), vec.end(),greater<int>());
};
void print(){
for(int i=0;i<vec.size();i++) cout<<vec[i]<<" ";
cout<<endl;
}
};
```

```
int main(){
int a[] = {1,2,3,4,5,6,7,8,9};
vector<int> v(a,a+9);
Array P(v);
cout<<P.mult()<<endl;
P.sorted();
P.print();
system("pause");
return 0;
}
```

Natija:

362880

9 8 7 6 5 4 3 2 1

Obyekt-funksiyalar uchun adapterlar. Standart kutubxonada unar va binar obyekt funksiyalar imkoniyatlarini kengaytirishga mo‘ljallaepgan adapterlar mavjud. Adappterlar maxsus sinflar bo‘lib quyidagi guruhlarga ajratilgan:

*Bog‘lovchilar (**binders**). Binar obyektni unar obyektga aylantiruvchi adapter. Argumentlar birini konkret qiymat bilan bog‘laydi.

*Inkor qiluvchilar (**negators**). Obyekt funksiya qiymatini teskariga aylantiradi.

Standart kutubxonaga ikki bog‘lovchi va inkor qiluvchi adapter kiritilgan:

bind1st(op,value) biror qiymatni binar obyekt-funksiya birinchi argumenti bilan bog`laydi;

bind2nd(op,value) biror qiymatni binar obyekt-funksiya ikkinchi argumenti bilan bog‘laydi;

not1(op) obyekt-funksiya bo‘lgan unar predikat qiymatini teskariga aylantiradi;

not2(op) obyekt-funksiya bo‘lgan binar predikat qiymatini teskariga aylantiradi.

Misol:

```
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
```

```
int k;
int a[] = {1,2,3,4,5,6,7,8,9};
vector<int> vec(a,a+9);
k=count_if( vec.begin(), vec.end(),
bind2nd( less_equal<int>(), 10 ));
cout<<k<<endl; //9
k=count_if( vec.begin(), vec.end(),
not1( bind2nd( less_equal<int>(), 10 )));
cout<<k<<endl; //0
return 0;
    }
```

Natija:

9

0

Obyekt-funksiya realizatsiyasi. Eng sodda holda obyekt funksiyani yaratish chaqirish amalini qo‘shimcha yuklashga asoslangan.

Misol:

```
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
class less_equal_ten {
public:
bool operator() ( int val )
{ return val <= 10; }
};
int main(){
int k;
int a[] = {1,2,3,4,5,6,7,8,9};
vector<int> vec(a,a+9);
k=count_if( vec.begin(), vec.end(),less_equal_ten());
cout<<k<<endl; //9
int ii;cin>>ii;
return 0;
}
```

Natija:

9

Albatta bu sinf imkoniyatlari cheklangan. Foydalanuvchiga solishtirish qiymatini kiritishga imkon berishi mumkin.

Misol:

```
#include<vector>
#include<algorithm>
using namespace std;
class less_equal_value {
public:
less_equal_value( int val ) : _val( val ) {};
bool operator() ( int val ) { return val <= _val; }
private:
int _val;
};
template < int _val >
class greater_value {
public:
bool operator() ( int val ) { return val >_val; }
};
int main(){
int k;
int a[] = {1,2,3,4,5,6,7,8,9};
vector<int> vec(a,a+9);
k=count_if( vec.begin(), vec.end(),less_equal_value(5));
cout<<k<<endl; //5
k=count_if( vec.begin(), vec.end(),greater_value<5>());
cout<<k<<endl; //4
int ii;cin>>ii;
return 0;
}
```

Natija:

5

4

for_each algoritmi bajarilgan amalni qaytaradi. Bu natijani obyekt funksiyada qaytarishga imkon beradi.

Misol:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class MeanValue{
long num;
long sum;
public:
MeanValue():num(0),sum(0){};
void operator()(int elem){
```

```
num++;
sum+=elem;
}
operator double(){
return static_cast<double>(sum)/static_cast<double>(num);
}
};
int main() {
vector<int> coll;
for(int i=1;i<5;i++) coll.push_back(i);
double mv=for_each(coll.begin(),coll.end(),MeanValue());
cout<<"mean value:"<<mv;
cin.get();
return 0;
}
```

Natija:

mean value:2.5

Sinf usullari uchun funksional adapterlar. Standart kutubxonaga sinf usulini chaqirishga imkon beruvchi boshqa adapterlar kiritilgan:

mem_fun_ref(op) obyekt usuli sifatida chaqirish;

mem_fun(op) obyekt usuliga ko‘rsatkich sifatida chaqirish.

Misol:

```
#include <iostream>
#include<vector>
#include<string>
#include<algorithm>
#include<functional>
using namespace std;
class Person{
string name;
public:
Person(string name){ this->name=name;}
void print(){
cout<<name<<endl;
};
void printpref(string pref){
cout<<pref<<name<<endl;
}
};
int main(){
string a[]={"Bill","Jessica","Bill", "Bill","Benni"};
```

```
vector<Person> vd;
for(int i=0;i<5;i++) vd.push_back(Person(a[i]));
for_each(vd.begin(),vd.end(),mem_fun_ref(&Person::print));
for_each(vd.begin(),vd.end(),bind2nd(mem_fun_ref(&Person::printpref),
"person:"));
return 0;
}
```

Bu misolda algoritmni

```
for_each(vd.begin(),vd.end(),&Person::print);
```

shaklda chaqirish kompilyasiya xatosiga olib keladi. Chunki algoritm uchinchi parametr sifatida uzatilgan ko‘rsatkich uchun qiymat olish **()** amalini chaqirishga urinadi. Funksional adapter sinf usulini chaqirishni amal chaqirishga aylantiradi.

Funksional adapter **ptr_fun** oddiy funksiyalarni boshqa funksional adapterlar bilan ishlatishga imkon beradi.

Misol:

```
#include <iostream>
#include<vector>
#include<string>
#include<algorithm>
#include<functional>
using namespace std;
bool checksize(string name){
return (name.size()==4);
};
bool compsize(string name,int k){
return (name.size()==k);
};
int main(){
string a[]={"Billi","Jessica","Bill", "Bill","Benni"};
vector<string> vd;
vector<string>::iterator pos;
for(int i=0;i<5;i++) vd.push_back(a[i]);
pos=find_if(vd.begin(),vd.end(),not1(ptr_fun(checksize)));
cout<<distance(vd.begin(),pos)<<" "<<*pos<<endl;
pos=find_if(vd.begin(),vd.end(),bind2nd(ptr_fun(compsize),4));
cout<<distance(vd.begin(),pos)<<" "<<*pos;
return 0;
}
```

Natija:

0 Billi

2 Bill

Nazorat savollari:

1. Konstanta iteratorlar.
2. Teskari iteratorlar.
3. Joylash iteratorlari.
4. Oqimli iteratorlar xususiyatlari.
5. Obyekt funksiyalar.
6. Arifmetik obyekt-funksiyalar.
7. Solishtiruvchi obyekt-funksiyalar.
8. Mantiqiy obyekt-funksiyalar.
9. Obyekt-funksiyalar uchun adapterlar vazifalari.
10. Funksional adapterlarlarni ko‘rsating.

Misollar:

1. Konstanta iteratorlarga dastur tuzing.
2. Oqimli iteratorlarga dastur tuzing.
3. Arifmetik obyekt-funksiyalarga dastur tuzing.
4. Mantiqiy obyekt-funksiyalarga dastur tuzing.
5. Funksional adapterlarlarga dastur tuzing.

Axborot tizimlarini yaratish

Tizim – bir vaqtning o‘zida yagona obyekt hamda maqsadga erishish uchun to‘plangan elementlar yig‘indisi sifatida tushuniladi.

Tizimlar bir-biridan tarkibi va qanday maqsadda qo‘llanilishi bilan ajratiladi.

Axborot tizimi deb, oldinga qo‘yilgan maqsadga erishish uchun axborotlarni saqlash, qayta ishlash va uzatish metodlari va vositalari yig‘indisiga aytiladi.

Obyektli yondoshishda axborot tizimi o‘zaro bog‘langan obyektlardan iborat. Axborot tizimi dasturiy ta’minoti asosiy va bajaruvchi qismlardan tashkil topadi. Bajaruvchi dasturlar asosiy sinflar usullaridan foydalangan holda foydalanuchi

bilan muloqotni ta'minlash uchun xizmat qiladi. Odatda bajaruvchi dasturlar vizual dasturlash vositalari yordamida yaratiladi. Asosiy dasturlar sinflardan tashkil topgan bo‘lib, konteyner sinflar va algoritmlar bu sinflarni yaratish jarayonini tezlashtiradi.

Ballar tahlili. Quyida talaba ballari tahlilini amalga oshirishga imkon beruvchi dastur keltirilgan. Dasturda vektor konteyner sinfidan va ba'zi algoritmlardan foydalanilgan:

```
#include<iostream>
#include<vector>
#include <numeric>
using namespace std;
const int maxi=1200;
class Balls{
vector<int> ball;
public:
Balls(){
ball.reserve(maxi);
};
void inp(int n){
if(ball.size()<ball.capacity()) ball.push_back(n);
}
void print(){
for(int i=0;i<ball.size();i++) cout<<ball[i]<<" ";
cout<<endl;
}
void show(){
cout<<"max="<<*max_element(ball.begin(),ball.end());
cout<<" min="<<*min_element(ball.begin(),ball.end());
cout<<" avg="<<accumulate(ball.begin(),ball.end(),0)/ball.size()<<endl;
};
};
int main(){
Balls D;
D.inp(3);D.inp(1);D.inp(2);
D.print();
D.show();
system("pause");
return 0;
}
```

Natija:

3 1 2

max=3 min=1 avg=2

Obyektlar vektori. Sinf **n** obyektli vektori quyidagicha yaratiladi: **vector<imya sinfa> vec(n)**

Elementlar initsializatsiyasi quyidagi tartibda amalga oshiriladi:

1. Ko‘zda tutilgan konstruktor yordamida sinf turidagi vaqtinchalik obyekt yaratiladi.
2. Vektor har bir elementiga nusxalovchi konstruktor qo‘llanadi, natijada har bir obyekt vaqtinchalik obyekt nusxasi bilan initsializatsiya qilinadi.
3. Vaqtinchalik obyekt o‘chiriladi.

Bunday initsializatsiya samaradorligi oddiy massivni initsializatsiya qilishdan past, chunki birinchidan vaqtinchalik obyekt yaratish va nusxalashga resurs kerak, ikkinchidan nusxalovchi konstruktor hisoblash jixatidan ko‘zda tutilgan konstruktorga qaraganla murakkabroqdir.

Loyihalash umumiy qoidasi quyidagicha: obyektlar vektori joylash uchun qulay. Agar umumiy elementlar soni oldindan ma'lum bo‘lsa, bo‘sh joy ajratib, so‘ngra elementlarni joylash lozim.

Konstruktorda xotira ajratiladi. Agar xotirada joy bo‘lsa, joylash amalga oshiriladi. Agar vektor bo‘sh bo‘lmasa, element o‘chiriladi.

Misol:

```
#include<iostream>
#include<vector>
#include <numeric>
using namespace std;
const int maxi=1200;
class Balls{
vector<int> ball;
public:
Balls(){
ball.reserve(maxi);
};
void inp(int n){
if(ball.size()<ball.capacity()) ball.push_back(n);
}
void print(){
for(int i=0;i<ball.size();i++) cout<<ball[i]<<" ";
```

```
cout<<endl;
}
void show(){
cout<<"max="<<*max_element(ball.begin(),ball.end());
cout<<" min="<<*min_element(ball.begin(),ball.end());
cout<<" avg="<<accumulate(ball.begin(),ball.end(),0)/ball.size()<<endl;
};
};
int main(){
Balls D;
D.inp(3);D.inp(1);D.inp(2);
D.print();
D.show();
system("pause");
return 0;
}
```

Natija:

3 1 2

max=3 min=1 avg=2

Xizmatchilar ro‘yxati. Xizmatchilar ro‘yxatini yarating. Ro‘yxatni kiritish va chiqarish amalga oshirilsin:

Xizmatchi sinfini yaratish.

Xizmatchi ma'lumotlarni kiritish va chiqarish usullarini yaratish. Xizmatchilar ro‘yxati sinfini yaratish.

Kiritish-chiqarish usullarini amalga oshirish.

```
#include<iostream>
#include<vector>
using namespace std;
class person{
string name;
int year;
public:
person(){};
person(string name,int year){
person::name=name;
person::year=year;
}
void show(){
cout<<"name="<<name<<" year="<<year<<endl;
}
};
```

```
int main(){
vector<person> v(3);
int i;
v[0]=person("bobbi",45);
v[1]=person("pit",30);
v[2]=person("smit",55);
for(i=0;i<3;i++) v[i].show();
cout<<endl;
return 0;
}
```

Tartiblangan ro‘yxat. Ro‘yxat va ikki tomonlama navbat sinfidan foydalanish. Ma'lumotlar tartiblangan holda saqlanuvchi ro‘yxat yarating.

```
#include<iostream>
#include<list>
using namespace std;
class SortList {
list<int> lt;
public:
SortList(){};
SortList(int*a, int*b){
lt=list<int>(a,b);
lt.sort();
lt.unique();
}
void del(int m){lt.remove(m);}
void ins(int m){list<int> p(1,m); lt.merge(p);}
void merge(SortList a){ lt.merge(a.lt);}
void print(){
list<int>::const_iterator p;
for(p=lt.begin();p!=lt.end();p++) cout<<*p<<" ";
cout<<endl;
};
};
int main() {
int a[]={3,3,1,2,2,5};
SortList ll(a,a+6); ll.print();
ll.del(1);ll.ins(4); ll.print();
system("pause");
return 0;
}
```

Natija:

1 2 3 5

Axborot-qidiruv tizimlari

Axborot-qidiruv tizimlari. Foydalanuvchi so‘roviga ko’ra, ma'lumotlarni kiritish, tizimlash, saqlash, chop etish kabi sodda ishlarini bajarish uchun mo‘ljallangandir. Bunday tizimlarni yaratishda assotsiativ konteynerlardan foydalanish samaradorlikni oshiradi. Chunksi assotsiativ konteynerlarda axborotni kiritish, o‘chirish va qidirish logarafmik vaqtni oladi. Axborot-qidiruv tizimlarini yaratishda birlamchi kalit tushunchasi katta ahamiyatga ega. Birlamchi kalit deb, har bir obyekt unikalligini ta'minlovchi maydon yoki maydonlar majmuasiga aytiladi. Birlamchi kalit assotsiativ massivlarda indeks rolini bajaradi.

Xizmatchilar ro‘yxati. Xizmatchilar ro‘yxatini yarating. Ro‘yxatni kiritish va chiqarish amalga oshirilsin:

Xizmatchi sinfini yaratish.

Xizmatchi ma'lumotlarni kiritish va chiqarish usullarini yaratish. Xizmatchilar ro‘yxati sinfini yaratish.

Kiritish-chiqarish usullarini amalga oshirish.

```
#include <iostream>
#include<set>
using namespace std;
class Person{
string name;
int age;
public:
Person(){};
Person(string s,int k):name(s),age(k){};
void show(){
cout<<name<<" "<<age<<endl;
};
friend bool operator<(Person a,Person b){
return a.name<b.name;
}
};
class TestSet{
set<Person> st;
public:
TestSet(){};
void insert(Person a){st.insert(a);};
void del(Person a){st.erase(a);};
```

```
void find(Person a){
if (st.count(a)!=0) cout<<"Yes"<<endl;
else cout<<"No"<<endl;
}
void show(Person a){
set<Person>::iterator pos;
pos=st.find(a);
if(pos!=st.end()) { a.show(); }
else cout<<"No";
};
};
int main(){
TestSet st;
st.insert(Person("alpha",1));
st.insert(Person("betta",2));
st.insert(Person("gamma",3));
st.find(Person("alpha",1));
st.show(Person("gamma",3));
system("pause");
return 0;
}
```

Natija:

Yes

gamma 3

Xizmatchilar ro‘yxati. Xizmatchilar ro‘yxatini yarating. Ro‘yxatni kiritish va chiqarish amalga oshirilsin:

Xizmatchi sinfini yaratish.

Xizmatchi ma'lumotlarni kiritish va chiqarish usullarini yaratish. Xizmatchilar ro‘yxati sinfini yaratish.

Kiritish-chiqarish usullarini amalga oshirish.

Misol:

```
#include <iostream>
#include<map>
using namespace std;
class TestMap{
struct Person{
string name;
int age;
public:
Person(){};
```

```
Person(string s,int k):name(s),age(k){};
void show(){
cout<<name<<" "<<age<<endl;
};
};
map<string,Person> st;
public:
TestMap(){};
void insert(string id, string s,int k){
st[id]=TestMap::Person(s,k);
};
void del(string id){
st.erase(id);
};
void find(string a){
if (st.count(a)!=0) cout<<"Yes"<<endl;
else cout<<"No"<<endl;
}
void show(string a){
map<string,TestMap::Person>::iterator pos;
pos=st.find(a);
if(pos!=st.end()) {
cout<<a<<" ";(pos->second).show();
}
else cout<<"No";
};
};
int main(){
TestMap st;
st.insert("I1","alpha",1);
st.insert("I2","betta",2);
st.insert("I3","gamma",3);
st.find("I1");
st.show("I3");
system("pause");
return 0;
}
```

Natija:

Yes

I3 gamma 3

Misol. Quyida multilug‘at sinfi berilgan bo‘lib, kalit sifatida **string** turi ishlatiladi, qiymat turi bo‘lsa, ixtiyoriy bo‘lishi mumkin.

```
#include <iostream>
```

```
#include<map>
using namespace std;
template<class T=string>
class Dictionary{
multimap<string,T> st;
public:
//
void insert(string a,T b){
st.insert(make_pair(a,b));
}
//
void find(string a){
typename multimap<string,T>::iterator where=st.find(a);
if (where!=st.end())
cout<<where->first<<" "<<where->second<<endl;
else cout<<"No";
}
//
void find_all(string a){
typename multimap<string,T>::iterator p1,p2,p;
p1=st.lower_bound(a);
p2=st.upper_bound(a);
for(p=p1;p!=p2;p++){
cout<<p->first<<" "<<p->second<<endl;
}
}
//
void statistic(){
map<string,int> mt;
typename multimap<string,T>::iterator pos;
for(pos=st.begin();pos!=st.end();poC++)
mt[pos->first]++;
map<string,int>::iterator p;
for(p=mt.begin();p!=mt.end();p++)
cout<<p->first<<" "<<p->second<<endl;
};
void print_all(){
 multimap<string,string>::iterator pos;
for(pos=st.begin();pos!=st.end();poC++)
cout<<pos->first<<" "<<pos->second<<endl;
}
};
int main(){
Dictionary<> st;
st.insert("alpha","1");
```

```
st.insert("betta","2");
st.insert("gamma","3");
st.insert("gamma","32");
st.find_all("gamma");
system("pause");
return 0;
}
```

Natija:
gamma 3
gamma 32

Misol. Quyida ishlarni rejalashtirish masalasini hal qilish uchun hosila konteynerlardan foydalanish ko‘rsatilgan:

```
#include <iostream>
#include<queue>
#include<vector>
using namespace std;
template<class T>
class PPlan{
priority_queue<T, vector<T>, greater<T> > qt;
public:
PPlan(T* a, T* b){
qt=priority_queue<T, vector<T>, greater<T> >(a,b);
}
void print(T time){
priority_queue<T, vector<T>, greater<T> > temp=qt;
T s1=T(),s=T(); T a;
while (!temp.empty()) {
a=temp.top();
s1+=a;
if(s1>time) break;
s=s1;
cout<<a<<" ";
temp.pop();
}
cout<<"\ns="<<s<<endl;
}
};
int main() {
int a[]={3,1,2,6};
PPlan<int> pt(a,a+4);
pt.print(4);
cout<<endl;
system("pause");
```

```
return 0;
}
```
Natija:
1 2
s=3

Qavslar to‘g‘riligini stek yordamida tekshirishga misol:

```
#include <iostream>
#include<stack>
using namespace std;
class TestStack{
stack<char> st;
public:
bool check(string a){
char c;
for(int i=0;i<a.size();i++){
c=a[i];
if (c!='('&&c!=')') return false;
if (st.size()==0||c=='(') {
st.push(c);continue;
}
if (st.top()=='(') st.pop();
else st.push(c);
};
if(st.size()==0) return true; else return false;
};
};
int main(){
TestStack st;
string a("(())");
cout<<st.check(a)<<endl;
system("pause");
return 0;
}
```
Natija:
1

O‘yin dasturlari

Kompyuter o‘yinlarini yaratishda qiziqarli va qulay interfeys yaratishga katta ahamiyat beriladi. Ayniqsa, multimediyali o‘yinlar katta qiziqish uyg‘otadi. Shu bilan birga intellektual o‘yinlar, bilim oshirishga qaratilgan o‘yinlar ommaviy

o‘yinlardan axhmiyati kattaroqdir. Quyida berilgan o‘yinlar oddiy interfeysga ega, lekin obyektli dasturlash tamoyillariga asoslangandir.

Uyum o‘yini. Quyida uyumda kim oxirgi bo‘ladi, o‘yini dasturi keltirilgan. O‘yin boshida M buyumlardan iborat uyum mavjud. Har bir yurishda ixtiyoriy uyum tanlanib ikkiga ajratiladi. Agar kimni yo‘lida ajratiladigan uyum qolmasa, u yutqazadi.

```
#include<iostream>
#include<vector>
using namespace std;
class Win{
int n;
public:
Win(int k) {n=k;};
int get_number(){return n;}
};
class IniError
{
int k;
public:
IniError(int k1) {k=k1;};
int get_value(){return k;}
};
class Error{
int n;
int k;
public:
Error(int k1,int k2) {n=k1;k=k2;};
int get_number(){return n;}
int get_value(){return k;}
};
class Play{
vector<int> pa;
int number;
int one;
void print(){
int k=pa.size();
for(int i=0;i<k;i++) cout<<pa[i]<<" ";
cout<<endl;
}
public:
Play(int s){
if (s<2) throw(IniError(s));
```

```
pa.push_back(s);
number=0;
if (s==1) one=1; else one=0;
}
void Step (int k,int index){
if((k<1) ||(k>=pa[index])) throw(Error(number,k));
pa.push_back(k);
pa[index]-=k;
if (k==1) one++;
if(pa[index]==1) one++;
if(one==pa.size()) throw(Win(number));
number=1-number;
print();
}
};
int main() {
int k;
int m;
int s=2;
try{
Play pa(s);
while(1){
cin>>k>>m;
pa.Step (k,m);
}
}
catch(Win a){
cout<<"Ura! "<<a.get_number()<<" o'yinchi yutdi";
}
catch(IniError a){
cout<<" qiymat "<<a.get_value()<<" hato";
}
catch(Error a){
cout<<a.get_number()<<" o'yinchi "<<a.get_value()<<" hato yurdi";
}
return 0;
}
```

O‘yin dasturi. Quyida so‘z topish o‘yinining dasturi keltirilgan. So‘z harfma-harf topiladi. Oxirgi harfni topgan yutadi. Hamma so‘zni birdaniga aytish mumkin. Noto‘g‘ri topilsa, o‘yinchi yutqazadi.

```
#include <iostream>
#include <string>
```

```
using namespace std;
class Play{
string st,tt;
public:
Play(string name){
st=name;
};
void play(){
cin>>s;
if(s==st) {cout<<"O'yinchi yuti!";
return;
}
if(s!=st)&&(s.size()>1) {cout<<"O'yinchi yutqazdi!";
return;
}
if(s!=st)&&(s.size()==1) {
for(int i=0;i<st.size();i++) {
if(st[i]==s[0]&&tt[i]==s[0]){cout<<"Bu harf bor";break;}
}
}
}
};
int main(){
system("pause");
return 0;
}
```

O‘yin dasturi. Quyida so‘z topish o‘yinining dasturi keltirilgan. So‘z harfma-harf topiladi. Oxirgi harfni topgan yutadi. Hamma so‘zni birdaniga aytish mumkin. Noto‘g‘ri topilsa, o‘yinchi yutqazadi.

```
#include <iostream>
#include <fstream>
#include <string>
#include<map>
using namespace std;
class TestMap{
map<string,bool> st;
public:
TestMap(string name){
string s;
ifstream in(name.c_str());
while(!in.eof()){
cin>>s;
```

```
st[s]=false;
}
};
void play(){
string s;
int k=0;
map<string,bool>::iterator pos;
cin>>s;
pos=st.find(s);
if(pos!=st.end()) {
if(!pos->second){
if(k==2){
cout<<"O'yinchi yutqazdi!";
return;
}
else k++;
}
else{
pos->second=true;k=0;
}
}
else{
cout<<"O'yinchi yutqazdi!";
return;
}
}
};
int main(){
TestMap st("hhh");
system("pause");
return 0;
}
```

Nazorat savollari:

1. Axborot tizimlari tushunchasi.
2. Obyektlar vektori.
3. Tartiblangan ro‘yxat yaratish usullari.
4. Axborot tizimlarini yaratishda Ro‘yxat konteyneri xususiyatlaridan foydalanish.
5. Axborot tizimlarini yaratishda ikki tomonlama navbat sinfidan foydalanish.

6. Axborot-qidiruv tizimlari.
7. Axborot-qidiruv tizimlari yaratishda To‘plam va xarita konteyneridan foydalanish.
8. Axborot-qidiruv tizimlari yaratishda multito‘plam va multixaritadan foydalanish.
9. Intellektual o‘yin dasturlarini yaratish.
10. Rejalashtirish va modellashtirishda hosila konteynerlar.

Misollar:

1. Chiziqli ro‘yxat yordamida avtobus saroyida avtobuslar mavjudligi to‘g‘risida axborotga ega bo‘lgan dasturni tuzing.
2. Chiziqli ro‘yxat kutubxonadagi kitoblar to‘g‘risida joriy axborotga ega bo‘lgan dastur tuzing.
3. Egiluvchan diskni to‘ldirishini modellashtiruvchi dastur yarating.
4. Uzoq yo‘nalish bo‘yicha yuradigan temir yo‘l vokzaldagi poyezdlarning avtomatlashgan axborot tizimi.
5. Xarita yordamida ingliz-o‘zbek lug‘atni shakllantiruvchi dastur yaratish. Lug‘at inglizcha so‘z, o‘zbekcha so‘z va so‘zga murojaatlar sonidan iborat bo‘lishi kerak.

ADABIYOT

Asosiy adabiyot:

1. Kernigan B., Ritchi D. YAzik programmirovaniya Si. M. Finansi i statistika. 1985.
2. Luis D. S i C++. Spravochnik., M: Binom, 1997.
3. Gradi Buch. Obyektno –orientirovannoy analiz i proektirovanie s primerami prilojeniy na C++. Nevskiy dialekt, 2001 g., 560 s.,
4. Grexem I. Obyektno orientirovannie metodi. Prinsipi i praktika. Vilyams., 2004, 879 s.,
5. Ivanova G.S. Obyektno orientirovannoe programmirovanie. Uchebnik., MGTU im Baumana, 2003, 320 s.
6. Asharina N.A. Osnovi programmirovaniya na yazikax Si, C++. Uchebniy kurs., M.: 2002
7. SHmidskiy YA.K. Prorammirovanie na yazike C++: Samouchitel. Uchebnoe posobie., Dialektika, 2004 g., 361 s.
8. Straustrup B. YAzik programmirovaniya C++. Trete izdanie, M.: Binom, 1999
9. Pol Ayra. Obyektno-orientirovannoe programmirovanie na C++. Vtoroe izdanie. – M.: Binom, 1999.

Qo‘shimcha adabiyot:

1. Krupnik A.B. Izuchaem C++. Piter. 2003, 251 s.
2. Meyers S. Naibolee effektivnoe ispolzovanie C++. 35 novix rekomendatsiy. DMK-Press, 2000, 304 s.
3. Nikolaenko D.V. Samouchitel po Visual C++., Spb, 2001.
4. Eldjer Dj. C++: kutubxona programmista, SPb: Piter, 1999.
5. Liberti D. Osvoy samostoyatelno C++: 10 minut na urok. Per s angl. Vilyams, 2004, 374 s.
6. SHildt G. Samouchitel C++. Vtoroe izdanie, SPb.: BHV, 1998.
7. Podbelskiy V.V. YAzik C++ – M.: Finansi i statistika, 1996.
8. Feyson T. Obyektno-orientirovannoe programmirovanie na C++ 4.5. – Kiev: Dialektika, 1996.
9. SHildt G. Teoriya i praktika C++., SPb.: BHV, 1996.

Test savollari

1. Qaysi ma'lumotlar turi suzuvchi vergul sonlar turiga kiradi?
A) char
B) int
C) void
+D) float
E) bool

2. Qaysi ma'lumotlar turi simvolli turga kiradi?
+A) char
B) int
C) void
D) float
E) bool

3. Mantiqiy turni ko‘rsating:
A) char
B) int
C) void
D) float
+E) bool

4. Qaysi ma'lumotlar turi butun sonlar turiga kiradi?
A) char
+B) int
C) void
D) float
E) bool

5. Uzunligi 32 bitdan kam bo‘lmagan ma'lumotni haqiqiy turi qaysi so‘z orqali ifodalanadi?
A) float
+B) double
C) short
D) int
E) long

6. Uzunligi 64 bitdan kichik bo‘lmagan ma'lumotning haqiqiy turi qaysi so‘z orqali ifodalanadi?
A) float
B) double
+C) long double
D) int
E) long

7. && va || amallar:
A) ikkita sonli qiymatlarni solishtiradi
B) ikkita sonli qiymatlarni kombinatsiyalaydi
+C) ikkita bulli (bool) qiymatlarni solishtiradi
D) ikkita bulli qiymatlarni kombinatsiyalaydi
E) to‘g‘ri javob yo‘q

8. Yangi tur kiritish amalini ko‘rsating:
+A typedef
B) define
C) sizeof
D) struct
E) to‘g‘ri javob yo‘q

9. Xotiradagi hajmni hisoblash amalini ko‘rsating:
A typedef
B) define
+C) sizeof
D) struct
E) to‘g‘ri javob yo‘q

10. Preprotsessor amalini ko‘rsating:
A typedef
+B) define
C) sizeof
D) struct
E) to‘g‘ri javob yo‘q

11.Qaysi turdagi ifodani qiymatlari mantiqiy deb hisoblanishi mumkin?
+A) har xil butun
B) har xil sonli
C) butun ishorasiz
D) haqiqiy
E) xoxlagan

12. Qanday shartlar bajarilganda X>Y && A<B rost qiymatga ega bo‘ladi?
+A) X>Y va A<B
B) X>Y va A>B
C) X<Y va A>B
D) X<Y va A<B
E) to‘g‘ri javob yo‘q

13. Qaysi simvol yordamida ko‘rsatkich aniqlaydigan manzil qiymatini olish mumkin?

A) '&'
B) '^'
+C) '*'
D) '%'
E) '@'

14. Qaysi simvol yordamida o‘zgaruvchi adresini olish mumkin?
+A) '&'
B) '^'
C) '*'
D) '%'
E) '@'

15. Modul amalini ko‘rsating:
+A) '&'
B) '^'
C) '*'
D) '%'
E) '@'

16. Ajratuvchi yoki amalini ko‘rsating:
+A) '&'
B) '^'
C) '*'
D) '%'
E) '@'

17. Razryadli kon'yunksiya amalini ko‘rsating:
+A) '&'
B) '^'
C) '*'
D) '%'
E) '@'

18. Ilova haqidagi noto‘g‘ri ibora aniqlansin:
A) ilova – shartli nom
+B) ilova – o‘zgaruvchi
C) ilova qayta belgilanmaydi
D) ilova nolga teng bo‘lmaydi
E) ilova uchun adres bo‘yicha qiymat olish avtomatik bajariladi

19. O‘zgaruvchilarning sonli qiymat o‘zlashtirish tartibini ko‘rsating:
a=b=c=10;
A) a,b,c
B) a,c,b

C) b,c,a
D) b,a,c
+E) c,b,a

20. Qaysi turli o‘zgaruvchi bilan razryad bo‘yicha amallarni qo‘llash mumkin emas:
+A) float
B) char
C) short int
D) int
E) to‘g‘ri javob yo‘q;

21. Massiv elementlaridan foydalanish nima orqali bajariladi:
A) FIFO yo‘nalish
B) LIFO yo‘nalish
C) nuqta amali
D) element ismi
+E) element indeksi

22. Ma'lumotlarni kiritish qaysi simvol orqali bajariladi?
A) '::'
B) '->'
+C) '>>'
D) '<<'
E) '=='

23. Ma'lumotlarni chiqarish qaysi simvol orqali bajariladi?
A) '::'
B) '->'
C) '>>'
+D) '<<'
E) '=='

24. Kvalifikatsion murojaat operatorini ko‘rsating:
+A) '::'
B) '->'
C) '>>'
D) '<<'
E) '=='

25. Qaysi so‘z yordamida ko‘p alternativli tanlash namoyish etiladi?
+A) switch
B) throw
C) public
D) struct

E) for

26. Qaysi so‘z yordamida shartli operator kiritiladi?
A) switch
B) throw
C) public
+D) if
E) for

27. Operator goto boshqaruvni nimaga uzatadi:
A) amaliga
+B) belgiga (metkaga)
C) o‘zgaruvchiga
D) funksiyaga
E) blokka

28. Siklni keyingi bosh iteratsiyasiga boshqaruvni qaysi operator yuboradi?
A) if
B) switch
C) break
+D) continue
E) goto

29. Qaysi so‘z yordamida sikldan chiqish bajariladi?
A) delete
B) new
C) void
+D) break
E) return

30. Qaysi so‘z yordamida dinamik xotira ajratiladi?
A) delete
+B) new
C) void
D) break
E) return

31. Qaysi so‘z yordamida dinamik xotira o‘chiriladi?
+A) delete
B) new
C) void
D) break
E) return

32. Shart operatori qaysi konstruksiya yordamida ifodalanadi?

A) while (ifoda_shart) {…}
B) do {…} while (ifoda_shart)
C) if (ifoda_shart) {…} else {…}
+D) if (ifoda_shart) else {…}
E) for (ifoda_1; ifoda_shart; ifoda_2){…}

33. So`ng shartli sikl qaysi konstruksiya yordamida bajariladi?
A) while (ifoda_shart) {…}
+B) do {…} while (ifoda_shart)
C) while (ifoda_shart) do {…}
D) for (ifoda _1; ifoda_shart; ifoda _2){…}
E) if (ifoda_shart) {…} else {…}

34. Quyidagi "while (ifoda_shart) {…}" konstruksiyadan foydalanish nimani bildiradi?
A) shartdan oldinsikl
B) shartdan keyingisikl
+C) parametriksikl
D) ko‘p alternativali to‘plam
E) shartli operator

35. Qaysi konstruksiya yordamida parametli sikl tavsiya etiladi?
A) while (ifoda_shart) {…}
B) do {…} while (ifoda_shart)
+C) for (ifoda _1 ifoda_shart; ifoda _2){…}
D) if (ifoda_shart) {…} else {…}
E) while (ifoda_shart) do {…}

36. Operatorlar bloki qaysi qavslar orqali belgilanadi?
+A) {…}
B) (…)
C) /…/
D) /*….*/
E) [….]

37. Quyida keltirilgan operatorlarning qaysilari sikl operatorlari?
A) if
+B) while
C) break
D) switch
D) to‘g‘ri javob yo‘q

38. Agar continue operatori sikl operatori ichida kelsa, u holda:
+A) u boshqaruvni siklning kelasi iteratsiyasining boshlanishiga uzatadi
B) u boshqaruvni siklning oldingi iteratsiyasining oxiriga uzatadi

C) u boshqaruvni belgidan keyin kelgan sikl iteratsiyasiga uzatadi
D) u boshqaruvni sikldan keyingi operatorga uzatadi
E) u boshqaruvni dastur boshiga uzatadi

39. Blokning ichida ifodalangan o‘zgaruvchi qachon ko‘rinadi?
A) ifodalangan nuqtadan dastur oxirigacha
B) ifodalangan nuqtadan funksiya oxirigacha
+C) ifodalangan nuqtadan blok oxirigacha
D) blok ichida
E) blok tashqarisida

40. Dasturda qaysi funksiya bo‘lishi shart?
A) local()
+B) main()
C) friend()
D) global()
E) inline()

41. Kutubxonali exit() funksiyasi nimadan chiqish uchun mo‘ljallangan:
A) o‘zi joylashgan sikldan
B) o‘zi joylashgan blokdan
C) o‘zi joylashgan funksiyadan
+D) o‘zi joylashgan dasturdan
E) o‘zi joylashgan fayldan

42. Qiymatni oluvchi funksiya o`z ichida qaysi so‘zlarni olishi shart?
A) delete
B) new
C) void
D) break
+E) return

43. Qiymatni olmagan funksiya qaysi so‘z orqali ifodalanadi?
A) delete
B) new
+C) void
D) break
E) return

44. Qo‘shimcha yuklangan funksiyalar:
A) bitta ismga ega bo‘lgan funkiyalar to‘plami
B) ular bir xil argumentlar va turlarga ega
C) dasturlash jarayonini osonlashtiradi
D) og`irlikni ko`tara olmasligi mumkin
+E) hamma javoblar to‘g‘ri

45. Agar funksiya inline so‘zi yordamida izohlangan bo‘lsa, kompilyatorning javobi qanday bo‘ladi?

A) kompilyator kompyuter xotirasida funksiya tashkil etadi

B) kompilyator kompyuter xotirasida funksiya tashkil etmaydi

+C) kompilyator kompyuter xotirasida funksiya tashkil etadi va uning nusxasini dastur kodiga ko`chiradi

D) kompilyator kompyuter xotirasida funksiya tashkil etadi, ammo uning nusxasini dastur kodiga ko`chirmaydi

E) to‘g‘ri javob yo‘q

46. Preprotsessor iostream.h preprotsessor fayli nimani bildiradi?

A) Ma'lumot fayllari bilan ishlash imkoni

B) Obyektni joylashtirish uchun xotirani dinamik ajratish

C) Obyektni joyashtirish uchun xotirani bo‘shatish

+D) Kiritish/chiqarish oqimlari bilan ishlash imkonini yaratish

E) Satrlar bilan ishlash imkonini yaratish

47. Qaysi so‘z yordamida struktura ta'riflanadi?

A) switch

B) throw

C) public

+D) struct

E) for

48. Qayd etish (perechislenie) qaysi kalitli so‘zdan boshlanadi?

+A) enum

B) struct

C) typedef

D) union

E) class

49. Birlashma qaysi kalitli so‘zdan boshlanadi?

A) enum

B) struct

C) typedef

+D) union

E) class

50. Struktura maydoniga murojaat etganda nuqtadan chapda qaysi operand joylashadi?

A) struktura maydoni

B) struktura ismi

+C) strukturali o‘zgaruvchi

D) strukturaning kalitli so‘zi

E) strukturaga ko‘rsatkich

51. Vorislik bu:
A) bir obyektga boshqa obyekt nusxasini qo‘shish
B) bir obyektga boshqa obyektga ilova qo‘shish
+C) bir sinfga boshqa sinf funksionalligini qo‘shish
D) sinf usullarini qayta ta'riflash
E) yagona obyektda ma'lumotlar va funksiyalarni jamlash

52. Inkapsulyasiya bu:
+A) yagona obyektda ma'lumotlar va funksiyalarni jamlash
B) bir obyektga boshqa obyekt nusxasini qo‘shish
C) yagona obyektda ma'lumotlar va shu ma'lumotlarga ko‘rsatkichlarni jamlash
D) yagona obyektda ma'lumotlar va shu ma'lumotlarga ilovalarni jamlash
E) bir sinf usullarini boshqasida qayta ta'riflash

53. Hamma obyektlar uchun umumiy bo‘lgan sinf a'zolari qaysi so‘z yordamida ta'riflanadi?
+A) static
B) protected
C) private
D) friend
E) public

54. Sinfdan tashqarida ta'riflangan funksiyaga sinf yopiq elementlariga murojaat huquqi qaysi so‘z yordamida beriladi?
A) static
B) protected
C) private
+D) friend
E) try

55. Qaysi so‘z yordamida faqat sinf ichida yoki uning avlodlarida sinfning a'zolaridan erkin foydalanish huquqini berish mumkin?
A) static
+B) protected
C) private
D) friend
E) public

56. Qaysi so‘z yordamida faqat sinf ichida sinfning a'zolaridan erkin foydalanish huquqini berish mumkin?
A) static
B) protected

+C) private
D) friend
E) public

57. Sinf a'zosiga sinf ichida va tashqarisida murojaat huquqini berish qaysi so‘z yordamida amalga oshiriladi?
A) switch
B) throw
+C) public
D) struct
E) protected

58. Sinf komponentasiga sinf nomi orqali murojaat qilish mumkin bo‘lishi uchun u qanday ta'riflanishi lozim?
A) static va protected
+B) static va public
C) static va private
D) friend va public
E) static va friend

59. Obyektni initsializatsiya qilish uchun ishlatiladigan usulni ko`rsating:
+A) konstruktor
B) destruktor
C) statik
D) bosh
E) initsializator

60. Qaysi javobda konstruktor xossasi to‘g‘ri ko‘rsatilgan?
A) konstruktor hech qanday turdagi qiymat qaytarmaydi
B) konstruktorga ko‘rsatkich ta'riflash mumkin emas
C) konstruktor adresini olish mumkin emas
D) konstruktorlar vorislikka o‘tmaydi
+E) hamma javoblar to‘g‘ri

61. Qaysi javobda destruktor xossasi to‘g‘ri ko‘rsatilgan?
A) destruktor parametrlarga ega emas
B) destruktorga ko`rsatkich ta'riflash mumkin emas
C) destruktor ko`zda tutilgan bo‘yicha chaqiriladi
D) destruktor sinfda yagonadir
+E) hamma javoblar to‘g‘ri

62. Agar sinf struct so‘zi bilan ta'riflangan bo‘lsa:
+A) hamma komponentalari umumiy hisoblanadi
B) hamma komponentalari xususiy hisoblanadi
C) hamma komponentalari himoyalangan hisoblanadi

D) hamma komponentalari statik hisoblanadi
E) usullarga ega bo`lolmaydi

63. Agar sinf class so‘zi bilan ta'riflangan bo‘lsa:
A) hamma komponentalari umumiy hisoblanadi
+B) hamma komponentalari xususiy hisoblanadi
C) hamma komponentalari himoyalangan hisoblanadi
D) hamma komponentalari statik hisoblanadi
E) usullarga ega bo‘lolmaydi

64. Agar funksiya sinf ichida ta'riflangan bo‘lsa, u:
A) tashqi (extern) funksiya hisoblanadi
B) virtual (virtual) funksiya hisoblanadi
C) statik (static) funksiya hisoblanadi
+D) joylashtiriluvchi (inline) funksiya hisoblanadi
E) lokal (local) funksiya hisoblanadi

65. Kechiktirilgan bog‘lanish mexanizmiga asoslangan usul qanday ataladi?
A) dinamik
+B) virtual
C) statik
D) asosiy
E) to‘g‘ri javob yo‘q

66. Qanday komponentalar vorislikka o‘tmaydi?
A) umumiy
B) himoyalangan
+C) xususiy
D) himoyalangan va xususiy
E) himoyalangan va umumiy

67. Qanday sinf abstrakt sinf deyiladi?
A) statik usulga ega sinf
+B) juda bo‘lmasa bitta sof virtual usulga ega sinf
C) hamma usullari virtual sinf
D) juda bulmasa bitta virtual usulga ega sinf
E) usullari bo‘lmagan sinf

68. Sof virtual funksiya to‘g‘ri ta'rifini ko‘rsating:
A) abstract tur funksiya_nomi(formal_parametrlar_ro`yxati) = 0;
B) virtual tur funksiya_nomi(formal_parametrlar_ro`yxati);
+C) virtual tur funksiya_nomi(formal_parametrlar_ro`yxati) = 0;
D) abstract tur funksiya_nomi(formal_parametrlar_ro`yxati);
E) tur funksiya_nomi(formal_parametrlar_ro`yxati) = 0;

69. Standart amallarni qo‘shimcha yuklash qaysi so‘z yordamida amalga oshiriladi?

+A) operator
B) catch
C) friend
D) try
E) template

70. Qo‘shimcha yuklash mumkin bo‘lmagan amalni ko‘rsating:

A) bo‘lish amali
B) razryadli qo‘shish amali
C) qiymat berish amali
D) razryadli surish amali
+E) shartli amal

71. Qayta yuklashga imkon beruvchi amalisni ko‘rsating:

A) ko‘rish sohasini ko‘rsatish amali
B) obyektga murojaat amali
C) preprotsessor amali
D) shartli amal
+E) to‘g‘ri ijavob yo‘q

72. Funksiya va sinflar shablonlarini ta'riflash qaysi so‘z yordamida amalga oshiriladi?

A) operator
+B) template
C) friend
D) try
E) catch

73. Sinf shabloni xususiyatini ko‘rsating:

+A) sinf shabloni alohida sinflarni qurish usulini belgilaydi
B) sinf shabloni alohida obyektlarni qurish usulini belgilaydi
C) sinf shabloni alohida obyektlarni initsializatsiya qilishusulini belgilaydi
D) sinf shabloni alohida sinflarni vorislikka o‘tish usulini belgilaydi
E) sinf shabloni alohida obyektlarni yo‘qotish usulini belgilaydi

74. Fayllar bilan ishlash uchun dasturga qaysi sarlavhali faylni qo‘shish lozim?

A) stdio.h
B) iostream.h
C) conio.h
+D) fstream.h
E) string.h

75. Fayl oxirini aniqlash uchun qanday funksiya ishlatiladi?
A) bad()
B) fail()
C) flush()
D) tellg()
+E) eof()

76. Fayl bilan ishlashda xatoni aniqlash uchun qanday funksiya ishlatiladi?
A) bad()
+B) fail()
C) flush()
D) tellg()
E) eof()

77. Faylni qo`shish rejimida ochib fayl ko‘rsatkichini fayl oxiriga joylashtiruvchi ochish rejimi qiymatini ko‘rsating:
+A) ios::app
B) ios::ate
C) ios::in
D) ios::nocreate
E) ios::noreplace

78. Qaysi obyekt – oqimlar dasturda main chaqirilishidan oldin ochilgan?
+A) extern istream cin; extern ostream cout; extern ostream cerr;
B) extern ostream cin; extern ostream cout; extern ostream cerr;
C) extern istream cin; extern istream cout; extern ostream cerr;
D) extern ostream cin; extern ostream cout; extern istream cerr;
E) extern ostream cin; extern istream cout; extern ostream cerr;

79. Qaysi manipulyator uchun #include <ionamip> ulanishi lozim?
A) endl
B) dec
C) hex
+D) setfill
E) ws

80. Qaysi manipulyator bo‘shlik simvollarini o‘tkazadi?
A) oct
B) dec
C) hex
D) setfill
+E) ws

81. Qaysi manipulyator sakkizlik tizimda chiqarishga imkon beradi?
+A) oct

B) dec
C) hex
D) setfill
E) ws

82. Qaysi manipulyator o‘n oltilik tizimda chiqarishga imkon beradi?
A) endl
B) dec
+C) hex
D) setfill
E) ws

83. Qaysi manipulyator yangi simvol belgisini qo‘yib ostream buferini tozalaydi?
+A) endl
B) dec
C) hex
D) setfill
E) ws

84. Qaysi manipulyator bo‘sh joyni berilgan simvol bilan to‘ldirishga imkon beradi?
A) setprecision
B) setw
C) setbase
+D) setfill
E) to‘g‘ri javob yo‘q

85. Qaysi manipulyator suzuvchi vergulli son aniqligini o‘rnatishga imkon beradi?
+A) setprecision
B) setw
C) setbase
D) setfill
E) to‘g‘ri javob yo‘q

86. Qaysi manipulyator kiritish chiqarish maydoni kengligini o‘rnatadi?
A) setprecision
+B) setw
C) setbase
D) setfill
E) to‘g‘ri javob yo‘q

87. Qaysi manipulyator butun sonlarni berilgan sanoq tizimida chiqarishga imkon beradi?
A) setprecision

B) setw
+C) setbase
D) setfill
E) to‘g‘ri javob yo‘q

88. Istisnolarni generatsiya qilish qaysi so‘z yordamida amalga oshiriladi?
A) switch
+B) throw
C) public
D) struct
E) class

89. Istisnolarni qayta ishlash blokini e'lon qilish qaysi so‘z yordamida amalga oshiriladi?
A) operator
+B) catch
C) friend
D) try
E) throw

90. Istisnolar hosil qilishi mumkin bo‘lgan blokni e'lon qilish qaysi so‘z yordamida amalga oshiriladi?
A) operator
B) catch
C) friend
+D) try
E) operator

91. Konteyner chiziqli ro‘yxat qaysi so`z bilan beriladi?
A) vector
B) deque
C) stack
+D) list
E) queue

92. Konteyner ikki tomonli navbat:
A) vector
+B) deque
C) stack
D) list
E) queue

93. Konteyner to‘plam hamma elementlari unikal:
A) multimap

+B) set
C) map
D) multiset
E) queue

94. Konteyner to‘plam hamma elementlari unikal bo‘lishi shart emas:
A) multimap
B) set
C) map
+D) multiset
E) queue

95. Konteyner karta har bir kalit bilan bitta qiymat bog‘langan:
A) multimap
B) set
+C) map
D) multiset
E) queue

96. Konteyner karta har bir kalit bilan bir yoki bir nechta qiymat bog‘langan:
+A) multimap
B) set
C) map
D) multiset
E) queue

97. Birinchi element o‘chirish usulini ko‘rsating
A) push_back
B) push_front
C) pop_back
+D) pop_front
E) front

98. Oxiriga qo‘shish usulini ko‘rsating
+A) push_back
B) push_front
C) pop_back
D) pop_front
E) front

99. Qiymat berish usulini ko‘rsating
A) back
B) front
C) insert
D) erase

+E) assign

100. O‘chirish usulini ko‘rsating
A) back
B) front
C) insert
+D) erase
E) assign

MUNDARIJA