Московский государственный технический университет
имени Н.Э. Баумана

А.Ю. Быков

Решение задач на языках программирования Си и Си++

Методические указания к выполнению лабораторных работ

Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2017

УДК 004.43
ББК 32.973-018.1
Б95

Издание доступно в электронном виде на портале *ebooks.bmstu.ru* по адресу: http://ebooks.bmstu.ru/catalog/117/book1572.html

Факультет «Информатика и системы управления»
Кафедра «Информационная безопасность»

Рекомендовано Редакционно-издательским советом МГТУ им. Н.Э. Баумана в качестве учебно-методического пособия

Быков, А. Ю.

Б95 Решение задач на языках программирования Си и Си++ : методические указания к выполнению лабораторных работ / А. Ю. Быков. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2017. — 244, [4] с. : ил.

ISBN 978-5-7038-4577-6

Рассмотрены особенности решения практических задач на языках программирования Си и Си++. Представлены примеры программирования с комментариями. Описаны возможности библиотеки классов MFC.

Для студентов, обучающихся по специальностям «Компьютерная безопасность» и «Информационная безопасность автоматизированных систем», а также по другим специальностям с изучением программирования.

УДК 004.43
ББК 32.973-018.1

© МГТУ им. Н.Э. Баумана, 2017
© Оформление. Издательство МГТУ им. Н.Э. Баумана, 2017

ISBN 978-5-7038-4577-6

Предисловие

Одним из наиболее распространенных языков программирования, особенно в операционных системах типа Linux, является язык Си++, также широко распространен его предшественник — язык Си. Предлагаемые вниманию учащихся методические указания по дисциплине «Алгоритмические языки» посвящены решению задач с применением этих языков. Издание состоит из двух частей. Первая часть включает в себя девять лабораторных работ, призванных изучить основные средства языка Си, вторая — состоит из шести работ, рассматривающих основные средства языка Си++ и библиотеки MFC.

Цель выполнения таких лабораторных работ заключается в получении студентами навыков и умений практической разработки программ на указанных языках и расширении знаний об этих языках.

Первая часть лабораторных работ призвана научить студентов разрабатывать программы, использующие следующие синтаксические конструкции и возможности языка Си (с некоторыми элементами языка Си++):

- операции для расчета выражений;
- условный оператор;
- операторы циклов;
- массивы, в том числе динамические;
- структуры;
- функции;
- динамические структуры данных, такие как линейные списки;
- функции ввода-вывода в файлы;
- возможности графического интерфейса пользователя для операционной системы Windows.

Вторая часть лабораторных работ обучит студентов разработке программ, использующих следующие синтаксические конструкции и возможности языка Си++:

- классы;
- перегрузка стандартных операций;
- наследование классов;

- абстрактные классы и полиморфизм;
- потоковая многозадачность;
- библиотека классов MFC.

При выполнении лабораторных работ целесообразно использовать программный продукт Microsoft Visual Studio версии 2013 или более поздних. Студенты могут получить этот программный продукт бесплатно, воспользовавшись программой DreamSpark (бывшая MSDN AA) [1], для этого достаточно зарегистрироваться на сайте библиотеки МГТУ им. Н.Э. Баумана.

По каждой лабораторной работе студенты должны подготовить отчет, в котором необходимо последовательно и полно представить все основные шаги алгоритма решения задачи. В тексте программы следует представить соответствующие комментарии.

В содержание отчета включают титульный лист, обозначают цель работы и условие задачи, представляют программу с комментариями, результаты работы программы (скриншот экрана). При необходимости можно представить результаты расчета контрольного примера небольшой размерности, разработанной программой, и ручной расчет этого же контрольного примера. В обязательном порядке в заключении к отчету приводятся выводы.

Типовые варианты лабораторных работ представлены в приложении к методическим указаниям. Студенты выбирают вариант в соответствии с номером студента внутри группы, представленным в системе «Электронный университет».

Введение

Язык программирования Си, разработанный в США сотрудниками фирмы Bell Lab в начале 1970-х гг. для операционной системы (ОС) UNIX, вместе с языком Си++, который дополнительно к возможностям языка Си включает в себя объектно-ориентированные средства, — наиболее распространенные языки программирования.

Первое описание языка Си дано его авторами — Б. Керниганом и Д. Ритчи [2]. Язык Си иногда называют языком программирования среднего уровня. С одной стороны, язык поддерживает операции низкого уровня (операции над битами), а базовые типы отражают те же объекты, что и язык Ассемблера (байты, машинные слова, символы, строки). С другой — он имеет основные управляющие конструкции, присущие языкам высокого уровня. Таким образом, язык Си можно использовать для решения как системных, так и прикладных задач (хотя язык создавался прежде всего для системного программирования).

Язык программирования Си++ разработан фирмой Bell Labs в начале 1980-х гг., создателем языка считается Бьерн Страуструп [3]. Он предложил ряд усовершенствований к языку Си, главное, включил в него объектно-ориентированные средства. Первоначально новый язык называли «Си с классами», название «Си++» появилось в 1983 г.

Синтаксис языка Си++ (Си) оказался настолько удачным, что многие языки программирования, созданные позднее для других целей, например для разработки интернет-приложений, наследовали основные элементы синтаксиса языка Си++. К таким языкам относятся: Java, C# (Си шарп), PHP, JavaScript. Изучив язык Си++, освоить данные языки можно достаточно быстро.

Часть 1
РЕШЕНИЕ ЗАДАЧ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ С ЭЛЕМЕНТАМИ ЯЗЫКА СИ++

Лабораторная работа № 1.1
Изучение операций языка Си. Программирование линейных и разветвляющихся алгоритмов

1.1.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, имеющих линейную структуру, и программ, реализующих разветвляющиеся алгоритмы, т. е. использующих условный оператор и (или) оператор-переключатель. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные константам и переменным, а также условному оператору и переключателю языка Си [2, 4];
- разработать программы на языке Си для решения предложенных вариантов заданий;
- отладить программы;
- выполнить с помощью программы решение контрольного примера и его ручной расчет.

Выполнив работу, нужно подготовить отчет.

1.1.2. Краткая характеристика объекта изучения. Понятия переменной и типа данных

Одним из основных понятий, используемых в процедурных языках программирования, является понятие переменной. **Переменная** — это поименованная либо адресуемая иным способом область памяти, адрес

которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (т. е. по указанному адресу памяти), называются значением этой переменной. Переменная принадлежит определенному типу данных.

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- множество значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

Переменная используется для хранения данных в процессе выполнения программ. Значение переменной можно изменять с помощью операции присваивания.

Стандартные типы языка Си

В языке Си существуют стандартные типы, которым могут принадлежать переменные.

В языке Си стандарта ANSI классификация типов имеет следующий вид (также указана размерность переменной типа в оперативной памяти):

- целые:
  - — беззнаковые:
    `unsigned char` — переменная этого типа занимает в памяти 8 бит;
    `unsigned short` — 16 бит;
    `unsigned int` — 32 бита (в Win32);
    `unsigned long` — 32 бита (в Win32);
  - — знаковые:
    `char` — 8 бит;
    `short` — 16 бит;
    `int` — 32 бита (в Win32);
    `long` — 32 бита (в Win32);
- вещественные:
  `float` — 32 бита;
  `double` — 64 бита;
  `long double` — 80 бит.

Назначение и диапазоны стандартных типов представлены в табл. 1.1.

Следует отметить особенность типа char (unsigned char) в языке Си, отличающегося слабой типизацией. Поэтому символьный тип исполь-

зуется двояко: с одной стороны, для хранения кодов символов в кодировке ASCII, с другой — для хранения целых чисел в заданном диапазоне.

Таблица 1.1

Назначение и диапазоны стандартных типов

Тип данных	Размер, бит	Диапазон значений	Назначение типа
unsigned char	8	0...255	Небольшие целые числа без знака и коды символов
char	8	–128...127	Небольшие целые числа со знаком и ASCII-коды
unsigned short	16	0...65 535	Целые числа без знака
short	16	–32 768...32 767	Целые числа со знаком
unsigned int	32	0...4 294 967 295	Большие целые числа без знака
int	32	–2 147 483 648... ...2 147 483 647	Большие целые числа со знаком
unsigned long	32	0...4 294 967 295	Большие целые числа без знака
long	32	–2 147 483 648... ...2 147 483 647	Большие целые числа со знаком
float	32	3.4E–38...3.4E + 38 (по модулю)	Научные расчеты (7 значащих цифр)
double	64	1.7E–308...1.7E + 308 (по модулю)	Научные расчеты (15 значащих цифр)
long double	80	3.4E–4932...3.4E + 4932 (по модулю)	Финансовые расчеты (19 значащих цифр)

Дополнительно компанией Microsoft введены так называемые платформенно-независимые целые типы, по сути, новые имена для уже существующих типов, но их размерность не будет изменяться при переходе к другим платформам (размерность этих типов в битах явным образом присутствует в названии):

```
__int8                  unsigned __int8
__int16                 unsigned __int16
__int32                 unsigned __int32
__int64 (long long)     unsigned __int64(unsigned long long)
```

Кроме того, дополнительно введены следующие типы, имена которых являются ключевыми словами:

`bool` — 1 байт, возможные значения `true` (1) и `false` (0);

`wchar_t` — 2 байта (соответствует типу `unsigned short`), обычно используется для хранения кодов символов в кодировке `Unicode` (1 символ кодируется 2 байтами).

Объявления переменных в языке Си

Общий формат объявления переменных:

```
[<спецификатор_класса_памяти>] [<модификатор>] <тип>
<имя1>[=<инициализатор1>], <имя2>[=<инициализатор2>],…,
<имяN>[=<инициализаторN>];
```

Необязательный элемент <спецификатор_класса_памяти> (необязательный элемент далее в записи форматов будет помещаться в квадратные скобки) — одно из ключевых слов: `auto`, `register`, `static`, `extern`.

Необязательный элемент <модификатор> — ключевые слова `const`, `volatile` и др.

Модификатор `const` означает, что инициализация (присвоение начального значения) переменной обязательна, и далее значение переменной изменять нельзя. Модификатор `volatile` означает, что данная переменная может быть изменена где-то в другом месте, а необязательно в этом потоке, например, в другом потоке (модификатор влияет на режим оптимизации работы компилятора).

Пример:

```
int i, j=10;
const float pi=3.14; /* Далее запрещено изменять значение
pi */
extern double x; /* Это есть описание, определение пе-
ременной в другом месте */
unsigned char C1='A', C2=10, C3;
```

Константы в языке Си

Кроме переменных в программе часто используются константы.

Константа — это значение, которое не может быть изменено в процессе работы программы.

В языке Си выделяют следующие разновидности констант, которые могут иметь разные типы и формы представления.

1. Целые константы

Целые константы существуют для представления в программе целых значений; константы могут иметь разные типы. Тип константы определяет ее представление в оперативной памяти: кодировку константы, объем в байтах, какой набор значений может представлять константа, а также операции, которые можно выполнять с константой. Существуют следующие типы целых констант:

• `int` (по умолчанию); в оперативной памяти константа данного типа кодируется, в Win32 — 4 байтами, примеры: `1245`, `6`, `175`, `5`, `1425`;

• `long`; в оперативной памяти константа кодируется 4 байтами в Win 32 и будет иметь этот тип, если диапазон выходит за тип `int` в операционных системах, в которых тип `long` занимает больше места, чем тип `int` или явно указывается тип с помощью суффикса `l (L)`, примеры: `12l`, `14567L`, `125234L`;

• `unsigned int` и `unsigned long`; для явного указания необходимо использовать суффикс `u (U)`, примеры: `105u (unsigned int)`, `105ul(unsigned long)`.

Целые константы всегда задают неотрицательные значения, для задания отрицательного значения используют операцию «унарный минус», которую применяют к целой константе: –1000, –200l.

Целые константы также могут иметь различные формы представления. Формы представления введены для удобства программиста, одно и то же значение константы может быть представлено в различных формах в исходном тексте программы, при этом после компиляции полученный исполняемый код и кодирование константы в памяти будут идентичны и не будут зависеть от формы представления. Для целых констант существуют три формы представления:

• в десятичной системе счисления (по умолчанию) — 1234, 378l, 346;

• в шестнадцатеричной системе счисления признаком данного представления являются два первых символа константы `0x` (или `0X`), в константе можно использовать шестнадцатеричные цифры: 0...9, *A* (*a*), *B* (*b*), ..., *F* (*f*), примеры: `0x10`, `0x10acd`, `0XFFFF`;

• в восьмеричной системе счисления признаком данного представления является первый символ 0, в константе можно использовать восьмеричные цифры: 0...7, примеры: `010`, `070`, `01237`;

2. Вещественные константы

Вещественные константы служат для представления вещественных значений, которые могут иметь целую и дробную части. По аналогии с целыми константами, существуют различные типы и формы пред-

ставления вещественных констант. Для вещественной константы можно применять операцию «унарный минус» в целях получения отрицательного значения.

Существуют вещественные константы следующих трех типов:

• `double` (по умолчанию); в оперативной памяти константа кодируется 8 байтами, примеры: `12.5`, `.123`, `0.5`, `1.` (наличие точки в таком представлении обязательно);

• `float` — 4 байта, используется суффикс *f (F)*, примеры: 10.5*f*, 0.123*F*;

• `long double` — 10 байт, используется суффикс *l (L)*, примеры: 10.5*l*, 0.9*L*.

Есть две формы представления вещественных констант:

1) форма с точкой (десятичная форма), примеры: `10.125`, `1.` (значение 1.0), .125 (значение 0.125), 0.125, представляется в форме десятичной дроби, состоит из трех основных элементов целой части, точки (наличие точки обязательно), дробной части, причем целая или дробная часть могут отсутствовать, тогда они считаются равными 0;

2) форма со знаком экспоненты (экспоненциальная форма), примеры: *1e*-5, 12.23*E*4*F* (тип константы `float`), представляется в виде мантиссы и порядка: мантисса записывается слева от знака экспоненты (*E* или *e*), порядок — справа; значение константы определяется как произведение мантиссы и возведенного в указанную в порядке степень числа 10.

3. Символьные константы

Изначально в языке Си стандарта ANSI символьные константы представлялись одним символом, который мог быть буквой, цифрой, знаком пунктуации или специальный символом, заключенным в апострофы, примеры: `'a'`, `'d'`, `'1'`, `'.'`, `' '`. Данные константы представлены в памяти типом `char` и занимают 1 байт. Значением символьной константы является числовое значение кода символа в кодировке, используемой в данной операционной системе, например в кодировке ASCII.

Символьные константы могут участвовать в арифметических операциях на правах полноценных целых чисел, хотя чаще их используют как символы. Некоторые символы представляются в символьных и строковых константах с помощью специальных управляющих последовательностей символов, начинающихся с `'\'` (косая обратная наклонная черта), называемых escape-последовательностями. Примеры символов: `'\\'` — обратный слеш, `'\''` — апостроф, `'\"'` — двойные кавычки, `'\n'` — конец строки (код 10), `'\r'` — возврат каретки (код 13), `'\a'` — звуковой сигнал (код 7), `'\b'` — возврат на шаг (забой) (код 8). Кроме того, любой символ можно представить в виде его

кода в восьмеричной или шестнадцатеричной системе счисления в форматах '\ooo' или '\xhh', где o — восьмеричная цифра, h — шестнадцатеричная цифра (значение восьмеричного кода не может превышать 255). Например, символ пробел с десятичным кодом 32 может быть записан как ' ', в шестнадцатеричном коде — '\x20', в восьмеричном — '\40'.

В некоторых компиляторах, например в Visual C++, разрешены многосимвольные константы (до четырех символов): 'asdf', 'GR', они представлены в памяти типом int (первый символ — младший байт).

4. Строковые константы

Это последовательность символов, заключенных в кавычки (не в апострофы). Внутри строковых констант допускается использовать escape-последовательности или коды символов, пример: "начало строки\nтекст с новой строки", при печати текст будет выводиться на две строки. Формально строковая константа является массивом символов. Во внутреннем представлении строки в оперативной памяти в конце присутствует нулевой символ '\0', так что физический объем памяти для хранения строки превышает количество символов, записанных между кавычками, на единицу.

5. Константы типа перечислений

Можно создавать перечисляемый тип, содержащий константы. Переменным этого типа можно присваивать значения только этих констант. Данные константы представляются в памяти точно так же, как константы типа int.

Формат перечислений следующий:

```
enum <имя_типа> {
    <имя1>[=<инициализатор1>],
    <имя2>[=<инициализатор2>],
    .........
    <имяN>[=<инициализаторN>] };
```

Идентификаторы <имя1>, <имя2>, ..., <имяN> выступают далее в качестве констант, по умолчанию, если нет инициализатора, первая константа инициализируется 0, каждая последующая — на 1 больше.

Пример перечисления:

```
enum A
{
a, // 0
b, // 1
```

```
c=10, // 10
d // 11
};
```

Далее в программе обращаться к данным константам следует таким образом: `int i=b;` переменной *i* будет присвоено значение 1.

6. Нулевой указатель

Существует еще одна **константа** — это так называемый **нулевой указатель**, для задания которого введена именованная константа NULL, используется для задания значения указателя, который ни на что не указывает, обычно соответствует значению 0, но не обязательно во всех системах.

Операторы-выражения и операции языка Си

Для вычисления значений используют так называемые **операторы-выражения**. Их строят из операндов и знаков операций. Выражение задает правило вычисления некоторого значения. Проведем следующую классификацию выражений:

- на базе операции «присваивания» «=»: `x=y+10;`
- на базе операций «инкремент», «декремент» (++ --): `i++; --j;`
- вызов функции: `f1();`
- комбинированные выражения: `x=i++ + f()`.

Под **операцией** будем понимать некоторое действие, выполняемое над операндами (аргументами операции). Результат операции — всегда некоторое значение определенного типа, которое может быть использовано справа от операции присваивания (может быть присвоено некоторой переменной).

Операции языка Си подразделяются на следующие классы: унарные, бинарные, существует одна тернарная операция.

Унарные операции:

«–» — «унарный минус», применяется к арифметическим операндам (целым и вещественным переменным или константам), результат операции — значение операнда с противоположным знаком;

«+» — «унарный плюс», ничего не делает, введен для симметрии с операцией «унарный минус»;

«*» — «обращение по адресу», применяется к указателям, результат операции — значение объекта (операнда), на который указывает указатель;

«&» — «получение адреса», результат операции — адрес объекта (переменной);

«~» — «поразрядное отрицание», применяется только к целым операндам, результат операции — целое значение, в котором разряды исходного операнда инвертированы;

«!» — «логическое отрицание» («логическое НЕ»), дает в результате значение 0, если операнд есть истина (не нуль), и значение 1, если операнд равен нулю (в Visual C++ тип результата bool); следует отметить, что в базовом Си стандарта ANSI отсутствовал в явном виде логический тип, который бы принимал два значения: «истина» и «ложь», вместо логического типа использовался, как правило, целый тип, значение 0 интерпретировалось как «ложь», любое значение, отличное от 0, как «истина»;

(<тип>) — «преобразование типа»

«sizeof» — «определение размера», предназначается для вычисления размера объекта или типа в байтах и имеет две формы:

sizeof выражение или sizeof(выражение)

sizeof(тип)

++ — «инкремент» (увеличение на 1), -- — «декремент» (уменьшение на 1) имеют две формы записи — префиксную, когда операция записывается перед операндом, и постфиксную.

Если операции используются сами по себе (в операторе только одна операция), то разницы между двумя формами нет.

Если операция применяется внутри выражения с другими операциями, то в префиксной форме сначала изменяется операнд, а затем его новое значение подставляется в выражение, а в постфиксной форме в выражение подставляется старое значение, а затем изменяется значение операнда. Например,

```
int i=10, j;
j=++i; // Префиксная форма операции
```

В результате выполнения данного фрагмента i и j будут равны 11 (переменной j присваивается новое значение i, увеличенное на 1), если изменить форму операции ++:

```
int i=10, j;
j=i++; // Постфиксная форма операции
```

то после выполнения i будет равно 11, а j — 10, переменной j присваивается старое значение переменной i, а затем оно увеличивается на 1. Операции чаще применяют к целым операндам, но их можно применять к вещественным операндам и даже к указателям.

Бинарные операции можно разделить на следующие подклассы:

Арифметические:

«+» — бинарный плюс;

«–» — бинарный минус;

«*» — умножение;

«/» — деление;

% — получение остатка от деления.

Первые четыре операции применяются к арифметическим операндам: целым или вещественным, операции «+» и «–» ограниченно могут применяться к указателям. Операция «%» применяется только к целым операндам.

Логические:

«&&» — логическое «И»;

«||» — логическое «ИЛИ».

Операнды логических операций могут иметь тип bool, арифметический тип (целый или вещественный) или быть указателями. Каждый операнд оценивается с точки зрения его эквивалентности нулю (операнд, равный нулю, рассматривается как «ложь», не равный нулю — как «истина»). В Visual C++ тип результата операции — bool.

Поразрядные:

«&» — поразрядное «И»;

«|» — поразрядное «ИЛИ»;

«^» — поразрядное исключающее «ИЛИ»;

«>>» — поразрядный сдвиг вправо;

«<<» — поразрядный сдвиг влево.

Данные операции применяются только к целочисленным операндам и работают с их двоичными представлениями. При выполнении операций «&», «|», «^» операнды сопоставляются побитово (первый бит первого операнда с первым битом второго, второй бит первого операнда со вторым битом второго и т. д.).

Операции сдвига сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом. При сдвиге влево «<<» освободившиеся разряды обнуляются. При сдвиге вправо «>>» освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом — в противном случае.

Операции отношения (*сравнения*):

«==» — равно (не путать с операцией присваивания «=»);

«!=» — не равно;

«>» — больше;

«<» — меньше;

«>=» — больше или равно;

«<=» — меньше или равно.

Операции «отношения» сравнивают первый операнд со вторым. Операнды могут быть арифметического типа или указателями. Результатом операции является значение «истина» (любое значение, не равное 0, как правило, 1) или «ложь» (0). В Visual C++ тип результата `bool`.

Операции «присваивания»:

«=» — простое присваивание.

Первый операнд должен быть *L*-значением, т. е. областью памяти, куда будет помещен результат операции, второй — выражением. Сначала вычисляется выражение, стоящее в правой части операции, а потом его результат записывается в область памяти, указанную в левой части. Так называемое ***L*-значение (*L-value*)** (леводопустимое значение — может быть использовано слева от операции «присваивания») обозначает любое выражение, адресующее некоторый участок памяти, в который можно занести значение.

«`op`=» (где `op` — символ бинарной операции) — комбинированное присваивание, комбинация бинарной операции с операцией «присваивания», например, «+=» — присваивание со сложением; по аналогии, существуют операции: «*=», «/=», «–=», «%=», «&=», «|=», «^=» и др. Комбинированные операции работают по следующему правилу:

`i+=10;` аналогично `i=i+10;`

Перечислим *другие бинарные операции*:

`()` — вызов функции;

`[ ]` — обращение к элементу массива;

«`.`» (`«точка»`) — обращение к полю переменной структурного типа;

«`->`» — обращение к полю переменной структурного типа через указатель;

«`,`» (`«запятая»`) — последовательное вычисление; может ставиться между выражениями (выражения вычисляются последовательно), результат операции — результат второго операнда (выражения).

Тернарная операция:

«`?:`» — условная операция.

Формат: `<операнд1> ? <операнд2> : <операнд3>`

Первый операнд имеет тип, заменяющий логический, — арифметический или указатель, если первый операнд имеет значение «истина», то результат операции — значение второго операнда, а если «ложь», то результат операции — значение третьего операнда. Пример:

```
y= x>=0 ? x : -x;
```

переменной *y* присваиваются значения модуля переменной *x*.

Если в одном выражении присутствует несколько разных операций, они выполняются в соответствии с приоритетами — в первую очередь с высоким приоритетом. Каждая операция в языке Си имеет свой приоритет. Всего существует 15 классов приоритетов. Если в одном выражении присутствует несколько одинаковых операций, они могут выполняться, или слева направо, или справа налево; это определяет такое свойство операций, которое называется **ассоциативностью** (порядок выполнения операции в выражении).

Приоритеты и ассоциативность операций языка Си представлены в табл. 1.2, операции даны в порядке убывания приоритета.

Таблица 1.2

Приоритеты и ассоциативность операций языка Си

Приоритет (ранг)	Операции	Наименование	Ассоциативность
1	`( )  []  ->  .`	Первичные	→
2	`!  ~  +  -  ++  --  &  *  (тип)  sizeof`	Унарные	←
3	`*  /  %`	Мультипликативные	→
4	`+  -`	Аддитивные	→
5	`«  »`	Поразрядный сдвиг	→
6	`<  <=  >=  >`	Отношение (сравнение)	→
7	`==  !=`	Отношение (сравнение)	→
8	`&`	Поразрядное «И»	→
9	`^`	Поразрядное исключающее «ИЛИ»	→
10	`|`	Поразрядное «ИЛИ»	→
11	`&&`	Логическое «И»	→
12	`||`	Логическое «ИЛИ»	→
13	`?:`	Условные	←
14	`=  *=  /=  %=  +=  -=  &=  ^=  |=  <<=  >>=`	Простое и составное присваивание	←
15	`,` (операция «запятая»)	Последовательное вычисление	→

Для изменения порядка выполнения операций используют круглые скобки. Примеры:

```
y = a + b * 10; /* В первую очередь выполняется *, затем +, далее = */
y = (a + b) * 10; /* В первую очередь выполняется + (из-за скобок), затем *, далее = */
a = b = c = 100; /* Операции = выполняются справа налево, всем переменным будет присвоено значение 100 */
```

Условный оператор

Условный оператор служит для бинарного ветвления фрагмента исходного кода программы, в зависимости от условия («истина» или «ложь») выполняется один или другой фрагмент кода (оператор).

Формат условного оператора:

```
if (<выражение>) <оператор1>
[ else <оператор2> ]
```

Примечание. *Далее при записи форматов различных конструкций примем следующие обозначения: оператор или выражение заключаем в скобки <>, точка с запятой («;») после оператора при описании формата не ставится, так как считается, что она входит в состав оператора или это составной оператор (блок), а элемент, заключенный в квадратные скобки* `[]` *(за исключением использования массивов), считается необязательным, т. е. может отсутствовать.*

Тип выражения должен быть любым типом, который заменяет логический тип (целым, вещественным указателем). Если значение выражения равно 0, то оно считается ложным, в противном случае — истинным. Пример:

```
if (x>=0) y=x; else y=-x; /* Переменной y присваивается значение модуля x */
```

Если необходимо объединить несколько операторов, их объединяют в составной оператор (блок). Пример:

```
if (x>0) { y=x; z=x+y; }
else { y=-x; z=x*x+y*y; }
```

Условный оператор может быть неполным, если часть, начинающаяся с `else`, отсутствует. В этом случае, в зависимости от условия, выполняется или не выполняется оператор после `if`.

Представляют интерес вложенные условные операторы. Рассмотрим пример:

```
if (x==1) if (y==1) printf("x==1 and y==1");
else printf("x==1 and y!=1");
```

В этом примере ключевое слово `else` относится ко второму `if`, первый оператор `if` является неполным. Следует отметить, что условный оператор `if ... else ...` считается одним оператором, он может сам находиться внутри условного оператора или тела цикла. Правило, позволяющее найти соответствие между `if` и `else` в случае вложенных условных операторов, следующее: ищем первое ключевое слово `else`, оно относится к ближайшему `if`, которое предшествует ему, ко второму `else` относится `if`, которое предшествует отмеченному ранее `if`, и т. д.

Если требуется изменить описанный порядок соответствия `if` и `else` при вложенных условных операторах, с помощью фигурных скобок необходимо организовать составной оператор. В примере

```
if (x==1) { if (y==1) printf("x==1 and y==1"); }
else printf("x!=1");
```

ключевое слово `else` относится к первому `if`, второй оператор `if` является неполным, поэтому после `else` печатаем `x не равно 1`, а значение переменной `y` может быть любым.

Оператор-переключатель

В отличие от условного, оператор-переключатель можно использовать для мультиветвления фрагмента исходного кода программы: фрагмент исходного кода подразделяется на несколько частей и в зависимости от условий выполняется одна часть из нескольких (или несколько из нескольких).

Формат переключателя:

```
switch(<переключающее_выражение>)
{
    case <константное_выражение1>: [<операторы>]
```

```
    case <константное_выражение2>: [<операторы>]
       ......
    case <константное_выражениеN>: [<операторы>]
    [ default: <операторы>      ]
}
```

Здесь <переключающее_выражение> — значение целочисленное или приводящееся к целому, например тип char;

<константное_выражениеX> — целочисленное или приводящееся к целому.

При выполнении переключателя управление передается на подходящую метку вида case <конст_выр>: , значение которой совпадает со значением переключающего выражения, или на метку default (она не обязательна), если ни одна из меток не сработала. После передачи управления выполняются все операторы до конца переключателя, вне зависимости от наличия меток. Для выхода из переключателя необходимо использовать оператор break;

Рассмотрим примеры:

```
int a;
    printf("a="); scanf("%d", &a);
    switch(a)
    {
    case 1: printf("\n a == 1");
    case 2: printf("\n a == 2");
    case 3: printf("\n a == 3");
    default:
          printf("\n a > 3");
    }
```

В данном фрагменте при вводе с клавиатуры значения 2 на печать будет выводиться:

```
a == 2
a == 3
a > 3
```

Управление передается на метку case 2: , и далее выполняются все последующие операторы, несмотря на наличие меток, т. е. работает принцип выбора несколько из нескольких. Чаще всего требуется реализовать принцип выбора один из нескольких. Для этого в пере-

ключателе необходимо использовать оператор `break;` — оператор выхода из цикла или переключателя. Приведенный выше пример перепишем следующим образом:

```
int a;
    printf("a ="); scanf_s("%d", &a);
    switch(a)
    {
    case 1: printf("\n a == 1"); break;
    case 2: printf("\n a == 2"); break;
    case 3: printf("\n a == 3"); break;
    default:
          printf("\n a > 3");
    }
```

При вводе с клавиатуры значения 2 будет выводиться значение `a == 2`, затем осуществляется выход из переключателя оператором `break;`

Некоторые функции ввода-вывода

Функции для вывода в поток stdout

Ниже представлены сокращенные заголовки некоторых функций для вывода в стандартный поток stdout, связанный с экраном монитора (консольным окном).

Вывод (печать) символа:

```
int putchar(int c);
```

Параметр *c* задает код печатаемого символа, функция возвращает код символа. Пример вызова:

```
putchar('A');
```

Вывод (печать) строки:

```
int puts(const char * s);
```

где *s* — указатель на строку, строка заканчивается символом с кодом 0, при выводе строки на печать в конец добавляется «\n» (переход на новую строчку), возвращаемое значение — код последнего символа

«\n» или значение EOF в случае ошибки. EOF (End Of File) — константа, объявленная в заголовочном файле `stdio.h`, ее значение –1. Пример вызова:

```
puts("Hello world");
```

Вывод данных в соответствии с заданным форматом (форматированный вывод):

```
int printf(const char *format, ...);
```

Эта функция с переменным количеством параметров, параметр `format` – указатель на строку, данная строка выводится на консоль без изменений, за исключением шаблонов преобразования, которые могут присутствовать в строке. Как правило, количество шаблонов преобразования соответствует количеству необязательных параметров, которые присутствуют при вызове функции, порядок следования шаблонов преобразования в строке соответствует порядку необязательных параметров функции. Каждый шаблон преобразования задает формат вывода соответствующего ему параметра, при выводе строки шаблон преобразования заменяется параметром в заданном им формате. Возвращаемое значение — количество выведенных (напечатанных) символов.

Шаблон преобразования в строке всегда начинается с символа «%». Формат шаблона преобразования:

```
%[flags][width][.prec][h:l:L]type
```

`flags` может быть одним из символов:

«-» — надпись выравнивается по левому краю (по умолчанию, по правому);

«+» — числу должен предшествовать знак (перед положительным числом ставится знак «+»);

«#» — случай будет рассмотрен ниже, после разбора поля *type*;

`width` — целая константа, задает минимальную ширину поля вывода в символах;

`prec` — целая константа, задает максимальное количество выводимых символов или цифр после точки при выводе вещественных чисел;

`h:l:L` — префиксы, используемые для работы с целыми или вещественными значениями, чтобы показать, что аргумент имеет тип `short, long (double);`

type — определяет тип аргумента и формат его вывода; возможные значения — это символы:

«d», «i» — тип int, вывод целого числа в десятичной системе счисления;

«o» — тип int, вывод целого числа в восьмеричной системе счисления (без *0* впереди);

«x», «X» — тип int, вывод целого числа в шестнадцатеричной системе счисления (без 0x или 0X впереди) с применением соответственно цифр abcdef и ABCDEF;

«u» — тип unsigned int, вывод целого числа без знака в десятичной системе счисления;

«c» — тип int (к нему приводится тип char), вывод отдельного символа, символ задается своим кодом;

«s» — тип char *, вывод символов строки, строка должна заканчиваться символом с кодом 0;

«f» — тип double, вывод вещественного в форме с точкой;

«e», «E» — тип double, вывод вещественного в форме со знаком экспоненты (печатается символ *e* или *E*);

«g», «G» — тип double, использует более короткий из форматов %e/%E или %f;

«p» — тип void *, печать значения указателя;

«%» — преобразование аргументов не выполняется, печатается символ «%».

Рассмотрим назначение в качестве флага символа «#». Данный символ используется с некоторыми типами аргументов. Поставленный перед кодами *g, G, f, e* и *E*, он гарантирует наличие десятичной точки даже в случае отсутствия десятичных цифр после точки. Если поставить символ «#» перед кодами формата *x* и *X*, шестнадцатеричное число будет выведено с префиксом 0x или 0X. Если же его поставить перед форматом «o», восьмеричное число будет выведено с префиксом 0.

Следует отметить, что существует более современная и безопасная функция форматированного вывода printf_s, она имеет такой же заголовок, что и printf. Основное различие между printf_s и printf заключается в том, что printf_s проверяет строку форматирования на наличие допустимых символов форматирования, тогда как printf только проверяет, не является ли строка формата указателем со значением NULL.

Примеры использования функции:

```
#include <stdio.h>
#include <stdlib.h>
```

```
  void main()
 {
  double x=10.55, y=-12.333;
  int a1=10, a2=20;
  char Str[]="Hello world";
  char Ch='A';
  printf("x=%fy=%fa1=%da2=%d", x, y, a1, a2);
        // Печать x=10.550000y=12.333000a1=10a2=20
  printf("\nx=%8.2fy=%-8.2fa1=%+4da2=%3d", x, y, a1, a2);
         // Печать x=10.55y=-12.33  a1= +10a2= 20
  printf("\n%s %c %%\n", Str, Ch); /* Печатается Hello
world A % */
  system("pause"); /* Остановка программы до нажатия любой
клавиши */
 }
```

Функции для считывания из потока stdin

Рассмотрим некоторые функции для считывания из стандартного потока stdin, по умолчанию связанного с клавиатурой. Все данные функции останавливают выполнение программы и ждут ввода с клавиатуры, после ввода данных требуется нажать клавишу «Enter».

Чтение символа:

```
int getchar();
```

Функция возвращает код символа. Пример вызова:

```
char Ch;
Ch=getchar();
```

Чтение строки символов:

```
char * gets(char * s);
```

где `s` — указатель на массив, куда записывается строка; строка читается, пока во входном потоке не появится символ «\n» (на клавиатуре нажата клавиша «Enter»), при этом в строку записывается символ с кодом 0, возвращаемое значение — адрес строки или NULL в случае ошибки. Пример вызова:

```
char Str[64];
printf("Введите строку символов:"); gets(Str);
```

Следует отметить, что компилятор Microsoft Visual C++ 2013 выдает ошибку при использовании `gets` (более ранние компиляторы выдавали предупреждение), сообщается, что функция `gets` является небезопасной и устаревшей, и предлагается воспользоваться другой функцией — `gets_s`.

Эту ошибку можно игнорировать, если в окне свойств проекта отключить проверку безопасности (рис. 1.1).

Функция `gets_s` за счет механизма шаблона функций, который введен в язык Си++, требует обязательного указания размера буфера, куда записывается строка.

Следующий фрагмент программы демонстрирует особенности применения `gets_s`

```
char Str1[32];
printf("Stroka1: ");
gets_s(Str1); /* Здесь такая форма разрешена, размер
массива определен статически */
char *Str2 = (char *)malloc(32);
printf("Stroka2: ");
gets_s(Str2, 31); /* Так как память выделена динамиче-
ски, то требуется указать максимальную длину строки */
```

Считывание данных и запись их в переменные в соответствии с заданным форматом:

```
int scanf(const char * format, ...);
```

Функция с переменным количеством параметров внешне похожа на `printf`, обязательный параметр `format` содержит шаблоны преобразования, данные преобразуются в соответствии с шаблонами преобразования и записываются в переменные, выступающие в качестве необязательных параметров, возвращаемое значение — количество параметров, успешно преобразованных, или EOF при ошибке. В строке, определяющей формат, кроме шаблонов преобразования могут быть и другие символы, это означает, что во входном потоке обязательно должны встретиться эти же символы, при этом они будут пропускаться.

Шаблон преобразования начинается с символа «%». Формат шаблона преобразования:

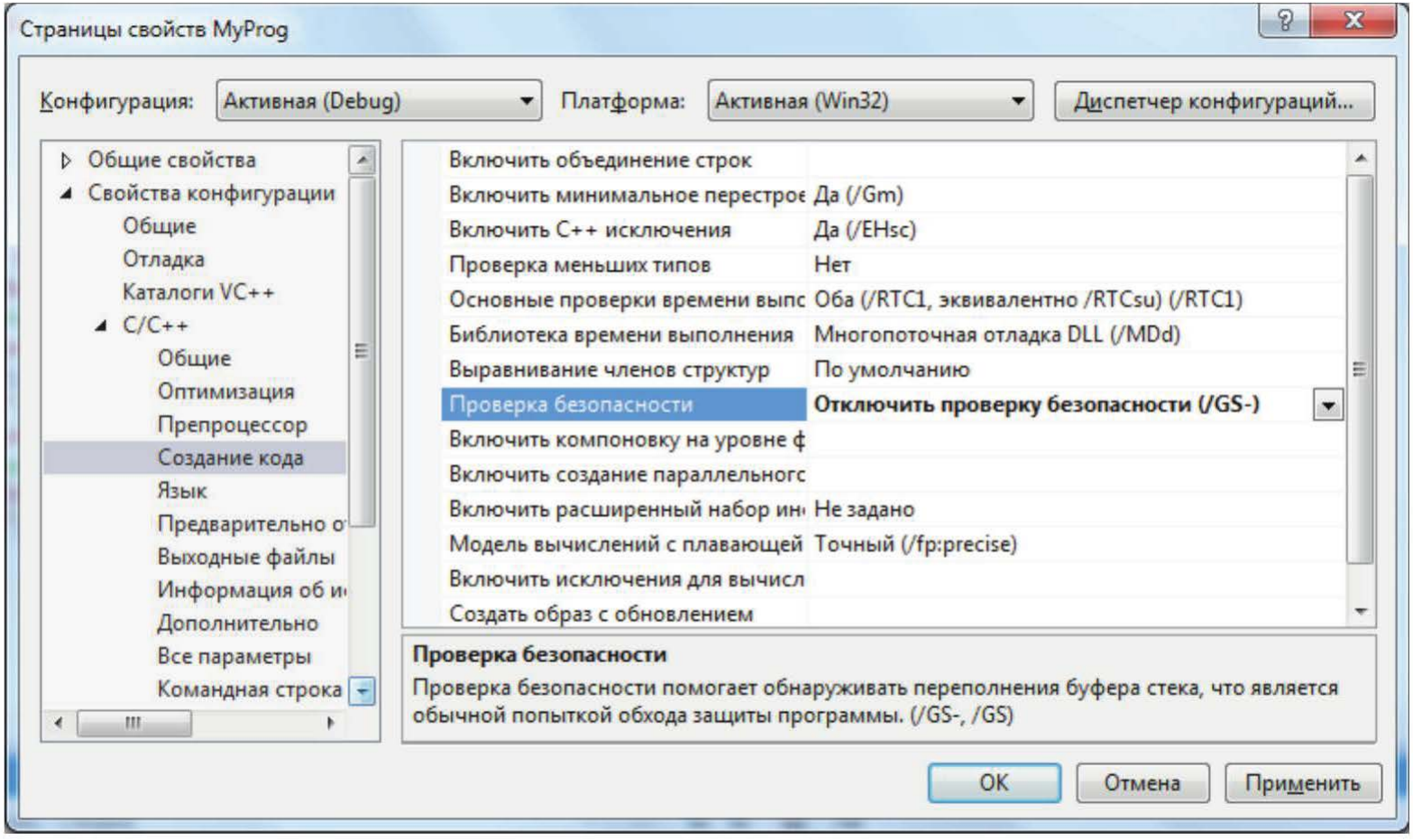


Рис. 1.1. Окно свойств проекта

`%[*][width][h:l:L]type`

«*» — необязательный символ; в этом случае поле сканируется, но в память не записывается; используется для пропуска полей;

`width` — целая константа; определяет максимальное количество символов, считанных из входного потока;

`[h:l:L]` — то же самое, что и в `printf`;

`type` — определяет тип аргумента и его формат; возможные значения — это символы:

«d» — тип `int *` (при вводе в качестве параметра передается адрес переменной — передача параметра по указателю), целая переменная в десятичной системе счисления;

«i» — тип `int *`, целое число может вводиться в десятичной, восьмеричной (0 в начале) или шестнадцатеричной (0x или 0X в начале) системах счисления;

«o» — тип `int *`, целая переменная в восьмеричной системе счисления (с нулем или без нуля впереди);

«x», «X» — тип `int *`, целая переменная в шестнадцатеричной системе счисления (с или без 0x/ 0X впереди);

«u» — тип `unsigned int *`, целая переменная в десятичной системе счисления без знака;

«c» — тип `char *`, ввод отдельного символа;

«s» — тип `char *`, ввод строки, автоматически в конец строки (массива символов) записывается символ с кодом 0, строка считывается из входного потока до любого символа разделителя, включая пробел;

«f», «e», «E», «g», «G» — тип `float *`, при вводе эти символы эквивалентны, ввод вещественного числа в форме с точкой или со знаком экспоненты;

«%» — присваивание не выполняется, в строке символ «%» обозначает, что во входном потоке должен быть символ «%».

Следует отметить, что компилятор Microsoft Visual C++ 2013 выдает ошибку при использовании `scanf` (более ранние компиляторы выдавали предупреждение), сообщается, что функция `scanf` небезопасна и устарела и предлагается воспользоваться функцией `scanf_s` с таким же заголовком. Эту ошибку можно игнорировать, если в окне свойств проекта отключить проверку безопасности (см. рис. 1.1).

В отличие от `scanf`, `scanf_s` требует, чтобы размер буфера был задан для всех входных параметров типа `c`, `s` (см. пример ниже).

Примеры использования функции:

```
#include <stdio.h>
#include <stdlib.h>
```

```
   void main()
{
   float x;
   int a;
   char Ch;
   double X;
   char Str[64];
   printf("x, a, Ch: ");
   scanf_s("%f %d %c", &x, &a, &Ch, 1);
   // Для scanf вызов будет  scanf("%f %d %c", &x, &a,
// &Ch);
   // Ввод значений через пробелы, как в строке format
   printf("x=%f a=%d Ch=%c", x, a, Ch);
   printf("\nX="); scanf_s("%lf", &X); // Ввод типа double
    printf("Str: ");
   scanf_s("%s", Str, 63); /* Ввод строки, указать размер
строки, чтобы не выйти за границу массива */
   // Для scanf вызов будет scanf("%s", Str);
   printf("propusk: "); scanf_s("%*d.%d", &a);
   /* Если в потоке вводится 10.12, то в переменной a за-
пишется 12 */
   printf("X=%f Str: %s a=%d\n", X, Str, a);
   system("pause");  /* Остановка программы до нажатия лю-
бой клавиши */
}
```

1.1.3. Задачи и порядок выполнения работы

Работа состоит из двух частей. Первая часть посвящена линейным алгоритмам, в которых для получения и хранения значений используются переменные и операции языка Си. В этой части необходимо решить некоторую математическую или физическую задачу. Первоначально необходимо разработать алгоритм решения задачи, для этого достаточно знаний школьного курса математики или физики. Особое внимание следует обратить на приоритеты операций внутри выражения и изменение приоритетов с помощью скобок [2, 4].

Вторая часть посвящена разветвляющимся алгоритмам, нужно использовать условный оператор или переключатель.

Исходные данные в задании могут быть любыми — они задаются переменными языка Си, значения переменных вводят с клавиатуры.

Студент разрабатывает программу на языке Си и выполняет ручной расчет для проверки работы программы.

Примеры выполнения работы

Пример 1

Условие задачи

Даны *x, y* (значения вводят с клавиатуры). Вычислить *a*, если

$$a = \frac{\sqrt{|x-1|} - \sqrt[3]{|y|}}{1 + \frac{x^2}{2} + \frac{y^2}{4}}.$$

Все значения вещественные.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application, в русифицированной версии — Консольное приложение Win32) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением `.cpp`, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
void main()
{
double x, y, a; // Определение переменных.
    // Ввод исходных данных.
printf_s("x="); scanf_s("%lf", &x);
printf_s("y="); scanf_s("%lf", &y);
    // Вычисление значения выражения.
a = (sqrt(fabs(x - 1)) - pow(fabs(y), 1. / 3)) / (1 +
x*x / 2 + y*y / 4);
printf_s("a=%f\n", a);
system("pause");  /* Остановка программы до нажатия лю-
бой клавиши */
}
```

Условие задачи

Определить, какая из двух точек — $M_1(x_1, y_1)$ или $M_2(x_2, y_2)$ — расположена ближе к началу координат. Вывести на экран дисплея координаты этой точки.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением `.cpp`, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
// Определить, какая из двух точек M1(x1,y1) или M2(x2,y2)
// расположена ближе к началу координат.
// Вывести на экран дисплея координаты этой точки.
void main()
{
    // Исходные данные задачи.
double x1, y1, x2, y2;
    // Ввод исходных данных.
printf_s("x1="); scanf_s("%lf", &x1);
printf_s("y1="); scanf_s("%lf", &y1);
printf_s("x2="); scanf_s("%lf", &x2);
printf_s("y2="); scanf_s("%lf", &y2);
double r1, r2; // Переменные для расчета расстояний.
    // Расчет расстояний.
r1 = sqrt(x1*x1 + y1*y1);
r2 = sqrt(x2*x2 + y2*y2);
if (r1<r2) // Точка 1 ближе к началу координат.
printf_s("x1=%f  y1=%f", x1, y1);
else
if (r2<r1) // Точка 2 ближе к началу координат.
    printf_s("x2=%f  y2=%f\n", x2, y2);
else // Точки находятся на одинаковом расстоянии.
printf_s("x1=%f y1=%f x2=%f y2=%f\n", x1, y1, x2, y2);
system("pause");  /* Остановка программы до нажатия любой клавиши */
}
```

Задания и вопросы для самоконтроля

1. Раскройте понятие оператора-выражения в языке Си.
2. Расскажите, на что влияют приоритеты операций и их ассоциативность.
3. Что такое *L*-значение (*L-value*) (леводопустимое значение) в языке Си?
4. Расскажите о назначении условного оператора.
5. Как использовать вложенные условные операторы и определить соответствие ключевых слов `else` и `if` друг другу?

Лабораторная работа № 1.2
Изучение операторов цикла в языке Си

1.2.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, реализующих циклические алгоритмы, т. е. использующих различные разновидности операторов цикла. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный циклам языка Си [2, 4];
- разработать программу на языке Си для решения заданного варианта;
- отладить программу;
- при необходимости (в зависимости от варианта задания) выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

1.2.2. Краткая характеристика объекта изучения

Операторы цикла используются для организации многократно повторяющихся вычислений. Тело цикла — оператор или составной оператор (блок) — может выполняться несколько раз; однократное вы-

полнение тела цикла называется итерацией. При переходе от итерации к итерации некоторые переменные могут изменять свои значения.

В языке Си существует три разновидности операторов цикла.

1. Цикл с предусловием имеет формат

```
while(<выражение_условие>) <оператор>
```

<выражение_условие> — значение этого выражения может быть любого типа языка Си, заменяющего логический (целый, вещественный, указатель); значение условия считается истинным, если оно отлично от нуля, при этом выполняется тело цикла; телом цикла может быть составной оператор или блок.

Фрагмент схемы алгоритма с циклом с предусловием представлен на рис. 1.2.

2. Цикл с постусловием имеет формат

```
do <оператор> while(<выражение_условие>);
```

Цикл работает аналогично предыдущему, но вначале выполняется оператор — тело цикла, затем проверяется выражение — условие (рис. 1.3).

3. Цикл `for` имеет формат

```
for([<инициализаторы>];[<выражение_условие>]; [<модификации>]) <оператор>
```

<инициализаторы> применяются для объявления и присвоения начальных значений переменным, используемым в цикле; можно записать несколько выражений, разделенных запятой.

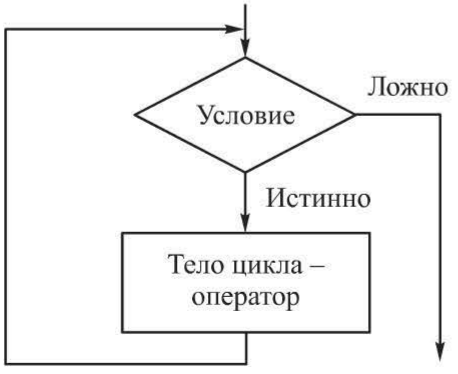


Рис. 1.2. Фрагмент схемы алгоритма с циклом с предусловием

Рис. 1.3. Фрагмент схемы алгоритма с циклом с постусловием

<выражение_условие> определяет условия продолжения цикла, задается так же, как в предыдущих двух видах циклов.

<модификации> выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла; здесь можно записать несколько выражений через запятую.

Телом цикла может быть простой оператор, составной оператор или блок (рис. 1.4).

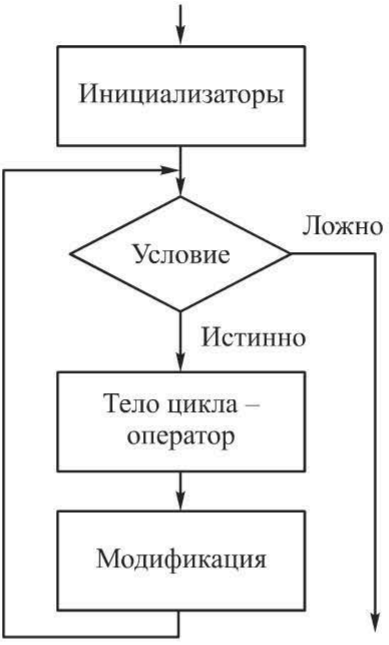


Рис. 1.4. Фрагмент схемы алгоритма с циклом `for`

Все три элемента в заголовке цикла (или некоторые из них) могут отсутствовать, но наличие «;» обязательно. Если выражение условие отсутствует, то считается, что оно всегда истинно. Пример циклов `for`:

```
for( ; ; ) { … }   // Бесконечный цикл.
for(int i=0;  i<100; i++) { … }
for(int i=0, j=20; i<j; i++, j--) { … }
for(int i=0; i<20; i+=2) { … }
```

1.2.3. Задачи и порядок выполнения работы

При разработке программы на языке Си студент использует один из видов его циклов. Защищая работу, следует обосновать выбор того или иного вида цикла. Если для выполнения задания требуются не-

которые исходные данные, то они могут быть любыми и вводятся с клавиатуры. В зависимости от варианта студент выполняет ручной расчет для проверки работы программы для задачи небольшой размерности или выводит известное точное значение, если используется разложение в ряд. Результаты работы программы, ручного расчета или известного точного значения выражения нужно представить преподавателю.

Примеры выполнения работы

Пример 1

Условие задачи

Вычислить сумму ряда

$S = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k}$ с точностью $\xi = 10^{-2}, 10^{-4}$. Точное значение составляет ln 2 (используется для проверки).

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением `.cpp`, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
void main()
{
    double e = 1e-4;
    double Sum = 0; // Рассчитываемая сумма.
    double si; // Очередное слагаемое в сумме.
    int k = 0; // Индекс.
    int sign = 1; /* Знак слагаемого, значение или 1
или -1 (или -1 в степени)*/
    do
    {
    k++; // Номер слагаемого (итерации).
    si = 1. / k; // Модуль слагаемого.
```

```
        Sum += sign*si; // Расчет суммы.
        sign *= -1; // Изменение знака слагаемого.
        } while (si >= e); /* Пока модуль слагаемого si не
меньше точности e, цикл продолжается */
        printf_s("Sum=%f  k=%d  ln(2)=%f\n", Sum, k, log(2.));
        system("pause");  /* Остановка программы до нажатия
любой клавиши */
        }
```

В результате выполнения программы в консольное окно будет выведено:

```
Sum=0.693197   k=10001  ln(2)=0.693147
```

Пример 2

Условие задачи

Сравнить скорость сходимости (количество слагаемых для достижения заданной точности $\xi = 10^{-2},..., 10^{-6}$) следующего разложения числа π:

$$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...\right).$$

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES     /* Необходимо использовать
константу M_PI */
#include <math.h>
int main(int argc, char* argv[])
{
double sum = 0, e, sl; /* Объявление вещественных пере-
менных */
int k = 1, znak = 1; /* Объявление целых переменных
(знаменатель и знак) */
printf_s("E="); scanf_s("%lf", &e); // Ввод точности.
```

```
   int n = 0; // Счетчик количества повторений цикла.
   do
   {
       sl = 4. / k; // Вычисление очередного слагаемого.
       sum += sl*znak; // Расчет суммы.
       k += 2; // Увеличение значения знаменателя дроби.
       znak *= -1; // Изменение знака слагаемого.
       n++; // Увеличение счетчика повторений цикла.
   } while (sl >= e); /* Пока слагаемое sl не меньше точности e, цикл продолжается */
   printf_s("Sum=%.10f  n=%d   PI=%.10f\n", sum, n, M_PI); // Вывод результатов.
   system("pause");  /* Остановка программы до нажатия любой клавиши */
   return 0;
   }
```

Задания и вопросы для самоконтроля

1. Перечислите разновидности операторов циклов в языке Си.
2. Что такое итерация цикла?
3. Что такое условие выполнения цикла?
4. Раскройте особенности цикла с предусловием.
5. Раскройте особенности цикла с постусловием.
6. Раскройте особенности цикла `for`.

Лабораторная работа № 1.3
Изучение массивов в языке Си

1.3.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, использующих массивы как одномерные, так и многомерные. Для достижения цели необходимо выполнить следующие задачи:

• изучить учебный материал, посвященный массивам в языке Си [2, 4];

• разработать программу на языке Си для решения заданных вариантов;

• отладить программу;

• выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

1.3.2. Краткая характеристика объекта изучения

Приведем следующее определение массива: **массив** — это производный тип, представляющий собой множество элементов, имеющих один и тот же тип и следующих в памяти один за другим.

Массивы бывают одномерными и многомерными.

Одномерные массивы

Формат определения одномерного массива:

```
<type> <имя_массива>[<конст_выражение>];
```

Примеры:

```
int A[100]; /* Объявление массива из 100 элементов типа int */
char C1[20], C2[150]; /* Объявление двух массивов элементов типа char из 20 и 150 элементов */
```

Описание массива может не содержать количество элементов.

```
extern float y[]; // Описание.
...;
float y[20]; // Определение.
```

Обращение к элементам массива осуществляется с помощью специальной операции [] — обращение по индексу:

```
int m[10];
m[0]=1; /* Элементу массива с индексом 0 присваивается значение 1 */
m[1]=2; /* Элементу массива с индексом 1 присваивается значение 2 */
… ;
```

```
m[9]=10; /* Элементу массива с индексом 9 присваивается
значение 10 */
```

Индексация всегда начинается с 0, последний элемент имеет индекс $n-1$.

Элемент массива может участвовать в любом выражении, где допустимо использование переменной соответствующего типа. Пример:

```
m[5]=m[3]+m[4]*100;
```

Инициализация массива

Массив может иметь инициализацию. Если массив имеет инициализацию, то размерность можно не указывать, она определяется по умолчанию.

```
int a1[4]={ 2, 5, 9, 10 }; /* Массив из четырех элементов */
int a2[7]={ 3, 4, 5 }; /* Массив из семи элементов, проинициализированы первые три */
char c1[]={ 'A', 'g', '1', 'y', 'I' }; /* Массив из пяти элементов, размерность определяется по умолчанию */
```

Если массив объявлен за пределами блоков, по умолчанию его элементы инициализируются 0.

Массивы и указатели

Имя массива, используемое внутри выражения без `[]`, является указателем на первый элемент массива (элемент с индексом 0). Данный указатель является константным, т. е. его значение нельзя изменять.

Для некоторого массива, например

```
int A[10];
```

выполняется равенство `A==&A[0]`.

Доступ к элементу массива возможен через индексное выражение или через операцию обращения по адресу:

```
for (int i=0; i<10; i++) A[i]=2*i;
for (int  i=0; i<10; i++) *(A+i)=2*i; /* Наличие скобок обязательно, так как приоритет операции * выше, чем операции + */
```

Кроме того, имя массива без скобок является указателем, тип которого соответствует указателю на тип элемента массива. Допустим следующий порядок работы с массивом:

```
int a[10];
int *p;
p=a;
for(int i=0; i<10; i++) p[i]=i*i;
или
for(int i=0; i<10; i++) *(p+i)=i*i;
```

Строки

Строка — это символьный массив, заканчивающийся символом с кодом 0 (символ NULL), служебный символ, являющийся признаком конца строки.

Все стандартные функции, работающие со строками, используют это свойство для определения конца строки. Примеры:

```
char str[]="ABCD"; /* Это строка, выделяется память под пять символов (последний 0) */
char c[]={'A', 'B', 'C', 'D' }; /* Это просто массив из четырех символов */
char *s="Пример строки"; /* Допустимо такое определение строк */
```

Зная признак конца строки, можно, например, вычислить длину строки:

```
int len=0;// Длина строки после цикла будет в этой переменной
for(; s[len]; len++) ;
```

Динамические массивы

Память для массива может выделяться и уничтожаться динамически с помощью специальных функций. Например, для выделения памяти могут использоваться функции: `malloc`, `calloc`, `realloc`, а для освобождения памяти — функция `free` (в языке Си++ существуют специальные операторы `new` и `delete`). Это особенно удобно в том случае, когда количество элементов заранее неизвестно, а становится известным только во время выполнения программы, например, количество элементов можно вычислять, вводить с клавиатуры и т. д.

Функция malloc имеет заголовок, определенный в заголовочном файле stdlib.h:

```
void * malloc(unsigned int _Size);
```

Данная форма заголовка является упрощенной; функция имеет один параметр, определяющий размер выделяемой динамически области памяти в байтах, возвращает указатель на выделяемую область памяти; тип указателя void *, что обеспечивает его неявное приведение к указателям на любые стандартные типы.

Возвращает выделенную память обратно операционной системе функция с упрощенным заголовком (заголовочный файл stdlib.h):

```
void free(void * _Memory);
```

Функция имеет один параметр — указатель на блок освобождаемой памяти.

Ниже представлен пример выделения памяти под массив, размерность которого вводится с клавиатуры. Далее массив заполняется псевдослучайными числами, выводится на печать, в конце работы программы выделенная память освобождается. Пример реализован на языке Си (файл с расширением .c). Если реализовывать его на языке Си++ (файл с расширением .cpp), то при использовании malloc необходимо применять операцию явного преобразования типа. Пример:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int *p; /* Указатель, может быть использован для выделения памяти под массив динамически */
    int n; /* Заранее неизвестное количество элементов массива */
    int i; // Индексная переменная.
    printf_s("n="); scanf_s("%d", &n);/* Ввод с клавиатуры количества элементов массива */
    p = malloc(n*sizeof(int));/*Выделение памяти под массив динамически в языке Си */
/* p=(int *) malloc(n*sizeof(int)); в языке Си++ требуется преобразование типа!!!!!! */
```

```
        /* Заполнение массива псевдослучайными числами и печать его */
        for (i = 0; i<n; i++)
        {
        p[i] = rand() % 100; /* Получение псевдослучайного целого числа в интервале 0…99 */
        printf_s("%d  ", p[i]);
        }
        free(p); /* Возвращение выделенной памяти операционной системе, когда в ней нет необходимости */
        system("pause");
    }
```

Многомерные массивы

Многомерный массив — это массив массивов. Пример определения:

```
int A[2][3];
```

Данное определение можно раскрыть следующим образом: *A* — массив из двух элементов; элементами массива являются массивы из трех элементов типа `int`.

Элементы следуют в памяти в следующем порядке:

```
A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2]
```

Обращаться к элементам массива можно с помощью индексного выражения или с помощью указателей:

```
A[0][1]=5;
A[i][j]=i+j; // или
*(*(A+i)+j)=i+j;
```

Инициализация многомерных массивов

Инициализацию многомерных массивов можно проводить по аналогии с одномерными массивами. Например:

```
int A[2][3][2]={ 1, 2, 3, 4, 5};
```

Инициализируются первые пять элементов по их расположению в оперативной памяти, т. е. элементы с индексами: 000, 001, 010, 011, 020.

Но удобнее использовать вложенные фигурные скобки, при этом самую левую размерность можно не указывать, она определяется по умолчанию:

```
int B[][5]={{ 1, 2, 3} , { 6, 7} , { 1, 2, 3, 4, 5 }};
```

В примере создается массив размерностью 3×5 (аналог матрицы 3×5, при этом в первой строке инициализируются три элемента, во второй — два, в третьей — все пять элементов).

Динамические многомерные массивы

По аналогии с одномерными массивами многомерные массивы можно создавать динамически. Рассмотрим пример создания двухмерного массива $n \times m$ (матрица $n \times m$), где значения n и m заранее неизвестны — вводятся с клавиатуры. В примере создается матрица $n \times m$, заполняется псевдослучайными числами, выводится на печать, в конце программы память освобождается:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int **p; /* Указатель на указатель может быть использован для выделения памяти под двухмерный массив динамически */
    int n, m; // Заранее неизвестные размерности массива.
    int i, j; // Индексные переменные.
  printf_s("n="); scanf_s("%d", &n); /* Ввод с клавиатуры количества строк матрицы */
  printf_s("m="); scanf_s("%d", &m); /× Ввод с клавиатуры количества столбцов матрицы */
  p = malloc(n*sizeof(int *)); /* Выделение памяти под массив указателей в языке Си */
  /* p=(int **)malloc(n*sizeof(int *)); В языке Си++ требуется преобразование типа */
  /* Выделение памяти для каждого указателя в массиве указателей */
  for (i = 0; i<n; i++) p[i] = malloc(m*sizeof(int)); /* В языке Си */
  // for(i=0; i<n; i++) p[i]=(int *)malloc(m*sizeof(int)); // В Си++ требуется преобразование типа
```

```
        /* Заполнение массива (матрицы) псевдослучайными чис-
лами и печать его */
        for (i = 0; i<n; i++)
        {
        for (j = 0; j<m; j++)
        {
        p[i][j] = rand() % 100; /* Получение псевдослучай-
ного целого числа */
        printf_s("%d  ", p[i][j]);
        }
        printf_s("\n"); /* После печати строки матрицы осу-
ществляется переход на новую строчку */
        }
        /* Освобождение памяти, порядок освобождения обратен
порядку выделения */
        for (i = 0; i<n; i++) free(p[i]); /* Освобождение
памяти  для строк матрицы */
        free(p);/* Освобождение памяти для массива указателей
        system("pause"); */
    }
```

1.3.3. Задачи и порядок выполнения работы

Лабораторная работа состоит из двух частей: 1) одномерные массивы; 2) многомерные массивы. Студенты выполняют обе части.

В первой части исходные массивы следует формировать с помощью генератора псевдослучайных чисел, количество элементов массива вводить с клавиатуры. Сформированный массив (массивы) необходимо вывести на экран, а также вывести результат работы программы. Используя массив небольшой размерности, провести контрольный расчет вручную. *При изменении массивов (сортировка, запись в обратном порядке и др.) новый массив не создавать.*

Во второй части работы используются двухмерные массивы (матрицы). Матрицы также формируют с помощью генератора псевдослучайных чисел, размерности матриц вводят с клавиатуры. Исходные данные и результаты работы выводят на печать. Для исходных данных небольшой размерности проводят ручной расчет для контроля результатов. *При изменении матриц (сортировка строки, запись строки в обратном порядке, транспонирование матрицы и др.) новые массивы (матрицы) не создавать.*

Пример 1

Условие задачи

Задан одномерный массив вещественных чисел. Написать программу, которая «переворачивает» массив, т. е. меняет местами его первый и последний элементы, второй и предпоследний и т. д.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    double *pMas; // Указатель на массив.
    int n; // Заранее неизвестная размерность массива.
    // Ввод размерности массива с клавиатуры.
    printf_s("n=");
    scanf_s("%d", &n);
    // Выделение памяти динамически.
    pMas = (double *)malloc(n*sizeof(double));
    /* Заполнение массива псевдослучайными числами в интервале [0, 100) и печать массива */
    for (int i = 0; i<n; i++)
    {
    pMas[i] = rand() % 100;
    printf("%f  ", pMas[i]);
    }
    // Переворачивание массива
    for (int i = 0; i<n / 2; i++)
    {
    /* Замена двух элементов местами с использованием буферной переменной */
    double buf = pMas[i];
    pMas[i] = pMas[n - 1 - i];
```

```
        pMas[n - 1 - i] = buf;
        }
        // Печать полученного массива.
        printf("\n");
        for (int i = 0; i<n; i++)
        printf("%f   ", pMas[i]);
        // Освобождение памяти.
        free(pMas);
        system("pause");/* Остановка программы, ожидание на-
жатия любой клавиши */
    }
```

Пример 2

Условие задачи

Матрицу **M**(m, n) заполнить натуральными числами от 1 до $m \times n$ по спирали, начинающейся в левом верхнем углу и закрученной по часовой стрелке.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
    #include <stdio.h>
    #include <stdlib.h>
    int main(int argc, char* argv[])
    {
    int **pMatr;
    int n, m;
    printf_s("n="); scanf_s("%d", &n);
    printf_s("m="); scanf_s("%d", &m);
    pMatr = (int **)malloc(n*sizeof(int *));
    for (int i = 0; i<n; i++)
        pMatr[i] = (int *)malloc(m*sizeof(int));
        int i = 0, j = 0, // Текущие координаты.
        minx = 0, maxx = m - 1, miny = 0, maxy = n - 1;
// Границы поля движения.
    int step = 1; /* Шаг перехода (может быть 1 или -1) */
    bool flag = true; // Направление движения.
    // true — горизонтальное, false — вертикальное.
```

```
for (int k = 0; k<n*m; k++)
{
pMatr[i][j] = k + 1;
if (flag)
{
if (step>0) if (j<maxx) j++;
    else {
           miny++;
           flag = false;
           i++;
                  }
                  else
                  if (j>minx) j--;
           else {
                  maxy--;
                  flag = false;
                  i--;
                  }
           }
           else // Движение было вертикальное.
           if (step>0) if (i<maxy) i++;
           else {
                  maxx--;
                  flag = true;
                  step = -1;
                  j--;
                  }
                  else
                  if (i>miny) i--;
                  else {
                         minx++;
                         flag = true;
                         step = 1;
                         j++;
                         }
    }
    for (int i = 0; i<n; i++)
    {
```

```
            for (int j = 0; j<m; j++)
                  printf_s("%4d", pMatr[i][j]);
            printf_s ("\n");
      }
      // Освобождение памяти.
      for (int i = 0; i<n; i++) free(pMatr[i]);
      free(pMatr);
      system("pause");/* Остановка программы, ожидание на-
жатия любой клавиши */
      return 0;
   }
```

Задания и вопросы для самоконтроля

1. Раскройте связь указателей и массивов в языке Си.
2. Что такое указатель на указатель?
3. Перечислите операции с указателями в языке Си.
4. Раскройте особенности создания многомерных массивов динамически.

Лабораторная работа № 1.4
Изучение структурных типов языка Си

1.4.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, использующих структурные типы данных. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный структурам в языке Си [2, 4];
- разработать программу на языке Си для решения заданных вариантов;
- отладить программу;
- выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

1.4.2. Краткая характеристика объекта изучения

Структура — это производный тип языка Си, включающий в себя множество элементов; элементы следуют в памяти один за другим и могут быть разнотипными.

Формат определения структуры следующий:

```
struct [<имя_структурного_типа>]
{
    ... // Объявления полей структуры.
} [<имя_переменной_структурного_типа>];
Пример:
struct A
{
            int x; // Поле структуры.
            float y; // Поле структуры.
};
```

После объявления структуры имя структуры является именем нового типа. Аналогично переменным стандартных типов могут объявляться переменные структурного типа, при этом каждая переменная структурного типа имеет свои копии полей структуры в оперативной памяти. Также можно объявлять указатели на структуры, массивы структур; полями структуры могут быть массивы или указатели.

```
A a1, a2; // Объявление переменных структурного типа.
```

Примечание. *Такой формат определения переменных структурного типа разрешен в языке Си++ , в исходном Си стандарта ANSI требуется при подобном объявлении переменных структурного типа дополнительно указывать ключевое слово* `struct`*:*

```
struct A a1, a2; /* Объявление переменных структурного
типа в языке Си стандарта ANSI */
```

Разрешено объявлять переменные структурного типа одновременно с объявлением структуры, и если далее переменные этого типа не будут создаваться, имя структуры может отсутствовать.

Пример

```
struct A
{
    int x; // Поле структуры.
```

```
    float y; // Поле структуры.
} a1, a2; /* Объявление переменных одновременно со струк-
турой */
Или
struct  /* Имя структуры отсутствует, далее переменные
этого типа не могут создаваться */
{
    int x; // Поле структуры.
    float y; // Поле структуры.
} a1, a2; /* Объявление переменных одновременно со струк-
турой */
```

Обращение к полям структуры проводят с помощью операции «.» — обращение к полю структуры через имя переменной или операции «->» — обращение к полю структуры через указатель:

```
a1.x=10;     a1.y=1.5;
A *pA=&a1; // Указатель на структуру.
pA->x=11; // или (*pA).x=11;
```

Инициализация структуры. Переменная структурного типа может быть определена с инициализацией полей. Инициализация структуры похожа на инициализацию массива. Например:

```
struct BOOK // Структура описывает некоторую книгу.
{
char * author; // Имя автора книги.
char *title;    // Заголовок книги.
char *firm;   // Название издательства.
int year, page; // Год издания и количество страниц.
};
BOOK book1={ " Б. Керниган, Д. Ритчи",
 "Язык программирования С", " Москва: Издательский дом\
«Вильямс»", 2009, 304 };
```

1.4.3. Задачи и порядок выполнения работы

В заданиях необходимо создать массив структур. Количество элементов массива заранее неизвестно и вводится с клавиатуры, т. е. массив создается динамически. Значения полей структур также вводятся

с клавиатуры. Особое внимание нужно обратить на поля структур, которые сами являются структурами. После ввода массива структур выводят все введенные данные на экран, вычисляют требуемые параметры, демонстрируют работу программы преподавателю.

Пример выполнения работы

Условие задачи

Определите структуру «Житель». Поля: Ф.И.О., переменная структурного типа «Адрес» (поля: улица; номер дома; номер квартиры); пол; возраст.

В программе задайте несколько жителей — фрагмент базы данных ЖЭК (массив переменных структурного типа) и определите, сколько детей до 7 лет проживают на заданной улице.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Adres // Структура «адрес».
{
    char street[64]; // Название улицы.
    int num_H; // Номер дома.
    int num_F; // Номер квартиры.
};
struct Gitel  // Структура житель
{
    char fio[128]; // Ф.И.О.
    Adres adr; // Адрес.
    char pol; // Пол. символы 'м' и 'ж'.
    int vozr;  // Возраст.
};
int main(int argc, char* argv[])
```

```
{
    Gitel *pG; // Указатель на массив структур.
    int n; // Количество элементов массива.
    printf_s("n=");
    scanf_s("%d", &n); /* Ввод количества элементов мас-
сива */
    pG = (Gitel *)malloc(n*sizeof(Gitel)); /* Выделение
памяти под массив структур */
    for (int i = 0; i<n; i++) /* Цикл ввода данных с
клавиатуры */
    {
        printf_s("Gitel N=%d", i + 1);
        printf_s("\nFIO: ");
        _flushall(); // Сброс всех буферов ввода-вывода.
        gets_s(pG[i].fio, 127);
        printf_s("Address: ");
        printf_s("\nStreet: ");
        fflush(stdin);
        gets_s(pG[i].adr.street, 63);
        printf_s("House: ");
        scanf_s("%d", &pG[i].adr.num_H);
        printf_s("Flat: ");
        scanf_s("%d", &pG[i].adr.num_F);
        printf_s("pol: ");
        fflush(stdin);
        scanf_s("%c", &pG[i].pol, 1);
        printf_s("Vozrast: ");
        scanf_s("%d", &pG[i].vozr);
    }
    for (int i = 0; i<n; i++) /* Цикл печати введенных
данных */
        printf_s("\n%s %s %d %d %c %d",
        pG[i].fio, pG[i].adr.street,
        pG[i].adr.num_H, pG[i].adr.num_F,
        pG[i].pol, pG[i].vozr);
    char StreetName[64];
    printf_s("\nStreet Name: ");
    fflush(stdin); // Сброс всех буферов ввода-вывода
  gets_s(StreetName, 63); /* Ввод названия улицы, на ко-
торой осуществляется поиск */
    int k = 0;
```

```
        for (int i = 0; i<n; i++) // Просмотр всех жителей.
              if (strcmp(StreetName, pG[i].adr.street) == 0)
/* Нахождение улицы */
              if (pG[i].vozr<7) k++; /* Нахождение ребенка
младше 7 лет */
        if (k) printf_s("\nk=%d\n", k); /* Вывод количества
найденных детей */
        else printf_s("Not found\n"); // Таких детей нет.
        free(pG); // Освобождение памяти.
        system("pause"); /* Остановка программы, ожидание
нажатия любой клавиши */
        return 0;
    }
```

Задания для самоконтроля

1. Раскройте назначение структуры в языке Си.
2. Напишите исходный код программы для создания массива структур динамически.
3. Перечислите операции для обращения к полям структур.

Лабораторная работа № 1.5
Изучение функций языка Си

1.5.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си при использовании функциональной декомпозиции предметной области и функций языка Си. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный функциям языка Си [2, 4];
- разработать программу на языке Си для решения заданного варианта;
- отладить программу;

• выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

1.5.2. Краткая характеристика объекта изучения

Функция — это именованная совокупность объявлений и операторов, предназначенная для выполнения некоторой отдельной подзадачи.

Использование функции позволяет выполнять так называемую функциональную декомпозицию задачи (предметной области). Большую сложную задачу разбивают на относительно простые подзадачи, и для решения каждой подзадачи разрабатывают функцию. Функции можно отлаживать отдельно друг от друга.

Функция в языке Си — основное понятие. Любая выполняемая программа содержит, по крайней мере, одну функцию — так называемую главную функцию, с именем main (в Windows-приложении эта функция носит имя WinMain).

Определение, описание и вызов функции

Следует различать определение функции, описание функции (заголовок) и вызов функции.

Формат определения функции:

```
[<модификаторы>] <тип_возвращаемого_значения>
<имя_функции>([<спецификация_формальных_параметров>])
{ <тело_функции> }
```

Спецификация формальных параметров в круглых скобках определяет параметры функции, один параметр отделяется от другого запятой. Спецификация формальных параметров имеет следующий формат:

```
<тип1> [<имя1>][=<умалчиваемое_значение>], <тип2> [<имя2>]
[=<умалчиваемое_значение>],...
```

Формальные параметры функции могут отсутствовать, в этом случае после имени функции идут пустые скобки (или в скобках может быть ключевое слово `void`). Имя формального параметра может отсутствовать, если этот параметр пока не используется в теле функции, но зарезервирован на будущее.

Тело функции является блоком, в нем локализованы формальные параметры заголовка. Важным оператором тела функции является оператор `return`; наличие данного оператора обязательно, если тип возвращаемого значения отличен от `void`. Если функция возвращает некоторое значение, то ее можно вызывать внутри оператора-выражения в любом месте, где разрешено использовать значение такого типа.

В качестве модификаторов функции может присутствовать модификатор класса памяти `extern` или `static`. По умолчанию, у функции класс памяти `extern`. Если функция определена с ключевым словом `static`, ее можно использовать только в том файле, где она определена.

Функция может быть определена только один раз. Определение функции может находиться в отдельном файле, но чтобы ее использовать в другом файле (или в том же файле, но выше места определения) необходимо до вызова функции включить описание функции. Описание функции, по сути, является заголовком функции без тела, оно имеет формат:

```
[<модификаторы>] <тип_возвращаемого_значения> <имя_функции> (<спецификация_формальных_параметров>);
```

Примечание. *Стандартные заголовочные файлы* `stdio.h`, `stdlib.h`, `math.h` *и др. содержат описания или заголовки стандартных функций, поэтому их необходимо подключать, если требуется вызывать стандартные функции, описанные в заголовочных файлах.*

Совокупность формальных параметров определяет сигнатуру функции. В описании имена параметров можно не использовать, важны количество и типы параметров. Также в описании не должны присутствовать умалчиваемые значения параметров, если они есть в определении функции.

Пример простой функции, которая возвращает сумму двух своих параметров:

```
double summa(double x, double y)
{
return x+y;
}
```

Описание или заголовок этой функции имеет вид

```
double summa(double, double);
```

Вызов функции имеет следующий формат (вызов функции часто выполняется внутри оператора-выражения):

```
<имя_функции>(<список_факт_параметров>);
```

При вызове функции вместо формальных параметров подставляют фактические (переменные, константы, указатели и др.) и выполняют операторы тела функции со значениями фактических параметров.

Следует отметить, что функцию можно вызывать «как функцию», когда используется возвращаемое значение и функцию вызывают внутри выражения:

```
S=summa(a, b); /* Переменной S присваивается значение,
возвращаемое функцией */
```

Функцию можно вызывать «как процедуру», когда возвращаемое значение не используется (если оно вообще не нужно, можно объявить тип `void`):

```
printf("x=%f", x); /* Функция printf возвращает значение
типа int, но оно здесь не используется */
```

Тип возвращаемого значения и параметры функции main

Функция `main` может иметь тип возвращаемого значения `int` (операционной системе задается код возврата программы, который можно использовать в пакетных командных файлах или при запуске одного процесса из другого). В предыдущих примерах функция `main` не имела параметров или они не использовались, но функция, кроме случая, когда список ее параметров пуст, может иметь три или два параметра, третий может отсутствовать, например:

```
void main(int argc, char *argv[], char *envp[])
{ …. }
```

Первые два параметра передают аргументы через командную строку (количество аргументов и массив строк).

Третий параметр — `envp` — задает контекст (среду выполнения), т. е. значения переменных среды окружения. Переменные находятся в массиве строк, каждая переменная среды окружения — это отдельная строка.

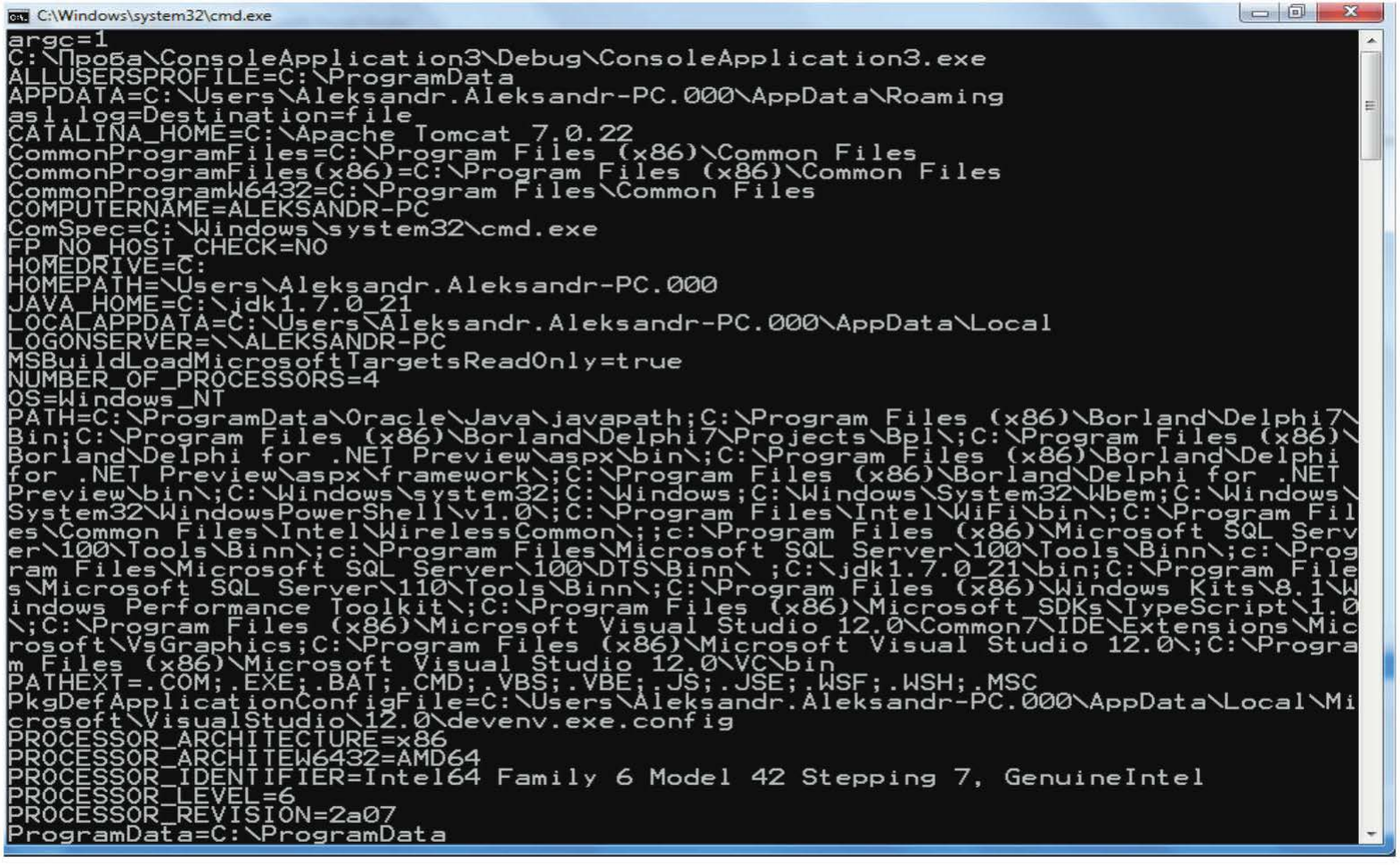


Рис. 1.5. Пример вывода параметров командной строки и переменных среды окружения

Ниже в примере выводятся на печать параметры командной строки (в любом случае существует хотя бы один параметр — это имя исполняемого файла) и возможные переменные среды окружения при запуске программы на некотором компьютере под управлением ОС Windows 7 (в разных средах они могут быть разными); результаты вывода представлены на рис. 1.5.

```
#include <stdio.h>
#include <locale.h>

void main(int argc, char *argv[], char *envp[])
{
    int i;
    setlocale(LC_ALL, "rus"); /* Возможность вывода рус-
ских символов в кодировке Windows-1251 на консоль */
    printf("argc=%d", argc); /* Печать количества пара-
метров командной строки */
    // Печать параметров командной строки.
    for (i = 0; i<argc; i++) printf("\n%s", argv[i]);
    // Печать переменных сред окружения.
    for (i = 0; envp[i]; i++) printf("\n%s", envp[i]);
}
```

1.5.3. Задачи и порядок выполнения работы

Обратите внимание, что в каждом задании указано, какую подзадачу должна решать разрабатываемая функция. Исходные данные (*n, m* и др.) программа должна получать *через аргументы командной строки* (*параметры функции main*). Если в задании необходимо в качестве исходных данных использовать массивы, их нужно сформировать с помощью генератора псевдослучайных чисел (*размерности массивов программа получает через аргументы командной строки*). На экран выводят исходные данные и полученные результаты. На примере с небольшой размерностью следует провести контрольный расчет для проверки работоспособности алгоритма.

Пример выполнения работы

Условие задачи

Решить задачу, используя функцию.

Даны два натуральных числа a и b — числитель и знаменатель дроби. Сократите дробь, разделив числа на их наибольший общий делитель (НОД). Функция должна находить НОД двух чисел по алгоритму Евклида.

Алгоритм Евклида:

1. Вычислим r — остаток от деления числа a на b ($a > b$), $a = bq + r$, $0 \le r < b$.
2. Если $r = 0$, то b есть искомое число (НОД).
3. Если $r! = 0$, заменим пару чисел (a, b) парой (b, r) и перейдем к шагу 1.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
unsigned  Evklid(int a, int b)
{
    // Находим НОД для a и b.
    int a1, b1;
    if (a>b)
    {
          a1 = a; b1 = b;
    }
    else
    {
          a1 = b; b1 = a;
    }
    int o;
    do
    {
          o = a1 % b1;
          if (o == 0) return b1;
          a1 = b1;
          b1 = o;
    } while (1);
}
```

```
int main(int argc, char *argv[])
{
    int a, b,    /* Исходные данные — числитель и знаменатель */
        nod;
    setlocale(LC_ALL, "rus"); /* Возможность вывода русских символов в кодировке Windows-1251 на консоль. */
    if (argc < 3) // Ошибка.
    {
    printf_s("Ошибка; параметров в командной строке не\
хватает для задания исходных данных. Для завершения\
нажмите любую клавишу \n");
    system("pause");
    return 1;
    }
    // Чтение значения из командной строки.
    if (sscanf_s(argv[1], "%d", &a) < 1) /* Чтение переменной a */
    {
    printf_s("Ошибка, неверный формат первого входного\
параметра. Для завершения нажмите любую клавишу \n");
    system("pause");
    return 1;
    }
    if (sscanf_s(argv[2], "%d", &b) < 1) /* Чтение переменной b */
    {
    printf_s("Ошибка, неверный формат первого входного\
параметра. Для завершения нажмите любую клавишу \n");
    system("pause");
    return 1;
    }
    nod = Evklid(a, b); /* Получение наибольшего общего делителя (НОД) */
    printf("a=%d, b=%d, nod=%d. Сокращенная дробь: %d/%d.\
Для завершения нажмите любую клавишу \n", a, b, nod, a/nod, b/nod);
```

```
    system("pause"); /* Остановка программы до нажатия
любой клавиши */
    return 0;
}
```

После компиляции и сборки программы ее необходимо запустить в командной строке. Если имя исполняемого файла `MyProg.exe`, необходимо в командной строке ввести команду `MyProg 66 100`

После имени исполняемого файла обязательно идут через пробелы два параметра, являющиеся целыми значениями — числителем и знаменателем дроби.

Результаты работы программы представлены на рис. 1.6.

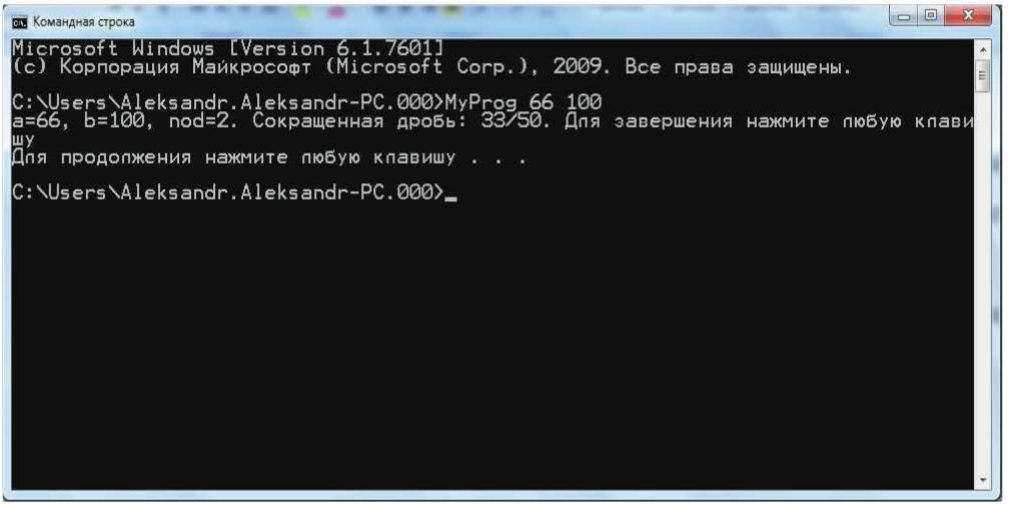


Рис. 1.6. Результаты работы программы

Задания для самоконтроля

1. Расскажите о назначении функций в языке Си.
2. Раскройте особенности понятий определения, описания и вызова функции, их различия.
3. Раскройте особенности понятий вызова функции как «функции» и вызова функции как «процедуры» в языке Си.
4. Расскажите о назначении параметра функции main.

Лабораторная работа № 1.6
Изучение динамических структур данных. Списки

1.6.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, использующих линейные списки, при работе с данными, имеющими динамическую структуру. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный линейным спискам [5];
- разработать программу на языке Си для решения заданного варианта;
- отладить программу.

Выполнив работу, нужно подготовить отчет.

1.6.2. Краткая характеристика объекта изучения

Статические структуры данных (обычные переменные, массивы, структуры, объединения и т. п.) имеют раз и навсегда заданные для них объем памяти и, соответственно, диапазон возможных значений.

Множество задач требует хранения данных с более сложной структурой, в процессе вычисления должны изменяться не только значения переменных, но и их структура и, соответственно, размер выделяемой памяти. Такие данные называют **данными с динамической структурой.** Иногда их называют данными с рекурсивной структурой [5].

Генеалогическое древо определяется как имя человека + два древа его родителей. Такое определение ведет к бесконечной структуре, реальные деревья всегда конечны.

Особенность рекурсивных структур — способность изменять размер. Для этого используют метод динамического распределения памяти, т. е. память под отдельные компоненты выделяется в тот момент, когда появляется необходимость. Это можно реализовать с помощью указателей.

Простейшая динамическая структура данных — линейный односвязный список. Каждый элемент списка содержит информационное поле и указатель на следующий элемент списка, в последнем элементе

этот указатель обычно равен 0 (NULL). Для работы со списком достаточно хранить в оперативной памяти в отдельной переменной указатель на первый элемент списка. Схема линейного односвязного списка представлена на рис. 1.7.

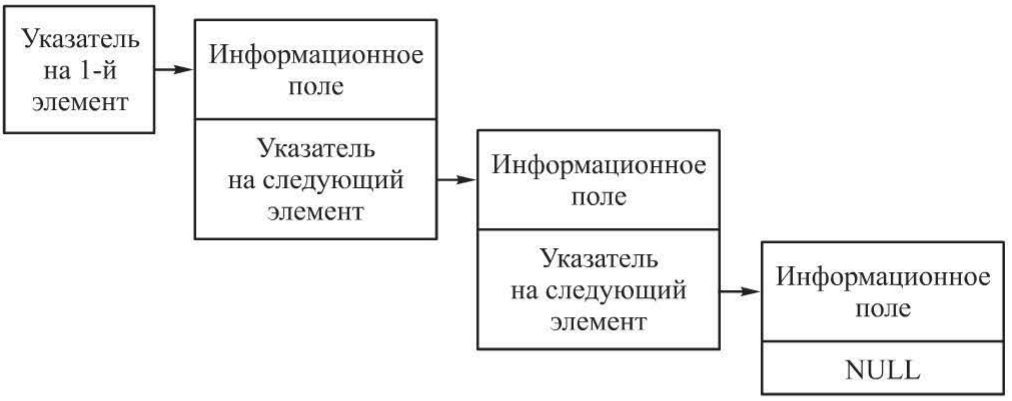


Рис. 1.7. Схема линейного односвязного списка

Структура, описывающая линейный односвязный список:

```
struct Inform { ... }; /* Информационная структура (задает информационное поле) */
struct List1
{
    Inform Inf;
    List1 *pNext;
};
```

Для работы с таким списком необходимо знать указатель на первый элемент списка (признак последнего элемента `pNext==NULL`); иногда удобно хранить указатель на последний элемент для ускорения доступа к последнему элементу, например, чтобы добавить элемент в конец списка. Более подробно с теоретическими основами для работы со списками можно ознакомиться в литературном источнике [5].

1.6.3. Задачи и порядок выполнения работы

В ходе лабораторной работы создается список объектов и осуществляется сортировка списка. Данные списка вводятся с клавиатуры, после каждого элемента идет запрос на ввод следующего элемента или завершение ввода (количество элементов заранее неизвестно). При со-

ртировке элементы списка остаются в оперативной памяти на «своих местах», изменяются только значения указателей, связывающие элементы. На экран выводится список до сортировки и после нее.

Пример выполнения работы

Условие задачи

Объект списка — работник (поля: Ф.И.О., дата приема на работу, должность, базовый оклад). Сортировка по полю «Оклад» методом прямого выбора.

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct Sotr // Работник.
{
    char fio[64]; // Ф.И.О.
    char date[16]; // Дата рождения.
    char dolg[32]; // Должность.
    double okl; // Оклад.
};

struct List // Список объектов.
{
    Sotr sotr; // Информационное поле.
    List *pNext; // Указатель на следующий элемент.
};

// Функция добавления элемента в начало списка.
void addFirst(List *& pF, // Указатель на начало списка.
    List* p) // Указатель на добавляемый элемент.
{
    p->pNext = pF;
    pF = p;
}
```

```
// Удаление элемента из начала списка
List * delFirst(List *&pF) /* Функция возвращает указа-
тель на удаляемый элемент */
{
    if (pF == 0) return 0;
    List *p = pF;
    pF = pF->pNext;
    return p;
}
// Добавление элемента перед заданным.
bool add(List *&pF, List * pZad, List *p)
{
    /* Функция возвращает true при нормальном завершении
и false в случае ошибки */
    if (pZad == pF) // Элемент будет первым.
    {
          p->pNext = pF;
          pF = p;
          return true;
    }
    List *pPred = pF; /* Указатель на предыдущий элемент
перед pZad */
    while (pPred->pNext != pZad && pPred->pNext)
          pPred = pPred->pNext;
    if (pPred->pNext == 0) return false; /* Элемента
pZad нет в списке */
    p->pNext = pZad;
    pPred->pNext = p;
    return true;
}
// Извлечение любого элемента p из списка.
List * del(List*& pF, List *p) /* Функция возвращает
указатель на извлеченный элемент */
{
    if (pF == 0) return 0;
    if (pF == p) // Удаление первого элемента.
    {
    pF = pF->pNext;
    return p;
    }
```

```
        else
        {
        List *pPred = pF; /* Указатель на предыдущий элемент
перед p */
        while (pPred->pNext != p && pPred->pNext)
        pPred = pPred->pNext;
        if (pPred->pNext == 0) return 0; /* Элемента p нет
в списке */
        pPred->pNext = p->pNext;
        return p;
        }
        while (delFirst(pF)); // Очистка списка.
    }
    int main(int argc, char* argv[])
    {
        List *pF = 0; // Список пуст.
        List *p;
        // Ввод списка.
        char Ch; /* Переменная для ввода, условие продолже-
ния ввода */
        do
        {
        p = (List *)malloc(sizeof(List)); /* Выделение па-
мяти под элемент */
        printf("\nFIO: ");
        fflush(stdin); gets_s(p->sotr.fio, 63);
        printf("Date: ");
        fflush(stdin); gets_s(p->sotr.date, 15);
        printf("Dolg: ");
        fflush(stdin); gets_s(p->sotr.dolg, 31);
        printf("Okl=");
        fflush(stdin); scanf_s("%lf", &p->sotr.okl);
        addFirst(pF, p); /* Добавление элемента в начало
списка */
        printf("For continue press Y or y else any key! ");
        Ch = _getche(); /* Чтение кода клавиши с печатью
символа */
        } while (Ch == 'Y' || Ch == 'y');
                        // Вывод списка.
        for (List *pi = pF; pi; pi = pi->pNext) /* Просмотр
списка */
        printf("\n%s %s %s oklad=%.2f", pi->sotr.fio, pi->sotr.
date, pi->sotr.dolg, pi->sotr.okl);
```

```
        // Сортировка списка.
        for (List *pi = pF; pi->pNext;)
        {
        // Поиск минимального элемента в списке.
        double min = pi->sotr.okl;
        List *pmin = pi;
        for (List *pj = pi->pNext; pj; pj = pj->pNext)
        if (pj->sotr.okl<min)
        {
        min = pj->sotr.okl;
        pmin = pj;
        }
        if (pi != pmin) /* Минимальный элемент сделать пер-
вым, он будет перед pi */
        {
        del(pF, pmin);
        add(pF, pi, pmin);
        }
        else pi = pi->pNext;
        }
        // Печать списка после сортировки.
        printf("\nSorted List:");
        for (List *pi = pF; pi; pi = pi->pNext) /* Просмотр
списка */
        printf("\n%s %s %s oklad=%.2f", pi->sotr.fio, pi->sotr.
date, pi->sotr.dolg, pi->sotr.okl);
        printf("\nFor exit press any key ");
        system("pause"); /* Остановка программы, ожидание
нажатия любой клавиши */

    return 0;
    }
```

Задания для самоконтроля

1. Раскройте особенности представления линейного односвязного списка в оперативной памяти.

2. Расскажите, какие данные нужно хранить в памяти для доступа к списку.

3. Раскройте особенности алгоритмов сортировки списков и основное их отличие от сортировки массивов.

Лабораторная работа № 1.7
Изучение стандартных функций ввода-вывода в языке Си

1.7.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си, использующих ввод-вывод в файлы. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный вводу-выводу в файлы, с помощью библиотеки стандартных функций языка Си [2, 4];
- разработать программу на языке Си для решения заданного варианта;
- отладить программу.

Выполнив работу, нужно подготовить отчет.

1.7.2. Краткая характеристика объекта изучения

Общие сведения о вводе-выводе в файлы

В Си отсутствуют встроенные операторы (операции) ввода-вывода. Для ввода-вывода используют стандартные библиотеки функций. Рассмотрим одну из стандартных библиотек ввода-вывода, большинство функций которой описаны в заголовочном файле stdio.h. Иногда функции этой библиотеки называют функциями ввода-вывода «высокого» уровня, так как эти функции не учитывают особенности конкретной архитектуры компьютера и используемой операционной системы. Эти функции присутствуют в эквивалентной форме во всех операционных системах, поддерживающих язык Си.

Рассмотрим некоторые понятия, используемые в этой библиотеке. Основное понятие — поток ввода-вывода.

Потоком (stream) называется источник или получатель данных, который можно ассоциировать с диском или другим внешним устройством ввода-вывода. Поток можно рассматривать как последовательность байтов (символов), поток связан с устройством ввода-вывода (файл, принтер, монитор, клавиатура и др.), с которым ведется обмен. С точки зрения программы поток не зависит от устройства, с которым он

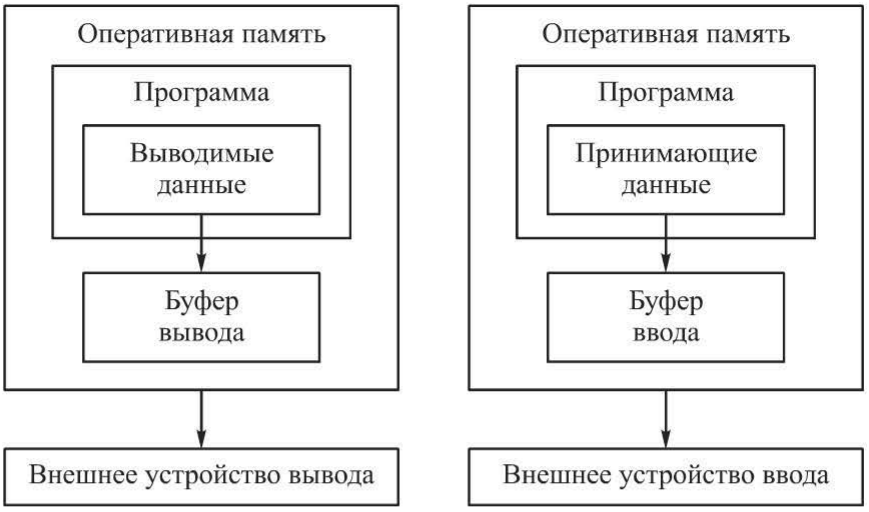


Рис. 1.8. Схема буферизированного ввода-вывода

связан, т. е. поток не зависит от технической реализации конкретного устройства ввода-вывода.

Классификация потоков ввода-вывода:

по направлению:

- входные;
- выходные;
- двунаправленные;

по использованию буфера в оперативной памяти:

- буферизированные;
- небуферизированные.

Использование буфера ввода-вывода в оперативной памяти позволяет в некоторых случаях существенно увеличить быстродействие программы, часто выполняющей операции ввода-вывода с внешними устройствами, быстродействие которых существенно меньше (на порядки), чем быстродействие оперативной памяти. При буферизированном вводе-выводе данные вначале отправляются в специальный буфер в оперативной памяти, по заполнении буфера они отправляются дальше (в переменные программы или во внешнее устройство вывода). Управляет буфером, как правило, операционная система, но существуют функции в библиотеке для сброса буфера. Схема буферизированного ввода-вывода представлена на рис. 1.8.

Открытие файла

Прежде чем работать с файлом на диске или сменном носителе, его нужно открыть. Для этого заранее объявляют указатель на структуру FILE.

Заголовок функции открытия файла:

```
FILE * fopen(const char *name, const char * mode);
```

Параметры: `name` — строка, содержащая имя файла; `mode` — строка, содержащая режим открытия файла, `mode` может состоять из двух частей, первая часть — обязательная, возможны значения-символы:

«r» — режим «чтение», файл должен существовать;

«w» — режим «запись», если файл существует, его содержимое стирается;

«a» — режим «добавление», если файл не существует, он создается заново, если файл существует, его содержимое не теряется, запись производится всегда в конец файла;

«r+» — режим «чтение + запись», файл должен существовать (как при чтении);

«w+» — режим «запись + чтение», если файл существует, его содержимое стирается (как при записи);

«a+» — режим «добавление + чтение», как в режиме «добавление» запись всегда производится в конец файла.

Вторая часть — необязательная, возможны значения-символы:

«t» — файл открывается в текстовом режиме (по умолчанию), при чтении последовательность символов «\r\n» преобразуется в «\n», при записи происходит обратное преобразование, кроме того, символ с кодом 1A интерпретируется как конец файла;

«b» — файл открывается в двоичном режиме, подобные преобразования не проводятся.

Функция возвращает указатель на структуру FILE или NULL в случае ошибки.

Примеры открытия файлов:

```
FILE *pF;
pF=fopen("MyFile.txt", "w");
```

Здесь создается файл в текущем каталоге операционной системы с именем `MyFile.txt`,

Файл создается для записи в текстовом режиме. Если файл с таким именем уже существовал, его старое содержимое стирается.

```
FILE *pF;
pF=fopen("C:\\Catalog1\\MyFile.dat", "rb");
```

Открывается файл с именем `MyFile.dat` на диске C: в каталоге `Catalog1`.

Файл открывается для чтения в двоичном режиме и должен существовать, в противном случае функция возвращает значение NULL.

Следует обратить внимание на то, что символ «\» внутри строки является служебным, поэтому чтобы его включить в строку в качестве обычного символа, перед ним ставится тот же символ «\».

```
FILE *pF;
pF=fopen("C:\\Catalog1\\MyFile.txt", "a+");
```

Открывается файл с именем `MyFile.txt` на диске C: в каталоге `Catalog1`.

Файл открывается для добавления и чтения в текстовом режиме и может существовать; если он не существует, то создается заново.

Следует отметить, что компилятор Microsoft Visual C++ 2013 выдает ошибку при использовании `fopen` (более ранние компиляторы выдавали предупреждение), сообщается, что функция `fopen` является небезопасной и устаревшей, и предлагается воспользоваться другой функцией — `fopen_s`. Функция имеет заголовок:

```
errno_t fopen_s(
    FILE** pFile,
    const char *filename,
    const char *mode
);
```

Основное различие в том, что указатель на структуру FILE передается в первый параметр функции через указатель. Пример открытия файла для записи:

```
FILE *pF;
fopen_s(&pF, "MyFile.txt", "w");
```

Закрытие файла

Открытые файлы используют ресурсы операционной системы, поэтому, если работа с ними завершена, их требуется закрывать. Кроме того, если открыт файл для записи и при завершении работы приложения он не закрыт, то, возможно, не все записанные данные окажутся в этом файле, если запись проводится через буфер ввода-вывода.

Заголовок функции для закрытия файла:

```
int fclose(FILE * stream);
```

Параметр `stream` — указатель на структуру FILE, который связан с открытым файлом, функция возвращает 0, если закрытие реализовано или EOF — в случае ошибки.

Определение конца файла

Когда проводятся операции чтения/записи в файл, существует понятие указателя текущей позиции файла. При открытии файла текущая позиция указателя находится в начале файла, значение указателя текущей позиции равно 0, далее при чтении или записи он перемещается по файлу, значение указателя текущей позиции соответствует смещению позиции от начала файла в байтах.

Функция, которая позволяет проверить, достиг ли указатель текущей позиции конца файла, имеет заголовок:

```
int feof(FILE * stream);
```

Параметр `stream` — указатель на структуру FILE, который связан с открытым файлом, возвращает не 0, если курсор находится на конце файла, или в противном случае 0.

Функции записи в файл

Следующие функции работают так же, как аналогичные функции для записи в поток `stdout`. Но эти функции имеют на один параметр больше. Параметр `stream` — указатель на структуру FILE, который связан с открытым файлом. Ниже представлены заголовки функций.

Запись символа в файл:

```
int fputc(int c, FILE *stream);
```

Запись строки символов в файл:

```
int fputs(const char * s, FILE * stream);
```

Запись данных в файл в соответствии с заданным форматом (форматированный вывод):

```
int fprintf(FILE * stream, const char * format, ...);
```

Рассмотрим еще одну полезную функцию записи, которая применяется, как правило, в двоичном режиме.

Запись данных в файл из оперативной памяти без преобразований:

```
unsigned fwrite(void * buf, unsigned size, unsigned
count, FILE *stream);
```

Параметры: `buf` — адрес в оперативной памяти, начиная с которого записываются данные; `size` — размер блока данных в байтах; `count` — количество блоков данных (при записи массива); `stream` — указатель на структуру FILE; возвращаемое значение — количество блоков, реально записанных в файл.

Функции чтения из файла

Следующие функции работают так же, как аналогичные функции для чтения из потока `stdin`. Данные функции также имеют на один параметр больше. Параметр `stream` — указатель на структуру FILE, который связан с открытым файлом. Ниже представлены заголовки функций.

Чтение символа из файла:

```
int fgetc(FILE * stream);
```

Чтение строки символов из файла:

```
char * fgets(char * s, int n, FILE * stream);
```

Параметр `n` — максимальное количество читаемых символов.

Считывание данных из файла и запись их в переменные в соответствии с заданным форматом:

```
int fscanf(FILE * stream, const char * format, ...);
```

Дополнительно рассмотрим функцию чтения, которая применяется в двоичном режиме.

Чтение данных из файла и запись их в оперативную память без преобразований:

```
unsigned fread(void *buf, unsigned size, unsigned count,
FILE *stream);
```

Параметры: `buf` — адрес в оперативной памяти, куда записываются данные; `size` — размер блока данных в байтах; `count` — количество блоков данных (при чтении массива); `stream` — указатель на структу-

ру FILE; возвращаемое значение — количество реально прочитанных блоков из файла.

Функции позицирования в файлах

При выполнении операций чтения/записи в файле требуется иногда перемещать указатель текущей позиции файла, например, при переходе от чтения к записи или наоборот (когда файл открыт в комбинированных режимах). Для перемещения указателя существуют специальные функции.

Функция для установки указателя текущей позиции в заданное положение:

```
int fseek(FILE * stream, long offset, int fromwhere);
```

Параметры: `stream` — указатель на структуру FILE; `offset` — смещение (в байтах) указателя текущей позиции от позиции, заданной следующим параметром; `fromwhere` — возможные значения заданы константами: `SEEK_SET` — начало файла, `SEEK_CUR` — текущее положение указателя, `SEEK_END` — конец файла. Возвращаемые значения: 0, если функция нормально завершает свою работу, не 0 — в случае ошибки.

Функция для установки указателя текущей позиции в начало файла:

```
void rewind(FILE * stream);
```

Параметр `stream` — указатель на структуру FILE.

Функция для получения текущего положения указателя позиции файла (смещение в байтах от начала файла):

```
long ftell(FILE * stream);
```

Параметр `stream` — указатель на структуру FILE; функция возвращает текущее положение указателя.

Функции для сброса буферов ввода-вывода

Иногда требуется явно проводить сброс буфера ввода-вывода. В частности, эта операция обязательна при переходе от операции записи к операции чтения (при открытии файла в комбинированном режиме), чтобы записанные данные гарантированно оказались записаны в файле.

Функция сброса заданного буфера ввода-вывода:

```
int fflush(FILE * stream);
```

Параметр `stream` — указатель на структуру FILE; функция возвращает 0 в случае нормального завершения и EOF — в случае ошибки.

Функция сброса всех буферов ввода-вывода:

```
int flushall();
```

Функция возвращает количество открытых потоков (входных и выходных), в случае ошибки возвращаемое значение не определено.

1.7.3. Задачи и порядок выполнения работы

В лабораторной работе нужно разработать два приложения. В первом приложении исходные данные вводят с клавиатуры (количество объектов, заданных структурным типом, заранее неизвестно) и сохраняют в файле. Во втором приложении данные, сохраненные в первом приложении в файле, читают из файла и выводят на экран. В зависимости от варианта данные можно сохранять в файле как в текстовом, так и в двоичном режиме.

Пример выполнения работы

Условие задачи

Структура книги (поля: Ф.И.О. автора, название, наименование издательства, год издания, количество страниц). Создается массив книг размерности *n*. Сохраняется в файле и читается из файла. Файл открывается в двоичном режиме.

Для решения задач в среде Microsoft Visual Studio 2013 были созданы стандартные консольные приложения (проект типа Win32 Console Application) с установленным свойством «пустой проект» (Empty project). В каждый из проектов добавлен файл с расширением `.cpp`, исходные коды приведены ниже.

Листинг программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct BOOK // Структура «книга».
{
    char Author[64];
    char Title[128];
    char Firm[64];
    int year, page;
};
int main(int argc, char* argv[])
{
    int n; /* Переменная, задающая количество элементов
массива */
    BOOK *pBook; // Указатель на массив структур.
    printf("n="); scanf_s("%d", &n); /* Ввод количества
книг */
    pBook = new BOOK[n]; /* Выделение памяти под массив
структур (книг) */
    for (int i = 0; i<n; i++) /* Цикл ввода данных о
книгах с клавиатуры */
    {
      printf("Book N=%d:\n", i + 1);
      printf("Author: ");
      fflush(stdin); gets_s(pBook[i].Author, 63);
      printf("Title: ");
      fflush(stdin); gets_s(pBook[i].Title, 127);
      printf("Firm: ");
      fflush(stdin); gets_s(pBook[i].Firm, 63);
      printf("year: "); scanf_s("%d", &pBook[i].year);
      printf("page: "); scanf_s("%d", &pBook[i].page);
    }
    FILE *pF;
    fopen_s(&pF, "MyBook.dat", "wb"); /* Открытие файла
для записи в двоичном режиме */
    fwrite(&n, sizeof(int), 1, pF); /* Запись в файл
количества элементов массива */
    fwrite(pBook, sizeof(BOOK), n, pF); /* Запись в файл
массива книг */
    fclose(pF); // Закрытие файла.
    system("pause"); /* Остановка программы, ожидание
нажатия любой клавиши */
    return 0;
}
```

Листинг программы с комментариями для чтения из файла

```
#include <stdio.h>
#include <stdlib.h>
struct BOOK // Структура «книга».
{
    char Author[64];
    char Title[128];
    char Firm[64];
    int year, page;
};
int main(int argc, char* argv[])
{
    int n; /* Переменная, задающая количество элементов
массива */
    BOOK *pBook; // Указатель на массив структур.
    FILE *pF;
    fopen_s(&pF, "MyBook.dat", "rb"); /* Открытие файла
для чтения в двоичном режиме */
    if (pF == 0) /* Ошибка открытия файла, например
файла не существует */
    {
        printf("Error, file not found");
        return 1;
    }
    fread(&n, sizeof(int), 1, pF); /* Чтение из файла
количества элементов массива */
    pBook = new BOOK[n]; /* Выделение памяти под массив
структур (книг) */
    fread(pBook, sizeof(BOOK), n, pF); /* Чтение из фай-
ла массива книг */
    fclose(pF); // Закрытие файла.
    for (int i = 0; i<n; i++) /* Цикл вывода данных о
книгах на экран */
        printf("%s. %s. %s, %d. - %d.\n", pBook[i].
Author, pBook[i].Title, pBook[i].Firm, pBook[i].year,
        pBook[i].page);
    system("pause");  /* Остановка программы до нажатия
любой клавиши */
    return 0;
}
```

Задания для самоконтроля

1. Перечислите режимы открытия файлов.
2. Объясните, в чем отличие работы с файлом в текстовом режиме от работы в двоичном режиме.
3. Перечислите режимы открытия файлов, в которых файл обязательно должен существовать до открытия.
4. Перечислите режимы открытия файлов, в которых файл создается заново (существующий файл очищается).

Лабораторная работа № 1.8
Изучение приложений с графическим интерфейсом пользователя для Windows

1.8.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки приложений с графическим интерфейсом пользователя для операционных систем типа Windows XP/7/8/10. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебный материал, посвященный разработке приложений с графическим интерфейсом пользователя для операционных систем типа Windows [6];
- разработать программу на языке Си для решения заданного варианта;
- отладить программу.

Выполнив работу, нужно подготовить отчет.

1.8.2. Краткая характеристика объекта изучения

Сообщения Windows

Операционная система (ОС) Windows предоставляет пользователю множество системных функций для использования ее возможностей и доступа к аппаратуре компьютера, эти функции составляют API

(Application Programming Interface) — Win API. Кроме того, ОС Windows обменивается информацией с приложением с помощью механизма сообщений. Сообщение — это структура в памяти типа MSG. Когда пользователь выполняет некоторые действия с элементами интерфейса или работает с мышью либо клавиатурой, ОС посылает приложению сообщение с информацией об этом. Все стандартные сообщения имеют идентификатор (номер), заданный разработчиками ОС. Примеры идентификаторов сообщений (определены в заголовочных файлах директивой #define, являются целыми константами): `WM_LBUTTONUP`, `WM_LBUTTONDOWN`, `WM_LBUTTONDBLCLK`, `WM_MOUSEMOVE`, `WM_CHAR`, `WM_KEYUP`, `WM_KEYDOWN`, `WM_ACTIVATE`, `WM_CREATE`, `WM_DESTROY`, `WM_MOVE`, `WM_SIZE`, `WM_KILLFOCUS`, `WM_PAINT`, `WM_COMMAND`, `WM_TIMER` [6].

Структура приложения в Windows

Приложение с графическим интерфейсом пользователя состоит из двух основных частей:

1) функция `WinMain` — точка входа в приложение (как правило, создает окно и запускает цикл обработки сообщений);

2) функция окна — является функцией обратного вызова, вызывается многократно операционной системой для обработки сообщений приложения.

Стандартный заголовок функции `WinMain` имеет вид

```
int WINAPI WinMain(HINSTANCE hIns, HINSTANCE hPrevIns,
LPSTR arg, int WinMode)
```

Назначение функции `WinMain`:
определение класса окна;
регистрация класса окна;
создание окна;
отображение окна;
запуск цикла обработки сообщений.

Стандартный заголовок функции окна (функции обратного вызова) имеет вид

```
LRESULT CALLBACK WinFun(HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam)
```

Внутри функции окна обычно для обработки сообщений используют переключатель, который имеет следующую структуру:

```
switch(message)
    {
    case WM_CREATE: // Создание окна.
        ....      break;
    case WM_DESTROY: // Завершение программы.
          PostQuitMessage(0);
          break;
    case WM_LBUTTONDOWN: // Нажатие левой клавиши мыши.
                ...     break;
    default: // Обработка сообщений, по умолчанию.
          return DefWindowProc(hwnd, message, wParam,
lParam);
    }
```

Вывод графики в Windows

Контексты устройств

Контекст устройства (Device Context) — некоторое логическое представление физического устройства (экрана монитора, принтера и т. д.) — внутренняя структура для управления информацией о выходном устройстве. Она содержит информацию о параметрах и атрибутах («перо», «кисть», шрифт и др.) вывода графики на устройство (например, дисплей или принтер). Вместо направления вывода непосредственно на аппаратное устройство приложение направляет его в контекст устройства, а затем Windows пересылает вывод в аппаратное устройство.

Типы контекстов устройств:

- контекст дисплея;
- контекст принтера;
- контекст в памяти (моделирует в памяти устройство вывода);
- информационный контекст (служит для получения данных от устройства).

Заголовки функции для получения контекста устройств (в программе используются хэндлы контекстов устройств — HDC):

```
HDC GetDC(HWND hWnd);
HDC BeginPaint(HWND hWnd, LPPAINTSTRUCT lpPaint);
```

Освобождение контекста устройств:

```
int ReleaseDC(HWND hWnd, HDC hDC);
BOOL EndPaint(HWND hWnd, CONST PAINTSTRUCT *lpPaint);
```

Графические «перья» и «кисти»

Заголовок функции для получения предопределенных «перьев» или «кистей»:

```
HGDIOBJ GetStockObject(int);
```

Возможные значения параметров функции для «перьев»:

```
WHITE_PEN, BLACK_PEN, NULL_PEN
```

для «кистей»:

```
WHITE_BRUSH, LTGRAY_BRUSH, GRAY_BRUSH, DKGRAY_BRUSH,
BLACK_BRUSH, NULL_BRUSH
```

Заголовок функции для создания «пера»:

```
HPEN      CreatePen(int, int, COLORREF);
```

Первый параметр определяет стиль, возможные значения (заданы константами в заголовочных файлах):

```
PS_SOLID                  /* _______ */
PS_DASH                     /* -------   */
PS_DOT                        /* .......   */
PS_DASHDOT          /* _._._._   */
PS_DASHDOTDOT             /* _.._.._   */
PS_NULL
```

Второй параметр определяет толщину.

Третий параметр определяет цвет; для получения цвета можно использовать макрос:

```
RGB(r,g,b)
```

Он возвращает тип, объявленный как

```
typedef    DWORD    COLORREF;
```

Заголовок функции для создания сплошной «кисти»:

```
HBRUSH   CreateSolidBrush(COLORREF);
```

Заголовок функции для создания «кисти» с заданным стилем:

```
HBRUSH   CreateHatchBrush(int, COLORREF);
```

Первый параметр определяет стиль, возможные значения, обозначенные в заголовочных файлах:

```
HS_HORIZONTAL      /* ----- */
HS_VERTICAL              /* ||||| */
HS_FDIAGONAL           /* \\\\\ */
HS_BDIAGONAL            /* ///// */
HS_CROSS                        /* +++++ */
HS_DIAGCROSS             /* xxxxx */
```

Заголовок функции для загрузки «пера» или «кисти» в контекст устройства:

```
HGDIOBJ SelectObject(HDC, HGDIOBJ);
```

После работы с объектами («пером» и «кистью») их нужно удалить. Заголовок функции для удаления имеет вид

```
BOOL DeleteObject(HGDIOBJ);
```

Функции для вывода графики

Заголовки некоторых функций для вывода графических изображений:

```
COLORREF SetPixel(HDC, int, int, COLORREF);
COLORREF SetTextColor(HDC, COLORREF);
COLORREF SetBkColor(HDC, COLORREF);
BOOL  TextOutA(HDC, int, int, LPCSTR, int);
BOOL  MoveToEx(HDC, int, int, LPPOINT);
BOOL LineTo(HDC, int, int);
BOOL Rectangle(HDC, int, int, int, int);
```

```
BOOL Ellipse(HDC, int, int, int, int);
BOOL  Polygon(HDC, CONST POINT *, int);
```

Более подробно особенности приложений с графическим интерфейсом пользователя для Windows рассмотрены в работе [6].

1.8.3. Задачи и порядок выполнения работы

В разработанном приложении необходимо реализовать некоторые функции простого графического редактора — рисование графических примитивов в окне с помощью мыши. Дополнительные функции (оцениваются дополнительными баллами): перерисовка, использование полосы прокрутки и др. Особое внимание следует обратить на структуру графического приложения, состоящего из двух основных функций — `WinMain` и обратного вызова (окна), а также на то, что основные функции приложения реализуются через механизм обработки сообщений в функции окна.

Пример выполнения работы

Условие задачи

Разработать приложение для рисования в окне с помощью мыши (функции простого графического редактора) линий. Начальная точка линии — точка положения курсора мыши в момент нажатия клавиши. Рисование осуществляется при перемещении мыши с нажатой клавишей, конечная точка — положение курсора в момент отпускания клавиши. Используется правая клавиша мыши. Линии следует рисовать сплошным черным «пером».

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное оконное приложение (проект типа Проект Win32 или Win32 Project в нерусифицированной версии) с установленным свойством «Пустой проект» (Empty project). В проект добавлен файл с расширением `.cpp`, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <windows.h>
LRESULT CALLBACK WinFun(HWND, UINT, WPARAM, LPARAM);
char WinName[]="Мое окно"; // Имя класса окна.
int WINAPI WinMain(HINSTANCE hIns, HINSTANCE hPrevIns,
LPSTR arg, int WinMode)
```

```
{
    HWND hwnd; // Дескриптор окна.
    MSG msg; /* Содержит информацию о сообщении, посы-
лаемом Windows */
    WNDCLASSEX wcl; // Определяет класс окна.
    // Определение класса окна.
    wcl.hInstance=hIns; // Дескриптор данного экземпляра.
    wcl.lpszClassName=WinName; //Имя класса.
    wcl.lpfnWndProc=WinFun; //Функция окна.
    wcl.style=0; //Стиль, по умолчанию.
    wcl.cbSize=sizeof(WNDCLASSEX); //Размер структуры.
    wcl.hIcon=LoadIcon(NULL, IDI_APPLICATION); /* Большая
пиктограмма*/
    wcl.hIconSm=LoadIcon(NULL, IDI_WINLOGO); /* Малая
пиктограмма*/
    wcl.hCursor=LoadCursor(NULL, IDC_ARROW); /* Форма
курсора */
    wcl.lpszMenuName=NULL; // Меню не используется.
    wcl.cbClsExtra=0; /* Дополнительная информация от-
сутствует */
    wcl.cbWndExtra=0;
    // Фон окна задается белым.
    wcl.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    // Регистрация класса окна.
    if (!RegisterClassEx(&wcl)) return 0;
    // Создание окна.
    hwnd=CreateWindow(
          WinName, // Имя класса окна.
          "Мое простое окно", // Заголовок.
          WS_OVERLAPPEDWINDOW, // Стандартное окно.
          CW_USEDEFAULT, /* Координата X определяется
Windows */
          CW_USEDEFAULT, /* Координата Y определяется
Windows */
          CW_USEDEFAULT, // Ширина определяется Windows.
          CW_USEDEFAULT, // Высота определяется Windows.
          HWND_DESKTOP, // Родительского окна нет.
          NULL, // Меню нет.
      hIns, // Дескриптор данного экземпляра приложения.
          NULL); // Дополнительных аргументов нет.
    // Отображение окна.
```

```
        ShowWindow(hwnd, WinMode);
        UpdateWindow(hwnd);
        // Цикл сообщений.
        while(GetMessage(&msg, NULL, 0, 0))
        {
        TranslateMessage(&msg); /* Перевод сообщений формата
виртуальных клавиш в сообщения-символы */
        DispatchMessage(&msg);
        }
        return msg.wParam;
    }

    struct Line // Линия.
    {
        int x1, x2, y1, y2;
    };
    struct LineList // Список линий.
    {
        Line L;
        LineList *pNext;
    };
    LineList *pFirst=0, *p;
    void add(LineList *&pF, LineList *p)
    { // Добавляем элемент в начало списка.
        p->pNext=pF;
        pF=p;
    }
    int x1, x2, y1, y2;
    HPEN pB, pW;

    /* Эта функция вызывается Windows, которая передает ей
сообщение из очереди сообщений */

    LRESULT CALLBACK WinFun(HWND hwnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
    {
        HDC hdc;
        PAINTSTRUCT ps;
        switch(message)
        {
        case WM_CREATE:
```

```
            pB=(HPEN)GetStockObject(BLACK_PEN);
            pW=(HPEN)GetStockObject(WHITE_PEN);
            break;
      case WM_RBUTTONDOWN:
            x2=x1=LOWORD(lParam);
            y2=y1=HIWORD(lParam);
            break;
      case WM_MOUSEMOVE:
            if (wParam & MK_RBUTTON) /* Определяем, на-
жата ли правая кнопка мыши */
            {
            hdc=GetDC(hwnd);
            SelectObject(hdc, pW);
            MoveToEx(hdc, x1, y1, (LPPOINT)NULL);
            LineTo(hdc, x2, y2);
            x2=LOWORD(lParam);
                  y2=HIWORD(lParam);
            SelectObject(hdc, pB);
            MoveToEx(hdc, x1, y1, (LPPOINT)NULL);
            LineTo(hdc, x2, y2);
            ReleaseDC(hwnd, hdc);
            }
            break;
      case WM_RBUTTONUP: // Отпускаем правую кнопку мыши.
            p=new LineList;
            p->L.x1=x1; p->L.x2=x2;
            p->L.y1=y1; p->L.y2=y2;
            add(pFirst, p);
            break;
      case WM_PAINT: // Перерисовка.
            hdc=BeginPaint(hwnd, &ps);
            p=pFirst;
            while(p)// Просмотр списка и рисование линии.
            {
            MoveToEx(hdc, p->L.x1, p->L.y1, (LPPOINT)NULL);
            LineTo(hdc, p->L.x2, p->L.y2);
            p=p->pNext;
            }
            EndPaint(hwnd, &ps);
            break;
      case WM_DESTROY: // Завершение программы.
```

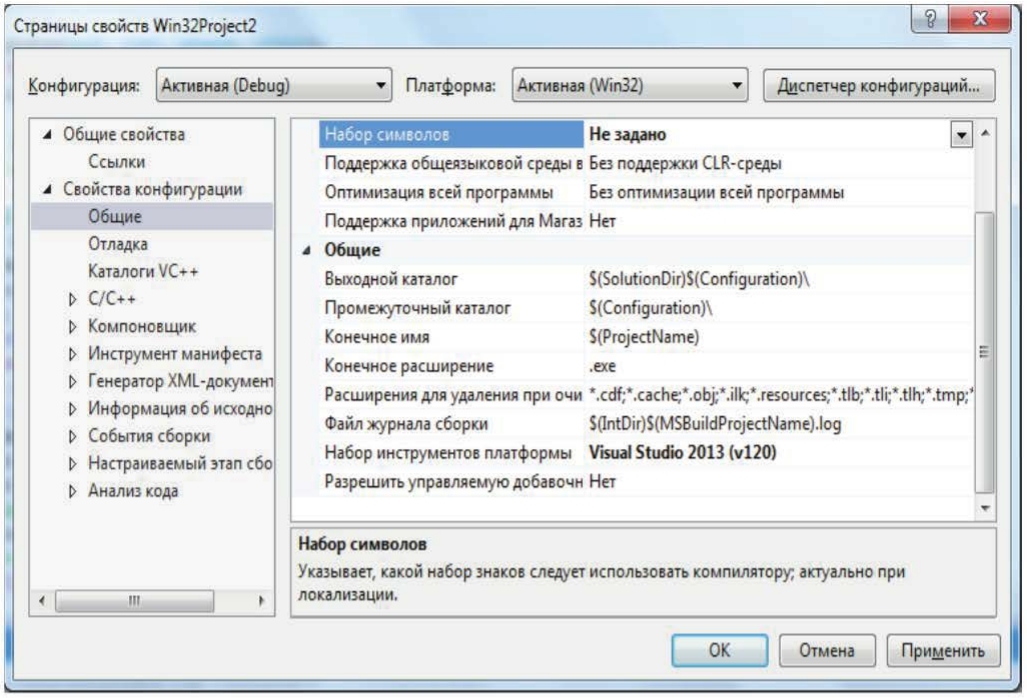


Рис. 1.9. Отключение набора символов `Unicode`

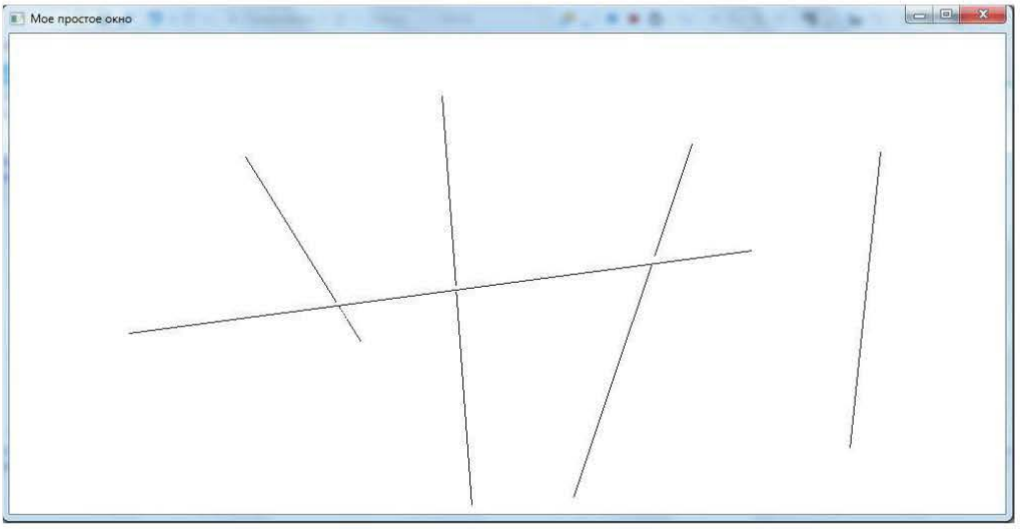


Рис. 1.10. Пример с результатами работы программы

```
            PostQuitMessage(0);
            break;
        default:
        /* Позволяет Windows обрабатывать любые сообщения,
не указанные в предыдущем случае */
            return DefWindowProc(hwnd, message, wParam,
lParam);
        }
        return 0;
    }
```

Следует отметить, что для запуска данной программы необходимо в окне свойств проекта отключить использование кодировки Unicode, т. е. установить опцию «Набор символов» в состояние «Не задано» (рис. 1.9).

Возможные результаты работы программы представлены на рис. 1.10.

Задания и вопросы для самоконтроля

1. Опишите назначение функции WinMain.
2. Опишите назначение функции окна.
3. Что такое класс класса окна и как он задается?
4. Приведите примеры сообщений Windows.
5. Раскройте особенности обработки сообщения «перерисовка окна».

Лабораторная работа № 1.9
Изучение диалоговых окон и элементов управления в Win API

1.9.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в изучении основных API-функций (Application Programming Interface) для работы с диалоговыми окнами и элементами управления и в овладении навыками разработки программ на языке Си++ с использованием диалоговых окон и элементов управления.

Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные использованию диалоговых окон и элементов управления в языке Си с помощью API-функций Windows [6];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программы;
- представить скриншот окна с результатами работы программы.

Выполнив работу, следует подготовить отчет.

1.9.2. Краткая характеристика объекта изучения

Классификация диалоговых окон

Диалоговые окна можно классифицировать по следующим признакам:

- *по модальности*
  - — модальные (блокируют работу пользователя с родительским окном до тех пор, пока пользователь не закроет модальное диалоговое окно);
  - — немодальные (не блокируют работу пользователя с родительским окном).
- *по назначению*
  - — окна сообщений;
  - — стандартные (выбор файла, цвета, шрифта и др.);
  - — специальные (их создает программист для решения задач).

Окна сообщений

Функция для создания окна сообщений может иметь заголовок:

```
int MessageBoxA(HWND hWnd , // Хэндл родительского окна.
    LPCSTR lpText, // Текст в окне.
    LPCSTR lpCaption,   // Заголовок.
    UINT uType); // Тип окна (наличие кнопок и иконка).
```

Возможные значения параметра `uType` задаются константами:

```
MB_OK
MB_OKCANCEL
```

```
MB_ABORTRETRYIGNORE
MB_YESNOCANCEL
MB_YESNO
MB_RETRYCANCEL
MB_ICONHAND
MB_ICONQUESTION
MB_ICONEXCLAMATION
MB_ICONASTERISK
```

Константы, задающие наличие кнопок и вид иконки (MB_ICON...), можно использовать совместно с помощью поразрядной операции «ИЛИ» (|).

Возвращаемое значение зависит от того, какая кнопка нажата для закрытия окна. Возможные значения:

```
IDOK
IDCANCEL
IDABORT
IDRETRY
IDIGNORE
IDYES
IDNO
```

Создание модального диалогового окна

Удобнее всего диалоговое окно вместе с элементами управления описывать в файле ресурсов. В этом случае функция для создания окна имеет заголовок:

```
DialogBox(hInstance, //Хэндл приложения.
lpTemplate, //Строка – название ресурса окна.
hWndParent, //Хэндл родительского окна.
lpDialogFunc); //Указатель на функцию окна.
```

Функция для закрытия окна:

```
BOOL EndDialog(HWND hDlg, // Хэндл закрываемого окна.
int nResult);//Значение, которые вернет DialogBox при выходе.
```

Для инициализации диалогового окна (задания начального состояния элементов управления, присвоения значений переменным и др.) необходимо обработать сообщение WM_INITDIALOG.

Ниже показан пример функции окна для диалогового окна, имеющего две стандартные кнопки — Ok и Cancel.

```
BOOL CALLBACK DialogFun(HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
    switch(message)
    {
    case WM_INITDIALOG:
          return TRUE;
    case WM_COMMAND:
          switch(LOWORD(wParam))
          {
          case IDOK:
          case IDCANCEL:
                EndDialog(hwnd, LOWORD(wParam));
                return TRUE;
          }
    }
    return FALSE;
}
```

Пример создания этого диалогового окна, определенного в файле ресурсов с идентификатором IDD_DIALOG1:

```
    if (DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1),
0, DialogFun)==IDOK)
          MessageBox(0, "Нажата Ok", "Сообщение", MB_OK);
    else
          MessageBox(0, "Нажата Cancel", "Сообщение",
MB_OK);
```

Управление диалоговым окном

Основные элементы управления: кнопка, контрольный переключатель, радиокнопка, текстовое поле, список и т. д.

Общие или дополнительные элементы управления (требуется подключение специальной библиотеки): строка состояния, спин, регулятор, индикатор процесса и т. д.

Большинство элементов управления — это разновидности окон. Функция для получения хэндла элемента управления имеет заголовок:

```
HWND GetDlgItem(HWND hDlg,  // Хэндл родительского окна.
  int nIDDlgItem); // Идентификатор элемента управления.
```

Функции для посылки сообщений элементам управления имеют заголовки:

```
LRESULT SendDlgItemMessage( HWND hDlg, /* Хэндл диало-
гового окна*/
int nIDDlgItem,  // Идентификатор элемента управления.
UINT Msg,  // Тип сообщения.
WPARAM wParam, LPARAM lParam); // Параметры сообщения.
LRESULT SendMessage(HWND hWnd,  /* Хэндл элемента управ-
ления */
UINT Msg,  // Тип сообщения.
WPARAM wParam,     LPARAM lParam); /* Параметры сообще-
ния */
```

Рассмотрим часть элементов, которые необходимо использовать при выполнении лабораторной работы.

Кнопка, контрольный переключатель, радиокнопка. Сообщение, приходящее от элементов управления `WM_COMMAND`, несет определенную информацию.

Младшее слово `wParam` содержит значение идентификатора элемента управления. Старшее слово `wParam` определяет действия с кнопкой (нотификационные сообщения), некоторые его возможные значения:

```
BN_CLICKED          // «Нажатие» на элемент.
BN_PAINT           // Перерисовка элемента.
BN_DOUBLECLICKED   // Двойной щелчок.
BN_SETFOCUS // Элемент получил фокус ввода с клавиатуры.
BN_KILLFOCUS   // Элемент потерял фокус ввода с клавиатуры.
```

Сообщения, которые можно посылать элементам:

```
BM_GETCHECK    BM_SETCHECK     BM_GETSTATE     BM_SETSTATE
BM_SETSTYLE      BM_CLICK      BM_GETIMAGE     BM_SETIMAGE
```

Текстовое поле. Сообщения от текстового поля также `WM_COMMAND`.

Некоторые нотификационные сообщения от текстового поля (старшее слово wParam):

```
EN_SETFOCUS // Элемент получил фокус ввода с клавиатуры.
EN_KILLFOCUS // Элемент потерял фокус ввода с клавиатуры.
EN_CHANGE   // Текст в текстовом поле изменен.
```

Основные сообщения, которые можно посылать текстовым полям:

```
WM_SETTEXT   // Поместить текст в текстовое поле.
WM_GETTEXT   // Извлечь текст из текстового поля.
```

Примеры

Поместить текст в текстовое поле:

```
SendDlgItemMessage(hwnd, IDC_EDIT1, WM_SETTEXT, 0,
(LPARAM)Text);
```

Получить текст из текстового поля:

```
SendDlgItemMessage(hwnd, IDC_EDIT2, WM_GETTEXT, 255,
(LPARAM)Text);
```

До этого где-то объявлен массив символов, в котором текст хранится как строка: `char Text[256];`

Список. Элементами списка являются строки. Индексация элементов начинается с нуля. Сообщение от списка такое же, как для рассмотренных выше элементов, — `WM_COMMAND`.

Ниже приведены коды некоторых нотификационных сообщений, приходящих от списка (старшее слово wParam):

```
LBN_SELCHANGE   // Выбор элемента изменен.
LBN_DBLCLK     // Двойной щелчок на элементе списка.
LBN_SETFOCUS // Элемент получил фокус ввода с клавиатуры.
LBN_KILLFOCUS /* Элемент потерял фокус ввода с клавиатуры */
```

Коды некоторых сообщений, посылаемых списку:

```
LB_ADDSTRING // Добавить строку в список.
```

```
LB_INSERTSTRING  // Вставить строку в список.
LB_DELETESTRING // Удалить элемент из списка.
LB_SETCURSEL // Установить выбор заданного элемента.
LB_GETCURSEL // Получить индекс выбранного элемента.
LB_GETTEXT // Получить текст из элемента.
LB_GETCOUNT // Получить количество элементов списка.
LB_RESETCONTENT // Очистить список.
```

Примеры посылки сообщений списку

Добавить строку текста в список:

```
SendDlgItemMessage(hwnd, IDC_LIST1, LB_ADDSTRING, 0,
(LPARAM)Text);
```

Получить индекс выделенного элемента:

```
int i=SendDlgItemMessage(hwnd, IDC_LIST1, LB_GETCURSEL,
0, 0);
```

Извлечь строку из элемента с индексом *i*:

```
SendDlgItemMessage(hwnd, IDC_LIST1, LB_GETTEXT, i,
(LPARAM)Text);
```

Удалить элемент с индексом *i*:

```
SendDlgItemMessage(hwnd, IDC_LIST1, LB_DELETESTRING, i, 0);
```

Очистить список:

```
SendDlgItemMessage(hwnd, IDC_LIST1, LB_RESETCONTENT, 0, 0);
```

1.9.3. Задачи и порядок выполнения работы

Условие задачи

Разработать приложение на основе диалогового окна — калькулятор для вычисления арифметических операций (+ –). Все проведенные операции добавляются в список, который можно очистить при нажатии кнопки. Калькулятор может иметь примерный вид, представленный на рис. 1.11 (в зависимости от варианта задания).

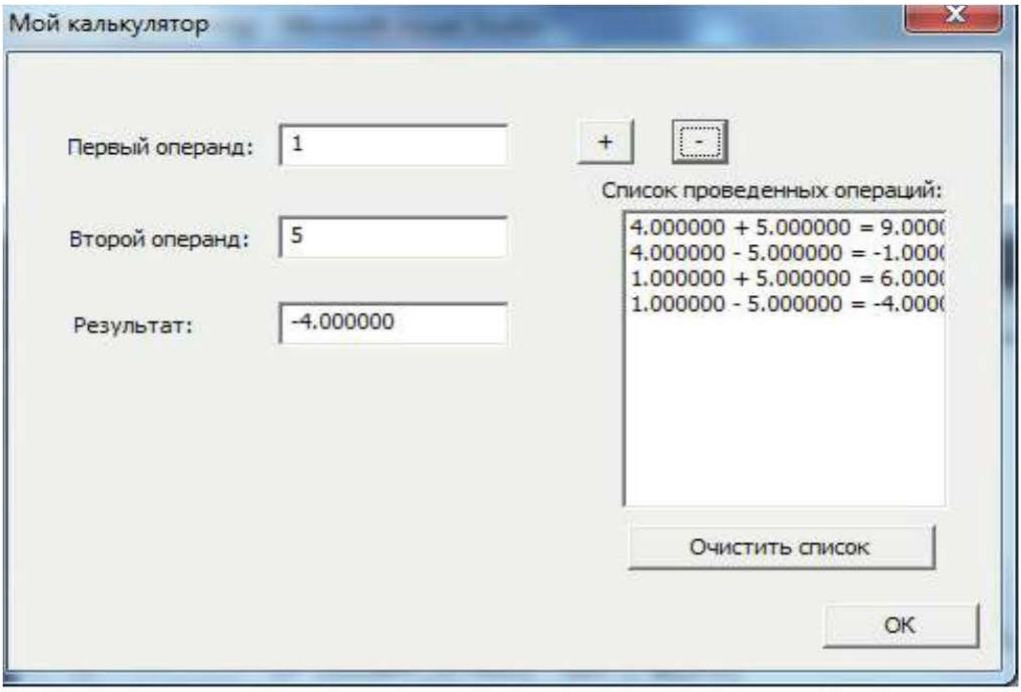


Рис. 1.11. Приложение типа калькулятор

Пример выполнения работы

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное оконное приложение (проект типа Win32 Project, в русифицированной версии — Проект Win32) с включенным свойством «Пустой проект» (Empty project). В проект добавлен файл с расширением `.cpp`, исходный код которого приведен в листинге ниже. В редакторе ресурсов создано диалоговое окно как ресурс (см. рис. 1.11). При создании ресурсов автоматически создаются файл `resource.h` и файл с расширением `.rc`, исходные тексты которых не приводятся. В этих файлах идентификатор диалогового окна `IDD_DIALOG1`, идентификаторы текстовых полей `IDC_EDIT1`, `IDC_EDIT2`, `IDC_EDIT3`, идентификаторы кнопок `IDC_BUTTON1`, `IDC_BUTTON2`, `IDC_BUTTON3`, идентификатор списка `IDC_LIST1`.

Листинг программы с комментариями

```
#include <stdio.h>
#include <Windows.h>
```

```
#include "resource.h"
BOOL CALLBACK DialogFun(HWND hwnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    char Text[64];
    double x, y, z;
    switch(message)
    {
    case WM_COMMAND:
       switch(LOWORD(wParam))
       {
        case IDOK:
        case IDCANCEL:
           EndDialog(hwnd, LOWORD(wParam));
           return TRUE;
    case IDC_BUTTON1: // Обработка сообщения от кнопки +.
    case IDC_BUTTON2: // Обработка сообщения от кнопки -.
           SendDlgItemMessage(hwnd, IDC_EDIT1, WM_GETTEXT,
63, (LPARAM)Text); /* Получить текст из текстового поля и
записать в массив Text */
           if (sscanf_s(Text, "%lf", &x)<1)
            {
    MessageBox(hwnd, "Неверный формат первого операнда",
  "Ошибка формата", MB_OK | MB_ICONHAND);
                        return TRUE;
                  }
                  SendDlgItemMessage(hwnd, IDC_EDIT2,
WM_GETTEXT, 63, (LPARAM)Text); /* Получить текст из тек-
стового поля и записать в массив Text */
                  if (sscanf_s(Text, "%lf", &y)<1)
                  {
                        MessageBox(hwnd, "Неверный формат\
второго операнда", "Ошибка формата", MB_OK | MB_ICONHAND);
                        return TRUE;
                  }
                  char Znak;
                  if (LOWORD(wParam)==IDC_BUTTON1) {
                  z=x+y; Znak='+'; }
                  if (LOWORD(wParam)==IDC_BUTTON2) {
                  z=x-y;  Znak='-'; }
                  sprintf(Text, "%f", z);
```

```
                    SendDlgItemMessage(hwnd, IDC_EDIT3,
WM_SETTEXT, 0, (LPARAM)Text); /* Помещаем текст из масси-
ва в текстовое поле*/
                    sprintf_s(Text, "%f %c %f = %f", x,
Znak, y, z);
                    SendDlgItemMessage(hwnd, IDC_LIST1,
LB_ADDSTRING, 0, (LPARAM)Text);
                    return TRUE;
               case IDC_BUTTON3: /* Обработка сообщения от
                    кнопки "Очистить список"*/
                    SendDlgItemMessage(hwnd, IDC_LIST1,
LB_RESETCONTENT, 0, 0);
                    return TRUE;
               }
          }
          return FALSE;
     }
     int APIENTRY WinMain(HINSTANCE hInstance,
                          HINSTANCE hPrevInstance,
                          LPSTR     lpCmdLine,
                          int       nCmdShow)
     {
          // TODO: Place code here.
```

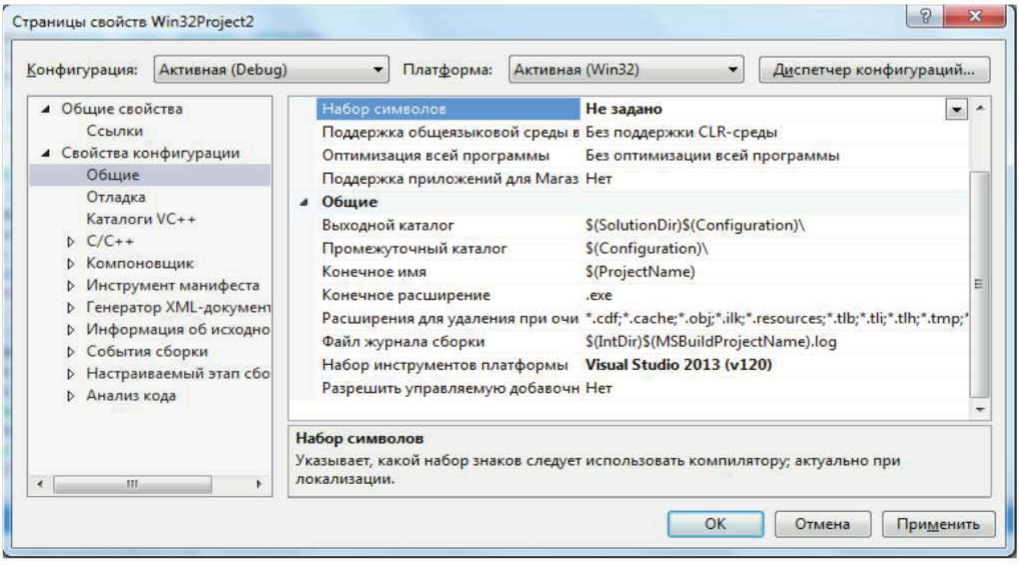


Рис.1.12. Отключение набора символов Unicode

```
        DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), 0,
DialogFun);
        return 0;
    }
```

Следует отметить, что для запуска данной программы необходимо в окне свойств проекта отключить использование кодировки Unicode, т. е. установить опцию «Набор символов» в состояние «Не задано», как показано на рис. 1.12.

Задания для самоконтроля

1. Раскройте особенности модальных диалоговых окон и их отличия от немодальных.
2. Перечислите основные шаги создания диалогового окна в Win API.
3. Перечислите основные элементы управления в диалоговом окне.
4. Расскажите, как обрабатываются сообщения от элементов управления в диалоговом окне.
5. Назовите функции Win API для изменения состояний элементов управления с помощью механизма отправки сообщений этим элементам.

Часть 2
РЕШЕНИЕ ЗАДАЧ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ++ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Лабораторная работа № 2.1
Изучение классов языка Си++

2.1.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в изучении основных понятий объектно-ориентированного языка программирования Си++ — классов и объектов и в овладении навыками разработки программ на языке Си++ с использованием объектно-ориентированных средств. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные основам объектно-ориентированного программирования на языке Си++ [3, 7];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программу;
- выполнить решение контрольного примера с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

2.1.2. Краткая характеристика объекта изучения

Понятие класса и объекта

Класс в языке Си++ — новый тип, определяемый программистом, включающий в себя данные (поля класса) и методы (функции) для их обработки. Переменные такого типа называются объектами [3, 7].

Формат объявления класса:

```
<Ключевое слово> <Имя_класса>
```

```
{
            <список компонент>
};
```

В качестве ключевого используется одно из трех слов:

```
struct     class     union.
```

Можно дать определение класса через структуру, которая была в языке Си: *класс — это структура, в которую введены методы для обработки полей.*

Объекты — это переменные типа класса. Формат определения объектов:

```
<Имя_класса> <Имя_объекта1>,…<Имя_объекта_N>;
```

Обращение к полям и методам класса внутри методов класса — просто по имени, а за пределами класса — через имя объекта и операцию «.» или через имя указателя на объект и операцию «–>». Каждый объект класса имеет в оперативной памяти свои копии полей класса.

Пример работы с полями

```
struct A {int i; void print() {printf("i=%d", i); }    };
A a1;   A *pA=&a1;
a1.i=10; a1.print();          pA->i=10; pA->print();
a1.A::i=10; a1.A::print(); pA->A::i=10; pA->A::print();
```

Доступность компонент класса

Свойство доступности определяет возможность доступа к полям и методам за пределами класса (через имя объекта или указатель на объект).

Существует три статуса доступа:

- `public` (полностью доступны за пределами класса);
- `private` (недоступны за пределами класса, можно обращаться к компонентам только в методах своего класса);
- `protected` (доступны только в своем классе и в производных классах).

По умолчанию: если класс определен с ключевым словом `struct`, то все компоненты имеют статус доступа `public`, если со словом `union` — тоже `public`, но все поля каждого объекта располагаются

в памяти, начиная с одного адреса. Если класс определен с ключевым словом `class`, то все поля и методы по умолчанию имеют статус доступа `private`.

Статус доступа можно изменить с помощью соответствующих модификаторов, что продемонстрировано в следующем примере:

```
struct A
{
  ... // Статус доступа public
private:
... // Статус доступа private
protected:
... // Статус доступа protected
};
class B
{
... // Статус доступа private
public:
... // Статус доступа public
protected:
... // Статус доступа protected
};
```

Основные элементы класса

Компонентные данные и функции класса. Компонентные данные (поля класса) и функции класса (методы класса) уже во многом рассмотрены выше. Следует дополнить, что методы класса могут быть определены внутри класса — в этом случае они, если нет ограничений, являются подставляемыми, но чаще всего класс содержит описание (заголовки) методов, а определения методов находятся за пределами класса. Эта возможность удобна в проектах, состоящих из многих файлов. Создается отдельный файл с расширением `.h` (заголовочный файл), в котором находится описание класса вместе с полями и заголовками методов. Определения методов находятся в файле реализации класса (файле с расширением `.cpp`). В этом случае, чтобы использовать класс в другом файле с исходным кодом, необходимо подключить заголовочный файл с описанием класса. Также внутри методов можно использовать умалчиваемые значения параметров (требования такие же, как к обычным функциям). В классах возможна перегрузка методов.

В следующем примере показаны перечисленные возможности:

```
struct Complex
{
    double real,  image; // Поля класса.
    void define(double re=0.0, double im=0.0) /* Определение метода внутри класса */
    {
    real=re; image=im;// Обращение к полям внутри метода.
    }
    void print(); // Описание метода.
};
void Complex::print() /* Определение метода за пределами класса */
    {
          printf("\nreal=%f   image=%f", real, image);
    }
```

Конструктор класса. Конструктор класса — специальный блок операторов (инструкций), вызываемый при создании объекта. Назначение: присвоение начальных значений полям, выделение памяти, открытие файлов, сетевых соединений и т. п. Имя конструктора совпадает с именем класса, конструктор не имеет возвращаемого значения. Возможна перегрузка конструкторов. Конструктор может определяться как внутри класса, так и за его пределами.

Формат определения конструктора внутри класса:

```
Имя_класса(Список_формальных_параметров)
{   Операторы_тела конструктора      }
```

По умолчанию, класс всегда имеет конструктор копирования вида *A*(*A*& *a*) {... } (*A* – имя класса), создающий копию объекта (происходит копирование полей), и если нет явного конструктора, то, по умолчанию, создается конструктор без параметров. Эти конструкторы можно переопределять.

Примеры вызовов конструкторов:

```
A a1;  A* pA=new A; /* Вызываются конструкторы без параметров */
A a2(3, 4); A * pA2=new A(3, 4);
// Вызываются конструкторы с двумя параметрами.
```

Для конструктора с одним параметром можно использовать форму:

```
A a1=5; A a2=a1; вместо A a1(5); A a2(a1);
```

Деструктор класса. Деструктор — специальный блок операторов (инструкций), служащий для деинициализации объекта (освобождения памяти, закрытия файлов и т. п.). Вызывается автоматически при удалении объекта, например, оператором `delete` или при выходе из блока, в котором существует объект. Не имеет возвращаемого значения и параметров. Может определяться как внутри класса, так и за его пределами. Пример деструктора:

```
~имя_класса()
{
      тело_деструктора
}
```

2.1.3. Задачи и порядок выполнения работы

В работе следует разобраться с понятием класса некоторой предметной области и соответствующему ему класса как типа языка Си++, введенного пользователем. Также важно знать отличие классов от объектов, назначение и порядок использования основных элементов класса в языке Си++: полей, методов (функций класса), конструкторов, деструктора. Нужно обратить внимание на способы создания массива объектов класса динамически.

Студент разрабатывает программу на языке Си++ в виде консольного приложения. В программе необходимо создать массив объектов некоторого класса, количество элементов массива заранее неизвестно и вводится с клавиатуры. Параметры каждого объекта также вводятся с клавиатуры. Для небольшого количества объектов студент выполняет ручной расчет в целях проверки работы программы. Результаты работы программы и ручного расчета следует представить преподавателю в отчете.

Условие задачи

Класс «автомобиль». Содержит поля (статус доступа `private`): название (марка); средний расход топлива (в литрах на 100 км).

Методы и конструкторы: конструктор для инициализации полей; метод расчета требуемого объема топлива для заданного пробега (метод

имеет один параметр, который задает требуемый пробег в километрах, возвращает требуемый объем топлива в литрах); метод печати параметров объекта.

При необходимости можно использовать другие методы и конструкторы.

Требуется создать массив объектов класса, рассчитать требуемый суммарный объем топлива для заданного пробега всех автомобилей (пробег вводится с клавиатуры), вывести на печать параметры объектов и рассчитанный объем топлива.

Пример выполнения работы

Отметим, что для создания массива объектов динамически, когда существует конструктор с параметрами для инициализации полей объекта, можно использовать два основных способа.

Особенности первого способа:

- в класс нужно включить конструктор без параметров и отдельный метод для инициализации полей, например, с именем *set* и количеством параметров, которые соответствуют количеству инициализируемых полей класса;
- объявить указатель на массив объектов; если имя класса `MyClass`, это объявление может иметь вид `MyClass *pOb;`
- задать количество объектов, например, ввести с клавиатуры как значение целой переменой `n` и создать массив объектов динамически `pOb=new MyClass[n];` здесь при создании каждого объекта вызывается конструктор без параметров, именно для этого его и нужно включить в класс;
- в цикле для каждого объекта ввести параметры для его инициализации и инициализировать уже созданный объект с помощью вызова метода `set`, это действие может иметь вид `pOb[i].set(...)`, где вместо многоточия следует указать фактические параметры.

Особенности второго способа (здесь не нужен конструктор без параметров и метод *set*):

- объявить указатель на указатель массива объектов; если имя класса `MyClass`, это объявление может иметь вид: `MyClass **ppOb;`
- задать количество объектов, например, ввести с клавиатуры как значение целой переменой `n` и создать массив указателей динамически `ppOb=new MyClass*[n];`
- в цикле для каждого объекта ввести параметры для его инициализации и создать объект динамически с помощью вызова конструк-

тора с параметрами, это действие может иметь вид `ppOb[i]=new MyClass(...);` , где вместо многоточия передаются фактические параметры для конструктора класса.

Ниже в листинге показаны оба способа, один из способов представлен в виде комментариев. Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application, в русифицированной версии — Консольное приложение Win32). Исходные коды основного файла проекта (файла с расширением `.cpp`) и файла `stdafx.h` приведены ниже.

Листинг программы с комментариями

```
// Основной файл проекта.
#include "stdafx.h"
using namespace std;/* В этом пространстве имен находятся объекты cin и cout */
/* cin — стандартный объект ввода, cout — стандартный объект вывода */
class Avt // Класс «автомобиль».
{
    char marka[64]; // Марка.
    double rash; // Расход топлива на 100 км.
public:
    Avt()   /* Конструктор без параметров создает «пустой» объект */
    {
        rash = marka[0] = 0;
    }
    void set(char mar[], double r) /* Функция для инициализации полей для созданного «пустого» объекта */
    {
        strcpy_s(marka, mar); /* Копирование строки, содержащей марку автомобиля */
        rash = r; /* Задать значение расхода топлива на 100 км */
    }
    Avt(char mar[], double r) /* Конструктор для инициализации полей */
    {
        strcpy_s(marka, mar); /* Копирование строки, содержащей марку автомобиля*/
```

```
            rash = r; /* Задать значение расхода топлива
на 100 км */
      }
      double getRash(double dlina) /* Функция возвращает нуж-
ное количество топлива для пробега заданного расстояния */
      {
      return rash*dlina / 100.;
      }
      void print() // Функция для печати полей объекта.
      {
      cout << "\nmarka: " << marka << "  rashod na 100 km="
<< rash;
      }
  };
  int _tmain(int argc, _TCHAR* argv[])
  {
      int n; // Неизвестное количество объектов.
      cout << "n="; cin >> n;   // Ввод с клавиатуры n
      double rast; /* Расстояние, для которого требуется
расчитать расход топлива */
      cout << "rast="; cin >> rast; /* Ввод с клавиатуры
расстояния */
      double SumRashod = 0; /* Суммарный расход топлива
для всех автомобилей*/
      char S[64]; double r; /* Вспомогательные переменные
для ввода марки автомобиля и расхода топлива */
  /*********************************************************
      Первый способ. Создание массива «пустых» объектов и
инициализация их с помощью функции set
  *********************************************************/
  /*
      Avt *pAvt; // Указатель на массив.
      pAvt=new Avt[n]; /* Для каждого объекта вызов кон-
структора без параметров, т.е. создание «пустых» объектов */
      // Цикл ввода данных для объектов.
      for(int i=0; i<n; i++)
      {
      cout<<"Object N="<<(i+1)<<":\n"<<"marka: ";
      cin>>S;   // Ввод марки автомобиля.
      cout<<"Rashod="; cin>>r;   // Ввод расхода топлива.
```

```
        pAvt[i].set(S, r); /* Вызов функции set для инициализации полей объектов */
        }
        // Цикл печати полей для объектов.
        for(int i=0; i<n; i++) pAvt[i].print();
        // Цикл для расчета суммарного расхода топлива.
        for(int i=0; i<n; i++)
        SumRashod+=pAvt[i].getRash(rast);

    /*********************************************************
        Конец первого способа
    *********************************************************/
    /*********************************************************
    Второй способ. Создание массива указателей на объекты, и далее создание каждого объекта динамически с помощью конструктора с инициализацией
    *********************************************************/
        Avt **ppA; // Указатель на массив указателей.
        ppA = new Avt*[n]; // Создание массива указателей.
        for (int i = 0; i<n; i++)
        {
        cout << "Object N=" << (i + 1) << ":\n" << "marka: ";
              cin >> S; // Ввод марки автомобиля.
              cout << "Rashod="; cin >> r; /* Ввод расхода топлива */
              ppA[i] = new Avt(S, r); /* Создание объекта динамически с вызовом конструктора с параметрами */
        }
        // Цикл печати полей для объектов.
        for (int i = 0; i<n; i++) ppA[i]->print();
        // Цикл для расчета суммарного расхода топлива.
        for (int i = 0; i<n; i++)
        SumRashod += ppA[i]-> getRash(rast);

    /*********************************************************
        Конец второго способа
    *********************************************************/
        cout << «"\nSumRashor=" << SumRashod;  /* Вывод на печать суммарного расхода топлива */
    system("pause"); /* Остановка программы до нажатия любой клавиши */
```

```
    return 0;
}
// Файл stdafx.h
#pragma once
#define WIN32_LEAN_AND_MEAN   /* Exclude rarely-used
stuff from Windows headers*/
#include <tchar.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
```

Задания для самоконтроля

1. Дайте определения классам и объектам в объектно-ориентированном программировании.
2. Перечислите статусы доступа полей и методов класса.
3. Расскажите о назначении конструктора класса.
4. Раскройте понятие перегрузки конструкторов.
5. Расскажите о назначении деструктора класса.

Лабораторная работа № 2.2
Изучение перегрузки стандартных операций в языке Си++

2.2.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих перегрузку стандартных операций. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные перегрузке стандартных операций в языке Си++ [3, 7];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программы;

• выполнить решение контрольного примера с помощью программы и ручной расчет контрольного примера.

Выполнив работу, нужно подготовить отчет.

2.2.2. Краткая характеристика объекта изучения

Перегрузка операций в языке Си++ — это возможность распространения действия стандартных операций на операнды, для которых операции первоначально не предназначались [3, 7]. Такое возможно, если хотя бы один из операндов является объектом класса. Для этого создается специальная так называемая оператор-функция, которая может быть как членом класса, так и функцией, не принадлежащей классу.

Формат определения оператор-функции имеет вид:

```
<тип_возвращаемого_значения>
operator <знак_операции>
(спецификация_параметров)
{
    операторы_тела_функции
}
```

Существует три способа перегрузки. Оператор-функция определяется как функция:

- не принадлежащая классу;
- принадлежащая классу;
- дружественная классу.

Особенности перегрузки операций. Можно перегружать только стандартные операции. Например, нельзя перегрузить операцию `'**'` (возведение в степень в языке Фортран — отсутствует в языке Си++). Не допускают перегрузки операции: `'.'`, `'.*'`, `'?:'`, `'::'`, `'sizeof'`, `'#'`, `'##'`. При перегрузке сохраняется арность операций (унарная операция остается унарной, а бинарная — бинарной). Бинарная операция перегружается либо как функция, не принадлежащая классу с двумя параметрами, один из которых обязательно объект (ссылка на объект) класса, либо как функция класса с одним параметром; первым операндом операции выступает объект класса, для которого вызывается функция. Бинарные операции `'='`, `'[]'`, `'->'` должны обязательно определяться как компонентные функции класса. Унарная операция перегружается либо как функция, не принадлежащая классу с одним параметром — объектом (ссылкой на объект) класса, либо как функция

класса без параметров, операндом операции выступает объект класса, для которого вызывается функция.

2.2.3. Задачи и порядок выполнения работы

Студент должен создать в работе класс и необходимые оператор-функции для перегрузки заданных в своем варианте операций. Особое внимание следует обратить на способы перегрузки унарных и бинарных операций и параметры оператор-функций для этих операций, когда оператор-функция является членом класса и когда не является. Знать те случаи, когда оператор-функция должна быть обязательно членом класса и случаи, когда оператор-функция обязательно не принадлежит классу. При защите работы необходимо обосновать выбор способа определения оператора-функции — внутри класса или вне его. При необходимости, студент выполняет расчет вручную, чтобы проверить работу программы для задачи небольшой размерности. Результаты работы программы и этого расчета нужно представить в отчете.

Условие задачи

Дан класс (например, с именем *Vector*), задающий вектор размерности *n*. Поля класса: указатель на массив, задающий вектор (тип элемента *double*), количество элементов (размерность) вектора (тип *int*). Массив нужно создавать динамически. Класс включает в себя: конструктор без параметров, задающий пустой вектор (количество элементов равно 0); конструктор, создающий объект «вектор» на основе обычного одномерного массива размерности *n*; деструктор.

Необходимо перегрузить операции и продемонстрировать их работу. Перегрузить операцию «[]» (обращение к элементу вектора по индексу), операцию «=» (копирование вектора или создание копии вектора), операцию «*» (умножение числа на вектора), на выходе — вектор такой же размерности, каждый элемент которого равен произведению соответствующего элемента исходного вектора на число.

Пример выполнения работы

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application). Исходные коды основного файла проекта (с расширением `.cpp`) и файла `stdafx.h` приведены ниже.

Листинг программы с комментариями

```
/* Project3.cpp: определяет точку входа для консольного
приложения*/
#include "stdafx.h"
class Vector // Класс  "вектор".
{
    double *p; // Указатель на массив (вектор).
    int n; /* Размерность вектора (количество элементов)
массива */
public:
    Vector(double *p, int n) /* Конструктор на входе
массив, задающий вектор */
    {
    this->n = n; // Задать количество элементов.
    this->p = new double[n]; // Выделение памяти.
    for (int i = 0; i<n; i++) this->p[i] = p[i];
// Копирование одного массива в другой
    }
    void print() // Печать вектора (массива).
    {
    cout << "\n";
    for (int i = 0; i<n; i++)
    cout << p[i] << " ";
    }
    Vector() { p = 0; n = 0; } /* Конструктор без па-
раметров, задает «пустой» объект */
    double& operator[](int index) /* Оператор-функция
(перегрузка операции обращения к элементу) */
    {
    return p[index];
    }
    Vector & operator =(Vector& v2) /* Оператор-функция
копирования объекта */
    {
    n = v2.n;
    p = new double[n];
    for (int i = 0; i<n; i++) p[i] = v2.p[i];
    return *this;
    }
    ~Vector() // Деструктор.
```

```
        {
        if (n>0) delete[] p; // Освобождение памяти.
        }
// Дружественная функция, определенная вне класса.
        friend Vector & operator *(double x, Vector& v2);
    };
   /* Умножение числа на вектор (первый операнд — не объ-
ект класса, функция обязательно определяется вне класса)*/
   Vector & operator *(double x, Vector& v2) /* Оператор-
функция вне класса */
   {
    double *p = new double[v2.n]; // Создание нового массива.
    for (int i = 0; i<v2.n; i++) p[i] = x*v2.p[i]; /* За-
полнение массива */
    Vector *pV = new Vector(p, v2.n); /* Создание нового
объекта на основе массива */
        delete[] p; // Освобождение массива.
        return *pV; // Возвращение ссылки на объект.
   }
   int _tmain(int argc, _TCHAR* argv[])
   {
        double m1[] = { 1, 2, 3, 4.5, 7 };
        Vector V1(m1, 5); // Создание объекта.
        V1.print(); // Печать объекта.
        cout << "\n";
        for (int i = 0; i<5; i++)
        cout << V1[i] << " "; // Пример обращения к операции [].
        V1[0] = 10.6; // Пример обращения к операции [].
        V1.print(); // Печать объекта.
        Vector V2; // Новый объект (вначале «пустой»).
        V2 = 100 * V1; /* Пример выполнения перегруженной опе-
рации */
        // V2=operator *(100, V1);
        V2.print(); // Печать полученного объекта.
        system("pause"); /* Остановка программы до нажатия
любой клавиши */
        return 0;
}
   /* stdafx.h: включаемый файл для стандартных системных
включаемых файлов */
   #pragma once
```

```
#include «targetver.h»
#include <stdio.h>
#include <tchar.h>
/* TODO: Установка здесь ссылок на дополнительные заголовки, требующиеся для программы */
#include <stdlib.h>
#include <iostream>
using namespace std;
```

Задания для самоконтроля

1. Назовите способы перегрузки операций в языке Си++.
2. Раскройте особенности перегрузки бинарной операции, когда первый операнд не является объектом класса.
3. Укажите, в каких целях в некоторых случаях оператор-функцию делают дружественной функцией класса.
4. Перечислите операции, для перегрузки которых оператор-функция обязательно должна принадлежать классу.

Лабораторная работа № 2.3
Изучение возможностей наследования классов

2.3.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих возможности наследования классов для решения различных задач. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные наследованию классов в языке Си++ [3, 7];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программы;
- представить результаты работы программы.

Выполнив работу, нужно подготовить отчет.

2.3.2. Краткая характеристика объекта изучения

Общие сведения о наследовании классов

Основная идея, используемая при наследовании классов, заключается в том, что на основе существующего класса (класс-родитель, или базовый класс), создается производный класс (класс-наследник, дочерний класс), который включает в себя поля и функции базового класса (наследует их) и содержит дополнительные поля (обладает новыми свойствами) и функции [3, 7].

Примеры определения производных классов:

```
class S: X, Y, Z
{ ... } ;
class B: public A
{….};
class D: public X, protected B
{….};
```

Статусы доступа при наследовании классов

Перед именем базового класса можно указать статус доступа наследования (одно из ключевых слов `public, protected, private`). Статус доступа наследования определяет статус доступа наследуемых полей и функций из базового класса внутри производного класса.

Производный класс может определяться с ключевым словом `struct` или `class` (с ключевым словом `union` производный класс не определяется). Если производный класс определен с ключевым словом `class`, то по умолчанию статус доступа наследования `private`.

Таблица 2.1

Статусы доступа при наследовании классов

Тип наследования доступа	Доступность в базовом классе	Доступность компонент базового класса в производном
public	public	public
	protected	protected

Окончание табл. 2.1

Тип наследования доступа	Доступность в базовом классе	Доступность компонент базового класса в производном
	private	недоступны
protected	public	protected
	protected	protected
	private	недоступны
private	public	private
	protected	private
	private	недоступны

Если производный класс определен с ключевым словом `struct`, то по умолчанию статус доступа наследования `public`.

Статусы доступа при наследовании классов представлены в табл. 2.1.

Особенности конструкторов при наследовании

Конструктор производного класса в первую очередь должен вызывать конструктор базового класса. Если это действие не выполняется явно, то, по умолчанию, вызывается конструктор без параметров (если он есть, если его нет, будет ошибка). Если класс имеет несколько базовых, то конструкторы базовых классов должны вызываться в порядке перечисления этих классов в списке базовых.

Особенности деструкторов при наследовании

Деструктор производного класса всегда неявно, по умолчанию после выполнения своего тела вызывает деструкторы базовых классов, причем порядок разрушения объекта (вызовов деструкторов) обратен порядку создания (вызова конструкторов).

Переопределение функций. Виртуальные функции

Если в производном классе объявлена функция с именем, типом возвращаемого значения и количеством и типами параметров, такая же, как в базовом классе, то данная функция является переопределенной. (Нельзя путать с перегрузкой функций!) С помощью простых

переопределенных функций реализуется механизм статического полиморфизма. (Полиморфизм — возможность функции в производном классе работать по-другому.)

Суть статического связывания: когда указатель одного типа ссылается на объект другого типа при наследовании классов, выбор переопределенного метода определяется типом указателя, а не типом объекта.

Суть динамического связывания: когда указатель одного типа ссылается на объект другого типа при наследовании классов, выбор переопределенного метода определяется типом объекта, а не типом указателя, для этого переопределенный метод должен быть объявлен виртуальным в базовом классе. С помощью динамического связывания реализуется механизм динамического полиморфизма.

Динамический полиморфизм при переопределении метода в производном классе обеспечивается объявлением заголовка метода с ключевым словом `virtual` в базовом классе. Встретив у функции модификатор `virtual`, компилятор создает для класса таблицу виртуальных функций, а в класс добавляет новый скрытый для программиста член — указатель на эту таблицу.

Таблица виртуальных функций хранит в себе адреса всех виртуальных методов класса (по сути, это массив указателей), а также всех виртуальных методов базовых классов этого класса. За счет этих указателей на функции обеспечивается динамическое связывание: при вызове метода через указатель, который имеет тип базового класса, но содержит адрес объекта производного класса (такое преобразование допустимо и выполняется неявно), вызывается именно метод производного класса (вызов метода определяется типом объекта).

2.3.3. Задачи и порядок выполнения работы

В работе первоначально необходимо создать базовый класс и на его основе создать производный класс. При этом необходимо использовать вызов конструктора базового класса внутри конструктора производного класса, а также возможности переопределения методов (функций) класса. Следует разобраться с понятием виртуального метода, а также статическим и динамическим полиморфизмом [3, 7]. Продемонстрировать данные возможности в выполняемом примере.

Условие задачи

Создать базовый класс «точка на плоскости». Элементы класса: поля, задающие координаты точки; конструктор для инициализации

полей; функция для печати значений полей. Создать производный класс «точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Пример выполнения работы

Для решения задачи в среде Microsoft Visual Studio 2013 было создано стандартное консольное приложение (проект типа Win32 Console Application) с установленным свойством «Пустой проект» (Empty project). В проект добавлен файл с расширением .cpp, исходный код которого приведен ниже.

Листинг программы с комментариями

```
#include <stdlib.h>
#include <iostream>
using namespace std;
class point // Базовый класс  «Точка на плоскости».
{
    double x, y; // Координаты точки.
public:
    point(double x, double y) /* Конструктор для инициализации полей */
    {
        this->x=x; this->y=y;
    }
    virtual void print() /* Метод для печати полей (виртуальный) */
    {
        cout<<"\nx="<<x<<" y="<<y; /* Печать значения полей */
    }
};
class point3d: public point /* Производный класс «Точка в пространстве»*/
```

```
{
    double z; // Новое поле — координата z.
public:
    point3d(double x, double y, double z): // Конструктор.
    point(x, y) // Явный вызов конструктора базового класса.
       {
      this->z=z;
       }
       void print() // Переопределенный метод print.
       {
      point::print(); /* Вызов метода базового класса в
переопределенном методе */
                cout<<" z="<<z; // Допечатывание поля z.
       }
};
int main(int argc, char* argv[])
{
point p1(1, 2); // Создание объекта с вызовом конструктора.
point3d p3(3, 4, 5); /* Создание объекта с вызовом кон-
структора */
point *pp; // Указатель типа базового класса.
pp=&p1; // Настройка указателя на объект базового класса.
pp->print(); // Вызов метода через указатель.
pp=&p3; /* Настройка указателя на объект производного класса
          (преобразование типа допустимо) */
pp->print(); /* Вызов метода через указатель, вызывает-
ся метод класса point3d. Если метод print в классе point
был объявлен без virtual, то вызывался бы метод print класс
point */
system("pause"); /* Остановка программы до нажатия любой
клавиши */
    return 0;
}
```

Задания и вопросы для самоконтроля

1. Раскройте понятие производного класса (класса-наследника) в объектно-ориентированном программировании,

2. Перечислите статусы доступа наследуемых полей и методов в производных классах. Как они определяются?

3. Раскройте особенности конструкторов в производных классах.

4. Раскройте особенности деструкторов в производных классах.
5. Дайте понятие переопределения функции в производном классе.
6. Раскройте понятие виртуальной функции.
7. Что такое статическое и динамическое связывание в языке Си++?

Лабораторная работа № 2.4
Изучение абстрактных классов

2.4.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++ с использованием абстрактных классов при наследовании. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные абстрактным классам в языке Си++ [3, 7];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программы;
- представить скриншот окна с результатами работы программы.

Выполнив работу, нужно подготовить отчет.

2.4.2. Краткая характеристика объекта изучения

Абстрактный класс — это класс, который имеет в своем составе хотя бы одну чистую виртуальную функцию [3, 7]. Такая функция не имеет тела и ничего не делает.

Формат объявления чистой виртуальной функции внутри класса:

```
virtual <тип_возвр_значения>
<имя_функции>(<список_форм_парам>)=0;
```

Пример объявления такой функции с именем `MyFun` и двумя параметрами типа `double` (имена параметров в заголовке необязательны):

```
virtual void MyFun(double, double)=0;
```

Нельзя создать объект абстрактного класса (тип указателя на абстрактный класс может быть). Абстрактный класс нужен, чтобы на его основе создавать обычные классы, являющиеся его «наследниками», в которых чистые виртуальные функции переопределяются или заменяются на обычные функции.

2.4.3. Задачи и порядок выполнения работы

В работе требуется создать абстрактный класс «геометрическая фигура на экране», который содержит чистую виртуальную функцию для рисования фигуры (неизвестно заранее, какая именно фигура). На основе этого абстрактного класса нужно создать производные классы, определяющие конкретные геометрические фигуры. При работе с объектами этих классов важно использовать массив указателей, имеющих тип абстрактного класса, что позволит для работы с объектами разных классов использовать один цикл.

Условие задачи

Создать абстрактный класс — «Геометрическая фигура» (на экране). Класс содержит координаты геометрического центра фигуры на экране и следующие поля: поле, задающее размер фигуры (например, расстояние от центра до вершины или радиус окружности, в пикселях); поле, задающее угловое положение фигуры; поле, задающее угловую скорость вращения фигуры; поле, определяющее направление движения (возможны два варианта: движение по вертикали и движение по горизонтали); поле, определяющее скорость движения; поле, определяющее цвет фигуры; поле, содержащее хэндл окна для рисования. При необходимости можно включить другие поля. Класс включает в себя: конструктор для инициализации полей, функцию, изменяющую угловое положение фигуры и положение на экране во время движения за один такт времени, и чистую виртуальную функцию (или функции) для рисования и стирания фигуры на экране.

На основе абстрактного класса «Фигура» следует разработать программу, содержащую описание трех графических классов: «Правильный многоугольник»; «Отрезок» («Линия»); «Половина окружности»; создать несколько объектов каждого из трех классов (не менее трех), для чего использовать один массив указателей типа базового класса «Фигура». Реализуя механизм полиморфизма, привести объекты классов в одновременное вращение вокруг их центров с различными угловыми скоростями и в движение с отскоком от краев окна в заданном режиме (по горизонтали или по вертикали). При этом использовать обработку

сообщений от таймера. Таймер периодически с интервалом несколько миллисекунд генерирует сообщение, а при его обработке стирается старая фигура и рисуется новая на новом месте.

Возможный внешний вид окна приложения с движущимися фигурами представлен на рис. 2.1.

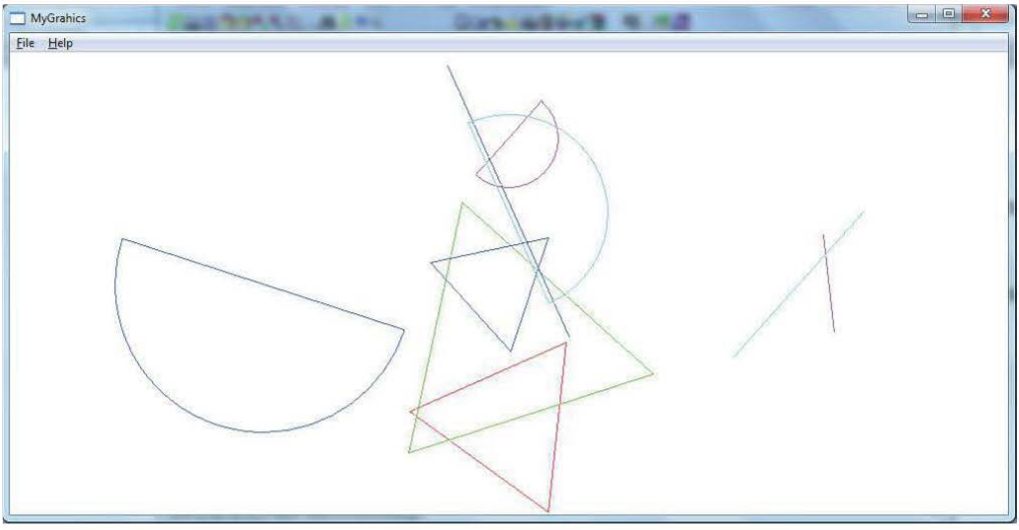


Рис. 2.1. Внешний вид окна приложения

Пример выполнения работы

В работе создается приложение Windows с графическим интерфейсом пользователя (Проект Win32 или Win32 Project для нерусифицированной версии).

В листинге, представленном ниже, приведен только код файла `MyGrahics.cpp`. В среде Microsoft Visual Studio 2013 было создано стандартное оконное приложение (Проект Win32 или Win32 Project для нерусифицированной версии) с именем `MyGrahics` (имя может быть другим), все остальные файлы — `stdafx.h`, `MyGrahics.h` и др. — стандартные, сгенерированы автоматически при создании проекта. В файл `stdafx.h` необходимо в конце добавить две строчки:

```
/* TODO: Установите здесь ссылки на дополнительные заголовки, требующиеся для программы */
#define _USE_MATH_DEFINES
#include <math.h>
```

Листинг файла MyGraphics.cpp с комментариями

```
// MyGraphics.cpp: определяет точку входа для приложения.
#include "stdafx.h"
#include "MyGraphics.h"
#define MAX_LOADSTRING 100
// Глобальные переменные:
HINSTANCE hInst;// Текущий экземпляр.
TCHAR szTitle[MAX_LOADSTRING];// Текст строки заголовка.
TCHAR szWindowClass[MAX_LOADSTRING]; /* Имя класса главного окна*/
// Заголовки функций.
ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
int APIENTRY _tWinMain(_In_ HINSTANCE hInstance,
                       _In_opt_ HINSTANCE hPrevInstance,
                            _In_ LPTSTR     lpCmdLine,
                            _In_ int        nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    // TODO: разместите код здесь.
    MSG msg;
    HACCEL hAccelTable;

    // Инициализация глобальных строк.
    LoadString(hInstance, IDS_APP_TITLE, szTitle,
MAX_LOADSTRING);
    LoadString(hInstance, IDC_MYGRAPHICS, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // Выполнить инициализацию приложения:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
  hAccelTable = LoadAcc elerators(hInstance, MAKEINTRESOURCE
(IDC_MYGRAPHICS));
```

```
        // Цикл основного сообщения:
        while (GetMessage(&msg, NULL, 0, 0))
        {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }
        }
        return (int) msg.wParam;
    }
    //
    //   ФУНКЦИЯ: MyRegisterClass()
    //
    //   НАЗНАЧЕНИЕ: регистрация класса окна.

    //
    ATOM MyRegisterClass(HINSTANCE hInstance)
    {
        WNDCLASSEX wcex;
        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc = WndProc;
        wcex.cbClsExtra = 0;
        wcex.cbWndExtra = 0;
        wcex.hInstance = hInstance;
        wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE
(IDI_MYGRAPHICS));
        wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName = MAKEINTRESOURCE(IDC_MYGRAPHICS);
        wcex.lpszClassName = szWindowClass;
        wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE
(IDI_SMALL));
        return RegisterClassEx(&wcex);
    }
    //
    //    ФУНКЦИЯ: InitInstance(HINSTANCE, int)
    //
    /*    НАЗНАЧЕНИЕ: сохранение обработки экземпляра и соз-
дание главного окна */
```

```
//
//    КОММЕНТАРИИ:
//
/* В данной функции дескриптор экземпляра сохраняется в
глобальной переменной, а также создается и выводит-
ся на экран главное окно программы */
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    hInst = hInstance; /* Сохранение дескриптора экзем-
пляра в глобальной переменной */
     hWnd = CreateWindow(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
NULL, NULL, hInstance, NULL);
    if (!hWnd)
    {
        return FALSE;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    return TRUE;
}
// Абстрактный класс «Фигура».
class Figure
{
protected: /* Поля наследуются; должен быть доступ в
производном классе */
    int x, y; // Координаты геометрического центра.
    int R; // Расстояние от центра до вершины.
    int Ang; // Угловое положение в градусах.
    int VAng; /* Угловая скорость (в градусах за интер-
вал срабатывания таймера) */
    int V; /* Скорость (в пикселях за интервал сраба-
тывания таймера) */
    int Napr; /* Направление движения 0 — вертикально,
1 — горизонтально*/
    COLORREF col; // Цвет фигуры.
    HWND hWnd; // Хэндл окна, где нужно рисовать фигуру
    int N_Reg; /* Направление перемещения (1 — вправо
или вниз; -1 — влево или вверх) */
```

```
  public:
      // Конструктор для инициализации всех параметров.
      Figure(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd);
      // Метод меняет положение фигуры за один такт таймера.
      virtual void step();
      /* Чистая виртуальная функция для рисования (стира-
ния) фигуры */
      virtual void draw(int Reg) = 0;
  };
  // Определение конструктора и функций за пределами класса.
  Figure::Figure(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd)
  {
      this->R = R;
      this->VAng = VAng;
      this->V = V;
      this->Napr = Napr;
      this->col = col;
      this->hWnd = hWnd;
      Ang = 0;
      N_Reg = 1; // Координаты увеличиваются.
      RECT rect; // Размеры окна.
      GetClientRect(hWnd, &rect); // Получение размеров окна.
      // Начальное положение фигуры в центре окна.
      x = rect.right / 2;
      y = rect.bottom / 2;
  }
  /* Определение метода, изменяющего положение фигуры за
один такт таймера */
  void Figure::step()
  {
      // Изменение углового положения фигуры.
      Ang += VAng;
      if (Ang >= 360) Ang -= 360;
      // Получение размеров окна.
      RECT rect;
      GetClientRect(hWnd, &rect);
      // Изменение положения центра фигуры.
      if (Napr == 1) // Движение горизонтально изменяется x.
```

```
      {
            x += V*N_Reg;
            if (N_Reg == 1) // Движение вправо.
            {
            if (x + R >= rect.right) /* Достижение правой
границы окна */
            N_Reg = -1; // Изменение направления движения.
            }
            else //  Движение влево.
            {
            if (x - R <= 0) // Достижение левой границы.
            N_Reg = 1; // Изменение направления движения.
            }
      }
      else // Движение вертикально изменяется y.
      {
            y += V*N_Reg;
            if (N_Reg == 1) // Движение вниз.
            {
            if (y + R >= rect.bottom) /* Достижение ниж-
ней границы окна */
            N_Reg = -1; // Изменение направления движения.
            }
            else //  Движение вверх.
            {
            if (y - R <= 0) // Достижение верхней границы.
            N_Reg = 1; // Изменение направления движения.
            }
      }
  }
  // Класс многоугольник.
  class MyPolygon : public Figure
  {
  protected:
      int N; // Количество вершин.
      POINT *p; // Массив координат вершин.
  public:
      // Заголовок конструктора.
      MyPolygon(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd, int N);
      void step(); /* Метод дополнительно считает новые
координаты вершин */
```

```
        void draw(int Reg); // Метод рисования фигуры.
    };
    // Определение конструктора.
    MyPolygon::MyPolygon(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd, int N) :
    // Вызов конструктора базового класса.
    Figure(R, VAng, V, Napr, col, hWnd)
    {
        this->N = N;
        p = new POINT[N]; // Создание массива координат вершин.
        // Расчет координат вершин.
        double A = Ang*M_PI / 180; /* Угол в градусах сле-
дует перевести в радианы */
        double A1 = 2 * M_PI / N; /* Угол между направле-
ниями на соседние вершины из центра фигуры */
        for (int i = 0; i<N; i++, A += A1)
        {
              p[i].x = x + R*cos(A);
              p[i].y = y - R*sin(A);
        }
    }
    void MyPolygon::step() /* Метод дополнительно считает
новые координаты вершин */
    {
        Figure::step(); // Вызов метода базового класса.
        // Расчет координат вершин многоугольника.
        double A = Ang*M_PI / 180; /* Угол в градусах сле-
дует перевести в радианы*/
        double A1 = 2 * M_PI / N; /* Угол между направле-
ниями на соседние вершины из центра фигуры */
        for (int i = 0; i<N; i++, A += A1)
        {
        p[i].x = x + R*cos(A);
        p[i].y = y - R*sin(A);
        }
    }

    void MyPolygon::draw(int Reg) // Метод рисования фигуры.
    {
    HPEN pen;
    if (Reg == 1) // Режим рисования фигуры.
```

```
        pen = CreatePen(PS_SOLID, 1, col);
    else // Режим стирания (белое перо).
        pen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
        HDC hdc;
        // Получение контекста устройства.
        hdc = GetDC(hWnd);
        SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
        MoveToEx(hdc, p[0].x, p[0].y, 0); /* Графический
курсор в первую вершину */
        for (int i = 1; i<N; i++)
              LineTo(hdc, p[i].x, p[i].y);
        LineTo(hdc, p[0].x, p[0].y); /* Соединение первой и
последней вершин */
        ReleaseDC(hWnd, hdc);
        DeleteObject(pen); // Удаление пера.
    }
    // Класс «Отрезок».
    class MyOtrezok : public Figure
    {
    protected:
        int x1, y1, x2, y2; // Координаты концов отрезка.
    public:
        // Заголовок конструктора.
        MyOtrezok(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd);
        void step(); /* Метод дополнительно считает новые
координаты концов отрезка */
        void draw(int Reg); // Метод рисования фигуры.
    };
    // Определение конструктора.
    MyOtrezok::MyOtrezok(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd) :
    // Вызов конструктора базового класса.
    Figure(R, VAng, V, Napr, col, hWnd)
    {
        // Расчет координат вершин.
        double A = Ang*M_PI / 180; /* Угол в градусах сле-
дует перевести в радианы */
        x1 = x + R*cos(A);
        y1 = y - R*sin(A);
```

```
        x2 = x - R*cos(A);
        y2 = y + R*sin(A);
    }
    void MyOtrezok::step() /* Метод дополнительно считает
новые координаты вершин */
    {
        Figure::step(); // Вызов метода базового класса.
        // Расчет координат вершин многоугольника.
        double A = Ang*M_PI / 180; /* Угол в градусах сле-
дует перевести в радианы*/
        x1 = x + R*cos(A);
        y1 = y - R*sin(A);
        x2 = x - R*cos(A);
        y2 = y + R*sin(A);
    }
    void MyOtrezok::draw(int Reg) // Метод рисования фигуры.
    {
        HPEN pen;
        if (Reg == 1) // Режим рисования фигуры.
              pen = CreatePen(PS_SOLID, 1, col);
        else // Режим стирания (белое перо).
              pen = CreatePen(PS_SOLID, 1, RGB(255, 255,
255));
        HDC hdc;
        // Получение контекста устройства.
        hdc = GetDC(hWnd);
        SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
        MoveToEx(hdc, x1, y1, 0); /* Графический курсор в
первую вершину */
        LineTo(hdc, x2, y2); /* Соединение первой вершины с
последней */
        ReleaseDC(hWnd, hdc);
        DeleteObject(pen); // Удаление пера.
    }
    // Класс «Половина круга».
    class MyPoluKrug : public Figure
    {
    public:
        // Заголовок конструктора.
```

```
      MyPoluKrug(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd);
      void draw(int Reg); // Метод рисования фигуры.
   };
   // Определение конструктора.
   MyPoluKrug::MyPoluKrug(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd) :
   // Вызов конструктора базового класса.
   Figure(R, VAng, V, Napr, col, hWnd)
   {
   }
   void MyPoluKrug::draw(int Reg) // Метод рисования фигуры.
   {
   HPEN pen;
   if (Reg == 1) // Режим рисования фигуры.
       pen = CreatePen(PS_SOLID, 1, col);
   else // Режим стирания (белое перо).
       pen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
   HDC hdc;
   // Получаем контекст устройства.
   hdc = GetDC(hWnd);
   SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
   MoveToEx(hdc, x, y, 0); // Помещение курсора в центр круга.
   AngleArc(hdc, x, y, R, Ang, 180); /* Рисование полу-
окружности */
      LineTo(hdc, x, y); /* Соединение полуокружности с
центром */
      ReleaseDC(hWnd, hdc);
      DeleteObject(pen); // Удаление пера.
   }
   Figure *pF[9]; // Будет создано девять объектов.

   //
   //   ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)
   //
   //   НАЗНАЧЕНИЕ:  обработка сообщения в главном окне.
   //
   //   WM_COMMAND — обработка меню приложения.
   //   WM_PAINT —  закраска главного окна.
```

```
   //  WM_DESTROY    — введение сообщения о выходе и воз-
вращении.
   //
   LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
   {
       int wmId, wmEvent;
       PAINTSTRUCT ps;
       HDC hdc;
       switch (message)
       {
       case WM_CREATE: // Создание окна.
       SetTimer(hWnd, 1, 10, 0); /* Таймер с номером 1
срабатывает через 10 мс */
       // Создание объектов.
   pF[0] = new MyPolygon(100, 1, 10, 0, RGB(255, 0, 0), hWnd, 3);
   pF[1] = new MyPolygon(150, 2, 5, 0, RGB(0, 255, 0), hWnd, 3);
   pF[2] = new MyPolygon(70, 3, 3, 1, RGB(0, 0, 255), hWnd, 3);
   pF[3] = new MyOtrezok(50, 4, 2, 1, RGB(255, 0, 255), hWnd);
   pF[4] = new MyOtrezok(100, 2, 1, 1, RGB(0, 255, 255), hWnd);
   pF[5] = new MyOtrezok(150, 1, 2, 0, RGB(0, 0, 255), hWnd);
   pF[6] = new MyPoluKrug(50, 2, 2, 0, RGB(255, 0, 255), hWnd);
   pF[7] = new MyPoluKrug(100, 1, 3, 0, RGB(0, 255, 255), hWnd);
   pF[8] = new MyPoluKrug(150, 3, 4, 1, RGB(0, 0, 255), hWnd);
       break;
       case WM_TIMER: // Сообщение от таймера.
       for (int i = 0; i<9; i++)
       {
       pF[i]->draw(0); // Стирание старой фигуры.
       pF[i]->step(); // Изменение положения фигуры.
       pF[i]->draw(1); // Рисование фигуры на новом месте.
       }
       break;
       case WM_COMMAND:
       wmId     = LOWORD(wParam);
       wmEvent = HIWORD(wParam);
       // Разобрать выбор в меню:
              switch (wmId)
              {
              case IDM_ABOUT:
```

```
        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd,
About);
        break;
                case IDM_EXIT:
        KillTimer(hWnd, 1); // Удаление таймера.
        DestroyWindow(hWnd);
        break;
                default:
        return DefWindowProc(hWnd, message, wParam, lParam);
                }
        break;
        case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        // TODO: добавьте любой код отрисовки...
        EndPaint(hWnd, &ps);
        break;
        case WM_DESTROY:
        KillTimer(hWnd, 1); // Удаление таймера.
        PostQuitMessage(0);
        break;
        default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
    }
    // Обработчик сообщений для окна «О программе».
    INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
        return (INT_PTR)TRUE;
        case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
        {
        EndDialog(hDlg, LOWORD(wParam));
        return (INT_PTR)TRUE;
        }
```

```
        break;
        }
        return  (INT_PTR)FALSE;
    }
```

Задания и вопросы для самоконтроля

1. Раскройте понятие абстрактного класса.
2. Что такое чистая виртуальная функция?
3. Раскройте особенности использования указателей, имеющих тип «указатель на абстрактный класс».

Лабораторная работа № 2.5
Изучение потоковой многозадачности

2.5.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих возможности потоковой многозадачности. Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные потоковой многозадачности в языке Си++ [6, 8];
- разработать программу на языке Си++ для решения заданного варианта задания;
- отладить программы;
- представить скриншот окна с результатами работы программы.

Выполнив работу, нужно подготовить отчет.

2.5.2. Краткая характеристика объекта изучения

Понятие многозадачности в Windows

В Windows существует два вида многозадачности:

- многозадачность, основанная на процессах;
- многозадачность, основанная на потоках (`thread`).

Процесс можно определить как копию (экземпляр) выполняющейся программы. В данном случае копия — понятие статическое, т. е. процесс в Windows — это объект, который не выполняется, а просто «владеет» выделенным ему адресным пространством. Другими словами, процесс — структура в памяти [6].

В адресном пространстве процесса находятся не только код и данные, но и потоки (thread) — выполняющиеся объекты. При запуске процесса автоматически запускается поток (он называется главным). Главный поток может запускать другие «дочерние» потоки.

Потоки могут работать параллельно (одновременно друг с другом) в многопроцессорных системах (в однопроцессорных системах работают «как бы параллельно» за счет временного разделения) с учетом их приоритетов и имеют доступ к ресурсам процесса (приложения).

Понятие потока (thread) как части процесса не следует путать с понятием потока ввода-вывода (stream).

Создание потока с помощью API-функций

Основные функции для работы с потоками имеют следующие заголовки (назначение параметров отмечено в виде комментариев).

Создать поток:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes, /* Дескриптор
защиты */
SIZE_T dwStackSize, /* Начальный размер стека (0 — как
у родительского потока) */
LPTHREAD_START_ROUTINE lpStartAddress, // Функция потока.
LPVOID lpParameter,  // Параметр потока.
DWORD dwCreationFlags,  // Опции создания.
LPDWORD lpThreadId     // Идентификатор потока.
);
```

Потоковая функция, указатель на которую передается в качестве третьего параметра (lpStartAddress) функции CreateThread, вызывается при старте потока и должна иметь следующий заголовок (имя может быть любым):

```
DWORD WINAPI ThreadFun(LPVOID param);
```

В качестве параметра в функцию передается указатель (можно передавать через указатель любые необходимые данные), значение которого определяется в четвертом параметре (`lpParameter`) функции `CreateThread`.

Приостановить поток:

```
DWORD SuspendThread(
HANDLE hThread     // Дескриптор (хэндл) потока.
);
```

Возобновить приостановленный поток:

```
DWORD ResumeThread(
HANDLE hThread    // Дескриптор (хэндл) потока.
);
```

Установить приоритет потока:

```
BOOL SetThreadPriority(
HANDLE hThread, // Дескриптор потока.
int nPriority    // Уровень приоритета потока.
);
```

Возможные значения уровней приоритета (целые константы определены в стандартном заголовочном файле):

```
   THREAD_PRIORITY_LOWEST    THREAD_PRIORITY_BELOW_NORMAL
THREAD_PRIORITY_NORMAL                THREAD_PRIORITY_HIGHEST
THREAD_PRIORITY_ABOVE_NORMAL    THREAD_PRIORITY_ERROR_RETURN
THREAD_PRIORITY_TIME_CRITICAL     THREAD_PRIORITY_IDLE
```

Нормальное завершение потока — это естественный выход из потоковой функции. Также поток можно завершить извне с помощью вызова функции:

```
BOOL TerminateThread(
HANDLE hThread,   // Дескриптор потока.
DWORD dwExitCode // Код завершения для потока.
);
```

Возможностью завершения потока извне следует пользоваться осторожно, так как если поток работает с некоторыми ресурсами (файлами, сетевыми соединениями и др.), то эти ресурсы не будут автоматически освобождены.

Синхронизация потоков

Проблема синхронизации возникает, когда два потока или более пытаются «одновременно» получить доступ к одним и тем же данным (или одному объекту). Объекты синхронизации в операционной системе Windows и их назначение представлены в табл. 2.2.

Таблица 2.2

Объекты синхронизации и их назначение

Синхронизирующий объект	Назначение	Используемые функции Win32 API
Взаимное исключение	Запрет доступа более чем одному потоку к общим ресурсам	CreateMutex WaitForSingleObject WaitForMultipleObjects ReleaseMutex CloseHandle
Критическая секция	Запрет доступа более чем одному потоку к одному фрагменту кода	InitializeCriticalSection EnterCriticalSection LeaveCriticalSection DeleteCriticalSection
Семафор	Ограничение количества потоков, имеющих одновременный доступ к общим ресурсам	CreateSemaphore WaitForSingleObject WaitForMultipleObjects ReleaseSemaphore CloseHandle
Событие	Позволяет потоку передавать сигналы другим потокам	CreateEvent SetEvent PulseEvent ResetEvent WaitForSingleObject WaitForMultipleObjects CloseHandle

Примеры использования объектов синхронизации:

1. Взаимное исключение

```
HANDLE HMutex; /* Объявляется глобально, и все потоки имеют доступ */
HMutex=CreateMutex( // Создание взаимного исключения.
NULL, // Атрибуты прав доступа, по умолчанию.
FALSE, // Взаимное исключение изначально свободно.
NULL); // Не присваивается имя взаимному исключению.
```

Работа с синхронизированным объектом:

```
WaitForSingleObject(HMutex, // Занять взаимное исключение
INFINITE); // Время ожидания бесконечное.
Count++; // Работа с синхронизированным объектом.
ReleaseMutex(HMutex); /* Освобождение взаимного исключения */
```

Удалить взаимное исключение:

```
CloseHandle(HMutex);
```

Рассмотрим функции `WaitForSingleObject` и `WaitForMultipleObjects`. `WaitForSingleObject` имеет заголовок:

```
DWORD WaitForSingleObject( HANDLE hObject, DWORD dwMilliseconds);
```

Когда поток вызывает эту функцию, первый параметр `hObject` идентифицирует объект ядра (объекты «Взаимное исключение», «Семафор» и «Событие» являются объектами ядра Windows [6]), поддерживающий состояния «свободен/занят». Второй параметр `dwMilliseconds` указывает, сколько времени (в миллисекундах) поток готов ждать освобождения объекта, `INFINITE` — означает *ждать без ограничения времени*. При выходе из функции она возвращает значение, которое можно использовать:

`WAIT_OBJECT_0` — объект свободен;
`WAIT_TIMEOUT` — заданное время ожидания (тайм-аут) истекло;
`WAIT_FAILED` — неверное значение параметра.

Функция `WaitForMultipleObjects` аналогична `WaitForSingleObject`, но она позволяет ждать освобождения сразу нескольких объектов или одного из них. Заголовок функции:

```
DWORD WaitForMultipleObjects( DWOHD dwCount, CONST HANDLE*
phObjects, BOOL fWaitAll, DWORD dwMilliseconds);
```

Параметр `dwCount` определяет количество ожидаемых объектов ядра, его значение в пределах от 1 до `MAXIMUM_WAIT_OBJECTS` (в заголовочных файлах Windows оно определено как 64). Параметр `phObject` — указатель на массив объектов ядра.

`WaitForMultipleObjects` приостанавливает поток и заставляет его ждать освобождения либо всех заданных объектов ядра, либо одного из них. Параметр `fWaitAll` это определяет, если он равен `TRUE`, функция не даст потоку возобновить свою работу, пока не освободятся все объекты, если `FALSE` — достаточно освободиться хотя бы одному объекту.

Параметр `dwMilliseconds` идентичен одноименному параметру функции `WaitForSingleObject`.

2. Критическая секция

Это фрагмент программы, защищенный от одновременного выполнения несколькими потоками. Критическую секцию в данный момент может выполнять только один поток.

`InitializeCriticalSection` — функция, которая создает объект под названием «критическая секция». Параметры функции: указатель на структуру CRITICAL_SECTION.

Поля данной структуры используются только внутренними процедурами, их смысл безразличен.

`EnterCriticalSection` — войти в критическую секцию. После выполнения этой функции поток становится владельцем секции. Следующий поток, вызвав данную функцию, будет находиться в состоянии ожидания. Параметр функции такой же, что и в предыдущей функции.

`LeaveCriticalSection` — покинуть критическую секцию. После этого второй поток, который был остановлен функцией `EnterCriticalSection`, станет владельцем критической секции. Параметр функции `LeaveCriticalSection` такой же, как и у предыдущих функций.

`DeleteCriticalSection` — удалить объект «критическая секция». Параметр аналогичен предыдущим.

3. Семафор

Работает по аналогии с взаимным исключением, только доступ к объекту может получить не один поток, а их количество определяется параметром `MaximumCount` (при занятии потоком семафора текущее значение счетчика уменьшается на 1, семафор полностью занят, если значение счетчика равно 0).

```
HANDLE  CreateSemaphore(   // Создание семафора.
LPSECURITY_ATTRIBUTES lpAttr,   // Атрибуты доступа.
LONG InitialCount,    // Начальное значение семафора.
LONG MaximumCount,   // Максимальное значение.
LPCTSTR lpName );    // Имя семафора.
Возможная схема использования:
WaitForSingleObject(HSem, // Занять семафор.
INFINITE); // Время ожидания бесконечное.
Count++; // Работа с синхронизированным объектом.
ReleaseSemaphore(HSem, 1, 0); /* Освободить одно место
семафора*/
```

4. Событие

Событие используется для того, чтобы один поток сообщил другому потоку, ожидающему наступления этого события, что событие произошло. Событием можно назначить любую точку в алгоритме программы, например некоторая подзадача решена, т.е. получены необходимые результаты, являющиеся исходными данными для ожидающего события потока.

```
HANDLE hEvent;
hEvent=CreateEvent(0,   // Создание объекта «Событие».
    false, /* TRUE — событие со сбросом вручную, FALSE
— событие с автосбросом */
    false, // Событие свободное (TRUE), занятое (FALSE).
    0);
     WaitForSingleObject(hEvent,
    INFINITE);   // Программа далее не выполняется.
Когда событие произошло, в другом потоке следует вызвать
функцию
    SetEvent(hEvent);
```

Синхронизация процессов

С помощью объектов синхронизации можно организовать синхронизацию различных процессов. Например, доступ к объекту синхронизации, созданному в одном процессе (приложении), можно получить в другом приложении с помощью вызова функции `OpenОбъект`. (Вместо термина *объект* идет имя объекта синхронизации.) Например, для объекта «Событие» вызов имеет вид:

```
HANDLE OpenEvent(
DWORD dwDesiredAccess, // Флаги доступа.
BOOL bInheritHandle, // Режим наследования.
LPCTSTR lpName // Имя события.
);
Возможные значения флагов доступа:
EVENT_ALL_ACCESS
EVENT_MODIFY_STATE
SYNCHRONIZE
```

Создание потока в языке Си++ с помощью стандартной библиотеки языка C++

Поддержка многопоточности в языке Си++ определена стандартом 2011 г. [8], до этого приходилось использовать платформенно-зависимые средства, как показано выше. Для создания потока необходимо использовать класс `thread` [8], входящий в пространство имен `std`. Ниже представлен листинг программы с комментариями. Программа создана как консольное приложение. В программе в главном потоке (функция `main`) создаются два дочерних потока, в потоковую функцию в качестве параметра передается строка текста (объект класса `string`), это строка печатается 10 раз.

```
// stdafx.h: включаемый файл для стандартных системных файлов
#pragma once
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
/* TODO: Установка здесь ссылок на дополнительные заголовки, требующиеся для программы */
#include <string>
#include <iostream>
#include <thread>
using namespace std;
/* Thread1.cpp: определение точки входа для консольного приложения*/
//
#include "stdafx.h"
void myfun(string str)  /* Потоковая функция может иметь параметры, при необходимости */
```

```
{
    for (int i = 0; i < 10; i++) // Печать строки 10 раз.
    cout << endl << str.data();
}
int _tmain(int argc, _TCHAR* argv[])
{
    thread th1(myfun, "Java"), th2(myfun, "C++");
/* Создание двух объектов, первый параметр конструктора
—указатель на потоковую функцию, следующие параметры пере-
даются в потоковую функцию при необходимости */
    th1.join(); // Ожидание завершения дочернего потока.
    th2.join(); // Ожидание завершения дочернего потока.
    system("pause"); /* Останавка программы до нажатия
любой клавиши */
    return 0;
}
```

Результаты работы программы представлены на рис. 2.2.

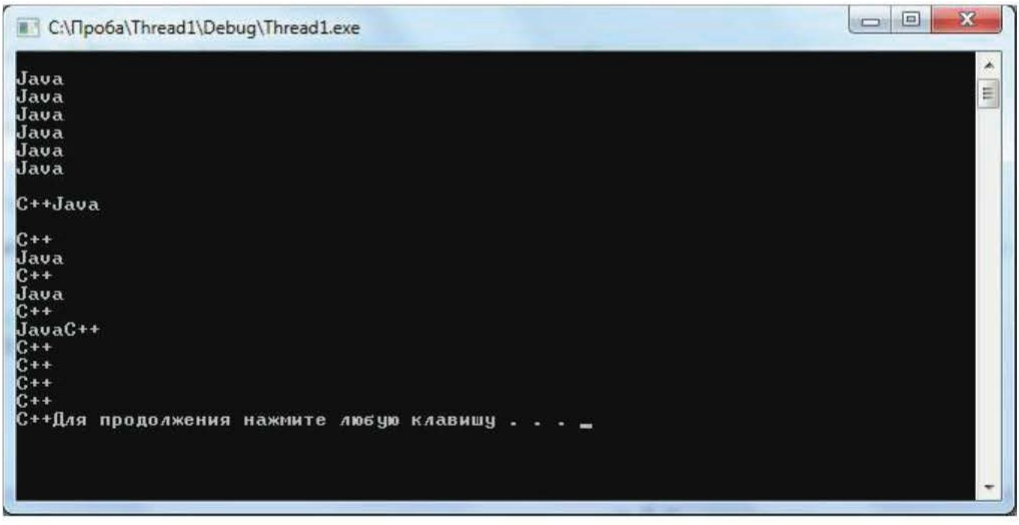


Рис. 2.2. Результаты работы многопоточной программы

На рис. 2.2 в некоторых случаях виден беспорядочный вывод (каждую строку нужно печатать с новой строчки, а это правило иногда нарушается), что связано с отсутствием синхронизации (два потока работают одновременно с одним объектом `cout`).

Для синхронизации можно использовать объект класса `mutex` [8]. Ниже представлена программа с синхронизацией при печати.

```
// stdafx.h: включаемый файл для стандартных системных файлов
#pragma once
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
/* TODO: Установка здесь ссылок на дополнительные заголовки, требующиеся для программы */
#include <string>
#include <iostream>
#include <thread>
#include <mutex>
using namespace std;
/* Thread1.cpp: определение точки входа для консольного приложения */
//
#include "stdafx.h"
mutex mut; // Создание объекта для синхронизации.
void myfun(string str)  /* Потоковая функция может иметь параметры, при необходимости */
{
    for (int i = 0; i < 10; i++) // Печать строки 10 раз.
    {
    mut.lock();   // Блокировка объекта.
    cout << endl << str.data();
    mut.unlock();   // Снятие блокировки.
    }
}
int _tmain(int argc, _TCHAR* argv[])
{
    thread th1(myfun, "Java"), th2(myfun, "C++");
/* Создание двух объектов, первый параметр конструктора — указатель на потоковую функцию, следующие параметры передаются в потоковую функцию, при необходимости */
    th1.join(); // Ожидание завершения дочернего потока.
    th2.join(); // Ожидание завершения дочернего потока.
    system("pause"); /* Остановка программы до нажатия любой клавиши */
    return 0;
}
```

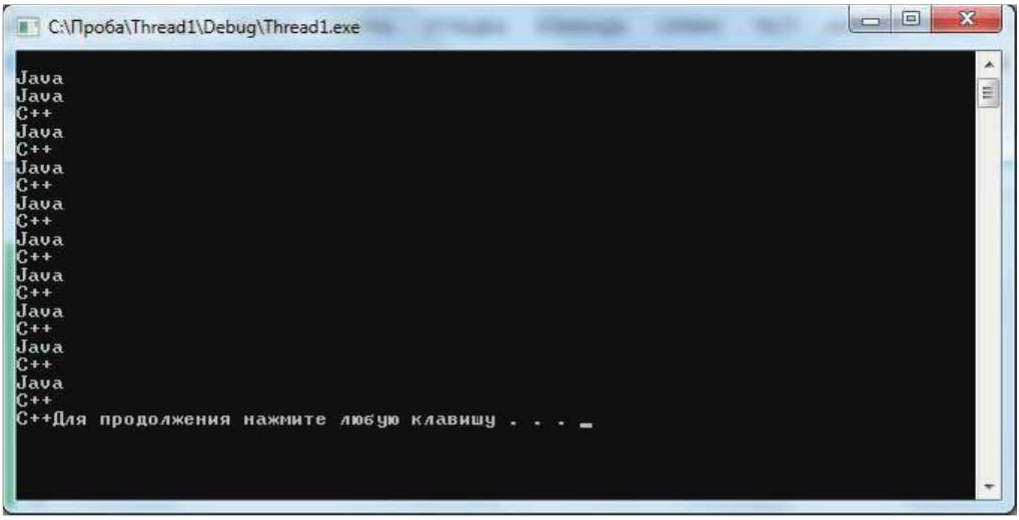


Рис. 2.3. Результаты работы многопоточной программы с синхронизацией

Результаты работы программы с синхронизацией представлены на рис. 2.3.

2.5.3. Задачи и порядок выполнения работы

Работа состоит из двух частей. В первой части разработанное в лабораторной работе 2.4 приложение следует преобразовать таким образом: для обеспечения движения геометрических фигур в окне вместо таймера нужно использовать потоки. Для каждой геометрической фигуры используется отдельный поток. Потоковая функция должна быть одна. В качестве параметра в потоковую функцию передается указатель на объект «Фигура». При необходимости использовать синхронизацию. В примере представлено приложение Windows с графическим интерфейсом пользователя (Проект Win32 или Win32 Project).

Во второй части обеспечивается синхронизация двух приложений.

Первое приложение. Приложение с потоками нужно преобразовать так, чтобы движение фигур в потоках начиналось не сразу, а после получения сигнала от второго приложения. При получения сигнала потоки начинают работать до тех пор, пока от второго приложения не придет другой сигнал, при получении этого сигнала потоки завершают свою работу.

Второе приложение — консольное приложение Windows (запускается только при запущенном первом приложении). После нажатия клавиши посылается сигнал для начала работы потоков в первом прило-

жении. После следующего нажатия клавиши посылается сигнал для завершения работы потоков в первом приложении.

Далее демонстрируется совместная работа двух приложений.

Внешний вид главного окна первого приложения аналогичен окну, представленному на рис. 2.1 (лабораторная работа 2.4).

Условие задачи приведено в лабораторной работе 2.4. Отличие в том, что для анимации каждой фигуры используется свой поток.

Пример выполнения работы

В листинге, представленном ниже, приведен только код файла `MyGrahics.cpp`. В среде Microsoft Visual Studio 2013 было создано стандартное оконное приложение (проект типа Проект Win32 или Win32 Project) с именем `MyGrahics` (имя может быть другим), все остальные файлы — `stdafx.h`, `MyGrahics.h` и др. — стандартные, сгенерированы автоматически при создании проекта. В файл `stdafx.h` в конце добавляют две строчки:

```
#define _USE_MATH_DEFINES
#include <math.h>
```

Листинг программы с комментариями

Часть 1

```
// MyGraphics.cpp: определение точки входа для приложения.
//
#include "stdafx.h"
#include "MyGraphics.h"
#define MAX_LOADSTRING 100
// Глобальные переменные:
HINSTANCE hInst; // Текущий экземпляр.
TCHAR szTitle[MAX_LOADSTRING]; // Текст строки заголовка.
TCHAR szWindowClass[MAX_LOADSTRING]; /* Имя класса главного окна */
// Заголовки функций.
ATOMMyRegisterClass(HINSTANCE hInstance);
BOOLInitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
int APIENTRY _tWinMain(_In_ HINSTANCE hInstance,
                       _In_opt_ HINSTANCE hPrevInstance,
                       _In_ LPTSTR     lpCmdLine,
                       _In_ int        nCmdShow)
```

```
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    // TODO: размещение здесь кода.
    MSG msg;
    HACCEL hAccelTable;
    // Инициализация глобальных строк.
  LoadString(hInstance, IDS_APP_TITLE, szTitle,
MAX_LOADSTRING);
  LoadString(hInstance, IDC_MYGRAPHICS, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // Выполнение инициализации приложения:
    if (!InitInstance (hInstance, nCmdShow))
    {
    return FALSE;
    }
    hAccelTable = LoadAccelerators(hInstance,
  MAKEINTRESOURCE(IDC_MYGRAPHICS));
    // Цикл основного сообщения:
    while (GetMessage(&msg, NULL, 0, 0))
    {
    if (!TranslateAccelerator(msg.hwnd, hAccelTable,
&msg))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }
    }
    return (int) msg.wParam;
}
//
//  ФУНКЦИЯ: MyRegisterClass()
//
//  НАЗНАЧЕНИЕ: регистрация класса окна.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
```

```
        wcex.cbClsExtra = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance = hInstance;
        wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE
(IDI_MYGRAPHICS));
        wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName = MAKEINTRESOURCE(IDC_MYGRAPHICS);
        wcex.lpszClassName = szWindowClass;
        wcex.hIconSm = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));
        return RegisterClassEx(&wcex);
    }
    //
    //    ФУНКЦИЯ: InitInstance(HINSTANCE, int)
    //
    /*    НАЗНАЧЕНИЕ: сохранение обработки экземпляра и соз-
дание главного окна */
    //
    //    КОММЕНТАРИИ:
    //
    /* В данной функции дескриптор экземпляра сохраняется в
глобальной переменной, а также создается и выводит-
ся на экран главное окно программы */
    //
    BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
    {
        HWND hWnd;
        hInst = hInstance; /* Сохранение дескриптора экзем-
пляра в глобальной переменной */
         hWnd = CreateWindow(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
NULL, NULL, hInstance, NULL);
        if (!hWnd)
        {
            return FALSE;
        }
        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);
        return TRUE;
    }
```

```
// Абстрактный класс «Фигура».
class Figure
{
protected: // Обеспечивается доступ в производном классе.
    int x, y; // Координаты геометрического центра.
    int R; // Расстояние от центра до вершины.
    int Ang; // Угловое положение (в градусах).
    int VAng; /* Угловая скорость (в градусах за интер-
вал срабатывания таймера) */
    int V; // Скорость (в пикселях за интервал таймера).
    int Napr; /* Направление движения: 0 — вертикально,
1 — горизонтально*/
    COLORREF col; // Цвет фигуры.
    HWND hWnd; // Хэндл окна, где рисуется фигура.
    int N_Reg; // Направление перемещения
    // (1 — вправо или вниз; -1 — влево или вверх).
    HDC hdc;
public:
    // Конструктор для инициализации всех параметров.
    Figure(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd);
// Метод изменяет положение фигуры за один такт таймера.
    virtual void step();
    /* Чистая виртуальная функция для рисования (стира-
ния) фигуры */
    virtual void draw(int Reg) = 0;
    // Деструктор для освобождения контекста устройства.
    virtual ~Figure();
};
// Определение конструктора и функций за пределами класса.
Figure::Figure(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd)
{
    this->R = R;
    this->VAng = VAng;
    this->V = V;
    this->Napr = Napr;
    this->col = col;
    this->hWnd = hWnd;
    Ang = 0;
    N_Reg = 1; // Увеличение координат.
```

```
        RECT rect; // Размеры окна.
        GetClientRect(hWnd, &rect); // Получение размеров окна.
        // Начальное положение фигуры в центре окна.
        x = rect.right / 2;
        y = rect.bottom / 2;
        // Получение контекста устройства.
        hdc = GetDC(hWnd);
    }
    /* Определение деструктора для освобождения контекста
устройства */
    Figure::~Figure()
    {
        ReleaseDC(hWnd, hdc);
    }
    /* Определение метода, изменяющего положение фигуры */
    void Figure::step()
    {
        // Изменение углового положения фигуры.
        Ang += VAng;
        if (Ang >= 360) Ang -= 360;
        // Получение размеров окна.
        RECT rect;
        GetClientRect(hWnd, &rect);
        // Изменение положения центра фигуры.
        if (Napr == 1) // Движение горизонтально изменяется x.
        {
        x += V*N_Reg;
        if (N_Reg == 1) // Движение вправо.
        {
        if (x + R >= rect.right) /* Достижение правой гра-
ницы окна */
        N_Reg = -1; // Изменение направления движения.
        }
        else //  Движение влево.
        {
        if (x - R <= 0) // Достижение левой границы.
        N_Reg = 1; // Изменение направления движения.
        }
        }
        else // Движение вертикально изменяется y.
```

```
        {
        y += V*N_Reg;
        if (N_Reg == 1) /* Движение вниз.
        {
        if (y + R >= rect.bottom) /* Достижение нижней гра-
ницы окна */
        N_Reg = -1; // Изменение направления движения.
        }
        else //  Движение вверх.
        {
        if (y - R <= 0) // Достижение верхней границы.
        N_Reg = 1; // Изменение направления движения.
        }
        }
    }

    // Класс «Многоугольник».
    class MyPolygon : public Figure
    {
    protected:
        int N; // Количество вершин.
        POINT *p; // Массив координат вершин.
    public:
        // Заголовок конструктора.
        MyPolygon(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd, int N);
        void step(); /* Метод дополнительно считает новые
координаты вершин */
        void draw(int Reg); // Метод рисования фигуры.
    };
    // Определение конструктора.
    MyPolygon::MyPolygon(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd, int N) :
    // Вызов конструктора базового класса.
    Figure(R, VAng, V, Napr, col, hWnd)
    {
        this->N = N;
        p = new POINT[N]; // Создание массива координат вершин.
        // Расчет координат вершин.
```

```
        double A = Ang*M_PI / 180; /* Перевод в радианы
угла в градусах */
        double A1 = 2 * M_PI / N; /* Задается угол между
направлениями на соседние вершины из центра фигуры */
        for (int i = 0; i<N; i++, A += A1)
        {
        p[i].x = x + R*cos(A);
        p[i].y = y - R*sin(A);
        }
    }
    void MyPolygon::step() /* Метод дополнительно считает
новые координаты вершин */
    {
        Figure::step(); // Вызов метода базового класса.
        // Расчет координат вершин многоугольника.
        double A = Ang*M_PI / 180; /* Перевод в радианы
угла в градусах */
        double A1 = 2 * M_PI / N; /* Задается угол между
направлениями на соседние вершины из центра фигуры */
        for (int i = 0; i<N; i++, A += A1)
        {
        p[i].x = x + R*cos(A);
        p[i].y = y - R*sin(A);
        }
    }
    void MyPolygon::draw(int Reg) // Метод рисования фигуры.
    {
        HPEN pen;
        if (Reg == 1) // Режим рисования фигуры.
        pen = CreatePen(PS_SOLID, 1, col);
        else // Режим стирания (белое перо).
        pen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));

        SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
        MoveToEx(hdc, p[0].x, p[0].y, 0); /* Перемещение
графического курсора в первую вершину */
        for (int i = 1; i<N; i++)
        LineTo(hdc, p[i].x, p[i].y);
        LineTo(hdc, p[0].x, p[0].y); /* Соединение последней
вершины с первой */
```

```
        DeleteObject(pen); // Удаление пера.
    }
    // Класс «Отрезок».
    class MyOtrezok : public Figure
    {
    protected:
        int x1, y1, x2, y2; // Координаты концов отрезка.
    public:
        // Заголовок конструктора.
        MyOtrezok(int R, int VAng, int V, int Napr, COLORREF
col, HWND hWnd);
        void step(); /* Метод дополнительно считает новые
координаты концов отрезка */
        void draw(int Reg); // Метод рисования фигуры.
    };
    // Определение конструктора.
    MyOtrezok::MyOtrezok(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd) :
    // Вызов конструктора базового класса.
    Figure(R, VAng, V, Napr, col, hWnd)
    {
        // Расчет координат вершин.
        double A = Ang*M_PI / 180; /* Перевод в радианы
угла в градусах */
        x1 = x + R*cos(A);
        y1 = y - R*sin(A);
        x2 = x - R*cos(A);
        y2 = y + R*sin(A);
    }

    void MyOtrezok::step() /* Метод дополнительно считает
новые координаты вершин */
    {
        Figure::step(); // Вызов метода базового класса.
        // Расчет координат вершин многоугольника.
        double A = Ang*M_PI / 180; /* Перевод в радианы
угла в градусах */
        x1 = x + R*cos(A);
        y1 = y - R*sin(A);
        x2 = x - R*cos(A);
        y2 = y + R*sin(A);
    }
```

```
void MyOtrezok::draw(int Reg) // Метод рисования фигуры.
{
    HPEN pen;
    if (Reg == 1) // Режим рисования фигуры.
    pen = CreatePen(PS_SOLID, 1, col);
    else // Режим стирания (белое перо).
    pen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
    MoveToEx(hdc, x1, y1, 0); /* Помещение графического
курсора в первую вершину */
    LineTo(hdc, x2, y2); /* Соединение последней верши-
ны с первой */
    DeleteObject(pen); // Удаление пера.
}
// Класс «Половина круга».
class MyPoluKrug : public Figure
{
public:
    // Заголовок конструктора.
    MyPoluKrug(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd);
    void draw(int Reg); // Метод рисования фигуры.
};
// Определение конструктора.
MyPoluKrug::MyPoluKrug(int R, int VAng, int V, int Napr,
COLORREF col, HWND hWnd) :
// Вызов конструктора базового класса.
Figure(R, VAng, V, Napr, col, hWnd)
{
}
void MyPoluKrug::draw(int Reg) // Метод рисования фигуры.
{
    HPEN pen;
    if (Reg == 1) // Режим рисования фигуры.
    pen = CreatePen(PS_SOLID, 1, col);
    else // Режим стирания (белое перо).
    pen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    SelectObject(hdc, pen); /* Загрузка пера в контекст
устройства */
```

```
        MoveToEx(hdc, x, y, 0); /* Помещение курсора в центр
круга */
        AngleArc(hdc, x, y, R, Ang, 180); /* Рисование по-
луокружности */
        LineTo(hdc, x, y); /* Соединение полуокружности с
центром */
        DeleteObject(pen); // Удаление пера.
    }
    /* Структура, в которой передаются данные для потоковой
функции */
    struct ForThread
    {
        Figure *pFig; // Указатель на объект «фигура».
        HANDLE hEvent; /* Хэндл события для синхронизации
потоков */
    };
    HANDLE hEvents[9]; /* Хэндлы событий для синхронизации по-
токов */
    Figure *pF[9]; // Будет создано девять объектов.
    ForThread Data[9]; /* Данные для потоковых функций для
каждого из девяти потоков */
    int flag = 1; // Условие работы цикла в потоках.
    /* Потоковая функция, параметр — указатель на структуру
данных ForThread */
    DWORD WINAPI ThreadFun(LPVOID par)
    {
        ForThread *pData = (ForThread *)par; /* Данные, ко-
торые переданы в потоковую функцию */
        while (flag)
        {
        // Рисование фигуры.
        pData->pFig->draw(1); // Рисование фигуры.
        Sleep(20); // Задержка.
        pData->pFig->draw(0); // Стирание старой фигуры.
        pData->pFig->step(); // Изменение положения фигуры.
        }
        SetEvent(pData->hEvent); /* Событие произошло, поток
завершается. */
        return 1;
    }
```

```
   //
   //   ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)
   //
   //   НАЗНАЧЕНИЕ:  обработка сообщения в главном окне.
   LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
   {
       int wmId, wmEvent;
       PAINTSTRUCT ps;
       HDC hdc;
       switch (message)
       {
       case WM_CREATE: // Создание окна.
       // Создание объектов.
       pF[0] = new MyPolygon(100, 1, 10, 0, RGB(255, 0,
0), hWnd, 3);
   pF[1] = new MyPolygon(150, 2, 5, 0, RGB(0, 255, 0), hWnd, 3);
   pF[2] = new MyPolygon(70, 3, 3, 1, RGB(0, 0, 255), hWnd, 3);
   pF[3] = new MyOtrezok(50, 4, 2, 1, RGB(255, 0, 255), hWnd);
   pF[4] = new MyOtrezok(100, 2, 1, 1, RGB(0, 255, 255), hWnd);
   pF[5] = new MyOtrezok(150, 1, 2, 0, RGB(0, 0, 255), hWnd);
   pF[6] = new MyPoluKrug(50, 2, 2, 0, RGB(255, 0, 255), hWnd);
   pF[7] = new MyPoluKrug(100, 1, 3, 0, RGB(0, 255, 255), hWnd);
   pF[8] = new MyPoluKrug(150, 3, 4, 1, RGB(0, 0, 255), hWnd);
       for (int i = 0; i<9; i++)
       {
       // Заполнение данных для потока.
       Data[i].pFig = pF[i]; // Указатель на объект «Фигура».
       Data[i].hEvent = hEvents[i] = CreateEvent(0,   /*
Создать объект  «событие» для синхронизации потоков */
       false, // true — событие со сбросом вручную.
       // false — событие с автосбросом.
       false, // свободное (true) занятое (false).
       0);
       CreateThread(0, 0, ThreadFun, &Data[i], 0, 0); /*
Создание потоков */
       }
       break;
       case WM_COMMAND:
       wmId    = LOWORD(wParam);
       wmEvent = HIWORD(wParam);
```

```
        // Разобрать выбор в меню:
        switch (wmId)
        {
        case IDM_ABOUT:
        DialogBox(hInst,
   MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
        break;
        case IDM_EXIT:
        flag = 0; /* Завершение всех потоков (выход из цик-
ла и из потоковой функции). Ожидание завершения всех по-
токов (наступления события) */
        WaitForMultipleObjects(9, hEvents, TRUE, INFINITE);
        for (int i = 0; i<9; i++) {
        delete pF[i]; // Удаление всех объектов (фигур).
        CloseHandle(hEvents[i]); // Закрытие событий.
        }
        DestroyWindow(hWnd);
        break;
        default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
   case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        // TODO: добавление любого кода отрисовки...
        EndPaint(hWnd, &ps);
        break;
        case WM_DESTROY:
              flag = 0; /* Завершение всех потоков (выход
из цикла и из потоковой функции). Ожидание завершения всех
потоков (наступления события)*/
        WaitForMultipleObjects(9, hEvents, TRUE, INFINITE);
        for (int i = 0; i<9; i++) {
        delete pF[i]; // Удаление всех объектов (фигур).
        CloseHandle(hEvents[i]); // Закрытие событий.
        }
              PostQuitMessage(0);
              break;
        default:
```

```
        return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
    }
    // Обработчик сообщений для окна «О программе».
    INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
        return (INT_PTR)TRUE;
        case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
         {
              EndDialog(hDlg, LOWORD(wParam));
              return (INT_PTR)TRUE;
              }
              break;
        }
        return (INT_PTR)FALSE;
    }
```

В части 2 необходимо обеспечить синхронизацию двух приложений. Первое приложение должно быть создано на основе приложения, исходный код которого приведен выше, укажем только необходимые изменения исходного кода. Нужно объявить глобально до потоковой функции и до функции окна хэндл объекта события для синхронизации приложений.

Листинг программы с комментариями

Часть 2

```
    /* Для синхронизации второго приложения используем объ-
ект «событие» */
    HANDLE hEvent; /* Объявляется глобально, и все потоки
имеют доступ */
```

В функции окна при обработке сообщения WM_CREATE создаем объект «Событие»:

```
   case WM_CREATE: // Создание окна.
       hEvent=CreateEvent(0, // Создание объекта «Событие».
       true, // TRUE — событие со сбросом вручную.
       false, // занятое (FALSE).
       L"MyEvent1"); /* Имя события для открытия в другом
приложении */
       // Создание объектов.
   pF[0]=new MyPolygon(100, 1, 10, 0, RGB(255, 0, 0), hWnd, 3);
   pF[1]=new MyPolygon(150, 2, 5, 0, RGB(0, 255, 0), hWnd, 3);
   pF[2]=new MyPolygon(70, 3, 3, 1, RGB(0, 0, 255), hWnd, 3);
               ...
```

Потоковая функция примет вид:

```
   /* Потоковая функция, параметр — указатель на структуру
данных ForThread */
   DWORD WINAPI ThreadFun(LPVOID par)
   {
       ForThread *pData=(ForThread *)par; /* Данные, кото-
рые переданы в потоковую функцию */
       WaitForSingleObject(hEvent, // Ожидание события.
       INFINITE); /* Ожидание до тех пор, пока другое при-
ложение не пошлет сигнал о том, что событие произошло */
       while(WaitForSingleObject(hEvent, 0)==WAIT_OBJECT_0)
/* Цикл работает пока объект «Событие» свободен */
       {
       // Рисование фигуры.
   pData->pFig->draw(1); // Рисование фигуры.
   Sleep(20); // Задержка.
   pData->pFig->draw(0); // Стирание старой фигуры.
   pData->pFig->step(); // Изменение положения фигуры.
       }
   SetEvent(pData->hEvent); /* Сигнал «Событие о завершении
потока» */
       return 1;
   }
```

Внутри потоковой функции поток останавливается и ожидает наступления события, которое будет установлено в другом приложении. Когда наступит событие, начинает работать цикл для анимации фигуры, условие продолжения цикла — пока объект «Событие» свободен.

Когда объект «Событие» будет занят в другом приложении, цикл в потоковой функции завершается и осуществляется выход из потоковой функции.

Исходный код второго приложения:

```
#include <Windows.h>
#include <conio.h>
#include <iostream>
#include <locale.h>
using namespace std;
int main()
{
    HANDLE hEvent; /* Объявляется глобально, и все потоки имеют доступ */
    setlocale(LC_ALL, "rus");
    hEvent=OpenEvent( /* Открытие объекта «Событие», созданного в другом приложении */
            EVENT_ALL_ACCESS, // Флаги доступа.
            TRUE, // Режим наследования.
            L"MyEvent1" // Имя события.
    );
    if (hEvent==0)
    {
    cout<< "Ошибка открытия объекта \"Событие \", \
созданного в другом приложении";
    _getch();
    return 0;
    }
    cout<< "Для запуска потоков в другом приложении \
нажмите любую клавишу" <<endl;
    _getch();
    SetEvent(hEvent); /* Объект «Событие» свободен (событие произошло) */
    cout<< "Для завершения потоков в другом приложении\
нажмите любую клавишу" <<endl;
    _getch();
    ResetEvent(hEvent); // Объект «Событие» занят.
    cout<< "Для выхода из приложения нажмите любую \
клавишу" <<endl;
    _getch();
    return 0;
}
```

Задания и вопросы для самоконтроля

1. Раскройте суть понятий потока и потоковой многозадачности.
2. Перечислите способы создания потока.
3. Что такое потоковая функция?
4. Раскройте понятие синхронизации потоков.
5. Перечислите объекты синхронизации в Win API.
6. Раскройте особенности синхронизации процессов в Windows.
7. Назовите класс, используемый для создания потока в стандартной библиотеке языка Си++.

Лабораторная работа № 2.6
Изучение библиотеки классов MFC

2.6.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки графического интерфейса пользователя с использованием классов библиотеки MFC (Microsoft Foundation Classes). Для достижения цели необходимо выполнить следующие задачи:

- изучить учебные материалы, посвященные библиотеке классов MFC [9];
- разработать программу на языке Си++ для решения заданного варианта;
- отладить программы;
- представить скриншот окна с результатами работы программы.

Выполнив работу, нужно подготовить отчет.

2.6.2. Краткая характеристика объекта изучения

Обзор упрощенной иерархии классов библиотеки MFC

Библиотека классов MFC (первоначальное название Application Framework eXtensions — AFX) используется для программирования графического интерфейса пользователя и решения других задач вместо API-функций Windows.

Для использования библиотеки подключается заголовочный файл `afxwin.h`.

Упрощенная иерархия некоторых классов библиотеки:

```
CObject
  — CWinThread // Архитектура приложения.
    • CWinApp // Класс «Приложение».
  — CDocTemplate // Классы шаблонов.
    • CSingleDocTemplate
    • CMultiDocTemplate
  — CDocument // Класс документа.
  — CWnd // Поддержка окон.
    • CFrameWnd // Масштабируемые окна.
      — CMDIFrameWnd
      — CMDIChildWnd
    • CDialog // Диалоговые окна.
      — CColorDialog // Диалоговое окно для выбора цвета.
      — CFileDialog // Диалоговое окно для выбора файла.
    • CView // Области просмотра.
    • CControlBar // Панели элементов управления.
      — CStatusBar
      — CDialogBar
      // Элементы управления.
    • CAnimateCtrl
    • CButton // Кнопка.
    • CComboBox // Комбинированный список.
    • CEdit // Поле редактора.
    • CListBox // Список.
    • CScrollBar // Полоса прокрутки.
    • CSliderCtrl // Регулятор.
    • CStatic
  — CException // Особые ситуации.
  — CFile // Работа с файлами.
  — CDC // Поддержка графики.
    • CClientDC
    • CWindowDC
    • CPaintDC
  — CGdiObject // Графические объекты.
    • CBitmap
    • CBrush
    • CFont
```

    - *CPen*
    - *CRgn*
    - *CPalette*
  - *CMenu* // Меню.
  - *CDatabase* // Поддержка ODBC.
  - *CDaoDatabase* // Поддержка DAO.
  - *CArray* // Массивы.
  - *CList* // Списки.

Классы, не порождаемые от CObject:

```
CArchive, CRuntimeClass, CPoint, CRect, CString, CTime ,…
```

Состав простейшего приложения в MFC

Простейшее приложение, создающее окно, состоит как минимум из двух классов:

- класс окна, где заголовок имеет вид:

```
class CMainWin: public CFrameWnd
```

- класс приложения, где заголовок имеет вид:

```
class CApp: public CWinApp
```

Обработка сообщений в MFC

Для обработки сообщений следует выполнить три действия:

1) включить в класс макрос:

```
DECLARE_MESSAGE_MAP() /* Макрос объявляет очередь со-
общений для класса */
```

2) включить макрокоманду в очередь сообщений:

```
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
// Начало очереди сообщений для класса.
ON_WM_CHAR() // Ввод символа.
ON_WM_LBUTTONDOWN() // Нажатие левой кнопки мыши.
```

```
ON_WM_LBUTTONUP() // Отпуск левой кнопки мыши.
ON_WM_MOUSEMOVE() // Перемещение мыши.
ON_COMMAND(ID_M1, OnCircle) // Обработка элемента меню.
ON_WM_PAINT() // Перерисовка.
END_MESSAGE_MAP() // Закрытие очереди сообщений.
```

3) включить в класс функцию-обработчика сообщения, например:

```
void CMainWin::OnChar(UINT Ch, UINT Count, UINT Flags) {…}
// Определение за пределами класса.
afx_msg void OnChar(UINT Ch, UINT Count, UINT Flags);
// Описание в классе.
```

Некоторые функции-обработчики

Ввод символа:

```
afx_msg void OnChar(UINT Ch, // Код символа.
UINT Count, // Количество введенных символов.
UINT Flags); /* Флаг, несущий дополнительную информацию,
            например, о нажатии клавиш Ctrl, Shift,… */
```

Нажатие левой кнопки мыши:

```
afx_msg void OnLButtonDown(UINT Flags,
// Флаг, несущий дополнительную информацию.
CPoint P); // Координаты курсора мыши.
```

Перемещение мыши:

```
afx_msg void OnMouseMove(UINT Flags, CPoint P);
```

Выбор элемента меню:

```
afx_msg void OnCircle();
```

Перерисовка:

```
afx_msg void OnPaint();
```

Для рисования в окне используют классы контекстов устройств, производные от класса CDC. Некоторые конструкторы классов контекстов устройств:

```
• CPaintDC(CWnd* pWnd) // Используется при перерисовке.
• CWindowDC(CWnd* pWnd)   // Поверхность всего окна.
• CClientDC(CWnd* pWnd)   // Клиентская область окна.
```

Заголовки некоторых методов класса CDC:

```
COLORREF SetPixel(int x, int y, COLORREF crColor);
COLORREF SetPixel(POINT point, COLORREF crColor);
CPoint MoveTo(int x, int y);
CPoint MoveTo(POINT point);
BOOL LineTo(int x, int y);
BOOL LineTo(POINT point);
BOOL Ellipse(int x1, int y1, int x2, int y2);
BOOL Ellipse(LPCRECT lpRect);
BOOL Polygon(LPPOINT lpPoints, int nCount);
BOOL Rectangle(int x1, int y1, int x2, int y2);
BOOL Rectangle(LPCRECT lpRect);
BOOL TextOut(int x, int y, LPCTSTR lpszString, int nCount);
BOOL TextOut(int x, int y, const CString& str);
COLORREF SetTextColor(COLORREF crColor);
COLORREF SetBkColor(COLORREF crColor);
```

Методы для загрузки объектов в контекст устройства:

```
// Загрузка стандартного объекта.
virtual CGdiObject* SelectStockObject(int nIndex);
// Загрузка других объектов.
CPen* SelectObject(CPen* pPen);
CBrush* SelectObject(CBrush* pBrush);
CFont* SelectObject(CFont* pFont);
CBitmap* SelectObject(CBitmap* pBitmap);
int SelectObject(CRgn* pRgn);
```

Графические объекты

Для задания «пера» используют класс `CPen`, конструкторы и некоторые методы класса `CPen`:

```
CPen();
CPen(int nPenStyle, int nWidth, COLORREF crColor);
BOOL CreatePen(int nPenStyle, int nWidth, COLORREF crColor);
BOOL CreateStockObject(int nIndex);
```

Для задания «кисти» используют класс `CBrush`. Конструкторы и некоторые методы класса `CBrush`:

```
CBrush();
CBrush(COLORREF crColor);
CBrush(int nIndex, COLORREF crColor);
CBrush(CBitmap* pBitmap);
BOOL CreateSolidBrush(COLORREF crColor);
BOOL CreateHatchBrush(int nIndex, COLORREF crColor);
BOOL CreateStockObject(int nIndex);
```

В библиотеке классов MFC можно создавать стандартные типы проектов в соответствии с шаблонами, например, можно создать однодокументное приложение (`Single document`). Классы однодокументного приложения:

```
class CMyProectApp : public CWinApp  // Класс-приложение.
class CMainFrame : public CFrameWnd // Класс главного окна.
class CMyProectDoc : public CDocument  // Класс-документ.
class CMyProectView : public CView  // Класс-представление.
```

Рассмотрим некоторые полезные методы.
Получить документ для представления:

```
CDocument* CView::GetDocument();
```

В классе главного окна есть методы:

```
CDocument* CFrameWnd::GetActiveDocument();
CView* CFrameWnd::GetActiveView();
```

В классе приложения есть поле — указатель на главное окно:

```
CWnd* m_pMainWnd;
```

Указатель на объект-приложение можно получить с помощью глобальной функции библиотеки MFC:

```
CWinApp * AfxGetApp();
```

Добавление функций-обработчиков сообщений и событий в однодокументном приложении. Осуществляется автоматически. Чтобы добавить обработчик сообщения, на вкладке «классы» выделяем требуемый класс, нажимаем правую клавишу мыши, выбираем элемент меню *Properties* (Свойства) и далее открываем вкладку Messages (Сообщения), выбираем тип сообщения и добавляем функцию-обработчика сообщений.

Рассмотрим, как добавить обработчик события, например «Выбор элемента меню»? В ресурсах выделяем элемент меню, нажимаем правую клавишу мыши, выбираем в меню элемент `Add Event Handler`, появляется окно для задания параметров функции-обработчика события, в котором задаем параметры и добавляем эту функцию.

Вывод графики и перерисовка

Вывод графики желательно выполнять внутри класса «Представление» (внутри функций-обработчиков этого класса).

Необходимо использовать классы контекстов устройств: `CClientDC`, `CWindowDC`.

Для перерисовки добавить код в метод

```
void CMyProectView::OnDraw(CDC* pDC)
```

класса представления после комментария

```
// TODO: add draw code for native data here
```

2.6.3. Задачи и порядок выполнения работы

Условие задачи

Разработать приложение с меню, в котором содержится несколько элементов (минимум два). Названия элементов должны соответствовать геометрическим фигурам (например, «круг», «квадрат», ...). При выборе элемента меню в окне рисуется соответствующая геометрическая фигура, граница — штриховая линия синего цвета, закраска — красными вертикальными линиями. Обеспечить перерисовку выведенных фигур, например, при сворачивании-разворачивании окна. Обеспечить

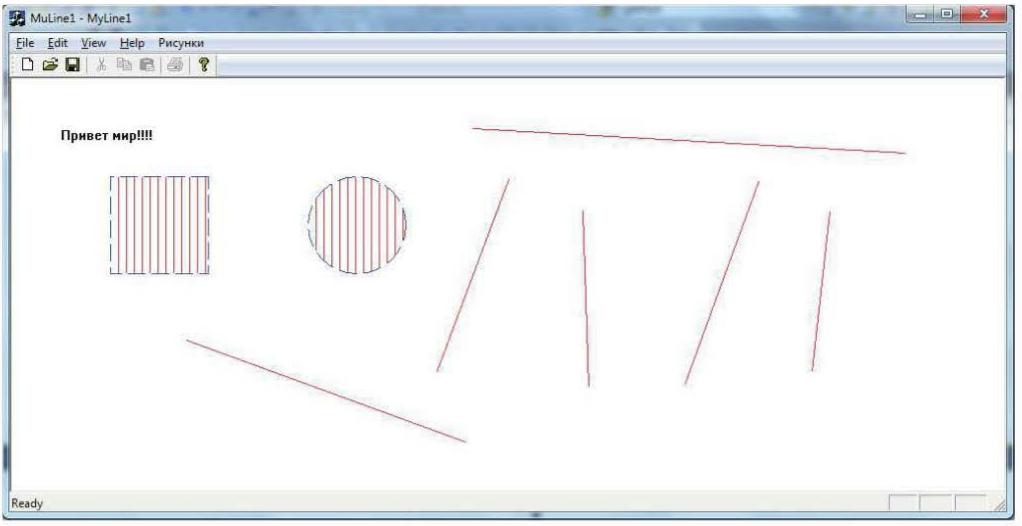


Рис. 2.4. Внешний вид окна приложения

ввод строки текста с клавиатуры и рисование с помощью мыши графического примитива в соответствии с вариантом задания, обеспечить перерисовку введенных элементов. Графический примитив — отрезок, начальная точка — точка положения курсора мыши в момент нажатия левой клавиши, рисование происходит при перемещении мыши с нажатой левой клавишей, конечная точка — положение курсора в момент отпускания клавиши. Для рисования отрезков использовать сплошное красное «перо». Возможный внешний вид окна приложения представлен на рис. 2.4.

Пример выполнения работы

В работе рекомендуется создать стандартное приложение MFC Application по шаблону MFC, окно для создания проекта представлено на рис. 2.5. В свойствах проекта необходимо задать тип приложения «Один документ» (`Single document`), вид окна со свойствами проекта представлен на рис. 2.6. Для упрощения интерфейса рекомендуется для параметра «Стиль проекта» установить значение «Стандарт MFC» или «Проводник Windows» (см. рис. 2.6). Все остальные параметры при создании проекта можно оставить без изменений (значения по умолчанию).

Для хранения данных внутри приложения задают необходимые поля в классе «Документ», вывод графики осуществляют через методы класса «Представление».

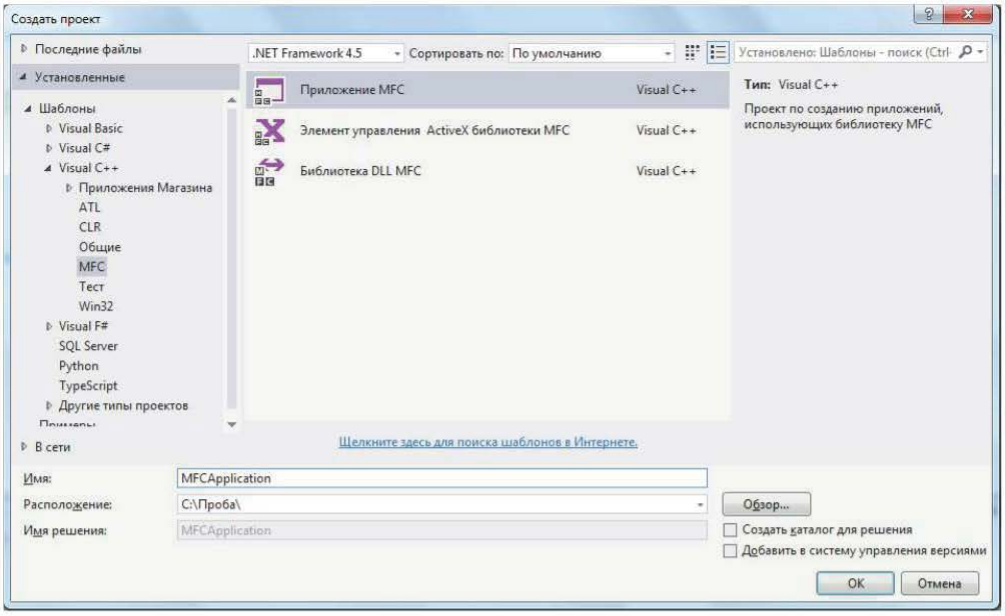


Рис. 2.5. Окно для создания проекта

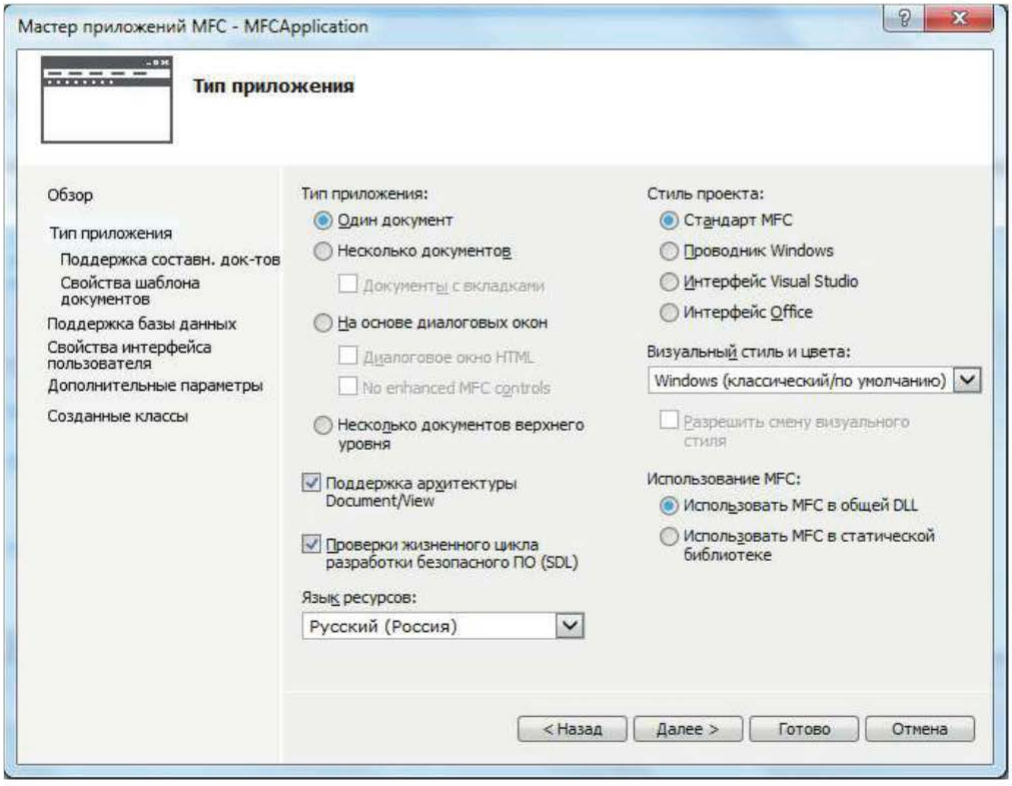


Рис. 2.6. Внешний вид окна со свойствами проекта

Фрагменты исходного кода, которые следует добавить в однодокументное приложение

Отрезки хранятся в оперативной памяти в виде линейного односвязного списка. Для этого создается новый класс — отрезок (линия) задают как элемент списка. Исходные коды класса:

```
// MyLine.h: interface for the MyLine class.

//////////////////////////////////////////////
#if !defined(AFX_MYLINE_H__65C81C1E_6FED_4C5E_8888_
7132A02976AB__INCLUDED_)
#define AFX_MYLINE_H__65C81C1E_6FED_4C5E_8888_7132A02976AB__
INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class MyLine: public CObject
{
public:
    void add(MyLine *&pF); /* Добавление элемента в на-
чало списка */
    MyLine();
    MyLine(CPoint p1, CPoint p2);
    virtual ~MyLine();
    CPoint p1, p2; // Координаты концов отрезка.
    MyLine *pNext; // Указатель на следующий элемент.
    static int Num; // Счетчик созданных объектов.
    DECLARE_SERIAL(MyLine)
};
#endif //!defined(AFX_MYLINE_H__65C81C1E_6FED_4C5E_...
// MyLine.cpp: implementation of the MyLine class.

//////////////////////////////////////////////////
#include «stdafx.h»
#include «MyLine1.h»
#include «MyLine.h»
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
```

```
//////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////
IMPLEMENT_SERIAL(MyLine, CObject, 1)
int MyLine::Num=0;
MyLine::MyLine()
{
    Num++;
}
MyLine::~MyLine()
{
    Num--;
}
MyLine::MyLine(CPoint p1, CPoint p2)
{
    this->p1=p1;
    this->p2=p2;
    Num++;
}
void MyLine::add(MyLine *& pF)
{
    pNext=pF;
    pF=this;
}
```

В класс документа включаем данные:

```
// Attributes
public:
MyLine *pF; // Указатель на первый элемент списка линий.
bool flagFig1, flagFig2; /* Флаги-признаки, что геометри-
ческие фигуры нарисованы */
CString text; // Строка текста, которая выводится.
```

В начало файла описания документа добавляем:

```
#include "MyLine.h"
```

Конструктор класса документа имеет вид

```
CMyLine1Doc::CMyLine1Doc()
```

```
{
    // TODO: add one-time construction code here
    pF=0;
    flagFig1=flagFig2=false; // Фигуры не нарисованы.
    text=""; // Строка текста пустая.
}
```

В описание класса «Представление» добавляем поля:

```
CPoint p1, p2; // Точки, задающие рисуемый отрезок.
    CPen pen1; // Перо для рисования фигур через меню.
    CBrush brush1; // Кисть для рисования фигур через меню.
```

В конструктор класса «Представление» добавляем код:

```
CMyLine1View::CMyLine1View()
{
/* TODO: Создание пера и кисти для рисования фигур
 через меню */
    pen1.CreatePen(PS_DASH, 1, RGB(0, 0, 255));
/* Создание синего пунктирного пера */
    brush1.CreateHatchBrush(HS_VERTICAL, RGB(255, 0, 0));
/* Создание красной кисти вертикальными линиями */
}
```

В класс «Представления» добавляем функции-обработчики событий и сообщений:

- нажатие левой клавиши мыши;
- перемещение мыши;
- отпускание левой клавиши мыши;
- выбор элемента меню с именем «Квадрат»;
- выбор элемента меню с именем «Круг»;
- ввод символа.

Как было отмечено выше, функции-обработчики добавляют в автоматизированном режиме. Например, для добавления обработчика сообщения «Нажатие левой клавиши мыши» в класс «Представление» выполняют следующие действия: в окне классов выделяют класс «Представление», в окне «Свойства» открывают вкладку «Сообщения», в списке сообщений выбирают сообщение с идентификатором WM_LBUTTONDOWN, активизируют этот элемент, в класс добавляется функция-обработчик сообщения (рис. 2.7).

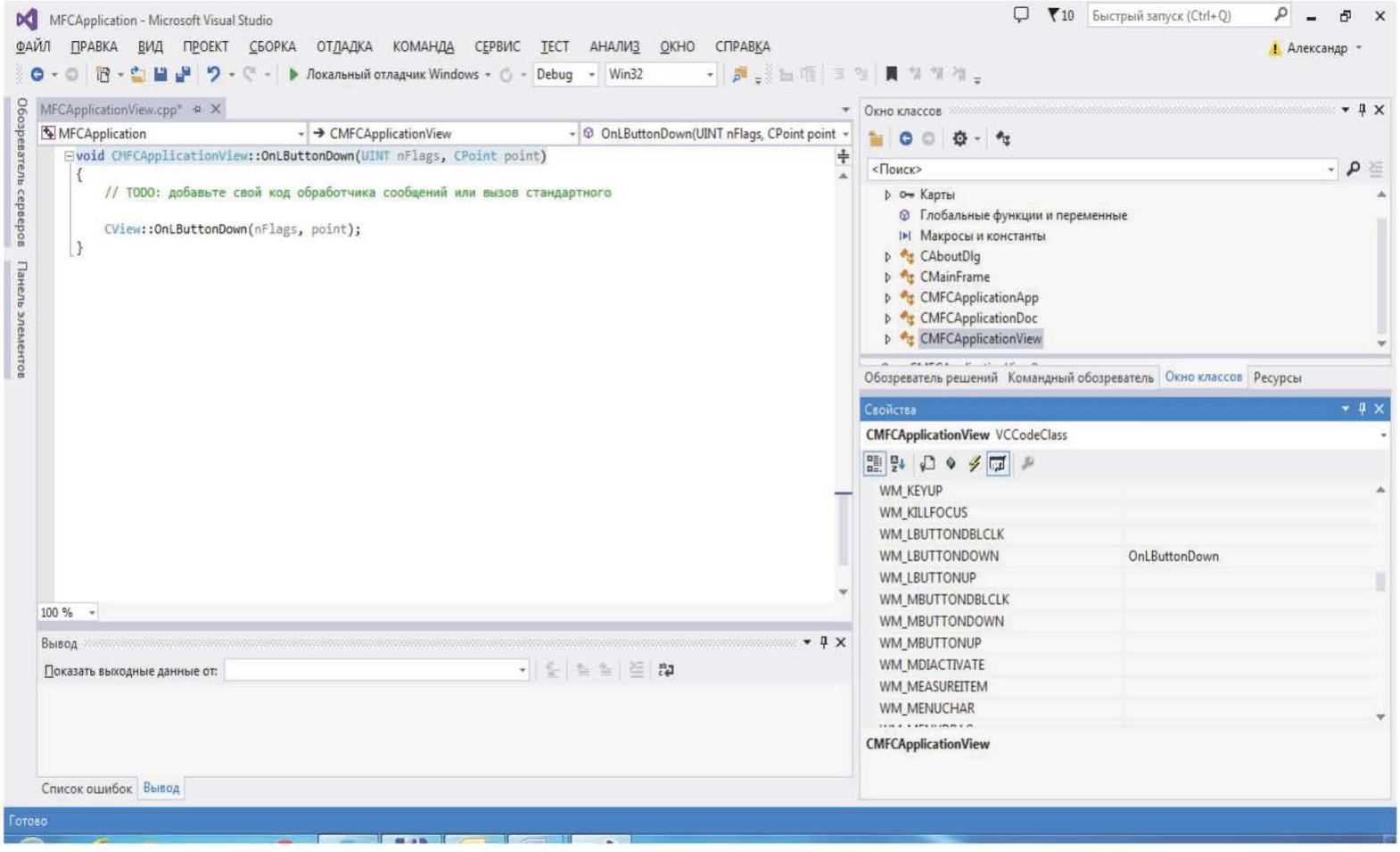


Рис. 2.7. Добавление в класс функции-обработчика сообщения

Сообщение «Ввод символа» добавляется в класс аналогично, только в списке сообщений выбираем сообщение с идентификатором `WM_CHAR`.

Для обработки события «Выбор элемента меню» необходимо выполнить следующее:

- открыть меню в файле ресурсов;
- выделить необходимый элемент меню и нажать правую клавишу мыши – появляется контекстное меню, в котором следует выбрать элемент «Добавить функцию-обработчика событий...», появляется диалоговое окно (рис. 2.8).

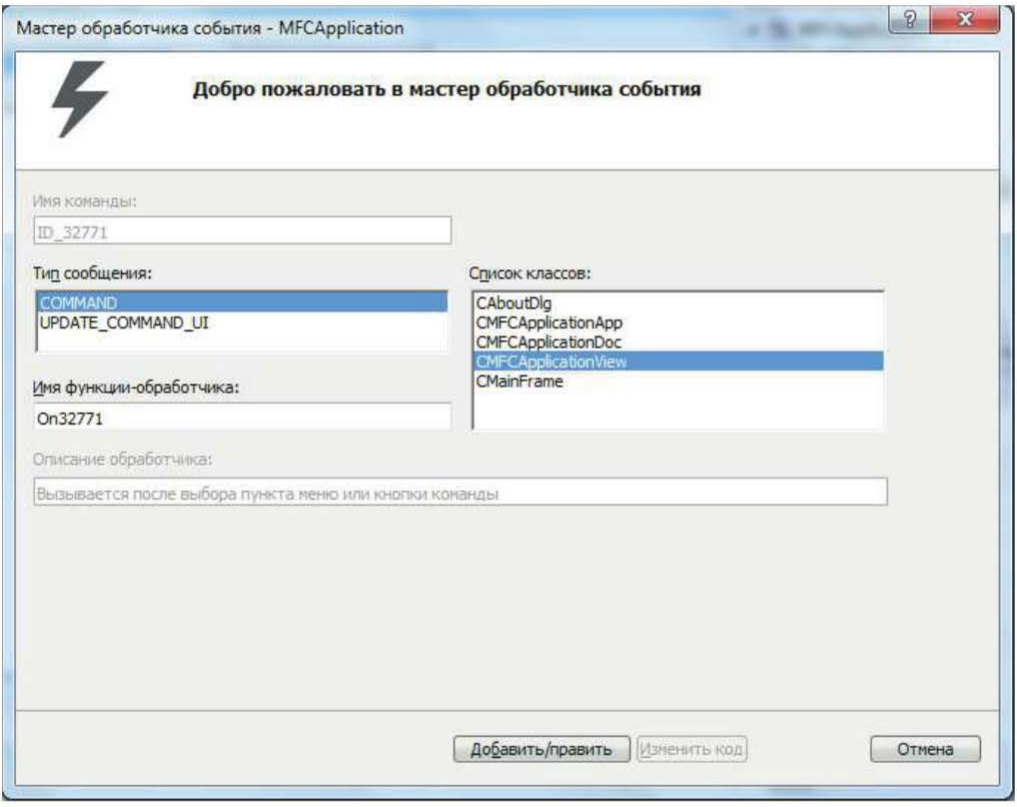


Рис. 2.8. Пример окна для добавления функции-обработчика события

В нем необходимо задать класс, куда добавляется функция-обработчик (класс «Представление»), и нажать кнопку «Добавить/править», открывается добавленная функция-обработчик для редактирования.

Исходные коды функций-обработчиков:

```
void CMyLine1View::OnLButtonDown(UINT nFlags, CPoint
point)
{
    // TODO: Add your message handler code here and/or
    // call default
    p1=p2=point;
    CView::OnLButtonDown(nFlags, point);
}

void CMyLine1View::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or
    // call default
    if (nFlags== MK_LBUTTON) // Нажатие левой кнопки мыши.
    {
          CPen pen1, pen2;
          pen1.CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
          pen2.CreateStockObject(WHITE_PEN);
          CClientDC d(this);
          d.SelectObject(&pen2);
          d.MoveTo(p1); d.LineTo(p2); // Стирание старой
                                              // линии.
          p2=point;
          d.SelectObject(&pen1);
          d.MoveTo(p1); d.LineTo(p2); // Рисование новой
                                              // линии.
    }
    CView::OnMouseMove(nFlags, point);
}
void CMyLine1View::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or
    call default
    MyLine *p=new MyLine(p1, p2); // Создание объекта-линии.
    CMyLine1Doc*  pDoc = GetDocument(); /* Указатель на
                                    объект «Документ» */
    p->add(pDoc->pF); // Добавление объекта в список.
    pDoc->SetModifiedFlag(1); // Документ изменен.
    CView::OnLButtonUp(nFlags, point);
}
```

```
void CMyLine1View::On32771()
{
    // TODO: Рисование квадрата.
     CClientDC dc(this); // Создание объекта «Контекст
                                          // устройства».
    dc.SelectObject(&pen1); // Загрузка пера.
    dc.SelectObject(&brush1); // Загрузка кисти.
    dc.Rectangle(100, 100, 200, 200); // Рисование квадрата.
 CMyLine1Doc* pDoc = GetDocument(); /* Указатель на объ-
                                          ект «Документ» */
    pDoc->flagFig1=true; /* Флаг — признак, что фигура на-
                                          рисована */
}
void CMyLine1View::On32772()
{
    // TODO: Рисование круга.
    CClientDC dc(this); // Создание объекта «Контекст уст-
                                          ройства».
    dc.SelectObject(&pen1); // Загрузка пера.
    dc.SelectObject(&brush1); // Загрузка кисти.
    dc.Ellipse(300, 100, 400, 200); // Рисование круга.
CMyLine1Doc* pDoc = GetDocument(); /* Указатель на объект
                                          «Документ» */
     pDoc->flagFig2=true; // Флаг — признак, что фигура
                                          // нарисована.
}
void CMyLine1View::OnChar(UINT nChar, UINT nRepCnt, UINT
nFlags)
{
    // TODO: Функция-обработчик сообщения — ввод символа.
     CMyLine1Doc* pDoc = GetDocument(); /* Указатель на
                                     объект «Документ» */
     pDoc->text+=(TCHAR)nChar; /* Дописывание символа к
                                          строке текста */
      CClientDC dc(this); // Создание объекта «Контекст
                                          // устройства»
  dc.TextOut(50, 50, pDoc->text); /* Вывод строки текста
                              в «Контекст устройства» */
                  CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Добавляем код в метод класса «Представление» для перерисовки:

```
void CMyLine1View::OnDraw(CDC* pDC)
{
    CMyLine1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
       CPen pen(PS_SOLID, 1, RGB(255, 0, 0));   /* Перо
                                  для рисования линий */.
       pDC->SelectObject(pen); /* Загрузка пера в «Кон-
                                      текст устройства» */
       MyLine *p=pDoc->pF; /* Указатель указывает на первый
                                          элемент списка */
    while(p) // Просмотр списка и рисование линии.
    {
          pDC->MoveTo(p->p1);
          pDC->LineTo(p->p2);
          p=p->pNext;
    }
    // Перерисовка строки текста.
      pDC->TextOut(50, 50, pDoc->text); /* Вывод строки
                       текста в «Контекст устройства» */
    if (pDoc->flagFig1) // Если фигура была нарисована.
    {
     pDC->SelectObject(&pen1); // Загрузка пера.
     pDC->SelectObject(&brush1); // Загрузка кисти.
     pDC->Rectangle(100, 100, 200, 200); // Рисование круга.
    }
    if (pDoc->flagFig2) // Если фигура была нарисована
    {
     pDC->SelectObject(&pen1); // Загрузка пера.
     pDC->SelectObject(&brush1); // Загрузка кисти.
     pDC->Ellipse(300, 100, 400, 200); // Рисование круга.
    }
}
```

Задания для самоконтроля

1. Перечислите классы, объекты которых определяют простейшее оконное приложение в MFC.
2. Назовите действия, которые необходимо выполнить для добавления функции-обработчика сообщения в приложение MFC.
3. Перечислите классы «Контексты устройств» в MFC.
4. Приведите примеры методов класса CDC для вывода графики.
5. Покажите, как создать «перо» и «кисть» в MFC.

Заключение

В настоящих методических указаниях рассмотрены содержание и особенности выполнения лабораторных работ по дисциплине «Алгоритмические языки». Работы выполняются в течение первого и второго семестров студентами, обучающимися на кафедре «Информационная безопасность», и в течение третьего и четвертого семестров — студентами факультета Главного учебно-исследовательского и методического центра МГТУ им. Н.Э. Баумана, обучающимися по специальностям этой кафедры.

Для каждой лабораторной работы приведены цель и задачи, требования к результатам ее выполнения, краткая характеристика объекта изучения, порядок выполнения работы и примеры выполнения. Также для подготовки студентов представлены вопросы для самоконтроля, в приложении заданы варианты.

Выполнение предложенных работ позволит студенту получить знания, навыки и умения по основным средствам языка Си, основным объектно-ориентированным возможностям языка Си++, возможностям библиотеки классов MFC, а также подготовиться к изучению таких объектно-ориентированных языков программирования, как Си# и Java, на последующих этапах обучения.

Литература

1. Распространение программного обеспечения по подписке DreamSpark Premium (MSDNAA). URL: http://msdnaa.lib.bmstu.ru/default.aspx (дата обращения: 20.04.2016).
2. *Керниган Б., Ритчи Д.* Язык программирования С. 2-е изд. М.: Вильямс, 2009. 304 с.
3. *Страуструп Б.* Язык программирования С++. М.: БИНОМ, 2011. 1136 с.
4. *Подбельский В.В., Фомин С.С.* Курс программирования на языке Си: учебник. М.: ДМК-Пресс, 2015. 384 с.
5. *Вирт Н.* Алгоритмы и структуры данных. Новая версия для Оберона. М.: ДМК-Пресс, 2010. 272 с.
6. *Румянцев П.В.* Азбука программирования в Win 32 API. М.: Горячая линия — Телеком, 2004. 312 с.
7. *Подбельский В.В.* Стандартный Си++ : учеб. пособие. М.: Финансы и статистика, 2014. 687 с.
8. *Уильямс Э.* Параллельное программирование на С++ в действии. Практика разработки многопоточных программ. М.: ДМК-Пресс, 2012. 672 с.
9. *Румянцев П.В.* MFC — внутренний мир. М.: Горячая линия — Телеком, 2015. 350 с.

Приложение
ВАРИАНТЫ ЗАДАНИЙ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Варианты заданий для лабораторной работы № 1.1

Часть 1

Вариант № 1

Угол α задан в градусах, минутах и секундах. Найти его значение в радианах (с максимально возможной точностью).

Вариант № 2

Угол α задан в радианах. Найти его значение в градусах, минутах и секундах.

Вариант № 3

Длина отрезка задана в дюймах (1 дюйм = 2,54 см). Перевести значение длины в метры, сантиметры и миллиметры. Так, например, 21 дюйм = 0 м 53 см 3,4 мм.

Вариант № 4

Заданы моменты начала и конца некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах измерения.

Вариант № 5

Заданы три корня кубического уравнения: x_1, x_2, x_3. Найти коэффициенты этого уравнения.

Вариант № 6

Найти корни квадратного уравнения, заданного своими коэффициентами, с положительным дискриминантом. Подставив корни в уравнение, убедиться в погрешности вычислений.

Вариант № 7

Заданы действительная и мнимая части комплексного числа $z = x + iy$. Преобразовать его в тригонометрическую форму и напечатать в виде выражения: $z = r(\cos \phi + i \sin \phi)$.

Для справки:

$r = \sqrt{x^2 + y^2}$; $\phi = \text{arctg}\frac{y}{x}$.

Вариант № 8

Владелец автомобиля приобрел новый карбюратор, который экономит 50 % топлива, новую систему зажигания, которая экономит 30 % топлива, и поршневые кольца, экономящие 20 % топлива. Верно ли, что его автомобиль теперь сможет обходиться совсем без топлива? Найти фактическую экономию для произвольно заданного сэкономленного количества вещества.

Вариант № 9

Из круга радиусом r вырезан прямоугольник, бóльшая сторона которого равна a. Найти максимальный радиус круга, который можно вырезать из этого прямоугольника.

Вариант № 10

Приближение sin x. Функция $y = \sin x$ на отрезке $\left[0, \frac{\pi}{2}\right]$ хорошо аппроксимируется разложением: $y = x - \frac{x^3}{6} + \frac{x^5}{120}$. Для заданного значения аргумента x вычислить y по этой формуле и сравнить с точным значением, вычисленным с помощью стандартной функции sin.

Вариант № 11

Под прямым углом к фарватеру пловцу надо преодолеть реку шириной b м. Его скорость в стоячей воде v_1, м/с; скорость течения

реки v_2, м/с. Под каким углом к фарватеру он должен плыть, чтобы его не снесло? Сколько времени займет переправа? Как изменится решение, если посередине реки скорость пловца снизится с v_1 до v_3, м/с?

Вариант № 12

Найти координаты вершины параболы $y = ax^2 + bx + c$.

Вариант № 13

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти площадь треугольника *ABC*.

Вариант № 14

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти сумму длин медиан треугольника *ABC*.

Вариант № 15

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти точку пересечения биссектрис треугольника *ABC* (центр вписанной в него окружности).

Вариант № 16

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти внутренние углы треугольника *ABC* (в градусах).

Вариант № 17

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти длину и основание высоты, опущенной из вершины *A* на сторону *BC*.

Вариант № 18

Треугольник задан координатами вершин на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти точку *D*, симметричную точке *A* относительно стороны *BC*.

Вариант № 19

В равнобедренном прямоугольном треугольнике известна высота h, опущенная на гипотенузу. Найти стороны треугольника.

Вариант № 20

У квадрата *ABCD* на плоскости известны координаты двух противоположных вершин — точек *A* и *C*. Найти координаты точек *B* и *D*.

Примечание. Расположение квадрата произвольное, его стороны не обязательно параллельны координатным осям.

Вариант № 21

Треугольник *ABC* задан длинами своих сторон. Найти длину высоты, опущенной из вершины *A*.

Вариант № 22

Заданы уравнения двух пересекающихся прямых на плоскости: $y = k_1 x + b_1$, $y = k_2 x + b_2$. Найти (в градусах, минутах и секундах) угол между ними, используя формулу $\operatorname{tg}\phi = \dfrac{(k_2 - k_1)}{(1 + k_1 k_2)}$.

Вариант № 23

Трехмерные векторы заданы координатами, например, $A(x_a, y_a, z_a)$, $B(x_b, y_b, z_b)$. Найти угол в градусах между векторами *A* и *B*, используя формулу $\cos\phi = \dfrac{(A,B)}{|A|\cdot|B|} = \dfrac{x_a x_b + y_a y_b + z_a z_b}{\sqrt{x_a^2 + y_a^2 + z_a^2}\cdot\sqrt{x_b^2 + y_b^2 + z_b^2}}$.

Вариант № 24

Трехмерные векторы заданы координатами, например $A(x_a, y_a, z_a)$, $B(x_b, y_b, z_b)$, $C(x_c, y_c, z_c)$. Найти объем пирамиды, построенной на векторах *A*, *B*, *C* как на сторонах.

Вариант № 25

Трехмерные векторы заданы координатами, например $A(x_a, y_a, z_a)$, $B(x_b, y_b, z_b)$, $C(x_c, y_c, z_c)$. Найти длину диагонали параллелепипеда, построенного на векторах *A*, *B*, *C* как на сторонах.

Вариант № 26

Определить периметр правильного n-угольника, описанного около окружности радиусом r.

Вариант № 27

Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

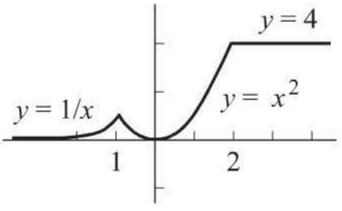
Часть 2

Вариант № 1

Даны действительные положительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x, y, z.

Вариант № 2

Дано действительное a. Для функции $f(a)$, график которой представлен на рисунке, вычислить $f(a)$.



Вариант № 3

Даны три действительных числа. Выбрать из них те, которые принадлежат интервалу (1, 3), или вывести сообщение, что таких чисел нет.

Вариант № 4

Даны действительные числа x, y. Если x, y отрицательные, каждое значение заменить его модулем; если отрицательное только одно из них, оба значения увеличить на 0,5; если оба значения неотрицательные

и ни одно из них не принадлежит отрезку [0,5; 2,0], оба значения уменьшить в 10 раз; в остальных случаях x, y оставить без изменения.

Вариант № 5

Определить и вывести на печать номер квадранта, в котором расположена точка $M(x, y)$; x и y — заданные вещественные числа.

Вариант № 6

Из величин, определяемых выражениями $a = \sin x$, $b = \cos x$, $c = \ln|x|$ при заданном x, определить и вывести на экран дисплея минимальное значение.

Вариант № 7

Определить, какая из двух точек — $M_1(x_1, y_1)$ или $M_2(x_2, y_2)$ — расположена ближе к началу координат. Вывести на экран дисплея координаты этой точки.

Вариант № 8

Определить, какая из двух фигур (круг или квадрат) имеет бóльшую площадь. Известно, что сторона квадрата равна a, радиус круга r. Вывести на экран название и значение площади бóльшей фигуры.

Вариант № 9

Определить, попадает ли точка $M(x, y)$ в круг радиусом r с центром в точке (x_0, y_0).

Вариант № 10

Найти количество положительных (отрицательных) чисел среди четырех целых чисел A, B, C, D.

Вариант № 11

Написать программу, определяющую, есть ли в введенном числе дробная часть.

Вариант № 12

Пройдет ли кирпич со сторонами a, b и c через прямоугольное отверстие со сторонами r и s? Стороны отверстия должны быть параллельны граням кирпича.

Вариант № 13

Может ли шар радиусом r пройти через ромбообразное отверстие с диагоналями p и q?

Вариант № 14

Вычислить значение выражения $y = \text{sqrt}\ (2X - 1)$ в случае, когда это возможно (т. е. если $(2X - 1) >= 0$). В противном случае вывести на сообщение «Задайте другое значение X».

Вариант № 15

Запросить рост и массу пользователя вашей программы. В случае, если масса больше, чем рост (−100), то вывести сообщение «Хорошо бы вам похудеть». В противном случае вывсти сообщение «Ваша масса в норме».

Вариант № 16

Если целое число M делится нацело на целое число N, вывести на экран частное от деления, в противном случае вывести сообщение «M на N нацело не делится».

Вариант № 17

Записать условный оператор, в котором значение переменной c вычисляется по формуле $a + b$, если a — нечетное, и $a \cdot b$, если a — четное.

Вариант № 18

Написать программу для подсчета суммы только положительных из трех данных чисел.

Вариант № 19

Заданы стороны треугольника a, b, c. Проверить, является ли этот треугольник остроугольным.

Вариант № 20

Заданы стороны треугольника a, b, c. Проверить, является ли этот треугольник тупоугольным.

Вариант № 21

Заданы три параметра a, b, c. Проверить, могут ли эти параметры быть длинами сторон треугольника.

Вариант № 22

Даны три значения. Вывести бóльшее из них.

Вариант № 23

Даны три переменные вещественного типа: A, B, C. Если их значения упорядочены по возрастанию, удвоить их; в противном случае заменить значение каждой переменной на значение с противоположным знаком. Вывести новые значения переменных A, B, C.

Вариант № 24

Даны вещественные числа a, b, c ($a \neq 0$). Выяснить, имеет ли уравнение $ax^2 + bx + c = 0$ вещественные корни.

Вариант № 25

Известны две скорости: одна в километрах в час, другая — в метрах в секунду. Какая из скоростей больше?

Вариант № 26

Даны радиус круга и сторона квадрата. У какой фигуры площадь больше?

Вариант № 27

Даны объемы и массы двух тел из разных материалов. Материал какого из тел имеет бóльшую плотность?

Вариант № 28

Известны сопротивления двух не соединенных друг с другом участков электрической цепи и напряжение на каждом из них. По какому участку протекает ток меньшей силы?

Варианты заданий для лабораторной работы № 1.2

Вариант № 1

Найти сумму первых k чисел последовательности Фибоначчи. Последовательность определяется законом:

$F_0 = F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 2$.

Вариант № 2

Определить первое число последовательности Фибоначчи, превышающее число A. Последовательность определяется законом:

$F_0 = F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 2$.

Вариант № 3

Найти все нечетные числа последовательности Фибоначчи, не превышающие заданное число A. Последовательность определяется законом:

$F_0 = F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 2$.

Вариант № 4

Вычислить сумму ряда $S = \sum_{k=1}^{\infty} (-1)^{k-1} / k$ с точностью $\xi = 10^{-2}, \ldots, 10^{-6}$.

Точное значение: ln 2.

Определить, как изменяется число итераций при изменении точности.

Вариант № 5

Вычислить: $\sqrt{3+\sqrt{6+\ldots\sqrt{60+\sqrt{63}}}}$.

Вариант № 6

Вычислить методом трапеций определенный интеграл:

$\int\limits_{\pi/6}^{\pi/3}\left(\mathrm{tg}^2x+\mathrm{ctg}^2x\right)dx$ при n = 20, 30, 40 (количество шагов разбиения интервала).

Вариант № 7

Вычислить сумму ряда $S=1+\sum\limits_{k=1}^{\infty}(-1)^k/(2k)!$ с точностью ξ = $= 10^{-2}, \ldots, 10^{-6}$. Точное значение: cos 1.

Определить, как изменяется число итераций при изменении точности.

Вариант № 8

Решить задачу, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-6}$.

Вычислить значение определенного интеграла методом трапеции:

$\int\limits_{0}^{2}e^x\left(1+\sin x\right)/\left(1+\cos x\right)dx$ при $\xi = 10^{-2},\ldots, 10^{-6}$.

Вариант № 9

Проверить численно первый замечательный предел $\lim\limits_{x\to 0}\dfrac{\sin x}{x}=1$, задавая значения x: 1; 1/2; 1/4; 1/8; ... до тех пор, пока левая часть равенства не будет отличаться от правой менее чем на заданную погрешность $\xi = 10^{-2},\ldots, 10^{-6}$. Определить, как изменяется число итераций в зависимости от ξ.

Вариант № 10

Проверить численно второй замечательный предел $\lim\limits_{n\to\infty}\left(1+\dfrac{1}{n}\right)^n=e$, задавая значения n: 1; 2; 3; 4; ... . При каком n исследуемое выражение отличается от e менее чем на заданную погрешность $\xi = 10^{-2},\ldots, 10^{-6}$?

Вариант № 11

Сравнить скорость сходимости (количество слагаемых для достижения заданной точности $\xi = 10^{-2},..., 10^{-6}$) следующих разложений числа π:

$$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...\right); \quad \pi = 3+4\left(\frac{1}{2\cdot 3\cdot 4}-\frac{1}{4\cdot 5\cdot 6}+\frac{1}{6\cdot 7\cdot 8}-...\right).$$

Вариант № 12

Сравнить скорость сходимости (число учитываемых элементов для достижения заданной точности $\xi = 10^{-2}, ..., 10^{-6}$) при вычислении числа e с помощью ряда и бесконечной дроби:

$$e = 2+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+...; \; e = 1+\cfrac{1}{1-\cfrac{1}{2+\cfrac{1}{3-\cfrac{1}{2+\cfrac{1}{5-...}}}}}.$$

Вариант № 13

Сравнить скорость сходимости (количество слагаемых для достижения заданной точности $\xi = 10^{-2}, ..., 10^{-6}$) следующих разложений числа π:

$$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...\right); \; \pi = \sqrt{6\left(1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+...\right)}.$$

Вариант № 14

Для заданных a и p вычислить $x = \sqrt[p]{a}$ по рекуррентному соотношению Ньютона:

$$x_{n+1} = \frac{1}{p}\left[(p-1)x_n + \frac{a}{x_n^{p-1}}\right]; \; x_0 = a.$$

Сколько итераций надо выполнить, чтобы для заданной погрешности $\xi = 10^{-2},..., 10^{-6}$ выполнялось соотношение $|x_{n+1} - x_n| \le \xi$?

Вариант № 15

Вычислить $x = \sqrt[3]{a}$ для заданного значения a, используя рекуррентное соотношение

$$x_{n+1} = \frac{1}{3}\left[x_n + 2\sqrt{\frac{a}{x_n}} \right];\ x_0 = a.$$

Сколько итераций надо выполнить, чтобы для заданной погрешности $\xi = 10^{-2}, ..., 10^{-6}$ выполнялось соотношение $|x_{n+1} - x_n| \le \xi$?

Вариант № 16

Для заданного $x > 1$ вычислить $y = \sqrt{x}$ по итерационной формуле $y_i = \frac{1}{2}\left(y_{i-1} + \frac{x}{y_{i-1}} \right)$ с заданной погрешностью $\xi = 10^{-2}, ..., 10^{-6}$, задав начальное приближение $y_0 = x$. Сравнить с результатом встроенной функции. Сколько итераций пришлось выполнить?

Вариант № 17

Вычислить натуральный логарифм, организовав цикл с точностью $\xi = 10^{-2}, ..., 10^{-6}$. Определить, как изменяется число итераций в зависимости от точности.

$$\ln x = 2\left[\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1} \right)^3 + ... + \frac{1}{2n-1}\left(\frac{x-1}{x+1} \right)^{2n-1} + ... \right],\ x > 0.$$

Вариант № 18

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, ..., 10^{-6}$. Определить, как изменяется число итераций в зависимости от точности.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ... + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + ...$$

Вариант № 19

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-6}$. Определить, как изменяется число итераций в зависимости от точности.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots .$$

Вариант № 20

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-8}$. Определить, как изменяется число итераций в зависимости от точности.

$$\sin x = x\left(1 - \frac{x^2}{\pi^2}\right)\left(1 - \frac{x^2}{4\pi^2}\right)\ldots\left(1 - \frac{x^2}{(n)^2\pi^2}\right)\ldots .$$

Вариант № 21

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-8}$. Определить, как изменяется число итераций в зависимости от точности.

$$\cos x = \left(1 - \frac{4x^2}{\pi^2}\right)\left(1 - \frac{4x^2}{9\pi^2}\right)\ldots\left(1 - \frac{4x^2}{(2n-1)^2\pi^2}\right)\ldots .$$

Вариант № 22

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-8}$. Определить, как изменяется число итераций в зависимости от точности.

$$\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + (-1)^n \frac{x^{2n+1}}{2n+1} + \ldots \ , \ x \in [-1, 1].$$

Вариант № 23

Вычислить тригонометрическую функцию, организовав цикл с точностью $\xi = 10^{-2}, \ldots, 10^{-8}$. Определить, как изменяется число итераций в зависимости от точности.

$$\ln(1-x) = -\left(x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots + \frac{x^n}{n} + \ldots\right), \quad 0 < x < 1.$$

Вариант № 24

Вычислить $\sqrt{1+\sqrt{2+\sqrt{3+\ldots+\sqrt{49+\sqrt{50}}}}}$.

Вариант № 25

Вычислить $\sqrt{2+\sqrt{2+\sqrt{2+\ldots+\sqrt{2+\sqrt{2}}}}}$; n слагаемых, n вводится с клавиатуры.

Вариант № 26

Вычислить с заданной точностью ($\xi = 10^{-2}, \ldots, 10^{-8}$):

$$\sqrt{2} = 1+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\ldots}}}}}.$$

Вариант № 27

Вычислить с заданной точностью ($\xi=10^{-2},\ldots, 10^{-8}$):

$$e = 2+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{6+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{8+\ldots}}}}}}}}}}}.$$

Варианты заданий для лабораторной работы № 1.3

Часть 1. Одномерные массивы

Вариант № 1

Задан одномерный массив целых чисел. Написать программу, которая находит сумму элементов массива.

Вариант № 2

Задан одномерный массив целых чисел. Написать программу, которая находит среднее арифметическое элементов массива.

Вариант № 3

Написать программу, которая находит скалярное произведение двух n-мерных векторов.

Указание. Скалярное произведение определяется по формуле

$$\left(\vec{a},\vec{b}\right)=\sum_{i=1}^{n}a_i b_i \cdot$$

Вариант № 4

Задан одномерный массив целых чисел. Написать программу, которая «переворачивает» массив, т. е. меняет местами его первый и последний элементы, второй и предпоследний и т. д.

Вариант № 5

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит $\max\left(a_1^2, a_2^2, ..., a_n^2\right)$.

Вариант № 6

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит $\min\left(a_1, 2a_2, ..., na_n\right)$.

Вариант № 7

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит $\min(a_1 + a_2, a_2 + a_3, ..., a_{n-1} + a_n)$.

Вариант № 8

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит $\max(a_1, a_1a_2, ..., a_1a_2 ... a_n)$.

Вариант № 9

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит количество четных чисел среди чисел a_1, a_2, ..., a_n.

Вариант № 10

Задан одномерный массив целых чисел. Написать программу, которая выводит элементы массива на экран и находит произведение элементов массива.

Вариант № 11

Задан одномерный массив целых чисел. Написать программу, которая выводит элементы массива на экран и находит среднее геометрическое элементов массива.

Вариант № 12

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит сумму четных чисел среди чисел a_1, a_2, ..., a_n.

Вариант № 13

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит сумму элементов с четными индексами.

Вариант № 14

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит сумму элементов с нечетными индексами.

Вариант № 15

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая меняет местами минимальный и максимальный элементы (или любую пару из этих элементов в случае, если таких элементов несколько).

Вариант № 16

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит индекс минимального элемента.

Вариант № 17

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит минимальный элемент среди элементов с четными индексами.

Вариант № 18

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит максимальный четный элемент.

Вариант № 19

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит сумму отрицательных элементов (при формировании массива обеспечить, чтобы были отрицательные элементы).

Вариант № 20

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая считает число нечетных элементов.

Вариант № 21

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая считает число элементов, кратных 3.

Вариант № 22

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая считает сумму элементов, кратных 7.

Вариант № 23

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая выводит индексы отрицательных элементов на экран (при формировании массива обеспечить, чтобы были отрицательные элементы).

Вариант № 24

Заданы целые числа a_1, a_2, ..., a_n. Число b вводится с клавиатуры. Проверить, есть ли в массиве элемент со значением b. Если элемент есть, вывести индекс (индексы) такого элемента.

Вариант № 25

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая находит количество положительных элементов (при формировании массива обеспечить, чтобы были отрицательные элементы).

Вариант № 26

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая отрицательные элементы заменяет на их абсолютные значения и печатает преобразованный массив (при формировании массива обеспечить, чтобы были отрицательные элементы).

Вариант № 27

Заданы целые числа a_1, a_2, ..., a_n. Написать программу, которая нечетные элементы удваивает и печатает преобразованный массив.

Часть 2. Многомерные массивы (матрицы)

Вариант № 1

Заданы матрица с элементами размерности $n \times m$ и вектор с элементами размерности m. Написать программу, которая находит произведение матрицы на вектор.

Указание. Результатом перемножения матрицы на вектор является вектор размерности n с компонентами $c_i = \sum_{j=1}^{m} a_{ij} b_j$.

Вариант № 2

Заданы две матрицы размерностей $n \times m$ и $m \times l$. Написать программу, которая находит произведение этих матриц.

Указание. Результатом перемножения двух матриц является матрица размерности $n \times l$ с элементами $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$.

Вариант № 3

Задана квадратная матрица размерности $n \times n$. Написать программу, которая находит транспонированную матрицу.

Указание. Транспонированием матрицы называется преобразование, при котором элементы, меняются местами, строки становятся столбцами, а столбцы — строками.

Вариант № 4

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет числа b_1, b_2, ..., b_n, равные суммам элементов строк матрицы.

Вариант № 5

Задана целочисленная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет числа b_1, b_2, ..., b_n, равные произведениям элементов строк матрицы.

Вариант № 6

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет числа b_1, b_2, ..., b_n, равные наименьшим значениям элементов строк матрицы.

Вариант № 7

Задана целочисленная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет действительные числа b_1, b_2, ..., b_n, равные значениям средних арифметических элементов строк матрицы.

Вариант № 8

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет числа b_1, b_2, ..., b_n, показывающие, сколько отрицательных элементов расположено в каждой строке матрицы.

Вариант № 9

Задана квадратная целочисленная матрица a_{ij} размерности $n \times n$. Написать программу, которая заменяет все отрицательные элементы матрицы на –1, положительные — на 1, а нулевые элементы оставляет без изменения.

Вариант № 10

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая вычисляет новую матрицу b_{ij}. Матрица получается путем деления всех элементов заданной матрицы a_{ij} на ее наибольший по модулю элемент.

Вариант № 11

Задана целочисленная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет числа b_1, b_2, ..., b_n, равные числу нечетных элементов в строках матрицы.

Вариант № 12

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет действительные числа b_1, b_2, ..., b_n, равные значениям средних геометрических элементов строк матрицы.

Вариант № 13

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет действительные числа b_1, b_2, ..., b_n, равные значениям минимальных элементов строк матрицы.

Вариант № 14

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет целые числа b_1, b_2, ..., b_n, равные индек-

сам максимальных элементов строк матрицы (если в строке 2 максимальных элемента и более, то использовать любой индекс).

Вариант № 15

Задана действительная матрица a_{ij} размерности $n \times m$. Написать программу, которая определяет целые числа $b_1, b_2, ..., b_n$, равные числу отрицательных элементов строк матрицы (при формировании матрицы обеспечить наличие в ней отрицательных элементов).

Вариант № 16

Матрицу **M**(m, n) заполнить натуральными числами от 1 до $m \times n$ по спирали, начинающейся в левом верхнем углу и закрученной по часовой стрелке.

Вариант № 17

Матрицу **K**(m, n) заполнить следующим образом. Элементам, находящимся на периферии (по периметру матрицы), присвоить значение 1; периметру оставшейся подматрицы — значение 2 и так далее до заполнения всей матрицы.

Вариант № 18

Символьную матрицу **A**(m, n) заполнить символами с кодами менее 128. Найти строки, содержащие только цифры от 0 до 9, или вывести сообщение, что таких строк нет.

Вариант № 19

Матрицу **A**(m, n) заполнить следующим образом. Для заданных k и l (k и l вводятся с клавиатуры) элементу a_{kl} присвоить значение 1; элементам, окаймляющим его (соседним с ним по вертикали, горизонтали и диагоналям) присвоить значение 2; элементам следующего окаймления — значение 3 и так далее до заполнения всей матрицы.

Вариант № 20

Матрица **K**(m, m) состоит из нулей и единиц. Найти в ней номера (индексы) хотя бы одной строки или хотя бы одного столбца, не содержащих единицы, либо сообщить, что таковых нет.

Вариант № 21

Латинским квадратом порядка n называется квадратная таблица размером $1n$, каждая строка и каждый столбец которой содержат все числа от 1 до n. Проверить, является ли заданная целочисленная матрица латинским квадратом.

Вариант № 22

Магическим квадратом порядка n называется квадратная таблица размером n, состоящая из чисел так, что суммы по каждому столбцу, каждой строке и каждой из двух диагоналей равны между собой. Проверить, является ли заданная целочисленная квадратная матрица магическим квадратом.

Вариант № 23

Клеточное поле размером есть результат игры в крестики-нолики на «бесконечном» поле (задать размерность поля равную n). Проверить, не закончена ли игра выигрышем крестиков? Считается, что крестики выиграли, если на поле найдется по горизонтали, вертикали или диагонали цепочка, состоящая подряд из 5 крестиков.

Вариант № 24

В каждой строке матрицы **A**(n, n) найти наибольший элемент и поменять его местами с соответствующим диагональным элементом.

Вариант № 25

Матрица **L**(n, k) состоит из нулей и единиц. Найти в ней самую длинную цепочку стоящих подряд нулей по горизонтали, вертикали или диагонали.

Вариант № 26

Символьную матрицу **A**(m, n) заполнить символами с кодами менее 128. Найти строки, содержащие только английские буквы, или вывести сообщение, что таких строк нет.

Варианты заданий для лабораторной работы № 1.4

Варианты № 1–12

Определите структуру «Студент», поля структуры: Ф.И.О., массив элементов структуры «Дисциплина» (не менее четырех элементов, результаты сдачи сессии); при необходимости можно использовать дополнительные поля (например, число элементов в массиве дисциплин). Структура «Дисциплина» включает в себя поля: название, оценка. Определите недостаток подобного представления данных, если описываются результаты сдачи сессии одной группой, и предложите способы его устранения.

В программе задайте группу студентов (массив переменных структурного типа) и выполните следующие действия.

1. Определите, сколько студентов имеют неудовлетворительную оценку хотя бы по одному предмету.
2. Определите, сколько студентов сдали все экзамены на «5».
3. Определите средний балл группы по заданной дисциплине.
4. Определите количество отличных оценок, полученных группой по всем предметам.
5. Определите, сколько студентов имеют неудовлетворительную оценку по заданному предмету.
6. Определите, сколько студентов имеют средний балл от «4» до «5».
7. Определите, какое количество неудовлетворительных оценок получено по всем предметам.
8. Определите, какой из предметов был сдан группой лучше всего.
9. Определите, сколько студентов не имеют задолженностей.
10. Определите, сколько студентов сдали сессию на «4» и «5».
11. Определите студентов с самым высоким средним баллом (один студент или несколько с одинаковыми баллами).
12. Определите средний балл каждого студента.

Варианты № 13–20

Определите структуру «Житель». Поля: Ф.И.О., переменная структурного типа «Адрес» (поля: улица, номер дома, номер квартиры), пол, возраст.

В программе задайте несколько жителей — фрагмент базы данных ЖЭК (массив переменных структурного типа) и выполните следующие действия.

1. Определите, сколько лиц женского и сколько мужского пола проживают в заданном доме.
2. Определите, сколько лиц мужского пола в возрасте старше 18 и младше 60 лет проживают на одной улице.
3. Определите, сколько лиц женского пола в возрасте старше 30 лет проживают в заданном доме.
4. Определите, сколько детей до 7 лет проживают на одной улице.
5. Определите, сколько лиц мужского и женского пола в возрасте до 50 лет проживают на заданной улице.
6. Определите, сколько детей от 1 года до 5 лет проживают в заданном доме.
7. Определите, сколько лиц мужского пола призывного возраста (от 18 до 27 лет) находится в базе данных.
8. Определите число пенсионеров (мужчины от 60 лет, женщины от 55 лет) в базе данных.

Варианты № 21–27

Определите структуру «Пассажир» и поместите в нее следующую информацию: Ф.И.О. пассажира, массив элементов структуры «Багаж» (поля: название предмета багажа, вес предмета в килограммах), при необходимости можно использовать дополнительные поля (например, число элементов в массиве багажа). В программе задайте несколько пассажиров (массив переменных структурного типа) и выполните следующие действия.

1. Определите число пассажиров, вес багажа которых превышает 30 кг.
2. Определите, имеется ли пассажир, багаж которого состоит из одной вещи массой более 20 кг.
3. Определите средний вес багажа для пассажира.
4. Определите количество пассажиров, вес багажа которых превосходит средний.
5. Определите количество пассажиров, имеющих более трех вещей в багаже.
6. Определите средний вес одного предмета багажа для каждого пассажира.
7. Определите пассажира с максимальным весом багажа.

Варианты заданий для лабораторной работы № 1.5

Вариант № 1

Решить задачу, используя функцию.

Дано четное число $n > 2$. Проверить для него гипотезу Гольдбаха, состоящую в том, что каждое четное число представимо в виде суммы двух простых чисел. Функция должна проверять, является целое число простым или нет (простым является число, если оно делится без остатка только на себя и на 1).

Вариант № 2

Решить задачу, используя функцию.

Дана целочисленная матрица размера $n \times m$. Преобразовать ее, переставив строки в порядке возрастания их наименьших элементов. Функция должна возвращать значение наименьшего элемента в одномерном массиве.

Вариант № 3

Решить задачу, используя функцию.

Дана матрица размера $n \times m$. Преобразовать ее, упорядочив каждую строку по неубыванию элементов. Функция должна сортировать одномерный массив в порядке неубывания его элементов.

Вариант № 4

Решить задачу, используя функцию.

Дана квадратная целочисленная матрица a порядка n. Выделяя на главной диагонали последовательно по одному элементу a_{ii}, можно получить n матриц, ограниченных элементами a_{11} и a_{ii}. Программа должна сформировать вектор, элементами которого являются наибольшие элементы всех таких матриц. Функция должна возвращать значение наибольшего элемента в матрице размерности n.

Вариант № 5

Решить задачу, используя функцию.

Определить количество простых чисел в целочисленном массиве $C(n)$. Функция должна проверять, является целое число простым или

нет (простым является число, если оно делится без остатка только на себя и на 1).

Вариант № 6

Решить задачу, используя функцию.

Дана вещественная матрица **A**(n, m). Упорядочить матрицу по невозрастанию сумм элементов ее строк. Функция должна возвращать значение суммы элементов одномерного массива.

Вариант № 7

Решить задачу, используя функцию.

Дана символьная матрица размера $n \times m$. Преобразовать ее таким образом, чтобы элементы каждой нечетной строки расположились в обратном порядке. Новую матрицу не заводить. Функция должна переписывать одномерный символьный массив в обратном порядке.

Вариант № 8

Решить задачу, используя функцию.

Даны три матрицы разных порядков ($n_1 \times m_1$, $n_2 \times m_2$, $n_3 \times m_3$). Найти сумму их наименьших элементов (считая, что в каждой матрице такой элемент единственный). Функция должна возвращать значение наименьшего элемента матрицы размерности $n \times m$.

Вариант № 9

Решить задачу, используя функцию.

Дан массив из n натуральных чисел. Определить количество чисел, в десятичной записи которых используется цифра 7. Функция должна проверять, есть ли в десятичной записи целого числа цифра 7 или нет.

Вариант № 10

Решить задачу, используя функцию.

Найдите наибольший общий делитель (НОД) трех натуральных чисел, используя соотношение НОД(a, b, c) = НОД (НОД(a, b), c). Функция должна находить НОД двух чисел по алгоритму Евклида.

Алгоритм Евклида

1. Вычислим r — остаток от деления числа a на b ($a > b$), $a = bq+r$, $0 \le r < b$.

2. Если $r = 0$, то b есть искомое число (НОД).

3. Если $r! = 0$, то заменим пару чисел *(a,b)* парой *(b,r),* и перейдем к шагу 1.

Вариант № 11

Решить задачу, используя функцию.

Дана матрица **A**(n, m). Определить суммы элементов тех строк матрицы, максимальные элементы которых не превышают среднего значения элементов матрицы, или вывести сообщение, что таких строк не существует. Функция должна вычислять значение суммы элементов одномерного массива, значение максимального элемента одномерного элемента либо среднее значение элементов матрицы (можно использовать сразу три функции).

Вариант № 12

Решить задачу, используя функцию.

Для заданного m получить таблицу первых m простых чисел. Функция должна проверять, является целое число простым или нет (простым является число, если оно делится без остатка только на себя и на 1).

Вариант № 13

Решить задачу, используя функцию.

Дан массив из n натуральных чисел. Определить количество чисел, десятичная запись которых не содержит одинаковых цифр. Функция должна проверять, содержит ли десятичная запись числа одинаковые цифры или нет.

Вариант № 14

Решить задачу, используя функцию.

В вещественном массиве размерности n найти все числа, у которых старшая значащая десятичная цифра есть 9 (числа сильно различаются по величине). Функция должна проверять для вещественного числа, является старшая значащая десятичная цифра 9 или нет.

Вариант № 15

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности n. Для каждого числа переставьте его десятичные числа так, чтобы получить максимально

возможное число, записанное теми же цифрами. Функция должна переставлять цифры одного натурального числа, возвращаемое значение не использовать (тип void).

Примечание: достаточно цифры числа отсортировать по невозрастанию.

Вариант № 16

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности *n*. Определить количество чисел-палиндромов в нем. Палиндром («перевертыш») – число, десятичная запись которого читается одинаково слева направо и справа налево (например, 14541). Функция должна проверять, является число палиндромом или нет.

Вариант № 17

Решить задачу, используя функцию.

Получить все шестизначные «счастливые» номера и подсчитать их количество. Функция должна проверять, является число «счастливым» или нет. «Счастливым» называется такое шестизначное число, у которого сумма первых трех цифр равна сумме последних трех.

Вариант № 18

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности *n*. Отсортировать массив в порядке неубывания сумм цифр десятичной записи чисел. Функция должна считать сумму десятичных цифр натурального числа.

Вариант № 19

Решить задачу, используя функцию.

Дата некоторого дня характеризуется тремя натуральными числами (число, месяц, год). Определите количество дней, прошедших между двумя датами (учитывать високосные годы, даты начинаются с 1970 г). Функция должна считать, сколько дней прошло с 1 января 1970 г. до текущей даты.

Вариант № 20

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности *n*. Отсортировать его в порядке неубывания количества единиц в двоичной записи чисел. Функция должна считать количество единиц в двоичной записи числа.

Вариант № 21

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности n. Вывести на печать все числа этого массива, являющиеся полными квадратами. Функция должна проверять, является число полным квадратом или нет.

Вариант № 22

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности n. Вывести на печать все числа этого массива, являющиеся степенями пяти. Функция должна проверять, является число степенью пяти или нет.

Вариант № 23

Решить задачу, используя функцию.

Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов», не превышающих заданного значения a. Функция должна проверять, является целое число простым или нет (простым является число, если оно делится без остатка только на себя и на 1).

Вариант № 24

Решить задачу, используя функцию.

Дан массив натуральных чисел размерности n. Найти число в массиве, имеющее максимальную сумму цифр в десятичном представлении. Если существует несколько чисел с одинаковой максимальной суммой, вывести на печать их все. Функция должна возвращать сумму цифр числа в десятичном представлении.

Вариант № 25

Решить задачу, используя функцию.

Даны два натуральных числа a и b – числитель и знаменатель дроби. Сократить дробь, разделив числа на их наибольший общий делитель (НОД). Функция должна находить НОД двух чисел по алгоритму Евклида.

Алгоритм Евклида

1. Вычислим r — остаток от деления числа a на b ($a > b$), $a = bq + r$, $0 \le r < b$.

2. Если $r = 0$, то b есть искомое число (НОД).

3. Если $r! = 0$, то заменим пару чисел (a, b) парой (b, r), и перейдем к шагу 1.

Вариант № 26

Решить задачу, используя функцию.

Составить программу для нахождения общего числа некоторой буквы (буква вводится с клавиатуры) в нескольких предложениях (предложение задается строкой в языке Си). Функция считает количество появлений заданной буквы в строке.

Вариант № 27

Решить задачу, используя функцию.

Даны три квадратных уравнения:

$$ax^2 + bx + c = 0;$$

$$bx^2 + ax + c = 0;$$

$$cx^2 + ax + b = 0.$$

Сколько из них имеют вещественные корни? Функция должна проверять, имеет квадратное уравнение вещественные корни или нет.

Варианты заданий для лабораторной работы № 1.6

В приложении организовать список объектов и сортировку списка (табл. 1).

Таблица 1

Варианты заданий

Параметры приложений			Метод сортировки		
Объект	Поля	Сортировка	Вставка	Прямой выбор	Обмен
«Сотрудник»	Ф.И.О., дата приема на работу, должность, базовый оклад	По Ф.И.О.	1	2	3
		По окладу	4	5	6
«Банковский вклад»	Название, сумма вклада, тип валюты, ставка (в процентах годовых)	По сумме вклада	7	8	9
		По названию	10	11	12
«Студент»	Ф.И.О., группа, номер зачетной книжки, массив четырех оценок за сессию	По Ф.И.О.	13	14	15
		По среднему баллу	16	17	18
«Книга»	Ф.И.О. автора, название, издательство, год издания, количество страниц	По Ф.И.О.	19	20	21
		По названию	22	23	24
«Автомобиль»	Название модели, массив расхода топлива из трех элементов (на трассе, в городе, смешанный), максимальная скорость, мощность	По названию модели	25	26	27
		По смешанному расходу топлива	28	29	30

Варианты заданий для лабораторной работы № 1.7

Таблица 2

Варианты заданий

Параметры приложений				Файл открывается в режиме	
Структура	Поля	Объекты задаются в приложениях		текстовом	двоичном
		первом	втором		
«Студент»	Ф.И.О., группа, номер зачетной книжки, массив четырех оценок за сессию	Списком	Массивом	1	2
		Массивом	Списком	3	4
«Книга»	Ф.И.О. автора, название, издательство, год издания, количество страниц	Массивом	Списком	5	6
		Списком	Массивом	7	8
«Автомобиль»	Модель, массив расхода топлива из трех элементов (на трассе, в городе, смешанный), максимальная скорость, мощность	Списком	Массивом	9	10
		Массивом	Списком	11	12
«Сотрудник»	Ф.И.О., дата приема на работу, должность, базовый оклад	Массивом	Списком	13	14
		Списком	Массивом	15	16
«Банковский вклад»	Название, сумма вклада, тип валюты, ставка в процентах годовых	Списком	Массивом	17	18
		Массивом	Списком	19	20
«Компьютер»	Тип процессора, тактовая частота, объем оперативной памяти, размер жесткого диска, тип видеокарты	Массивом	Списком	21	22
		Списком	Массивом	23	24
«Программное средство криптозащиты»	Название разработчика, номер версии, название реализуемого алгоритма, наличие лицензии ФСБ — да, нет	Списком	Массивом	25	26
		Массивом	Списком	27	28

Варианты заданий для лабораторной работы № 1.8

Разработайте приложение для рисования в окне с помощью мыши (функции простого графического редактора). При рисовании фигур (кроме кругов по следу курсора мыши при перемещении мыши с нажатой клавишей) начальная точка фигуры (конец отрезка, вершина прямоугольника) определяется координатами курсора мыши в момент нажатия клавиши мыши, конечная точка фигуры (другой конец отрезка, противоположная вершина прямоугольника и т. п.) определяется координатами курсора мыши в момент отпускания клавиши мыши. Рисование происходит при перемещении курсора мыши с нажатой клавишей. При рисовании овалов, вписанных в прямоугольники, сами прямоугольники не рисуются. При рисовании кругов по следу курсора при перемещении мыши с нажатой клавишей диаметр рисуемых кругов не менее 50 пикселей.

Варианты заданий и их параметры представлены в табл. 3.

Таблица 3

Варианты заданий

Номер варианта	Фигура	Цвет		Тип границы	Клавиша мыши
		Граница	Закраска		
1	Круги по следу курсора	Синяя	Красная	Сплошная	Левая
2		Красная	Зеленая	Пунктирная	Правая
3	Прямоугольник	Синяя	Красная	Сплошная	Левая
4		Красная	Зеленая	Пунктирная	Правая
5	Овал	Синяя	Красная	Сплошная	Левая
6		Красная	Зеленая	Пунктирная	Правая
7	Отрезок	Синяя	—	Сплошная	Левая
8		Красная	—	Пунктирная	Правая
9	Круги по следу курсора	Зеленая	Желтыми линиями по горизонтали	Сплошная	Левая
10		Синяя	Зелеными линиями по вертикали	Точечная	Правая

Продолжение табл. 3

Номер варианта	Фигура	Цвет		Тип границы	Клавиша мыши
		Граница	Закраска		
11	Прямоугольник	Желтая	Фиолетовая	Штрихпунктирная	Левая
12		Зеленая	Красная	Штрихдипунктирная	Правая
13	Овал	Фиолетовая	Красными линиями по вертикали и горизонтали	Штрихпунктирная	Левая
14		Зеленая	Красными линиями по вертикали	Штрихдипунктирная	Правая
15	Отрезок	Зеленая	—	Штрихдипунктирная	Левая
16		Желтая	—	Штрихпунктирная	Правая
17	Отрезок	Синяя	—	Сплошная	Левая
18		Красная	—	Пунктирная	Правая
19	Круги по следу курсора	Синяя	Красная	Сплошная	Левая
20		Красная	Зеленая	Пунктирная	Правая
21	Прямоугольник	Зеленая	Желтыми линиями по горизонтали	Сплошная	Левая
22		Синяя	Зелеными линиями по вертикали	Пунктирная	Правая
23	Овал	Желтая	Фиолетовая	Сплошная	Левая
24		Зеленая	Красная	Пунктирная	Правая
25	Отрезок	Фиолетовая	—	Сплошная	Левая
26		Зеленая	—	Точечная	Правая

Окончание табл. 3

Номер варианта	Фигура	Цвет		Тип границы	Клавиша мыши
		Граница	Закраска		
27	Круги по следу курсора	Зеленая	Красными линиями по вертикали и горизонтали	Штрихпунктирная	Левая
28		Желтая	Красными линиями по вертикали	Штрих-дипунктирная	Правая
29	Прямоугольник	Синяя	Красная	Штрихпунктирная	Левая
30		Красная	Зеленая	Штрих-дипунктирная	Правая
31	Овал	Синяя	Красная	Штрих-дипунктирная	Левая
32		Красная	Зеленая	Штрихпунктирная	Правая

Варианты заданий для лабораторной работы № 1.9

Разработать приложение на базе диалогового окна — калькулятор. Калькулятор может иметь примерный вид, представленный на рис. 1.11 (или другой с требуемыми функциями). При нажатии соответствующей кнопки результат помещается в текстовое поле и проведенная операция добавляется в список проведенных операций. Следует предусмотреть кнопку для очистки списка.

Вариант № 1

Разработать калькулятор для вычисления операции объединения двух множеств. Множество является множеством целых чисел, которые вводятся в текстовое поле через пробелы.

Вариант № 2

Разработать калькулятор для вычисления операции пересечения двух множеств. Множество является множеством целых чисел, которые вводятся в текстовое поле через пробелы.

Вариант № 3

Разработать калькулятор для вычисления операции разности двух множеств. Множество является множеством целых чисел, которые вводятся в текстовое поле через пробелы.

Вариант № 4

Разработать калькулятор для вычисления операции симметричной разности двух множеств. Множество является множеством целых чисел, которые вводятся в текстовое поле через пробелы.

Вариант № 5

Разработать калькулятор для вычисления двух арифметических операций (+, –).

Вариант № 6

Разработать калькулятор для получения целой части вещественного числа и округления вещественного числа до ближайшего целого.

Вариант № 7

Разработать калькулятор для вычисления остатка от деления и целочисленного деления целых чисел.

Вариант № 8

Разработать калькулятор для длины вектора на плоскости, заданного координатами x, y. Начало вектора находится в точке с координатами (0, 0).

Вариант № 9

Разработать калькулятор для перевода целых чисел, вводимых в десятичной системе счисления, в шестнадцатеричную систему счисления.

Вариант № 10

Разработать калькулятор для перевода целых чисел, вводимых в десятичной системе счисления, в восьмеричную систему счисления.

Вариант № 11

Разработать калькулятор для перевода целых чисел, вводимых в десятичной системе счисления, в двоичную систему счисления.

Вариант № 12

Разработать калькулятор для вычисления площади треугольника. Треугольник задается длинами трех сторон (использовать формулу Герона).

Вариант № 13

Разработать калькулятор для вычисления площади параллелограмма. Параллелограмм задается длинами двух сторон и углом между ними в градусах.

Вариант № 14

Разработать калькулятор для вычисления площади эллипса. Эллипс задается длинами большой и малой полуосей.

Вариант № 15

Разработать калькулятор для вычисления площади треугольника. Треугольник задается длинами двух сторон и углом между ними в градусах.

Вариант № 16

Разработать калькулятор для вычисления двух арифметических операций (*, /).

Вариант № 17

Разработать калькулятор для вычисления тригонометрических функций sin, cos (угол задается в градусах).

Вариант № 18

Разработать калькулятор для вычисления тригонометрических функций tg, ctg (угол задается в градусах).

Вариант № 19

Разработать калькулятор для вычисления квадратного корня и возведения аргумента в любую степень.

Вариант № 20

Разработать калькулятор для вычисления десятичного и натурального логарифмов.

Вариант № 21

Разработать калькулятор для вычисления экспоненты и функций $2x$.

Вариант № 22

Разработать калькулятор для вычисления операций сложения и вычитания двух комплексных чисел (комплексные числа задаются действительной и мнимой частями).

Вариант № 23

Разработать калькулятор для вычисления тригонометрических функций arcsin, arccos (угол выдается в градусах).

Вариант № 24

Разработать калькулятор для вычисления тригонометрических функций arctg, arcctg (угол выдается в градусах).

Вариант № 25

Разработать калькулятор для вычисления логарифма по любому допустимому основанию.

Вариант № 26

Разработать калькулятор для вычисления поразрядных операций «И» и «ИЛИ» над целыми числами.

Вариант № 27

Разработать калькулятор для вычисления поразрядных операций «НЕ» и «исключающее ИЛИ» над целыми числами.

Варианты заданий для лабораторной работы № 2.1

Во всех вариантах требуется следующее: описать класс, включающий заданные поля и методы (функции)[9] разработать программу, которая создает массив объектов (размерность *n* массива вводится с клавиатуры) и выполняет требуемые действия.

Вариант № 1

Класс «Аппаратно-программное средство защиты (СЗ) от несанкционированного доступа» (НСД). Параметры (поля класса) — название, номер класса защищенности от НСД (Существует семь классов защищенности от НСД, наивысший — 1-й, самый низкий — 7-й. Например, если требуется обеспечить защищенность по 3-му классу, можно использовать СЗ с классами 1, 2 или 3.) Статус доступа всех полей private. Класс включает в себя: конструктор; при необходимости — функции доступа к полям; функцию, проверяющую, можно ли это СЗ использовать для заданного класса (номер заданного класса — «Параметр функции»); функцию печати параметров СЗ. Вывести на печать параметры тех СЗ, которые можно использовать для заданного класса защищенности; номер класса защищенности вводится с клавиатуры.

Вариант № 2

Класс «Криптографический метод защиты информации». Параметры (поля класса): название, тип (симметричный или несимметричный). Статус доступа всех полей private. Класс включает в себя: конструктор,

при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать все методы заданного типа; тип вводится с клавиатуры.

Вариант № 3

Класс «Прямоугольник». Параметры (поля класса): длина, ширина. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию вычисления площади. Вывести на печать все параметры прямоугольников, площади которых превышают заданное значение, которое вводится с клавиатуры.

Вариант № 4

Класс «Сотрудник предприятия». Параметры (поля класса): Ф.И.О., оклад, надбавка к окладу в процентах. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию вычисления зарплаты (зарплата = оклад + надбавка в процентах оклада). Вывести на печать параметры всех сотрудников и их суммарную зарплату.

Вариант № 5

Класс «Автомобиль». Параметры (поля класса): марка, максимальная скорость (км/ч). Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать параметры тех автомобилей, максимальная скорость которых превышает заданное значение, введенное с клавиатуры.

Вариант № 6

Класс «Студент». Параметры (поля класса): Ф.И.О., массив из четырех оценок за последнюю сессию. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию проверки возможности получения студентом стипендии (все оценки без троек). Вывести на печать список всех студентов, получающих стипендию.

Вариант № 7

Класс «Квадратное уравнение» $ax^2 + bx + c = 0$ ($a \neq 0$). Параметры (поля класса): *a, b, c*. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию расчета дискриминанта. Вывести на печать параметры тех уравнений, которые имеют вещественные корни.

Вариант № 8

Класс «Полином» $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 (a_n \neq 0)$.Параметры (поля класса): *n*, массив коэффициентов a_n, a_{n-1}, ... a_0. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию вычисления значения полинома при заданном *x* (*x* — параметр функции). Вывести на печать параметры всех полиномов и сумму их значений при заданном значении *x*, вводимом с клавиатуры.

Вариант № 9

Класс «Книга». Параметры (поля): автор, название, количество страниц. Статус доступа всех полей private. Класс включает в себя: конструктор, функцию печать параметров, при необходимости — функции доступа к полям. Вывести на печать параметры книги с максимальным количеством страниц.

Вариант № 10

Класс «Межсетевой экран» (МЭ). Параметры (поля класса): название, номер класс защищенности. (Существует пять классов защищенности МЭ, наивысший — 1-й, самый низкий — 5-й. Например, если требуется использовать МЭ 3-го класса защищенности, то можно использовать МЭ с классами 1, 2 или 3.) Статус доступа всех полей private. Класс включает в себя: конструктор; при необходимости — функции доступа к полям; функцию, проверяющую, можно ли МЭ использовать для заданного класса (номер заданного класса — параметр функции); функцию печати параметров МЭ. Вывести на печать параметры тех МЭ, которые можно использовать для заданного класса защищенности; номер класса защищенности вводится с клавиатуры.

Вариант № 11

Класс «Персональный компьютер» (ПК). Параметры (поля класса): название процессора, тактовая частота. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать параметры всех компьютеров в порядке невозрастания тактовой частоты.

Вариант № 12

Класс «Банковский вклад». Параметры (поля класса): Ф.И.О. владельца, текущая сумма, годовая процентная ставка (проценты начисляются ежегодно с капитализацией начисленных процентов с основной суммой). Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров, функцию расчета суммы на счету через заданное число лет (число лет — параметр функции). Вывести на печать параметры всех вкладов и суммарную сумму на счетах через заданное число лет, которое вводится с клавиатуры.

Вариант № 13

Класс «Программа-антивирус». Параметры (поля класса): название, количество вредоносных программ в базе. Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать параметры антивирусов и антивирус с самой большой базой вредоносных программ.

Вариант № 14

Класс «Вещественное число», записанное с точкой. Параметры: его значение (вещественный тип) и запись (строковое представление числа в десятичной системе счисления с точкой). Статус доступа всех полей private. Класс включает в себя: конструктор, при необходимости — функции доступа к полям, функцию, определяющую количество цифр в целой части числа в десятичной записи, функцию печати параметров. Напечатать все числа, сумму введенных чисел и суммарное количество цифр в целых частях всех чисел.

Вариант № 15

Класс «Предложение». Параметры: массив слов ($n < 10$) и их количество. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую количество слов длиннее пяти букв, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры предложений и процент слов длиннее пяти букв в заданном тексте.

Вариант № 16

Класс «Выражение», состоящее из целых чисел и знаков операций (скобок нет). Параметры: массив значений чисел ($n < 10$), количество чисел и массив знаков операций (тип char). Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, вычисляющую результат (приоритеты операций не учитывать, считать приоритет операций одинаковым), при необходимости — функции доступа к полям, функцию печати параметров. Ввести несколько выражений и вывести результаты в порядке, обратном вводу.

Вариант № 17

Класс «Товар в магазине». Параметры: наименование, количество и стоимость. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую суммарную стоимость товара, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры всех товаров и суммарную стоимость всех товаров в магазине.

Вариант № 18

Класс «Товар в магазине». Параметры: наименование, количество и закупочная цена. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую стоимость товара исходя из заданного процента прибыли (процент прибыли — параметр функции), при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры всех товаров и суммарную стоимость всех товаров в магазине с учетом заданного процента прибыли, который вводится с клавиатуры.

Вариант № 19

Класс «Студент». Параметры (поля): Ф.И.О., массив экзаменационных оценок ($m = 4$). Статус доступа всех полей private. Класс включает в себя: конструктор и функцию определения среднего балла, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры всех студентов и трех самых сильных студентов группы.

Вариант № 20

Класс «Ангар». Параметры (поля): ширина и длина. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую площадь помещения, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры ангаров и площадь склада, состоящего из этих ангаров.

Вариант № 21

Класс «Квартира». Параметры (поля): общая площадь и стоимость 1 м^2. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую стоимость квартиры, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры всех квартир и все квартиры, стоимость которых не превышает заданную сумму (сумма вводится с клавиатуры).

Вариант № 22

Класс «Квартира». Параметры (поля): стоимость и количество комнат. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую среднюю стоимость одной комнаты, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры всех квартир и все квартиры, стоимость одной комнаты в которых не превышает заданную сумму (сумма вводится с клавиатуры).

Вариант № 23

Класс «Выставочные экспонаты». Параметры (поля): название, время экспонирования (в днях), стоимость одного дня экспонирования. Статус доступа всех полей private. Класс включает в себя: конструктор

и функцию определения стоимости экспонирования, при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать параметры экспонатов и экспонат, стоимость экспонирования которых максимальна.

Вариант № 24

Класс «Книга». Параметры (поля): автор, название, количество экземпляров и количество желающих ее прочитать читателей. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию определения средней длины очереди на чтение каждого экземпляра, при необходимости — функции доступа к полям, функцию печати параметров. Напечатать параметры книг и наиболее читаемую книгу в библиотеке.

Вариант № 25

Класс «Выражение», состоящее из целых чисел и знаков операций (скобок нет). Параметры (поля): строка, содержащая выражение. Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую количество операций, при необходимости — функции доступа к полям. Ввести несколько выражений и определить суммарное количество операций в них.

Вариант № 26

Класс «Вектор на плоскости». Параметры (поля): координаты конца вектора: x, y (начало вектора в точке с координатами 0, 0). Статус доступа всех полей private. Класс включает в себя: конструктор, функцию печати параметров, при необходимости — функции доступа к полям, функцию вычисления длины вектора. Вывести на печать все вектора и вектор с наибольшей длиной.

Вариант № 27

Класс «Скаковая лошадь». Параметры: кличка и массив рекордов, содержащий пять лучших результатов, показанных лошадью на скачках (результат определяется временем). Статус доступа всех полей private. Класс включает в себя: конструктор и функцию, определяющую среднее время, показанное лошадью, при необходимости — функции доступа к полям, функцию печати параметров. Вывести на печать параметры лошадей и среднее время по всей конюшне.

Варианты заданий для лабораторной работы № 2.2

Дан класс (например, с именем *Vector*), задающий вектор размерности *n*. Поля класса: указатель на массив, задающий вектор (тип элемента *int* или *double*, в зависимости от варианта), — массив должен создаваться динамически, число элементов (размерность) вектора (тип *int*). Класс включает в себя: конструктор без параметров, задающий пустой вектор (число элементов равно 0), конструктор, создающий объект «вектор» на основе обычного одномерного массива размерности *n*, деструктор.

Необходимо перегрузить операции и продемонстрировать их работу. Перегрузить операцию «[]» (обращение к элементу вектора по индексу) и операцию «=» (копирование вектора или создание копии вектора). Остальные перегружаемые операции выбрать в соответствии со своим вариантом. Варианты заданий представлены в табл. 4.

Таблица 4

Варианты заданий

Номер варианта	Операция	Типы операндов и результата для перегруженной операции			Тип элемента вектора	Описание назначения перегруженной операции
		Первый операнд	Второй операнд	Результат		
1	+	Vector&	Vector&	Vector&	double	Сложение векторов одинаковой размерности; на выходе вектор такой же размерности, элемент которого равен сумме соответствующих элементов двух векторов
2	+	Vector&	double *	Vector&	double	
3	+	double *	Vector&	Vector&	double	
4	+	Vector&	Vector&	Vector&	double	Сложение векторов; на выходе вектор, длина которого равна сумме длин векторов; вначале идут элементы первого вектора, затем второго; если один из векторов задан обычным массивом, то считать, что его длина равна длине вектора, заданного объектом класса
5	+	Vector&	double *	Vector&	double	
6	+	double *	Vector&	Vector&	double	
7	*	Vector&	double	Vector&	double	Умножение вектора на число; на выходе вектор такой же размерности, каждый элемент которого равен произведению соответствующего элемента исходного вектора на число
8	*	double	Vector&	Vector&	double	
9	++	Vector&	_____	Vector& (тот же объект)	int	Каждый элемент исходного вектора увеличивается на 1; оператор-функция должна возвращать ссылку на тот же объект
10	*	Vector&	Vector&	double	double	Скалярное произведение векторов (одинаковой размерности); на выходе значение этого произведения
11	*	Vector&	double *	double	double	
12	*	double *	Vector&	double	double	

Продолжение табл. 4

Номер варианта	Операция	Типы операндов и результата для перегруженной операции			Тип элемента вектора	Описание назначения перегруженной операции
		Первый операнд	Второй операнд	Результат		
13	^	Vector&	Vector&	Vector&	int	Побитовая операция с двумя векторами одинаковой размерности, на выходе вектор такой же размерности, элемент которого равен результату побитовой операции ^ соответствующих элементов двух векторов
14	^	Vector&	int *	Vector&	int	
15	^	int *	Vector&	Vector&	int	
16	--	Vector&	_____	Vector& (тот же объект)	int	Каждый элемент исходного вектора уменьшается на 1, оператор-функция должна возвращать ссылку на тот же объект
17	&	Vector&	Vector&	Vector&	int	Побитовая операция с двумя векторами одинаковой размерности; на выходе вектор такой же размерности, элемент которого равен результату побитовой операции & соответствующих элементов двух векторов
18	&	Vector&	int *	Vector&	int	
19	&	int *	Vector&	Vector&	int	
20	|	Vector&	Vector&	Vector&	int	Побитовая операция с двумя векторами одинаковой размерности; на выходе вектор такой же размерности, элемент которого равен результату побитовой операции | соответствующих элементов двух векторов
21	|	Vector&	int *	Vector&	int	
22	|	int *	Vector&	Vector&	int	
23	<<	ostream&	Vector&	ostream&	double	Поразрядный сдвиг влево; реализовать для вставки всех элементов вектора в поток вывода через пробелы; вначале в поток записывается размерность вектора, затем элементы

Окончание табл. 4

Номер варианта	Операция	Типы операндов и результата для перегруженной операции			Тип элемента вектора	Описание назначения перегруженной операции
		Первый операнд	Второй операнд	Результат		
24	>>	istream&	Vector&	istream&	double	Поразрядный сдвиг вправо; реализовать для чтения всех элементов вектора из потока ввода; вначале из потока читается размерность вектора, затем элементы
25	<<	Vector&	int	Vector&	int	Поразрядный сдвиг влево; реализовать для циклического сдвига всех элементов вектора влево на заданное число позиций (число позиций определяет второй операнд операции)
26	>>	Vector&	int	Vector&	int	Поразрядный сдвиг вправо; реализовать для циклического сдвига всех элементов вектора вправо на заданное число позиций (число позиций определяет второй операнд операции)

Варианты заданий для лабораторной работы № 2.3

Вариант № 1

Создать базовый класс «Вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`), определяющей конец вектора (начало вектора находится в точке с координатами 0; 0); конструктор для инициализации полей; функция для вычисления длины вектора, функция для печати полей и длины вектора. Создать производный класс «Вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для вычисления длины вектора; переопределенная функция для печати полей и длины вектора. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 2

Создать базовый класс «Точка на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`); конструктор для инициализации полей; функция для печати значений полей. Создать производный класс «Точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 3

Создать базовый класс «Квадрат». Элементы класса: поле, задающее длину стороны (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления площади квадрата; функция для печати поля и площади квадрата. Создать производный класс «Куб». Элементы класса: конструктор для инициализации поля; переопреде-

ленная функция для вычисления объема (вместо площади) куба (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 4

Создать базовый класс «Прямоугольник». Элементы класса: поля, задающие длины сторон (статус доступа `protected`); конструктор для инициализации полей; функция для вычисления площади прямоугольника; функция для печати полей и значения площади. Создать производный класс «Прямоугольный параллелепипед». Элементы класса: дополнительное поле, задающее высоту; конструктор для инициализации полей; переопределенная функция для вычисления объема (вместо площади) прямоугольного параллелепипеда (внутри переопределенной функции должна вызываться функция из базового класса); переопределенная функция для печати полей и значения объема. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 5

Создать базовый класс «Круг». Элементы класса: поле, задающее радиус; конструктор для инициализации поля (статус доступа `protected`); функция для вычисления площади круга (площадь круга πr^2); функция для печати полей и площади. Создать производный класс «Шар». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема (вместо площади круга) шара (площадь шара $4/3\pi r^3$). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 6

Создать базовый класс «Автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа `protected`); конструктор

для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «Грузовой автомобиль». Элементы класса: дополнительно поле, содержащее грузоподъемность автомобиля в тоннах; конструктор для инициализации полей; переопределенная функция печати параметров автомобиля (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 7

Создать базовый класс «Вещественное число». Элементы класса: поле, задающее значение числа (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления модуля числа; функция для печати поля и модуля числа. Создать производный класс «Комплексное число». Элементы класса: дополнительно поле, задающее значение мнимой части числа; конструктор для инициализации полей; переопределенная функция для вычисления модуля числа (модуль числа — корень квадратный из суммы квадратов вещественной и мнимой частей числа); переопределенная функция для печати полей и модуля числа. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 8

Создать базовый класс «Вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для печати координат вектора. Создать производный класс «Вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати координат вектора (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 9

Создать базовый класс «Квадрат». Элементы класса: поле, задающее длину стороны (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления периметра квадрата; функция для печати длины стороны и периметра. Создать производный класс «Прямоугольник». Элементы класса: дополнительное поле, задающее другую сторону; конструктор для инициализации полей; переопределенная функция для вычисления периметра прямоугольника; переопределенная функция для печати длин сторон и периметра. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 10

Создать базовый класс «Автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа `protected`); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «Автобус». Элементы класса: дополнительно поле, содержащее максимальное число перевозимых пассажиров; конструктор для инициализации полей; переопределенная функция печати параметров автобуса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 11

Создать базовый класс «Школа». Элементы класса: поле, содержащее название школы; поле, содержащее количество обучаемых в школе (статус доступа `protected`); конструктор для инициализации полей; функция для печати параметров школы. Создать производный класс «Специализированная школа». Элементы класса: дополнительно поле, содержащее название специализации школы; конструктор для инициализации полей; переопределенная функция печати параметров школы (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций

обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 12

Создать базовый класс «Круг». Элементы класса: поле, содержащее значение радиуса круга (статус доступа `protected`); конструктор для инициализации поля; функция для печати радиуса круга. Создать производный класс «Эллипс». Элементы класса: дополнительно поле, содержащее значение второй полуоси эллипса (для задания первой полуоси использовать наследуемое поле радиуса круга); конструктор для инициализации полей; переопределенная функция печати параметров эллипса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 13

Создать базовый класс «Сотрудник предприятия». Компоненты класса – поля: Ф.И.О., оклад, надбавка за стаж (в процентах оклада за 1 год), стаж (в годах), статус доступа полей `protected`; конструктор для инициализации полей; функция для вычисления зарплаты; функция для печати параметров сотрудника. Создать производный класс «Начальник подразделения». Дополнительные поля: процентная надбавка к окладу за выполнение обязанностей начальника, название подразделения. Переопределить функцию для исчисления зарплаты и функцию для печати параметров начальника. Внутри переопределенных функций вызывать соответствующие функции из базового класса. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 14

Создать базовый класс «Счет в банке». Компоненты класса — поля: Ф.И.О. владельца, начальная сумма счета, ставка вклада (проценты в год), время существования вклада (в годах), статус доступа полей `protected`; конструктор для инициализации полей; функция для вычисления суммы на счете с учетом начисленных процентов за время

существования вклада; функция для печати параметров счета. Создать производный класс «Привилегированный счет». Дополнительное поле: процент кредита, предоставляемого по счету (проценты доступной на счете суммы с учетом времени существования вклада). Переопределенная функция для исчисления суммы на счете с учетом доступного кредита. Переопределенная функция для печати параметров счета. Внутри переопределенных функций вызывать соответствующие функции из базового класса. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

Вариант № 15

Создать базовый класс «Вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для вычисления длины вектора; функция для печати полей и длины вектора. Создать производный класс «Вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для вычисления длины вектора; переопределенная функция для печати полей и длины вектора. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 16

Создать базовый класс «Точка на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`); конструктор для инициализации полей; функция для печати значений полей. Создать производный класс «Точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 17

Создать базовый класс «Квадрат». Элементы класса: поле, задающее длину стороны (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления площади квадрата; функция для печати поля и площади квадрата. Создать производный класс «Куб». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема (вместо площади) куба (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 18

Создать базовый класс «Прямоугольник». Элементы класса: поля, задающие длины сторон (статус доступа `protected`); конструктор для инициализации полей; функция для вычисления площади прямоугольника; функция для печати полей и значения площади. Создать производный класс «Прямоугольный параллелепипед». Элементы класса: дополнительное поле, задающее высоту; конструктор для инициализации полей; переопределенная функция для вычисления объема (вместо площади) прямоугольного параллелепипеда (внутри переопределенной функции должна вызываться функция из базового класса); переопределенная функция для печати полей и значения объема. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 19

Создать базовый класс «Круг». Элементы класса: поле, задающее радиус; конструктор для инициализации поля (статус доступа `protected`); функция для вычисления площади круга (площадь круга πr^2); функция для печати полей и площади. Создать производный класс «Шар». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема (вместо площади круга) шара (площадь шара $4/3\pi r^3$). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 20

Создать базовый класс «Автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа `protected`); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «Грузовой автомобиль». Элементы класса: дополнительно поле, содержащее грузоподъемность автомобиля в тоннах; конструктор для инициализации полей; переопределенная функция печати параметров автомобиля (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 21

Создать базовый класс «Вещественное число». Элементы класса: поле, задающее значение числа (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления модуля числа; функция для печати поля и модуля числа. Создать производный класс «Комплексное число». Элементы класса: дополнительно поле, задающее значение мнимой части числа; конструктор для инициализации полей; переопределенная функция для вычисления модуля числа (модуль числа — корень квадратный из суммы квадратов вещественной и мнимой частей числа); переопределенная функция для печати полей и модуля числа. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 22

Создать базовый класс «Вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа `protected`), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для печати координат вектора. Создать производный класс «Вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати координат вектора (внутри переопределенной функции должна вызываться функция из базо-

вого класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 23

Создать базовый класс «Квадрат». Элементы класса: поле, задающее длину стороны (статус доступа `protected`); конструктор для инициализации поля; функция для вычисления периметра квадрата; функция для печати длины стороны и периметра. Создать производный класс «Прямоугольник». Элементы класса: дополнительное поле, задающее другую сторону; конструктор для инициализации полей; переопределенная функция для вычисления периметра прямоугольника; переопределенная функция для печати длин сторон и периметра. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 24

Создать базовый класс «Автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа `protected`); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «Автобус». Элементы класса: дополнительно поле, содержащее максимальное количество перевозимых пассажиров; конструктор для инициализации полей; переопределенная функция печати параметров автобуса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 25

Создать базовый класс «Школа». Элементы класса: поле, содержащее название школы; поле, содержащее количество обучаемых в школе (статус доступа `protected`); конструктор для инициализации полей; функция для печати параметров школы. Создать производный класс «Специализированная школа». Элементы класса: дополнительно поле, содержащее название специализации школы; конструктор для иници-

ализации полей; переопределенная функция печати параметров школы (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 26

Создать базовый класс «Круг». Элементы класса: поле, содержащее значение радиуса круга (статус доступа `protected`); конструктор для инициализации поля; функция для печати радиуса круга. Создать производный класс «Эллипс». Элементы класса: дополнительно поле, содержащее значение второй полуоси эллипса (для задания первой полуоси использовать наследуемое поле радиуса круга); конструктор для инициализации полей; переопределенная функция печати параметров эллипса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 27

Создать базовый класс «Сотрудник предприятия». Компоненты класса — поля: Ф.И.О., оклад, надбавка за стаж (в процентах оклада за 1 год), стаж (в годах), статус доступа полей `protected`; конструктор для инициализации полей; функция для исчисления зарплаты; функция для печати параметров работника. Создать производный класс «Начальник подразделения». Дополнительные поля: процентная надбавка к окладу за выполнение обязанностей начальника и название подразделения. Переопределить функцию для исчисления зарплаты и функцию для печати параметров начальника. Внутри переопределенных функций вызывать соответствующие функции из базового класса. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Вариант № 28

Создать базовый класс «Счет в банке». Компоненты класса — поля: Ф.И.О. владельца, начальная сумма счета, ставка вклада (проценты

в год), время существования вклада (в годах), статус доступа полей `protected`; конструктор для инициализации полей; функция для вычисления суммы на счете с учетом начисленных процентов за время существования вклада; функция для печати параметров счета. Создать производный класс «Привилегированный счет». Дополнительное поле: процент кредита, предоставляемого по счету (проценты доступной на счете суммы с учетом времени существования вклада). Переопределенная функция для исчисления суммы на счете с учетом доступного кредита. Переопределенная функция для печати параметров счета. Внутри переопределенных функций вызывать соответствующие функции из базового класса. Создать по одному объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить статический полиморфизм, показать его особенности в программе.

Варианты заданий для лабораторной работы № 2.4

Создать абстрактный класс «Геометрическая фигура» (на экране). Класс содержит следующие поля: координаты геометрического центра фигуры на экране; поле, задающее размер фигуры (например, расстояние от центра до вершины или радиус окружности, в пикселях); поле, задающее угловое положение фигуры; поле, задающее угловую скорость вращения фигуры; поле, определяющее направление движения (возможны два варианта: движение по вертикали и движение по горизонтали); поле, определяющее скорость движения; поле, определяющее цвет фигуры; поле, содержащее хэндл окна для рисования; при необходимости можно включить другие поля. Класс включает в себя: конструктор для инициализации полей, функцию, изменяющую угловое положение фигуры и положение на экране во время движения за один такт времени, и чистую виртуальную функцию (или функции) для рисования и стирания фигуры на экране. На основе абстрактного класса «Фигура» разработать три производных класса, задающих геометрические фигуры своего варианта, варианты представлены в табл. 5. Создать несколько объектов каждого из трех классов (не менее трех), для этого использовать один массив указателей типа базового класса «Фигура». Реализуя механизм полиморфизма, привести объекты классов в одновременное вращение вокруг их центров с различными угловыми ско-

ростями и в движение с отскоком от краев окна в заданном режиме (по горизонтали или по вертикали). При этом использовать обработку сообщений от таймера. Таймер периодически с интервалом в несколько миллисекунд генерирует сообщение, при обработке сообщения стирается старая фигура и рисуется новая на новом месте.

Геометрические фигуры:

1 – отрезок (линия);

2 – окружность с вырезанной четвертью;

3 – правильный многоугольник (количество вершин — поле класса);

4 – ромб (не квадрат);

5 – прямоугольник (не квадрат);

6 – равнобедренный прямоугольный треугольник;

7 – параллелограмм, не являющийся ромбом или прямоугольником.

Таблица 5

Варианты заданий

Номер варианта	Номер фигуры	Номер варианта	Номер фигуры
1	1, 2, 3	15	1, 6, 7
2	1, 2, 4	16	2, 3, 4
3	1, 2, 5	17	2, 3, 5
4	1, 2, 6	18	2, 3, 6
5	1, 2, 7	19	2, 3, 7
6	1, 3, 4	20	2, 4, 5
7	1, 3, 5	21	2, 4, 6
8	1, 3, 6	22	2, 4, 7
9	1, 3, 7	23	3, 4, 5
10	1, 4, 5	24	3, 4, 6
11	1, 4, 6	25	3, 4, 7
12	1, 4, 7	26	3, 5, 6
13	1, 5, 6	27	3, 5, 7
14	1, 5, 7	28	3, 6, 7

Варианты заданий для лабораторной работы № 2.5

Часть 1

Программу, разработанную в лабораторной работе № 2.4, преобразовать следующим образом: для обеспечения движения геометрических фигур в окне вместо таймера использовать потоки. Для каждой геометрической фигуры использовать свой поток. Потоковая функция должна быть одна, объект «Фигура» передается в потоковую функцию через параметр (для передачи дополнительных данных в потоковую функцию в качестве параметра можно задать указатель на структуру). При необходимости использовать синхронизацию.

Часть 2

Необходимо обеспечить синхронизацию двух приложений.

Первое приложение. Приложение с потоками преобразовать так, чтобы движение фигур в потоках начиналось не сразу, а после получения сигнала от второго приложения. При получения сигнала потоки начинают работать до тех пор, пока от второго приложения не придет другой сигнал, при получении второго сигнала потоки завершают свою работу.

Второе приложение — консольное приложение Windows (запускается только при запущенном первом приложении). После нажатия клавиши посылается сигнал для начала работы потоков в первом приложении. После следующего нажатия клавиши посылается сигнал на завершение работы потоков в первом приложении.

Продемонстрировать совместную работу двух приложений.

Варианты заданий для лабораторной работы № 2.6

В заданиях использовать библиотеку классов MFC.

Разработать приложение с меню, в котором содержится несколько элементов (минимум два). Названия элементов соответствуют геометрическим фигурам (например, «Круг», «Квадрат» и т. д.). При выборе элемента меню в окне рисуется соответствующая геометрическая фигура. Обеспечить перерисовку выведенных фигур, например, при сворачивании-разворачивании окна. В этом же приложении обеспечить ввод строки текста с клавиатуры и рисование с помощью мыши, варианты рисуемых с помощью мыши фигур выбрать из табл. 3 (см. с. 212–214), обеспечить перерисовку введенных элементов.

ОГЛАВЛЕНИЕ

Учебное издание

Быков Александр Юрьевич

Решение задач на языках программирования Си и Си++

Редактор *Л.В. Сивай*
Художник *Я.М. Ильина*
Корректор *Л.В. Забродина*
Компьютерная графика *О.В. Левашовой*
Компьютерная верстка *Ю.В. Калиничевой*

Оригинал-макет подготовлен
в Издательстве МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты
Студии Артемия Лебедева.

Подписано в печать 13.04.2017. Формат 60×90 $^{1}/_{16}$.
Усл. печ. л. 15,5. Тираж 50 экз. Изд. № 012-2016. Заказ

Издательство МГТУ им. Н. Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru
www.baumanpress.ru

Отпечатано в типографии МГТУ им. Н. Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
baumanprint@gmail.com