Московский государственный технический университет
имени Н.Э. Баумана

Ю.Е. Алексеев, А.В. Куров

Обработка нечисловых типов данных в среде MS VS C++

Учебное пособие

Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2017

УДК 681.3.06
ББК 32.973-018
А47

Издание доступно в электронном виде на портале *ebooks.bmstu.ru* по адресу: http://ebooks.bmstu.ru/catalog/199/book1618.html

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

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

Рецензенты:
канд. техн. наук *С.М. Авдеева*,
канд. техн. наук, доцент *Т.Н. Ничушкина*

Алексеев, Ю. Е.

А47 Обработка нечисловых типов данных в среде MS VS C++ : учебное пособие по дисциплине «Информатика» / Ю. Е. Алексеев, А. В. Куров. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2017. — 194, [2] с. : ил.

ISBN 978-5-7038-4638-4

Рассмотрена работа со следующими типами и структурами данных: символьным, строковым, структурным, файловым и указателями, описаны операции, которые можно выполнять с данными каждого типа. Приведены сведения о стандартных функциях обработки этих данных и примеры программ, позволяющих лучше уяснить основные особенности работы с каждым конкретным типом данных. Представлены комплекты заданий (не менее 25 вариантов).

Для студентов 1-го курса МГТУ им. Н.Э. Баумана, обучающихся по машино- и приборостроительным специальностям.

УДК 681.3.06
ББК 32.973-018

ISBN 978-5-7038-4638-4

Предисловие

Настоящее издание посвящено изложению основ программирования и решения типовых инженерных задач на языке С в среде VS C++ [1–3]. В данном пособии рассмотрены нечисловые и структурированные типы данных: символы, строки, структуры, указатели, файлы. Поскольку современный специалист при решении практических задач сталкивается с обработкой не только числовых данных, он должен владеть навыками обработки данных и других типов, представленных в используемом языке программирования.

В первой главе пособия рассмотрен символьный тип данных. Авторы знакомят читателя с особенностями обработки символов в языке С, которые состоят, в частности, в том, что символьные данные можно обрабатывать так же, как целые, поскольку символы представляются в памяти ЭВМ своими кодами. Приведены сведения о стандартных функциях обработки символов.

Вторая глава посвящена строкам. Внимание акцентировано на том, что строка символов является одномерным символьным массивом, оканчивающимся нулевым символом. Рассмотрены функции ввода строк, особенности использования этих функций, а также проблемы, возникающие при вводе и выводе строк, содержащих в своем составе символы кириллицы. Предложены функции перекодировки, позволяющие справиться с этими затруднениями. Приведены сведения о стандартных функциях обработки строк.

Третья глава включает в себя вопросы обработки структур, правила объявления структур и обращения к полям структур. Обращено внимание на возможность передачи в функции массивов как полей структур, что позволит начинающим программистам избежать ошибок при передаче массивов в подпрограммы. Описан частный случай структур — объединения, с помощью которых можно экономить память в тех случаях, когда разные структуры в качестве значений одного поля должны иметь данные разных типов.

Четвертая глава пособия содержит материал, связанный с обработкой файлов. Рассмотрены особенности открытия (подготовки к работе) файлов, дано описание стандартных функций, обеспечивающих выполнение необходимых операций при работе с файлами. Приведены примеры программ обработки файлов как в режиме последовательного доступа (для текстовых и бинарных файлов), так и в режиме прямого доступа для бинарных файлов.

Пятая глава посвящена работе с указателями, поскольку они играют особую роль в языке С. Приведены правила и примеры объявления указателей на разные объекты языка (константы, переменные, функции), а также на переменные различных типов. Описаны операции, выполняемые с указателями. Обращено внимание на взаимосвязь указателей и массивов в языке С. Подробно изложены вопросы работы с динамическими данными, доступ к которым осуществляется с использованием указателей. Рассмотрены особенности выделения памяти, а также работы с динамическими переменными, одномерными и двумерными массивами, передачи этих данных в функции. Даны сведения о стандартных функциях для работы с динамической памятью и их использовании в конкретных программах. Изложены вопросы организации динамических структур данных, в частности однонаправленных и двунаправленных списков. Представлен перечень операций, выполняемых со списками, приведены примеры программ, в которых каждая операция реализована в виде отдельной функции. Описана особая разновидность динамических структур — бинарных деревьев, играющих важную роль в программировании.

Примеры, приведенные в пособии, помогут студенту за ограниченное время ознакомиться с новыми для него типами данных и освоить основные правила обработки этих данных.

Для закрепления изучаемого материала и приобретения необходимого опыта разработки программ обработки данных нечисловых и структурированных типов авторами предложены по каждой теме задания, выполнение которых послужит приобретению навыков и умений при решении рассматриваемых задач.

Задания имеют индивидуальный характер, их можно использовать при проведении лабораторных работ, подготовке к рубежным контролям знаний, а также при проведении контрольных мероприятий.

1. Символьный тип данных

1.1. Общие сведения и библиотечные функции

Символьный тип относится к числу базовых (стандартных) типов. Символьная константа представляет собой любой символ, заключенный в апострофы. В памяти ЭВМ символы представляются своим двоичным кодом (порядковым номером). Символьные переменные объявляются с помощью ключевого слова char. Обычно для хранения символа используется 1 байт. Поскольку в памяти ЭВМ символ представляется в виде целого, то, как и другие целые типы, символ может быть со знаком и без знака.

Для объявления символьного типа без знака используется ключевое слово unsigned char. Если применяется символьный тип со знаком char, то хранимые значения находятся в диапазоне от –128 до +127, при использовании беззнаковых символьных данных их значения лежат в диапазоне от 0 до 255. Оба типа определены в стандарте ANSI C, в котором применяются для кодирования символов таблицы ASCII. Первые 128 кодов (т. е. первая страница кодовой таблицы) во всех таблицах ASCII одинаковы и представляют управляющие символы с кодами от 0 до 31 и с кодом 127, а с остальными кодами — печатные символы. Вторая страница таблицы содержит символы алфавита страны локализации (для России — прописные и строчные буквы русского алфавита), символы псевдографики и некоторые другие. Для алфавита одной страны могут использоваться разные коды.

Переменным типов char и unsigned char при определении можно присваивать как значения числовых констант в десятичной форме или в шестнадцатеричной форме с предшествующими знаками 0x, например, 0x46 равно 70, так и значения символьных констант с изображением символа между апострофами.

Пример объявления символьных переменных:

```
char c=0x46,ch='F',d=-1, dh='я'; //-128..+127
unsigned char cu=70, chu='F',du=255,dhu='я';//0..+255
```

где переменные **ch**, **ch**, **cu**, и **chu** будут иметь начальное значение символа **'F'**, имеющего код 0x46 в шестнадцатеричной форме и код 70 — в десятичной; переменные **d** и **dh** типа **char** имеют один и тот же код (–1), соответствующий русской букве **'я'** и находящийся на второй странице таблицы ASCII; переменные **du** и **dhu** типа **unsigned char** имеют один и тот же код (255), соответствующий русской букве **'я'**, также находящейся на второй странице таблицы ASCII.

Любой символ можно задать, указывая символ обратной косой черты и его шестнадцатеричный код, например, **'\x46'**.

Часть символов-констант задается двумя символами, первым из которых является символ **'\'**, за которым следует печатный символ. Такие комбинации называют эскейп-последовательностями. Они представляют управляющие символьные константы, не имеющие графического изображения и используемые обычно при вводе/выводе в управляющей строке, где записываются без апострофов, например:

```
printf("%d\t%c\n",25,'F');
```

т. е. вывод 25, знака табуляции **\t**, символа **'F'** и переход на новую строку. Вне строк их следует заключать в апострофы, например:

'\n' или **'\xa'** — переход на новую строку;

'\t' или **'\x9'** — горизонтальная табуляция;

'\a' или **'\x7'** — звуковой сигнал;

'\b' или **'\x8'** — возврат на шаг **'\x8'**.

В языке С имеются и другие управляющие последовательности:

\f — перевод страницы;

\r — возврат каретки;

\t — горизонтальная табуляция;

\v — вертикальная табуляция;

**** — обратная косая черта;

\' — апостроф;

\? — вопросительный знак и др. [4].

В языке С с символьными данными можно выполнять те же операции, что и с целыми: сложение, вычитание, умножение, деление, определение остатка от деления, отношение, а также логические операции (отрицание, логическое умножение, логическое сложение, поразрядная конъюнкция, поразрядная дизъюнкция, поразрядное исключающее ИЛИ, поразрядное отрицание).

Следующая программа демонстрирует выполнение арифметических и логических операций над данными символьного типа, при этом получаемый результат выводится в виде целого и в виде символа:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{signed char a,b,c;
  a=55;
  b=4;
//демонстрация выполнения арифметических операций

  c=a+b;
  printf("a=%d,a=%c,b=%d,b=%c,c=%d,c=%c\n",a,a,b,b,c,c);
  c=a-b;
  printf("\n a=%d,a=%c,b=%d,b=%c,c=%d,c=%c\n",a,a,b,b,c,c);
  c=a*b;
  printf("\n a=%d,a=%c,b=%d,b=%c,c=%d,c=%c\n",a,a,b,b,c,c);
  c=a/b;
  printf("\n a=%d,a=%c,b=%d,b=%c,c=%d,c=%c\n",a,a,b,b,c,c);
  c=a%b;
  printf("\n a=%d,a=%c,b=%d,b=%c,c=%d,c=%c\n",a,a,b,b,c,c);
  c=-a;
  printf("\n c=%d,c=%c  ",c,c);
//демонстрация выполнения поразрядных
//логических операций

  c=60;  printf("\n c=%d,c=%c\n",c,c);
  c=~a;printf("\n c=%d,c=%c  ",c,c);
  c=a^b; printf("\n a^b c=%d,c=%c  ",c,c);
  c=a|b;printf("\n c=%d,c=%c  ",c,c);
  c=a&b;printf("\n c=%d,c=%c  ",c,c);
  //демонстрация выполнения  логических операций
```

```
  c=!a;printf("\n c=%d,c=%c  ",c,c);
  c=a||b;printf("\n c=%d,c=%c  ",c,c);
  c=a&&b;printf("\n c=%d,c=%c  ",c,c);
  getch();
  return 0;
}
```

Окно вывода результатов работы программы показано на рис. 1.1.

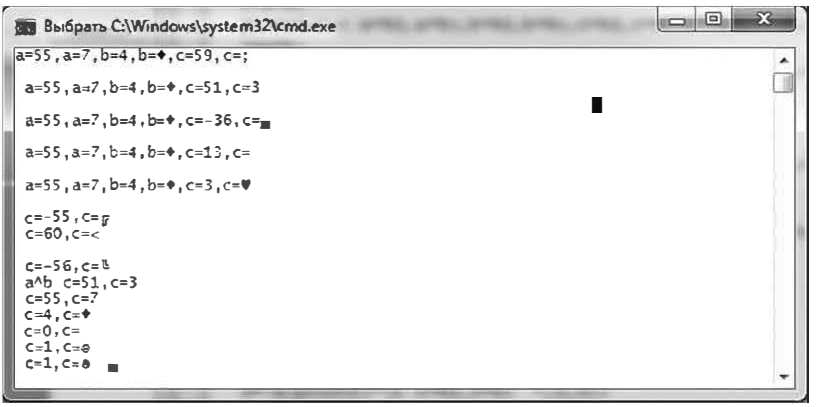

Рис. 1.1

Ввод символьных данных можно выполнять с помощью функции **scanf** по спецификации **%c**, вывод — с помощью функции **printf**. Кроме того, в стандартной библиотеке **<stdio.h>** имеется несколько функций, предназначенных специально для ввода/вывода символов.

Функция **int getchar(void)** вводит символ со стандартного устройства ввода и возвращает его в форме целого. Однако возвращаемое значение можно присвоить переменной типа **char**, что, как правило, и делают на практике. После набора символа надо нажать клавишу <Enter>.

Функция **int putchar(int ch)** печатает символ, хранящийся в **ch**. Функция возвращает код выводимого символа.

Для более удобного ввода символов используют функции **int getch(void)** и **int getche(void)**. Прототипы этих функций находятся в библиотеке **<conio.h>**. Функция **getch** ожидает нажатия любой клавиши на клавиатуре, после чего немедленно возвращает введенное значение. Символ, вводимый с клавиатуры, на

экране не отображается. При использовании функции `getche` вводимый символ отображается на экране, в этом и состоит различие двух этих функций.

Следует обратить внимание на ввод/вывод символов кириллицы (русских букв). При работе в среде Windows используется стандарт ANSI C для кодировки символов, а режим консольных приложений имитируется в среде MS DOS, где применяется стандарт ASCII. Для первой страницы кодовой таблицы, включающей латиницу (буквы латинского алфавита), арабские цифры и все наиболее употребимые символы, в том числе управляющие, кодировки двух стандартов совпадают, а для второй страницы, включающей символы кириллицы, кодировки двух стандартов различаются. Поэтому при вводе/выводе символов кириллицы требуется выполнять перекодировку символов.

Данные функции могут иметь следующий вид:

```
char Rus(char ch)
//Функция перекодирования кириллицы при выводе
{
   char ch1;

      if (ch >= -64 && ch <= -17)
         ch1 = (char)(-64+ch);//A..п
      else if (ch <= 0 && ch >= -16)
         ch1 =  (char)(-16+ch);//р..я
      else if (ch == -72) //e
         ch1 =  (char)(-15);
      else if (ch == -88) //E
         ch1 = (char)(-16);
      else
         ch1= ch;//остальные символы

   return ch1;
}
char RusIn(char ch)
//Функция перекодирования кириллицы при вводе
{
   char ch1;

      if (ch >= -128 && ch <= -81)
         ch1 = (char)(64+ch);//A..п
```

```
        else if (ch <= -16 && ch >= -32)
           ch1 =  (char)(16+ch);//p..я
        else if (ch == -15) //e
           ch1 =  (char)(-72);
        else if (ch == -16) //E
           ch1 = (char)(-88);
        else
           ch1= ch;//остальные символы

    return ch1;
}
```

Функция Rus позволяет построить следующую функцию, которая будет применяться далее для вывода поясняющих русских текстов, задаваемых в виде строковых констант, или вывода указанного количества символов из массива:

```
void printRus(const char *s,int n=120)
{
    //Вывод либо n символов (по умолчанию 120),
    //либо до символа с кодом 0
    int i;
    for(i=0; s[i]!=0 && i<n; i++)
       printf("%c",Rus(s[i]));
}
```

При работе с символьными данными следует иметь в виду, что заглавные символы латиницы образуют один непрерывный интервал. Проверить, является ли символ заглавной буквой латинского алфавита, можно с помощью следующего условного оператора:

```
if (ch>='A' && ch<='Z')
   printRus( " Заглавная буква латинского алфавита");
```

Строчные буквы латинского алфавита также образуют один непрерывный интервал, поэтому можно использовать аналогичный оператор:

```
if (ch>='a' && ch<='z')
   printRus( " Строчная буква латинского алфавита");
```

Непрерывный интервал образуют цифровые символы. Проверить, является ли символ цифрой, можно с помощью того же условного оператора:

```
if (ch>='0' && ch<='9')
   printRus( " Цифровой символ");
```

Среди всех символов принято выделять печатаемые (отображаемые на экране терминала) и управляющие. К печатаемым относятся все символы, расположенные между пробелом (код 32) и тильдой (код 254). Управляющие символы имеют коды, которые находятся в интервале от 0 до 31, а также код 127.

Для обработки символьных данных существует целый ряд функций, объявленных в заголовочном файле `<ctype.h>`:

- `int isalnum(int ch);` возвращает ненулевое значение, если аргумент является буквой или цифрой, и нуль, если символ не является буквенно-цифровым;
- `int isalpha(int ch);` возвращает ненулевое значение, если аргумент является буквой, и нуль, если символ не является буквенным;
- `int iscntrl(int ch);` возвращает ненулевое значение, если аргумент является управляющим символом, и нуль, если символ не является управляющим;
- `int isdigit(int ch);` возвращает ненулевое значение, если аргумент является цифрой, и нуль, если символ не является цифрой;
- `int isgraph(int ch);` возвращает ненулевое значение, если аргумент является любым печатаемым символом, и нуль, если символ не является печатаемым;
- `int islower (int ch);` возвращает ненулевое значение, если аргумент является строчной буквой, и нуль, если символ не является строчной буквой;
- `int isprint(int ch);` возвращает ненулевое значение, если аргумент является печатаемым символом, включая пробел, и нуль в противном случае;
- `int ispunct(int ch);` возвращает ненулевое значение, если аргумент является знаком пунктуации, и нуль в противном случае (пробел не является символом пунктуации);
- `int isspace(int ch);` возвращает ненулевое значение, если аргумент является пробельным символом, и нуль в противном случае;
- `int isupper(int ch);` возвращает ненулевое значение, если аргумент является прописной буквой, и нуль в противном случае;

• `int isxdigit(int ch);` возвращает ненулевое значение, если аргумент является шестнадцатеричной цифрой, и нуль в противном случае. К шестнадцатеричным цифрам относятся все символы из следующих интервалов: `A–F`, `a–f`, `0–9`;

• `int tolower(int ch);` возвращает строчный эквивалент символа `ch`, если аргумент является буквой, и тот же символ `ch` в противном случае;

• `int toupper(int ch);` возвращает прописной эквивалент символа `ch`, если аргумент является буквой, и тот же символ `ch` в противном случае.

1.2. Примеры программ обработки символьных данных

Пример 1. В каждой строке символьной матрицы подсчитать количество букв, цифр и символов, не являющихся буквами и цифрами. Вывести матрицу так, чтобы рядом со строкой матрицы выводились найденные количества.

Текст программы:

```
#include "stdafx.h"
#include<stdio.h>
#include<conio.h>

char Rus(char ch)
{//Функция перекодирования кириллицы при выводе
   char ch1;

      if (ch >= -64 && ch <= -17)
         ch1 = (char)(-64+ch);//A..п
      else if (ch <= 0 && ch >= -16)
         ch1 =  (char)(-16+ch);//р..я
      else if (ch == -72) //e
         ch1 =  (char)(-15);
      else if (ch == -88) //E
         ch1 = (char)(-16);
      else
         ch1= ch;//остальные символы

   return ch1;
}
```

```
void printRus(const char *s,int n=120)
{//функция printRus перекодирования
 //кириллицы и вывода
   int i;
   for(i=0; s[i]!=0;i++)
      printf("%c",Rus(s[i]));
}
int _tmain(int argc, _TCHAR* argv[])
{
   const int mm=20,nn=20;
   char a[mm][nn];
   int kb,kz,knb,i,j,m,n;
   printRus("Введите размерности матрицы\n");
   scanf("%d%d",&m,&n);
   printRus("Введите матрицу\n");
   for (i=0;i<m;i++)
   {
      for (j=0;j<n;j++)
         a[i][j]=getche();
      printf("\n");
   }
   printRus("Обработка строк матрицы\n");
   printRus(
"Строка Колич. буквКолич. цифрКолич. прочих\n");
   for (i=0;i<m;i++)
   {
//присваивание начальных значений количествам
      kb=0;kz=0;knb=0;
      for (j=0;j<n;j++)
      {//подсчет количеств и вывод
         if (a[i][j]>='A'&& a[i][j]<='Z'
          || a[i][j]>='a' && a[i][j]<='z')
            kb++;
         else if (a[i][j]>='0'
               && a[i][j]<='9')
            kz++;
         else
            knb++;
   }
      printf("  %d%9d%11d%11d\n",i,kb,kz,knb);
   }
   getch();
   return 0;
}
```

```
/*
   Протокол ввода - вывода

Введите размерности матрицы
2 5
Введите матрицу
A1 5D
ASD3=
Обработка строк матрицы
Строка Колич. буквКолич. цифрКолич. прочих
  02 2 1
  13 1 1
*/
```

Пример 2. В заданное число строк двумерного символьного массива ввести последовательности символов различной длины. Предполагается, что символы кириллицы не используются, чтобы не применять перекодировку символов. Переформировать последовательности так, чтобы в начале каждой строки располагались буквенные символы, а за ними все остальные символы. Вывести из массива полученные последовательности символов в поряде возрастания кодов их первых букв без перестановки данных между строками массива.

При вводе матрицы следует учитывать, что при нажатии клавиши <Enter> генерируется код 10 (перевод строки), который не должен рассматриваться как элемент матрицы. Поэтому цикл ввода элементов в строки массива (внутренний цикл) оформляется как цикл с заранее неизвестным числом повторений и с выходом из цикла при вводе с кодом 10 или всей строки.

Строки матрицы выводятся в поряде возрастания кодов их первых букв без перестановки данных между строками массива. Для этого создается массив указателей на строки матрицы. Если на шаге алгоритма упорядочения одна из двух строк больше другой, то выполняется только обмен значениями соответствующих указателей на эти строки матрицы.

Текст программы:

```
/*
!!!Проект подготовлен для создания:
  либо рабочего варианта программы, когда не активна директива
#define debug, т.е. сделана комментарием так:
```

```
//#define debug,
  либо отладочного варианта, когда директива #define debug
активна.
  В рабочем варианте разрешен ввод исходных данных, а в отла-
дочном вместо ввода используются начальные значения перемен-
ных, заданные при их определении.
*/
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
//#define debug
char Rus(char ch) //Функция перекодирования кириллицы при выводе
{
   char ch1;

      if (ch >= -64 && ch <= -17)
         ch1 = (char)(-64+ch);//A..п
      else if (ch <= 0 && ch >= -16)
         ch1 =  (char)(-16+ch);//р..я
      else if (ch == -72) //e
         ch1 =  (char)(-15);
      else if (ch == -88) //E
         ch1 = (char)(-16);
      else
         ch1= ch;//остальные символы

   return ch1;
}
void printRus(const char *s,int n=120)
{
   //Вывод либо n символов (по умолчанию 120),
   //либо до символа с кодом 0
   int i;
   for(i=0; s[i]!=0 && i<n; i++)
      printf("%c",Rus(s[i]));
}

int _tmain(int argc, _TCHAR* argv[])
{
   const int mm=20, nn=20;
   char  a[mm][nn]={{'0','s','f'},{'1','a','r','t'}
            ,{'2','f','g','h','j'},{'b','n'}}
   ,c
   , *z[mm], *pz;
```

```
   int m=4,n=5,i,j,k,flag,na[mm]={3,4,5,2};
#ifndef debug
   printRus("Введите размерности массива\n");
   scanf("%d%d",&m,&n);
   printRus("Введите символы в строки массива\n");
   scanf("%c",&c);
   for (i=0;i<m;i++)
   {
      j=0;
      do
      {
         scanf("%c",&a[i][j]);
         if (a[i][j]!=10)
            j++;
         else
            break;
      }while (j<n);
      na[i]=j;
   }
#endif
  printRus("\nВывод последовательностей символов,");
  printRus(" введенных в строки массива\n");
   for (i=0;i<m;i++)
   {
      for (j=0;j<na[i];j++)
         putchar(a[i][j]);
      printf("\n");
   }
//перестановка символов в строках массива
   for (i=0;i<m;i++)
   {
      k=-1;
      for (j=0;j<na[i];j++)
      if (((a[i][j]>='A') && (a[i][j]<='Z'))
       || ((a[i][j]>='a') && (a[i][j]<='z')))
      {
         k++;
         c=a[i][j];
         a[i][j]=a[i][k];
         a[i][k]=c;
      }
}
printRus("\nВывод последовательностей символов");
```

```
printRus(" из строк массива после перестановки");
printRus(" символов\n");
for (i=0;i<m;i++)
{
   for (j=0;j<na[i];j++)
      putchar(a[i][j]);
   printf("\n");
}
for(i=0;i<m;i++)
   z[i]=a[i];
for(i=0;i<m-2;i++)
{
   int flag=1;
   for(j=0;j<m-i-1;j++)
      if(z[j][0]>z[j+1][0])
      {
         pz=z[j];
         z[j]=z[j+1];
         z[j+1]=pz;
         k=na[j];
         na[j]=na[j+1];
         na[j+1]=k;
         flag=0;
      }
      if(flag)
      break;
}

printRus("\nВывод отсортированных");
printRus("последовательностей символов\n");
for (i=0;i<m;i++)
{
      printRus(z[i],na[i]);
      printf("\n");
}

getch();
return 0;
}
/*
   Протокол ввода - вывода программы в рабочем режиме

Введите размерности массива
3 7
```

```
Введите символы в строки массива
1sd
12as
3ghjk

Вывод последовательностей символов, введенных в строки массива
1sd
12as
3ghjk

Вывод последовательностей символов из строк массива после
перестановки символов
sd1
as12
ghjk3

Вывод отсортированных последовательностей символов
as12
ghjk3
sd1*/
```

Пример 3. Определить в символьной матрице симметричные строки и вывести матрицу в виде матрицы, размещая рядом с каждой строкой признак симметричности. Для ввода, вывода, определения симметричности строки создать и использовать функции. При вводе символов в строку массива желательно набрать ровно столько символов, сколько столбцов во вводимой матрице, и нажать клавишу <Enter>.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
char Rus(char ch)
{//Функция перекодирования кириллицы при выводе
   char ch1;

      if (ch >= -64 && ch <= -17)
         ch1 = (char)(-64+ch);//A..п
      else if (ch <= 0 && ch >= -16)
         ch1 =  (char)(-16+ch);//p..я
      else if (ch == -72) //e
         ch1 =  (char)(-15);
```

```
        else if (ch == -88) //E
            ch1 = (char)(-16);
        else
            ch1= ch;//остальные символы

    return ch1;
}

void printRus(const char *s,int n=120)
{
    //Вывод либо n символов (по умолчанию 120),
    //либо до символа с кодом 0
    int i;
    for(i=0; s[i]!=0 && i<n; i++)
        printf("%c",Rus(s[i]));
}

void wwod(int nn,int *m,int *n,char *a)
{//функция ввода исходных данных
    int i,j;
    char ch;
    printRus("\nВведите число строк и число");
    printRus("столбцов матрицы\n");
    scanf("%d%d",m,n);
    printRus("\nВведите матрицу\n");
    for (i=0;i<*m;i++)
    {
        j=0;
        do
        {
            scanf("%c",&a[i*(nn)+j]);
            if (a[i*(nn)+j]!=10)
                j+=1;
        }while(j<*n);
    }
}

void wiwod(int nn,int m,int n,char *a)
{//функция вывода матрицы
    int i,j;
    printRus("\n число строк матрицы, число");
    printRus(" столбцов матрицы\n");
    printf(" %d%d\n",m,n);
```

```
   printRus("   матрица:\n");
   for (i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
         putchar(a[i*(nn)+j]);
      printf("\n");
   }
}
bool simmetr(char *a,int n)
{//функция определения симметричности строки
 //(массива символов)
   int i;
   i=0;
   while (a[i]==a[n-i-1] && i<=n/2)
      i++;
   if (i<=n/2)
      return false;
   else
      return true;
}

////////////////////////////////////
int _tmain(int argc, _TCHAR* argv[])
{//основная функция
   const int mm=10, nn=15;
   char a[mm][nn]={
{'a','a','a','a','a'},{'a','b','b','a','a'}};
   int m=2,n=5,i,j;
   printRus("Ввод матрицы\n");
   wwod(nn,&m,&n,&a[0][0]);
   printRus("Контрольный вывод");
   wiwod(nn,m,n,&a[0][0]); //?a[0]
   printRus("\nВывод строк матрицы ");
   printRus("с пояснениями\n");
   for (i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
         putchar(a[i][j]);
      if (simmetr(a[i],n))
         printRus("  симметрично\n");
  else
         printRus("  не симметрично\n");
  }
```

```
 getch();
 return 0;
}
/*
   Протокол ввода - вывода

Ввод матрицы

Введите число строк и число столбцов матрицы
2 5

Введите матрицу
12321
QWERT
Контрольный вывод
число строк матрицы, число столбцов матрицы
 25
  матрица:
12321
QWERT

Вывод строк матрицы с пояснениями
12321  симметрично
QWERT  не симметрично
*/
```

1.3. Задания на обработку символьных матриц

Во всех приведенных ниже заданиях, если не оговорено специально, предполагается использование букв русского алфавита, а также вывод на русском языке всех пояснений и правильный ввод/вывод данных с буквами кириллицы.

1. В каждой строке символьной матрицы $S(N, M)$, $N \le 10$, $M \le 20$ определить количество гласных букв. Вывести исходную матрицу и рядом с ней (в виде столбца) найденное количество гласных букв. Переформировать исходную матрицу так, чтобы строки были упорядочены по убыванию количества гласных. Вывести переформированную матрицу. Новых матриц не вводить.

2. В символьной матрице $SIM(K, L)$, $K \le 15$, $L \le 20$ определить все симметричные строки. Вывести исходную матрицу и рядом с каждой строкой вывести признак симметрии (0 — обычная, 1 — симметричная). Переформировать исходную матрицу так, чтобы

сначала располагались симметричные строки, а затем — несимметричные. Вывести переформированную матрицу. Новых матриц не вводить.

3. В символьной матрице $A(M, N)$, $M \le 12$, $N \le 15$ элементы каждой строки упорядочить по возрастанию. Вывести исходную и преобразованную матрицы. В преобразованной матрице строки упорядочить по первому символу строки по возрастанию и вывести ее. Новых матриц не вводить.

4. В символьной матрице $R(K, L)$, $K \le 10$, $L \le 18$ определить все символы, которые встретились в ней ровно один раз. Найденные символы сохранить в одномерном массиве в порядке возрастания их кодов. Вывести исходную матрицу и полученный массив.

5. В символьной матрице $C(M, N)$, $M \le 12$, $N \le 16$ определить все согласные буквы, которые в ней не встретились ни разу. Найденные буквы занести в одномерный массив в алфавитном порядке. Вывести исходную матрицу и полученный массив.

6. В символьной матрице $X(K, M)$, $K \le 14$, $M \le 19$ подсчитать, сколько раз в ней встретился каждый цифровой символ ('0', '1', …, '9'). Вывести исходную матрицу, под матрицей — цифровые символы, под каждым символом — то количество раз, которое встретился цифровой символ.

7. В символьной матрице $Z(L, M)$, $L \le 18$, $M \le 20$ определить гласные буквы, которые ни разу в ней не встретились. Найденные буквы разместить в одномерном массиве в алфавитном порядке. Вывести исходную матрицу и полученный массив.

8. В символьной матрице $B(K, M)$, $K \le 16$, $M \le 18$ в каждой строке разместить сначала небуквенные символы, а затем буквенные символы. Вывести исходную и преобразованную матрицы. Переформировать преобразованную матрицу так, чтобы сначала размещались в ней строки, начинающиеся с цифровых символов, и вывести ее. Новых матриц не вводить.

9. В символьной матрице $Q(L, M)$, $L \le 15$, $M \le 20$ в каждой строке сначала расположить гласные буквы, затем согласные. Предполагается, что в матрице содержатся только буквенные символы. Вывести исходную и преобразованную матрицы. Строки преобразованной матрицы упорядочить в алфавитном порядке, учитывая только первую букву строки, и вывести ее. Новых матриц не вводить.

10. Из символьной матрицы $ZS(K, M)$, $K \le 18$, $M \le 25$ составить одномерный массив, переписав в него все элементы матрицы. Упорядочить полученный массив в алфавитном порядке (по возрастанию кодов символов) и переписать их в исходную матрицу построчно. В преобразованной матрице подсчитать в каждом столбце количество гласных букв. Вывести исходную и преобразованную матрицы в виде матриц по строкам, добавляя перед каждым символом шесть пробелов, под каждым столбцом матрицы вывести найденное количество гласных.

11. В символьной матрице $MT(K, L)$, $K \le 10$, $L \le 18$ в каждой строке определить наибольшую длину небуквенной серии (в строке может быть несколько небуквенных серий). (Небуквенная серия — последовательность небуквенных символов, ограниченная буквенными символами или началом (концом) строки.) Вывести исходную матрицу и рядом с каждой строкой — полученные значения длин серий. Преобразовать исходную матрицу так, чтобы ее строки располагались в порядке возрастания длин небуквенных серий и вывести ее. Новых матриц не вводить.

12. В символьной матрице $W(K, M)$, $K \le 16$, $M \le 20$ подсчитать по каждому столбцу частоту появления согласных букв (отношение количества согласных букв к общему количеству символов). Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом — полученное значение частоты. Упорядочить затем столбцы преобразованной матрицы по возрастанию частоты согласных и вывести ее. Новых матриц не вводить.

13. Переписать элементы символьной матрицы $V(L, M)$, $L \le 15$, $M \le 22$ в одномерный массив. Переформировать элементы в одномерном массиве так, чтобы сначала располагались небуквенные символы, затем — буквенные. Переписать элементы одномерного массива в матрицу по столбцам. Подсчитать по каждому столбцу матрицы количество небуквенных символов. Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом преобразованной матрицы — найденное количество небуквенных символов. Новых матриц не вводить.

14. В символьной матрице $A(K, M)$, $K \le 16$, $M \le 18$ найти все строки, совпадающие со строками символьной матрицы $B(K, M)$,

$K \le 16$, $M \le 18$. Переформировать строки матрицы A так, чтобы найденные строки располагались бы в ее начале, а остальные строки — в конце. Вывести исходные матрицы A и B и преобразованную матрицу A.

15. Ввести две символьные матрицы $SL(L, M)$ и $Z(K, M)$, $K \le 10$, $L \le 15$, $M \le 18$. Проверить, можно ли из символов каждой строки матрицы SL составить слова, записанные в строках матрицы Z. Символы исходной матрицы SL можно переставлять, но каждый символ использовать не более одного раза. Вывести исходные матрицы SL и Z, причем рядом с каждой строкой матрицы SL указать номера строк матрицы Z, которые можно составить из символов строки исходной матрицы SL. (Например, первая матрица содержит строку РЕЙТИНГ, а вторая матрица — строки РИНГ, ГИМН, НЕТ. Из первой матрицы можно составить первую и третью строки второй матрицы, а вторую строку составить нельзя.)

16. В символьной матрице $Q(K, M)$, $K \le 16$, $M \le 22$ найти повторяющиеся строки и оставить их в матрице в одном экземпляре. Строки полученной матрицы упорядочить по алфавиту (в порядке возрастания кодов) первого символа строки. Вывести исходную и преобразованную матрицы. Новых матриц не вводить.

17. Из символьной матрицы $B(K, M)$, $K \le 16$, $M \le 18$ сформировать две новые матрицы BUK и $NBUK$. Исходная матрица пробелов не содержит. B матрицу BUK переписать из исходной матрицы B все буквенные символы, а в матрицу $NBUK$ — все небуквенные символы. Элементы в новые матрицы заносить последовательно в строки без пропусков. Незаполненные элементы новых матриц заполнить пробелами. Буквенные элементы матрицы B переписать в одномерный массив A, упорядочить его по алфавиту и составить из этих элементов матрицу BU с тем же количеством столбцов, как и у матрицы B, переписав в нее элементы массива A по строкам. Если последняя строка матрицы BU окажется неполной, то дополнить ее пробелами. (Матрица BU может содержать меньше строк, чем матрица B.) Вывести исходную матрицу B и только те строки новых матриц, которые содержат переписанные символы.

18. В символьной матрице $FS(L, M)$, $L \le 15$, $M \le 22$ найти все гласные буквы, которые встретились в ней ровно по одному разу. Найденные буквы занести в одномерный массив, который упоря-

дочить затем в алфавитном порядке. Вывести исходную матрицу и полученный массив.

19. В каждом столбце символьной матрицы *SMA*(*K*, *M*), $K \le 16$, $M \le 18$ подсчитать количество знаков препинания ('.', ',', ':', ';', '-', '"', '?', '!'). Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом — найденное количество знаков препинания. Расположить столбцы исходной матрицы по возрастанию найденного количества и вывести преобразованную матрицу. Новых матриц не вводить.

20. В символьной матрице *GL*(*M*, *N*), $M \le 12$, $N \le 16$, которая не содержит пробелов, заменить все гласные буквы пробелами. Вывести исходную и преобразованную матрицы. Упорядочить строки преобразованной матрицы по убыванию содержащихся в каждой строке пробелов и вывести ее. Новых матриц не вводить.

21. В каждом столбце символьной матрицы *AL*(*K*, *L*), $K \le 10$, $L \le 18$ подсчитать частоту появления знаков арифметических операций ('+', '–', '*', '/'). Частота — отношение количества найденных в строке знаков операций к общему количеству символов строки. Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом — значение частоты. Упорядочить столбцы преобразованной матрицы по возрастанию найденных частот и вывести ее. Новых матриц не вводить.

22. В каждом столбце символьной матрицы *MC*(*K*, *L*), $K \le 20$, $L \le 15$ определить наибольшую длину буквенной серии (в столбце может быть несколько буквенных серий). Буквенная серия — последовательность букв, обрамленная небуквенными символами или началом (концом) столбца. (Например, последовательность abcd2+fghjk- содержит две буквенные серии: abcd и fghjk.) Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом — найденные значения длин серий. Упорядочить столбцы преобразованной матрицы по убыванию длин буквенных серий и вывести ее. Новых матриц не вводить.

23. В каждой строке символьной матрицы *SIM*(*M*, *N*), $M \le 12$, $N \le 15$ подсчитать отношение количества небуквенных символов к общему количеству символов строки. Вывести исходную матрицу,

а рядом с каждой ее строкой — найденные отношения. Упорядочить строки преобразованной матрицы по возрастанию найденных отношений и вывести ее. Новых матриц не вводить.

24. В символьной матрице $SM(K, N)$, $K \le 16$, $N \le 18$ заменить все небуквенные символы символом '*' (звездочка). Вывести исходную и преобразованную матрицы. Упорядочить столбцы преобразованной матрицы по возрастанию количества содержащихся в них звездочек и вывести ее. Новых матриц не вводить.

25. В символьной матрице $W(M, N)$, $M \le 19$, $N \le 17$ определить все симметричные столбцы. Вывести исходную матрицу и под каждым ее столбцом — признак симметрии (0 — обычный столбец, 1 — симметричный). Переформировать столбцы исходной матрицы так, чтобы сначала располагались симметричные столбцы, затем — несимметричные. Вывести преобразованную матрицу. Новых матриц не вводить.

26. Элементы каждого столбца символьной матрицы $Q(L, M)$, $L \le 15$, $M \le 22$ упорядочить в алфавитном порядке (по возрастанию кодов символов). Вывести исходную и преобразованную матрицы. В преобразованной матрице столбцы упорядочить по алфавиту по первому символу столбца и вывести ее. Новых матриц не вводить.

27. Каждый столбец символьной матрицы $ZK(K, N)$, $K \le 14$, $N \le 12$ переформировать таким образом, чтобы сначала в нем располагались буквенные символы, а затем небуквенные. Вывести исходную и преобразованную матрицы. Столбцы полученной матрицы упорядочить в алфавитном порядке по первому символу столбца и вывести ее. Новых матриц не вводить.

28. В каждом столбце символьной матрицы расположить сначала согласные буквы, а затем — гласные. Матрица содержит только буквенные символы. Вывести исходную и преобразованную матрицы. Строки преобразованной матрицы упорядочить в алфавитном порядке по первому символу строки и вывести ее. Новых матриц не вводить.

29. В каждом столбце символьной матрицы $MTS(K, L)$, $K \le 20$, $L \le 15$ определить наибольшую длину последовательности цифровых символов (таких последовательностей в каждом столбце может быть несколько). Вывести исходную матрицу в виде матрицы по строкам, добавляя перед каждым символом шесть пробелов, а под каждым столбцом — полученные длины последовательностей.

Упорядочить столбцы матрицы по убыванию найденных длин цифровых последовательностей и вывести полученную матрицу. Новых матриц не вводить.

30. Строки символьной матрицы $G(M, N)$, $M \le 14$, $N \le 16$, в которых чередуются гласные и согласные буквы, расположить в начале матрицы, а строки, в которых не обнаружено чередование, разместить в конце матрицы. Матрица содержит только буквенные символы. Вывести исходную и преобразованную матрицы. Элементы строк преобразованной матрицы, в которых нет чередования гласных и согласных, упорядочить в порядке, обратном алфавитному, и вывести ее. Новых матриц не вводить.

Вопросы для самопроверки

1. Как определить символьную переменную с начальным значением?
2. Как записать символьную константу?
3. Можно ли символьной переменной присвоить целочисленное значение?
4. Можно ли переменной целого типа присвоить символьное значение?
5. Какой формат применяется при выводе символьного значения?
6. Какие операции можно использовать для символьных данных?

2. Строки

2.1. Общие сведения и библиотечные функции

Строка в языке С представляет собой последовательность символов в одномерном массиве символов, заканчивающуюся нулевым символом (нуль-символом). Нулевым символом является символ с нулевым кодом, '\0'. Определяя массив символов, предназначенный для хранения строки, необходимо предусмотреть место для нулевого (последнего) символа, т. е. количество элементов должно быть на единицу больше, чем наибольшее предполагаемое количество символов в строке. Например, объявление строки для хранения 15 символов должно выглядеть следующим образом: `char s[16];` последний, 16-й, элемент предназначен для хранения нулевого символа.

Количество символов строки плюс нуль-символ может быть меньше длины массива. Строка может иметь длину 0, т. е. состоять только из нуль-символа, и меняться в процессе работы программы. Более того, в памяти, выделенной для массива, может содержаться несколько строк, а указателями на их начала могут служить адреса ячеек памяти массива. Однако в дальнейшем по умолчанию будем считать, что строка находится в начале массива и адресуется его именем.

Строковая константа представляет собой произвольную последовательность символов, заключенную в кавычки, например:

```
"Это строка символов", "MGTU".
```

Символьные массивы можно при определении инициализировать строковыми константами, при этом последний нуль-символ, который следует за последним заданным символом, формируется автоматически, например:

```
char a[21]="Student number 234".
```

Если массив при определении инициализируется, то его размерность можно опускать, при этом компилятор сам выделит память, достаточную для размещения всех символов строки и завершающего нуль-символа, например:

```
char ss[]="Ivanov Petr".
```

Ввод строк может осуществляться с помощью функции scanf со спецификатором преобразования **%s**, например:

```
char ss[101]; scanf("%s",ss);
```

Однако в этом случае выполняется ввод символов строки до тех пор, пока не встретится какой-либо разделитель (ввод осуществляется до первого разделителя). К числу разделителей относятся символы пробела, горизонтальной табуляции (клавиша <TAB>), вертикальной табуляции (клавиши <Ctrl> + <K>), прогона бумаги (страницы) (клавиши <Ctrl> + <L>), перевод строки (клавиша <Enter>). Таким образом, использовать функцию **scanf** для ввода в символьный массив, например, предложения "Группа студентов" нельзя, так как будет введено только первое слово "Группа" (до первого пробела). Однако в буфере ввода останется второе слово, и оно будет введено очередным вызовом функции **scanf** в указанный в ней в качестве параметра символьный массив. После ввода к строке автоматически добавляется символ конца строки (нуль-символ).

В общем случае процесс ввода слов можно представить так. Пользователь набирает последовательность символов текста (слов), между которыми могут быть разделители. До нажатия клавиши <Enter> вводимую последовательность символов можно редактировать (добавлять символы, удалять символы клавишами <Backspace> и <Delete>, передвигать курсор окна программы влево и вправо клавишами <→>, <←>). После нажатия клавиши <Enter> набранная в окне программы последовательность символов целиком перемещается в буфер (т. е. поток) ввода, откуда вызовы функции **scanf** перемещают слова с добавляемыми к ним нуль-символами в массивы.

Функцию **scanf** удобно использовать в тех случаях, когда уже при вводе требуется выделить отдельные слова вводимого текста и разместить их в отдельных массивах. Ниже представлен пример

фрагмента программы выделения нажатием клавиши <Enter> из входного потока символов, заканчивающихся символом с кодом 10 (перевод строки), и размещения в элементах двумерного массива символов:

```
char  cc,
   q[33][16];   //массив, в элементах которого
 //будут размещены слова из вводимого текста
int   nq;//количество элементов массива q,
//которые будут заполнены словами
for(i=0;; i++)
{
   scanf("%s%c",q[i],&cc);
   printf("%s%d\n",q[i],cc);//для контроля ввода
   if(cc==10)
   {
      nq=i;
      break;
   }
}
```

Если требуется ввести в массив весь текст, то следует использовать функцию

```
char *gets(char *str);
```

где **str** — это указатель на массив символов, в который записываются вводимые пользователем символы. В качестве результата функция также возвращает **str**. Например:

```
char a[201]; gets(a);
```

С помощью функции **gets** символы подготовленного текста вводятся в массив до тех пор, пока не будет введен символ перевода строки. Этот символ не записывается в строку, а вместо него в качестве последнего записывается нуль-символ. Таким образом, для ввода строки надо нажать клавишу <Enter>, но до ее нажатия можно исправить неправильно набранные символы, перемещая курсор клавишей <Backspace>.

Необходимо иметь в виду, что функция **gets** не проверяет границы массива, в который вводятся символы, и если пользователь введет больше символов, чем помещается в массиве, то введутся не все символы, а самое главное — не будет сформирован признак конца строки (не запишется нуль-символ).

Для вывода строк можно использовать функцию **printf** со спецификатором **%s**, например:

```
char str[101]; printf("Строка - %s", str);
```

или функцию **puts**.

Функция **puts** отображает на экране символы строки своего аргумента, т. е. массива, после чего переводит курсор на новую строку в окне программы:

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

где **str** — указатель на массив с выводимой строкой; в качестве результата функция возвращает целое значение — количество выведенных символов, а в случае ошибки — значение константы **EOF**. Функция puts позволяет использовать те же управляющие последовательности символов, что и функция **printf** (переход на новую строку, табуляцию и т. д.).

Функция **puts** является более предпочтительной, чем функция **printf**, при выводе строк и символов, если при выводе не требуется выполнять преобразование данных. Функция **puts** использует намного меньше ресурсов, так как выводит только строку символов, но не выводит числа и не выполняет преобразования данных. Эта функция занимает меньше места и работает быстрее.

Введенные функцией **gets** строки с русскими буквами выводятся функцией puts правильно, а строковые констаны — неправильно. Для латинских букв такого несоответствия нет.

В языке С операции, выполняемые над строками, отсутствуют. Единственным исключением является инициализация массива при его определении значением строковой константы. Например, нельзя одной строке присвоить значение (скопировать данные) другой строки. Для этого надо написать цикл поэлементного копирования элементов одной строки в другую.

Пример копирования элементов одной строки в другую:

```
char s1[101], s2[101];
 printf("wwedite stroku\n");
 gets(s1);
 int i=-1;
   do{ i++;
 s2[i]=s1[i];}
```

```
while (s1[i]!='\0');
   printf("\n stroka s2= ");
puts(s2);
getch();
return 0;
```

Каждый раз писать самостоятельно фрагменты программ, выполняющих стандартные действия над строками, нецелесообразно. Удобнее использовать библиотечные функции стандартной библиотеки языка Си стандарта ANSI C, объявленные в заголовочном файле string.h. Функции обработки строк используют только с наборами символов кодировки стандарта ASCII на всех платформах, поддерживающих Си. Для представления символов ASCII (таблицы ASCII) используют однобайтовое кодирование. Первые 128 кодов (первая страница таблицы ASCII) во всех таблицах одинаковы и представляют управляющие символы с кодами от 0 до 31, а с бóльшими кодами — печатные символы. При вводе управляющие символы генерируются соответствующими клавишами или их комбинациями (см. подразд. 1.1), а при выводе задаются в управляющей строке управляющими символьными константами, например, `\n` — перевод строки (код 10), `\t` — табуляция (код 9), `\r` — возврат каретки (код 13), `\f` — прогон страницы (код 12). При выводе функцией `printf` можно также задавать коды управляющих символов в списке вывода и использовать для них спецификатор `%c`. Следующие две строки кода эквивалентны:

```
printf("\n%s\t%c\n","aA",'A');
printf("%c%c%s%c%c%c%c\n",10,13,"aA",9,65,10,13);
```

Коды и символы первой страницы таблицы ASCII, начиная с символа пробела,

```
 32   33 ! 34 " 35 # 36 $ 37 % 38 & 39 ' 40 ( 41 )
 42 * 43 + 44 , 45 - 46 . 47 / 48 0 49 1 50 2 51 3
 52 4 53 5 54 6 55 7 56 8 57 9 58 : 59 ; 60 < 61 =
 62 > 63 ? 64 @ 65 A 66 B 67 C 68 D 69 E 70 F 71 G
 72 H 73 I 74 J 75 K 76 L 77 M 78 N 79 O 80 P 81 Q
 82 R 83 S 84 T 85 U 86 V 87 W 88 X 89 Y 90 Z 91 [
 92 \ 93 ] 94 ^ 95 _ 96 ` 97 a 98 b 99 c100 d101 e
102 f103 g104 h105 i106 j107 k108 l109 m110 n111 o
112 p113 q114 r115 s116 t117 u118 v119 w120 x121 y
122 z123 {124 |125 }126 ~127
```

выводятся с помощью следующего програмного кода:

```
for(i=32; i<128; i++)
   printf("%4d %c  ",i,i);
```

Вторая страница таблицы содержит символы алфавита страны локализации, символы псевдографики и некоторые другие. Для алфавита одной страны могут использоваться разные коды. По умолчанию в консольных приложениях среды VS C++ при вводе применяются коды, отличные от кодов символьных констант, задаваемых в исходном тексте программы. При этом необходимо иметь в виду следующее:

1) вводимые строки с буквами русского алфавита при выводе отображаются правильно, но их начертание в окне наблюдения при отладке изменяется;

2) строки-константы с буквами русского алфавита при выводе отображаются неправильно, но их начертание в окне наблюдения при отладке не изменяется;

3) сравнение одинаковых введенных букв с такими же буквами-константами приводит к отрицательному результату.

Допустимо использование и другого варианта кодирования букв русского алфавита, для чего следует добавить в программу директиву

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

и оператор

```
setlocale( LC_CTYPE, ".1251" );
```

В этом случае можно констатировать следующее:

1) вводимые строки с буквами русского алфавита при выводе отображаются неправильно и их начертание в окне наблюдения при отладке изменяется;

2) строки-константы с буквами русского алфавита при выводе отображаются правильно и их начертание в окне наблюдения при отладке не изменяется;

3) сравнение одинаковых введенных букв с такими же буквами-константами дает отрицательный результат.

Вариант кодирования с использованием кодовой таблицы, устанавливаемой по умолчанию, представляется более предпочтительным, так как многие работы с текстами, например сортировки строк, можно проводить без привлечения символьных констант

русских букв. Если остановиться на этом варианте, то сделать свою программу обработки строк с русскими буквами пригодной для всех случаев работы с вводимыми текстами и задаваемыми в исходном коде программы можно по следующей технологии:

1) все вводимые тексты с русскими буквами сразу перекодировать к кодам соответствующих констант;

2) выполнять все работы с перекодированными введенными текстами;

3) перед выводом строк (с вводимыми и перекодированными русскими буквами, а также русских строк-констант) осуществлять обратное перекодирование.

Для перекодировки символов при вводе можно использовать следующую функцию:

```
char *RusIn(char ss[],const char s[])
{//
//Функция перекодирования кириллицы: источник s, приемник ss
 //Используется после ввода в s
   int i;
   for (i=0;s[i] != '\0'; i++)
   {
      if (s[i] >= -128 && s[i] <= -81)
         ss[i] = (char)(64+s[i]);//A..п
      else if (s[i] <= -17 && s[i] >= -32)
         ss[i] = (char)(16+s[i]);//р..я
      else if (s[i] == -15) //e
         ss[i] = (char)(-72);
      else if (s[i] == -16) //E
         ss[i] = (char)(-88);
      else
         ss[i]= s[i];//остальные символы
   }
   ss[i]='\0';
   return ss;
}
```

Совместить ввод и перекодирование позволяет функция

```
void GetsIn(char ss[],char s[])
{//ввод в s и перекодирование введенного в ss
   char sd[257];
   gets(s);
   RusIn(ss,s);
}
```

Для перекодировки символов при выводе можно использовать следующую функцию:

```
char *RusOut(char ss[],char s[])
{//Функция выполняет обратное перекодирование
   int i;
   for (i=0;s[i] != '\0'; i++)
   {
      if (s[i]>= -64 && s[i] <= -17)
         ss[i]=(-64+s[i]); //A..п
      else if (s[i]>= -16 && s[i] <= 0)
         ss[i]=(char)(-16+s[i]); //р..я
      else if (s[i] == -72)
         ss[i]=(char)(-15); //е
      else if (s[i] == -88)
         ss[i]=(char)(-16); //E
      else
         ss[i]=s[i];
   }
   ss[i]='\0';
   return ss;
}//char *RusOut(char ss[],char s[])
```

Совместить обратное перекодирование и ввод позволяет функция

```
void OutPuts(char s[],int f='\n')
{//преобразование к коду вводимых русских букв и вывод
   char sd[257];
   RusOut(sd,s);
   if(f!='\n')
      printf("%s",sd);
   else
      puts(sd);
}
```

При работе со строками, включающими буквы русского алфавита, желательно создать заголовочный файл (например, `RusString.h` с функциями `RusIn`, `GetsIn`, `RusOut`, `OutPuts`) и подключить его к проекту программы. В дальнейшем будем считать, что такой файл создан и находится в папке D:\temp.

Следующая программа предназначена для проверки работы функций `RusIn`, `GetsIn`, `RusOut`, `OutPuts` и сравнения введенных русских букв с соответствующими константами:

```
#include "stdafx.h"
#include "string.h"
#include "stdio.h"
#include "D:\temp\RusString.h"
int _tmain(int argc, _TCHAR* argv[])
{
   char
      ss[128],//массив для ввода строк
      sss[] = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ абвгдеёж-
зийклмнопрстуфхцчшщьыъэюя";
/* для дальнейших сравнений кодов введенных русских букв до
   перекодирования по RusIn(ss,ss); и после перекодирования
   с кодами констант таких же букв введем по gets(ss);
   следующий текст:
АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЪЭЮЯ
абвгдеежзийклмнопрстуфхцчшщьыъэюя
*/
OutPuts("\n  введите строку из русских:");
   gets(ss);
OutPuts("\n  вывод введенной строки из русских");
OutPuts(" букв без перекодирования по puts");
   puts(ss);
//до перекодирования по RusIn(ss,ss);
//введенных в ss русских букв
OutPuts("\n  до перекодирования по RusIn(ss,ss); ");
OutPuts("введенных в ss русских букв");
if(ss[1] == sss[1])
  OutPuts("код введенной Б совпадает с кодом 'Б'");
else
{
  OutPuts("код введенной Б ");
  OutPuts("НЕ совпадает с кодом 'Б'");
}
   RusIn(ss,ss);//перекодирования
//после перекодирования по RusIn(ss,ss);
//введенных в ss русских букв
OutPuts("\n после перекодирования по ");
OutPuts("RusIn(ss,ss); ");
OutPuts("введенных в ss русских букв");
   if(ss[1] == sss[1])
OutPuts("код введенной Б совпадает с кодом 'Б'");
   else
```

```
    {
OutPuts("код введенной Б НЕ совпадает с кодом Б'");
OutPuts("\n  вывод строки-константы из русских");
OutPuts(" букв с перекодированием по OutPuts");
    }
    OutPuts(sss);
OutPuts("\n  вывод строки-константы из русских");
OutPuts(" букв без перекодирования по puts");
    puts(sss);
    return 0;
}
```

Далее даны сведения о некоторых библиотечных функциях работы со строками, объявленных в заголовочном файле **string.h**. Ниже приведены прототипы функций, в которых тип **size_t** представляет беззнаковое целое, имеющее тот же тип, что и результат оператора **sizeof**:

• **void *memchr(const void *buf, int ch, size_t count)** определяет первое вхождение символа **ch** в первых **count** символах массива **buf** и возвращает указатель на первый из символов **ch** в массиве **buf** или нулевой указатель, если символ **ch** не найден;

• **int memcmp (const void *buf1, const void *buf2, size_t count)** сравнивает в лексико-графическом порядке первые **count** символов двух строк **buf1** и **buf2**; отрицательный результат означает, что первая строка меньше второй, нулевой результат — строки равны, положительный — первая строка больше второй;

• **void *memcpy(void *buf1, const void *buf2, size_t count)** копирует **count** символов из массива **buf2** в массив **buf1**. Если заданные массивы перекрываются, то поведение функции не определено. Функция возвращает значение указателя **buf1**;

• **void *memmove(void *masto, const void *masfrom, size_t count)** копирует **count** символов из массива **masfrom** в массив **masto**. Если заданные массивы перекрываются, то процесс копирования проходит корректно. В этом случае содержимое массива **masfrom** будет помещено в массив **masto**, но содержимое массива **masfrom** изменится. Функция возвращает значение указателя **masto**;

• **void *memset(void *buf, int ch, size_t count)** копирует параметр **ch** в первые **count** символов массива, адресуемого параметром **buf** и возвращает значение указателя **buf**. Функция ис-

пользуется в основном для инициализации массива известным значением;

• `char *strcat(char *str1, const char *str2)` присоединяет к строке `str1` копию строки `str2` и завершает строку `str1` нулевым символом. Конечный нуль-символ, первоначально завершающий строку `str1`, перезаписывается первым символом строки `str2`. Строка `str2` при этом не изменяется. Если заданные массивы перекрываются, то поведение функции не определено. Функция возвращает значение указателя `str1`. При выполнении операций с массивами символов контроль нарушения их границ не осуществляется, поэтому программист должен предусмотреть достаточный размер массива `str1`, позволяющий вместить как его исходное содержимое, так и содержимое массива `str2`. В противном случае не хватит места для записи нуль-символа, что приведет к неправильной обработке массива;

• `char *strchr(const char *str, int ch)` возвращает указатель на первое вхождение параметра `ch` в строку `str`. Если указанный символ не будет найден, то возвращается нулевой указатель;

• `int strcmp(const char *str1, const char *str2)` сравнивает в лексикографическом порядке две заданные строки и возвращает целое значение, определяющее результат сравнения. Отрицательный результат означает, что первая строка меньше второй, нулевой — строки равны, положительный — первая строка больше второй;

• `int strcoll(const char *str1, const char *str2)` сравнивает строку, адресуемую указателем `str1`, со строкой, адресуемой указателем `str2`. Сравнение выполняется с учетом значения параметра `locale`, заданного с помощью функции `setlocale`. Функция возвращает целое значение, определяющее результат сравнения. Отрицательный результат означает, что первая строка меньше второй, нулевой — строки равны, положительный — первая строка больше второй;

• `char *strcpy(char *str1, const char *str2)` копирует содержимое строки `str2` в строку `str1`. Строка `str2` должна содержать завершающий нуль-символ. Функция возвращает значение указателя `str1`;

• `size_t strcspn(const char *str1, const char *str2)` возвращает длину начальной подстроки в строке, адресуемой параметром `str1`, которая не содержит ни одного символа из строки, адресуемой параметром `str2`. Другими словами, функция возвращает индекс первого символа в строке `str1`, который совпадает с любым символом в строке `str2`;

• `char *strerror(int errnum)` возвращает указатель на строку, содержащую системное сообщение об ошибке, связанной со значением `errnum`;

• `size_t strlen(const char *str)` возвращает длину строки, адресуемой параметром `str`, причем строка должна заканчиваться символом конца строки. При этом символ конца строки '`\0`' не учитывается;

• `char *strncat(char *str1, const char *str2, size_t count)` присоединяет к строке, адресуемой параметром `str1`, не более `count` символов строки, адресуемой параметром `str2`, завершая строку `str1` нулевым символом. Конечный нуль-символ, первоначально завершающий строку `str1`, перезаписывается первым символом строки `str2`. Строка `str2` в результате конкатенации не изменяется. Если строки перекрываются, то поведение функции не определено. При использовании этой функции программист должен позаботиться о том, чтобы строка `str1` имела достаточную длину, позволяющую вместить как исходную строку, так и присоединяемую. В противном случае нуль-символ не записывается, что приводит к ошибкам обработки строки;

• `int strncmp(char *str1, const char *str2, size_t count)` сравнивает в лексикографическом порядке не более `count` символов двух заданных строк и возвращает целое значение, определяющее результат сравнения. Отрицательный результат означает, что первая строка меньше второй, нулевой — строки равны, положительный — первая строка больше второй. Строки должны заканчиваться нуль-символами. Если одна из строк содержит менее `count` символов, то сравнение заканчивается при обнаружении первого нуль-символа;

• `char *strncpy(char *str1, const char *str2, size_t count)` копирует не более `count` символов из строки, адресуемой параметром `str2`, в строку, адресуемую параметром `str1`. Строка `str2`

должна содержать завершающий нуль-символ. Функция возвращает значение указателя **str1**. Если заданные массивы символов перекрываются, то поведение функции не определено. Если длина строки, адресуемой параметром **str2**, меньше значения **count**, то в конец строки-результата **str1** добавляются недостающие нулевые символы. Если длина строки, адресуемой параметром **str2**, больше значения **count**, то строка-результат не будет заканчиваться символом конца строки. Функция возвращает значение указателя **str1**;

• **char *strpbrk(const char *str1, const char *str2)** возвращает указатель на первый символ в строке, адресуемый параметром **str1**, который совпадает с любым символом в строке, адресуемой параметром **str2**. Символы конца строки, которыми должны заканчиваться строки, в расчет не принимаются. Если совпадений нет, то возвращается нулевой указатель;

• **char *strrchr(const char *str, int ch)** возвращает указатель на последнее вхождение символа **ch** в строку, адресуемую параметром **str**. Если совпадение не обнаружено, то возвращается нулевой указатель;

• **size_t strspn(const char *str1, const char *str2)** возвращает длину начальной подстроки, адресуемой параметром **str1**, которая состоит только из символов, содержащихся в строке, адресуемой параметром **str2**. Другими словами, функция возвращает индекс первого символа в строке **str1**, который не совпадает ни с одним из символов строки **str2**;

• **char *strstr(const char *str1, const char *str2)** возвращает указатель на первое вхождение подстроки, адресуемой параметром **str2**, в строку, адресуемую параметром **str1**. Если совпадение не обнаружено, то возвращается нулевой указатель;

• **char *strtok(char *str1, const char *str2)** возвращает указатель на следующую лексему (слово) в строке, адресуемой параметром **str1**, и на место разделителя за последним символом лексемы записывает нуль-символ. Символы, образующие строку, адресуемую параметром **str2**, представляют собой разделители, которые определяют лексему. При отсутствии лексемы, подлежащей возврату, возвращается нулевой указатель. Для выделения лексем из строки при первом обращении к функции параметр **str1**

должен указывать на начало этой строки. При последующих обращениях к функции в качестве параметра **str1** нужно использовать нулевой указатель. При каждом обращении к функции можно применять разные наборы разделителей;

• **size_t strxfrm(char *str1, const char *str2, size_t count)** преобразует строку, адресуемую параметром **str2**, таким образом, чтобы ее могла использовать функция **strcmp**, и помещает результат преобразования в строку, адресуемую параметром **str1**. После преобразования результат вызова функции **strcmp**, параметром которой является строка **str1**, будет совпадать с результатом вызова функции **strcoll**, использующей исходную строку, на которую указывает параметр **str2**. В массив, адресуемый параметром **str1**, записывается не более **count** символов. Преобразование связано с учетом особенностей национальных алфавитов кодовой таблицы, устанавливаемой функцией **setlocale**.

2.2. Примеры программ обработки строк (массивов символов)

Пример 1. Составить программу нахождения в тексте на русском языке слова, содержащего максимальное количество букв '**а**', '**ы**' и '**и**'.

Решение этой задачи разобьем на три этапа.

На первом этапе выполним ввод слов (в конце слов допускаются знаки препинания) текста в элементы массива, состоящего из массивов символов (массива слов), и изменение кодов символов введенных слов на принятые для символьных констант.

На втором этапе найдем в массиве слов индекс слова, удовлетворяющего условию задачи, и определим количество символов в слове.

На третьем этапе при отсутствии нужных слов в тексте введем соответствующее сообщение, в противном случае выведем найденное слово, предварительно удалив в его конце знак препинания, если таковой есть, и перекодировав символы для правильного вывода.

Текст программы:

```
#include "stdafx.h"
#include "string.h"
```

```
#include "stdio.h"
#include "D:\\temp\\RusString.h"
int _tmain(int argc, _TCHAR* argv[])
{
   char
      s[100][20],//массив ля ввода слов
      smax[20], //для слова с максимальным
         //количеством гласных букв
      cc
      ;
   int
      ns,//количество введенных в s слов
      nmax, //искомое максимальным количеством
         // букв А и Я
      imax,//место этого слова в массиве s
      lmax,//длина этого слова
      i,j,k
      ;
/* строка для отладки
искомое, максимальным: количеством гласных букв.
*/
//Этап 1
   //ввод слов в массивы s[i],i=0,1,2,.. и
   //подсчет их количества ns

   OutPuts("Введите текст");
   for(i=0;;i++)
   {
      scanf("%s%c",s[i],&cc);
//    printf("%s%d\n",s[i],cc);//для отладки
      RusIn(s[i],s[i]);//перекодировка для
            // сравнения с константами
      if(cc==10)
      {
         ns=i;
//       printf("ns=%d\n",ns+1);//для отладки
         break;
      }
   }
//Этап 2
//поиск слова с максимальным количеством букв а ы и
   ns++;
   nmax=0;
   for(i=0;i<ns;i++)
```

```
    {
        k=0;
        for(j=0;s[i][j]!=0;j++)
        {

            switch(s[i][j])
            {
                case 'а':case 'ы':case 'и':
                    k++;
                    break;
            }
        }//for(j=0;s[i][j]!=0;j++)
        if(k>nmax)
        {
            nmax=k;
            imax=i;
            lmax=j;
        }
    }//for(i=0;i<ns;i++)
//Этап 3
    if(nmax==0)
//правильный вывод строковой константы с русскими буквами
        OutPuts("Нет слов с буквами а ы и");
    else
    {
        if(s[imax][lmax-1]>32)
            s[imax][lmax-1]=0;//удалить знак
                              //препинания
        //Подготовить строку в s[imax]
        //для вывода и вывести
        OutPuts("Найденное слово:");
        OutPuts(s[imax]);
    }

    return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите текст
искомое, максимальным: с количеством гласных букв.

Найденное слово: максимальным
Для продолжения нажмите любую клавишу . . .
```

Пример 2. Во введенной строке символов, включающей и буквы русского алфавита, найти все слова, начинающиеся с заглавных букв. Найденные слова вывести в алфавитном порядке.

Решение этой задачи разобьем на три этапа. На первом этапе выделим из строки все слова. Под словом будем понимать произвольную последовательность символов, не содержащую символов-разделителей. К символам-разделителям отнесем следующие символы: пробел, запятую, точку, вопросительный знак, восклицательный знак, двоеточие, точку с запятой, тире. Следует отметить, что каждый пользователь может сам сформировать этот список в соответствии со своими запросами. Выделить слова можно, используя функцию **strtok**. Эта функция позволяет последовательно находить указатель на начало очередного слова в исходной строке и отмечать нуль-символом конец слова. Найденные указатели сохраняют в массиве **sl** (т. е. массиве всех слов) и подсчитывают их количество.

На втором этапе из найденного массива всех слов выберем только требуемые слова и сформируем из них второй массив указателей **zag** (массив требуемых слов).

На третьем этапе выполним сортировку указателей во втором массиве по возрастанию слов с использованием подготовленной функции **sort**.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "string.h"
#include "D:\\temp\\RusString.h"

void sort(char *zag[], int n);

int _tmain(int argc, _TCHAR* argv[])
{
   const int nn=250;
   char s[nn], //исходная строка символов
  *sl[nn/2],//массив указателей на слова строки
  *c, //указатель на очередное слово
  *zag[nn/2],//массив указателей на слова,
        //начинающиеся с заглавных букв
ch;
   int m,i,n,o;
```

```
/* текст для отладки
Подключите RusString.h. Он содержит функции RusIn, GetsIn,
RusOut, OutPuts для работ с русскими буквами.
*/
OutPuts("Подготовьте текст и нажмите Enter:\n");
   gets(s);
   OutPuts("\nКонтрольный вывод:\n");
   puts(s);
   RusIn(s,s);//перекодирование введенного текста
   c=strtok(s," ,.?!:;"); // поиск первого слова
   m=0;
   while(c!=NULL) //цикл поиска последующих слов
   {
      sl[m]=c; //занесение указателя на
   //очередное слово в массив указателей на слова
   m++;
   c=strtok(NULL," ,.?!:;"); //поиск очередного слова
   }
   if(m==0)
     //не найдено ни одного слова
     OutPuts("Во введенном тексте нет слов\n");
   else
   {
      //вывод найденных слов
      OutPuts ("\nСписок найденных слов:");
      for(i=0;i<m;i++)
      OutPuts(sl[i]);// printf("%s\n",sl[i]);
   //поиск слов, начинающихся на заглавные буквы
      n=0;
      o=0;
      for(i=0;i<m;i++)
         if (sl[i][0]>='A'&&sl[i][0]<='Z'
          || sl[i][0]>='A'&&sl[i][0]<='Я')
      {//Найдено слово, начинающееся с большой буквы
//Включение этого слова в массив zag требуемых слов
            zag[n]=sl[i];
            n++;
         }
      if(n>0)
      {
   //Вывод слов, начинающихся с большой буквы
      OutPuts("\nСписок слов, начинающихся с");
      OutPuts("большой буквы:");
```

```
      for(i=0;i<n;i++)
      OutPuts(zag[i]);
   //упорядочение слов в массиве zag слов и вывод
   sort(zag,n);
   OutPuts("\nСписок слов после упорядочения:");
   for(i=0;i<n;i++)
         OutPuts(zag[i]);//printf("%s\n",zag[i]);
      }
else
   OutPuts("\nСписок слов, начинающихся с большой буквы:");
   }
   getch();
   return 0;
}

void sort(char *zag[], int n)
{//функция сортировки массива слов методом пузырька
   int i,k;
   char *c;
   bool flag;
   k=0;
   do
   {
      flag=false;
      k++;
      for (i=0;i<n-k;i++)
         if (strcmp(zag[i],zag[i+1])>0)
         {

            c=zag[i];
               zag[i]=zag[i+1];
            zag[i+1]=c;
            flag=true;
         }
   }while(flag);
}
```

Протокол ввода/вывода имеет следующий вид:

```
Подготовьте текст и нажмите Enter:
Подключите RusString.h. Он содержит функции RusIn, GetsIn,
RusOut, OutPuts для работ с русскими буквами.

Контрольный вывод:
```

```
Подключите RusString.h. Он содержит функции RusIn, GetsIn,
RusOut, OutPuts для работ с русскими буквами.

Список найденных слов:
Подключите
RusString
h
Он
содержит
функции
RusIn
GetsIn
RusOut
OutPuts
для
работ
с
русскими
буквами

Список слов, начинающихся с большой буквы:
Подключите
RusString
Он
RusIn
GetsIn
RusOut
OutPuts

Список слов после упорядочения:
GetsIn
OutPuts
RusIn
RusOut
RusString
Он
Подключите
```

Пример 3. Найти в строке символов все слова, в которых количество цифр составляет не менее заданного значения.

При решении этой задачи так же, как и в примере 2, сначала строим массив всех слов `sl`, содержащихся в исходной строке. За-

тем с помощью функции **zifra** определяем, содержит ли слово требуемое количество цифровых символов. Слова, удовлетворяющие поставленному условию, переписываем во второй массив **zif** (массив требуемых слов).

Перекодирование введенных русских букв не выполняем, поскольку сравнений с константами не требуется, но для вывода поясняющих текстов — констант на русском языке используем функцию **OutPuts**.

Текст программы:

```
//Найти в строке все слова, в которых количество цифр
//не менее заданного
#include "stdafx.h"
#include "string.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

bool zifra(char *s,int k)
{//подпрограмма анализа количества цифр в слове
   int n;
   n=0; //текущее количество цифровых символов
   for (int j=0; j<strlen(s) && n<k; j++)
   {
      if (s[j]>='0'&&s[j]<='9') //анализ
         //очередного символа на цифру
      n++;
   }
   if (n>=k)
      return true;
   else
      return false;
}

int _tmain(int argc, _TCHAR* argv[])

{const int nn=250;//максимальная длина
                  //строки символов
   char
      s[nn], //исходная строка символов
  *sl[nn/2], //массив указателей на слова строки
  *c, //указатель на текущее слово
```

```
  *zif[nn/2]//массив указатель на слова,
      // удовлетворяющие условию
      ;
   int m,i,n,k;
/* текст для отладки
ФЫВ1И,Й66. , 333  .,4У4У4УQ:?AS5;ZXC6V6V6V6V6:
*/
   //Ввод исходных данных
   OutPuts("Подготовьте текст и нажмите Enter:");
   gets(s);
   OutPuts("\nКонтрольный вывод:");
   puts(s);
   OutPuts("\nВведите количество цифр: ",0);
   scanf("%d",&k);
   //Выделение слов
   c=strtok(s," ,.?!;:");
   m=0;
   while (c!=NULL)
   {
      sl[m]=c;
      m++;
      c=strtok(NULL," ,.?!;:");
   }
 //Вывод результата по выделению слов
   if (m==0)
      OutPuts("\nВ строке нет слов:\n");
   else
   {
      //вывод найденных слов
      OutPuts ("\nСписок найденных слов:");
      for (i=0;i<m;i++)
         printf("%s\n",sl[i]);
      //поиск слов с количеством цифр,
      //большим заданного k
      n=0;
      for (i=0;i<m;i++) //цикл по всем
                  //выделенным словам
         if (zifra(sl[i],k))
         {
         zif[n]=sl[i];//занесение слова,
         //удовлетворяющего условию,
         // в результирующий массив
```

```
          n++;
          }
       if (n>0)
       {//вывод слов с количеством цифр,
        //большим заданного k
        OutPuts("\nСписок слов с количеством");
        OutPuts(" цифр, большим заданного: ");
          printf("k=%d:\n",k);
        for(i=0;i<n;i++)
             printf("%s\n",zif[i]);
       }
       else
         printf("%s k=%d s\n",
RusOut(s,
"\nСлов с количеством цифр, большим заданного"),
k,RusOut(&s[50]," НЕТ!\n"));
   }
   getch();
   return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Подготовьте текст и нажмите Enter:
ФЫВ1И,Й66. , 333 .,4У4У4УQ:?AS5;ZXC6V6V6V6V6:

Контрольный вывод:
ФЫВ1И,Й66. , 333 .,4У4У4УQ:?AS5;ZXC6V6V6V6V6:
Введите количество цифр: 2

Список найденных слов:
ФЫВ1И
Й66
333
4У4У4УQ
AS5
ZXC6V6V6V6V6

Список слов с количеством цифр, большим заданного k=2:
Й66
333
4У4У4УQ
ZXC6V6V6V6V6
```

Пример 4. Подсчитать в строке количество симметричных слов, состоящих из нескольких букв, и вывести результат.

Для решения поставленной задачи надо выделять слова и проверять очередное слово на симметричность. Проверка на симметричность выполняется путем сравнения символов, находящихся на одинаковом расстоянии от центра симметрии слова. Если очередную пару сравниваемых символов образуют одинаковые символы, то процесс сравнения продолжается. Если очередную пару образуют разные символы, то продолжать проверку нет смысла, в этом случае выполняется выход из цикла и формируется признак несимметричности.

При проверке слова на симметричность можно отказаться от перекодирования вводимых данных для русских букв, так как они не будут сравниваться с соответствующими символами-константами. Если необходимо, чтобы и все выводимые поясняющие тексты были на русском языке и не нуждались в перекодировании для правильного вывода, то следует использовать альтернативное кодирование с помощью заголовочного файла `locale.h` и оператора `setlocale(LC_CTYPE, ".1251" );`. В этом случае при наличии русских букв для контрольного вывода строки нужно отказаться от контрольного вывода вводимых данных в готовой программе. Однако при отладке программы, когда требуется наблюдать текущие значения введенных символов, их необходимо перекодировать. В представленной ниже программе операторы, подлежащие удалению после завершения отладки, помечены комментариями **`//для отладки.`**

Текст программы:

```
// Подсчет количества симметричных слов

#include "stdafx.h"
#include "conio.h"
#include "string.h"
#include <locale.h>
#include "D:\temp\RusString.h"//Для отладки
int _tmain(int argc, _TCHAR* argv[])
{  const int nn=250;
   chars[nn], //исходная строка символов
  *c, //указатель на очередное слово
  *sim[nn/2]; //массив указателей на
              //слова,являющиеся симметричными
```

```
   int i,l,m,n;
   bool flag; //признак симметричности слова
setlocale( LC_CTYPE, ".1251" );
printf("Введите строку\n");
   gets(s);
   printf("\nnКонтрольный вывод:\n");//Для
                          //отладки
   RusIn(s,s); //Для отладки
   puts(s);//Для отладки
/* Для отладки
 АА,,я "АННА" ОНО казак,  шалаш?
*/
   c=strtok(s," ,.?!:;"); // поиск первого слова
   n=0;
   while(c!=NULL) //цикл поиска последующих слов
   {
      m=strlen(c);
      RusIn(c,c);//Для отладки
      for(i=0;i<m/2;i++)
         if(c[i]!=c[m-1-i])
            break;
      if(m>1 && i==m/2)
         n++;
  c=strtok(NULL," ,.?!:;");//поиск очередного слова
   }

   if (n==0)
      printf("\nСимметричных слов нет");
   else
      printf(
"\nКоличество симметричных слов равно %d\n",n);

   getch();
   return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите строку
 АА,,я "АННА" ОНО казак, шалаш?

Количество симметричных слов равно 5
```

Пример 5. Найти в строке, содержащей буквы только латинского алфавита, максимальное слово и подсчитать, сколько раз в нем встретился каждый символ этого слова.

Для решения этой задачи необходимо выделить все слова, содержащиеся в строке. При выделении очередного слова следует сравнить его с максимальным словом и, если оно превышает максимальное, скопировать текущее слово в максимальное.

Чтобы определить, сколько раз встретился каждый символ в слове, необходимо каждый символ слова сравнить с каждым из последующих символов этого слова. Для этого надо записать вложенный цикл. Во внешнем цикле перебираются все символы слова, а во внутреннем — все последующие символы слова, которые сравниваются с текущим символом. При их совпадении счетчик увеличивается на единицу. Начальное значение счетчика устанавливается равным единице, так как наличие очередного символа означает, что один раз этот символ уже встретился.

При нахождении совпадающего символа его надо удалить из строки, чтобы в дальнейшем этот символ не мог рассматриваться в качестве нового символа. Для этого следует в еще одном вложенном (третьем) цикле осуществить сдвиг всех последующих символов на одну позицию в сторону начала слова. В этом случае количество анализируемых символов (длина строки) уменьшается на единицу. При этом также надо на единицу уменьшить текущее значение параметра цикла, поскольку после сдвига символов строки влево в текущей позиции оказывается еще не анализировавшийся символ.

Текст программы:

```
// Нахождение максимального слова в строке,
//содержащей только латинские буквы, и подсчет,
//сколько раз в нем встретился каждый символ этого слова
#include "stdafx.h"
#include "conio.h"
#include "string.h"
#include <locale.h>

int _tmain(int argc, _TCHAR* argv[])
{  char s[100], //исходная строка символов
max[100];  //максимальное слово
char *c; //указатель на очередное слово
```

```
int i,j,n,k ;
/* Строка для отладки
prst ppppppprrrrttt trsptrsrppp
*/
   setlocale( LC_CTYPE, ".1251" );
   printf("Введите строку:\n");
   gets(s);
   printf("Исходная строка  -%s",s);
   //задание начального значения для максимального слова -
   //пустое слово
   strcpy(max,"");
   c=strtok(s," ,.?!:;");
   while (c!=NULL)//цикл выделения слов и поиска максимального
   {
      if (strcmp(c,max)>0)
         strcpy(max,c);
      c=strtok(NULL," ,.?!:;");
   }
   n=strlen(max);
   printf("\nМаксимальное слово - %s",max);
   printf("\nбуквакаличество");
   for (i=0;i<n;i++)//цикл анализа каждого
               // символа слова
   {
      k=1;
   //цикл сравнения очередного символа со всеми
      //последующими символами слова
      for (j=i+1;j<n;j++)

         if (max[i]==max[j])
         {
            k++;
//удаление символа-копии (перепись всех последующих
//символов со сдвигом влево на одну позицию
            for (int l=j;l<n;l++)
               max[l]=max[l+1];
            n--;
            j--;
         }
      printf("\n %c-%d",max[i],k);
   }
 getch();
 return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите строку:
prst ppppppprrrrttt trsptrsrppp
Исходная строка -prst ppppppprrrrttt trsptrsrppp
Максимальное слово - trsptrsrppp
букваколичество
 t-2
 r-3
 s-2
 p-4
```

2.3. Задания на обработку строк

Слово — последовательность символов, не содержащая символов-разделителей. Слова друг от друга отделяются одним или несколькими символами-разделителями. В начале и конце строки могут стоять разделители (произвольное количество). Строки могут содержать буквы русского алфавита. Пояснения к выводимым данным и полученным результатам должны быть на русском языке. Следует минимизировать количество перекодировок строк в рабочем варианте программы, а после отладки программы отладочные операторы нужно превратить в комментарии.

1. Найти в строке все слова, начинающиеся с заданной буквы. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

2. Найти в строке все слова, оканчивающиеся на заданную букву. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

3. Найти в строке все слова, в которых заданная буква встречается более одного раза. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

4. Найти в строке все слова, являющиеся симметричными. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

5. Найти в строке все слова, в которых гласные и согласные чередуются. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

6. Найти в строке все слова, в которых буквы не повторяются. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

7. Найти в строке все слова, в которых каждая буква встречается более одного раза. Найденные слова распечатать в алфавитном порядке. Если нужных слов нет, то выдать сообщение.

8. Найти в строке самое короткое слово и самое длинное. Подсчитать, сколько раз каждая буква слова встречается в этом слове.

9. Найти в строке самое короткое слово и самое длинное слово, в которых нет повторяющихся букв.

10. Найти в строке самое короткое слово и самое длинное слово, в которых каждая буква встречается ровно по два раза.

11. Найти в строке самое короткое слово и самое длинное слово, в которых одновременно присутствуют все гласные буквы.

12. Найти в строке все слова, начинающиеся с заданной буквы. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

13. Найти в строке все слова, оканчивающиеся на заданную букву. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

14. Найти в строке все слова, в которых заданная буква встречается более одного раза. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

15. Найти в строке все слова, являющиеся несимметричными. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

16. Найти в строке все слова, в которых гласные и согласные чередуются. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

17. Найти в строке все слова, в которых есть не менее трех повторяющихся букв. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

18. Найти в строке все слова, в которых каждая гласная буква встречается более одного раза. Найденные слова распечатать в порядке, обратном алфавитному. Если нужных слов нет, то выдать сообщение.

19. Найти в строке самое короткое слово и самое длинное слово, которые начинаются и оканчиваются на заданные буквы. Подсчитать, сколько раз каждая буква слова встречается в этих словах.

20. Найти в строке самое короткое слово и самое длинное слово, в которых средняя буква совпадает с заданной. Рассматривать

слова только с нечетным количеством букв. Если таких слов нет, то выдать сообщение.

21. Найти в строке все слова, начинающиеся с заглавной буквы, и распечатать их в алфавитном порядке. Если таких слов нет, то выдать сообщение.

22. Найти в строке все слова, начинающиеся с гласной буквы и оканчивающиеся согласной. Найденные слова распечатать в алфавитном порядке. Если таких слов нет, то выдать сообщение.

23. Найти в строке все слова, начинающиеся на гласную букву и оканчивающиеся гласной. Найденные слова распечатать в порядке, обратном алфавитному. Если таких слов нет, то выдать сообщение.

24. Найти в строке все слова, начинающиеся с заглавной согласной буквы и заканчивающиеся согласной. Найденные слова распечатать в алфавитном порядке. Если таких слов нет, то выдать сообщение.

25. Найти в строке все слова, начинающиеся с заглавной гласной буквы и оканчивающиеся гласной. Найденные слова распечатать в порядке, обратном алфавитному. Если таких слов нет, то выдать сообщение.

26. Найти в строке все слова, в которых все символы встречаются ровно по два раза. Распечатать их в алфавитном порядке. Если таких слов нет, то выдать сообщение.

27. Найти в строке все слова, в которых буквенные и небуквенные символы чередуются. Найденные слова распечатать в алфавитном порядке. Если таких слов нет, то выдать сообщение.

28. Найти в строке все симметричные слова, начинающиеся с гласной буквы и оканчивающиеся гласной. Найденные слова распечатать в порядке, обратном алфавитному. Если таких слов нет, то выдать сообщение.

29. Найти в строке все слова, начинающиеся с заглавной согласной буквы и оканчивающиеся согласной. Найденные слова распечатать в порядке, обратном алфавитному. Если таких слов нет, то выдать сообщение.

30. Найти в строке все слова, начинающиеся с заглавной гласной буквы, в которых количество согласных превышает количество гласных. Найденные слова распечатать в порядке, обратном алфавитному. Если таких слов нет, то выдать сообщение.

Вопросы для самопроверки

1. Как записывается строковая константа?
2. Что называют строкой в языке Си?
3. Как определить символьный массив с начальным значением-строкой?
4. Каким символом заканчивается строка в языке Си?
5. Может ли строка быть пустой (иметь нулевую длину)?
6. Можно ли использовать оператор присваивания для копирования строки из одного символьного массива в другой?
7. Как скопировать строку в символьный массив?
8. Как определить длину строки?
9. Можно ли сравнивать строки?
10. Как формулируется правило сравнения строк?
11. Какая библиотечная функция используется для ввода строк?
12. Какая библиотечная функция используется для вывода строк?
13. Какой формат применяется для вывода строки функцией printf?
14. Как можно укоротить строку, оставив только заданное количество передних символов?
15. Как преобразовать строковое представление числа к целому типу?
16. Как преобразовать строковое представление числа к вещественному типу?
17. Как преобразовать целое число в строку?
18. Как преобразовать вещественное число в строку в естественной форме, в экспоненциальной форме?
19. Как выполнить конкатенацию строк?
20. Какую библиотечную функцию следует использовать для проверки вхождения одной строки в другую строку?
21. Какая библиотечная функция предназначена для проверки вхождения символа в строку?
22. Какая библиотечная функция используется для копирования строки?

3. Типы данных «структура» и «объединение»

3.1. Общие сведения о структурах

Структура — это совокупность полей, принадлежащих к различным типам. Поля структуры могут иметь любой тип, кроме типа этой структуры, но могут быть указателями на него. Для объявления структур используется ключевое слово struct, после него указывается имя типа (структуры), за которым в фигурных скобках следуют объявления типа каждого поля:

```
struct [имя типа]
{
   тип1 поле1;
   тип2 поле2;
   ...
   типN полеN;
};
```

Например, структура, представляющая собой точку на плоскости, задаваемую двумя координатами, может быть объявлена следующим образом:

```
struct point
{
   float x;
   float y;
};
```

или

```
typedef struct
{
   float x;
   float y;
} point;
```

Поля структуры называют также членами, элементами. С помощью структур обычно описывают сложные данные, характери-

зующиеся несколькими значениями различных типов. Например, сотрудник может быть представлен фамилией, именем, отчеством, полом, годом рождения, окладом; студент — фамилией, именем, отчеством, полом, годом рождения и названием группы, оценками. В этих случаях удобно характеризовать каждый объект не совокупностью отдельных переменных, а одной переменной, имеющей несколько полей, причем эти поля рассматриваются как одно целое, а не как разрозненный набор элементов (переменных).

В программе поля структуры могут иметь те же имена, что и другие переменные и поля в других структурах. В разных структурах могут быть одинаковые названия полей. Однако это может приводить к ошибкам (можно легко перепутать поля разных структур), поэтому такой подход не соответствует принципам хорошего стиля программирования.

Имя структуры при ее объявлении фактически означает объявление нового типа данных. Имя структуры можно рассматривать далее как имя типа (например, `point a,b;`). Можно не вводить имя типа, а после объявления структуры указать имена переменных:

```
struct point
{
   float x,y;
} a,b;
```

Имя типа структуры можно использовать сразу после его объявления, а определение структуры дать позже. Так обычно поступают при объявлении типа структуры, являющейся элементом списка:

```
struct tElem; //объявление
struct tUkaz
{
   tElem *p;
   tUkaz *right, *left;
};
struct tElem //определение
{
   int nom;
   char fam[30];
};
```

Для инициализации структур значения ее элементов записывают в фигурных скобках в порядке их описания:

```
struct
{
   char fam[30];
   int god;
   float doch;
} person = {"Петров", 1958, 1050.56};
```

При инициализации массива структур следует заключать в фигурные скобки каждый элемент массива, принимая во внимание, что многомерный массив — это массив, состоящий из массивов:

```
struct mpoint
{
   float x,y;
} mas[2][3]={
      {{2.2,-4.4}, {4.5, -9.8}, {7.7, -6.6}},
      {{1.1,0.7}, {34.6,-12.5}, {9.9,-0.5}}
       };
```

Для переменных одного и того же типа определена операция присваивания, при выполнении которой происходит поэлементное копирование полей одной структуры в другую структуру. Например, если объявлены переменные

```
point p1, p2;
```

то можно записать оператор присваивания

```
p2=p1;
```

Операции над структурой выполняются для отдельных полей, т. е. проводится поэлементная обработка.

Структуру можно передавать в функцию и возвращать из функции в качестве результата.

При работе со структурами следует принимать во внимание, что размер памяти, занимаемый структурой, может не равняться сумме размеров ее элементов, так как поля могут выравниваться по границе слов.

Доступ к полям структуры осуществляется с помощью операции выбора, обозначаемой символом точка '.', если работа ведется непосредственно с переменной, и с помощью той же операции, но

обозначаемой в виде стрелки «->» (парой знаков минус и больше), если используется указатель на структуру, например:

```
point a, *b;
   a.x=4.4; a.y=-9.7;
   b->x=34.5; b->y=8.6;
```

Элементом структуры может быть другая структура, объявленная ранее (с другим именем типа), или определенная внутри этой структуры (с именем типа или без). В этом случае доступ к ее элементам выполняется с помощью двух операций выбора:

```
struct S1
{
   int i;
   float r;
};
struct S2
{
   S1 z; //поле типа структура S1
   float c;
};
S2 y[2];
y[0].z.i=2;//поле i структуры типа S1
y[0].z.r=2.2; //поле r структуры типа S1
y[0].c=8.02;
y[1].z.i=6;//поле i структуры типа S1
y[1].z.r=-12.3; //поле r структуры типа S1
y[1].c=1.75;
```

Поля разных структур могут иметь одинаковые имена, при этом никаких проблем не возникает, поскольку они имеют разные области видимости. Вне области видимости полей структуры имена ее полей могут использоваться и для других объектов. Например, в следующем фрагменте программы имя `q` является и именем поля, и именем переменной определяемой структуры, а имя `y` присвоено имени поля и простой переменной типа `float`:

```
struct {float q,y;} q;
float y;
q.q=5.5f; q.y=-90.9f; y=10.1f;
printf("a.a=%5.1f a.y=%5.1f y=%5.1f",q.q,q.y,y);
```

3.2. Примеры программ обработки структур

Пример 1. Массив структур, содержащих два поля — фамилию и рост, упорядочить по полю «рост».

Текст программы:

```
//Упорядочить массив структур
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

typedef  struct
{
   char f[30];//фамилия
   int r; //рост
}sap;
int _tmain(int argc, _TCHAR* argv[])
{
   int n,k,kol,i,flag;
   sap a[10],b;
OutPuts("Введите количество n элементов массива ");
      //printf("wwedite kol-wo n\n ");
   scanf("%d",&n);
   printf("\n");
   OutPuts("Введите элементы массива структур");
  //printf("wwedite elem massiwa structur\n");
   for(i=0;i<n;i++)
   {
      printRus("Фамилия: ");
      scanf("%s",a[i].f);
      printRus("Рост: ");
      scanf("%d",&a[i].r);
      printf("\n");
   }
   OutPuts("Контрольный вывод");
  //printf("sapisi \n");
   for(i=0;i<n;i++)
   {
      printf("%s  ",a[i].f);
      printf("%d \n",a[i].r);
   }
   printf("\n");
```

```
//сортировка методом пузырька массива структур
k=0;
do
{
       flag=0;
       k++;
       for (i=0;i<n-k;i++)
          if (a[i].r>a[i+1].r)
          {
             b=a[i];
             a[i]=a[i+1];
             a[i+1]=b;
             flag=1;
          }
} while (flag);
OutPuts("Вывод упорядоченного массива структур");
 // printf("sapisi \n");
   for(i=0;i<n;i++)
   {
      printf("%s  ",a[i].f);
      printf("%d \n",a[i].r);
   }
   getch();
   return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите количество элементов массива n
3

Введите элементы массива структур
Фамилия: Сидоров
Рост: 183

Фамилия: Петров
Рост: 175

Фамилия: Иванов
Рост: 168

Контрольный вывод
Сидоров 183
```

```
Петров 175
Иванов 168

Вывод упорядоченного массива структур
Иванов 168
Петров 175
Сидоров 183
*/
```

Пример 2. Применить структуру для передачи массивов в функцию.

Структуры удобно использовать при передаче массивов в качестве параметров в функцию. При передаче массивов (особенно двумерных) возникают определенные проблемы, связанные с обработкой переменных, являющихся указателями на указатель. Использование структур значительно упрощает решение этой задачи. Можно объявить структуру, содержащую в качестве полей передаваемый массив и его размеры (для одномерного массива — количество элементов, для двумерного — количество строк и столбцов). В качестве параметра в функцию передается адрес структуры (формальный параметр — простая переменная типа указатель).

Чтобы использовать структуру при обработке матриц, необходимо составить функцию, которая на основе матрицы $A(mm, nn)$, где $mm \le 15$, $nn \le 20$, формирует матрицу B по следующему правилу: если элемент матрицы A меньше нуля, то одноименный элемент результирующей матрицы вычисляется как сумма этого элемента и первого элемента строки матрицы A, в которой он расположен, в противном случае очередной элемент определяется как частное от деления элемента на первый элемент строки (предполагается, что первые элементы строк не равны нулю).

Текст программы:

```
//использование структуры для передачи массива
//

#include "stdafx.h"
#include <conio.h>
#include "D:\\temp\\RusString.h"

   const int nn=30;
```

```
   typedef  struct
   {
      int m,n;
      float mas[nn][nn];
   }sap;

   void f(sap a,sap *b);

int _tmain(int argc, _TCHAR* argv[])
{
   int i,j;
   sap a,b;
   OutPuts("Введите количество строк m ");
   OutPuts("и столбцов n массива");
   scanf("%d%d",&a.m,&a.n);
   OutPuts("Введите элементы массива");
   for (i=0;i<a.m;i++)
      for (j=0;j<a.n;j++)
         scanf("%f",&a.mas[i][j]);
   f(a,&b);
   OutPuts("Результат");
   for (i=0;i<b.m;i++)
   {
      for (j=0;j<b.n;j++)
      printf("%6.2f  ",b.mas[i][j]);
      printf("\n");
   }
   getch();
   return 0;
}

void f(sap a,sap *b)
{//функция формирования новой матрицы
   int i,j;
   b->m=a.m;
   b->n=a.n;
   for (i=0;i<a.m;i++)
     for (j=0;j<a.n;j++)
       if (a.mas[i][j]<0)
      b->mas[i][j]=a.mas[i][j]+a.mas[i][0];
       else
      b->mas[i][j]=a.mas[i][j]/a.mas[i][0];
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите количество строк m и столбцов n массива
2 3
Введите элементы массива
2 -3 4
-3 3 -4
Результат
 1.00-1.00 2.00
 -6.00-1.00-7.00
*/
```

Пример 3. Задан массив структур с полями следующего вида: фамилия спортсмена, название страны, количество оценок спортсмена, массив значений оценок (не более пяти), средняя оценка. Среднюю оценку вычислить при вводе данных. Упорядочить массив по убыванию средней оценки. Определить страну, чьи спортсмены имеют наибольшую сумму средних оценок.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "string.h"
#include "D:\\temp\\RusString.h"

struct sport{ //исходная структура
   char fam[30];
   char stran[30];
   int k;
   float oc[5];
   float sr;
};
struct res{ //результирующая структура
 char stran[30];
 float s;
};
const int nn=20;
void wwod(int *n,sport a[]);
void wiwod(int n,sport a[]);
void upor(int n,sport a[]);
void pobed(int n,sport a[],res *max);

int _tmain(int argc, _TCHAR* argv[])
```

```
{ int n;
sport a[nn];
res max;
wwod(&n,a);
//wiwod(n,a);//контрольный вывод
upor(n,a);
   OutPuts("\nУпорядоченные структуры:\n");
wiwod(n,a);
pobed(n,a,&max);
printRus("\n\nСтрана победителя - ");
printf("%s\n", max.stran);
printRus("Средняя оценка победителя ");
printf("%6.2f\n", max.s);
getch();
   return 0;
}
//функция ввода исходных данных - количества
//элементов массива и массива структур
void wwod(int *n,sport a[])
{
   int i;
   OutPuts("Введите количество структур");
   scanf("%d",n);
   OutPuts("\nВведите данные в структуры\n");
   for (i=0;i<*n;i++)
   {
      printRus("Введите фамилию ");
      scanf("%s",a[i].fam);
      printRus("Введите страну ");
      scanf("%s",a[i].stran);
      printRus("Введите количество оценок ");
      scanf("%d",&a[i].k);
      printRus("Введите оценки " );
      a[i].sr=0;
      for (int j=0;j<a[i].k;j++)
      {
         scanf("%f",&a[i].oc[j]);
         a[i].sr+=a[i].oc[j];
      }
      a[i].sr/=a[i].k;
   }
}
```

```
void wiwod(int n,sport a[])
{//функция вывода массива структур
   for (int i=0;i<n;i++)
   {
      printRus("фамилия ");
      puts(a[i].fam);
      printRus("страна ");
      puts(a[i].stran);
      printRus("количество оценок ");
      printf("%2d",a[i].k);
      printRus("\nоценки ");
      for (int j=0;j<a[i].k;j++)
      {
         printf("%6.2f",a[i].oc[j]);
      }
      printRus("\nсредняя оценка ");
      printf("%6.2f",a[i].sr);
      printf("\n-----------------\n");
   }
}
//функция упорядочения массива структур
void upor(int n,sport a[])
{ int i,k;
 bool flag;
 sport b;
 k=0;
 do
 { flag=false;
 k++;
   for (i=0;i<n-k;i++)
if (a[i].sr<a[i+1].sr)
    { b=a[i];
    a[i]=a[i+1];
      a[i+1]=b;
      flag=true;
    }
}while (flag);
}
void pobed(int n,sport a[],res *max)
{//функция определения страны спортсмена
 //с максимальной средней оценкой
   int i,j,kol;
   res b[nn];
   bool flag;
```

```
   kol=-1;
   //формирование массива структур,
   //включающих название страны и средние оценки
   for (i=0;i<n;i++)
   {
      j=0;
      flag=true;
//цикл проверки наличия информации о текущей стране
      //в результирующем массиве
      while ((j<=kol) && flag)
      {
         if (strcmp(a[i].stran,b[j].stran)==0)
         {
           flag=false;
   //если страна найдена, то к сумме баллов
   //прибавляется балл очередного спортсмена
   //этой страны
            b[j].s+=a[i].sr;
         }
         else
            j++;
      }
   //если текущая страна отсутствует в результирующем массиве,
   //то структура добавляется в новый массив
      if (flag)
      {
         kol++;
         strcpy(b[kol].stran,a[i].stran);
         b[kol].s=a[i].sr;
      }
   }
//поиск страны с максимальной суммой средних оценок
   max->s=-1;
   for(i=0;i<=kol;i++)
      if (b[i].s>max->s)
         *max=b[i];
   for(i=0;i<=kol;i++)
   {
      printRus("\nстрана - ");
      printf("%s",b[i].stran);
      printRus(",  сумма средних оценок  - ");
      printf("%6.2f",b[i].s);
   }
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите количество структур
4

Введите данные в структуры

Введите фамилию Иванов
Введите страну Россия
Введите количество оценок 2
Введите оценки 2 3
Введите фамилию Кук
Введите страну Англия
Введите количество оценок 3
Введите оценки 1 2 3
Введите фамилию Петров
Введите страну Россия
Введите количество оценок 1
Введите оценки 3
Введите фамилию Ньютон
Введите страну Англия
Введите количество оценок 4
Введите оценки 1 2 3 4

Упорядоченные структуры:

фамилия Петров
страна Россия
количество оценок 1
оценки 3.00
средняя оценка 3.00
------------------
фамилия Иванов
страна Россия
количество оценок 2
оценки 2.00 3.00
средняя оценка 2.50
------------------
фамилия Ньютон
страна Англия
количество оценок 4
оценки 1.00 2.00 3.00 4.00
средняя оценка 2.50
------------------
```

```
фамилия Кук
страна Англия
количество оценок 3
оценки 1.00 2.00 3.00
средняя оценка 2.00
-----------------

страна - Россия, сумма средних оценок -5.50
страна - Англия, сумма средних оценок -4.50

Страна победителя - Россия
Средняя оценка победителя 5.50
*/
```

3.3. Задания на обработку структур

В каждом варианте заданий работа ведется с массивом структур.

1. Поля структур: фамилия жителя, город проживания, название улицы, номер дома и квартиры. Найти в массиве структур всех жителей, живущих в разных городах, но имеющих одинаковый адрес, и сформировать массив фамилий этих жителей. Вывести исходный и полученный массивы. Вывести для жителей, имеющих одинаковый адрес, следующую информацию: адрес, а под ним — фамилию и город проживания.

2. Поля структур: фамилия абитуриента, массив баллов, набранных на каждом из трех экзаменов, изучаемый иностранный язык. Для задаваемого проходного балла составить список абитуриентов, выдержавших конкурс. Полученный список разбить на списки в соответствии с изучаемым иностранным языком. Определить количество различных иностранных языков, изучаемых абитуриентами. Все списки отсортировать в алфавитном порядке фамилий.

3. Поля структур: фамилия конькобежца, результаты, показанные на каждой из четырех дистанций (500, 1500, 5000, 10 000 м), т. е. минуты и секунды, результат, набранный в многоборье. Для каждого спортсмена определить результат, показанный в многоборье, который вычисляется как сумма количества секунд, затраченных в среднем на преодоление 500 м на каждой из дистанций. Упорядочить список конькобежцев по возрастанию результатов, набранных в многоборье.

4. Поля структур: название печатного издания, тип издания (газета, журнал), периодичность выхода, цена. Сформировать список изданий в таком порядке: газеты (по алфавиту), затем — журналы (по алфавиту). Найти самую дешевую и дорогую ежедневные газеты, самый дешевый еженедельник, самый дешевый и дорогой журналы.

5. Поля структур: фамилия кандидата для занятий в спортивной секции, рост, год рождения, размер костюма. Сформировать список зачисленных в секцию. В секцию зачисляются кандидаты, имеющие рост не меньше *A* см и возраст в пределах от *C* до *D* лет. Определить для зачисленных в секцию количество костюмов каждого размера.

6. Поля структур: название города, средняя дневная температура, средняя ночная температура, направление ветра, влажность. Отсортировать массив структур в алфавитном порядке названий городов. Сформировать три массива. В первый массив занести структуры для городов, в которых и ночная, и дневная температуры положительны; во второй массив — структуры для городов, в которых ночная температура отрицательная, а дневная — положительная; в третий массив — структуры для городов, в которых и ночная, и дневная температуры отрицательны.

7. Поля структур: фамилия хоккеиста, номер, амплуа (вратарь, защитник, нападающий), год рождения, рост, вес. Сформировать список так, чтобы сначала располагались сведения о вратарях, затем о защитниках, а в конце — о нападающих. В пределах каждой из трех групп сведения должны располагаться в порядке возрастания номеров спортсменов. Определить самого молодого хоккеиста и самого возрастного хоккеиста, самого тяжелого и самого высокого.

8. Поля структур: название станции, номер зоны, стоимость взрослого билета, стоимость детского билета. Расположить структуры в порядке возрастания номеров зон. Найти и выдать информацию об интересующей станции, а также информацию о самом дорогом и самом дешевом билетах.

9. Поля структур: название дисциплины, количество еженедельных часов лекций, семинаров, лабораторных работ (по каждому виду занятий отдельно), количество рейтингов, зачетов и экзаменов по дисциплине (отдельно по каждой дисциплине). Подсчитать количество учебных занятий в неделю, количество рейтингов,

экзаменов и зачетов, сдаваемых в семестре. Выдать информацию о дисциплине с наибольшим и наименьшим количеством еженедельных занятий.

10. Поля структур: фамилия, пол, размер костюма, размер обуви, размер головного убора. Исходный массив структур разбить на два: с информацией о мужчинах и женщинах, каждый из массивов отсортировать по алфавиту фамилий. Подсчитать количество требуемой мужской обуви размеров 41, 42, 43 (по отдельности), а также размеров 35, 36 для женщин.

11. Поля структур: фамилия пассажира, вес багажа, количество вещей багажа, стоимость доплаты. Сформировать массив со сведениями о пассажирах, вес груза которых превышает разрешенный для бесплатного провоза (A кг), и массив со сведениями о пассажирах, имеющих более двух вещей. Оба массива отсортировать в алфавитном порядке фамилий. При вводе исходных данных вычислить значение поля «доплата» для сверхнормативного груза (в рублях за 1 кг сверхнормативного веса багажа).

12. Поля структур: фамилия автора книги, название книги, год издания, издательство. Сформировать массив структур с информацией о книгах по языку Паскаль. В массиве разместить сначала книги, изданные в определенном издательстве (ввести с клавиатуры), а затем все остальные. Определить количество книг по заданной теме, выпущенных в свет до заданного года.

13. Поля структур: фамилия автора книги, название книги, номер шкафа, где хранится книга, номер полки, номер ряда на полке. Сформировать массив структур с информацией о книгах, хранящихся в заданном шкафу, и упорядочить структуры по возрастанию номера полки. Выдать сведения о книгах, фамилия авторов которых начинается с буквы с «А» по букву «К».

14. Поля структур: название учебной дисциплины, название контрольного мероприятия (домашнее задание, аттестация, рубежный контроль, контрольная работа), объем работы в часах для мероприятия, месяц и день сдачи контрольного мероприятия. Упорядочить массив по возрастанию даты сдачи контрольного мероприятия. Подсчитать общий объем работ по каждой дисциплине и суммарный объем.

15. Поля структур: фамилия пациента, год рождения, год и месяц последнего обследования у врача. Составить массив структур

с информацией о тех пациентах, которые должны проходить обследование в текущем году. Известно, что пациенты в возрасте до 25 лет проходят обследование один раз в три года, в возрасте от 25 до 50 лет — раз в два года, старше 50 лет — каждый год. Полученный массив упорядочить по возрастанию даты обследования.

16. Поля структур: фамилия участника конференции, дни участия, т. е. строка, в которой через пробел записаны номера дней (например, 2 3 4 5), тип гостиничного номера (люкс, одноместный, двухместный), затраты на проживание, затраты на питание, суммарные затраты. Зная стоимость однодневного проживания в номере каждого типа, а также ежедневные затраты на питание, вычислить при вводе данных значения последних трех полей. Отсортировать массив в порядке возрастания суммарных затрат.

17. Поля структур: название дисциплины, номер семестра, в котором преподается дисциплина, форма проверки знаний (экзамен или зачет). Упорядочить массив в порядке возрастания номеров семестров. Для каждого семестра дисциплины упорядочить по алфавиту. Подсчитать по каждому семестру количество сдаваемых зачетов и экзаменов.

18. Поля структур: фамилия футболиста, амплуа игрока (вратарь, защитник, полузащитник, нападающий), год рождения, количество забитых голов. Упорядочить массив по возрастанию количества забитых голов. Определить самых результативных игроков: защитника, полузащитника, нападающего, а также самых результативных игроков в возрасте до 20 лет и в возрасте свыше 30 лет.

19. Поля структур: фамилия абитуриента, массив из трех экзаменов (оценки могут быть 2; 3; 3,5; 4; 4,5; 5). Ввести количество свободных мест и определить список абитуриентов, подлежащих зачислению на обучение, а также проходной балл. Среди лиц с полупроходным баллом отобрать тех, у кого больше пятерок, при равенстве этого показателя — сравнить по первой оценке.

20. Поля структур: тип геометрической фигуры (круг, прямоугольник, треугольник), массив характеристик (радиус, две стороны, сторона и высота), площадь, строка, содержащая формулу определения площади. В зависимости от типа фигуры вычислить для каждого элемента массива структур значение поля «площадь». Отсортировать массив по возрастанию площадей.

21. Поля структур: значение операнда `x`, значение операнда `y`, количество выполняемых операций (не более шести), массив символов (не более шести), указывающих тип операции, массив результатов выполнения операций (не более шести), среднее значение элементов массива. Вычислить для каждой структуры массив результатов выполнения каждой заданной операции и среднее для вычисленных значений. Упорядочить массив по возрастанию средних значений. Операции обозначаются символами «+», «-», «*», «/», «^» (возведение в степень), «l» — логарифмирование.

22. Поля структур: фамилия спортсмена, название страны, количество оценок спортсмена, массив значений оценок (не более пяти), средняя оценка. Вычислить при вводе данных значение поля «среднее значение». Упорядочить массив по убыванию средней оценки. Определить страну, чьи спортсмены имеют наибольшую сумму средних оценок.

23. Поля структур: фамилия автора работы, тип работы (домашнее задание, курсовой проект, дипломный проект), год написания, курс, на котором написана работа. Предполагая, что обучение продолжается шесть лет, домашнее задание хранится в течение одного года после окончания обучения, курсовой проект — два года, а диплом — три года, сформировать три списка по трем типам работ, которые должны быть утилизированы в текущем году. Каждый из списков составить в алфавитном порядке фамилий авторов.

24. Поля структур: фамилия студента, пол, название группы, три оценки. Расположить структуры в массиве так, чтобы информация о студентах одной группы располагалась подряд. Информация о студентах в пределах каждой группы должна быть упорядочена по алфавиту. Определить всех девушек-отличниц.

25. Поля структур: фамилия абонента, номер телефона, пол, год установки. Сформировать массив структур с информацией об абонентах того же пола, что и Вы, не старше Вас и установивших телефон в заданном диапазоне лет. Полученный массив упорядочить в алфавитном порядке фамилий.

26. Поля структур: фамилия студента, массив баллов, набранных за каждый из четырех модулей, массив максимальных баллов, начисляемых за каждый модуль, суммарный набранный балл, отметка о зачете. Вычислить для каждого студента общее количество набранных баллов и сформировать признак: «сдано», «не сдано».

Для получения зачета надо за каждый модуль набрать не менее 60 % баллов от максимально возможного.

27. Поля структур: марка автомобиля, государственный номер регистрации, массив расстояний, пройденных в течение пяти лет, суммарное пройденное расстояние. Вычислить при вводе данных значение поля суммарного расстояния, упорядочить список автомобилей по убыванию суммарного пробега и сформировать список автомобилей, у которых средний пробег больше среднего по всем автомобилям.

28. Поля структур: фамилия участника соревнований по современному пятиборью, год рождения, массив результатов по каждому виду соревнований, итоговый результат. Вычислить при вводе данных итоговый результат для каждого участника соревнований. Упорядочить список участников соревнований по убыванию сумм результатов, определить победителей в каждом виде, найти самого успешного спортсмена среди молодых (до 25 лет).

29. Поля структур: название города, средняя годовая температура, среднее годовое количество осадков, количество солнечных дней. Упорядочить список городов по убыванию средней годовой температуры, найти город, имеющий наименьшее среднее годовое количество осадков при количестве солнечных дней, которое не больше заданного.

30. Поля структур: фамилия хоккеиста, число забитых шайб, число голевых передач, штрафное время, количество игр, результативность. Вычислить при вводе данных значение поля «результативность», для этого количество голов умножить на 2 и прибавить количество результативных передач. Упорядочить список игроков по убыванию результативности, найти самого полезного игрока. Самым полезным считать игрока, который набрал в среднем за игру не более А минут штрафного времени и при этом добился максимальной средней за игру результативности.

3.4. Объединения

Объединение представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу. Такое наложение полей друг на друга необходимо в тех случаях, когда в разные моменты времени данные, расположенные по этому адресу, должны обрабатываться по-разному. Для объявления объ-

единений используется слово union (вместо struct). Длина объединения равна наибольшей длине его полей. В каждый момент времени переменная хранит только одно значение, а ответственность за его правильное использование несет программист. Объединения позволяют экономить память в тех случаях, когда известно, что больше одного поля одновременно использоваться не будет.

Например, студенты по определенной дисциплине сдают в семестре либо зачет, либо экзамен, поэтому можно не отводить сразу два поля для хранения информации о сданном зачете/экзамене. В этом случае определение объединения имеет следующий вид:

```
union prow
{
   char sach;
   int ekz;
} x;
```

Если студент сдает зачет, то поле **sach** будет хранить, например, символ **'s'** (сдано) или **'n'** (не сдано), если экзамен — поле **ekz** будет хранить полученную студентом оценку (2, 3, 4 или 5). Поскольку одновременно по дисциплине сдается или экзамен, или зачет, то в переменной **x** хранится только одно значение (или символ, или целое число), поэтому для его хранения можно отвести одну и ту же область памяти. Эта область памяти должна быть достаточной для сохранения самого длинного из возможных данных. Переменной **x** можно присвоить значение любого из двух типов, а затем использовать это значение в выражениях, но строго по правилам работы с конкретным типом данных. Переменная будет хранить значение того типа, который был присвоен при последнем обращении к переменной.

Программист должен следить в своей программе за тем, какое значение было присвоено переменной. Если ей было присвоено значение одного типа, а обработка ведется по правилам, присущим другому типу, то результат будет системно зависимым и труднопредсказуемым.

Объединения могут использоваться в структурах и массивах и наоборот. Способ обращения к члену объединения и структуре (или к члену структуры в объединении) полностью совпадает с обращением к элементу вложенной структуры.

3.5. Примеры программ обработки объединений

Пример 1. Создать массив структур с информацией о фамилии студента, типе контроля знаний (зачет или экзамен) и сданном зачете (экзамене). Упорядочить массив в алфавитном порядке фамилий.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include <string.h>
#include "D:\\temp\\RusString.h"

typedef union{
char sach;
int ekz;} prow;

typedef  struct
{ char f[30];
  int tip;
  prow oc;
}sap;

int _tmain(int argc, _TCHAR* argv[])
{
   sap a[10],b;
   int n,k,kol,i,flag;
printRus("Введите количество элементов массива: ");
   scanf("%d",&n);
   printf("\n");
   OutPuts("Введите элементы массива");
   for(i=0;i<n;i++)
   {
      printRus("\nВведите фамилию: ");
      scanf("%s",&a[i].f);
printRus("Введите тип 0 - зачет, 1 - экзамен: ");
      scanf("%d",&a[i].tip);
      printRus("Введите оценку: ");
      switch (a[i].tip) {
        case 0: {a[i].oc.sach=getche(); break;}
        case 1: {scanf("%d",&a[i].oc.ekz); break;};
      }
   }
   OutPuts("\nКонтрольный вывод записей");
```

```
   for(i=0;i<n;i++)
   {
      printf("%s  \n",a[i].f);
      printf("%d  ",a[i].tip);
      switch (a[i].tip) {
         case 0: {printf("  %c  \n" ,
            a[i].oc.sach); break;}
         case 1: {printf("  %d  \n",
            a[i].oc.ekz); break;};
      }
   }
   k=0;
   do
   {
      flag=0;
      k++;
      for (i=0;i<n-k;i++)
      if (strcmp(a[i].f,a[i+1].f)>0)
      {
         b=a[i];
         a[i]=a[i+1];
         a[i+1]=b;
         flag=1;
      }
}while (flag);
   OutPuts("\nВывод упорядоченных записей");
   for(i=0;i<n;i++)
   {
      printf("%s  ",a[i].f);
      printf("%d  ",a[i].tip);
      switch (a[i].tip) {
         case 0: {printf("  %c  \n",
            a[i].oc.sach); break;}
         case 1: {printf("  %d  \n",
            a[i].oc.ekz); break;};
      }
   }
 getch();
 return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите количество элементов массива: 2

Введите элементы массива
```

```
Введите фамилию: fff
Введите тип 0 - зачет, 1 - экзамен: 0
Введите оценку: s
Введите фамилию: aaa
Введите тип 0 - зачет, 1 - экзамен: 1
Введите оценку: 4

Контрольный вывод записей
fff
0 s
aaa
1 4

Вывод упорядоченных записей
aaa 1 4
fff 0 s
```

Пример 2. Использовать объединение для задания параметров трех разных геометрических фигур: круга, прямоугольника и треугольника. Создать массив структур с информацией о типе геометрической фигуры (круг, прямоугольник, треугольник), ее параметрах (радиус, основание и высота, три стороны). Вычислить значение поля «площадь» для структур массива.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#define _USE_MATH_DEFINES M_PI //M_PI - значение числа пи
#include "math.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   union fig
   {
      float r;
      float pr[2];
      float tr[3];
   };
   struct geom
   {
     char tip;//допустимые значения 'k', 'p', 't'
     fig a;
```

```
      float pl;
    };
    geom f[3];
    int i,n;
    printRus("Введите количество элементов массива: ");
    scanf("%d",&n);
    OutPuts("Введите элементы массива");
    for (i=0;i<n;i++)
    {
printRus("Введите тип  'k' или 'p' или 't': ");
      // scanf("%c",&f[i].tip);
       f[i].tip=getche();
       switch (f[i].tip)
       {
          case 'k':
            printRus("\nВведите радиус: ");
            scanf("%f",&f[i].a.r);
            break;
          case  'p':
             printRus("\nВведите длины сторон
                       прямоугольника: ");
             for (int j=0;j<2;j++)
                scanf("%f",&f[i].a.pr[j]);
             break;
          case  't':
             printRus("\nВведите длины сторон
                       треугольника: ");
             for (int j=0;j<3;j++)
             scanf("%f",&f[i].a.tr[j]);
             break;
       }
    }

 for (i=0;i<n;i++)
       switch (f[i].tip)
       {
          case 'k': f[i].pl=M_PI*f[i].a.r*f[i].a.r;
             break;
          case 'p': f[i].pl=f[i].a.pr[0]*f[i].a.pr[1];
             break;
          case 't':
          float p=(f[i].a.tr[0]+f[i].a.tr[1]+f[i].a.tr[2])/2.0;
             f[i].pl=sqrt(p*(p-f[i].a.tr[0])*(p-f[i].a.tr[1])
             *(p-f[i].a.tr[2]));
```

```
            break;
      }
   OutPuts("\nВычисленные площади фигур");
   printRus("Площадь Тип \n");
   for (i=0;i<n;i++)
      printf("%6.2f\t%c\n",f[i].pl,f[i].tip);
   getch();
   return 0;
}
```

Протокол ввода/вывода имеет следующий вид:

```
Введите количество элементов массива: 3
Введите элементы массива
Введите тип 'k' или 'p' или 't': k
Введите радиус: 1
Введите тип 'k' или 'p' или 't': p
Введите длины сторон прямоугольника: 1 1
Введите тип 'k' или 'p' или 't': t
Введите длины сторон треугольника: 1 1 1

Вычисленные площади фигур
Площадь Тип
 3.14 k
 1.00 p
 0.43 t
```

Вопросы для самопроверки

1. Что представляет собой тип данных «структура»?
2. Как выполняется объявление структуры?
3. Как выполняются операции обработки структур?
4. Как осуществляется обращение к полям структур: а) при работе с переменной-структурой; б) в случае использования указателя на структуру?
5. Можно ли имя поля структуры использовать в той же программе в качестве имени другой переменной, имени поля другой структуры?
6. Что такое объединение? Для чего предназначены объединения и в каких случаях целесообразно использовать объединения?

4. Файлы

4.1. Основные сведения о файлах

Файл обычно определяют как поименованный набор данных, хранимых на внешнем носителе. Язык С рассматривает любой файл как последовательный набор байтов, в котором текущий байт при вводе/выводе отмечен положением файлового курсора. В начале работы с файлом этот курсор указывает на самый первый, нулевой байт, а по мере последовательного ввода (или вывода) смещается к байту, номер которого больше текущего на количество прочитанных (или записанных) байтов. Для ускорения обмена данными с внешними запоминающими устройствами каждому открытому файлу (т. е. подготовленному к работе) выделяется в оперативной памяти буфер для ввода и/или вывода. При вводе данных буфер единовременно заполняется требуемой порцией байтов из файла (при последовательном доступе это позволит значительно реже обращаться к внешнему устройству, что особенно важно для дисковых устройств), а при выводе также уменьшается количество обращений к внешнему устройству, поскольку обычно вывод выполняется только после заполнения буфера. При открытии файла ему ставится в соответствие поток. В начале выполнения программы автоматически открываются три файла и связанные с ними потоки: стандартный ввод, стандартный вывод и стандартная ошибка. Потоки обеспечивают каналы передачи данных между файлом и программой. Стандартный поток ввода позволяет программе считывать данные с клавиатуры, а стандартные потоки вывода и ошибок — выводить данные на экран. Для начала работ с другими файлами необходимо явно выполнять открытие файла, а при завершении работ с файлом — его закрытие.

Функция, открывающая файл, возвращает указатель типа `FILE`, который определяется в заголовочном файле `<stdio.h>`. Этот ука-

затель следует присвоить переменной соответствующего типа, называемой файловой переменной, или указателем файла. Она будет представлять в дальнейшем файл во всех библиотечных функциях работы с ним, в том числе и в функции закрытия файла.

В языке С различают файлы двух типов: текстовые и бинарные.

Текстовые файлы представляют собой последовательность строк файла (т. е. последовательностей символов, заканчивающихся символом конца строки файла с кодом 10). Длины строк файла могут различаться, в том числе иметь и нулевую длину, т. е. состоять только из одного символа конца строки файла. При выводе числовых данных в текстовый файл осуществляется их преобразование из внутреннего представления в оперативной памяти к символьному, принятому для восприятия человеком, а при вводе числовых данных в файл выполняется обратное преобразование (из символьной формы во внутреннее представление). Текстовые файлы предназначены не только для хранения данных на внешних запоминающих устройствах, но и для просмотра в текстовых редакторах.

Бинарные файлы — это последовательность элементов определенного типа, а значит, определенного размера, причем компоненты такого файла хранятся в двоичном коде, т. е. выводятся из оперативной памяти в файл без преобразования к символьному виду. Поэтому для бинарных файлов ввод и вывод данных выполняется быстрее, чем для текстовых файлов.

Следует заметить, что работа с текстовыми файлами возможна только в режиме последовательного доступа. В этом случае для получения доступа к очередной строке файла или ее части надо считать всю предшествующую им информацию.

С бинарными файлами можно работать как в режиме последовательного, так и в режиме прямого доступа, т. е. получать доступ к требуемому элементу файла без просмотра всех предшествующих элементов.

4.2. Библиотечные функции для работы с файлами

Стандартная библиотека поддерживает функции открытия файла, чтения данных из файлов и записи данных в файлы, закрытия файла, а также ряд других функций. Кроме того, в стандартной библиотеке определены несколько макросов: `NULL`, `EOF`, `SEEK_SET`,

SEEK_CUR, SEEK_END. Макрос NULL определяет пустой (null) указатель. Макрос EOF предназначен для проверки достижения конца файла. Макросы SEEK_SET, SEEK_CUR, SEEK_END используются функцией fseek и будут рассмотрены ниже.

Открытие файла выполняется с помощью функции

```
FILE *fopen(const char *path, const char *mode)
```

Строковая константа или указатель на строку path представляет собой допустимое имя файла. Можно указывать полное имя, включающее путь к файлу, или только имя файла, в этом случае подразумевается, что файл находится в текущей папке, где хранятся файлы приложения.

Строковая константа или указатель на строку символов mode задает режим открытия файла. Ниже перечислены некоторые из них:

"r" — открытие текстового файла для чтения;

"w" — открытие (создание) текстового файла для записи;

"a" — открытие текстового файла для добавления в его конец;

"rb" — открытие бинарного файла для чтения;

"wb" — открытие бинарного файла для записи;

"ab" — открытие бинарного файла для добавления в его конец;

"rb+" — открытие бинарного файла для чтения/записи;

"wb+" — открытие (создание) бинарного файла для записи/чтения;

"ab+" — открытие бинарного файла или создание, если его нет, для добавления в его конец или для чтения/записи.

Функция fopen возвращает указатель, который следует присвоить файловой переменной. Открытие файла считается успешным, если возвращаемое функцией значение не равно NULL.

Приведенный ниже фрагмент программы представляет порядок открытия файла на примере создания текстового файла a.txt для записи информации в файл:

```
FILE *f;
. . .
if ((f=fopen("a.txt","w")==NULL)
{/*Ошибка при открытии файла. Далее расположить операторы обработки этого события*/
    . . .
}
```

```
    else
    {/*Успешное открытие файла. Далее расположить операторы для
работы с файлом */
        . . .
    }
```

Для правильного использования режимов работы с файлом надо принимать во внимание следующее.

Если попытаться открыть несуществующий файл для чтения, то обращение к функции **fopen** завершится ошибкой.

При открытии существующего файла для добавления новые строки будут добавлены в его конец, иначе будет создан и открыт для добавления новый файл.

При открытии файла для записи в случае его отсутствия файл создается. Если открывается для записи существующий файл, то сначала его содержимое будет удалено.

Закрытие файла выполняется функцией **int fclose(FILE *f)**. Возврат нуля означает успешное закрытие файла, а значение –1 — ошибку. Ошибка закрытия происходит при преждевременном удалении съемного накопителя информации или при отсутствии свободного места на внешнем запоминающем устройстве (ВЗУ). Функция **fclose** записывает все данные, которые еще оставались в буфере вывода, и проводит закрытие файла (для дальнейшей работы с ним его придется открывать заново). Однако иногда требуется вывести данные из буфера на ВЗУ, не закрывая файл. Такую работу для файла, представленного указателем **f**, выполняет функция

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

возвращающая в случае успеха значение 0, иначе — значение **EOF**. При обращении к функции с пустым (т. е. **NULL**) указателем файла заданная работа будет выполнена для всех файлов, открытых для вывода.

В процессе работы функций ввода данных могут возникать ошибки по различным причинам, одной из которых является достижение конца файла. Для определения конца файла можно использовать функцию

```
int feof(FILE*f)
```

где **f** — указатель файла.

Функция возвращает ненулевое значение, если достигнут конец файла, иначе — значение нуль. Эта функция используется чаще всего при организации циклов, которые должны завершаться при достижении конца файла.

Рассмотрим функции, выполняющие ввод и вывод данных различных типов.

Запись символа в файл осуществляется функцией

```
int fputc(int ch, FILE *f)
```

где `ch` — выводимый символ; `f` — указатель файла.

При успешной записи возвращается код записанного символа, а в случае ошибки — значение **EOF**. Напомним, что между символьными и целыми переменными автоматически выполняется приведение типа, т. е. если символьной переменной присвоить код символа, то она будет иметь значение этого символа. Так, если было объявлено

```
char ch; int i;
```

то при выполнении оператора

```
i=ch=fputc('A',f);
```

переменная `ch` получит значение `'A'`, а переменная `i` — значение 65 (0x41 в шестнадцатеричной форме).

Чтение символа из файла осуществляется функцией

```
int fgetc(FILE *f)
```

где `f` — указатель файла.

В случае успеха возвращается считанный символ, в случае ошибки или достижения конца файла — значение **EOF**.

Запись строки в файл выполняется функцией

```
int fputs(const char *st, FILE *f)
```

где `st` — строковая константа или указатель на записываемую строку; `f` — указатель файла.

В случае ошибки функция возвращает значение **EOF**, иначе — значение 0. Данная функция не записывает в текстовый файл символ с кодом 0, обозначающий конец выводимой строки, но интерпретирует управляющие символы, например, при выполнении оператора

```
fputs("x\ty\nX\tY", f);
```

В файле будет две строки, в которых символы будут разделены знаками табуляции:

```
x  y
X  Y
```

Чтение строки файла реализует функция

```
char *fgets(const char *st, int len, FILE *f)
```

где **st** — указатель на символьный массив, в который помещается вводимая строка; **len** — число, ограничивающее количество считываемых символов; **f** — указатель файла.

Функция **fgets** считывает из файла строку и делает это до тех пор, пока не будет прочитан символ конца строки файла, количество прочитанных символов не станет равным **len** – 1 или не будет достигнут конец файла. Если был прочитан символ с кодом 0, то он записывается в массив. При считывании строка дополняется символом конца строки. В случае успешного завершения функция возвращает указатель на массив с введенной строкой, в случае ошибки или достижения конца файла — пустой указатель **NULL**. Чтобы выяснить причину возврата значения **NULL** следует использовать функцию **feof**.

Установку курсора файла на его начало выполняет функция

```
void rewind(FILE *f)
```

где **f** — указатель файла.

Стирание (удаление) файла реализует функция

```
int remove(const char *st)
```

где **st** — полное или сокращенное имя файла.

В случае успешного выполнения функция возвращает нулевое значение, в противном случае — ненулевое.

Переименование файла осуществляет функция

```
int rename(const char *oldfname, const char *newfname)
```

где **oldfname** — старое имя файла; **newfname** — новое имя файла.

Имя **newfname** не должно совпадать ни с одним из существующих в каталоге имен файлов. При успешном завершении функция возвращает нулевое значение, в случае ошибки — ненулевое.

Вывод в файл данных разных типов выполняется функцией

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

Эта функция выводит в файл, указанный переменной `f`, значения аргументов из списка, обозначенного здесь как многоточие, в соответствии со строкой формата `format`. Возвращаемое значение равно количеству реально выведенных символов, в случае возникновения ошибки возвращается отрицательное число.

Для чтения из файла данных разных типов предназначена функция

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

где функция `fscanf` считывает данные из файла, указанного переменной `f`, в соответствии с форматом, задаваемым строкой `format`; многоточие означает список адресов переменных, значения которых считываются из файла.

Функция возвращает количество аргументов, которым действительно присвоены значения. Если функция возвращает значение `EOF`, то это свидетельствует об ошибке, произошедшей до первого присваивания.

При работе с бинарными файлами для чтения и записи данных используются следующие функции:

- `size_t fread(void *buf, size_t kol, size_t n, FILE *f)` — для считывания данных из файла, указанного переменной f, в область памяти, адрес которой задает переменная `buf`. Длина каждого элемента данных (в байтах) задается значением `kol`, а количество считываемых элементов данных определяется параметром `n`. Функция возвращает количество считанных элементов. Если при чтении данных обнаружится конец файла или произойдет ошибка, то возвращаемое значение может быть меньше значения `n`;
- `size_t fwrite(const void *buf, size_t kol, size_t n, FILE *f)` — для записи данных в файл, указанный переменной `f`, из области памяти, адрес которой задает переменная `buf`. Длина каждого элемента данных (в байтах) задается значением `kol`, а количество записываемых элементов данных определяется параметром `n`. Функция возвращает количество записанных элементов. Если при выполнении функции отсутствует ошибка, возвращаемый результат будет равен значению `n`.

При работе с файлами в режиме прямого доступа необходимо устанавливать файловый указатель к началу нужного элемента файла. Для этого предназначена функция

```
int ftseek(FILE *f, longint kb, int ot)
```

При обращении к этой функции указатель текущей позиции файла, представленного переменной `f`, смещается на `kb` байт относительно точки отсчета, задаваемой третьим параметром `ot`. Этот параметр может принимать одно из трех значений: `SEEK_SET` (или 0) — в качестве точки отсчета задается начало файла; `SEEK_CUR` (или 1) — в качестве точки отсчета выбирается текущая позиция указателя; `SEEK_END` (или 2) — точкой отсчета является конец файла. При успешном завершении работы функция возвращает нулевое значение, в случае ошибки — ненулевое.

Для определения текущей позиции указателя файла следует использовать функцию `longint ftell(FILE *f)`. Она возвращает текущее значение указателя текущей позиции в файле, связанном с указателем файла `f`. При ошибочном завершении функции возвращается значение –1.

Прямой доступ используется при работе с бинарными файлами. Это связано с тем, что все элементы имеют одинаковую длину (в отличие от текстовых файлов, где строки могут иметь разные длины), и проще вычислить адрес первого байта нужного элемента на внешнем носителе информации. Однако надо иметь в виду, что созданный текстовый файл можно открыть и в бинарном режиме, а файл, созданный в бинарном режиме, — открыть как текстовый, в том числе, например, в блокноте. Использовать блокнот иногда удобно при отладке программ, создающих бинарные файлы, если их элементы содержат тексты, хотя числа не будут отображаться правильно.

4.3. Примеры программ обработки текстовых файлов

Пример 1. Создать первый текстовый файл из произвольных строк. Переписать строки первого файла во второй текстовый файл так, чтобы в нем не было повторяющихся строк.

Алгоритм создания второго файла состоит в следующем: необходимо считывать очередную строку из первого файла и проверять

ее со всеми строками второго файла. Если равная строка найдена во втором файле, то цикл проверки следует закончить и эту строку не записывать во второй файл. Если же строка первого файла не будет обнаружена во втором файле, то ее следует записать во второй файл.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include <string.h>
#include "d:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   FILE *f, *f1;
   char a[80],b[80],c[80]={""};
   int flag;
   f=fopen("f1.txt","w");
   printRus("\nВведите строку - ");
   gets(a);
//Ввод с клавиатуры строк и запись их в файл,
//пока не будет введена пустая строка
   while (strcmp(a,c)!=0)//strlen(a)>0
   {
      strcat(a,"\n");/////
      fputs(a,f);
      printRus("Введите строку - ");
      gets(a);
   }
   fclose(f);

//Вывод содержимого первого файла
   f=fopen("f1.txt","r");
   OutPuts("Содержимое файла:");
   while (fgets(a,80,f)>0)
      OutPuts(a,' ');
   rewind(f);
//Формирование второго файла
   f1=fopen("f2.txt","w+");
   while (fgets(a,80,f)!=NULL)
//цикл просмотра строк первого файла
   {
      flag=1; //признак наличия считанной
```

```
            //строки во втором файле - 0
        rewind(f1);
        while (fgets(b,80,f1)!= NULL && flag)
        {
//Цикл чтения строк из второго файла и сравнения
// считанной строки со строкой из первого файла
           if (strcmp(a,b)==0)
              flag=0;
        }
        if (flag)
//записать строку во второй файл
           fputs(a,f1);
    }
    fclose(f);fclose(f1);
    f1=fopen("f2.txt","r");
    printf("\n sodershimoe file2 \n");
    while (fgets(a,80,f1)!=NULL)
       OutPuts(a,' ');
    getch();
    return 0;
}
```

Пример 2. Создать файл с информацией о студентах. Каждая строка файла должна иметь следующую структуру (в скобках указаны длины полей): номер студента (4), пробел, группа (8), пробел, фамилия (12), оценка 1, оценка 2, оценка 3, средний балл (5). Отсортировать список студентов по фамилиям в порядке, обратном алфавитному (использовать массив), а также найти трех худших студентов по сумме баллов и записать информацию о них во второй файл. Сортировку выполнить при вводе данных из файла, т. е. поместить элемент в массив сразу на требуемое место. При вводе данных вычислить средний балл и занести его сразу в файл.

Для удобства обработки данных следует представить в программе информацию о каждом студенте в виде структуры. Для записи и чтения данных в файл целесообразно выполнить форматный ввод/вывод. Поскольку заранее объем файла не известен, то удобно использовать динамический массив для хранения данных, считанных из файла. При помещении в массив элемента, считанного из файла, поиск места в массиве осуществляется в цикле путем сравнения считанной фамилии с очередной фамилией, хранящейся в массиве структур. Если считанная фамилия больше теку-

щей, то необходимо вставить ее перед текущим элементом массива. Все элементы массива, начиная с текущего, смещаются на одну позицию в сторону конца массива.

Поиск трех худших студентов можно выполнить, используя тот же алгоритм сортировки, но сравнение структур следует вести по полю «средний балл» и располагать элементы по возрастанию среднего балла, а не по убыванию.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "stdlib.h"
#include "d:\\temp\\RusString.h"

typedef struct
{
   char
      nom[5],
      grup[9],
      fam[13];
   int
      oc[3];
   float sr;
}st;
void wwod(char *path,char *regim)
{//функция ввода данных в файл
   FILE *f;
   char nom[5],grup[9],fam[13];
   int oc[3],i;
   float sr;
   f=fopen(path,regim);
   printRus("\nВведите номер студента - ");
   scanf("%s",nom);
   while (strcmp(nom,"")!=0)
   {
      printRus("Введите номер группы - ");
      scanf("%s",grup);
      printRus("Введите фамилию - ");
      scanf("%s",fam);
      printRus("Введите три оценки - ");
      for (i=0;i<3;i++)
```

```
            scanf("%d",&oc[i]);
        fprintf(f,"%4s%8s%12s",nom,grup,fam);
        for (i=0;i<3;i++)
            fprintf(f,"%3d",oc[i]);
        sr=(float)(oc[0]+oc[1]+oc[2])/3;
        fprintf(f,"%5.2f",sr);
        fprintf(f,"\n");
        printRus("\nВведите номер студента - ");
        gets(nom); gets(nom);
    }
    fclose(f);
}
void wiwod(char *path,char *regim)
{//функция просмотра содержимого файла
    FILE *f;
    char nom[5],grup[9],fam[13],buf[80];
    int oc[3],i;
    f=fopen(path,regim);
    while(fgets(buf,79,f)!=NULL)
        printf("%s",buf);
    fclose(f);
}
void sortirovka(char *path)
{
//функция заполнения массива данными файла с одновременной
//сортировкой по фамилиям в порядке, обратном алфавитному
    FILE *f;
    st stud, *s;
    int i,k=0,j,l;
    bool flag;
    f=fopen(path,"r");
    s=(st *)malloc(0);
    printf ("\n ischodnij file\n");
    while(!feof(f))
    {
        fscanf(f,"%4s%8s%12s",stud.nom,stud.grup,stud.fam);
        for (i=0;i<3;i++)
        fscanf(f,"%d",&stud.oc[i]);
        fscanf(f,"%f",&stud.sr);
        fscanf(f,"\n");
        printf("%4s %8s %12s %3d%3d%3d %5.2f\n"
            ,stud.nom,stud.grup,stud.fam,stud.oc[0]
            ,stud.oc[1],stud.oc[2],stud.sr);
```

```
        flag=true;
        j=0;
        //цикл поиска места вставки в массив
        //структуры, считанной из файла
        while (flag&&j<k)
        {
           if (strcmp(s[j].fam,stud.fam)<0)
              flag=false;
           else
              j++;
        }
        k++;
        //выделение памяти
        s=(st *)realloc(s,k*sizeof(st));
        for (l=k-1;l>j;l--)//цикл сдвига
        //элементов массива перед вставкой и
           s[l]=s[l-1];
        s[j]=stud;   //вставка элемента на
              //освободившееся место
    }
    printf("\notsortirowannij massiw\n");
    for (i=0;i<k;i++)
    printf("%4s %8s %12s %3d%3d%3d%5.2f\n"
       ,s[i].nom,s[i].grup,s[i].fam,s[i].oc[0]
       ,s[i].oc[1],s[i].oc[2],s[i].sr);
    fclose(f);
    f=fopen(path,"w");
    for (i=0;i<k;i++)
       fprintf(f,"%4s%8s%12s%3d%3d%3d%5.2f\n"
          ,s[i].nom,s[i].grup,s[i].fam,s[i]
          .oc[0],s[i].oc[1],s[i].oc[2],s[i].sr);
    fclose(f);
}
void tri(char *path)
{
//функция поиска трех худших студентов
//по среднему баллу
    FILE *f;
    st stud, *s;
    int i,k=0,j,l;
    bool flag;
    char gr[9];
    f=fopen(path,"r");
```

```
    s=(st *)malloc(0);
    //цикл чтения содержимого файла
    while(fgets(stud.nom,5,f)!= NULL)
    {
        fscanf(f,"%s%s",stud.grup,stud.fam);
        for (i=0;i<3;i++)
            fscanf(f,"%d",&stud.oc[i]);
        fscanf(f,"%f\n",&stud.sr);
        flag=true;
        j=0;
        //Цикл поиска места в массиве для
        //вставки считанного элемента
        while (flag&&j<k)
            if (s[j].sr>stud.sr)
                flag=false;
            else
                j++;
        k++;
        s=(st *)realloc(s,k*sizeof(st));
        for (l=k-1;l>j;l--) //цикл переписи
                //элементов массива для
            s[l]=s[l-1]; //освобождения места
                //для нового элемента

        s[j]=stud; //запись в массив нового
                //элемента
    }
    //запись в файл с именем temp трех худших
    //студентов или всех, если их меньше трех
    f=fopen("temp","w");
    if (k>3) k=3;
    for (i=0;i<k;i++)
    {
        fprintf(f,"%4s%8s%12s%3d%3d%3d%5.2f\n"
            ,s[i].nom,s[i].grup,s[i].fam,s[i].oc[0]
            ,s[i].oc[1],s[i].oc[2],s[i].sr);
    }
    fclose(f);
}

int _tmain(int argc, _TCHAR* argv[])
{
    char ch,path[20];
```

```
    do
    {
        printRus("Введите имя файла - ");
        scanf("%s",path);
        printf("\n%s",path);
        OutPuts("\n\n Меню");
        OutPuts("\n1. WWOD Ввод данных в файл");
        OutPuts("\n2. WIWOD Вывод данных из файла");
        OutPuts("\n3. sortirovka Сортировка данных ");
        OutPuts("\n4. tri Запись в файл temp трех худших");
        OutPuts("\n5. Выход");
        ch=getch();
        switch (ch)
        {
            case '1': wwod(path,"w"); break;
            case '2': wiwod(path,"r"); break;
            case '3': sortirovka(path);break;
            case '4': tri(path);break;
        }
    }while(ch!='5');
    return 0;
}
```

Пример 3. Отсортировать текстовый файл.

Сортировка текстового файла выполняется с использованием дополнительного файла. Сначала можно подсчитать общее количество строк исходного файла, затем в цикле искать минимальную строку исходного файла. Минимальная строка записывается в результирующий файл. Затем содержимое исходного файла, за исключением минимальной строки, переписывается в промежуточный файл. Исходный файл уничтожается. Промежуточный файл переименовывается, при этом он получает имя исходного файла. Данная процедура повторяется в цикле столько раз, сколько элементов содержит файл. В конце цикла исходный файл уничтожается, а результирующий файл получает имя исходного файла.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "stdlib.h"
```

```
#include "d:\\temp\\RusString.h"

void wwod(char *path,char *regim)
{
//функция записи произвольных строк в исходный файл
   FILE *f; char fam[81];
   f=fopen(path,regim);
   printf("\nwwedite stroku - ");
   gets(fam);
   while (strcmp(fam,"")!=0)
   {
      fputs(fam,f);
      fputs("\n",f);
      printf("\nwwedite stroku - ");
      gets(fam);
   }
   fclose(f);
}

void wiwod(char *path,char *regim)
{//функция просмотра содержимого файла
   FILE *f; char fam[81];
   f=fopen(path,regim);
   while(fgets(fam,80,f)!=NULL)
   {
      printf("%s",fam);
   } ;
   fclose(f);
}

void sortirovka(char *path)
{//функция сортировки
   FILE *f, *fd,*fr;
   int i,k,imin,kol,j;
   char fam[81],min[81],buf[81];
   f=fopen(path,"r");
   kol=0;
   fr=fopen("res","w");
   //цикл определения объема исходного файла
   //(количества строк)
   while(fgets(fam,80,f)!=NULL)
      kol++;
```

```
    k=kol;
    fclose(f);
    printRus("\nКоличество строк в файле = ");
    printf("%d",kol);
    for (i=1;i<=kol;i++) //цикл сортировки файла
    {
       f=fopen(path,"r");
       fgets(min,80,f);
       imin=1;
       for (j=2;j<=k;j++)//поиск минимальной
                 //строки в файле
       {
          fgets(fam,80,f);
          if (strcmp(fam,min)<0)
          {
             strcpy(min,fam);
                imin=j;
          }
       }
       fputs(min,fr);  //запись минимальной
                       //строки в результирующий файл
//перепись содержимого исходного файла в
//дополнительный файл (без минимальной строки)
       fd=fopen("dop","w");
       rewind(f);
       //перепись всех строк, расположенных до
       //минимальной строки
       for (j=1;j<imin;j++)
       {
          fgets(buf,80,f);
          fputs(buf,fd);
       }
       fgets(buf,80,f);//пропуск минимальной строки
//перепись всех строк, расположенных после минимальной строки
       for (j=imin+1;j<=k;j++)
       {
          fgets(buf,80,f);
          fputs(buf,fd);
       }
       fclose(f);
       fclose(fd);
       //уничтожение исходного файла
```

```
        remove(path);
        //переименование дополнительного файла
        rename("dop",path);
        k--;
    }
    fclose(fr);
    fclose(f);
    //уничтожение исходного файла
        OutPuts(
        "\nУничтожение исходного файла");
    //переименование полученного файла
        OutPuts(
        "Переименование полученного файла");
    rename("res",path);
}

int _tmain(int argc, _TCHAR* argv[])
{
   char ch,path[20],rech[20];
   do
   {
   OutPuts("\n Меню");
   OutPuts("0. Ввод имени файла");
   OutPuts("1. WWOD Ввод данных в файл");
   OutPuts("2. WIWOD Вывод данных из файла");
   OutPuts("3. sortirovka Сортировка данных ");
   OutPuts("4. Выход");
   ch=getch();
   switch (ch)
   {
   case '0': OutPuts("Введите имя файла");
//         printf("wwedite imja file - \n");
          gets(path);
                          OutPuts("КОНТРОЛЬНЫЙ ВЫВОД");
                          printf("\n%s",path);break;
   case '1': wwod(path,"w"); break;
   case '2': wiwod(path,"r"); break;
   case '3': sortirovka(path);break;
       }
   }while(ch!='4');
   return 0;
}
```

4.4. Задания на обработку текстовых файлов

1. Сформировать текстовый файл с информацией о книгах. Для каждой книги указать автора, название, год выпуска. Сформировать второй текстовый файл и занести в него информацию о книгах по языку Паскаль (в названии должно присутствовать слово Паскаль), выпущенных после указанного года.

2. Сформировать текстовый файл с информацией о группе спортсменов. Для каждого спортсмена указать фамилию и результат в беге на дистанцию 5000 м (минуты, секунды, десятые и сотые доли секунды). Сформировать второй файл и занести в него информацию о шести лучших спортсменах.

3. Сформировать текстовый файл и занести в каждую строку файла информацию о названии газеты или журнала, признак (газета или журнал) и цену издания. Сформировать новый текстовый файл с информацией о самой дорогой и самой дешевой газете и о самом дорогом и самом дешевом журнале.

4. Сформировать текстовый файл с информацией о названиях городов и численности их жителей. Сформировать второй файл с названиями пяти самых крупных городов.

5. Сформировать текстовый файл с названиями городов и информацией о средней температуре за год в этих городах. Сформировать два новых файла: в один поместить названия городов, где температура превышает среднее значение по всем городам, в другой — названия городов, где температура ниже средней.

6. Сформировать текстовый файл из произвольных слов. Определить для этих слов среднее количество букв в каждом слове и сформировать второй файл из слов, длина которых превышает среднюю, с указанием длины каждого слова.

7. Сформировать текстовый файл из произвольных слов. Определить по каждому слову отношение количества гласных букв к общему количеству букв (символов) слова. Сформировать новый файл и занести в него слова, для которых доля гласных составляет более 40 %, с указанием доли гласных.

8. Сформировать текстовый файл из произвольных слов. Затем создать новый файл и занести в него слова, содержащие в своем составе цифровые символы, с указанием количества таких символов в слове.

9. Сформировать произвольный текстовый файл. Затем создать второй файл и занести в него слова из первого файла, которые в нем встретились более одного раза.

10. Сформировать произвольный текстовый файл. Затем создать второй файл и занести в него слова из первого файла, которые в нем встретились не более одного раза.

11. Сформировать произвольный текстовый файл. Затем создать второй файл и занести в него слова из первого файла, в которых встречаются символы, не являющиеся буквами и цифрами. Для каждого слова указать (занести во второй файл) количество таких символов.

12. Сформировать произвольный текстовый файл. Затем создать второй файл и занести в него слова из первого файла, которые целиком состоят из небуквенных символов. Для каждого слова указать (занести во второй файл) его длину.

13. Сформировать произвольный текстовый файл. Затем создать второй файл и занести в него слова из первого файла, которые являются симметричными. Для каждого слова указать (занести во второй файл) его длину. Во втором файле найти самое длинное и самое короткое слова.

14. Сформировать текстовый файл с информацией о фамилии абитуриента, двух оценках (например, по математике и физике) и признаке (зачет/незачет по русскому языку). Для вводимого проходного балла сформировать список абитуриентов, зачисленных на обучение, и записать его во второй файл с указанием суммарного балла (оценки могут быть 2; 3; 3,5; 4; 4,5; 5).

15. Сформировать текстовый файл с информацией о названиях городов, средней ночной и средней дневной температурах в этих городах для определенного месяца. Сформировать второй файл и записать в него названия городов и сведения о среднесуточной температуре для пяти самых холодных городов.

16. Сформировать текстовый файл с информацией о фамилиях фигуристов и их оценках, полученных у пяти судей. Сформировать новый файл с информацией о первых трех лучших фигуристах (записать фамилию и суммарный балл).

17. Сформировать текстовый файл, записав в него фамилии студентов, год их рождения, пол (символ). Сформировать второй файл из фамилий студентов, удовлетворяющих определенным требованиям (например, мужчин моложе 25 лет).

18. Сформировать текстовый файл, содержащий информацию о марках компьютеров, их тактовых частотах, объемах оперативной и внешней памяти. Создать второй файл, записав в него сведения о компьютерах, удовлетворяющих заданным требованиям (ввести с клавиатуры).

19. Сформировать текстовый файл с информацией об автомобилях (марка, максимальная скорость, количество пассажиров, расход бензина на 100 км). Сформировать второй файл и записать в него информацию о марках автомобилей, удовлетворяющих заданным требованиям (ввести с клавиатуры).

20. Сформировать текстовый файл с информацией о фамилиях студентов и их оценках (всего четыре оценки). Сформировать второй файл и занести в него информацию о студентах, получающих стипендию, т. е. не имеющих троек и двоек.

21. Сформировать текстовый файл с информацией об индексе учебной группы, количестве студентов в ней и количестве студентов, успешно сдавших сессию. Создать второй файл и записать в него индекс группы и процент успеваемости в ней для групп, в которых успеваемость выше заданного значения.

22. Сформировать текстовый файл, в который записать произвольные слова. Создать второй файл и внести в него слова, доля согласных букв в которых не более доли гласных. Записать во второй файл также вычисленные доли гласных и согласных.

23. Сформировать текстовый файл и записать в него фамилии людей и номера их телефонов, а также признаки телефона (служебный или домашний). Сформировать второй файл и записать в него фамилии и телефоны лиц, у которых фамилия начинается на заданные буквы (множество этих букв задается с клавиатуры), причем сначала в файле должна присутствовать информация о домашних телефонах, а затем о служебных.

24. Сформировать текстовый файл и записать в него фамилии студентов и баллы, набранные студентами при выполнении четырех заданий. Для вводимых с клавиатуры сведений о максимальном количестве баллов, проставляемых за каждое задание, и необходимом проценте баллов от максимального балла для каждого задания, требуемых для допуска к экзамену, сформировать второй файл с информацией о фамилии студента, суммарном набранном балле и допуске к экзамену.

25. Сформировать текстовый файл и записать в него фамилии людей и номера их телефонов. Сформировать второй файл и занести в него фамилии и номера телефонов тех лиц, для записи номеров телефонов которых потребовалось не более заданного количества различных цифр.

26. Сформировать текстовый файл, содержащий произвольные слова. Создать второй файл и записать в него все слова, в которых количество различных символов составляет не менее половины от общего количества символов слова. Помимо самих слов во второй файл записать долю различных символов от общего количества символов слова.

27. Сформировать текстовый файл, содержащий фамилии студентов, количество экзаменов, которое сдавал каждый студент, и оценки, полученные на экзаменах. Создать второй файл и записать в него фамилии студентов, закончивших сессию без двоек, средний балл которых превышает средний балл по всей группе. При подсчете среднего балла двойки во внимание не принимать.

28. Сформировать текстовый файл, содержащий произвольные действительные числа. Создать второй файл и записать в него все числа, которые расположены между максимальным и минимальным или между минимальным и максимальным числами исходного файла.

29. Сформировать текстовый файл, содержащий произвольные слова. Найти в файле слова с наибольшей и наименьшей долей гласных букв. Создать второй файл и записать в него все слова, расположенные между найденными словами.

30. Сформировать текстовый файл, содержащий произвольные слова. Подсчитать долю различных символов от общего количества символов слова. Создать второй файл и записать в него все слова, в которых доля различных символов меньше среднего значения этой величины для всех слов файла. Во втором файле для каждого слова указать долю различных символов, расположенных между найденными словами.

4.5. Примеры программ обработки бинарных файлов

Пример 1. Создать бинарный файл, записать в него произвольные действительные числа, а затем на место максимального элемента — сумму положительных элементов, а на место минимального элемента — сумму отрицательных элементов.

Ввод элементов осуществляется по запросу, выводимому программой. Если пользователь хочет прекратить ввод, он должен ввести символ N. Для продолжения ввода следует ввести любой другой символ. При поиске минимального и максимального элементов запоминается адрес этих элементов, т. е. их смещение относительно начала файла. При записи найденных сумм на места этих элементов предварительно файловый указатель устанавливается по найденному адресу.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "stdlib.h"
#include "d:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
float ma[7]={-2,5,8,-3,-1,9,0};
   float a,max,min,sp,so;
   FILE *f;
   int k,imax,imin,nn;
   char c;
   f=fopen("f1.bin","wb");
   nn=sizeof(float);
   do
   {
//цикл ввода с клавиатуры и записи чисел в файл
      OutPuts("\nВведите число");
      scanf("%f",&a);
      fwrite(&a,nn,1,f);
      OutPuts("Продолжить ввод? (Y/N)");
      c=getch();
   }while(c!='N');
   fclose(f);
   f=fopen("f1.bin","rb+");
   OutPuts("\n Содержимое файла");
   //цикл просмотра содержимого исходного файла
   while (fread(&a,nn,1,f)>0)
   printf("\n a=%5.2f",a);
   rewind(f);
   so=sp=0;
   max=-1e10; min=1e10;
```

```
    //цикл поиска минимального, максимального
    //элементов, их адресов,вычисления сумм
    while (fread(&a,nn,1,f)>0)
    {
       if (a>0) sp+=a;
       if (a<0) so+=a;
       if (a>max)
       {
          max=a;
          imax=ftell(f)-nn;
       }
       if (a<min)
       {
          min=a;
          imin=ftell(f)-nn;
       }
    }
 fseek(f,imax,0);
 fwrite(&sp,nn,1,f);   //запись суммы положительных на
                       //место максимального
 fseek(f,imin,0);
 fwrite(&so,nn,1,f);   //запись суммы отрицательных на
                       //место минимального
 rewind(f);
    OutPuts("\nСодержимое файла");
              //цикл просмотра содержимого измененного файла
    while (fread(&a,nn,1,f)>0)
    printf("\n a=%5.2f",a);
    getch();
    fclose(f);
    return 0;
}
```

Пример 2. Отсортировать бинарный файл, содержащий структуры с информацией о студентах, в алфавитном порядке фамилий. Структура содержит следующие поля: фамилия, количество оценок (не более пяти), массив оценок.

Создание файла и запись структур в файл, вывод содержимого файла в окно программы и сортировку файла следует оформить в виде трех функций. Для окончания ввода в ответ на запрос программы необходимо ввести символ `N` (как и в предыдущем примере).

Сортировку файла выполняют с помощью алгоритма модифицированного пузырька. Для этого в цикле каждый раз считываются из файла два соседних элемента. Если фамилия первого студента больше фамилии второго студента, то считанные из файла структуры надо записать в файл в обратном порядке. Для этого курсор файла смещается от текущей позиции в сторону начала файла на количество байтов, равное объему двух считанных структур. После этого структуры записываются в обратном порядке. Переменная `flag`, фиксирующая факт обмена, получает при этом значение «истина», что свидетельствует о необходимости дальнейшей реализации внешнего цикла (файл еще не отсортирован).

При очередном выполнении внутреннего цикла должны быть прочитаны из файла две очередные структуры. Это обеспечивается путем перемещения файлового указателя на величину `i·n` байт от начала файла. Здесь `i` — номер первой структуры, считываемой на очередном шаге цикла (совпадает со значением параметра цикла), `n` — объем одной структуры в байтах.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "stdlib.h"
#include "io.h"
#include <string.h>
#include "d:\\temp\\RusString.h"

typedef int mas[5];
typedef  struct
{
   char fam[32];
   int n;
   mas oc;
}stud;

void sozdf(char path[]);
void wiwodf(char path[]);
void sortf(char path[]);

int _tmain(int argc, _TCHAR* argv[])
{
   char path[80];
   sozdf(path);
```

```
    wiwodf(path);
    sortf(path);
    OutPuts("\n Отсортированный файл");
    wiwodf(path);
    getch();
    return 0;
}
void sozdf(char path[])
{//Создание файла и запись структур данных в файл
    stud a;
    int i,n;
    FILE *f;
    char c;
    OutPuts("\n Введите имя файла");
    gets(path);
    f=fopen(path,"wb");
    do
    {
        OutPuts("\n Введите фамилию");
        scanf("%s",a.fam);
        OutPuts("\n Введите количество оценок");
        scanf("%d",&a.n);
        OutPuts("\n Введите оценки");
        for (i=0;i<a.n;i++)
            scanf("%d",&a.oc[i]);
        fwrite(&a,sizeof(a),1,f);
        OutPuts("\n Продолжить ввод? (Y/N)");
        c=getch();
    }while(c!='N');

  fclose(f);
}
void wiwodf(char path[])
{//Вывод из файла на экран
    stud a;
    int i,n;
    FILE *f;
    f=fopen(path,"rb");
    n=sizeof (a);
    while (fread(&a,n,1,f)>0)
    {
printRus("\n Фамилия: "); printf("%s",a.fam);
       printf("\n kol-wo ocenok -  %3d",a.n);
```

```
        printf("\n ocenki -  ");
        for (i=0;i<a.n;i++)
            printf("%3d ",a.oc[i]);
    }
    fclose(f);
}
void sortf(char path[])
{//Сортировка структур в файле
    stud a,b;
    int i,n,kk,k;
    FILE *f;
    bool flag;
    f=fopen(path,"rb+");
    n=sizeof (a);
    kk=filelength(fileno(f))/n;
    k=0;
    do
    {
        k++;
        flag=false;
        for (i=0;i<kk-k;i++)
        {
            fseek(f,i*n,SEEK_SET);
            fread(&a,n,1,f);
            fread(&b,n,1,f);
            if (strcmp(a.fam,b.fam)>0)
            {
                fseek(f,-2*n,SEEK_CUR);
                fwrite(&b,sizeof(a),1,f);
                fwrite(&a,sizeof(a),1,f);
                flag=true;
            }
        }
    }while(flag);
    fclose(f);
}
}
```

Пример 3. Переформировать файл, содержащий структуры с информацией о студентах, так, чтобы структуры с информацией о двоечниках находились бы в конце файла. Отсечь часть файла с информацией о двоечниках.

В целях исключения зацикливания программы и для удобства ее организации следует переформировать файл так, чтобы структуры с информацией об успевающих студентах перемещались бы в начало файла. Для этого необходимо обменивать местами в файле структуру с информацией об успевающем студенте со структурой из начала файла. Номер элемента файла, на место которого помещается структура с информацией об успевающем студенте, хранится в переменной `k`. Следует задать ее начальное значение равным –1. В случае обнаружения структуры с информацией об успевающем студенте (она должна храниться в переменной `a`) необходимо: запомнить смещение этой структуры от начала файла (в программе это значение хранит переменная `m`); номер структуры с информацией об успевающих студентах увеличить на единицу, подвести файловый указатель к началу структуры со смещением в `k·n` байт (`n` — объем одной структуры в байтах); прочитать хранящуюся там структуру (переменная `b`); на ее место записать ранее считанную структуру `a` с информацией об успевающем студенте. После этого возвращаются к элементу файла, имеющему смещение `m`, выводят структуру `b` и продолжают просмотр элементов файла.

Для определения объема файла и изменения его размеров (отсечения части файла) следует использовать функции из библиотеки `<io.h>`.

Получить размер файла в байтах можно с помощью функции

```
long int filelength( int fd)
```

Эта функция возвращает объем файла, выраженный в байтах. Параметр `fd` — дескриптор файла.

Для получения дескриптора файла следует обратиться к функции

```
int _fileno(FILE *f),
```

где `f` — указатель на файл.

Изменение объема файла осуществляет функция

```
int chsize( int fd, long int size),
```

где `fd` — дескриптор файла; `size` — новый объем файла, выраженный в байтах.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "stdlib.h"
#include "io.h"
#include <string.h>
#include "d:\\temp\\RusString.h"

typedef int mas[5];
typedef  struct
{
   char fam[32];
   int n;
   mas oc;
}stud;

void sozdf(char path[]);
void wiwodf(char path[]);
void peref(char path[]);

int _tmain(int argc, _TCHAR* argv[])
{
   char path[80];
   sozdf(path);
   wiwodf(path);
   peref(path);
   OutPuts("\n Отсечение файла");
   wiwodf(path);
   getch();
   return 0;
}
void sozdf(char path[])
{//Создание файла и запись структур данных в файл
   stud a;
   int i,n;
   FILE *f;
   char c;
   OutPuts("\n Введите имя файла");
   gets(path);
   f=fopen(path,"wb");
   do
   {
      OutPuts("\n Введите фамилию");
      scanf("%s",a.fam);
```

```
      OutPuts("\n Введите количество оценок");
      scanf("%d",&a.n);
      OutPuts("\n Введите оценки");
      for (i=0;i<a.n;i++)
         scanf("%d",&a.oc[i]);
      fwrite(&a,sizeof(a),1,f);
      OutPuts("\n Продолжить ввод? (Y/N)");
      c=getch();
   }while(c!='N');
  fclose(f);
}
void wiwodf(char path[])
{//Вывод из файла на экран
   stud a;
   int i,n;
   FILE *f;
   f=fopen(path,"rb");
   n=sizeof (a);
   while (fread(&a,n,1,f)>0)
   {
      printRus("\n Фамилия: "); printf("%s",a.fam);
      printRus("\n Количество оценок: "); printf("%d",a.n);
      printRus("\n Оценки: ");
      for (i=0;i<a.n;i++)
         printf("%3d ",a.oc[i]);
   }
   fclose(f);
}
void peref(char path[])
{
   stud a,b;
   int i,n,kk,k,j,m;
   FILE *f;
   bool flag;
   f=fopen(path,"rb+");
   n=sizeof (a);
   kk=filelength(fileno(f))/n;
   k=-1;
   for (i=0;i<kk;i++)
   {
      fseek(f,i*n,SEEK_SET);
      fread(&a,n,1,f);
      flag=false;
```

```
        j=0;
        while (!flag && j<a.n)
            if (a.oc[j]==2)
                flag=true;
            else
                j++;
        if (!flag)
        {
            m=ftell(f)-n;
            k++;
            fseek(f,k*n,SEEK_SET);
            fread(&b,n,1,f);
            fseek(f,k*n,SEEK_SET);
            fwrite(&a,n,1,f);
            fseek(f,m,SEEK_SET);
            fwrite(&b,n,1,f);
        }
    }
    fclose(f);
    f=fopen(path,"rb+");
    chsize( fileno(f), (k+1)*n);
    fclose(f);
}
```

Пример 4. Создать бинарный файл, содержащий структуры с информацией о студентах. Структура содержит следующие поля: фамилия студента, количество оценок (не более пяти), массив оценок, размер стипендии. Необходимо, используя клавиатуру, ввести исходные данные (фамилии, количество оценок, массив оценок) и записать их в файл. На основе этой информации провести расчет стипендии и скорректировать структуры файла. Размер базовой стипендии задается константой. Двоечники стипендию не получают. Троечники получают базовую стипендию. Хорошисты (нет троек и двоек, но не все оценки отличные) получают стипендию с коэффициентом 1,5, а отличники — удвоенную стипендию.

При решении этой задачи каждую операцию реализуют в отдельной подпрограмме-функции. Функция `init` дает возможность инициализировать файл, т. е. вводить имя файла и режим его работы, а также открывать его в соответствии с заданным режимом. Функция `wwod` позволяет вводить данные с клавиатуры и записывать их в файл. Функция `wiwod` обеспечивает просмотр содержи-

мого файла и вывод информации на экран. Функция **stipen** дает возможность рассчитать размер стипендии и скорректировать содержимое файла. При вводе данных поле стипендии не заполняется. Для удобства работы с программой предусматривается меню, для чего используется функция **menu**, обеспечивающая вывод меню на экран.

Расчет стипендии основан на определении количества различных оценок и анализе этого количества. После расчета стипендии необходимо записать обратно в файл очередную считанную и скорректированную структуру. Для этого выполняют подвод файлового курсора к началу только что считанной структуры и вывод измененной структуры в файл. После этого осуществляется считывание очередного элемента файла.

В качестве параметра в функции передается адрес указателя на файловую переменную.

Ниже после текста программы представлен пример протокола ввода/вывода. Обратите внимание, что вызов функции **init** установки требуемого режима открытия файла необходимо выполнять как при создании файла, так и при дальнейших работах с файлом.

Текст программы и протокол ввода/вывода:

```
#include "stdafx.h"
#include "stdlib.h"
#include "conio.h"
#include "string.h"
#include "d:\\temp\\RusString.h"

typedef int mas[5];
typedef  struct
{
   char f[32];
   float stip;
   int n;
   mas oc;
}stud;

void init(FILE **f);
void wwod(FILE **f);
void wiwod(FILE **f);
void stipen(FILE **f);
void menu();
```

```
int _tmain(int argc, _TCHAR* argv[])
{
   char ch,path[100];
   FILE *f;
   do
   {
      menu();
      ch=getch();
      switch (ch)
      {
         case '1': init(&f);break;
         case '2': wwod(&f);break;
         case '3': wiwod(&f);break;
         case '4':  stipen(&f);break;
         case '5':    break;
      }
   }while(ch!='5');
   return 0;
}
//Вывод меню
void menu()
{
   printf("\n1.INIT");
   printf("\n2. WWOD");
   printf("\n3. WIWOD");
   printf("\n4. STIPEN");
   printf("\n5. WIXOD");
   printf("\n");
}

//Инициализация файла
void init(FILE **f)
{
   char reg[10],path[30];
   printRus("Введите имя файла - ");
   scanf("%s",path);
   printRus("Введите режим - ");
   scanf("%s",reg);
   printRus("\nИмя файла  - ");printf("%s",path);
   printRus("\nРежим - ");printf("%s\n",reg);
   *f=fopen(path,reg);
}
```

```
//Ввод данных с клавиатуры и запись их в файл
void wwod(FILE **f)
{
   int nn;
   stud a;
   char c;
   nn=sizeof(a);
   do
   {
   printRus("\nВведите фамилию - ");
      scanf("%s",a.f);
   printRus("Введите количество оценок - ");
      scanf("%d",&a.n);
   printRus("Введите оценки - ");
      for (int i=0;i<a.n;i++)
         scanf("%d",&a.oc[i]);
      fwrite(&a,sizeof(a),1,*f);
   printRus("Продолжить ввод? (Y/N)");
      c=getch();
   }while(c!='N');
   fclose(*f);
}

//Просмотр содержимого файла и вывод данных на экран
void wiwod(FILE **f)
{
   int nn;
   stud a;
   nn=sizeof(a);
   while (fread(&a,nn,1,*f)>0)
   {
printRus("\nФамилия - "); printf("%s",a.f);
printRus("\nКоличество оценок - "); printf("%3d",a.n);
      printRus("\nОценки - ");
      for (int i=0;i<a.n;i++)
         printf("%3d ",a.oc[i]);
printRus("\nСтипендия - "); printf("%7.2f",a.stip);
   }
   fclose(*f);
}
//Расчет стипендии
void stipen(FILE **f)
{
```

```
// базовый размер стипендии
   const float st=1200;
   stud a;
   int i,j,dw,tr,che,ot,n,nom,ob;
   n=sizeof (a);nom=0;
   fseek(*f,0,SEEK_END);
   ob=ftell(*f);
   nom=ob/n;
   rewind(*f);
   j=0;
   while (fread(&a,n,1,*f)>0)
   {
      dw=0;tr=0;che=0; ot=0;
      for (i=0;i<a.n;i++)
      switch (a.oc[i])
      {
         case 2: {dw++;break;}
         case 3: {tr++;break;}
         case 4: {che++;break;}
         case 5: {ot++;break;}
      }
      if (dw>0)
         a.stip=0;
      if (ot==a.n)
         a.stip=2*st;
      if (dw==0&&tr!=0)
         a.stip=st;
      if (dw==0 && tr==0 && ot<a.n)
         a.stip=(float)1.5*st;
      fseek(*f,-n,SEEK_CUR);
      fwrite(&a,n,1,*f);
      j++;
      fseek(*f,j*n,SEEK_SET);
   }
}

/*
//ПРИМЕР ПРОТОКОЛА ВВОДА-ВЫВОДА РАБОТЫ ПРОГРАММЫ
1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 1 И ВВЕСТИ:
```

```
Введите имя файла - аааааа
Введите режим - wb
   //КОНТРОЛЬНЫЙ ВЫВОД
Имя файла - аааааа
Режим - wb

1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 2 И ВВЕСТИ:
Введите фамилию - ааа
Введите количество оценок - 2
Введите оценки - 2 3
Продолжить ввод? (Y/N)
      //НАЖАТЬ ЛЮБУЮ КЛАВИШУ, КРОМЕ N
Введите фамилию - bbb
Введите количество оценок - 2
Введите оценки - 3 4
Продолжить ввод? (Y/N)
      //НАЖАТЬ ЛЮБУЮ КЛАВИШУ, КРОМЕ N
Введите фамилию - ссс
Введите количество оценок - 2
Введите оценки - 4 5
Продолжить ввод? (Y/N)
      //НАЖАТЬ ЛЮБУЮ КЛАВИШУ, КРОМЕ N
Введите фамилию - ddd
Введите количество оценок - 2
Введите оценки - 5 5
Продолжить ввод? (Y/N)
      //НАЖАТЬ КЛАВИШУ N
1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 1 И ВВЕСТИ:
Введите имя файла - аааааа
Введите режим - rb+
   //КОНТРОЛЬНЫЙ ВЫВОД
Имя файла - аааааа
Режим - rb+
```

```
1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 4 (ВЫЗОВ STIPEN)
1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 1 И ВВЕСТИ:
Введите имя файла - аааааа
Введите режим - rb+
   //КОНТРОЛЬНЫЙ ВЫВОД
Имя файла - аааааа
Режим - rb+

1.INIT
2. WWOD
3. WIWOD
4. STIPEN
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 4 (ВЫЗОВ WIWOD)
Фамилия - aaa
Количество оценок -2
Оценки -23
Стипендия - 0.00
Фамилия - bbb
Количество оценок -2
Оценки -34
Стипендия - 1200.00
Фамилия - ccc
Количество оценок -2
Оценки -45
Стипендия - 1800.00
Фамилия - ddd
Количество оценок -2
Оценки -55
Стипендия - 2400.00
1.INIT
2. WWOD
3. WIWOD
4. STIPEN
```

```
5. WIXOD
      //НАЖАТЬ КЛАВИШУ 5 (ВЫХОД ИЗ ПРОГРАММЫ)
Для продолжения нажмите любую клавишу . . .
*/
```

4.6. Задания на обработку бинарных файлов

Во всех заданиях требуется использовать меню, ввод и вывод данных сопровождать подсказками. Каждый пункт задания реализовать в виде отдельной функции.

1. Создать файл, состоящий из структур с информацией о студентах. Структура содержит следующие поля: фамилия, количество оценок, массив оценок (не более пяти). Скорректировать содержимое файла по результатам пересдач неудовлетворительных оценок. В этом случае программа должна выдавать информацию о всех неудовлетворительных оценках и запрашивать новую оценку. Обеспечить просмотр содержимого файла.

2. Создать файл, состоящий из структур с информацией о сотрудниках. Структура содержит следующие поля: фамилия, имя, год рождения, образование (среднее, высшее, кандидат наук, доктор наук). Осуществить поиск структур по заданной фамилии и коррекцию полей этой структуры. Выполнить добавление новых структур в конец файла. Обеспечить просмотр содержимого файла.

3. Создать файл, состоящий из структур с информацией о спортсменах-фигуристах. Структура содержит следующие поля: фамилия, результат в обязательной и прозвольной программах, итоговая оценка. Осуществить сначала вывод первых двух оценок. Обеспечить затем вычисление итоговой оценки по формуле Оц = $= K_1 \cdot Оц_1 + K_2 \cdot Оц_2$, где K_1, K_2 — весовые коэффициенты, вводимые пользователем. Обеспечить просмотр содержимого файла.

4. Создать файл, состоящий из структур с информацией о студентах. Структура содержит следующие поля: фамилия, количество оценок, массив оценок (не более пяти), размер стипендии. Выполнить вывод в файл исходных данных (фамилии и оценки). Рассчитать размер стипендии для каждого студента по следующему правилу: при наличии двойки стипендия не выплачивается, при наличии тройки или при всех четверках выплачивается обычная стипендия А руб., при четверках и одной пятерке — повышенная на 25 %, при всех пятерках — повышенная на 50 %. Полученные значения занести в файл. Предусмотреть возможность коррекции

оценок (в случае пересдач). Обеспечить просмотр содержимого файла.

5. Создать файл, состоящий из структур с информацией о студентах. Структура содержит следующие поля: фамилия, количество модулей (не более четырех), массив баллов, набранных на каждом модуле, итоговый результат (зачет/незачет). Выполнить вывод в файл исходных данных (фамилии и баллы). Определить итоговый результат и занести его в файл. Зачет ставится при результате от 60 баллов и выше, в противном случае — незачет. Предусмотреть возможность коррекции оценок (в случае пересдач). Обеспечить просмотр содержимого файла.

6. Создать файл, состоящий из структур с информацией о биатлонистах. Структура содержит следующие поля: фамилия, результат (минуты, секунды с десятыми и сотыми долями, количество промахов), итоговый результат. Выполнить вывод в файл исходных данных (фамилии и результат). Определить итоговый результат и занести его в файл. Итоговый результат формируется путем прибавления к исходному результату одной минуты штрафа за каждый промах. Обеспечить просмотр содержимого файла.

7. Создать файл, состоящий из структур с информацией о пригородных билетах. Структура содержит следующие поля: название станции, номер зоны, стоимость полного билета, стоимость детского билета. Выполнить вывод в файл исходных данных (название станции и номер зоны). Определить для каждой станции стоимость билетов и занести ее в файл. Стоимость проезда вычисляется путем умножения номера зоны на А руб., детский билет стоит в четыре раза дешевле. Обеспечить просмотр содержимого файла и выдачу информации по запросу о названии станции, стоимости билетов и номере зоны.

8. Создать файл, состоящий из структур с информацией о погодных условиях в разных городах. Структура содержит следующие поля: название города, месяц года, средняя температура и количество осадков для этого месяца. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла и выдачу информации об определенном городе по запросу. Отсортировать содержимое файла в алфавитном порядке городов.

9. Создать файл, состоящий из структур с информацией о результатах соревнований. Структура содержит следующие поля:

фамилия спортсмена, название страны, занятое место. Выполнить вывод в файл исходных данных. Сформировать второй файл с информацией о результатах каждой страны, каждая структура второго файла содержит следующие поля: название страны, количество золотых, серебряных, бронзовых медалей. Обеспечить просмотр содержимого каждого файла и выдачу информации о количестве медалей для определенной страны по запросу.

10. Создать файл, состоящий из структур с информацией о номерах телефонов. Структура содержит следующие поля: фамилия, имя человека, номер его телефона. Выполнить вывод в файл исходных данных. Предусмотреть добавление новых структур в конец файла и возможность коррекции структур файла (изменение номера телефона). Обеспечить просмотр содержимого файла и выдачу информации об определенном человеке по запросу.

11. Создать файл, состоящий из структур с информацией о запасах товаров в магазине. Структура содержит следующие поля: название товара, нормативный запас, фактическое количество. Выполнить вывод в файл исходных данных. Предусмотреть добавление новых структур в конец файла и коррекцию структур файла (изменение текущего количества). Обеспечить просмотр содержимого файла и выдачу информации о товарах с достаточным запасом, а также выдачу информации о товарах с запасом меньше нормативного.

12. Создать файл, состоящий из структур с информацией о книгах библиотеки. Структура содержит следующие поля: название книги, автор, наличие в библиотеке (есть, выдана, вообще отсутствует), наличие в читальном зале (есть, нет). Выполнить вывод в файл исходных данных. Предусмотреть добавление новых структур в конец файла и коррекцию структур файла. Обеспечить просмотр содержимого файла и выдачу информации о книге по запросу.

13. Создать файл, состоящий из структур с информацией о блюдах столовой. Структура содержит следующие поля: название блюда, цена, наличие в столовой (есть, нет). Выполнить вывод в файл исходных данных. Предусмотреть добавление новых структур в конец файла и коррекцию структур файла. Обеспечить просмотр содержимого файла и выдачу информации о наличии запрашиваемого блюда и его стоимости, а также обо всех имеющихся в наличии блюд и их стоимости.

14. Создать файл, состоящий из структур с информацией о студентах. Структура содержит следующие поля: фамилия, количество полученных домашних заданий (не более четырех), массив номеров вариантов, массив с информацией о выполнении задания (сдал/не сдал). Осуществить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о сданных заданиях. Обеспечить просмотр содержимого файла, а также выдачу информации о студентах, сдавших все задания, и о студентах, не сдавших ни одного задания.

15. Создать файл, состоящий из структур с информацией о книгах библиотеки. Структура содержит следующие поля: название книги, автор, год издания. Выполнить вывод в файл исходных данных. Предусмотреть добавление новых структур в конец файла. Обеспечить просмотр содержимого файла и выдачу информации о книгах автора по запросу.

16. Создать файл, состоящий из структур с информацией о подписных изданиях. Структура содержит следующие поля: название издания, вид (газета или журнал), индекс, цена подписки на месяц. Выполнить вывод в файл исходных данных и сортировку файла по названиям в алфавитном порядке. Предусмотреть добавление новых структур в файл с сохранением упорядоченности. Обеспечить просмотр содержимого файла.

17. Создать файл, состоящий из структур с информацией о жителях городов. Структура содержит следующие поля: название города, численность жителей, изменение численности жителей за год. Выполнить вывод в файл исходных данных и сортировку файла по названиям городов в алфавитном порядке. Предусмотреть добавление новых структур в файл с сохранением упорядоченности. Обеспечить просмотр содержимого файла и выдачу информации о городах, имеющих положительный прирост населения и отрицательный.

18. Создать файл, состоящий из структур с информацией о погоде в разных городах. Структура содержит следующие поля: название города, среднегодовая температура. Выполнить вывод в файл исходных данных и сортировку файла в порядке возрастания температуры. Предусмотреть добавление новых структур в файл с сохранением упорядоченности. Обеспечить просмотр содержимого файла и выдачу информации о городах, имеющих температуру выше средней по всем городам и температуру ниже средней.

19. Создать файл, состоящий из структур с информацией о студентах. Структура содержит следующие поля: фамилия, количество сдаваемых зачетов (не более шести), массив с информацией о полученном результате (сдал/ не сдал). Выполнить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о сданных зачетах. Обеспечить просмотр содержимого файла и выдачу информации о студентах, сдавших все зачеты, а также о студентах, имеющих задолженности.

20. Создать файл, состоящий из структур с информацией о сотрудниках. Структура содержит следующие поля: фамилия, оклад, сумма к выдаче. Выполнить вывод в файл исходных данных (сначала фамилии и оклады). Для задаваемых значений суммы, не облагаемой налогом, и налога вычислить для каждого сотрудника значение последнего поля и занести в файл. Предусмотреть возможность коррекции содержимого файла (изменение оклада у сотрудников). Обеспечить просмотр содержимого файла.

21. Создать файл, состоящий из структур с информацией о преподавателях. Структура содержит следующие поля: фамилия, количество принимаемых экзаменов (не более десяти), массив с информацией о датах экзаменов. Выполнить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о датах экзаменов. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу информации о преподавателе с заданной фамилией по запросу.

22. Создать файл, состоящий из структур с информацией о странах. Структура содержит следующие поля: название страны, название столицы. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу информации о заданной стране по запросу. Упорядочить файл по названиям стран в алфавитном порядке.

23. Создать файл, состоящий из структур с информацией о городах. Структура содержит следующие поля: название города, численность жителей. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из четырех видов информации о городах с численностью

населения: 1) более 1 млн; 2) от 0,5 до 1 млн; 3) от 0,1 до 0,5 млн; 4) менее 0,1 млн. Упорядочить файл по названиям городов в алфавитном порядке.

24. Создать файл, состоящий из структур с информацией об изучаемых дисциплинах. Структура содержит следующие поля: название дисциплины, количество семестров, в которых изучается дисциплина, массив номеров семестров. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из двух видов информации: 1) для заданной дисциплины — о номерах семестров, в которых она изучается; 2) для заданного семестра — о названиях всех изучаемых дисциплин. Упорядочить файл по названиям дисциплин в алфавитном порядке.

25. Создать файл, состоящий из структур с информацией о телепередачах. Структура содержит следующие поля: название телепередачи, день и время ее трансляции. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из двух видов информации: 1) для заданной передачи — о дне и времени ее трансляции; 2) для заданных даты и времени — о названии телепередачи. Упорядочить файл по названиям передач в алфавитном порядке.

26. Создать файл, состоящий из структур с информацией о городах. Структура содержит следующие поля: название города, название самой большой реки, протекающей в этом городе. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из двух видов информации: 1) для заданного города — о названии реки; 2) для заданной реки — о названиях всех городов, где она протекает. Упорядочить файл по названям городов в алфавитном порядке.

27. Создать файл, состоящий из структур с информацией об экзаменах. Структура содержит следующие поля: название группы студентов, количество сдаваемых экзаменов (не более пяти), массив с информацией о датах экзаменов. Выполнить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о датах экзаменов. Обеспечить просмотр содержимого фай-

ла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из двух видов информации: 1) для заданной группы — о дате экзаменов; 2) для заданной даты — о названиях всех групп, сдающих экзамены в этот день. Упорядочить файл по названиям групп.

28. Создать файл, состоящий из структур с информацией о странах. Структура содержит следующие поля: название страны, название наиболее известного горного массива. Выполнить вывод в файл исходных данных. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из двух видов информации: 1) для заданной страны — о названии горного массива; 2) для заданного горного массива — о названиях всех стран, где он расположен. Упорядочить файл по названиям стран в алфавитном порядке.

29. Создать файл, состоящий из структур с информацией об экзаменах. Структура содержит следующие поля: фамилия студента, признак наличия задолженности, причина задолженности (уважительная/неуважительная). Выполнить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о наличии задолженности. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из трех видов информации: 1) список успевающих студентов; 2) список студентов, не успевающих по уважительной причине; 3) список студентов, не успевающих по неуважительной причине. Упорядочить файл по фамилиям в алфавитном порядке.

30. Создать файл, состоящий из структур с информацией о преподавателях. Структура содержит следующие поля: фамилия преподавателя, индекс кафедры, должность (ассистент, старший преподаватель, доцент, профессор). Выполнить вывод в файл исходных данных. Предусмотреть возможность коррекции информации о кафедре и должности преподавателя. Обеспечить просмотр содержимого файла. Предусмотреть добавление новых структур в файл. Обеспечить выдачу по запросу одного из трех видов информации: 1) о преподавателях с заданной фамилией; 2) о всех преподавателях заданной кафедры; 3) о всех преподавателях, занимаемых заданную должность. Упорядочить файл по фамилиям в алфавитном порядке.

Вопросы для самопроверки

1. Что называют файлом?
2. Какие виды файлов есть в языке Си?
3. Из каких компонентов (элементов) состоит текстовый файл?
4. Как называют компоненты текстовых файлов?
5. Какими символами заканчиваются строки файлов?
6. Есть ли ограничения на длины строк файлов?
7. Может ли строка файла иметь нулевую длину?
8. Можно ли просматривать текстовый файл в текстовых редакторах?
9. С чего начинается работа с текстовым файлом и чем заканчивается?
10. В каких режимах можно открывать текстовый файл?
11. Как определить файловую переменную?
12. Что представляют параметры функции fopen?
13. Что называют курсором файла?
14. В чем различие операторов чтения строковых данных из файла scanf и fgets?
15. Какие методы доступа применимы к бинарным файлам?
16. Какие библиотечные функции используются для прямого доступа к данным бинарного файла?
17. Могут ли компоненты бинарного файла иметь разную длину?
18. Выполняется ли преобразование данных при выводе в бинарный файл?
19. Какая библиотечная функция используется для обнаружения конца файла?
20. Какие библиотечные функции предназначены для вывода в текстовый файл?
21. Какие библиотечные функции служат для ввода из текстового файла?
22. Какие библиотечные функции используются для вывода в бинарный файл?
23. Какие библиотечные функции предназначены для ввода из бинарного файла?
24. Какая библиотечная функция используется для закрытия файла?
25. Какие библиотечные функции предназначены для удаления файла и переименования файла?

5. Тип данных «указатель»

5.1. Общие сведения

Объявления и присваивания

Указатель — это переменная, хранящая адрес некоторой области памяти. Указатели играют в языке С большую роль. Это связано с тем, что некоторые операции можно выполнить только с помощью указателей, а также с тем, что использование указателей в ряде случаев позволяет записать код более компактно и эффективно по сравнению с другими способами.

В языке различают три вида указателей: на объект, на функцию и на void (пустой тип). Разные виды указателей различаются свойствами и набором допустимых операций. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим типом данных.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного). Простейшее объявление указателя на объект имеет следующий вид:

```
Тип *имя;
```

Тип может быть любым, кроме ссылки и битового поля, причем тип к этому моменту может быть только объявлен, но еще не определен. Поэтому в структуре, например, может присутствовать указатель на структуру того же типа.

Символ «*» относится непосредственно к имени, поэтому при объявлении нескольких указателей требуется ставить его перед каждым именем.

Примеры объявления нескольких указателей:

1) `char *c1, *ch, *c2;`
2) `int *n,*m;`
3) `float *a, *b;`

Здесь в первом случае объявлены три указателя на символьный тип с именами `c1`, `ch`, `c2`, во втором случае — два указателя на целое с именами `n`, `m`, в третьем случае — два указателя на тип `float` с именами `a`, `b`.

Указатель на `void` применяется, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, в одной и той же области памяти в разные моменты времени предполагается хранить адреса объектов разных типов).

Указателю на `void` можно присвоить значение указателя любого типа, а также сравнить его с любым указателем, но перед выполнением каких-либо действий с областью памяти, адрес которой он хранит, требуется преобразовать его к конкретному типу явным образом.

Указатель на функцию содержит адрес в сегменте кода, по которому располагается исполняемый код функции, т. е. адрес, по которому передается управление при вызове функции. Указатели на функцию используются для косвенного вызова функции: не через ее имя, а через обращение к переменной, хранящей ее адрес, а также для передачи имени функции в другую функцию в качестве параметра. Указатель на функцию имеет тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа», и объявляется конструкцией вида

```
Тип (*имя) (список типов аргументов);
```

Пример объявления типа указателя на функцию:

```
float (*fun1) (float, float);
```

задает указатель с именем `fun1` на функцию, возвращающую значение типа `float` и имеющую два аргумента типа `float`.

Указатель может быть константой или переменной, а также указывать на константу или переменную. Рассмотрим следующие примеры:

```
float a;//переменная типа float
const float b=2.1; //константа типа float
float *p;//указатель на переменную типа float
const float *pa;//указатель на константу
                //типа float
float * const cc=&a;//указатель-константа на
                    //переменную типа float
const float *const pb=&b;//указатель-константа
                         //на константу типа float
```

Модификатор const, находящийся между именем указателя и звездочкой («*»), относится к самому указателю и запрещает его изменение, а const, расположенный слева от звездочки, задает постоянное значение, на которое он указывает. Для инициализации указателей используется операция получения адреса **&**. Значения указателей можно вводить и выводить. Для этого применяется спецификация **p**, например

```
float *p10,*p11;
scanf("%p %p",&p10,&p11);
printf("\n p10=%p p11=%p",p10,p11);
```

Необходимо иметь в виду, что адрес представляет собой восьмиразрядное шестнадцатеричное число, например введены значения **0022FFAB** и **0022FFBB**. При этом следует отметить, что ввод адресов большого смысла не имеет, поскольку пользователю не известно, какая информация хранится в памяти по заданному адресу.

Начальное значение адресу можно присвоить при его объявлении, т. е. выполнить инициализацию. Это можно реализовать разными способами:

1) с помощью операции получения адреса:

```
foat a=6.6; float *p=&a;
```

или

```
float *p(&a);
```

2) путем присвоения значения другого инициализированного указателя:

```
foat a=6.6; float *p=&a; float *pp=p;
```

3) посредством присвоения указателю адреса некоторой области памяти явно:

```
char *ch=(char *)0x0044FFBB;
```

Шестнадцатеричная константа **0x0044FFBB** с помощью операции приведения типов (**char ***) преобразуется к типу «указатель на char».

Если значение указателю будет присваиваться в ходе выполнения программы, то целесообразно в начале программы задать ему значение пустого указателя **NULL** (0). Это в определенной смысле аналог нулевого значения для числовых типов. Поскольку объек-

тов с нулевым значением адреса не существует, то пустой указатель используется для проверки, ссылается ли указатель на конкретный объект или нет. Присваивание нулевого значения указателю часто помогает избежать ошибок в программах, связанных с применением неинициализированных указателей.

Операции с указателями

С указателями можно выполнять следующие операции: разадресация (косвенное, т. е. по адресу, обращение к объекту), присваивание, сложение с константой, вычитание, инкремент, декремент, сравнение, приведение типов.

Операция разадресации, или разыменования, предназначена для доступа к содержимому памяти, адрес которой хранится в указателе. Эту операцию можно использовать как для получения хранимого в памяти значения, так и для изменения этого значения (если эта величина не объявлена как константа). Операция разадресации записывается как

```
*<имя переменной типа указатель>
```

Пример операции разадресации с указателями:

```
float *d, b,x;
b=4.89;
x=2.2;
d=&b;
*d=*d/(*d+x);
```

В данном примере указателю `d` будет присвоено значение адреса переменной `b`, а затем содержимое этой области памяти (фактически значение переменной `b`) будет изменено в соответствии с записанным оператором присваивания, т. е. в область памяти, адрес которой хранит переменная `d`, занесется содержимое этой области (значение `b`), деленное на сумму слагаемых `(*d+x)`, первое из которых есть то же самое, что и числитель, а второе — значение переменной `x`.

Указателю можно присваивать значение адреса, а также значение другого указателя такого же типа (связанного с тем же типом). Присваивание указателя другого типа требует явного приведения типов. Это иллюстрируется следующими примерами:

```
float *d,*b, c=7.8;
int m=9;
```

```
b=&c;
d=b;
printf("res=%6.2f",*d);
d=(float *)&m;
printf("res=%6.2f",*d);
```

Однако надо иметь в виду, что в первом выводе значения `*d` результат будет отпечатан правильный, т. е. на экране пользователь увидит значение 7.8, а во втором выводе содержимое памяти, где хранится целое значение 9, будет рассматриваться как действительная величина и результат будет получен, отличный от 9.

Присваивание значения без явного преобразования типов допускается также в случае использования указателей `void *`, стоящих в левой части оператора присваивания:

```
void *p1;
float *a;
p1=a;
```

Обратное присваивание `a=p1;` является неверным, поэтому требуется преобразование типов в явном виде: `a=(float *)p1;`

Таким образом, неявное преобразование указателей производится только к типу `void`. Присваивание указателей на объекты указателям на функции (и наоборот) недопустимо. Запрещено присваивать значения указателям-константам. Например, последовательность операторов

```
float aa;
float * const cc; // cc является указателем-константой
cc=&aa;
```

является неправильной, так как делается попытка присваивания указателю-константе некоторого значения (адреса переменной).

Указателям на константу и переменную допускается присваивать значения. Приведенная ниже последовательность операторов является допустимой:

```
float aa;
const float *pa; // pa является указателем на константу
pa=&aa;
```

Здесь указателю на константу присваивается значение адреса переменной.

Арифметические операции с указателями автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например с массивами.

Пример сложения указателя с константой:

```
float *a;
const int n=4;
a=a+n;
```

Здесь при сложении с константой указатель фактически получает значение `a+n*sizeof(float)`, так как в данном случае указатель связан с типом `float`.

Аналогично сложению можно вычитать константу, т. е. складывать с отрицательным значением:

```
a=a-n;
```

Разность двух указателей — это разность их значений, деленная на размер в байтах элемента типа данных, с которым связан указатель. Например, можно записать такую последовательность операторов:

```
float aa,bb;
float *pr1,*pr2;
int nr;
pr1=&bb;
pr2=&aa;
nr=pr1-pr2;
printf("\npr1=%p pr2=%p  nr=%4d\n" ,pr1,pr2,nr);
```

Результатом будет строка вывода:

```
pr1=0043FAD4 pr2=0043FAE0 nr= -3
```

Из этой записи видно, что разность двух указателей является величиной целочисленной, показывающей фактически, каким по счету является элемент, на который указывает первый указатель, относительно элемента, на который указывает второй указатель. Применять вычитание просто к указателям нет смысла, так как между переменными, на которые указывают рассматриваемые указатели, в памяти могут размещаться данные разных типов, занимающие в памяти соответственно и разное количество байтов.

Поэтому разность указателей используют при работе с массивами, когда элементы массива размещаются в соседних ячейках памяти.

Инкремент (++) и декремент (--) являются, по сути, частными случаями сложения указателя с константой. Эти операции прибавляют к указателю или вычитают из указателя величину `sizeof(тип)`, где `тип` — тип данных, с которым связан указатель.

С указателями можно выполнять операции отношения. Например, если `pr1`, `pr2` — однотипные указатели, то можно записать следующие операторы:

```
printf(
"\npr1=pr2 =%d  pr1!=pr2 =%d pr1>pr2=%d pr1<pr2=%d\n",
   pr1==pr2,pr1!=pr2,pr1>pr2,pr1<pr2);
printf(
"\npr1>=pr2 =%d pr1<=pr2=%d\n",pr1>=pr2,pr1<=pr2);
```

Результатом будут строки вывода:

```
pr1=pr2 =0  pr1!=pr2 =1 pr1>pr2=0 pr1<pr2=1

pr1>=pr2 =0 pr1<=pr2=1
```

Указатели нельзя складывать, умножать, делить.

Приоритеты операций при работе с указателями

Операции разадресации, инкремента, декремента и взятия адреса имеют одинаковый приоритет и выполняются справа налево. Рассмотрим примеры выполнения этих операций:

- `*++p` — здесь сначала выполнится операция инкремент указателя (сложение значения с единицей, т. е. прибавление к его значению количества байтов, отводимых для хранения данных того типа, с которым связан указатель), а затем будет получено содержимое памяти по новому адресу;
- `*p++` — в этом выражении сначала будет использовано значение переменной, хранящейся по адресу `p`, а затем указатель увеличится на количество байтов, занимаемых значением типа, с которым связан указатель, поскольку инкремент является здесь постфиксным;
- `(*p)++` — в этом выражении значение переменной, хранящейся по адресу `p`, увеличится на единицу;

• `++*p` — в приведенном выражении осуществляется сначала разадресация и полученное значение переменной, хранящейся по адресу `p`, увеличивается на единицу;

• `++*++p` — в данном выражении сначала увеличивается значение указателя на единицу, затем выполняется разадресация и полученное значение переменной, хранящейся по новому адресу, увеличивается на единицу.

Пример 1. Следующая программа иллюстрирует выполнение операций с указателями:

```
#include "stdafx.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
   float a[5];
   float *p;
   a[0]=7.7; a[1]=8.8;
   a[2]=9.9; a[3]=10.8; a[4]=17.7;
   p=a;
   printf("a[0]=%6.2f  *p=%6.2f p=%p",a[0],*p,p);
  printf("\na[1]=%6.2f  *++p=%6.2f  ++p=%p adres [1]=%p",
            a[1],*++p,p,&a[1]);
   printf("\na[1]=%6.2f    ",a[1]);
   (*p)++ ;
   printf("\n (*p)++=%6.2f",*p);
   ++*p;
   printf("\n ++*p=%6.2f",*p);
   ++*++p ;
   printf("\n ++*++p=%6.2f",*p);
   getch();
   return 0;
}
```

Взаимосвязь массивов и указателей

В языке С существует очень тесная взаимосвязь между указателями и массивами. Любую операцию, выполняемую с помощью индексации массива, можно провести с применением указателей. В программе, приведенной в примере 1, был объявлен массив `float a[5];` и указатель на действительную величину `float *p;`. Поскольку имя массива в языке С есть не что иное, как адрес первого элемента массива, то присваивание `p=a;` означает, что указа-

тель теперь хранит адрес первого элемента массива `a`. То же самое можно было выполнить и с помощью оператора присваивания `p=&a[0];`. Если выполнить оператор присваивания `p+=i;`, то указатель будет хранить адрес i-го элемнта, расположенного после `p`, в данном случае просто i-го элемента массива `a`. Тогда выражение `*(p+i)` означает не что иное, как содержимое i-го элемента массива, т. е. `a[i]`. При вычислении выражения `a[i]` компилятор сам преобразует его в форму `*(a+i)`, так как эти две формы являются тождественными. Если применить к этим двум выражениям адресную операцию `&`, то можно получить также две тождественные записи: `&a[i]` и `a+i`. Указатель можно использовать и в выражениях с индексом — `p[i]` будет соответствовать записи `*(p+i)`. Таким образом, выражение в виде обращения к элементу массива по индексу эквивалентно ссылке по указателю со смещением.

Однако есть и различие между именем массива и указателем. Указатель является переменной, поэтому операторы `p=a; p++;` допустимы, а операторы с именем массива, подобные приведенным (см. опрерации с указателями), недопустимы: `a=p; a++;`

5.2. Динамические переменные и массивы

Динамические переменные

Динамические переменные — это переменные, память для хранения которых выделяется в ходе выполнения программы. Такие переменные хранятся в динамической памяти переменного размера (куче). Обращение к таким переменным выполняется с помощью указателей, хранящих их адреса. Фактически у динамических переменных нет имен. Создание динамической переменной означает выделение для нее памяти в соответствии с ее типом (занимаемым объемом) в динамической области памяти, занесение в выделенный участок памяти значения и использование этого значения в ходе выполнения программы. Для выделения памяти под динамическую переменную служит операция new (введена в языке C++):

```
new <базовый тип указателя>.
```

В результате выполнения этой операции система находит свободный участок памяти в динамической области, достаточный для хранения переменной указанного типа, выделяет требуемую память и адрес начала выделенной области помещает в указатель,

при этом выделенный участок памяти определяет как занятый. Оператор присваивания переменной-указателю полученного адреса выделенной памяти может быть записан следующим образом:

```
<переменная-указатель> = new <базовый тип указателя>;
```

Занятый участок памяти можно освобождать, т. е. возвращать в кучу. Для этого предназначена операция **delete** (введена в языке C++):

```
delete <указатель>;
```

Для доступа к выделенной области памяти используется разыменование указателя (разадресация). Конструкция

```
*<.имя переменной-указателя>
```

фактичеки является динамической переменной, к которой можно применять все операции, допустимые для переменных базового типа.

Пример 1. Следующая программа демонстрирует ввод и выполнение четырех арифметических операций с динамическими переменными целого и действительного типов:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"
int _tmain(int argc, _TCHAR* argv[])
{
   int *m,*n,*sk,*rk,*pk,*dk;
   m=new int;  n=new int;  sk=new int;
   rk=new int;  pk=new int;  dk=new int;
   printRus("Введите два целых числа: ");
   scanf("%d %d",m,n);
   *sk=*m+*n; *rk=*m-*n; *pk=*m**n; *dk=*m/(*n);
   printf("\n s=%3d rasn=%3d  pr=%3d ch=%3d",*sk,*rk,*pk,*dk);
   float *a,*b,*sr,*rr,*pr,*dr;
   a=new float;  b=new float;  sr=new float;  rr=new float;
   pr=new float;
   dr=new float;
   printRus("\nВведите два вещественных числа: ");
   scanf("%f %f",a,b);
   *sr=*a+*b; *rr=*a-*b; *pr=*a**b; *dr=*a/(*b);
```

```
    printf("\n s=%6.2f rasn=%6.2f pr=%6.2f ch=%6.2f \n",
           *sr,*rr,*pr,*dr);
    getch();
    delete m,n,sk,rk,pk,dk,a,b,sr,rr,pr,dr;
    return 0;
}
```

Динамические массивы

Память для динамических массивов выделяется в ходе выполнения программы в соответствии с фактичеким количеством элементов. Это создает определенные удобства по сравнению с применением статических массивов, так как каждый раз выделяется минимально необходимый объем памяти для хранения элементов массива. При использовании статических массивов приходится выделять память с запасом, т. е. в расчете на максимально возможное (прогнозируемое) количество элементов. Объявление указателя на динамический массив записывается так же, как и на динамическую переменную:

```
<тип компонентов массива> *<идентификатор указателя>;
```

Для выделения памяти под массив служит операция **new**, при ее выполнении необходимо указать тип элементов массива и их количество:

```
<указатель-идентификатор массива>
=new <тип элементов массива> [<количество>];
```

В результате выполнения операции выделится непрерывная область памяти для размещения заданного количества элементов массива. Объем памяти определяется как произведение количества элементов и объема памяти, занимаемого одним значением указанного типа. Адрес начала выделенной области памяти заносится в указатель.

Для освобождения памяти используется оператор **delete**, размерность при этом не указывается:

```
delete[]<идентификатор указателя>;
```

Пример 2. Приведенная в примере программа поясняет работу с динамическим одномерным массивом. В программе вводится количество элементов массива, затем выделяется память, выполняется ввод элементов динамического массива, а затем находится

максимальный элемент массива и вычисляется произведение всех его элементов. В заключение выводятся элементы массива, полученный результат, после чего выполняется освобождение выделенной памяти.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   int n,i;
   float *p,max,pp;
   printRus("Введите количество элементов: ");
   scanf("%d",&n);
   p=new float[n];
   OutPuts("Введите элементы массива ");
   for (i=0;i<n;i++)
      scanf("%f",&p[i]);
   OutPuts("Вывод массива ");
   for (i=0;i<n;i++)
      printf("%6.2f",p[i]);
   max=p[0];
   pp=1;
   for (i=0;i<n;i++)
   {
      if (p[i]>max)
         max=p[i];
      pp*=p[i];
   }
   printf("\n max=%6.2f pr=%6.2f\n",max,pp);
   delete []p;
   getch();
   return 0;
}
```

Двумерные динамические массивы

Двумерный динамический массив (матрицу) удобно рассматривать как одномерный массив, состоящий из одномерных массивов. Поэтому каждой строке матрицы (одномерному массиву)

можно поставить в соответствие указатель, хранящий адрес первого элемента очередной строки. Совокупность этих указателей образует одномерный массив указателей, этому массиву можно сопоставить указатель на массив указателей, т. е. указатель на указатель. В связи с этим при объявлении динамической матрицы следует объявить указатель на указатель:

```
<тип элементов массива> ** <идентификатор указателя>;
```

Идентификатор указателя является в данном случае указателем на массив указателей. После объявления указателя следует выделить память под матрицу, причем это выполняется в два этапа.

На первом этапе надо выделить память для массива указателей. Подобный массив может быть объявлен следующим образом:

```
<тип элементов> *<идентификатор>[<количество строк матрицы>];
```

В этом объявлении используется правило «суффикс крепче префикса», т. е. данная конструкция воспринимается как массив указателей (а не указатель на массив). Память под массив указателей выделяется, как и ранее, операцией **new**:

```
<идентификатор> = new <тип элементов> *[ количество строк];
```

Пример выделения памяти для динамических массивов:

```
a=new float *[20];
n=new char *[30];
```

При выполнении первого оператора выделяется память под массив указателей на тип **float**, состоящий из 20 элементов, адрес заносится в переменную-указатель **a**. При выполнении второго оператора выделяется память под массив указателей на тип **char**, состоящий из 30 элементов, адрес массива заносится в переменную-указатель **n**.

На втором этапе необходимо выделить память для хранения всех элементов каждой строки матрицы. Обычно это осуществляется в цикле с помощью операции **new**, при этом очередному указателю (элементу массива) присваивается адрес первого элемента очередной строки матрицы. Здесь следует отметить, что количество элементов строк динамической матрицы может быть различным, поэтому можно работать с динамическими матрицами различной конфигурации. Доступ к элементам массива указателей

при выделении памяти подстроки динамической матрицы можно выполнять несколькими способами: путем смещения от начала массива, с использованием индекса, с помощью вспомогательного текущего указателя. Приведем примеры применения каждого способа выделения памяти для первой матрицы:

1) путем смещения:

```
for (int i=0; i<20;i++)
   *(a+i)=new float[i+1];
```

2) с использованием индекса:

```
for (int i=0; i<20;i++)
   a[i]=new float[i+1];
```

3) с помощью вспомогательного текущего указателя:

```
float **p;
for (p=a; p<a+20;p++)
   *p=new float[i+1];
```

Освобождение памяти, выделенной под динамическую матрицу, выполняется в два этапа: сначала освобождается память, выделенная под каждую строку матрицы, а затем освобождается память, выделенная под массив указателей на каждую строку матрицы. Это может выглядеть следующим образом:

```
for (int i=0;i<20;i++)
   delete [] a[i];
   delete []a;
```

Пример 3. Приведенная в этом примере программа, в которой выполняется работа с динамической матрицей: выделяется память под динамическую матрицу и одномерный динамический массив, осуществляется ввод элементов матрицы, а затем вычисляется сумма элементов каждой строки матрицы и заносится в одномерный массив.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   int m,n,i,j;
   float **a, *b;
```

```
    printRus("Введите количество строк и столбцов: ");
    scanf("%d %d",&m,&n);
    a=new float*[m];
    for (i=0;i<m;i++)
       a[i]=new float[n];
    OutPuts("Введите элементы матрицы ");
    for (i=0;i<m;i++)
       for (j=0;j<n;j++)
          scanf("%f",&a[i][j]);
    OutPuts("Вывод матрицы ");
    for (i=0;i<m;i++)
    {
       for (j=0;j<n;j++)
          printf("%8.2f",a[i][j]);
       printf("\n");
    }
    b=new float[m];
    for (i=0;i<m;i++)
    {
       b[i]=0;
       for (j=0;j<n;j++)
          b[i]+=a[i][j];
    }
    OutPuts("Суммы элементов строк матрицы ");
    for (i=0;i<m;i++)
       printf("%6.2f\n",b[i]);
    delete []b;
    for (i=0;i<m;i++)
       delete []a[i];
    delete []a;
    getch();
    return 0;
}
```

Пример 4. Приведенная в этом примере программа демонстрирует работу с динамической матрицей, имеющей треугольный вид. При этом в каждой строке матрицы вычисляется произведение элементов.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"
```

```
int _tmain(int argc, _TCHAR* argv[])
{
   int m,i,j;
   float **a, *b;
   printRus("Введите количество строк: ");
   scanf("%d",&m);
   a=new float*[m];
   for (i=0;i<m;i++)
      a[i]=new float[i+1];
   OutPuts("Введите элементы матрицы ");
   for (i=0;i<m;i++)
      for (j=0;j<i+1;j++)
         scanf("%f",&a[i][j]);
   OutPuts("Вывод матрицы ");
   for (i=0;i<m;i++)
   {
      for (j=0;j<i+1;j++)
         printf("%8.2f",a[i][j]);
      printf("\n");
   }
   b=new float[m];
   for (i=0;i<m;i++)
   {
      b[i]=1;
      for (j=0;j<i+1;j++)
         b[i]*=a[i][j];
   }
   OutPuts("Произведения элементов строк матрицы ");
   for (i=0;i<m;i++)
      printf("%6.2f\n",b[i]);
   delete []b;
   for (i=0;i<m;i++)
      delete []a[i];
   delete []a;
   getch();
   return 0;
```

В программах, приведенных в примерах 2–4, обращение к элементам динамических массивов осуществлялось с помощью индексов. Можно использовать и другие способы адресации, одним из которых является использование смещения. Например, выделение памяти под каждую строку матрицы можно выполнить посредством смещения следующим образом:

```
   for (i=0;i<m;i++)
      *(a+i)=new float[n];
```

Пример 5. Здесь представлена программа для решения задачи, рассмотренной в примере программы 4, другим способом: с помощью дополнительного указателя вида

```
   float **p;
   for (p=a;p<a+m;p++)
      *p=new float[n];
```

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   int m,n,i,j;
   float **a, *b,**p;
   printRus("Введите количество строк и столбцов: ");
   scanf("%d %d",&m,&n);
   a=new float*[m];
   for (p=a;p<a+m;p++)
      *p=new float[n];
   OutPuts("Введите элементы матрицы ");
   for (i=0;i<m;i++)
      for (j=0;j<n;j++)
         scanf("%f",&a[i][j]);
   OutPuts("Вывод матрицы ");
   for (i=0;i<m;i++)
   {
      for (j=0;j<n;j++)
         printf("%8.2f",a[i][j]);
      printf("\n");
   }
   b=new float[m];
   for (i=0;i<m;i++)
   {
      b[i]=0;
      for (j=0;j<n;j++)
         b[i]+=a[i][j];
   }
```

```
    OutPuts("Суммы элементов строк матрицы ");
    for (i=0;i<m;i++)
       printf("%6.2f\n",b[i]);
    delete []b;
    for (i=0;i<m;i++)
       delete []a[i];
    delete []a;
    getch();
    return 0;
}
```

Доступ к элементам динамической матрицы

При обращении к переменной по ее имени компилятор обращается по адресу к памяти, отведенной для хранения значения переменной. Элементы массивов хранятся в смежных ячейках памяти, поэтому для вычисления адреса элемента одномерного массива достаточно знать адрес первого элемента (в языке С имя массива задает адрес первого элемента) и номер (индекс) элемента. При определении адреса элемента двумерного массива надо знать способы хранения элементов в памяти: элементы многомерных массивов могут храниться так, что быстрее всего возрастает первый индекс (хранение по столбцам) или быстрее всего возрастает последний индекс (хранение по строкам). В языке С используется второй способ — хранение по строкам. Для вычисления адреса элемента двумерного массива надо знать адрес первого элемента строки, в которой расположен этот элемент, и номер столбца, т. е. смещение этого элемента от начала строки. Поскольку элементы многомерных массивов хранятся в памяти также в смежных ячейках, то обращаться к элементам многомерных массивов можно как к элементам одномерного массива. Например, к элементу массива `a[i][j]` можно обратиться, указав два индекса или один индекс, при этом надо знать количество элементов в столбце: `a[i*nn+j]`, где `nn` — количество столбцов. Такую запись может использовать в своих программах сам программист, по этому же выражению ведет вычисление адресов и компилятор.

Доступ к элементам массивов можно осуществлять с помощью указателей и адресной арифметики. Как было сказано выше, адрес

i-й строки матрицы хранится в i-м элементе массива указателей, т. е. a[i] — это и есть адрес i-й строки матрицы. То же самое можно записать и по-другому: *(a+i). Данная запись может трактоваться как содержимое элемента массива, имеющего смещение i от начала массива, адрес первого элемента которого хранит переменная a.

Опираясь на адрес первого элемента строки, можно вычислять адрес и, следовательно, получать доступ к элементам этой строки. Доступ можно осуществить несколькими способами:

1) задать смещение относительно начала массива: *(*(a+i)+j), где *(a+i) — адрес первого элемента i-й строки матрицы; (*(a+i)+j) — адрес j-го элемента i-й строки матрицы; *(*(a+i)+j) — сам элемент, расположенный в j-м столбце i-й строки матрицы;

2) использовать текущий указатель на i-ю строку p=*(a+i); и затем обратиться к нужному элементу: *(p+j);. Разновидностью данного способа является доступ к адресу i-й строки по ее индексу: p=a[i]; *(p+j);;

3) задать индекс для доступа к i-й строке матрицы: *(a[i]+j);;

4) использовать наиболее привычный для начинающих программистов способ, предусматривающий указание обоих индексов: a[i][j].

Пример 6. Приведенная здесь программа поясняет применение рассмотренных способов получения доступа к элементам динамической матрицы. Сначала вычисляются элементы матрицы, затем они выводятся на монитор, при этом используются различные способы обращения к элементам матрицы.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   int m,n,i,j;
   float **a, *p, **pp,*pt;
   printRus("Введите количество строк и столбцов: ");
   scanf("%d %d",&m,&n);
```

```
   a=new float*[m];
   for (i=0;i<m;i++)
      a[i]=new float[n];
   //вычисление элементов матрицы
   for (i=0;i<m;i++)
      for (j=0;j<n;j++)
         a[i][j]=(i+1)*(j+1);
   //вывод матрицы:
   OutPuts("Вывод матрицы 1");
   //с использованием индексов
   for (i=0;i<m;i++)
   {
      for (j=0;j<n;j++)
         printf("%6.2f",a[i][j]);
      printf("\n");
   }
   //с использованием адресов со смещением
   OutPuts("Вывод матрицы 2");
   for (i=0;i<m;i++)
   {
      for (j=0;j<n;j++)
         printf("%6.2f",*(*(a+i)+j));
      printf("\n");
   }
   //с использованием указателя на текущую строку
   OutPuts("Вывод матрицы 3");
   for (i=0;i<m;i++)
   {
      p=*(a+i);//или p=a[i]+j;
      for (j=0;j<n;j++)
         printf("%6.2f",*(p+j));
      printf("\n");
   }
//с использованием указателя на текущую строку
и указателя на текущий элемент
   OutPuts("Вывод матрицы 4");
   for (pp=a;pp<a+m;pp++)
   {
      for (pt=*pp;pt<*pp+n;pt++)
         printf("%6.2f",*pt);
      printf("\n");
```

```
    }
    for (i=0;i<m;i++)
        delete []a[i];
    delete []a;
    getch();

    return 0;
}
```

Библиотечные функции C для работы с динамическими переменными и массивами

Для выделения памяти при работе с динамическими переменными и массивами предназначены стандартные функции **malloc** и **calloc**, для освобождения — функция **free**, для изменения размера ранее выделенной памяти — функция **realloc**. Для использования этих функций необходимо подключить заголовочный файл **<stdlib.h>**.

Функция **void *malloc(size_t size)** возвращает указатель на первый байт области памяти размером **size**, которая была выделена в динамически распределяемой области памяти. Если для удовлетворения запроса нет достаточного объема памяти, то возвращается нулевой указатель. Поэтому перед применением выделенной памяти всегда следует проверять полученный указатель на нулевое значение.

Пример использования функции **malloc**:

```
float *a;
if ((a=(float *)malloc(n))==NULL)
    printf("ошибка");
else
    for (int i=0;i<n;i++)
        scanf("%f",&a[i]);
```

Освобождение памяти, выделенной функцией **malloc**, осуществляется функцией **free**. Функция **void free(void *ptr)** возвращает в динамическую память блок памяти, адресуемый указателем **ptr**, после этого память становится доступной для выделения в дальнейшем. При передаче нулевого указателя функция никаких действий не выполняет. Функцию можно использовать с указателем, который был ранее получен с помощью одной из функций динамического распределения памяти. В противном слу-

чае произойдет ошибка управления памятью, что может привести к самым негативным последствиям.

Функция `void *calloc(size_t num, size_t size)` выделяет память, размер которой равен значению выражения `num*size`, т. е. память, достаточную для размещения массива, содержащего `num` элементов размером `size`. Все разряды выделенной памяти инициализируются нулями. Как и функция `malloc`, она возвращает указатель на первый байт выделенной области. Если для удовлетворения запроса нет достаточного объема памяти, то возвращается нулевой указатель. Поэтому перед использованием выделенной памяти всегда следует проверять полученный указатель на нулевое значение:

```
float *a;
if ((a=(float *) calloc(n, size(float)))==NULL)
   printf("ошибка");
else
   for (int i=0;i<n;i++)
      scanf("%f",&a[i]);
```

Функция `void *realloc(void *ptr, size_t size)` изменяет размер блока ранее выделенной памяти, адресуемой указателем `ptr` в соответствии с заданным размером `size`. Значение параметра `size` может быть больше или меньше, чем перераспределяемая область. Функция возвращает указатель на блок памяти, при этом не исключается перемещение блока памяти, например, при его увеличении, а содержимое старого блока копируется в новый блок. Если указатель `ptr` имеет нулевое значение, то функция просто выделяет требуемый объем памяти и возвращает указатель на этот блок памяти. Если значение параметра `size` равно нулю, то память, адресуемая параметром `ptr`, освобождается.

Если в динамически распределяемой области памяти нет достаточного объема свободной памяти, то возвращается нулевой указатель, а исходный блок памяти остается неизменным.

Пример 7. Приведенная в этом примере программа демонстрирует использование функции `malloc` при работе с динамическим одномерным массивом.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
```

```
#include "stdlib.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   float *a;
   int n;
   printRus("Введите количество элементов массива: ");
   scanf("%d",&n);
   a=(float *)malloc(n*sizeof (float));
   if (!a)
      OutPuts("\nОшибка!");
   else
   {
      printRus("Введите элементы массива: ");
      for(int i=0;i<n;i++)
         scanf("%f",&a[i]);
      OutPuts("Вывод массива ");
      for(int i=0;i<n;i++)
         printf("%5.1f ",a[i]);
      printf("\n");
      free(a);
   }
   getch();
   return 0;
}
```

Пример 8. Приведенная здесь программа демонстрирует использование функции `calloc`. В этой программе вводятся начальное значение аргумента, шаг его изменения и количество точек, в которых вычисляется значение функции $y = x^2$. Затем запрашивается дополнительное количество точек и вычисляются значения функции в последующих точках. При этом необходимо изменить размерность массива, что и выполняется с использованием функции `calloc`. Однако надо иметь в виду, что, выделяя новый участок памяти, функция не копирует содержимое старого участка в новый массив. В связи с этим приходится вводить еще один указатель, который хранит адрес первого элемента старого массива. После выделения нового участка памяти осуществляется копирование содержимого старого массива в новую область памяти, а потом вычисляются дополнительные значения функции.

Текст программы:

```
#include "stdafx.h"
#include "conio.h"
#include "stdlib.h"
#include "D:\\temp\\RusString.h"

int _tmain(int argc, _TCHAR* argv[])
{
   float *y, *p;
   int n;
   OutPuts("Введите количество точек");
   scanf("%d",&n);
   y=(float *)
      calloc(n,sizeof (float));
   if (!y)
      OutPuts("Ошибка");
   else
   {
      float x,xn,dx;
      int i,nd;
      OutPuts("Введите начальное x и шаг dx");
      scanf("%f %f",&xn,&dx);
      for (i=0;i<n;i++)
      {
         x=xn+i*dx;
         y[i]=x*x;
      }
      OutPuts("Вывод массива y:");
      for(int i=0;i<n;i++)
         printf("%6.2f ",y[i]);
      OutPuts("\nВведите дополнительное количество точек");
         scanf("%d",&nd);
      p=y;
      y=(float *)
         calloc(n+nd,sizeof (float));
      for (i=0;i<n;i++)
         y[i]=p[i];
      for (i=n;i<n+nd;i++)
      {
         x=xn+i*dx;
         y[i]=x*x;
      }
```

```
        OutPuts("Вывод массива y:");
        for(int i=0;i<n+nd;i++)
            printf("%6.2f ",y[i]);
        free(y);
    }
    getch();
    return 0;
}
```

Если использовать функцию **realloc**, то не потребуется копировать ранее вычисленные значения функции в новую область памяти. В этом случае программа будет выглядеть следующим образом:

```
int _tmain(int argc, _TCHAR* argv[])
{
    float *y, *p;
    int n;
    OutPuts("Введите количество точек");
    scanf("%d",&n);
    y=(float *)
        calloc(n,sizeof (float));
    if (!y)
        OutPuts("Ошибка");
    else
    {
        float x,xn,dx;
        int i,nd;
        OutPuts("Введите начальное x и шаг dx");
        scanf("%f %f",&xn,&dx);
        for (i=0;i<n;i++)
        {
            x=xn+i*dx;
            y[i]=x*x;
        }
        OutPuts("Вывод массива y:");
        for(int i=0;i<n;i++)
            printf("%6.2f ",y[i]);
        OutPuts("\nВведите дополнительное количество точек");
            scanf("%d",&nd);

        y=(float *)realloc(y,(n+nd)*sizeof (float));
        for (i=n;i<n+nd;i++)
```

```
        {x=xn+i*dx;
        y[i]=x*x;
        }
     OutPuts("Вывод массива y:");
      for(int i=0;i<n+nd;i++)
       printf("%6.2f ",y[i]);
       free(y);
 }
 getch();
 return 0;
}
```

Передача в функции динамических массивов

При передаче из подпрограмм в качестве параметров динамических массивов, являющихся результатами, приходится использовать указатели на указатели (две звездочки) при работе с одномерными массивами или даже указатель на указатель на указатель (три звездочки) при работе с двумерными массивами. Это связано с тем, что при передаче параметров по указателю (адресу) необходимо передавать именно адрес передаваемого объекта. Поскольку динамический одномерный массив определяется с помощью указателя, то при передаче в функцию необходимо передавать адрес этого указателя. Соответственно, если динамическая матрица определяется как указатель на указатель, то передача параметра представляет собой передачу указателя на указатель массива, который хранит указатели на строки матрицы.

Пример 9. В приведенной здесь программе используются функция ввода исходных данных (количества элементов и самих элементов динамического массива), функция вывода элементов массива, а также функция преобразования элементов массива путем умножения каждого элемента на отношение максимального элемента к минимальному элементу. Выделение динамической памяти для массива осуществляется в функции ввода исходных данных.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
```

```
#include "D:\\temp\\RusString.h"

void input(float **a,int *n);
void output(float *a,int n);
void preob(float *a,int n);

int _tmain(int argc, _TCHAR* argv[])
{
   float *a;
   int n;
   OutPuts("Ввод исходных данных");
   input(&a,&n);
   OutPuts("Вывод исходного массива");
   output(a,n);
   //преобразования массива
   preob(a,n);
   OutPuts("\nВывод преобразованного массива");
   output(a,n);
   getch();
   delete a;
   return 0;
}

void input(float **a,int *n)
{//Введите количество элементов
 //и значения самих элементов динамического массива
   OutPuts("Введите количество элементов массива");
   scanf("%d",n);
   *a=new float[*n];
   OutPuts("Введите элементы массива");
   for(int i=0;i<*n;i++)
      scanf("%f",&(*a)[i]);
}

void output(float *a,int n)
{//функция вывода одномерного массива
   for (int i=0;i<n;i++)
      printf("%5.1f ",a[i]);
}
void preob(float *a,int n)
{//функция преобразования одномерного массива
   float max,min;
```

```
   max=a[0]; min=a[0];
   for (int i=1;i<n;i++)
      if (a[i]>max)
         max=a[i];
      else
         if(a[i]<min)
            min=a[i];
   float y=max/min;
   for (int i=0;i<n;i++)
      a[i]*=y;
}
/* Протокол ввода - вывода

Ввод исходных данных
Введите количество элементов массива
4
Введите элементы массива
3 1 5 4
Вывод исходного массива
 3.0 1.0 5.0 4.0
Вывод преобразованного массива
 15.0 5.025.020.0
*/
```

Пример 10. Приведенная в этом примере программа иллюстрирует работу с динамическими матрицами, имеющими треугольный вид. Здесь так же организованы функции ввода исходных данных, вывода матрицы и обработки матриц. Обработка матриц заключается в вычитании двух матриц и формировании результата, представляющего собой третью треугольную матрицу.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "D:\\temp\\RusString.h"
void input(float ***a,int *n);
void output(float **a,int n);
void preob(float **a,float **b,float ***c,int n);

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    float **a,**b,**c;
    int n,m;
    OutPuts("Ввод матрицы a");
    input(&a,&n);
    OutPuts("Ввод матрицы b");
    input(&b,&m);
    OutPuts("Вывод матрицы a");
    output(a,n);
    OutPuts("Вывод матрицы b");
    output(b,m);
    if (n==m)
    {
       preob(a,b,&c,n);
       OutPuts("Вывод матрицы c");
       output(c,n);
    }
    getch();
    return 0;
}

void input(float ***a,int *n)
{//функция ввода исходных данных
    OutPuts("Введите количество строк матрицы");
    scanf("%d",n);
    *a=new float *[*n];
    for (int i=0;i<*n;i++)
       (*a)[i]=new float [i+1];
    OutPuts("Введите элементы матрицы");
    for(int i=0;i<*n;i++)
       for (int j=0; j<=i;j++)
          scanf("%f",&(*a)[i][j]);
}

void output(float **a,int n)
{//функция вывода матрицы
    for (int i=0;i<n;i++)
    {
       for (int j=0; j<=i;j++)
          printf("%5.1f ",a[i][j]);
       printf("\n");
    }
}
```

```
void preob(float **a,float **b,float ***c,int n)
{//функция нахождения разности двух матриц
   *c=new float *[n];
   for (int i=0;i<n;i++)
      (*c)[i]=new float [i+1];
   for (int i=0;i<n;i++)
      for (int j=0; j<=i;j++)
         (*c)[i][j]=a[i][j]-b[i][j];
}
/* Протокол ввода - вывода

3
Введите элементы матрицы
1
1 1
1 1 1
Ввод матрицы b
Введите количество строк матрицы
3
Введите элементы матрицы
2
2 2
2 2 2
Вывод матрицы a
 1.0
 1.0 1.0
 1.0 1.0 1.0
Вывод матрицы b
 2.0
 2.0 2.0
 2.0 2.0 2.0
Вывод матрицы c
 -1.0
 -1.0-1.0
 -1.0-1.0-1.0
 */
```

Вопросы для самопроверки

1. Что представляет собой переменная-указатель?
2. Какие виды указателей имеется в составе языка С?
3. Как объявляются указатели?

4. Как получить доступ к содержимому памяти, адрес которой хранится в указателе?

5. Совместимы ли между собой указатели на разные типы данных?

6. Как осуществить преобразование указателя на один тип данных в указатель на другой тип данных?

7. Как объявляются указатели на константу?

8. Как объявляются указатели-константы?

9. Можно ли осуществлять ввод и вывод указателей?

10. Какие операции можно выполнять с указателями?

11. Что означает прибавление к указателю константы?

12. В чем состоит взаимосвязь указателей и массивов в языке С?

13. Что такое динамическая переменная?

14. Как осуществляется выделение памяти для хранения динамических переменных? Как выполняется освобождение памяти?

15. Как выделить и освободить память для одномерного динамического массива, для двумерного динамического массива?

16. Какую форму могут иметь динамические двумерные массивы?

17. Как выполняется доступ к элементам динамических двумерных массивов?

18. Какие стандартные функции используются при работе с динамическими массивами?

19. Как передаются в функцию динамические одномерные и двумерные массивы?

6. Динамические структуры данных

Динамические структуры данных — это совокупности элементов, представляющих данные и связи между ними типа отношений предшествования /следования. Элементы реализуются динамически создаваемыми структурами с полями, содержащими «полезные» данные любого типа (основного, составного или типа указатель), и полями-указателями, организующими связи между элементами. Динамические структуры данных создаются после старта программы с применением средств динамического выделения памяти. Состав их элементов может изменяться в процессе выполнения программы, а после использования эти элементы удаляются, выделенная для них память освобождается. Для доступа к динамической структуре в программе должны быть одна или несколько переменных типа указатель на элементы структуры, а для переходов от одного элемента к другому используются связи между ними. Динамические структуры данных применяются для таких объектов, как различные последовательности (линейные и циклические, односвязные и многосвязные списки), иерархические структуры (деревья) и сети (графы). Далее в этом разделе будут рассмотрены работы с некоторыми видами списков и бинарными деревьями.

В процессе работы с динамическими структурами данных память выделяется во время выполнения программы. Данные могут создаваться не одномоментно, а в течение некоторого времени в процессе реализации некоторого алгоритма, причем сами данные должны сохранять связь между собой. Для решения подобной задачи служат динамические структуры, к числу которых относятся линейные списки и бинарные деревья.

Элемент любой динамической структуры данных представляет собой структуру (как тип данных). Поля структуры можно разбить на две группы: поля первой группы, хранящие полезную информацию, и поля второй группы, хранящие адрес (адреса) элементов, с которыми имеется связь. Можно сказать, что поля

второй группы в некотором смысле имеют служебное назначение. Поля данных могут быть любого типа: основного, составного или типа указатель.

6.1. Списки

Линейный список

Линейный список представляет собой динамическую структуру данных, в которой каждый элемент содержит ссылку на следующий элемент. Такой список называют однонаправленным, или односвязным, поскольку каждый элемент связан с другим элементом только одной связью (каждый элемент хранит только один указатель на следующий элемент).

Двунаправленный список отличается от однонаправленного тем, что каждый элемент списка хранит две связи (два указателя): на следующий элемент списка и на предыдущий элемент.

На практике используются и кольцевые списки, в которых имеется связь между последним и первым элементами.

Можно рассматривать и *N*-связные списки, в которых элементы связаны с несколькими (*N*) другими элементами списка.

Над списками принято выполнять следующие операции: начальное формирование списка (создание первого элемента), добавление элемента в конец списка, чтение данных из элемента списка, вставка элемента в заданное место списка (до или после заданного элемента), удаление заданного элемента списка, упорядочение элементов списка по содержащимся в них данным.

В зависимости от того, какие из перечисленных операций допустимо выполнять, различают также стеки и очереди. Стек представляет собой односвязный список, для которого разрешено добавлять или удалять элементы только с одного конца списка, называемого вершиной стека.

Очередь также представляет собой односвязный список, для которого разрешено только два действия: добавление элемента в конец (хвост) списка (очереди) и удаление элемента только из начала (головы) списка.

Пример работы с однонаправленным списком

Рассмотрим сначала организацию однонаправленных списков. Каждый элемент такого списка помимо полей, хранящих полезную

информацию, имеет одно поле, хранящее адрес соседнего элемента. Для осуществления доступа к элементам списка необходимо также хранить в некоторой переменной (указателе) адрес первого элемента списка.

При работе с однонаправленным списком можно отметить, что операции добавления нового элемента в пустой список или непустой список ничем не различаются. В связи с этим имеется одна функция добавления нового элемента в список. В этом случае выделяется память для хранения нового элемента, затем добавляемое значение заносится в поле **inf**, а адрес предыдущего элемента — в поле **adr**.

Рассмотрим этот процесс по шагам, используя рис. 6.1–6.4, где элементы списка представлены парами прямоугольников, соответствующими полям структуры с именами **inf** (информационное поле) и **adr** (поле типа «указатель на элементы списка»). Отдельными прямоугольниками представлены переменные **nach** (указатель на хвост списка) и **t** (вспомогательная переменная — указатель на добавляемый в список новый элемент); стрелки вида ●⟶ соответствуют связям объектов по указателям; фигуры ●—● — значению **NULL** указателя; цифры в прямоугольниках означают содержимое поля.

Исходное состояние списка показано на рис. 6.1.

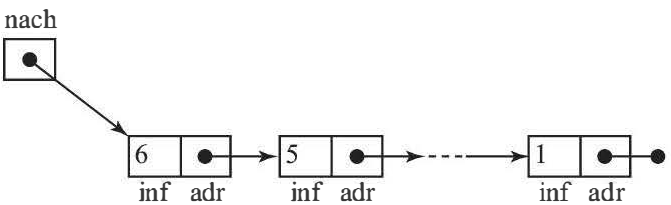

Рис. 6.1

Шаг 1. Создание нового элемента с сохранением его адреса в переменной **t** и ввод в его поле значения 7 (см. рис. 6.2).

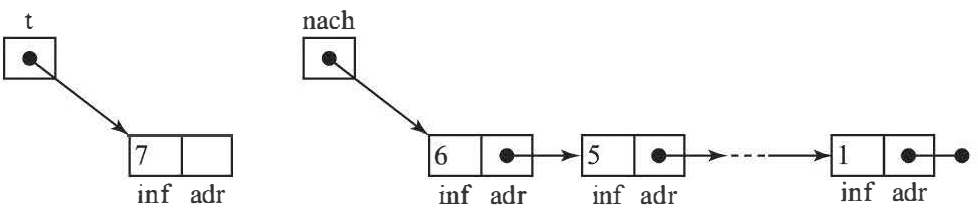

Рис. 6.2

Шаг 2. Создание связи от нового элемента с хвостовым элементом исходного списка копированием адреса из переменной **nach** в поле **adr** нового элемента (см. рис. 6.3).

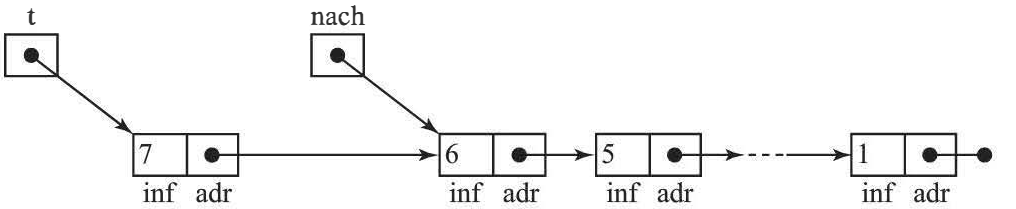

Рис. 6.3

Шаг 3. Создание связи от переменной **nach** к новому элементу копированием адреса из переменной **t** в переменную **nach**, теперь хвостовым элементом списка становится новый элемент (см. рис. 6.4).

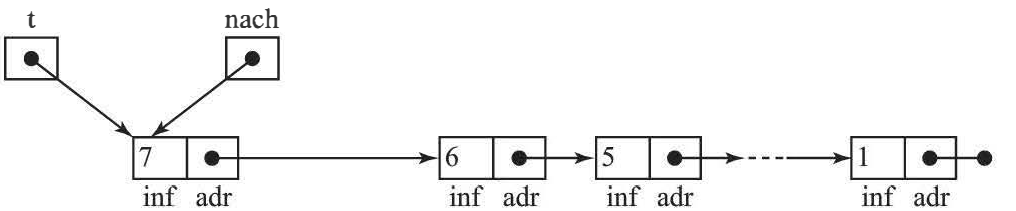

Рис. 6.4

Просмотр содержимого списка реализован в цикле. Переменная **t**, хранящая адрес очередного элемента списка, получает последовательно в ходе выполнения цикла значения адресов всех просматриваемых элементов списка, пока адрес не станет нулевым. При этом выводятся значения просматриваемых элементов.

Определение длины списка выполняется аналогично просмотру содержимого списка, только внутри цикла производится не вывод данных из просмотренных элементов, а подсчет количества элементов.

Поиск заданного элемента в списке основан также на просмотре содержимого списка. Цикл просмотра выполняется, пока нужный элемент не будет найден (предполагается, что искомый элемент присутствует в списке в одном экземпляре) и пока не будет достигнут конец списка. Функция поиска в качестве результата возвращает адрес найденного элемента (если элемент не будет

найден, возвращается адрес первого элемента списка). Кроме того, функция в качестве параметров возвращает номер найденного элемента. Если заданный элемент не будет найден, то возвратится нулевое значение, а также признак вхождения искомого элемента в список (найден, не найден).

Вставка элемента перед заданным элементом списка предполагает сначала поиск заданного элемента в списке, так как вставка возможна только в том случае, если заданный элемент присутствует в списке. Если заданный элемент найден, то выделяется память для хранения нового (добавляемого) элемента в список. В поле `inf` заносится добавляемое значение, в адресное поле — адрес впереди стоящего элемента (этот адрес хранит найденный элемент списка), а в адресную часть найденного элемента — адрес добавленного элемента.

Вставка элемента после заданного элемента списка также начинается с поиска заданного элемента в списке. Вставка нового элемента после заданного, на первый взгляд, может вызвать определенные проблемы, так как список однонаправленный и нет возможности, зная адрес текущего элемента, получить доступ к предыдущему элементу. Однако задачу вставки нового элемента после заданного легко свести к предыдущей задаче, т. е. к вставке нового элемента перед заданным. Для этого выделяется память для хранения нового элемента. Полям нового элемента присваиваются значения полей заданного элемента, а в поле `inf` заданного элемента заносится добавляемое значение.

При удалении заданного элемента следует рассмотреть три возможных варианта расположения заданного элемента: 1) в начале списка (начальный, первый элемент); 2) в конце списка (конечный, последний элемент); 3) в середине списка — срединный элемент (он не является ни первым, ни последним).

В первом варианте достаточно изменить значение переменной nach, хранящей адрес первого элемента списка. При удалении последнего элемента списка надо предварительно получить доступ к предпоследнему элементу списка, а затем изменить значение адресной части этого элемента, присвоив ей значение `NULL`.

При удалении срединного элемента списка следует поступить так же, как и при добавлении нового элемента после заданного элемента. Полям найденного (формально удаляемого) элемента

присваиваются значения полей следующего элемента, стоящего перед формально удаляемым, а затем удаляется следующий элемент, стоящий перед формально удаляемым. Как и при добавлении в список, здесь сначала осуществляется поиск заданного (удаляемого) элемента в списке. Поиск позволяет не только убедиться в наличии (или отсутствии) нужного элемента, но и определить его номер.

Для удобства взаимодействия пользователя с программой предусмотрено меню, позволяющее выбрать одну из описанных операций.

Представленная в примере программа демонстрирует выполнение основных операций при работе со списком: добавление нового элемента в список, просмотр содержимого списка, определение длины (количества элементов), поиск заданного элемента в списке, добавление нового элемента перед заданным элементом, добавление нового элемента после заданного элемента, удаление заданного элемента. В этой программе известен фактически адрес не первого, а последнего элемента списка, который хранится в переменной с именем **nach**.

В программе использованы те же имена, что и на рис. 6.1–6.4.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include <math.h>
#include "D:\\temp\\RusString.h"

//typedef ukaz *elem;
typedef struct elem
{
   int inf;
   elem *adr;
};

void  add(elem **nach); // добавление элемента в список
void prosm(elem *nach); //просмотр содержимого списка
int dlina(elem *nach); //определение длины списка
elem* poisk(int b,elem *nach, bool *pr,int *n);//поиск
                                  //заданного элемента в списке
bool wstawkapered(elem *nach); //вставка нового
                                  //элемента перед заданным
```

```
bool wstawkaposle(elem *nach); //вставка нового
                               //элемента после заданного
elem* udalenie(elem *nach,bool *pr); //удаление
                                     //заданного элемента
void menu();

int _tmain(int argc, _TCHAR* argv[])
{
   elem *nach, *q;
   int b,n;
   bool pr;
   char ch;
   nach=NULL;
   do
   {
      menu();
      ch=getch();
      switch (ch)
      {
         case '1':add(&nach);
            break;
         case '2':if (nach==NULL)
                     OutPuts("Список пуст");
                  else
                     prosm(nach);
            break;
         case '3':printRus("Длина списка = ");
            printf("%d\n",dlina(nach));
            break;
         case '4':printRus("Введите значение для поиска: ");
            scanf("%d",&b);
            q=poisk(b,nach,&pr,&n);
            if (pr)
            {
               printRus("Номер элемента ");
               printf("%d\n",n);
               printRus("Адрес элемента ");
               printf("%p\n",q);
            }
            else
               OutPuts("Нет элемента в списке ");
            break;
         case '5':if (wstawkapered(nach))
                  OutPuts("Элемент вставлен ");
```

```
            else
               OutPuts("Вставка невозможна ");
            break;
         case '6':if (wstawkaposle(nach))
               OutPuts("Элемент вставлен ");
            else
               OutPuts("Вставка невозможна ");
            break;
         case '7':nach=udalenie(nach,&pr);
            if (pr)
               OutPuts("\nЭлемент удален");
            else
               OutPuts("\nУдаление невозможно ");
            break;
         case '8':
            break;
         default:{}
      }
   }while (ch!='8');
   return 0;
}

void add(elem **nach)//elem*
{// добавление элемента в список
   elem *t;
   int a;
   printRus(
      "Введите число для добавляемого элемента: ");
   scanf("%d",&a);
   t=new(elem);
   t->inf=a;
   t->adr=*nach;
   *nach=t;
}

void prosm(elem *nach)
{//просмотр содержимого списка
   elem *t;
   t=nach;
   while (t!=NULL)
   {
      printf("t->inf= %d\n",t->inf);
      t=t->adr;
   }
}
```

```
int dlina(elem *nach)
{//определение длины списка
   int dl=0;
   elem* t;
   t=nach;
   while (t!=NULL)
   {
      dl++;
      t=t->adr;
   }
   return dl;
}
elem* poisk(int b,elem *nach, bool *pr,int *n)
{//поиск заданного элемента в списке
   elem* t,*q;
   (*n)=0;
   t=nach;
   q=t;
   *pr=false;
   while ((t!=NULL)&&(!*pr))
   {
      (*n)++;
      if (t->inf==b)
      {
         *pr=true;
         q=t;//printf("\n adres w poiske %p",q);
      }
      else
         t=t->adr;
   }
   if (!*pr) *n=0;
   return q;
}
bool wstawkapered(elem *nach)
{//вставка заданного элемента перед заданным
   int pris,dob,n;
   bool pr;
   elem *t,*q;
   OutPuts("Введите значение того элемента,");
   printRus(
      "перед которым нужно вставить новый элемент: ");
```

```
    scanf("%d",&pris);
    q=poisk(pris,nach,&pr,&n);
    if (pr)
    {
        t=new(elem);
        printRus(
            "Введите значение добавляемого элемента: ");
        scanf("%d",&dob);
        t->inf=dob;
        t->adr=q->adr;
        q->adr=t;
    }
    return pr;
}
bool wstawkaposle(elem *nach)
{//вставка заданного элемента после заданного
    int pris,dob,n;
    bool pr;
    elem *t,*q;
    OutPuts("Введите значение того элемента,");
    printRus(
        "после которого нужно вставить новый элемент: ");
    scanf("%d",&pris);
    q=poisk(pris,nach,&pr,&n);
    if (pr)
    {
        t=new(elem);
        printRus(
            "Введите значение добавляемого элемента: ");
        scanf("%d",&dob);
        t->inf=q->inf;
        t->adr=q->adr;
        q->adr=t;
        q->inf=dob;
    }
    return pr;
}
elem* udalenie(elem *nach,bool *pr)
{//удаление заданного элемента
    int pris,n,dl,i;
    elem *t,*q;
    printRus("Введите значение удаляемого элемента: ");
```

```
   scanf("%d",&pris);
   q=poisk(pris,nach,pr,&n);
   printRus("Удаляется элемент ");
   printf("n=%d, pr=%d\n",n,*pr);
   if (*pr)
   {
      dl=dlina(nach);
      if (n==1) //удаление первого элемента
      {
         t=q;
         printRus(
"Удаляется первый элемент с позиции ");
         printf("%d",n);
         nach=(nach)->adr;
         delete t;
      }
      else
      {
         if (n==dl) //удаление последнего элемента
         {
            t=nach;
            printRus(
"Удаляется последний элемент с позиции ");
            printf("%d",n);
            for (i=0;i<n-2;i++)
               t=t->adr;
            t->adr=NULL;
            delete q;
         }
         else //удаление срединного элемента
         {
            printRus(
"Удаляется срединный элемент с позиции ");
            printf("%d",n);
            t=q->adr;
            q->adr=t->adr;
            q->inf=t->inf;
            delete t;
         }
      }
   }
   return nach;
}
```

```
void menu()
{
   OutPuts("\n\tВыберите пункт меню \n");
   OutPuts("1 Добавить элемент");
   OutPuts("2 Просмотр списка");
   OutPuts("3 Длина списка");
   OutPuts("4 Поиск элемента");
   OutPuts("5 Добавить перед ");
   OutPuts("6 Добавить после ");
   OutPuts("7 Удалить элемент");
   OutPuts("8 Выход");
}
```

Пример работы с двунаправленным списком

Двунаправленные списки позволяют упростить работу и обеспечить просмотр элементов списка в обоих направлениях. Элементы такого списка имеют по два поля-указателя, одно из которых **Ladr** содержит адрес соседнего слева элемента (рис. 6.5) или **NULL**, если соседнего слева нет, а другое поле **Radr** — адрес соседнего справа элемента или **NULL**, если соседнего справа нет. При работе с двунаправленными списками для доступа к элементам обычно используют два указателя: **nach** хранит адрес первого элемента, а **kon** — адрес последнего элемента. В этом случае начало и конец становятся равноправными, определение начального и последнего элементов является условным. На рис. 6.5 стрелки вида ●⟶ соответствуют связям объектов по указателям; фигуры ●—● — значению **NULL** указателя; цифры 1–3 в прямоугольниках означают содержимое поля; **nach** и **inf** — поля структуры.

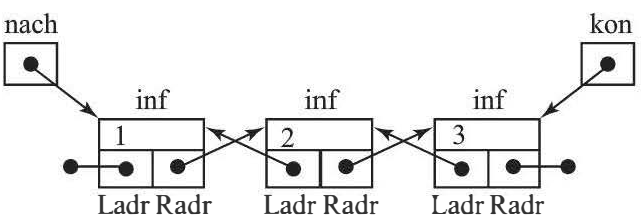

Рис. 6.5

Добавление нового элемента в конец двунаправленного списка отличается от такой же операции для однонаправленного списка. Для добавления нового элемента в двунаправленный список вы-

полняются разные действия при добавлении первого (в пустой список) и непервого (в непустой список) элементов. При добавлении первого элемента указатели на соседние элементы (поля **Ladr** и **Radr**) получают нулевое значение, а указатели на первый и последний (**nach** и **kon**) элементы — одинаковые значения, т. е. адрес первого (единственного на этот момент) элемента. При добавлении не первого элемента в двунаправленный список указатель (поле **Radr**) на последующий элемент получает нулевое значение, а указатель на предшествующий элемент (поле **Ladr**) — адрес предшествующего, добавленного в список перед этим элемента (адрес из переменной **kon**). При этом значение указателя **nach** на первый элемент не изменяется, а изменяется значение указателя **kon** на последний добавляемый элемент.

Приведенная в примере программа позволяет выполнять такие операции, как добавление нового элемента в конец существующего списка, просмотр содержимого списка, определение длины списка, поиск заданного элемента в списке, вставка нового элемента после заданного элемента или перед заданным элементом, удаление заданного элемента из списка.

Текст программы:

```
#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include <math.h>
#include "D:\\temp\\RusString.h"

typedef struct elem
{
   int inf;
   elem *Ladr;
   elem *Radr;
};
void add(elem **nach, elem **kon);// добавление
                                  //элемента в список
void prosm(elem *nach);//просмотр содержимого списка
int dlina(elem *nach);//определение длины списка

elem* poisk(int b,elem *nach, bool *pr,
int *n);//поиск заданного элемента в списке
```

```
bool wstawkapered(elem **nach);//вставка нового
                                //элемента перед заданным
bool wstawkaposle(elem *nach,elem **kon);//вставка
                          //нового элемента после заданного
bool udalenie(elem **nach,elem **kon);//удаление
                                      //заданного элемента
void menu();

int _tmain(int argc, _TCHAR* argv[])
{
   elem *nach,*kon,*q;
   int b,n;
   bool pr;
   int ch;
   nach=NULL;
   kon=NULL;
   do
   {
      menu();
      scanf("%d",&ch);
      switch (ch)
      {
         case 1:add(&nach,&kon);
            break;
         case 2:if (nach==NULL)
                   OutPuts("Список пуст");
            else
               prosm(nach);
            break;
         case 3:printRus("Длина списка = ");
                printf("%d\n",dlina(nach));
            break;
         case 4:printRus("Введите значение для поиска: ");
            scanf("%d",&b);
            q=poisk(b,nach,&pr,&n);
            if (pr)
            {
               printRus("Номер элемента ");
               printf("%d\n",n);
               printRus("Адрес элемента ");
               printf("%p\n",q);
            }
            else
```

```
                OutPuts("Нет элемента в списке ");
             break;
          case 5:if (wstawkapered(&nach))
                OutPuts("Элемент вставлен ");
             else
                OutPuts("Вставка невозможна ");
             break;
          case 6: if (wstawkaposle(nach,&kon))
                OutPuts("Элемент вставлен ");
             else
                OutPuts("Вставка невозможна ");
             break;
          case 7: if (udalenie(&nach,&kon))
                OutPuts("\nЭлемент удален");
             else
                OutPuts("\nУдаление невозможно ");
             break;
          case 8:
             break;
          default:{}
       }
    }while (ch!=8);
    return 0;
}

void add(elem **nach, elem **kon)
{// добавление элемента в список
    int a;
    elem *t;
    printRus("Введите значение элемента: ");
    scanf("%d",&a);
    t=new(elem);
    t->inf=a;
    if (*nach==NULL)
    {
       t->Ladr=NULL;
       t->Radr=NULL;
       *nach=t;
       *kon=t;
    }
    else
    {
       t->Radr=NULL;
```

```
        t->Ladr=*kon;
        (*kon)->Radr=t;
        *kon=t;
    }
}

void prosm(elem *nach)
{ //просмотр содержимого списка
    elem *t;
    t=nach;
    while (t!=NULL)
    {
        printf("t->inf= %d\n",t->inf);
        t=t->Radr;
    }
}

int dlina(elem *nach)
{//определение длины списка
    int dl=0;
    elem* t;
    t=nach;
    while (t!=NULL)
    {
        dl++;
        t=t->Radr;
    }
    return dl;
}

elem* poisk(int b,elem *nach, bool *pr,int *n)
{//поиск заданного элемента в списке
    elem* t,*q;
    *n=0;
    t=nach;
    q=t;
    *pr=false;
    while ((t!=NULL)&&(!*pr))
    {
        (*n)++;
        if (t->inf==b)
        {
            *pr=true;
```

```
            q=t;
        }
        else
            t=t->Radr;
    }
    if (!*pr)
        *n=0;
    return q;
 }

bool wstawkapered(elem **nach)
 {//вставка заданного элемента перед заданным (левее, т. е.
ближе к началу списка)
    int pris,dob,n;
    bool pr;
    elem *t,*q;
    OutPuts("Введите значение того элемента,");
    printRus("перед которым нужно вставить новый элемент: ");
    scanf("%d",&pris);
    q=poisk(pris,*nach,&pr,&n);
    if (pr)
    {
        elem*t=new(elem);
        printRus("Введите значение добавляемого элемента: ");
        scanf("%d",&dob);
        t->inf=dob;
        if (n>1)
        {
            t->Radr=q;
            t->Ladr=q->Ladr;
            q->Ladr=t;
            t->Ladr->Radr=t;
        }
        if(n==1)
        {
            t->Radr=q;
            t->Ladr=q->Ladr;
            q->Ladr=t;
            *nach=t;
        }
    }
    return pr;
 }
```

```
bool wstawkaposle(elem *nach,elem **kon)
 {//вставка заданного элемента после заданного
   int pris,dob,n;
   bool pr;
   elem *t,*q;
   OutPuts("Введите значение того элемента,");
   printRus("после которого нужно вставить новый элемент: ");
   scanf("%d",&pris);
   q=poisk(pris,nach,&pr,&n);
   if (pr)
   {
      elem*t=new(elem);
      printRus("Введите значение добавляемого элемента: ");
      scanf("%d",&dob);
      t->inf=dob;
      if (n<dlina(nach)) //q!=kon
      {
         t->Radr=q->Radr;
         t->Ladr=q;
         q->Radr=t;
         t->Radr->Ladr=t;
      }
      if (n==dlina(nach)) //q=kon
      {
         t->Ladr=q;
         q->Radr=t;
         t->Radr=NULL;//q->Radr;
         *kon=t;
      }
   }
   return pr;
 }

bool udalenie(elem **nach,elem **kon)
{//удаление заданного элемента
   int pris,n,dl,i;
   elem *t,*q;
   bool pr;
   printRus("Введите значение удаляемого элемента: ");
   scanf("%d",&pris);
   q=poisk(pris,*nach,&pr,&n);
   if (pr)
```

```
    {
        if (q==*nach) //удаление первого элемента
        {
            t=q;
            *nach=(*nach)->Radr;//*nach=t->Radr;
            if (*nach==NULL)
                *kon=NULL; //удаление единственного элемента
            else
                (*nach)->Ladr=NULL;
            delete q;
        }
        else
        if (q==*kon) //удаление последнего элемента
        {
            t=q;
            t->Ladr->Radr=NULL;
            *kon=t->Ladr;
            delete q;
        }
        else //удаление срединного элемента
        {
            t=q->Ladr;
            t->Radr=q->Radr;
            q->Radr->Ladr=q->Ladr;
            delete q;
        }
    }
    return pr;
 }

void menu()
{
    OutPuts("\n\tВыберите пункт меню \n");
    OutPuts("1 Добавить элемент");
    OutPuts("2 Просмотр списка");
    OutPuts("3 Длина списка");
    OutPuts("4 Поиск элемента");
    OutPuts("5 Добавить перед");
    OutPuts("6 Добавить после");
    OutPuts("7 Удалить элемент");
    OutPuts("8 Выход");
}
```

6.2. Задания на обработку списков

1. Создать двунаправленный список, хранящий произвольные действительные числа. Обеспечить просмотр элементов списка, добавление новых элементов. Удалить из списка все отрицательные элементы.

2. Из файла с целыми числами прочитать эти числа и создать двунаправленный список, занося в него эти числа по возрастанию непосредственно при формировании списка. Вывести данные из полученного списка, а затем повторяющиеся элементы оставить в одном экземпляре и вывести содержимое преобразованного списка.

3. Составить однонаправленный список, хранящий информацию о многочлене *n*-го порядка. Элемент списка хранит коэффициент очередного слагаемого и показатель степени этого слагаемого. Если коэффициент равен нулю, то соответствующего элемента в списке быть не должно. Составить подпрограммы: 1) проверки двух многочленов на равенство, заданных в виде списков; 2) сложения двух многочленов, заданных в виде списков, и представления результата в виде списка.

4. Составить однонаправленный список, хранящий информацию о многочлене *n*-го порядка. Элемент списка хранит коэффициент очередного слагаемого и показатель степени этого слагаемого. Если коэффициент равен нулю, то соответствующего элемента в списке быть не должно. Составить подпрограммы: 1) вычисления значения многочлена в заданной точке; 2) формирования списка с информацией о многочлене, являющимся производной исходного многочлена.

5. Составить однонаправленный список, хранящий информацию о многочлене *n*-го порядка. Элемент списка хранит коэффициент очередного слагаемого и показатель степени этого слагаемого. Если коэффициент равен нулю, то соответствующего элемента в списке быть не должно. Составить подпрограммы: 1) вычисления значения многочлена в заданной точке; 2) формирования списка с информацией о многочлене, являющимся первообразной от исходного многочлена. Вычислить значение определенного интеграла в заданных пределах.

6. Просматривая файл с символами, создать двунаправленный список, содержащий эти символы. Удалить из списка все цифро-

вые символы, расположенные перед заданной буквой. Вывести содержимое исходного и полученного списков.

7. Просматривая файл с символами, создать двунаправленный список, содержащий эти символы. Добавить в список перед каждой последовательностью и после каждой последовательности цифровых символов фигурные скобки. Вывести содержимое исходного и полученного списков.

8. Ненулевые элементы одномерного массива записать в однонаправленный список. Элемент списка хранит значение элемента и его индекс. Создать два списка для элементов двух массивов. На их основе сформировать третий список, элементы которого представляют собой попарные произведения одноименных компонентов исходных массивов. Вывести содержимое исходных списков и полученного.

9. Ненулевые элементы одномерного массива записать в однонаправленный список. Элемент списка хранит значение элемента и его индекс. Создать два списка для элементов двух массивов. На их основе сформировать третий список, элементы которого представляют собой элементы массива, являющегося суммой элементов первых двух массивов (складываются одноименные элементы). Вывести содержимое исходных списков и полученного.

10. В файле хранится последовательность символов, содержащая круглые, квадратные и фигурные скобки (открывающие и закрывающие). Проверить баланс скобок в этой последовательности (для каждой открывающей скобки должна быть соответствующая закрывающая скобка, скобки разных типов должны быть правильно вложены друг в друга). Для проверки использовать стек. Вывести всю сроку символов, если баланс скобок соблюден, и часть строки до первого нарушения баланса скобок.

11. В файле содержатся слова (слово — последовательность символов без пробелов), слова друг от друга отделены одним или несколькими пробелами. Проверить в этой последовательности слов баланс фигурных скобок. Для проверки использовать стек. В качестве результата вывести все фигурные скобки или часть их до первого нарушения баланса.

12. В файле содержатся слова, возможно, с ошибками (неправильная буква, отсутствие какой-либо буквы). Слова начинаются с разных букв и первая буква — правильная. В другом файле те же

слова записаны правильно и хранятся в алфавитном порядке. Считывая каждое слово из исходного файла и организуя его хранение в виде двунаправленного списка (элемент списка хранит одну букву слова), проверить его правильность. Эталонное слово при считывании из файла также заносится в список. Вывести исходный и скорректированный первый файл.

13. Матрица с большим количеством нулей может быть представлена в виде двунаправленного списка, который содержит только ненулевые элементы с указанием номеров строки и столбца. Представить две матрицы в виде списков и получить список для третьей матрицы, представляющей сумму первых двух.

14. Матрица с большим количеством нулей может быть представлена в виде двунаправленного списка, который содержит только ненулевые элементы с указанием номеров строки и столбца. Найти по каждой ненулевой строке матрицы сумму элементов и сформировать однонаправленный список из сумм элементов по каждой строке матрицы.

15. С клавиатуры ввести сведения о студентах (фамилия и номер группы — в виде строки). Сформировать двунаправленный список из вводимых элементов, размещая их сразу при вводе в алфавитном порядке фамилий. Полученные сведения записать в бинарный файл (каждая структура содержит фамилию студента и номер группы). Содержимое полученного файла вывести в окно программы.

16. В текстовом файле содержатся фамилии студентов и их оценки (по три для каждого студента). Считывая эти данные из файла, занести их в двунаправленный список в алфавитном порядке фамилий сразу при добавлении новых элементов в список. Данные из полученного списка записать в бинарный файл, состоящий из структур. Каждая структура включает следующие поля: фамилия, массив из трех оценок. Вывести в окно программы содержимое полученного файла.

17. С клавиатуры ввести элементы матрицы и для ненулевых элементов (нулевых может быть много) сформировать двунаправленный список, содержащий элемент матрицы, а также номера строки и столбца матрицы этого элемента. Для каждой строки матрицы определить произведение элементов (включая и строки, состоящие целиком из нулевых элементов) и занести их в стек. Данные из стека вывести в окно программы.

18. В текстовом файле записана последовательность слов. Считывая последовательно слова из файла, занести их в однонаправленный список в порядке возрастания их длин. Слова одинаковой длины расположить в списке в алфавитном порядке. Упорядоченную последовательность слов вывести во второй файл. Вывести содержимое полученного файла в окно программы.

19. С клавиатуры ввести строки символов. Занести строки в однонаправленный список таким образом, чтобы сначала располагались строки, содержащие только цифровые символы, затем все остальные строки. Цифровые строки должны располагаться в списке в порядке возрастания. Полученную последовательность строк занести в файл и вывести в окно программы.

20. В текстовом файле хранятся сведения о студентах: фамилии и три оценки. Считывая информацию из файла, сформировать двунаправленный список, каждый элемент которого содержит фамилию студента, три оценки и сумму оценок. При формировании списка студентов упорядочить элементы по убыванию суммы баллов. Элементы списка с одинаковой суммой расположить в алфавитном порядке фамилий. Содержимое списка вывести в окно программы и сохранить в бинарном файле структур.

21. С клавиатуры ввести элементы матрицы. Ненулевые элементы (нулевых может быть много) занести в двунаправленный список, каждый элемент которого содержит элемент матрицы, его номера строки и столбца. Для каждой строки матрицы (включая полностью нулевые) определить среднее арифметическое ее элементов и занести их в стек. Вывести содержимое списка.

22. В файле хранятся слова. Считывая слова из файла, сформировать из них двунаправленный список таким образом, чтобы слова в нем были расположены сразу по возрастанию количества содержащихся в них гласных букв. Слова с одинаковым количеством гласных разместить по алфавиту. Вывести содержимое списка. Удалить из списка все слова с заданным количеством гласных. Вывести содержимое измененного списка.

23. С клавиатуры ввести элементы матрицы. Ненулевые элементы, номера их строк и столбцов занести в двунаправленный список. Для каждой строки матрицы определить среднее геометрическое ее элементов (в том числе и для полностью нулевых строк) и занести эти значения в однонаправленный список. Вывести содержимое обоих списков.

24. С клавиатуры ввести элементы квадратной матрицы. Ненулевые элементы, их номера строки и столбца занести в двунаправленный список. Из полученного списка сформировать стек, в который записать все элементы главной диагонали (включая и нулевые). Содержимое списка и стека вывести.

25. С клавиатуры ввести элементы матрицы. Ненулевые элементы, номера их строки и столбца занести в двусвязный список. Найти максимальный и минимальный элементы всей матрицы, их номера строк и столбцов, используя список. Вывести содержимое списка и полученный результат.

26. С клавиатуры ввести элементы матрицы. Ненулевые элементы, номера их строки и столбца занести в двусвязный список. Для каждой строки всей матрицы определить максимальный элемент, номера строки и столбца, где он расположен (использовать список), и занести их однонаправленный список. Вывести содержимое обоих списков.

27. В текстовом файле хранятся слова. Считывая эти слова из файла, сформировать из них двунаправленный список. В списке слова расположить по возрастанию количества содержащихся в них букв. Слова с одинаковым количеством букв упорядочить по алфавиту. Вывести содержимое списка. Удалить из списка копии повторяющихся слов, оставив каждое слово в одном экземпляре. Вывести содержимое преобразованного списка.

28. В файле содержится информация о названии мероприятия, дате и времени его проведения. Считывая эту информацию, сформировать двунаправленный список в соответствии с возрастанием даты, для одинаковой даты мероприятия расположить по возрастанию времени их начала. Вывести содержимое списка. Для вводимой даты сформировать список с названиями мероприятий и времени их проведения. Новый список упорядочить по возрастанию времени. Вывести содержимое полученного списка.

29. Представить многочлен n-го порядка в виде однонаправленного списка. Каждый элемент списка содержит коэффициент многочлена и степень аргумента, нулевые коэффициенты в список не заносить. Вычислить значение многочлена в заданной точке. Создать список из коэффициентов многочлена, являющегося произведением двух многочленов. Вычислить значение многочлена-произведения в заданной точке. Вывести содержимое

списков с коэффициентами исходных двух многочленов и списка, хранящего коэффициенты многочлена-произведения. Вывести вычисленные значения многочленов и их произведения.

30. В файле содержится информация о результатах лыжных соревнований: фамилия, время: часы (целое), минуты (целое), секунды с десятыми долями (действительное). Считывая эту информацию из файла, сформировать двунаправленный список и занести считываемую информацию в порядке возрастания времени, затраченного на преодоление дистанции. Элемент списка содержит фамилию спортсмена и его результат. Для известного нормативного времени составить два списка: один с фамилиями спортсменов, выполнивших норматив, второй — с фамилиями спортсменов, превысивших норматив. Фамилии в списках должны быть расположены в алфавитном порядке. Содержимое всех списков вывести.

6.3. Деревья бинарные

Термин «дерево» был введен в теории графов. Рассмотрим пример графического представления дерева (рис. 6.6).

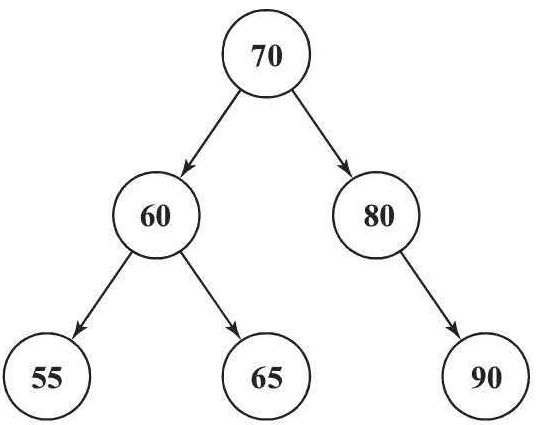

Рис. 6.6

Как и любой граф, дерево состоит из узлов (вершин), соединенных линиями. Если узлы находятся в отношении следования/предшествования, то линии имеют стрелки (см. на рис. 6.6) и называются дугами, а без стрелок — ребрами. Узлы могут содержать метки, в качестве которых могут выступать или тексты, или числа, или их комбинации.

В дереве всегда есть только один узел (у которого нет предшествующих узлов), называемый корнем дерева (на рис. 6.6 это узел с числом 70). Узлы, за которыми не следуют другие, называют терминальными узлами, или листьями (на рис. 6.6 это узлы с числами 55, 65, 90).

Деревья издавна использовались для наглядного представления различных видов данных с иерархической структурой: генеалогии родóв, властных структур, классификаций объектов природы и пр. Из объектов, имеющих отношение к информатике, древовидную структуру имеют файловые структуры запоминающих устройства и классы в ряде объектно-ориентированных систем программирования.

При разработке компьютерных программ деревья, как и списки, реализуются с помощью переменных типа структура и объектов классов, содержащих поля-указатели. Сами эти переменные соответствуют узлам дерева, а поля-указатели реализуют связи между узлами.

Структура бинарного дерева

Наиболее простым по организации и использованию в программе являются бинарные деревья. Их элементы, соответствующие узлам дерева-графа, имеют только два поля-указателя для связи с другими элементами. Дерево в программе, элементы которого имеют тип

```
struct telement
{
   int inf;
   telement *L, *R;
};
```

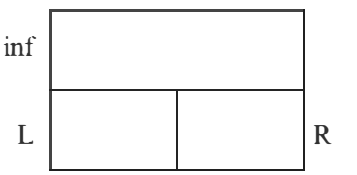

Рис. 6.7

и изображаются в виде таблиц, представлено на рис. 6.7.

Все работы с деревом в программе начинаются с корневого элемента, поэтому в ней должна быть хотя бы одна переменная, содержащая его адрес. На рис. 6.8 такую переменную, которая должна иметь тип telement, представляет элемент с именем `p0`. Листьями дерева, представленного на рис. 6.8, являются элементы с числами 55, 65 и 90. У соответствующих переменных оба поля указателей L и R должны иметь значение `NULL`, что на рисунке обозначено значком «⊥»; `inf` — поле структуры для данных.

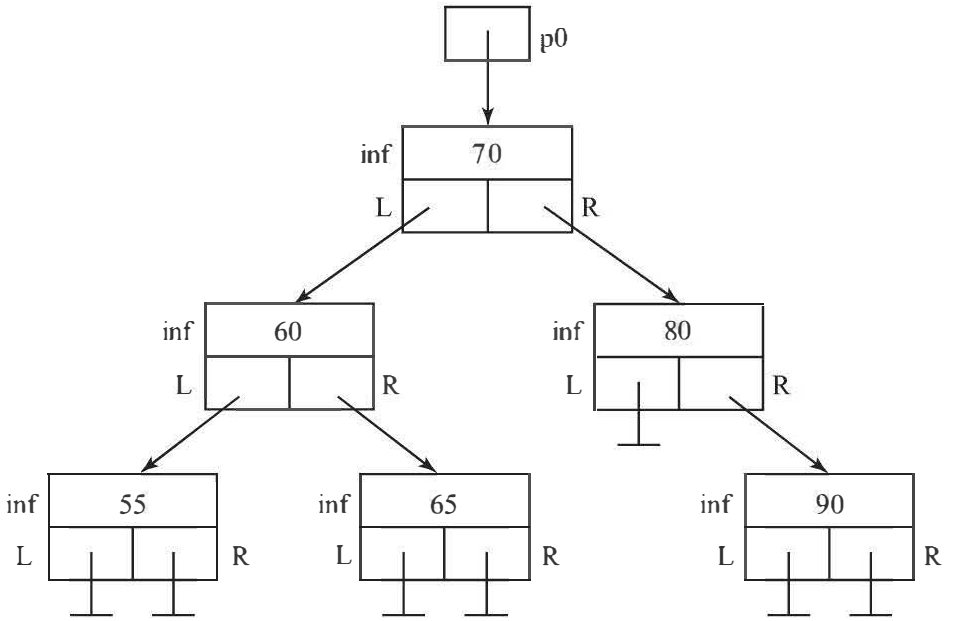

Рис. 6.8

Если в дереве отношения следования узлов задают дуги, то для размещения информации в узлах необходимо использовать некие дополнительные условия. Например, в корне каждого поддерева должно быть число большее чисел во всех остальных его элементах. Другой пример: в каждом поддереве должно быть число такое, что все элементы его поддерева по левой ветви должны иметь меньшие числа, а по правой — бóльшие (именно так размещены числа в дереве рис. 6.8 и именно такой способ размещения данных на дереве будет применен в рассматриваемых далее примерах, а дерево будем называть упорядоченным). При этом следует иметь в виду, что программа должна строить дерево последовательным добавлением элементов.

Изначально дерево должно быть пустым, т. е. переменная `p0` должна иметь значение `NULL`. Далее очередной элемент будет добавляться как новый лист к узлу, найденному при движении от корня дерева по следующему алгоритму: 1) если добавляемое в дерево число меньше числа очередного узла, то выполняется переход к следующему узлу — корню левого поддерева или к корню правого поддерева (считаем, что все числа в создаваемом дереве разные); 2) если соответствующего поддерева нет, то по этой ветке добавляется новый узел с заданным числом. Напри-

мер, если в дерево на рис. 6.8 нужно добавить число 75, то новый узел будет размещен по левой ветке узла с числом 80, а если добавить число 67, то новый узел будет размещен по правой ветке узла с числом 65.

Примеры работы с бинарным деревом

Пример функции, реализующей алгоритм добавления элементов дерева:

```
void nov_uzel(int x,telement **p0)
{//добавление в дерево p0 нового узла с числом x
   telement *p, *pd; int d;
   p=new telement;
   p->info=x;
   p->L=NULL;
   p->R=NULL;
   if ((*p0) == NULL)
   {
      *p0=p;
      return;
   }
   pd=*p0;
   do
   {
      d=pd->info;
         if(x<d)
         {
            if(pd->L == NULL)
            {
               (pd->L) = p;
               return;
            }
            pd=pd->L;
         }
         else
         if(x>d)
         {
            if(pd->R == NULL)
            {
               pd->R = p;
               return;
            }
```

```
            pd=pd->R;
         }
      }
      while (true);
}//void nov_uzel(int x,telement **p0)
```

Проверку работы этой функции проще всего выполнить, построив по данным из массива `minf1` дерево и сделав вывод данных из его узлов. Таким образом:

```
   telement *p0;
//Создание дерева с числами ИЗ МАССИВА minf1
   int minf1[6]={70,60,80,55,65,90};
   p0=NULL;
   for(i=0; i<6; i++)
      nov_uzel(minf1[i],&p0);
//Вывод из дерева
   vyvod_d(p0);
```

Функция `vyvod_d` реализует алгоритм обхода дерева слева с выводом числа из узла при отсутствии у него левого поддерева и обхода левого поддерева с возвратом в текущий узел при наличии поддерева. При этом проще всего использовать рекурсию (как и для многих работ с деревом).

Пример функции `vyvod_d`, реализующей обход:

```
void vyvod_d(telement *p0)
{ //Вывод из упорядоченного дерева
   if(p0==NULL)
   {
      printf(Ruc("\n Дерево пусто.\n"));
      return;
   }
   if(p0->L != NULL)
      vyvod_d(p0->L);
   printf("%d\n",p0->info);
   if(p0->R != NULL)
      vyvod_d(p0->R);
}//void vyvod_d(telement *p0)
```

Такой алгоритм просмотра содержимого дерева, реализованного в функции `vyvod_d`, приведет к выводу чисел в следующем порядке: 55, 60, 65, 70, 80, 90, т. е. по возрастанию. Причем при левом обходе

упорядоченного дерева, построенного с помощью функции **vyvod_d**, результат не будет зависеть от того, в каком порядке числа располагались в массиве и добавлялись в дерево.

В этом примере порядок чисел в массиве **minf1** специально был подобран так, чтобы дерево оказалось сбалансированным. Это означает, что на пути от корня дерева к любому листу количество узлов будет либо максимальным, либо на единицу меньшим максимального (например, дерево на рис. 9 является сбалансированным). Если бы в массиве числа располагались в порядке возрастания или убывания, то дерево выродилось бы в список. Сбалансированность дерева является важной его характеристикой, влияющей на время поиска данных в дереве.

Отметим, что для вывода чисел из дерева не по возрастанию, а по убыванию следует просто выполнить обход дерева справа, заменив во всех операторах указатели **p0->L** на элементы **p0->R**, а указатели **p0->R** на элементы **p0->L**.

После использования дерева его следует уничтожить: освободить память, выделенную для его узлов, а указателю на корень дерева присвоить значение **NULL**. Для этого предназначена такая рекурсивная функция:

```
void del_d(telement *p0[])
{  //удаление дерева
   if(p0[0]->L != NULL)
      del_d(&p0[0]->L);
   if(p0[0]->R != NULL)
      del_d(&p0[0]->R);
   delete *p0;
   *p0=NULL;
}//void del_d(telement *p0[])
```

Следующий фрагмент программы удалит дерево, присвоит указателю **p0** на корень значение **NULL**, а вызов функции **vyvod_d(p0);** приведет к выводу текста «Дерево пусто».

```
   del_d(&p0);
   vyvod_d(p0);
```

Между упорядоченным деревом и упорядоченным массивом общим является то, что поиск нужной информации имеет временну́ю сложность, пропорциональную логарифму N, где N — число

элементов. На массиве это достигается методом половинного деления, т. е. отбрасыванием на каждом шаге половины области, где не может находиться искомое значение, а на дереве — движением от его корня по ветвям к поддеревьям, в которых может содержаться искомое значение. В сбалансированном дереве количество узлов на таком пути не может превышать значения двоичного логарифма от $N + 1$.

По мере добавления данных в дерево степень его сбалансированности может уменьшаться и в какой-то момент потребуется перестройка дерева с целью обеспечения его сбалансированности. Для этого предлагается следующий прием: вывести данные из дерева в массив в порядке возрастания, а затем из упорядоченного массива вновь построить дерево.

Для построения упорядоченного массива служит представленная ниже рекурсивная функция **d_m**, алгоритм которой подобен функции **vyvod_d**, выводящей данные в окно программы:

```
void d_m(telement *p0, int minf[], int *nminf)// или *p0[]
{//из упорядоченного дерева *p0 построить упорядоченный
 //массив int minf[]
   static int n0=0;
   if((p0)->L != NULL)
      d_m(p0->L,minf,nminf);
   //printf("%d\n",p0->info);//отладка
   *nminf=n0; n0++;
   minf[*nminf]=p0->info;
   if(p0->R != NULL)
      d_m(p0->R,minf,nminf);
   *nminf=n0;
}//void d_m(telement *p0, int minf[], int *nminf)
```

Построение из упорядоченного массива сбалансированного упорядоченного дерева выполняет следующая рекурсивная функция:

```
void m_d(int minf[],int il, int ir, telement **p0)// или *p0[]
 {//из упорядоченного массива int minf[] построить
  //упорядоченное дерево *p0
    int ic=(ir+il)/2; telement *z;
   z=new telement;
   if(ir-il>=2)
   {
```

```
      z->info=minf[ic];
      m_d(minf,il,ic-1,&(z->L));
      m_d(minf,ic+1,ir,&(z->R));
   }//if(ir-il>=2)
   else if(il+1==ir)
   { //il==ir-1
      z->info=minf[il];
      z->L=NULL;
      m_d(minf,ic+1,ir,&(z->R));
   }
   else
   { //il==ir
      z->info=minf[ir];
      z->L=NULL;
      z->R=NULL;
   }

   p0[0]=z;
}
```

Алгоритм поиска значения в упорядоченном дереве реализует такая рекурсивная функция:

```
telement *poisk(telement *p0, int x)
{//функция вернет адрес узла, содержащего x,
 //если таковой есть, иначе вернет NULL
   if(p0->info == x)
      return p0;
   if(p0->info > x)
      if(p0->L != NULL)
         poisk(p0->L,x);
      else
         return NULL;
   else
   if(p0->info < x)
      if(p0->R != NULL)
         poisk(p0->R,x);
      else
         return NULL;
}//telement *poisk(telement *p0, int x)
```

Пример использования этой функции:

```
   pp=poisk(p0,55);
   printf("\n%s%p\t",Ruc("Адрес узла = "),pp);
```

```
   if(pp!=NULL)
      printf("%s%d\n", Ruc("Значение в поле info узла =
"),pp->info);
   pp=poisk(p0,25);
   printf("\n%s%p\t",Ruc("Адрес узла = "),pp);
   if(pp!=NULL)
      printf("%s%d\n", Ruc("Значение в поле info узла =
"),pp->info);
```

Различия между упорядоченными массивами и деревьями проявляются прежде всего при добавлении и удалении элементов. Наиболее простой вариант добавления нового узла в дерево, представленный уже рассмотренной функцией `nov_uzel`, требует прохождения от корня дерева до листа, к которому будет прикреплен новый узел, что имеет логарифмическую вычислительную сложность для сбалансированного дерева. Для массива такая же работа потребует в среднем сдвига половины указателей на элементы на одну позицию (если элементы занимают много памяти, то имеет смысл представлять их в массиве указателями, чтобы при добавлении нового элемента смещать в массиве только указатели).

Деревья строятся и используются в оперативной памяти машины. В случаях каких-либо аварийных ситуаций и прочих причин потери данных или просто при необходимости завершения сеанса работы должны создаваться копии деревьев на внешних носителях. Для этого может использоваться функция, подобная функции `d_m`, но выводящая данные не в массив, а в бинарный файл.

Чтобы восстановить из файла дерево как сбалансированное, для ускорения работ следует сначала скопировать все из этого файла в массив и уже из массива восстанавливать дерево ранее рассмотренной функцией `m_d`.

Вопросы для самопроверки

1. Что представляют собой динамические структуры данных?
2. Что такое однонаправленный линейный список, двунаправленный?
3. Какие операции выполняются над линейными списками?
4. Какие сложности возникают при добавлении нового элемента в однонаправленный список, при удалении элемента из списка?

Литература

Алексеев Ю.Е., Куров А.В. Практикум по программированию на языке С в среде VS C++. Ч. 1. wwwcdl.bmstu.ru/iu7/PraktikumC.html

Алексеев Ю.Е., Куров А.В. Практикум по программированию на языке С в среде VS C++. Ч. 2. wwwcdl.bmstu.ru/iu7/PraktikumC2.html

Керниган Б.И., Ритчи Д.М. Язык программирования С: пер. с англ. М.: Вильямс, 2006. 304 с.

Пахомов Б.И. C/C++ и MS Visual C++ 2008 для начинающих. СПб.: БХВ-Петербург, 2008. 624 с.

Шилдт Г. Полный справочник по C++: пер. с англ. М.: Вильямс, 2008. 800 с.

Оглавление

Учебное издание

Алексеев Юрий Евтихович
Куров Андрей Владимирович

Обработка нечисловых типов данных в среде MS VS C++

Редактор *О.М. Королева*
Художник *Я.М. Ильина*
Корректор *Н.В. Савельева*
Компьютерная графика *Е.Н. Комаровой*
Компьютерная верстка *Н.Ф. Бердавцевой*

Оригинал-макет подготовлен
в Издательстве МГТУ им. Н.Э. Баумана.

В оформлении использованы шрифты
Студии Артемия Лебедева.

Подписано в печать 21.02.2017. Формат 60×90/16.
Усл. печ. л. 12,25. Изд. № 453-2015.
Тираж 50 экз. 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