Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

**О. А. Кравченко, С. М. Мовшович,
Е. В. Коробейникова**

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

**КУРС ЛЕКЦИЙ
по одноименной дисциплине
для студентов специальности 1-40 01 02
«Информационные системы и технологии
(по направлениям)» дневной формы обучения**

Электронный аналог печатного издания

Гомель 2010

УДК [004.45+004.33](075.8)
ББК 32.973.26-018я73
К78

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 7 от 09.03.2009 г.*)

Рецензент: канд. техн. наук, доц. каф. «Вычислительная математика и программирование»
ГГУ им. Ф. Скорины *Л. А. Цурганова*

Кравченко, О. А.

К78 Основы алгоритмизации и программирования : курс лекций по одноим. дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. формы обучения / О. А. Кравченко, С. М. Мовшович, Е. В. Коробейникова. – Гомель : ГГТУ им. П. О. Сухого, 2010. – 111 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-942-5.

Рассмотрены вопросы алгоритмизации и программирования задач обработки базовых структур данных: числовых и символьных массивов, структур и массивов структур, текстовых и бинарных файлов, а также использование поразрядных логических операций для обработки данных.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения.

УДК [004.45+004.33](075.8)
ББК 32.973.26-018.я73

ISBN 978-985-420-942-5

1. СТРОКИ И СИМВОЛЫ

1.1. Работа с символами

Для хранения отдельных символов используются переменные типа ***char***. Пример описания: ***char t, p*****;** // Описаны 2 переменные символьного типа.

Возможными значениями символьной переменной могут быть различные символы. Такими символами могут быть почти все знаки из кодовой таблицы ПК за исключением специальных управляющих знаков, не имеющих графического изображения. Например, можно записать присваивания: **t='Q'; p='p'**; здесь **t** и **p** – переменные, которые могут принимать разные значения, а **'Q'** и **'p'** – конкретные значения, присваиваемые этим переменным. После этого в ячейке памяти, которая зарезервирована для переменной **t**, будет записан код символа '**Q**', а для переменной **p** – код символа '**p**'. ***Переменная p и значение 'p' – это не одно и то же!***

Код символа – это целое число. Занимает в памяти 1 байт. Коды всех символов сведены в кодовую таблицу, имеющую 16 строк и 16 столбцов, пронумерованных шестнадцатиричными цифрами: 0, 1, …, 9, A, B, C, D, E, F. Таким образом, кодовая таблица хранит коды 256 символов. Из кодовой таблицы можно определить код символа в шестнадцатиричной системе счисления – это последовательность шестнадцатиричных цифр, составленная из номера столбца и номера строки. Например, символ, стоящий на пересечении столбца 9 и строки F, имеет шестнадцатиричный код $9F_{16}$. Можно перевести код в десятичную систему счисления: $9F_{16}=9*16+15=159_{10}$.

Существуют специальные программы вывода таблицы кодов символов на экран и вставки символов в текст программы. Например, **ascii.com**.

Таблица ASCII кодов, применяемая в системе программирования Borland C++ 3.1 for DOS, приведена на рис. 1.1.

Хотя в языке С можно одновременно использовать переменные типов ***int*** и ***char***, все же отметим, что если объявлено ***int a; char b;*** и назначено ***a=5; b='5';***, то ***a*** и ***b*** – это не одно и то же: переменная ***a*** имеет числовое значение ***5***, а ***b*** – значение символа ***'5'***. Число ***5*** в двоичном коде равно ***101***, а двоичный код символа ***'5'*** есть ***110101*** ($110101_2= 35_{16}=53_{10}$).

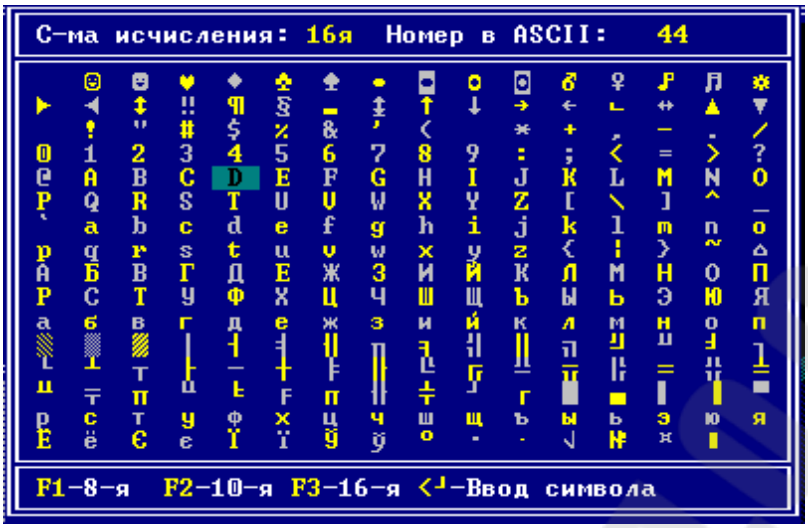

Рис. 1.1. Пример кодовой таблицы

Ввод-вывод символов

Рассмотрим наиболее распространенные функции ввода символа с клавиатуры и его вывода на экран дисплея (консольный ввод-вывод): *getchar*, *putchar* (для ввода-вывода символа); *gets*, *puts* (для ввода-вывода строки); *scanf*, *printf* (для форматированного ввода-вывода); и функцию *fflush* очистки буфера *stdin*.

Функция *getchar* предназначена для ввода символа, не имеет параметров, возвращает целое число – код введенного символа. Если символ не может быть прочитан, то возвращает значени EOF. Обращение к функции может иметь вид: *getchar*(). Функция может использоваться в выражении, например, ch=getchar();.

Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода символа и/или нажатия клавиши Enter, если буфер клавиатуры (*stdin*) пуст. Если буфер клавиатуры не пуст, то введенным считается символ, находящийся в буфере. Поэтому перед использованием функции *getchar()* рекомендуется очищать буфер клавиатуры обращением к функции *fflush*: ***fflush(stdin)*;.**

Функция *putchar* предназначена для вывода символа на стандартное устройство вывода. Имеет один параметр типа *int* (код выводимого символа) или char. Если вывод успешен, возвращает значение кода символа, иначе, возвращает значение EOF. После вывода символа на экран дисплея курсор не переходит к началу новой строки.

Пример ввода и вывода символа.

```
/* Ввод-вывод символа */
#include <stdio.h>
```

```
 main()
 {
 int ch;                                  //Код вводимого символа
 puts("Введите любой символ");    //Вывод строки - подсказки
 ch=getchar();                           /* Ввод символа и присваивание
его кода переменной ch */
 puts("Вы ввели символ");           //Вывод строки - сообщения
 putchar(ch);              /* Вывод символа, определенного кодом ch */
 printf("\n");                           /* Перевод курсора к началу новой
строки */
 putchar('A');           /* Вывод символа 'A', заданного константой */
 printf("\n");
 putchar(65);             /* Вывод символа 'A', заданного кодом символа
*/
 fflush(stdin);            //Очистка буфера ввода
 getchar();   /* Ввод символа и/или нажатия Enter для организации за-
держки смены окна Output на окно редактора текста*/
 return(0);
}
```

Как видно из примера, функция *getchar()* может использоваться в операторе присваивания. В этом случае код введенного символа присваивается переменной типа *int* (в примере – переменной ch).

Функции *putchar('A')* и *putchar(65)* выводят один и тот же символ *'A'*, символьной константой и кодом символа (65).

Если требуется ввести в одном операторе данные разных типов (в том числе и символ), то удобнее использовать **функцию *scanf*** (для ввода символа используется форматная спецификация %c).

Если требуется вывести в одном операторе данные разных типов (в том числе и символ), то удобнее использовать **функцию *printf***. При этом для вывода символа используется форматная спецификация %c; для вывода кода символа в десятичной, восьмеричной или шестнадцатеричной системе счисления используются спецификации %d, %o или %X, соответственно.

Пример. Ввести значение символьной переменной, вывести значение в виде символа и десятичного, восьмеричного и шестнадцатеричного кодов.

Текст программы

```
/* Ввод символьной переменной и вывод в различных формах*/
#include<stdio.h>
#include<conio.h>
main()
{
 char b;                          //Объявление символьной переменной
 puts("Введите один символ");
 b=getchar();                     //Ввод значения переменной b
 printf("\tb=");
 putchar(b);                      //Вывод значения переменной b
 printf("\nДесятичный код b=%d\n",b); //Вывод десятичного кода b
 printf("\tВосьмеричный код b=%o\n",b); //Вывод восьмеричного кода
 printf("\t\tШестнадцатеричный код b=%X\n",b);
 printf("b='%c'\n",b);
 getch();
 return(0);
}
```

Ниже приведены результаты выполнения программы **(**серым цветом выделены символы, которые вводил с клавиатуры пользователь):

```
Введите один символ
*
      b=*
Десятичный код b=42
      Восьмеричный код b=52
            Шестнадцатеричный код b=2A
b='*'

Введите один символ
{
      b={
Десятичный код b=123
      Восьмеричный код b=173
            Шестнадцатеричный код b=7B
b='{'
Введите один символ
#
```

```
      b=#
Десятичный код b=35
      Восьмеричный код b=43
            Шестнадцатеричный код b=23
b='#'

Введите один символ
5
      b=5
Десятичный код b=53
      Восьмеричный код b=65
            Шестнадцатеричный код b=35
b='5'
```

Определение принадлежности символа какому-либо множеству

В библиотеке ***ctype*** определен целый ряд функций, проверяющих принадлежность символа какому-либо множеству: множеству букв (***isalpha***), букв или цифр (***isalnum***), разделителей (***isspace***), знаков пунктуации (***ispunct***), цифр (***isdigit***), видимых символов (***isgraph***), букв верхнего регистра (***isuper***), букв нижнего регистра (***islower***), печатаемых символов (***isprint***) и т. д.

Аргументом каждой из этих функций является символ. Функция возвращает значение ***true***, если символ принадлежит конкретному множеству символов, или ***false*** в противном случае.

Пример. Ввести символ с клавиатуры и проверить, является ли он знаком пунктуации.

```
//Текст программы
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
void main()
{
  char a;
  puts("Введите один символ");
  a=getchar();
  if(ispunct(a))
```

```
    printf("Символ %c явл. знаком пунктуации\n",a);
  else
    printf("Символ %c не явл. знаком пунктуации\n",a);
  getch();
}
```

В приведенном тексте программы использовалась функция ispunct(a) для проверки принадлежности введенного символа множеству знаков пунктуации.

Ниже приведены результаты выполнения программы **(**серым цветом выделены символы, которые вводил с клавиатуры пользователь):

```
Введите один символ
.
Символ . явл. знаком пунктуации
Введите один символ
^
Символ ^ явл. знаком пунктуации
Введите один символ
e
Символ e не явл. знаком пунктуации
Введите один символ
#
Символ # явл. знаком пунктуации
Введите один символ
[
Символ [ явл. знаком пунктуации
Введите один символ
-
Символ - явл. знаком пунктуации
Введите один символ
n
Символ n не явл. знаком пунктуации
```

Следующий *пример* показывает, что символьная переменная может быть определена символом или кодом символа.

```
#include<stdio.h>
void main()
```

```
{
 char c,c1;
 c='A';          //символьная переменная определена символом
 c1=65;        //символьная переменная определена кодом символа
 printf("%c %d\n", c, c);          //1
 printf("%d %c\n", c1, c1);       //2
 if(c1=='A' && c== 65)
   puts("Обе переменные определяют символ  A");    //3
  else puts("****");
}
```

Результаты выполнения программы показывают, что значением переменных символьного типа c, c1 на печати будет символ, если выводить значение по форматной спецификации %c, и код символа, если выводить значение по форматной спецификации %d:

```
A 65         //результат выполнения оператора //1
65 A        //результат выполнения оператора //2
Обе переменные определяют символ  A//рез. выполнения оп. //3
```

Пример использования функции ***char*** для определения кода символа. Функция ***char*** имеет один аргумент – символ. Возвращает код своего аргумента.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int cod;        //код символа
 char S;         //символ
 puts("Введите один символ");
 S=getchar();
 cod=char(S);
 putchar(S);       //1
 printf("\t\n");
 printf("%c %d %d\n",S, S, cod);    //2
 getch();
}
```

Результаты выполнения программы:

```
Введите один символ
2
      2                    //результат выполнения оператора //1

       2 50 50       //результат выполнения оператора //2

Введите один символ
%
%
      % 37 37
Введите один символ
D
D
      D 68 68
```

1.2. Строки

1.2.1. Понятие и описание строки

В языке С отсутствует специальный строковый тип. ***Строка в С – это массив символов***. В памяти ЭВМ строка представляется как массив элементов типа ***char***, в конце которого помещается символ **'\0'** – ***завершающий нуль-символ***.

Строковая константа – это последовательность символов, заключенная в кавычки ("). Например, " Задача имеет множество решений". В памяти представляется как массив элементов типа ***char*** с завершающим нуль-символом (**'\0'**).

Общий вид описания строки:

char <имя строки>**[**<длина строки с учетом нуль-символа>**]**;

```
Например,     char S[100];
Или
 const int N=100; //Удобно задавать длину строки с помощью
  char S[N];         //именованной константы
```

В такой строке S можно хранить 99 символов и завершающий нуль-символ.

При описании строки можно выполнять ее инициализацию. Например,

char S1[100]= "Исходный массив";

При этом нуль-символ автоматически формируется за последним символом символьной константы.

Если строка при описании инициализируется, то можно опускать длину строки. Например,

char S1[]= "Исходный массив"; //16 символов

Примечание. Имя строки, как и любого массива – это указатель-константа. Поэтому ошибкой будет попытка использовать имя строки в некоторых операциях адресной арифметики. Например, нельзя выполнить такую "пересылку" строки символов в массив:

char a[20]; //***Так***
a="Строковая константа"; //***нельзя***

Описание динамической строки

Для размещения строки в динамической памяти необходимо описать указатель на Char, а затем выделить память с помощью ***new*** или malloc (new предпочтительнее).

Например,

char *S3=new char[m]; //m должно быть уже определено

Или

char *S3;
S3=new char[m];

Примечание. Динамические строки, как и другие массивы, нельзя инициализировать.

Например, оператор,

char *S4="На нуль делить нельзя";

Создает не строковую переменную, а указатель на строковую константу, изменить которую невозможно. S4 – адрес первого символа строковой константы.

1.2.2. Ввод-вывод строк

Рассмотрим ввод-вывод строк с помощью функций, унаследованных из библиотеки C: ***gets*** и ***puts***, ***scanf*** и ***printf***.

Функции ***gets*** и ***puts*** используются, если работа производится только со строками. Функции ***scanf*** и ***printf*** удобнее использовать в том случае, если в одном операторе требуется ввести или вывести данные разных типов.

Функция ***gets*** предназначена для ввода строки. Имеет один параметр, задающий адрес области памяти, в которую помещаются символы вводимой строки. В языке С имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: **gets***(name)*, где *name* – переменная строкового типа – имя вводимой строки. Выполняя эту функцию, программа приостанавливает свою работу и ждет от пользователя ввода последовательности символов и/или нажатия клавиши Enter. Символ новой строки в строку не включается, вместо него в строку заносится нуль-символ. Функция возвращает указатель на строку s, а в случае возникновения ошибки или конца файла – NULL.

Функция ***puts*** предназначена для вывода строки на стандартное устройство вывода. Имеет один параметр, задающий адрес области памяти, из которой на экран выводятся символы. Как уже отмечалось, имя переменной, имеющей строковый тип, является этим адресом. Обращение имеет вид: *puts(name)*, где name – переменная-строка – имя выводимой строки или строка символов, заключенная в кавычки. После вывода строки курсор перемещается к началу новой строки экрана, т. е. завершающий нуль-символ строки заменяется на символ новой строки. Возвращает неотрицательное значение при успехе или EOF при ошибке.

Функция ***printf*** предназначена для вывода форматированной последовательности данных.

Функция ***scanf*** предназначена для ввода данных в заданном формате. Обращение имеет вид:

scanf(nf,&a1,&a2,...)

Здесь *nf – форматная* строка; *&a1,&a2,...* – список ввода – указатели на значения вводимых переменных *a1, a2, ...*

Подробно эти функции рассматривались в разделе "Консольный ввод-вывод"[].

1.2.3. Операции над строками

1.2.3.1. Реализация операции присваивания

Поскольку строка является массивом, а не специальным типом данных, то для строк не определена операция присваивания. Присваивание можно выполнить посимвольно, т. е. «вручную», или с помощью стандартных функций ***strcpy*** и ***strncpy***.

Способ 1

Пример

```
s[0]='В'; s[1]='в'; s[2]='о'; s[3]='д';
```

способ 2

Для использования функций ***strcpy*** и ***strncpy*** к программе следует подключить заголовочный файл <***string.h***> предложением #include <***string.h***>

Обращение к **функции *strcpy*** имеет вид:

strcpy(s1,s2);

Функция ***strcpy*** копирует все символы строки ***s2***, включая завершающий нуль-символ, в строку ***s1*** и возвращает ***s1***.

Пример

```
/* Копирование строки s2 в строку s1 с помощью strcpy  */
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
 int n;                    //Длина строки
 char s2[20]="Скоро сессия!";  //Исходная строка
 n=strlen(s2)+1;           //Длина s1
 char *s1=new char[n];     //Строка - копия s1
 // char s1[20];
 strcpy(s1,s2);            //Копирование s2 в s1
 puts(s2);                 //Вывод s2
 puts(s1);                 //Вывод s1
 puts(strcpy(s1,s2));      //strcpy возвращает s1
 getch();
 return(0);
}
```

Результат выполнения программы – три строки:

```
Скоро сессия!
Скоро сессия!
Скоро сессия!
```

Обращение к **функции *strncpy*** имеет вид:

strncpy(s1,s2,n);

Функция ***strncpy*** копирует не более ***n*** символов из строки ***s2*** в строку ***s1*** и возвращает ***s1***. При этом если длина исходной строки ***s2*** превышает или равна ***n***, нуль-символ в конец строки ***s1*** не добавляется. В противном случае строка ***s1*** дополняется нуль-символами до *n*-го символа. Если строки перекрываются, поведение не определено.

Пример

```
/* Копирование строки s2 в строку s1 с помощью strncpy  */
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
  int n;                        //Длина строки
  char s2[20]="Скоро сессия!";     //Исходная строка
  n=strlen(s2)+1;               //Длина s1
  //char *s1=new char[n];         //Строка - копия s1
  char s1[20];
  strncpy(s1,s2,n);             //Копирование s2 в s1
  puts(s2);                    //Вывод s2
  puts(s1);                    //Вывод s1
  puts(strncpy(s1,s2,n));         //strncpy возврващает s1
  getch();
return(0);
}
```

Результат выполнения программы такой же, как и в примере на с. 13.

Примечание 1. В рассмотренных примерах использована функция strlen(s), возвращающая фактическую длину строки s, не включая нуль-символ.

Примечание 2. Программист должен сам заботиться о том, чтобы в строке-приемнике (s1) хватило места для строки-источника (s2), и о том, чтобы строка всегда имела нуль-символ.

Выход за пределы строки и отсутствие нуль-символа является распространенными причинами ошибок в программах обработки строк.

1.2.3.2. Преобразование строки в число

Преобразование строк в числа можно выполнить с помощью функций ***atoi, atol, atof.*** Обратные преобразования – с помощью функций ***itoa***.

Функция *atoi(s)* преобразует строку, содержащую символьное представление целого числа в соответствующее целое число. Призна-

ком конца числа служит первый символ строки, который не может быть интерпретирован как принадлежащий целому числу. Если преобразование не удалось, возвращает 0.

Функция *atol(s)* преобразует строку, содержащую символьное представление длинного целого числа в соответствующее целое число.

Функция *atof(s)* преобразует строку, содержащую символьное представление вещественного числа в соответствующее вещественное число двойной точности.

Для использования функций ***atoi(s) atol(s)*** и ***atof(s)*** к программе следует подключить заголовочный файл <***stdlib.h***> предложением #include < ***stdlib.h***>

Пример. Данные об участнике соревнований (номер участника, рост и вес) содержатся в строке символов. Вывести номер, рост и вес участника соревнований, зная, что номер записан в самом начале строки, рост, начиная с позиции 11, вес – с позиции 26.

```
/* Преобразование строки в число */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
main()
{
 char s[]="10) Рост – 162 см., вес – 63.4кг";     //Строка
 int n;                                  //Целое число
 long h;                                 //Длинное целое
 double w;                               //Вещественное число
 n=atoi(s);                              //Преобразование в целое
 h=atol(&s[11]);                         //Преобразование в длинное целое
 w=atof(&s[26]);          //Преобразование в вещественное число
 printf("Участник # %d. Его рост %ld см и вес %lf кг\n",n,h,w);
 puts("Исходная строка");
 puts(s);
 puts("Первая подстрока");
 puts(&s[11]);
 puts("Вторая подстрока");
 puts(&s[26]);
 getch();
```

```
 return(0);
}
```

Результат выполнения программы:

```
Участник # 10.  Его рост 162 см и вес 63.400000 кг
Исходная строка
10) Рост – 162 см, вес – 63.4 кг
Первая подстрока
162 см, вес – 63.4 кг
Вторая подстрока
63.4 кг
```

Заметим, что строка определяется адресом ее нулевого символа. Для всей строки s таким адресом является имя строки ***s***. Число **10** в символьном представлении находится в строке ***s*** в самом ее начале. Поэтому аргументом функции ***atoi*** является строка ***s***. Число **162** в символьном представлении находится в строке ***s***, начиная с позиции **11**. Поэтому аргументом функции ***atol*** является подстрока строки ***s***, определяемая адресом **&s[11]**. Число **63.4** в символьном представлении находится в строке ***s***, начиная с позиции **26**. Поэтому аргументом функции ***atof*** является подстрока строки ***s***, определяемая адресом **&s[26]**.

1.2.3.3. Поиск подстроки в строке

Функция *strstr(s1,s2)* выполняет поиск подстроки ***s2*** в строке ***s1*** (первого вхождения подстроки ***s2*** в строку ***s1***). Обе строки должны завершаться нуль-символами. В случае успешного поиска функция возвращает указатель на найденную подстроку. В случае неудачи – NULL.

Пример. Определить, содержится ли строка ***s2*** в строке ***s1*** в качестве подстроки.

```
      #include <stdio.h>
#include <string.h>
#include <conio.h>
main()
{
 const int n=81;
```

```
 char s1[n],s2[n];
 char *p;
 clrscr();
 puts("Введите строку s1");
 gets(s1);
 puts("Введите строку s2");
 gets(s2);
 p=strstr(s1,s2);
 if(p)
  {
   printf("Подстрока '%s'\n начинается в строке: %s\n",s2,s1);
   printf("символом '%c'\n этим символом начинается подстрока:
%s\n",*p,p);
  }
    else puts("NO");
 getch();
 return(0);
}
```

Заметим, что ***p*** – символ в строке ***s1***, с которого начинается ***s2***; ***p*** – подстрока, начиная с символа ****p*** до конца строки ***s1***.

Пример выполнения программы

Введите строку s1

Скоро ли сессия? Так хочется сдать экзамен по ОАиП!

Введите строку s2

экзамен

Подстрока ' экзамен '

начинается в строке: Скоро ли сессия? Так хочется сдать экзамен по ОАиП!

символом 'э'

этим символом начинается подстрока: экзамен по ОАиП!

1.2.3.4. Сцепление двух строк (конкатенация)

Функция *strcat(s1,s2)* присоединяет строку ***s2*** в конец строки ***s1*** и возвращает указатель на строку, совпадающий с первым аргументом. При этом сначала из строки ***s1*** удаляется завершающий нуль-символ. В конце новой строки ***S1*** помещается ***'\0'***.

Программист должен сам обеспечить достаточную длину строки s1.

Пример

```
char s1[40]="Скоро Новый Год!";
char s2[]="Скоро первая сессия!";
printf(" %s\n", strcat(s1,s2));
```

Будет получена строка, полученная присоединением строки ***s2*** в конец строки ***s1***:

Скоро Новый Год! Скоро первая сессия!

Пример. Вторым параметром функции ***strcat*** является текстовая константа:

```
char s3[50]="Успешной сдачи";
printf("%s\n\t%s\n", strcat (s3," 1-ой сессии!!!"), s3);
```

Будет получено две строки:

Успешной сдачи 1-ой сессии!!!
Успешной сдачи 1-ой сессии!!!

Видно, что строка, возвращенная функцией ***strcat,*** совпадает со строкой ***s3.***

Пример

```
char s4[35]="Успехов и здоровья";
char s5[]=" в Новом Году!";
char *p;
p=strcat(s4, s5);
pintf("%s\n%s\n",s4,p);
```

Будет получено две строки:

Успехов и здоровья в Новом Году!
Успехов и здоровья в Новом Году!

В рассмотренном примере описан указатель на char: ***char *p;***. Указатель ***p*** определяет строку, возвращаемую функцией: ***p=strcat(s4, s5);***. Строка ***s4*** совпадает со строкой, определяемой указателем ***p***.

Функция *strncat(s1,s2,n)* присоединяет ***n*** символов строки, на начало которой указывает ***s2***, в конец строки, на начало которой указывает ***s1*** и возвращает указатель на строку, совпадающий с первым аргументом. Сформированная строка ***s1*** ограничивается ***'\0'***.

Если n больше длины строки ***s2***, то выполняется простая конкатенация.

Если длина строки ***s2*** меньше ***n***, то все символы *s2* присоединяются к строке ***s1***.

1.2.3.5. Определение позиции первого вхождения символа из заданного набора символов

Функция ***strcspn(s1,s2***) сопоставляет каждый символ строки ***s1*** со всеми символами строки, на начало которой указывает ***s2***, и возвращает позицию первого вхождения символа строки ***s2*** в строке ***s1***. Символ '/0' в сравнении не участвует.

Если строка ***s1*** начинается с символа, встречающегося в строке ***s2***, то функция возвращает значение нуль. Если строка ***s1*** не содержит ни одного символа строки ***s2***, то возвращаемое функцией значение совпадает с длиной строки ***s1***.

Примеры

```
printf("%d %d %d\n",
strcspn("asdf",  "hjyars"),    //s1 начанается с 'a'
strcspn("asdf", "ghtdu"),    //в s1 'd' в позиции № 2
strcspn("asrt", "hj"));       //в s1 не символов s2
```

Получим:

0 2 4

Здесь 0 и 2 – позиции в строке s1, 4 – длина строки s1.

1.2.3.6. Сравнение двух строк

Сравнение строк производится посимвольно слева направо. Большей считается та строка, в которой первый несовпадающий символ имеет больший код в кодовой таблице.

Функция *strcmp(s1,s2)* сравнивает строки ***s1*** и ***s2***. Возвращает отрицательное значение, если ***s1<s2***, нулевое, если ***s1=s2*** или положительное значение, если ***s1>s2***.

Функция ***strncmp(s1,s2, n)*** сравнивает строку ***s1*** и первые ***n*** символов строки ***s2***. Возвращает отрицательное значение, если ***s1< чем первые n символов s2***, нулевое, если ***s1=первым n символам s2*** или положительное значение, если ***s1> чем первые n символов s2***.

Пример. Ввести две строки и вывести их в лексикографическом порядке.

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

```
#include<string.h>
#include<conio.h>
main()
{
  char s1[20];
  char s2[20];
  puts("Введите 1-ю строку ");
  gets(s1);
  puts("Введите 2-ю строку ");
  gets(s2);
  if(strcmp(s1,s2)<0)
      {
       puts(s1);
       puts(s2);
      }
   else
    if(strcmp(s1,s2)>0)
      {
        puts(s2);
        puts(s1);
      }
     else puts("Строки совпадают");
  getch();
  return(0);
}
```

Примеры результатов выполнения программы:

Введите 1-ю строку
abcdefg
Введите 2-ю строку
abcdefg
Строки совпадают

Введите 1-ю строку
bcdert
Введите 2-ю строку
bccertyu
bc**c**ertyu
bc**d**ert

```
Введите 1-ю строку
rtyu
Введите 2-ю строку
rtdh
rtdh
rtyu
```

1.3. Примеры решения задач по обработке строк

Пример 1. Сколько раз подстрока s1 содержится в строке s?

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
        main()
{
 char s[20];
 char s1[20];
 puts("Введите строку ");
 gets(s);
 puts("Введите подстроку ");
 gets(s1);
 int k=0, i=0;
 char *p=s;
 int n=strlen(s1);
 printf("n=%d %c\n",n,*p);
 while(strstr(p,s1) && i<n)
  {
   k=k+1;
   p=p+n+1; printf("%c\n",*p);
  }
 puts("В строке:");
 puts(s);
 printf("%d раз встречается %s\n",k,s1);
 getch();
 return(0);
}
```

Пример 2

мама мыла, мама стирала, мама варила, будила студента – строка s

мама – подстрока s1

Эта же задача с использованием srtncmp

```
/*Сколько раз подстрока s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
 char s[20];
 char s1[20];
 puts("Введите строку ");  gets(s);
 puts("Введите подстроку ");  gets(s1);
 int k=0,i=0;
 int m=strlen(s);
 int n=strlen(s1);
 printf("m=%d n=%d\n",m,n);
 while(i<m)
  {
   if(strncmp(s1,&s[i],n)==0)
     {
       k=k+1; i=i+n;
      }
   else i=i+1;
  }
 puts("В строке:");
 puts(s);
 printf("%d раз(а) встречается %s\n", k, s1);
 getch();
return(0);

}
/*Сколько раз слово s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
 char s[20];
```

```
 char s1[20];
 puts("Введите строку ");  gets(s);
 puts("Введите слово ");  gets(s1);
 int k=0,i=0;
 int m=strlen(s);
 int n=strlen(s1);
 printf("m=%d n=%d\n",m,n);
 while(i<m)

  {
   if(strncmp(s1,&s[i],n)==0)
{ if(i==0)
   {
    if(isspace(s[i+n]) || ispunct(s[i+n]) || (s[i+n]=='\0'))
      { k++; i=i+n; }
      else i++;
   }
  else
   if((isspace(s[i-1])||ispunct(s[i-1])) &&
    (isspace(s[i+n])|| ispunct(s[i+n]))||(s[i+n]=='\0'))
    { k++; i=i+n; }
   else i++;
 }
   else i++;
  }
 puts("В строке:");
 puts(s);
 printf("%d раз(а) встречается %s\n",k,s1);
 getch();

}

/*Сколько раз слово s1 встречается в строке s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
```

```
 char s[20];
 char s1[20];
 puts("Введите строку ");  gets(s);
 puts("Введите слово ");  gets(s1);
 int k=0,i=0;
 int m=strlen(s);
 int n=strlen(s1);
 printf("m=%d n=%d\n",m,n);
 //Добавление пробела в начало слова
 for(i=m;i>=0;i--)
   s[i+1]=s[i];
 s[0]=' ';
 puts(s);
 //Поиск в строке
 while(i<m+1)
    if(strncmp(s1,&s[i],n)==0)
{
      if((isspace(s[i+n]) || ispunct(s[i+n]) || (s[i+n]=='\0')) &&
      (isspace(s[i-1])||ispunct(s[i-1])))
        { k++; i=i+n; }
        else i++;
 }
    else i++;
 //Удаление из строки символа s[0]
 for(i=1;i<=m+1;i++)
    s[i-1]=s[i];
 puts("В строке:");
 puts(s);
 printf("%d раз(а) встречается %s\n",k,s1);
 getch();

}
/*Сколько букв в третьем слове строки s? */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
```

```
 char s[80];
 int k,k3,i,n;
 puts("Введите строку ");  gets(s);
 n=strlen(s);
 k=0;//Кол. слов
 //Поиск в строке
 i=0; //Номер символа
 while(i<n && k!=3)
   {
    if(isalpha(s[i]))   //Слово началось
{
 k++;
 k3=0; //Кол. букв в слове
 while(isalpha(s[i]))
   {
    k3++; i++;
   }
}
    i++;
  }
 if(k==3)
   {
    printf("В третьем слове строки:");
    puts(s);
    printf(" %d символ(ов,а) \n",k3);
   }
 else
   {
     puts("В строке:");
     puts(s);
     puts("меньше трех слов");
   }
 getch();
}
/*Вставить пробел в начало строки (с использ. указателей) */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
```

```
void main()
{
  char *s,*p;
  puts("Введите строку ");  gets(s);
  int m=strlen(s);
  p=s-1;
  *p=' ';
  puts(s);
  puts(p);
  s=p;
  puts(s);
  getch();

}

/*  Вставить пробел перед'.'  */
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char *s,*p,*c;
  puts("Введите строку ");  gets(s);
  puts(s);
  int m=strlen(s);
  p=strchr(s,'.');//Позиция точки
  puts(p);
  strncpy(c,s,p-s); //Подстрока до точки
  puts(c);
  strcat(c," ");
  strcat(c,p);
  s=c;
  puts(s);
  puts(p);
  puts(c);
  getch();

}
```

2. ЗАПИСИ (СТРУКТУРЫ)

2.1. Определение записи (структуры)

Запись (record) – структурированный (составной) тип данных, состоящий из фиксированного числа компонентов разного типа, называемых ***полями*** (Helds) записи.

Запись связывает элементы разных типов в один объект. Этим запись отличается от массива (в массиве все элементы имеют одинаковый тип). Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются, т.е. каждому полю записи присваивается имя. Например, такую информацию о студенте, как фамилия, имя, отчество, адрес, возраст, экзаменационные оценки, средняя успеваемость можно объединить в одну запись, имеющую пять полей. Имена и типы полей приведены в таблице 2.1.

Таблица 2.1

Запись	**Поля записи**				
	Fio	**Adress**	**Age**	**Oc**	**Sr**
Типы полей	Строка	Строка	Целое	Массив целочис.	Вещ.

Поля записи могут иметь любой допустимый в Си тип данных, в том числе поля сами могут быть структурами (записями). Но поле не может иметь тип этой же структуры, но может быть указателем на нее.

Описание структурной переменной состоит из двух этапов:

1. Описание шаблона структуры.
2. Описание структурной переменной.

Синтаксис описания шаблона структуры:

```
struct  <имя_ шаблона>
  {
    <тип1>   <имя_поля1>;
    <тип2>   <имя_поля2>;
    …
    <типN>   <имя_поляN>;
  };
```

где <тип1>, <тип2>, … ,<типN> – любые основные типы (int, char, float, и т. д.), массив, указатель, структура, объединение.

Пример

```
 struct  Student
 {
        char *fio;        //Фамилия     – указатель на char
  char  Adress[40];     //Адрес       – строка
  int   Age;            //Возраст     – целое
  int   oc[4];          //Оценки      – целочис. массив
        float  sr;      //Средний балл – вещественное
 } ;
```

Шаблон структуры определяет новый тип, имя которого можно использовать наряду со стандартными типами

Синтаксис описания структурной переменной (записи):

struct <имя_шаблона> <имя_переменной>;

или

struct <имя_шаблона> <список имен переменных>;

Ключевое слово *struct* можно опускать в таком описании.

Примеры

```
 struct  Student S;
 Student S, S1, S2;
```

Компилятор выделяет под структурную переменную число байтов, не всегда равное сумме длин отдельных полей из-за влияния дополнительного фактора внутреннего представления структурных переменных, называемого выравниванием. Точное число выделенных байтов определяется с помощью стандартной функции ***sizeof***:

sizeof(*struct*<имя_шаблона>);

Например, *sizeof* (*struct* Student); возвращает 56 байт.

Можно совмещать описание шаблона структуры и структурной переменной в одном предложении. Разрешается выполнять инициализацию полей структурной переменной при ее описании. Для инициализации структурной переменной значения ее полей перечисляются в фигурных скобках в порядке, соответствующему описанию полей.

Пример совмещения описания шаблона, описания структурных переменных и инициализации полей переменной в одном предложении:

```
struct Student                  //Описание шаблона структуры
    { char *fio;                //Фамилия     – указатель на char
      char  Adress[40];         //Адрес       – строка
```

```
    int     Age;            //Возраст       – целое
    int     oc[4];          //Оценки        – целочис. массив
            float   sr;             //Средний балл - вещественное
  }
 S, S1,                     //Описание переменных S,S1,S2
S2={ "Иванов",              //и инициализация полей S2
     "г.  Жлобин, ул. Цветочная, д.1, кв.43 ",
      1991,
      {9,6,8,5},      //Инициализация поля - массива  (int oc[4]; )
       0
    };
```

Для переменных одного итого же структурного типа определена ***операция присваивания***. Например, можно записать S1=S2; После выполнения такого оператора присваивания значения полей структуры S1 будут совпадать со значениями соответствующих полей структуры S2.

Вложенные структуры

Структуру, имеющую поле, являющееся структурой, называют вложенной структурой. Шаблон вкладываемой структуры должен быть известен до описания вложенной структуры. Например,

```
struct fios                     //Описание вкладываемой структуры
       { char *f,*im,*ot; };

typedef struct Student      //Описание собственного типа MY_Student
         { struct fios fio;         //поле  - структура
           char Adress[40];
                 int  Age;
           int  oc[4];
           float sr;
         } MY_Student;

/* Описание структурной переменной S с ее инициализаций */
 MY_Student S={  {"Иванов", "Петр", "Сидорович"},
                  "Река Сож",
                   1988,
                   {5,4,5,4},
                    0
                };
```

Доступ к отдельным полям структурной переменной

Для доступа к полям структуры используется операция выбора . (точка), с помощью которой конструируется ***составное имя***:

< имя_структурной_переменной >.<имя поля> – *составное имя*, где . – операция *выбора* или *ссылки на поле* (обычная точка).

Например, S.Age, S.oc[0], S.oc[i]

Доступ к компонентам поля, являющегося структурой, осуществляется через две операции выбора. Например,

S.fio.f, S.fio.im, S.fio.ot

Структурные переменные и указатели

Синтаксис описания указателя:

<тип>*<имя_переменной>;

Таким образом, для описания указателя на структуру должен быть создан новый тип.

-> – операция доступа к полям структурной переменной через указатель (минус больше).

Пример

```
 typedef struct Student
     { char   fio[30];
       char    Adress[40];
       int      Age;
       int      oc[4];
       float    sr;
     } MY_Student;  //имя нового типа
  MY_Student *S3            //Указатель на структуру
```

Примеры обращения к полям:

S3->Age, S3->oc[0], S3->oc[i], S3->fio, S3-> sr

Пример программ работы со структурами:

```
#include <stdio.h>
main()
{
/*              Описание шаблона структуры                    */
struct Student
     { char *fio;            //Фамилия      – указатель на char
       char   Adress[40];    //Адрес        – строка
       int    Age;           //Возраст      – целое
```

```
        int     oc[4];          //Оценки          – целочис. массив
                float   sr;             //Средний балл – вещественное
        } ;
 struct Student S;              //Описание структурной переменной S
       int i;
 float sr;
 S.fio="Петушков";       //Присваивание полю значения
 printf("Введите адрес студента %sa ",S.fio);
 gets(S.Adress);              //Ввод значения поля
 S.Age=1987;                  //Присваивание полю значения
 S.oc[0]=3;  S.oc[1]=5; S.oc[2]=4; S.oc[3]= S.oc[2];
 sr=0;                        //Вычисление среднего балла
 for(i=0;i<=3;i++)
     sr=sr+S.oc[i];
 sr=sr/4;
 S.sr=sr;                  //Присваивание полю вычисленного значения
/*    Вывод полей записи   S */
 printf(" Средний балл студента %sa", S.fio);
 printf(" %d года рождения,\n проживающего по адресу: %s,",
                        S.Age,S.Adress);
 printf(" равен %5.2f\n", S.sr);
 fflush(stdin);   getchar();
 return(0);
}
```

Из примера видно, что имя поля структуры и имя переменной могут совпадать, поскольку у них разные области видимости. Более того, поля разных структур могут совпадать. Но делать так не рекомендуется, поскольку себя запутать легче, чем компилятор.

Вид экрана после выполнения приведенной программы:

Введите адрес студента Петушкова ул. Солнечная, д. 1, кв. 4
Средний балл студента Петушкова 1987 года рождения,
проживающего по адресу: ул. Солнечная, д. 1, кв. 4, равен 4.00

Здесь серым цветом выделен текст, который пользователь набрал на клавиатуре.

Массивы структур (записей)

Описание массива структур не отличается от описания массива обычных переменных.

Пример

```
Struct Man
        {
          char   fio[31];          //ФИО
           int    year;            //Год рождения
          float  pay;             //Оклад
        };
Man d[3], *p=d;     //массив структур из трех элементов и
                          //указатель, инициализированный адресом
                          //первого элемента массива
```

Примеры обращения к полям:

d[i].year, (*(d+i)).year, (p+i)->year

```
/*  Пример. Поиск в массиве структур, вводимых с клавиатуры  */
/*   Фамилии сотрудников, имеющих оклад выше среднего  */

#include <stdio.h>
#include <string.h>
#include <iostream.h>

void main()
       {
 const int lfio=20, lpay=5, lo=7,     //длины полей фио, г.рожд., оклада
ldb=10;
 struct Man
     {  char   fio[lfio+1];     //фио
        int    year;           //год рожд.
        float  pay;            //оклад
     };
 Man db[ldb];                  //массив структур
 int i, n;
 float s;  // Средний оклад
 puts("Число записей?(1<n<=10)");   cin>>n;
/*Ввод массива записей */
```

```
  for(i=0; i<n; i++)
     {
      puts("Фамилия? ");          cin>>db[i].fio;
      puts("Год рождения? ");   cin>>db[i].year;
      puts("Оклад? ");              cin>>db[i].pay;
     }
/* Вывод массива записей в форме таблицы*/
  puts(" Список сотрудников");
  puts(" ┌──────────────────┬──────┬──────┐ ");
  puts(" │ ФИО              │ г.р. │Оклад │ ");
  puts(" ├──────────────────┼──────┼──────┤ ");
  for(i=0; i<n; i++)
    printf("|%-20s|% 5d | % 7.2f |\n", db[i].fio, db[i].year, db[i].pay);
  puts(" └──────────────────┴──────┴──────┘ ");
  puts("Фамилии сотрудников, имеющих оклад выше среднего");
//Вычисление среднего оклада
  s=0;
  for(i=0; i<n; i++)
      s+=db[i].pay; //s=s+db[i].pay;
  s/=n;   //s=s/n;

 //Определение и вывод фамилий
  for(i=0; i<n; i++)
    if(db[i].pay>s)  printf("%-s\n", db[i].fio);
  fflush(stdin); getchar();
}
```

Пример. Описать структуру с именем Regrz, содержащую следующие поля:

1. ФИО – студента.
2. Номер группы.
3. Название предмета.
4. Дата поступления работы.
5. ФИО – преподавателя, проверяющего работу.
6. Оценка о зачете (зачет, незачет).

Написать программу (Регистрация контрольных работ заочников), выполняющую следующие действия:

– ввод с клавиатуры данных в массив;
– добавлять записи в массив;
– выводить на экран все записи в виде таблицы;
– удалять из массива запись с заданным номером;
– осуществлять поиск в соответствии с запросами:
  – удалять все записи по конкретной группе;
  – изменять фамилию заданного студента;
  – вывести все данные о зачтенных контрольных работах конкретного студента;
– результаты поиска выводить на экран в виде таблицы.

Решение

Выполнил студент
Группы ИТ-11
Иваненко Н. И.
Проверил преподаватель
Кравченко О. А.

Для упрощения основного алгоритма можно его разделить на несколько подзадач:
– добавление записей (input);
– вывод записей в виде таблицы (output);
– удаление записи по номеру (delnum);
– удаление записей по конкретной группе (delgr);
– замена фамилии студента (change);
– вывод всех зачтенных работ конкретного лица (outstud).

Реализуем первую подзадачу. Для этого разработаем алгоритм ввода/добавления записей. Его графическая схема изображена на рис. 2.1.

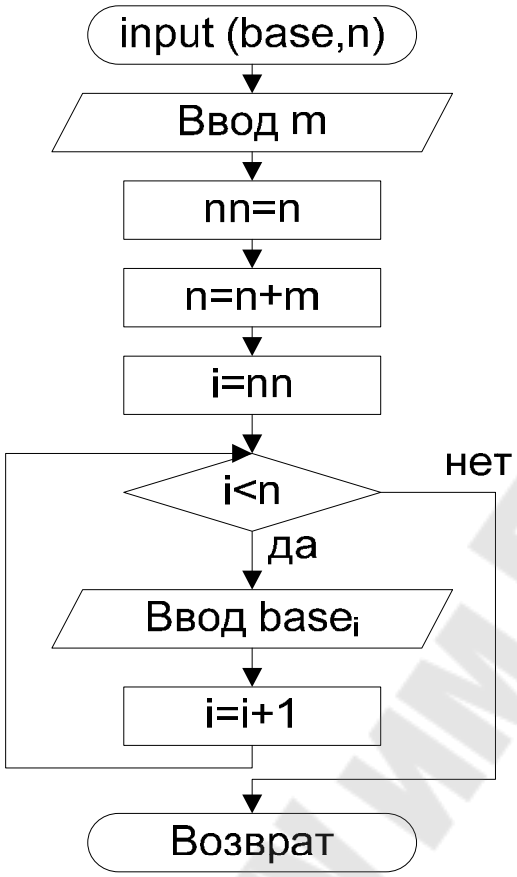

Рис. 2.1. Алгоритм ввода/добавления записей в массив записей

В таблице 2.2 указаны типы используемых в алгоритме переменных.

Таблица 2.2

Имя переменных в условии	**Имя переменных на С**	**Тип**	**Комментарий**
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Формальный параметр – количество записей в массиве
	i	int	Параметр цикла
	m	int	Количество добавляемых записей
	nn	int	Начальное
	k	int	Параметр для ввода по полям

Реализуем вторую подзадачу. Разработаем алгоритм вывода записей. Его графическая схема изображена на рис. 2.2.

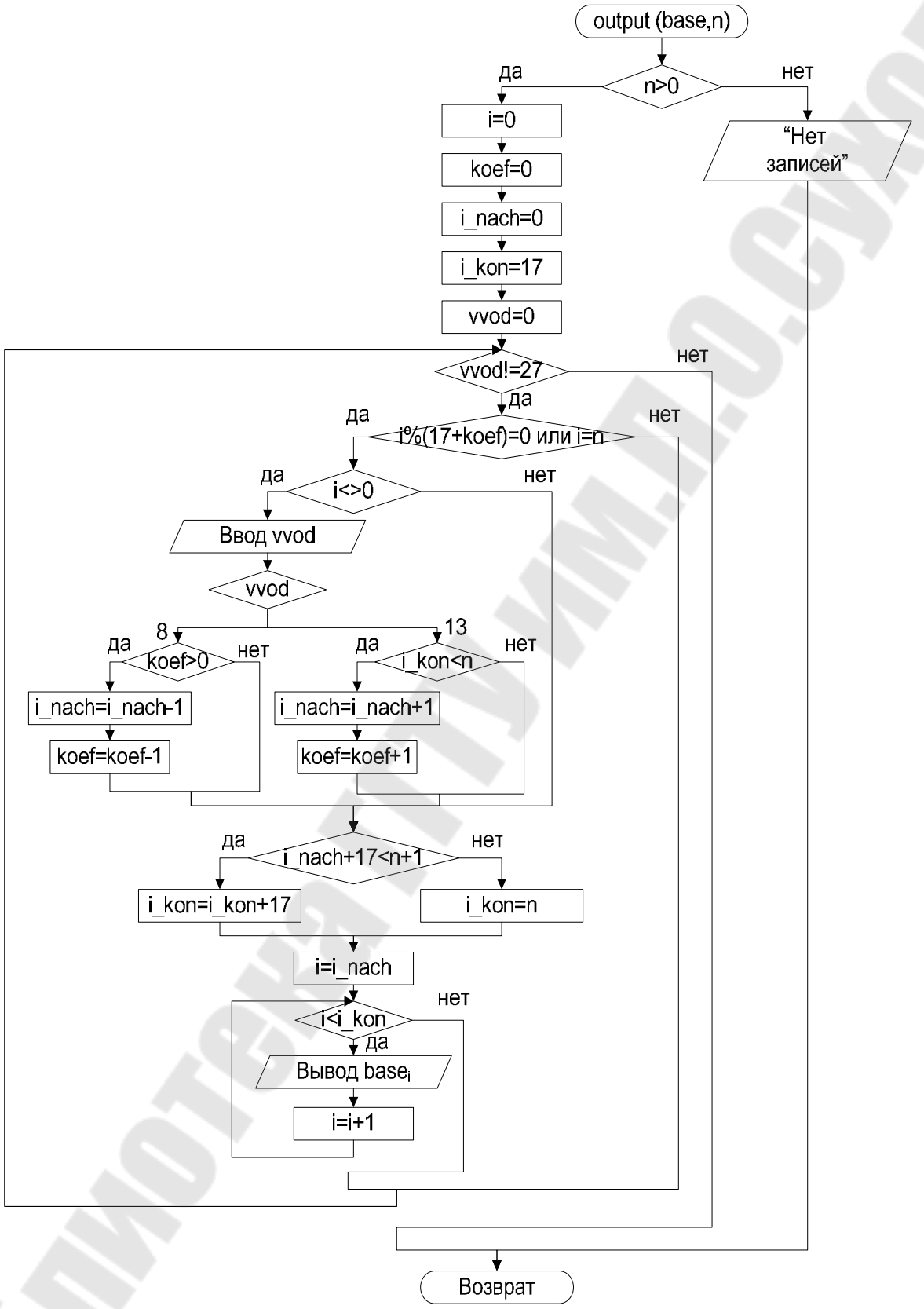

Рис. 2.2. Алгоритм вывода записей

В таблице 2.3. указаны типы используемых в алгоритме переменных.

Таблица 2.3

Имя переменных в условии	Имя переменных на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	i_nach	int	Номер начальной записи
	i_kon	int	Номер конечной записи
	vvod	int	Параметр для организации прокрутки
	koef	int	Параметр для вычисления количества записей на страницу

Реализуем третью подзадачу. Разработаем алгоритм удаления записи по номеру. Его графическая схема изображена на рис. 2.3.

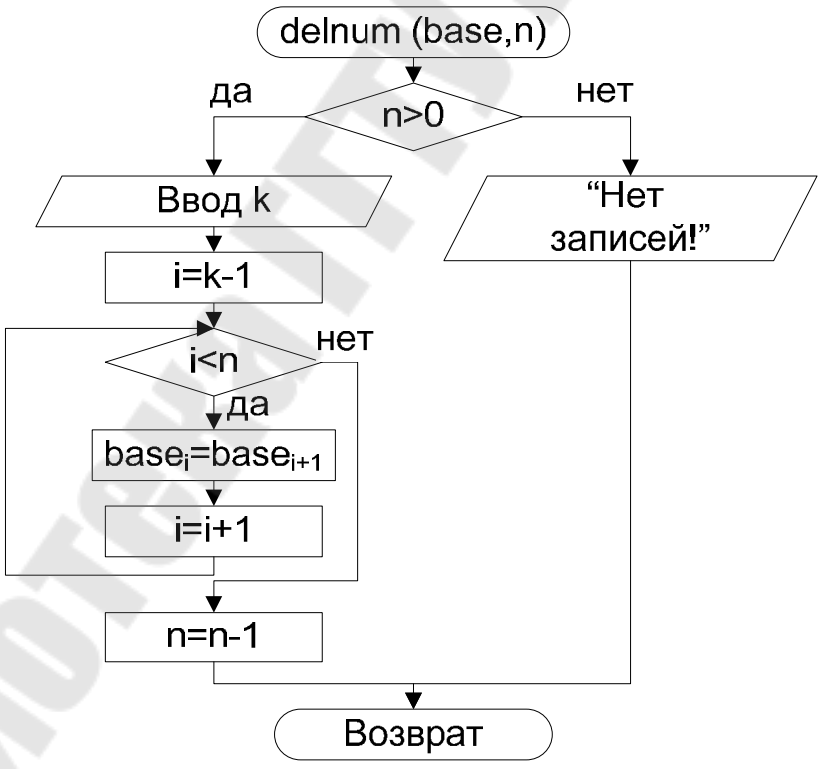

Рис. 2.3. Алгоритм удаления записи по номеру

В таблице 2.4 указаны типы используемых в алгоритме переменных.

Таблица 2.4

Имя переменных в условии	Имя переменных на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	k	int	Номер удаляемой записи

Реализуем четвертую подзадачу. Разработаем алгоритм удаления записи по группе. Его графическая схема изображена на рис. 2.4.

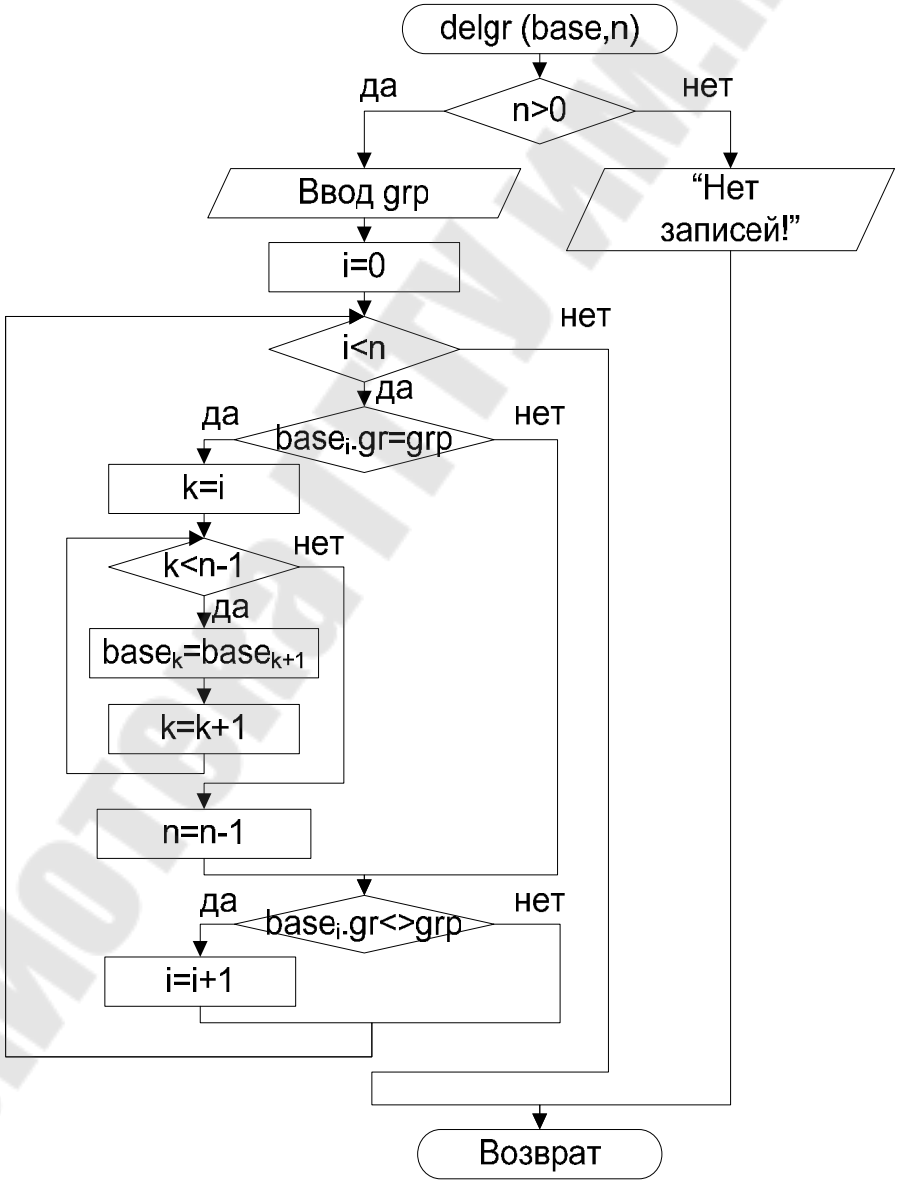

Рис. 2.4. Алгоритм удаления записи по группе

В таблице 2.5 указаны типы используемых в алгоритме переменных.

Таблица 2.5

Имя переменных в условии	Имя переменных на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	k	int	Номер удаляемой записи
	grp	char	Номер группы

Реализуем пятую подзадачу. Разработаем алгоритм смены фамилии конкретного студента. Его графическая схема изображена на рис. 2.5. В таблице 2.6 указаны типы используемых в алгоритме переменных.

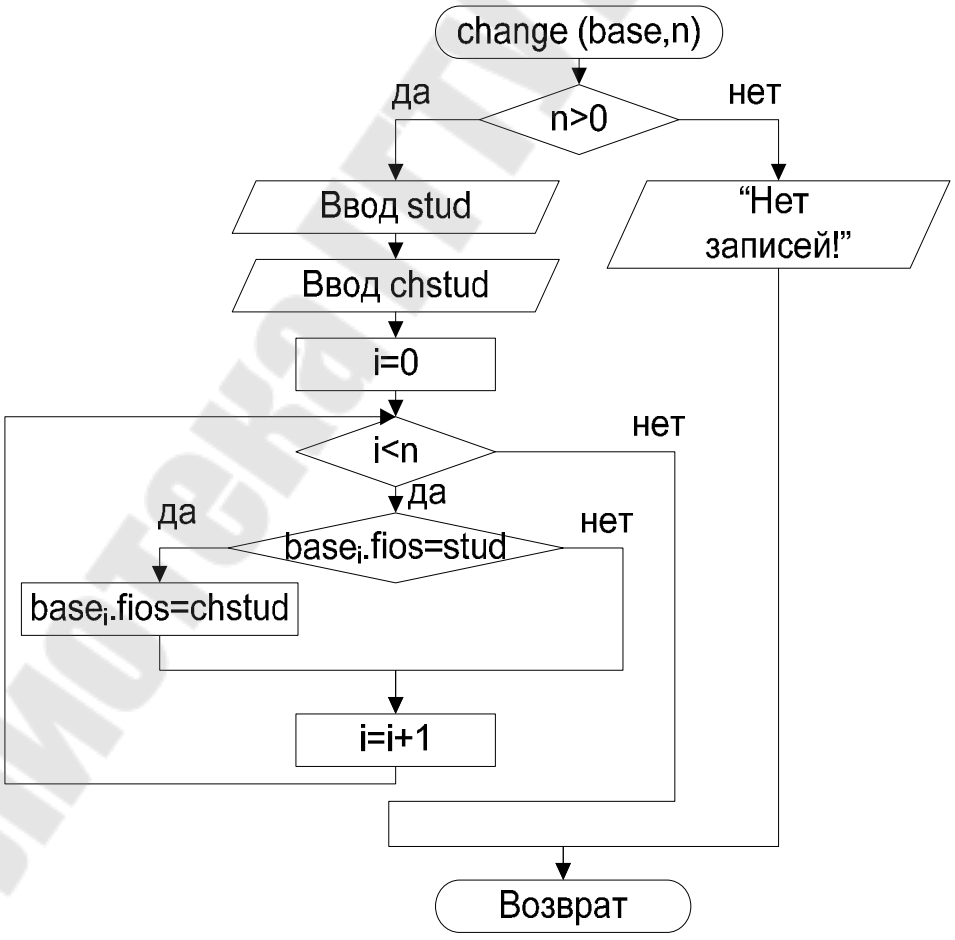

Рис. 2.5. Алгоритм смены фамилии конкретного студента

Таблица 2.6

Имя переменных в условии	Имя переменных на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	j	int	Параметр цикла
	stud	char	Начальная фамилия студента
	chstud	char	Конечная фамилия студента

Реализуем шестую подзадачу. Разработаем алгоритм вывода зачтенных работ конкретного студента. Его графическая схема изображена на рис. 2.6. В таблице 2.7 указаны типы используемых в алгоритме переменных.

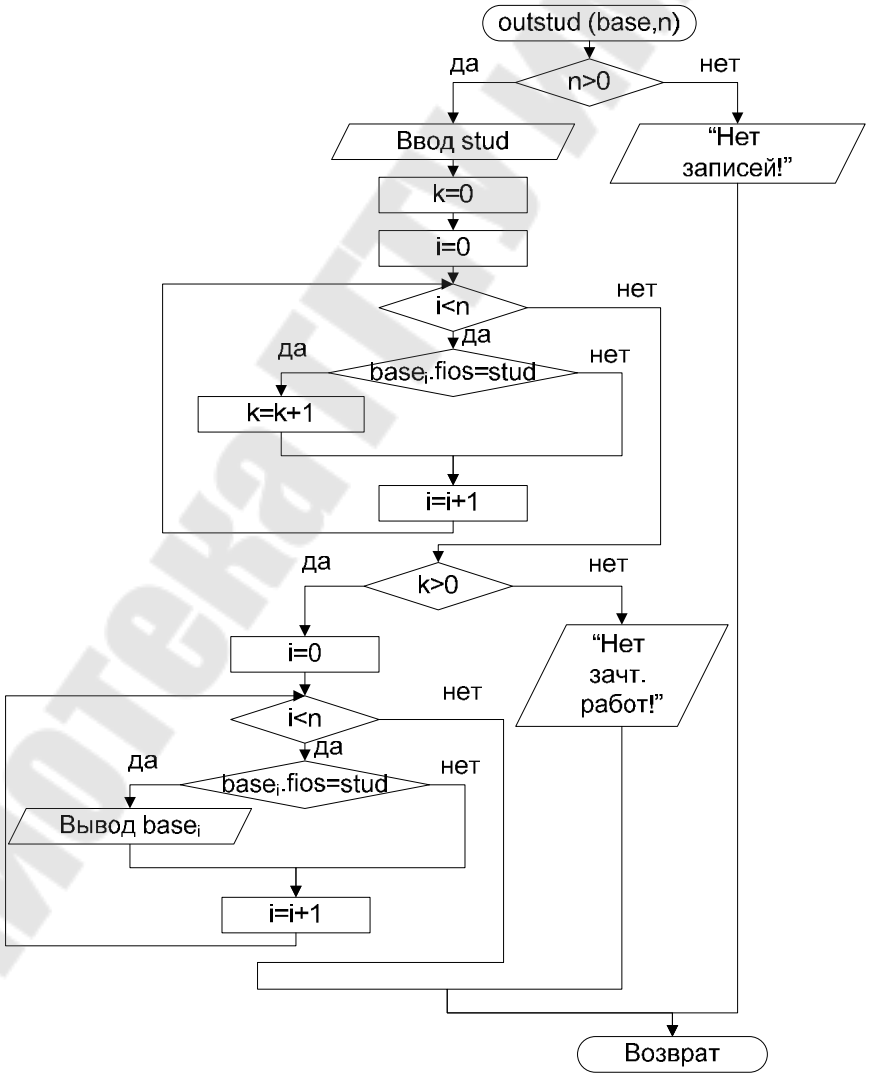

Рис. 2.6. Алгоритм вывода зачтенных работ конкретного студента

Таблица 2.7

Имя переменных в условии	Имя переменных на С	Тип	Комментарий
	base	zaochn	Формальный параметр – указатель на структуру
	n	int	Количество записей во всей структуре
	i	int	Параметр цикла
	k	int	Параметр для подсчета зачтенных работ
	stud	char	Фамилия студента

Основной алгоритм выглядит следующим образом.

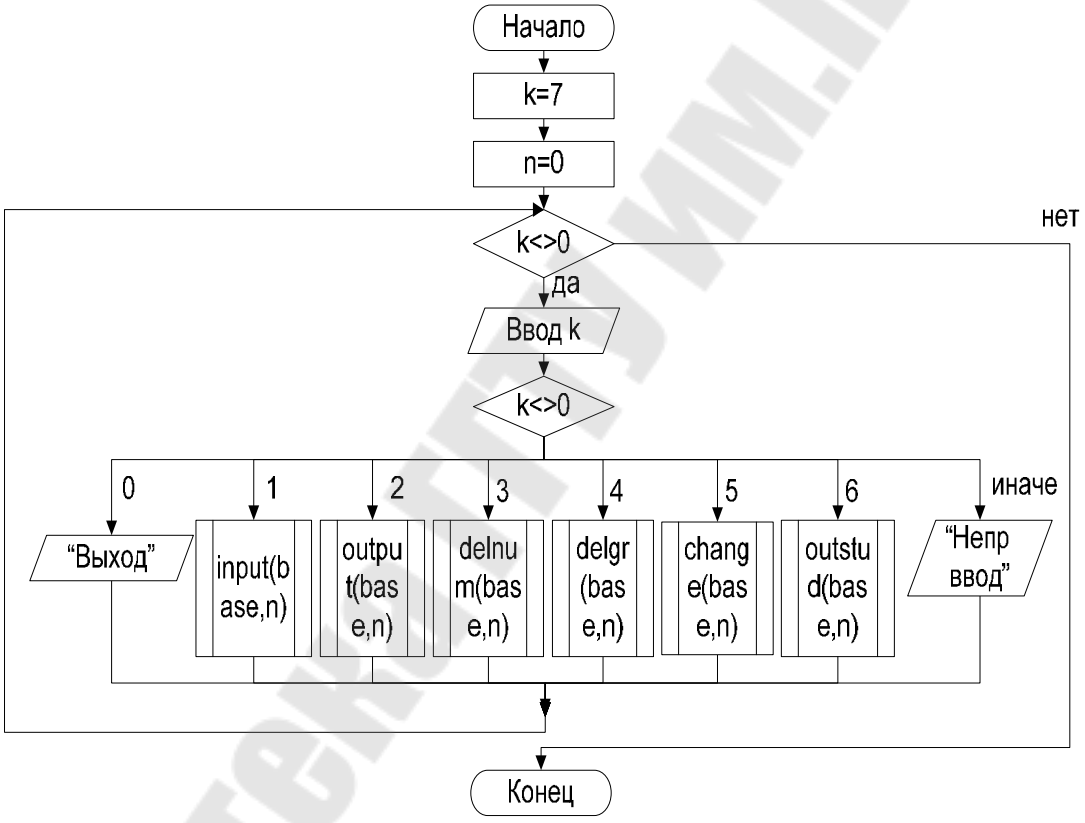

Рис. 2.7. Основной алгоритм (k – параметр для организации меню)

Листинг программы:

```
/* Выполнил студент группы ИТ-11 Иваненко Никита Игоревич */
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct Regrz
```

```
                {
                 char fios[21];
                 char gr[6];
                 char subj[7];
                 char date[10];
                 char fiop[21];
                 char zach[8];
                } zaochn;
void input (zaochn *base, int *n);
void output (zaochn *base, int n);
void delnum (zaochn *base, int *n);
void delgr (zaochn *base, int *n);
void change (zaochn *base, int n);
void outstud (zaochn *base, int n);
main()
{
 int k,n;
 zaochn base[100];
 k=7;
 n=0;
 while (k!=0)
        {
         textcolor(11);
         textbackground(1);
         clrscr();
         puts("\t\t╔═══════════════════════════════════════════════════╗");
         puts("\t\t║                Главное меню.                      ║");
         puts("\t\t║              Выберите действие:                   ║");
         puts("\t\t╠═══╤═══════════════════════════════════════════════╣");
         puts("\t\t║ 1 │      Добавление записи(-сей)                  ║");
         puts("\t\t╟───┼───────────────────────────────────────────────╢");
         puts("\t\t║ 2 │  Вывод записей на экран в виде таблицы        ║");
         puts("\t\t╟───┼───────────────────────────────────────────────╢");
         puts("\t\t║ 3 │      Удаление записи по номеру                ║");
         puts("\t\t╟───┼───────────────────────────────────────────────╢");
         puts("\t\t║ 4 │ Удаление записи(-сей) по конкретной группе    ║");
         puts("\t\t╟───┼───────────────────────────────────────────────╢");
         puts("\t\t║ 5 │      Замена фамилии студента                  ║");
         puts("\t\t╟───┼───────────────────────────────────────────────╢");
         puts("\t\t║ 6 │ Вывод всех зачтенных работ конкретного лица   ║");
```

```
puts("\t\t╟──────┼──────────────────────────────────────────────────────────
──────────╢");
        puts("\t\t╟  0   │       Завершение работы программы                 ╢");
        puts("\t\t╚═══════════════════════════════════════════════════════════");
        printf ("\t\tВаш выбор(0-6):");
        scanf ("%d",&k);
        switch (k)
                {
                case 1 : {
                        input(base,&n);
                        puts("Добавление записи(-сей) окончено. Нажмите лю-
бую клавишу для продолжения...");
                        getch();
                        }
                break;
                case 2 : {
                        output(base,n);
                        puts("Вывод записи(-сей) окончен. Нажмите любую
клавишу для продолжения...");
                        getch();
                        }
                break;
                case 3 : {
                        delnum(base,&n);
                        puts("Удаление записи окончено. Нажмите любую кла-
вишу для продолжения...");
                        getch();
                        }
                break;
                case 4 : {
                        delgr(base,&n);
                        puts("Удаление записи(-сей) окончено. Нажмите лю-
бую клавишу для продолжения...");
                        getch();
                        }
                break;
                case 5 : {
                        change(base,n);
                        puts("Изменение записи(-сей) окончено. Нажмите лю-
бую клавишу для продолжения...");
                        getch();
```

```
                }
            break;
            case 6 : {
                outstud(base,n);
                puts("Вывод записи(-сей) окончен. Нажмите любую
клавишу для продолжения...");
                getch();
                }
            break;
            case 0 : {
                textcolor(14);
                clrscr();
                puts("Выход... Нажмите любую клавишу для продол-
жения...");
                getch();
                }
            break;
            default :  {
                 textcolor(12);
                 clrscr();
                 puts("Неправильный ввод! Вводите число от 0 до 6!");
                 getch();
                }
            break;
           }
      }
 return(0);
}

void input (zaochn *base, int *n)
{           //Функция для ввода/добавления записей в структуру
 int i,m,nn,k;
 textcolor(10);
 clrscr();
 puts("Введите кол-во записей:");
 scanf("%d",&m);
 nn=*n;
 *n+=m;
 for (i=nn;i<*n;i++)
      {
       if (i%21==0||nn!=0&&i==nn)
            {
```

```
                clrscr();
                k=4;
puts("┌─────┬──────────────────────────┬──────┬──────────┬──────────┬──────────┐");
                puts("│  №  │     ФИО студента         │  Гр. │Предм.│Дата п.р.│ ФИО преподавателя │  Оц.  │");
puts("├─────┼──────────────────────────┼──────┼──────────┼──────────┼──────────┤");
                }
            printf("│%3d│",i+1);
            scanf("%s",&base[i].fios);
            gotoxy(26,k);
            printf("│");
            scanf ("%s",&base[i].gr);
            gotoxy(32,k);
            printf("│");
            scanf ("%s",&base[i].subj);
            gotoxy(39,k);
            printf("│");
            scanf ("%s",&base[i].date);
            gotoxy(49,k);
            printf("│");
            scanf("%s",&base[i].fiop);
            gotoxy(70,k);
            printf("│");
            scanf ("%s",&base[i].zach);
            gotoxy(78,k);
            printf("│\n");
            k++;
           }
puts("└─────┴──────────────────────────┴──────┴──────────┴──────────┴──────────┘");
        }

void output (zaochn *base, int n)
{           //Функция для вывода записей в виде таблицы
 int i,i_nach,i_kon,vvod,koef;
 clrscr();
 if (n>0)
```

```
{
i=0;
koef=0;
i_nach=0;
i_kon=17;
vvod=0;
while (vvod!=27)
        {
        if (i%(17+koef)==0||i==n)
                {
                if (i!=0)
                   {
puts(" └─────┴──────────────────────────────┴────────┴─────────┴─────
────────┴───────────────────────────┴───────────┘ ");
                   if (i_nach>0) printf("Нажмите \"Backspace\" для
прокрутки текста вверх.\n");
                   if (i<n) printf("Нажмите \"Enter\" для прокрутки
текста вниз.\n");
                   printf("Нажмите \"Esc\" для выхода.\n");
                   vvod=getch();
                   switch (vvod)
                        {
                        case 8 : if (koef>0)
                                    {
                                    i_nach=i_nach-1;
                                    koef--;
                                    }
                        break;
                        case 13 : if (i_kon<n)
                                    {
                                    i_nach=i_nach+1;
                                    koef++;
                                    }
                        break;
                        }
                   }
                clrscr();
puts(" ┌─────┬──────────────────────────────┬────────┬─────────┬─────
────────┬───────────────────────────┬───────────┐ ");
          puts("│ № │   ФИО студента   │ Гр. │Предм.│Дата п.р.│
ФИО преподавателя │ Оц. │");
```

```
puts("┌──┬──────────────────────────┬──────┬──────┬──────┐
└──┴──────────────────────────┴──────┴──────┴──────┘");
                }
            if (i_nach+17<n+1) i_kon=i_nach+17;
                else i_kon=n;
            for(i=i_nach;i<i_kon;i++) printf("│%3d│%-20s│%-5s│%-6s│%-9s│%-20s│%-7s│\n",i+1,base[i].fios, base[i].gr, base[i].subj, base[i].date, base[i].fiop, base[i].zach);
            if (vvod==27)
puts("└──┴──────────────────────────┴──────┴──────┴──────┘");
            }
    }
      else puts ("Нет записей для вывода!");
}

void delnum (zaochn *base, int *n)
{           //Функция для удаления записи по номеру
 int i,j,k;
 textcolor(13);
 clrscr();
 if (*n>0)
     {
       puts("Введите № удаляемой записи:");
       scanf("%d",&k);
       for (i=k-1;i<*n;i++)
            for(j=0;j<22;j++)
                {
                 if (j<22)
                    {
                     base[i].fios[j]=base[i+1].fios[j];
                     base[i].fiop[j]=base[i+1].fiop[j];
                    }
                 if (j<11) base[i].date[j]=base[i+1].date[j];
                 if (j<9)  base[i].zach[j]=base[i+1].zach[j];
                 if (j<8)  base[i].subj[j]=base[i+1].subj[j];
                 if (j<7)  base[i].gr[j]=base[i+1].gr[j];
                }
       *n=*n-1;
     }
   else puts ("Нет записей для удаления!");
}
```

```
void delgr (zaochn *base, int *n)
        {          //Функция для удаления записей по конкретной группе
         int i,j,k;
         char grp[6];
         textcolor(11);
         clrscr();
         if (*n>0)
             {
              puts("Введите название группы для удаления записи(-сей):");
              scanf("%s",&grp);
              i=0;
              while (i<*n)
              {
                if (strcmp(base[i].gr,grp)==0)
                                {
                                  for (k=i;k<*n-1;k++)
                                        for (j=0;j<22;j++)
                                                {
                                                 if (j<22)
                                                    {
                                                      base[k].fios[j]=base[k+1].fios[j];
                                                      base[k].fiop[j]=base[k+1].fiop[j];
                                                    }
                                                 if                                    (j<11)
base[k].date[j]=base[k+1].date[j];
                                                 if                                     (j<9)
base[k].zach[j]=base[k+1].zach[j];
                                                 if                                     (j<8)
base[k].subj[j]=base[k+1].subj[j];
                                                 if (j<7)  base[k].gr[j]=base[k+1].gr[j];
                                                }
                                    *n=*n-1;
                                }
                if (strcmp(base[i].gr,grp)!=0) i++;
             }
             }
        else puts ("Нет записей для удаления!");
        }

        void change (zaochn *base, int n)
        {               //Функция для смены фамилии студента с заданной на за-
данную
```

```
        int i,j;
        char stud[21],chstud[21];
        textcolor(12);
        clrscr();
        if (n>0)
            {
             puts("Введите начальную фамилию студента:");
             scanf("%s",&stud);
             puts("Введите фамилию студента для смены:");
             scanf("%s",&chstud);
             for (i=0;i<n;i++) if (strcmp(base[i].fios,stud)==0) for (j=0;j<22;j++)
if (strlen(chstud)+1>j) base[i].fios[j]=chstud[j];
                                                                    else
base[i].fios[strlen(chstud)+1]='\0';
            }
            else puts ("Нет записей для изменения!");
        }

        void outstud (zaochn *base, int n)
        {          //Функция для вывода зачтенных работ конкретного лица
         int i,k;
         char stud[21];
         clrscr();
         if (n>0)
            {
             puts("Введите фамилию студента для вывода записи(-сей):");
             scanf("%s",&stud);
             k=0;
             for                    (i=0;i<n;i++)                    if
(strcmp(base[i].fios,stud)==0&&strcmp(base[i].zach,"зачет")==0) k++;
             if (k>0)
           {

puts("┌──┬──────────────────────┬──────┬─────────┬──────────┬──────────────────┬──────┐");
            puts("│ № │     ФИО студента     │ Гр. │Предм.│Дата п.р.│   ФИО
преподавателя  │  Оц.  │");
puts("├──┼──────────────────────┼──────┼─────────┼──────────┼──────────────────┼──────┤");
            k=1;
            for (i=0;i<n;i++)
```

```
        if      (strcmp(base[i].fios,stud)==0&&strcmp(base[i].zach,"зачет")==0)
printf(" │%3d │%-20s │%-5s │%-6s │%-9s │%-20s │%-7s │\n",k,base[i].fios,
base[i].gr, base[i].subj, base[i].date, base[i].fiop, base[i].zach);

puts(" └────┴──────────────────────────────────────┴──────┴──────┴─────
──────┴──────────────────────────────────────┴───────┘ ");
          }
         else printf ("У студента %s нет зачтенных работ!\n",stud);
           }
         else puts ("Нет записей для вывода!");
       }
```

Тесты

Был создан массив из 25 записей:

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	физ	13.12.2006	Кравченко	незачет
4	Петров	ОС	инф	14.01.2007	Кротенок	зачет
5	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
6	Петров	ОС	инф	17.02.2007	Ковалев	незачет
7	Петров	ОС	физ	18.02.2007	Ярчак	зачет
8	Петров	ОС	мат	21.03.2007	Авакян	незачет
9	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
10	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
11	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
12	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
13	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
14	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
15	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
16	Иванов	МК	физ	14.01.2007	Курбатова	зачет
17	Иванов	МК	физ	16.01.2007	Кравченко	зачет
18	Иванов	МК	инф	17.02.2007	Кротенок	зачет
19	Иванов	МК	физ	18.02.2007	Дробышевский	зачет
20	Иванов	МК	инф	21.03.2007	Ковалев	незачет
21	Иванов	МК	физ	22.03.2007	Ярчак	незачет
22	Иванов	МК	мат	23.03.2007	Авакян	незачет
23	Иванов	МК	физ	01.04.2007	Курбатова	зачет
24	Рыбин	ОС	инф	16.01.2007	Кравченко	зачет
25	Рыбин	ОС	инф	18.02.2007	Кротенок	незачет

1. Удаление записи по номеру.
   k=20

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	физ	13.12.2006	Кравченко	незачет
4	Петров	ОС	инф	14.01.2007	Кротенок	зачет
5	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
6	Петров	ОС	инф	17.02.2007	Ковалев	незачет
7	Петров	ОС	физ	18.02.2007	Ярчак	зачет
8	Петров	ОС	мат	21.03.2007	Авакян	незачет
9	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
10	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
11	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
12	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
13	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
14	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
15	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
16	Иванов	МК	физ	14.01.2007	Курбатова	зачет
17	Иванов	МК	физ	16.01.2007	Кравченко	зачет
18	Иванов	МК	инф	17.02.2007	Кротенок	зачет
19	Иванов	МК	инф	21.03.2007	Ковалев	незачет
20	Иванов	МК	физ	22.03.2007	Ярчак	незачет
21	Иванов	МК	мат	23.03.2007	Авакян	незачет
22	Иванов	МК	физ	01.04.2007	Курбатова	зачет
23	Рыбин	ОС	инф	16.01.2007	Кравченко	зачет
24	Рыбин	ОС	инф	18.02.2007	Кротенок	незачет

2. Вывод зачтенных работ конкретного лица.
   stud=’Петров’

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Петров	ОС	мат	11.11.2006	Авакян	зачет
2	Петров	ОС	физ	12.11.2006	Курбатова	зачет
3	Петров	ОС	инф	14.01.2007	Кротенок	зачет
4	Петров	ОС	физ	16.01.2007	Дробышевский	зачет
5	Петров	ОС	физ	18.02.2007	Ярчак	зачет

3. Удаление записей по конкретной группе.

grp='ОС'

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
2	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
3	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
4	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
5	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
6	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
7	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
8	Иванов	МК	физ	14.01.2007	Курбатова	зачет
9	Иванов	МК	физ	16.01.2007	Кравченко	зачет
10	Иванов	МК	инф	17.02.2007	Кротенок	зачет
11	Иванов	МК	физ	18.02.2007	Дробышевский	зачет
12	Иванов	МК	инф	21.03.2007	Ковалев	незачет
13	Иванов	МК	физ	22.03.2007	Ярчак	незачет
14	Иванов	МК	мат	23.03.2007	Авакян	незачет
15	Иванов	МК	физ	01.04.2007	Курбатова	зачет

4. Смена фамилии студента.

stud='Иванов'; chstud='Петренко'

№	ФИО студента	Гр.	Предм.	Дата п.р.	ФИО преподавателя	Оц.
1	Сидоров	ПЭ	физ	11.10.2006	Курбатова	зачет
2	Сидоров	ПЭ	физ	12.10.2006	Кравченко	зачет
3	Сидоров	ПЭ	инф	13.10.2006	Кротенок	незачет
4	Сидоров	ПЭ	физ	14.10.2007	Дробышевский	зачет
5	Сидоров	ПЭ	инф	11.11.2006	Ковалев	незачет
6	Сидоров	ПЭ	физ	12.11.2006	Ярчак	незачет
7	Сидоров	ПЭ	мат	13.12.2006	Авакян	незачет
8	Петренко	МК	физ	14.01.2007	Курбатова	зачет
9	Петренко	МК	физ	16.01.2007	Кравченко	зачет
10	Петренко	МК	инф	17.02.2007	Кротенок	зачет
11	Петренко	МК	физ	18.02.2007	Дробышевский	зачет
12	Петренко	МК	инф	21.03.2007	Ковалев	незачет
13	Петренко	МК	физ	22.03.2007	Ярчак	незачет
14	Петренко	МК	мат	23.03.2007	Авакян	незачет
15	Петренко	МК	физ	01.04.2007	Курбатова	зачет

3. СОРТИРОВКА

3.1. Общие сведения

Сортировка – это операция, упорядочивающая последовательность (массив) элементов по ключам. **Ключ** – это некоторое числовое свойство элемента массива. То есть каждому элементу массива ставится в соответствие некоторое число, называемое ключом элемента.

Если мы имеем числовой массив, то ключ элемента – это сам элемент. В этом случае сортировка массива – это упорядочивание массива (перестановка его элементов) таким образом, чтобы получилась неубывающая или невозрастающая последовательность.

Если каждый элемент исходного массива представляет собой слова, составленные из букв русского алфавита, то ключ, по которому может быть упорядочен массив, связывается с порядковым номером буквы в алфавите. По этому принципу упорядочиваются слова в словарях – из двух слов первым помещается то слово, ключ которого меньше. Здесь принимается, что отсутствие буквы (т. е. пустая строка) имеет меньший ключ, чем любая другая буква. Так слово «студент» помещается в словаре перед словом «студентка».

Если слова состоят из букв разных алфавитов и цифр, как, например, имена файлов и папок, то любая цифра имеет меньший ключ, чем любая буква, а любая латинская буква имеет меньший ключ по сравнению с любой русской буквой. При сортировке таких имен в качестве ключа используется код символа в некоторой кодовой таблице. По этому принципу, например, отсортированы имена встроенных функций MS Excel в категории «Полный алфавитный перечень». Этот принцип упорядочивания еще называют лексикографическим порядком или расширенным алфавитом.

Разработкой различных алгоритмов сортировки информации, хранящейся в оперативной памяти компьютера или на его жестком диске, программисты занимаются уже давно. Интерес к этой проблеме обусловлен тем, что по мнению специалистов, 25 % всего времени обработки информации расходуется на сортировку данных.

Ясно, что с отсортированными данными работать легче, чем с произвольно расположенными. Когда элементы отсортированы, то проще найти нужный элемент или установить, что его нет.

Уточним терминологию.

Если элементы массива связаны отношениями $a_0 < a_1 < \ldots < a_{n-1}$, то говорят, что массив упорядочен по **возрастанию**. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \le a_1 \le ... \le a_{n-1}$, то говорят, что массив упорядочен по **неубыванию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

Если элементы массива связаны отношениями $a_0 > a_1 > ... > a_{n-1}$, то говорят, что массив упорядочен по **убыванию**. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \ge a_1 \ge ... \ge a_{n-1}$, то говорят, что массив упорядочен по **невозрастанию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

3.2. Классификация методов сортировки

Все методы сортировки можно разделить на пять групп:

1. Методы извлечения.
2. Методы включения.
3. Методы обменов.
4. Методы слияния.
5. Методы распределения.

Общая концепция **методов извлечения** заключается в следующем: из исходного массива извлекается минимальный элемент и меняется местами с первым элементом массива, затем извлекается минимальный элемент из части массива, начиная со второго элемента, и меняется местами со вторым элементом и т. д. Последний раз минимальный элемент выбирается из двух последних элементов массива. В результате получится массив, упорядоченный по неубыванию.

Различные методы извлечения отличаются объектом извлечения (минимальный или максимальный элемент) и, соответственно, объектами перестановки (первый или последний элемент), а также условием окончания процесса сортировки.

В качестве примера рассмотрим следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

Выберем минимальный элемент из всех элементов массива: a_3=0 и переставим его с первым элементом массива. Получим следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	8	4	8	0	3

Теперь выберем минимальный элемент из элементов массива a_1, a_2, …, a_7 (это элемент a_6=0) и переставим его с элементом a_1. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	3	8	4	8	1	3

Дальнейшие шаги по сортировке массива приведены ниже:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	1	8	4	8	3	3
0	0	1	3	4	8	8	3
0	0	1	3	3	8	8	4
0	0	1	3	3	4	8	8
0	0	1	3	3	4	8	8

Идея **методов включения** состоит в том, что сначала первый элемент массива рассматривается как упорядоченный массив и в этот массив включается следующий элемент исходного массива так, чтобы получился упорядоченный по неубыванию массив из двух элементов. Затем в полученный упорядоченный массив включается третий элемент массива так, чтобы опять-таки получился упорядоченный массив. Процесс продолжается до тех пор, пока не будет включен последний элемент.

Различные алгоритмы включения отличаются способами выбора элемента для включения, способами определения места включения и методами самого включения.

В качестве примера рассмотрим тот же массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

После включения элемента a_1=1 в массив, состоящий из одного элемента a_0=8, получится следующий массив:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	8	3	0	4	8	0	3

Далее включаем элемент a_2=3 в массив a_0, a_1. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3

Продолжая этот процесс, мы последовательно будем получать следующие состояния массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	8	4	8	0	3
0	1	3	4	8	8	0	3
0	0	1	3	4	8	8	3
0	0	1	3	3	4	8	8

Идея **методов обменов** состоит в следующем: в исходном массиве выбирается пара элементов и они сравниваются между собой. Если их положение не удовлетворяет требованию упорядоченности, то элементы переставляются. Затем выбирается следующая пара элементов и так до тех пор, пока не получим упорядоченный массив.

Различные алгоритмы обменов отличаются способами выбора пары элементов для сравнения и перестановки, а также условиями окончания процесса сортировки.

В качестве примера рассмотрим сортировку по неубыванию того же массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
8	1	3	0	4	8	0	3

Сравним первые два элемента и переставим их в соответствии с требованием неубывания:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	8	3	0	4	8	0	3

Теперь проделаем ту же процедуру для элементов a_1 и a_2. Получим:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3

Продолжая этот процесс, мы последовательно будем получать следующие состояния массива:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	8	0	4	8	0	3
1	3	0	8	4	8	0	3
1	3	0	4	8	8	0	3
1	3	0	4	8	8	0	3
1	3	0	4	8	0	8	3
1	3	0	4	8	0	3	8

Закончился первый просмотр массива. В результате массив не упорядочился, но максимальный элемент массива занял свое окончательное место – в конце массива. Теперь повторим процесс для элементов a_0, a_1, …, a_6.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	3	0	4	8	0	3	8
1	0	3	4	8	0	3	8
1	0	3	4	0	8	3	8
1	0	3	4	0	3	8	8

После окончания второго просмотра массива два последних элемента заняли свое окончательное место. Проведем третий просмотр массива без рассмотрения двух последних элементов.

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	4	0	3	8	8
0	1	3	0	4	3	8	8
0	1	3	0	3	4	8	8

Уже три последних элемента заняли свои окончательные места. Повторим процесс:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	3	0	3	4	8	8
0	1	0	3	3	4	8	8

Последний шаг процесса сортировки:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	1	3	3	4	8	8

Метод слияния применяется в том случае, когда имеются два (или больше) упорядоченных массива и требуется соединить исходные массивы в один общий упорядоченный массив.

Пусть, например, имеются два массива:

A_0	A_1	A_2	A_3	A_4
0	0	1	3	7

B_0	B_1	B_2	B_3
9	3	3	0

Требуется получить общий массив, упорядоченный по неубыванию.

Ясно, что для решения задачи надо сравнивать элементы двух массивов, причем в первом массиве надо выбирать элементы слева направо, а во втором массиве – справа налево.

Получим следующий массив:

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0	0	0	1	3	3	3	7	9

Метод распределения употребим в тех случаях, когда в исходном массиве имеется заданное, известное заранее, количество различных ключей (значений). Например, имеется список студентов с оценками по пятибалльной системе, полученными на экзамене. Нам известно заранее, что оценки могут быть 5, 4, 3 и 2. Поэтому для упорядочения массива по невозрастанию можно сначала выбрать все записи с оценками 5, затем с оценками 4, потом с оценками 3 и, наконец, с оценками 2.

3.3. Алгоритм сортировки методом извлечения

Упорядочим заданный целочисленный массив по **неубыванию** на основе алгоритма **извлечения минимального** элемента.

Разработаем **алгоритм** сортировки на основе извлечения минимального элемента. Сначала запишем необходимые действия в **словесной форме**:

1) найдем минимальный элемент среди всех элементов массива $a_0, a_1, ..., a_{n-1}$ и определим его номер. Пусть это будет элемент a_m;

2) поменяем местами элементы a_0 и a_m. Таким образом, минимальный элемент массива окажется на своем окончательном месте;

3) теперь найдем минимальный элемент среди элементов массива, начиная со второго $a_1, a_2, ..., a_{n-1}$ и определим его номер. Опять обозначим этот элемент a_m;

4) поменяем местами элементы a_1 и a_m. В результате два первых элемента массива окажутся на своих окончательных местах;

5) на заключительном этапе этого процесса надо выбрать минимальный элемент из двух последних элементов массива a_{n-2} и a_{n-1}. После чего этот элемент должен быть поставлен на предпоследнее место.

Теперь обобщим представленные шаги алгоритма следующим образом: нахождение минимального элемента a_m среди элементов $a_k, a_{k+1}, ..., a_{n-1}$ и последующая перестановка элементов a_m и a_k, при этом указанный процесс должен повторяться при изменении k от 0 до n–2.

Сформулированному алгоритму соответствует **графическая схема алгоритма**, приведенная на рис. 3.1.

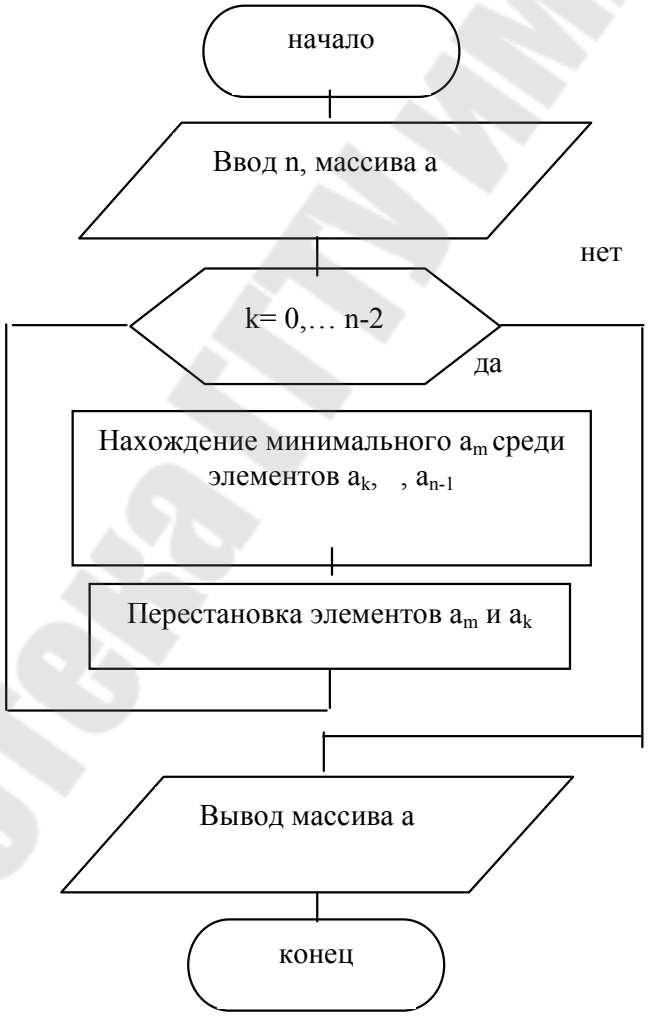

Рис. 3.1. Схема алгоритма сортировки массива по неубыванию методом извлечения

Мы не будем детализировать блок нахождения минимального элемента в отрезке массива $a_k, a_{k+1}, ..., a_{n-1}$, т. к. ранее это было изучено.

Отладку алгоритма следует провести для следующих тестов:

1) массив содержит все одинаковые элементы;

2) исходный массив упорядочен по неубыванию;

3) исходный массив упорядочен по невозрастанию;

4) элементы массива неупорядочены, причем в массиве несколько минимальных элементов.

3.4. Алгоритм сортировки методом обменов

Упорядочим заданный целочисленный массив по **неубыванию** на основе алгоритма «**пузырька**», относящегося к алгоритмам **обменов**.

Сначала запишем необходимые действия в **словесной форме**:

1) сравним элементы a_0 и a_1. Если не выполняется условие, $a_0 \le a_1$, то меняем местами эти элементы и сравниваем элементы a_1 и a_2. Так сравниваем и при необходимости меняем местами все пары исходного массива. Последней рассматривается пара a_{n-2} и a_{n-1}. Так заканчивается первый просмотр массива, при котором максимальный элемент окажется на последнем месте. Это напоминает процесс вскипания воды: первым всплывает самый большой пузырек. Именно поэтому рассматриваемый способ сортировки называется методом "пузырька";

2) второй просмотр массива опять-таки начинается со сравнения элементов a_0 и a_1. Последней рассматривается пара a_{n-3} и a_{n-2}. В результате на месте элемента a_{n-2} окажется второй по величине элемент массива (всплыл второй по величине пузырек);

3) третий просмотр массива начинается с проверки пары a_0 и a_1, заканчивается проверкой пары a_{n-4} и a_{n-3}, на месте элемента a_{n-3} окажется третий по величине элемент массива;

4) при последнем просмотре будут сравниваться только элементы a_0 и a_1.

Теперь обобщим представленные шаги алгоритма следующим образом: просмотр массива состоит в проверке условия $a_i \le a_{i+1}$ и перестановке этих элементов при невыполнении условия неубывания, при этом значение переменной *i* изменяется от 0 до некоторого *k*, т. е. последнее проверяемое условие будет $a_k \le a_{k+1}$. Первый просмотр происходит при *k*=*n*–2, следующий при *k*=*n*–3, затем при *k*=*n*–4 и т. д. до *k*=0.

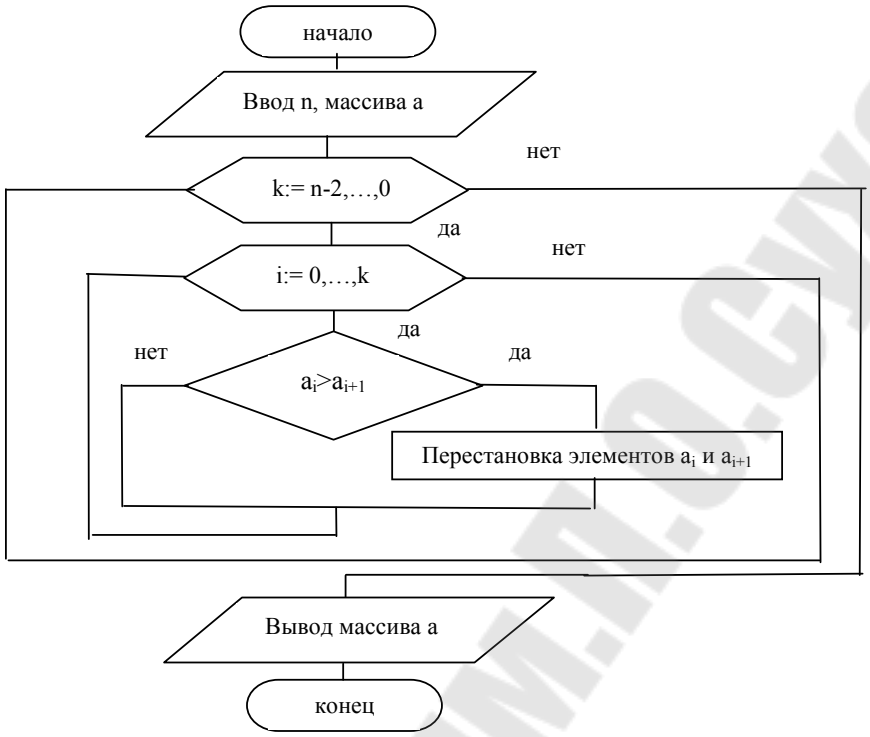

Рис. 3.2. Схема алгоритма сортировки массива по неубыванию методом «пузырька»

Сформулированному алгоритму соответствует **графическая схема алгоритма**, приведенная на рис. 3.2.

Отладку алгоритма следует провести для тех же тестов, что и в предыдущем проекте.

3.5. Минимизация числа просмотров при сортировке методом «пузырька»

При сортировке методом «пузырька» часто встречается ситуация, когда массив уже отсортирован, а просмотры массива продолжаются. Чтобы вовремя прекратить процесс сортировки, будем фиксировать факт перестановки в какой-нибудь переменной.

Для этой цели лучше всего подходит переменная логического типа. Она принимает одно из двух значений: ***True*** или ***False***.

С целью ускорения сортировки пузырьковым методом будем присваивать переменной ***W*** значение ***True*** всякий раз, когда после проверки очередной пары происходила перестановка значений сравниваемых элементов. Очередной оборот цикла для организации нового просмотра (цикл по переменной ***k***) будем выполнять только в том случае, когда при предыдущем просмотре была сделана хотя бы одна перестановка. Перед началом цикла проверки упорядоченности (цикл

по переменной ***i***) надо переменной ***W*** присвоить значение ***False***, признак того, что пока перестановок не было.

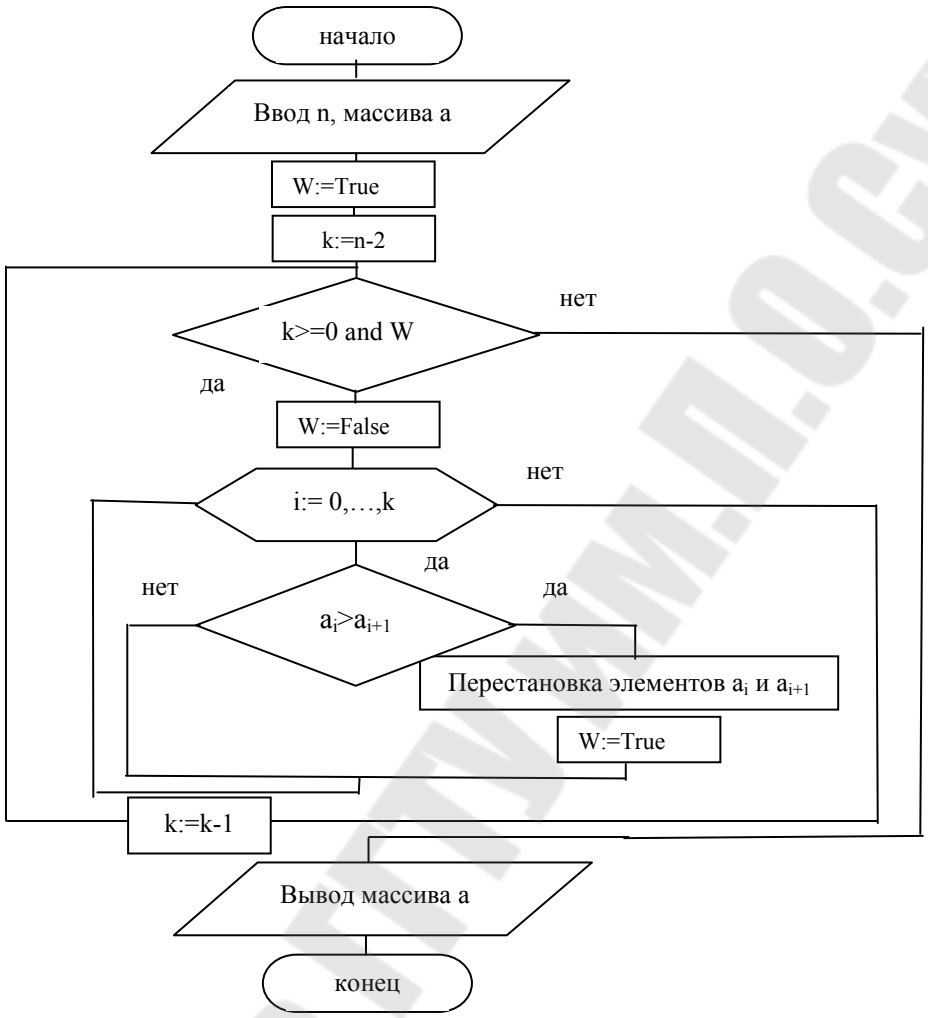

Рис. 3.3. Схема алгоритма сортировки массива по неубыванию методом «пузырька» с минимизацией числа просмотров

Схема алгоритма приведена на рис. 3.3.

Обратите внимание на блок ***W:= True*** перед циклом по переменной ***k***. Такой блок обеспечивает выполнение цикла первый раз.

3.6. Сортировка методом обменов за один просмотр «с возвращением»

Рассмотрим еще один алгоритм сортировки методом обменов. Как и в алгоритме методом «пузырька», мы будем последовательно проверять на упорядоченность по неубыванию пары, начиная с $a_0 \le a_1$ и до $a_{n-2} \le a_{n-1}$. Однако после перестановки элементов, например a_k и a_{k+1}, мы не будем сразу продолжать просмотр слева на-

право, а будем устанавливать правильное местоположение элемента a_{k+1}, проверяя пары элементов справа налево. Таким образом, идея рассматриваемого алгоритма состоит в том, что после нахождения пары, не удовлетворяющей условию неубывания, т. е. пары $a_{k-1} > a_k$, мы в упорядоченной части массива $a_0, \ldots, a_{k-1}$ отыскиваем такое место для элемента a_k, чтобы отсортированной оказалась часть массива $a_0, \ldots, a_k$. После этого можно продолжать просмотр массива слева направо, т. е. можно переходить к проверке условия $a_k > a_{k+1}$.

Схема описанного алгоритма приведена на рис. 3.4.

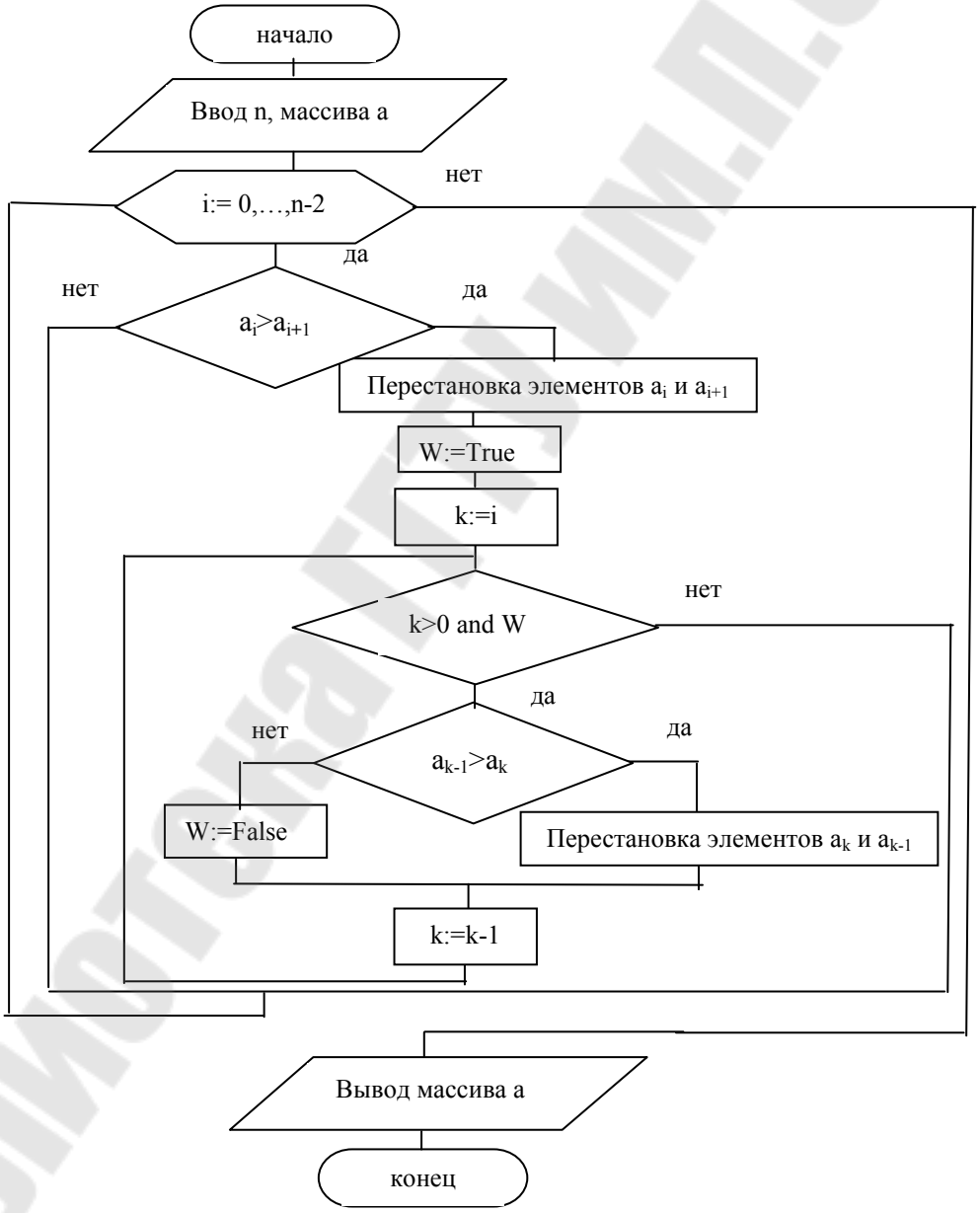

Рис. 3.4. Схема алгоритма сортировки массива по неубыванию методом обменов за один просмотр «с возвращением»

В этом алгоритме внутренний цикл обеспечивает возврат от найденной неупорядоченной пары $a_i > a_{i+1}$ к началу массива. Перед началом этого цикла переменной ***W*** присвоено значение ***True***. Данный цикл заканчивается при выполнении одного из двух условий:

1) ***k=0*** – проверены и переставлены все пары элементов, предшествующие элементу a_i;

2) ***W=False*** – нашлась упорядоченная пара $a_{k-1} \le a_k$.

3.7. Алгоритм сортировки методом слияния

Рассмотрим алгоритм создания упорядоченного по невозрастанию массива методом слияния двух массивов, один из которых упорядочен по невозрастанию, а другой – по неубыванию.

Имеем массив A из n элементов и массив B из m элементов:

$a_0 \le a_1 \le \ldots \le a_{n-1}$ – неубывание;

$b_0 \ge b_1 \ge \ldots \ge b_{m-1}$ – невозрастание.

Требуется получить массив c из n+m элементов:

$c_0 \ge c_1 \ge \ldots \ge c_{n+m-1}$ – невозрастание.

План решения задачи

Пусть i – номер очередного элемента массива C. Тогда i=0,1,2, …, n+m–1.

Пусть k – номер максимального элемента среди оставшихся в массиве A. Сначала k=n–1. По мере переписывания элементов из массива A в массив C k уменьшается на единицу (k=k–1).

Пусть l – номер максимального элемента среди оставшихся в массиве B. Сначала l=0. По мере переписывания элементов из массива B в массив C l увеличивается на единицу (l=l+1).

Если не исчерпаны элементы в массивах A и B, то $c_i = \max\{a_k, b_l\}$. Если же исчерпаны элементы в массиве A, то $c_i = b_{l,}$, затем l=l+1. Если же исчерпаны элементы в массиве B, то $c_i = a_k$, затем k=k–1.

Схема описанного алгоритма приведена на рис. 3.5.

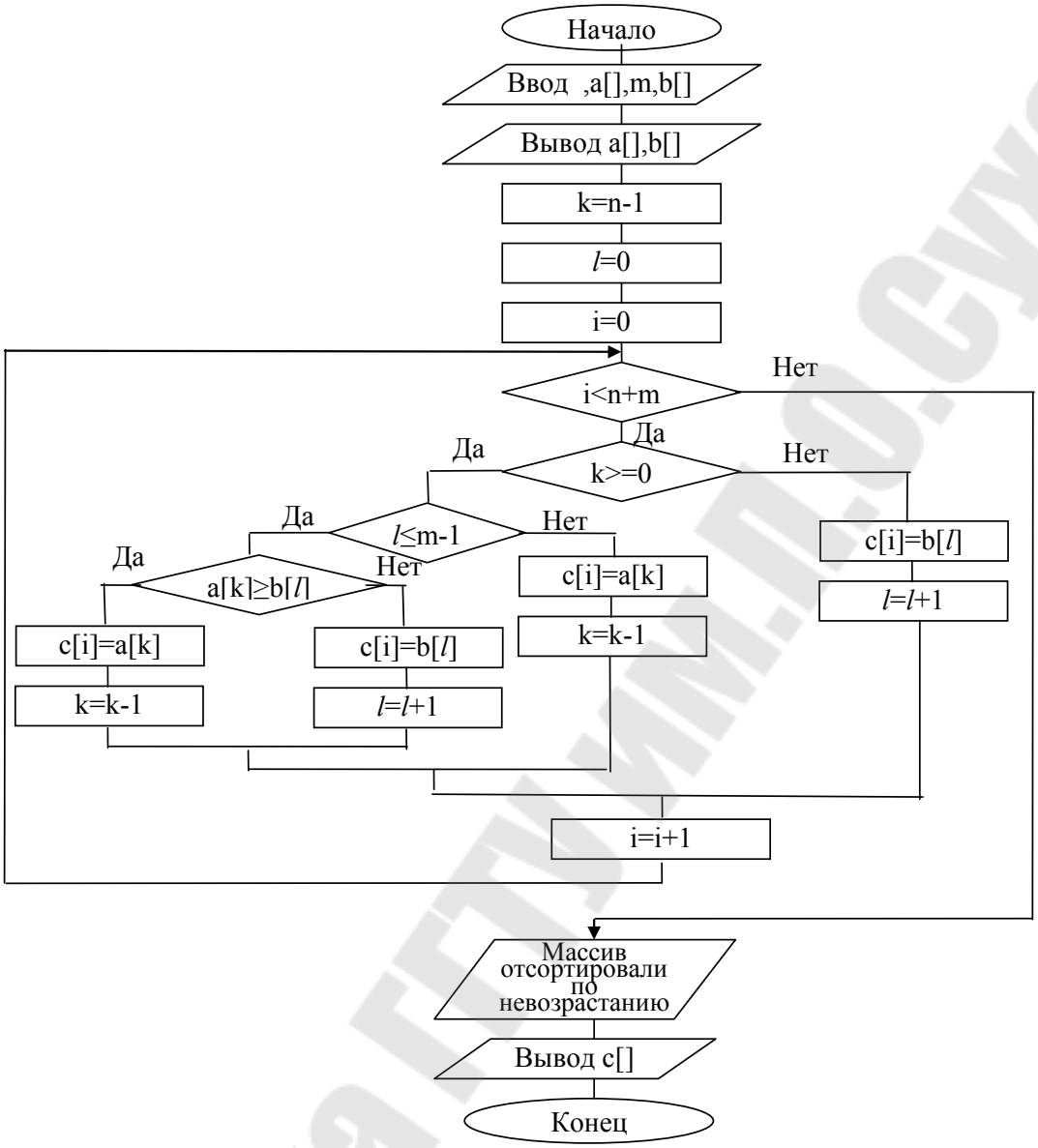

Рис. 3.5. Схема алгоритма сортировки массива слиянием

Текст программы, реализующий алгоритм сортировки по невозрастанию массива **c** слиянием упорядоченных массивов **a** и **b** (массив **a** упорядочен по неубыванию, **b** – по невозрастанию).

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  float a[50],b[50];
```

```
  int n,m,k,l,i;
  puts("Введите количество элементов массива A, упорядоченного по
неубыванию");
  scanf("%d",&n);
  for(i=0;i<n;i++)
        {
         printf("Введите a[%d]\n",i);
         scanf("%f",&a[i]);
        }
  puts("Введите количество элементов массива B, упорядоченного по
невозрастанию");
  scanf("%d",&m);
  for(i=0;i<m;i++)
        {
         printf("Введите b[%d]\n",i);
         scanf("%f",&b[i]);
        }
  clrscr();
  puts("Исходные данные");
  puts("Массив A");
  for(i=0;i<n;i++) printf("%5.1f",a[i]);
  printf("\n");
  puts("Массив B");
  for(i=0;i<m;i++) printf("%5.1f",b[i]);
  printf("\n");
  float *c=new float[m+n];
  k=n–1; l=0;
  for(i=0;i<n+m;i++)
   {
       if(k≥0)
        {if(l≤m–1)
          {if(a[k]≥b[l])  { c[i]=a[k]; k--;}
               else { c[i]=b[l]; l++;}
          }
         else { c[i]=a[k]; k--;}
        }
       else { c[i]=b[l]; l++;}
```

```
 }
puts("Массив отсортировали по невозрастанию");
for(i=0;i<n+m;i++)  printf("%5.1f",c[i]);
delete[]c;
      }
```

Отладку программы следует провести для следующих тестов:

1) все элементы первого массива больше элементов второго массива;

2) все элементы второго массива больше элементов первого массива;

3) все элементы первого массива меньше максимального элемента и больше минимального элемента второго массива;

4) все элементы второго массива меньше максимального элемента и больше минимального элемента первого массива;

5) элементы заданных массивов связаны соотношением:

$b_{m-1}<a_0<b_0<a_{n-1}$, при этом, по крайней мере, одно и то же число содержится в обоих массивах;

6) элементы заданных массивов связаны соотношением:

$a_0<b_{m-1}<a_{n-1}<b_0$, при этом, по крайней мере, одно и то же число содержится в обоих массивах.

3.8. Алгоритм сортировки распределением

Сортировку распределением в отличие от ранее рассмотренных методов нельзя применять для упорядочивания произвольного массива. Данный метод употребим в том случае, когда в исходном числовом массиве имеется небольшое количество различных значений (ключей) и множество этих значений известно заранее.

Рассмотрим сортировку массива a по невозрастанию методом распределения по массиву ключей b, упорядоченному по неубыванию. Схема алгоритма приведена на рис. 3.6. Здесь k – массив-счетчик элементов массива a (k[j] – количество элементов массива a, равных значению b[j]).

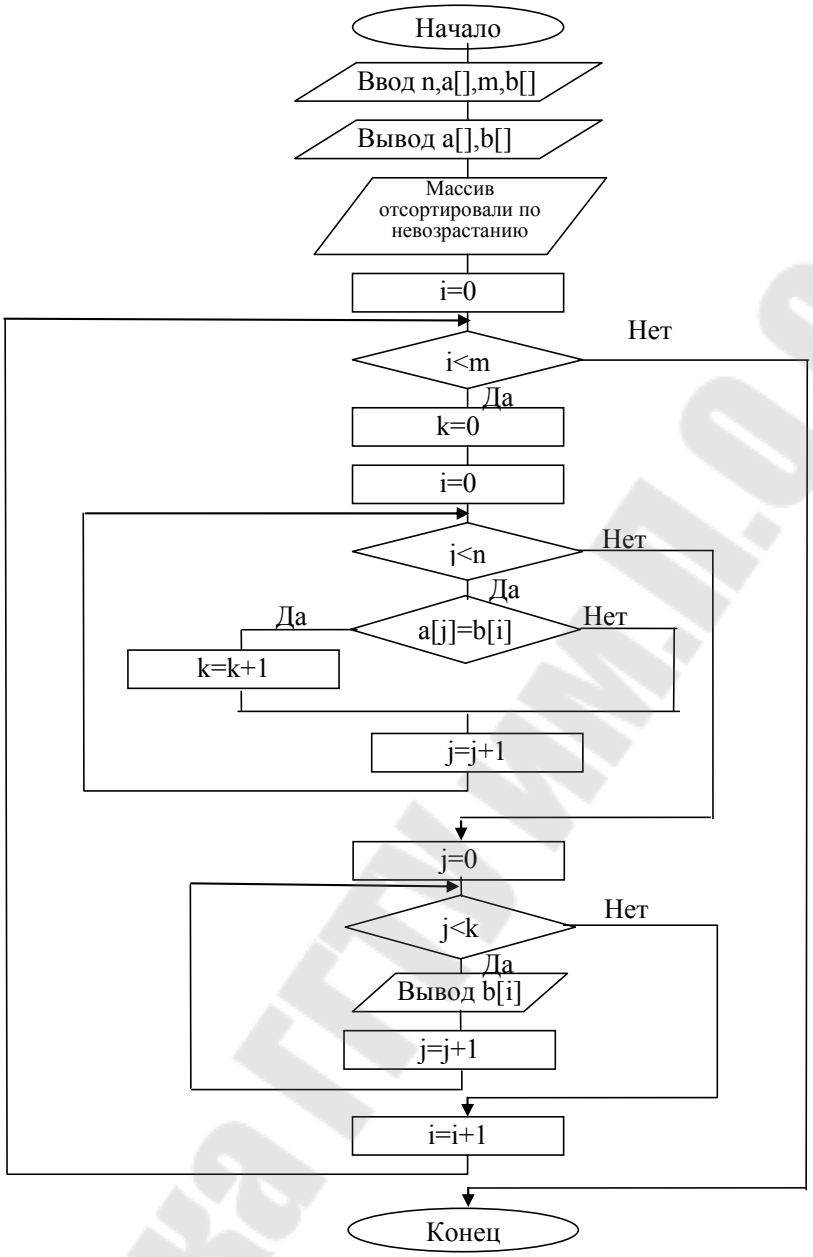

Рис. 3.6. Схема сортировки массива методом распределения

Текст программы

```
#include<stdio.h>
#include<conio.h>
main()
{
  clrscr();
  float a[50],b[20];
  int n,k,j,i,m;
  puts("Введите количество элементов массива");
```

```
 scanf("%d",&n);
 for(i=0;i<n;i++)
        {
         printf("Введите a[%d]\n",i);
         scanf("%f",&a[i]);
        }
 clrscr();
 puts("Исходный массив");
 for(i=0;i<n;i++) printf("%5.1f",a[i]);
 printf("\n");
 puts("Введите количество ключей");
 scanf("%d",&m);
 for(i=0;i<m;i++)
        {
         printf("Введите ключ b[%d]\n",i);
         scanf("%f",&b[i]);
        }
 printf("\n");
 puts("Массив ключей");
 for(i=0;i<m;i++) printf("%5.1f",b[i]);
 printf("\n");

 for(j=0;j<m;j++)
  {k[j]=0;
   for(i=0;i<n;i++)
     if(a[i]==b[j]) k[j]++;  //Кол-во совпавших элементов
// Формирование упорядоченного массива a
  L=0;
  for(j=0; j<m; j++)
    for(i=0; i<k[j]; i++)
          {
            L=L+1;
            a[L]=b[j];
          }
 puts("Отсортированный массив a:");
 for(i=0;i<n;i++) printf("%5.1f",a[i]);
 printf("\n");
```

```
 getch();
 return(0);
}
```

Отладку программы следует провести для тех же тестов, что и в методе обменов.

3.9. Пример перестановки строк матрицы в порядке невозрастания сумм ее строк

В матрице mxn переставить строки таким образом, чтобы получилась последовательность

F1 > F2 >...> Fm, где F – сумма всех элементов *i*-й строки.

Таблица 3.1

Таблица соответствия

Имя в задаче	Имя на С	Тип	Комментарий
x	x	float	исходные данные (массив)
n	n	int	количество столбцов
m	m	int	количество строк
t	t	float	вспомогательная (буферная) переменная
a	a	int	вспомогательная переменная
j	j	int	номер столбца
i	i	int	номер строки
s1	s1	float	сумма элементов i-й строки
s2	s2	float	сумма элементов a-й строки

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int x[20][20],t;
  int n,m,i,a,j,s1,s2;
  clrscr();
  puts("Введите количество строк");
  scanf("%d",&m);
  puts("Введите количество столбцов");
  scanf("%d",&n);
  for(i=0;i<m;i++)      //Ввод матрицы
```

```
        for(j=0;j<n;j++)
          {
           printf("Введите x[%d][%d] : ",i,j);
           scanf("%d",&x[i][j]);
          }
  clrscr();
  puts("Исходный массив:");
  for(i=0;i<m;i++)    //Вывод матрицы
   {
        for(j=0;j<n;j++)
          printf(" %4d ",x[i][j]);
        printf("\n");
        }
  for(i=0;i<m;i++)
   {
        s1=0;
        for(j=0;j<n;j++)     //находим хар-ку (сумму) i строки
          s1=s1+x[i][j];
        a=i+1;
        while(a<m)    //сравниваем сумму i строку со всеми последую-
щими
         {      //и переставляем строку i со строкой с наибольшей суммой
          s2=0;
          for(j=0;j<n;j++)
            s2=s2+x[a][j];
          if(s1<s2)
           {s1=s2;  //хар-ка i строки становится равной s2
                for (j=0;j<n;j++)
                 {
                  t=x[i][j];       //меняем местами элементы
                  x[i][j]=x[a][j];
                  x[a][j]=t;
                 }
           }
          a++;
         }
   }
  printf("\n");
```

```
 puts("Искомый массив:");
 for(i=0;i<m;i++)    //Вывод результата
  {
       for(j=0;j<n;j++)
         printf(" %4d ",x[i][j]);
       printf("\n");
       }
}
```

Тесты

Данные	*Результат*
1. 7 4 6 3 1 10 3 2 12 0 0 1 -4 4 6 1	7 4 6 3 1 10 3 2 12 0 0 1 -4 4 6 1
2. 2 5 -5 0 12 0 -3 5 14 6 2 3 11 3 4 0	14 6 2 3 11 3 4 0 12 0 -3 5 2 5 -5 0
3. 0 0 -4 1 2 -1 5 2 3 2 4 11	2 4 11 5 2 3 1 2 -1 0 0 -4

3.10. Применение алгоритма сортировки методом «пузырька» для расположения записей в массиве записей в лексикографическом порядке

Дан массив записей, содержащий данные о студентах университета. Полями записей являются: ФИО, название факультета, группа, средний балл.

Требуется отсортировать и вывести записи массива в лексикографическом порядке названий факультета.

Для сортировки использовался метод пузырька, графическая схема которого изображена на рис. 3.2.

Поскольку в алгоритме требуется сравнивать величины не числовые, а строковые (поля Факультет), то для сравнения строк ис-

пользована функция strcmp(str1,str2), которая подробно рассматривалась в разделе 1.2.3.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
//Объявления шаблона структурной переменной
struct student
  {
   char fio[25];
   char facultet[10];
   char grup[11];
   double sr;
  }stud[10],p;

main()
{
  int i,j,n,k;
  i=0;
  puts("Введите количество записей");
  scanf("%d",&n);
  //Ввод массива записей с консоли
  for (i=0;i<n; i++)
   {
     puts("Введите ФИО");
     scanf("%s",&stud[i].fio)
     puts("Введите факультет");
     scanf("%s",&stud[i].facultet);
     puts("Введите группу");
     scanf("%s",&stud[i].grup);
     puts("Введите средний балл");
     scanf("%f",&stud[i].sr)
    }

//Сортировка массива записей по  полю факультет в //алфавитном порядке
  for (j=0;j<n-1;j++)
    for(k=n-1;k>j;k--)
```

```
    if (strcmp(stud[k].facultet,stud[k-1].facultet)<0)
       {
     //поменять записи местами
          p=stud[k];
          stud[k]=stud[k-1];
          stud[k-1]=p;
       }
//Вывод на экран данных в виде таблицы
printf("-------------------------------------------------\n");
  for (j=0;j<i;j++)
    printf("|%22s|%10s|%10s|%6.2f|\n",stud[j].fio,          stud[j].facultet,
stud[j].grup, stud[j].sr);
printf("-------------------------------------------------\n");

getch();
return(0);
}
```

3.11. Использование встроенной функции qsort из библиотеки stdlib для сортировки массива записей

Дан массив записей, состоящих из полей: имя, фамилия, год рождения.

Вывести в алфавитном порядке записи по полю фамилия, используя встроенную функцию сортировки qsort.

Прототип функции qsort имеет вид:

void qsort(void *buf, size_t num, size_t size, int (*compare) (const void *, const void *));

Функция qsort() сортирует массив, адресуемый параметром-указателем buf. (Для сортировки используется алгоритм быстрой сортировки (алгоритм quicksort), разработанный Ч.Э.Р. Хоаром (C.A.R. Hoare). Быстрая сортировка считается наилучшим алгоритмом сортировки общего назначения.) Количество элементов в массиве задается параметром num, а размер (в байтах) каждого элемента – параметром size.

Для сравнения двух элементов массива используется функция, передаваемая через параметр compare. Функция compare должна иметь следующее описание:

int func_name(const void *arg1, const void *arg2);

Она должна возвращать значение меншее нуля, если arg1 меньше arg2; нуль, если arg1 равен arg2 и значение больше нуля, если arg1 больше arg2.

Массив сортируется в порядке возрастания, т.е. по самому младшему адресу будет записан наименьший элемент.

Для применения функции сортировки qsort создадим тип данных TStudent для описания массива структур и функцию sort_cmp для сравнения двух структур:

```
            typedef struct //Описание типа TStudent
            {
              char name[20];  //Имя
              char family[20]; //Фамилия
              int year;            //Год рождения
            } TStudent;
  /* Описание функции sort_cmp, которая будет являться параметром
                                  функции qsort*/
int sort_cmp( const void *a, const void *b)
  {
     TStudent *x = (TStudent *)a; //x – указатель на первую из сравни-
                                          //ваемых записей
     TStudent *y = (TStudent *)b; //y - pointer указатель на вторую из
                                           //сравниваемых записей
     return( strcmp( x->family, y->family ) ); //compare by family
   }

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

//Создание нового типа TStudent
typedef struct
{
  char name[20];  //Имя
  char family[20]; //Фамилия
  int year;            //Год рождения
} TStudent;
/*
  /* Описание функции sort_cmp, которая будет являться параметром
                                  функции qsort*/
```

```
int sort_cmp( const void *a, const void *b)
  {
     TStudent *x = (TStudent *)a; //x - pointer to the first student
     TStudent *y = (TStudent *)b; //y - pointer to the second
      return( strcmp( x->family, y->family ) ); //compare by family
  }

int main(void)
{
  int  i;

  //Ввод данных
  const int StudCount = 4;
  TStudent students[StudCount] = {
        {"Alexander", "Farberov", 1987},
        {"Dmitry", "Hrabrov", 1988},
        {"Denis", "Shulga", 1988},
        {"Boris", "Abramovich", 2666}
  };

  //Сортировка
  qsort( (void *)students, StudCount, sizeof(TStudent), sort_cmp );

 //Вывод
  puts("The students are:");
  for (i = 0; i < StudCount; i++)
    printf("%d. %s %s %d\n", i+1,
       students[i].family, students[i].name, students[i].year);
  return 0;
}
```

4. ФАЙЛЫ В ЯЗЫКЕ С

4.1. Общие понятия

Под файлом понимается поименованная область внешней памяти ПК (жесткого диска, гибкой дискеты, электронного «виртуального диска» и др.), хранящая данные. Под файлом понимается также логическое устройство – потенциальный источник или приемник информации.

Любой файл имеет следующие характеристики (атрибуты):

– ***Имя файла***. Составляется по правилам составления идентификаторов в рассматриваемой ОС, например, ***C:\MCDOC\d.txt***).

– ***Тип компонентов***. Например, файл может представлять собой последовательность строк или последовательность байтов.

– ***Длина файла***. Это число компонент файла.

– ***Указатель файла***. Это переменная специального типа, предназначенная для указания на компонент (позицию) файла. Значение указателя файла изменяется после каждого выполнения операции чтения или записи данных (рис. 4.1).

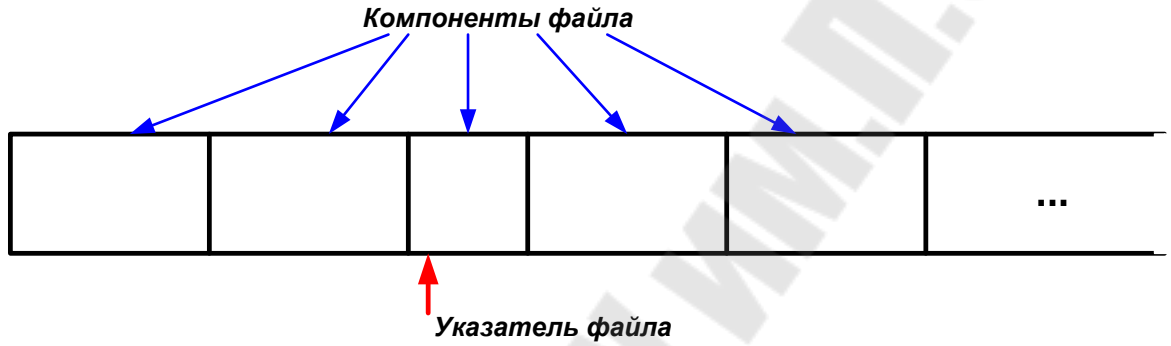

Рис. 4.1

Логические устройства

Такие стандартные аппаратные устройства ПК, как клавиатура, экран дисплея, печатающее устройство (принтер) и коммуникационные каналы ввода-вывода, определяются специальными именами, которые называются ***логическими устройствами***:

– ***con*** – логическое имя, которое определяет консоль (клавиатуру или экран дисплея);

– ***prn*** – логическое имя принтера. Если к ПК подключено несколько принтеров, доступ к ним осуществляется по логическим именам LPT1, LPT2, LPT3. Первоначально prn и LPT1– синонимы;

– ***aux*** – логическое имя коммуникационного канала, который используется для связи ПК с двумя машинами. Коммуникационный канал может осуществлять передачу и прием данных. Как правило, имеется 2 коммуникационных канала: ***com1*** и ***com2***. Первоначально ***aux*** и ***com1*** – синонимы;

– ***NUL*** – логическое имя «пустого» устройства. Чаще всего используется в отладочном режиме как устройство-приемник информа-

ции неограниченной емкости. При обращении к ***NUL*** как к источнику информации, выдается признак конца файла (***EOF***).

Понятие потока

При вводе-выводе данные рассматриваются как поток байтов.

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику, таким образом, физически ***поток*** в С – это файл или устройство.

Чтение данных из потока называется ***извлечением***, вывод в поток – ***помещением*** или ***включением***.

Обмен с потоками для увеличения скорости передачи данных производится, как правило, через специальную область ОП – ***буфер***. При выводе фактическая передача данных происходит после заполнения буфера. При вводе фактическая передача данных происходит, если буфер исчерпан.

По направлению обмена потоки можно разделить на ***входные*** (данные вводятся в память), ***выходные*** (данные выводятся из памяти) и ***двунаправленные*** (происходит как извлечение, так и включение).

По виду устройств, с которыми работает поток, можно разделить потоки на ***стандартные***, ***файловые*** и ***строковые***.

Стандартные потоки предназначены для передачи данных от клавиатуры в память ПК, или из памяти ПК на экран дисплея и принтер.

Файловые потоки предназначены для обмена информацией с файлами на внешних носителях данных.

Строковые потоки – для работы с массивами символов в ОП.

По структуре данных поток может быть ***текстовым*** или ***бинарным*** (двоичным).

Текстовый поток – это последовательность строк, каждая из которых имеет нуль или более ASCII-символов и заканчивается символом "\n" – конца строки и перехода к следующей строке. Текстовые файлы могут быть просмотрены и отредактированы с клавиатуры любым текстовым редактором.

Бинарный (***двоичный***) поток – это последовательность байтов без учета разделения на строки. Каждая программа для своих бинарных файлов определяет собственную структуру.

Стандартные библиотеки для работы с потоками

Для работы с потоками существуют стандартные библиотеки ***stdio***, ***string***, ***stdlib***, ***io***, которые становятся доступными из функций пользователя после использования директивы препроцессора ***#include***, например,

#include <stdio.h>

Вызов подсказки для получения списка функций библиотеки:

1) в среде С выполнить команду HELP→Contents и нажать Enter;
2) выбрать раздел Header Files и нажать Enter;
3) выбрать название библиотеки и нажать Enter;
4) выбрать функцию.

4.2. Работа с файлами (потоками)

4.2.1. Этапы работы с файлами (потоками)

Файл в программе на языке С – это переменная-указатель на тип ***FILE***, называемая ***файловой переменной***.

Работа с файлами состоит их трех этапов:

1. Открытие файла.
2. Обработка файла.
3. Закрытие файла.

Открытие файла (потока)

Поток можно открыть для **чтения**, **записи**, и для ***чтения и записи*** с помощью стандартной функции ***fopen***, прототип которой имеет вид:

FILE *fopen (const char *filename, const char*mode);

Здесь первый параметр функции (***filename***) – имя открываемого файла в виде строки символов, второй (***mode***) – режим открытия файла (тип доступа к файлу) – строка из одного и более символов:

"***r***" – файл открывается для чтения;

"***w***" – открывается пустой файл для записи (если файл существует, то его содержимое теряется);

"***a***" – файл открывается для добавления информации в конец файла; если файла нет, он создается;

"***r+***" – файл открывается для чтения и записи (файл должен существовать);

"***w+***" – открывается пустой файл для записи и чтения (если файл существует, то его содержимое теряется);

"***a+***" – файл открывается для чтения и добавления данных в конец файла;

"***t***" – открытие файла в текстовом режиме;

"***b***" – открытие файла в двоичном режиме.

Примечание. ***По умолчанию*** (когда в режиме открытия файла отсутствует *t* или ***b***) файл открывается в ***текстовом режиме***.

Возможны следующие режимы доступа: ***"w+b", "wb", "rw+", "w+t", "rt+"*** и др.

При успешном открытии потока функция ***fopen*** возвращает указатель на структуру типа ***FILE***. Эта структура связана с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.).

Возвращаемое функцией значение нужно сохранить и использовать для ссылки на открытый файл.

Если произошла ошибка при открытии файла, то возвращается **NULL**.

Таким образом, для работы с функцией ***fopen*** в программе на языке С необходимо:

1. Объявить переменную-указатель на структуру типа ***FILE***, например:

FILE* f;

Эта переменная (***f***) называется файловой переменной или файлом.

2. Файловой переменной присвоить значение, возвращаемое функцией ***fopen***, например:

f=fopen("c:\mdoc\d.txt", "r+");

Примечание. Описание и инициализация файловой переменной (например, ***f***) может быть осуществлено в одном операторе, например:

FILE*f = fopen ("c:\mdoc\d.txt","r+");

После открытия файла все действия над файловой переменной будут отождествляться с действиями над внешним файлом (физиче-

ским файлом), т. е. с файловой переменной связывается определенный физический поток. При этом структуре типа ***FILE*** выделяется область **ОП**, адрес которой возвращает функция ***fopen***. Поток связывается со структурой типа ***FILE***.

Структура типа ***FILE*** используется функциями ввода/вывода для хранения информации, связанной с устройством или файлом. Указатель на тип ***FILE*** используется для операций с файлом с помощью библиотечных функций. Его передают библиотечным функциям в качестве параметра. Такие библиотечные функции называются функциями ввода/вывода.

Примечание. 1. При открытии потока с ним связывается область памяти – буфер. 2. При аварийном завершении программы выходной буфер может быть не выгружен, поэтому возможна потеря данных. 3. С помощью функций **setbuf** и **setvbuf** можно управлять размерами и наличием буферов. 4. Перед началом работы программы операционная система автоматически открывает пять стандартных потоков:

1) **stdin** – стандартный ввод;
2) **stdout** – стандартный вывод;
3) **stdprn** – стандартная печать;
4) **stderr** – стандартный вывод сообщений об ошибках;
5) **stdaux** – стандартный дополнительный поток (стандартные порты).

Потоки 1), 2), 4) относятся к консоли (***con***), 3) – к ***prn***, 5) – к ***aux*** (коммуникационный канал), т. е., в начале работы программы автоматически открываются потоки следующими операторами:

1) ***FILE *stdin=fopen***("***con***", "***r***");
2) ***FILE *stdout=fopen***("***con***", "***w***");
3) ***FILE *stdprn=fopen("prn", "w")***;
4) ***FILE *stderr=fopen***("***con***", "***w***");
5) ***FILE *stdaux=fopen***("***aux***", "***wb***");

Файлы ***stdin*** и ***stdout*** можно переназначать при запуске exe-программы или в программе с помощью функции ***freopen***. Ее прототип:

File ****freopen***(***const char *filename***, ***const char *mode***, ***FILE *stream***);

где ***freopen*** – имя функции;
filename – имя файла;
mode – режим работы файла;
stream – поток.
Функция ***freopen*** работает аналогично функции ***fopen***, но предварительно закрывает поток ***stream***, если тот был ранее открыт.

Закрытие файла (потока)

Для закрытия потоков, используемых в программе, применяются стандартные функции ***fclose*** и ***fcloseall.***

Один заданный поток закрывается функцией ***fclose***. Ее прототип:

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

где ***stream –*** поток.

Пример обращения к функции ***fclose***:

fclose (***f***);

Если поток успешно закрыт, функция ***fclose*** возвращает значение 0. Если при закрытии потока произошла ошибка – значение ***EOF*** (-1).

Функция ***fcloseall*** закрывает все потоки, открытые с помощью ***fopen***, кроме ***stdin***, ***stdout***, ***stderr***. Прототип функции ***fcloseall***:

int fcloseall();

Функция ***fcloseall*** возвращает количество закрытых потоков или значение ***EOF*** (-1), если при закрытии произошла ошибка.

Примечание. Функция ***perror*** позволяет вывести сообщение об ошибке при неуспешном закрытии файла. Ее прототип:

void perror(***const char *s***);

Пример

```
FILE * fp;
fp=fopen("d.dat", "r");
if(!fp) perror("Нельзя открыть файл для чтения");
   else fclose ("d.dat");
```

Удаление файла

Удалить файл можно с помощью функции ***remove***. Ее прототип:

int remove (***const char *filename***);

где ***filename*** – указатель на строку с именем физического файла (имя файла).

Функция ***remove*** возвращает значение 0 при успешном удалении файла и не нуль в противном случае. Открытый файл необходимо предварительно закрыть.

Пример открытия и закрытия файла

В текстовый файл с именем ***rez.txt*** записываются результаты выполнения программы в начало нового файла, если файл не существует, или в конец существующего файла.

```
#include <stdio.h>
void main ()
{
  int n;  //номер теста
  float y,x;
  FILE *f;          //Описание файловой переменной f (файла)
  puts(“Введите x”);
  scanf(“%f”, &x);
  y=2*x;
  f=fopen("rez.txt", "a");  //Открытие файла
  puts("Номер теста ?");
  scanf("%d",&n);
  fprintf(f,"Тест N %d\n",n);  //Запись в файл f значения
  fprintf(f,"%f  %f\n",x,y);
  fclose(f);                          //Закрытие файла
 fflush(stdin);  getchar();
 }
```

В результате первого выполнения программы в текущем каталоге будет создан файл с именем ***rez.txt***. В его начало запишутся значения ***n***, ***x***, ***y***. При последующих запусках программы на выполнение значения ***n, x, y*** будут записываться в конец файла ***rez.txt***, на что указывает режим "***a***" открытия файла в функции fopen. Отсутствие символа "***t***" или "***b***" в режиме открытия файла «говорит» о работе с текстовым файлом (но не расширение ***txt*** в имени файла).

4.3. Ввод-вывод в поток

4.3.1. Основные понятия

Ввод-вывод в поток можно осуществить разными способами:

– в виде последовательности байтов;

– в виде символов и строк;

– с использованием форматных преобразований.

Операции ввода-вывода выполняются, начиная с текущей позиции потока. Текущая позиция определяется положением ***указателя***

потока. Указатель потока устанавливается на начало или конец файла при открытии в соответствии с режимом открытия. После каждой операции ввода-вывода указатель потока автоматически изменяется.

4.3.2. Позиционирование в файле

4.3.2.1. Функции получения текущего положения указателя потока

ftell и ***fgetpos***

Прототип функции ***ftell*** (из <stdio.h>):

***long int ftell(File *f)*;**

Прототип функции ***fgetpos*** (из <stdio.h>):

***long int fgetpos (File *f, fpos_t *pos)*;**

Функция ***fgetpos*** возвращает текущую позицию в файле, связанном с потоком ***f***, и копирует значение текущей позиции по адресу ***pos***. Это значение позднее может использоваться функцией ***fsetpos***. Возвращаемое значение имеет тип ***fpos_t***, который используется функциями ***fgetpos*** и ***fsetpos*** для хранения текущей позиции файла:

typedef long **fpos_t***;*

4.3.2.2. Функции задания положения указателя *fseek* и *fsetpos*, *rewind*

Прототип функции ***fseek***:

***int fseek (File *f, long off, int org)*;**

Функция перемещает текущую позицию в файле, связанном с потоком ***f***, на позицию ***off***, отсчитываемую от значения ***org***, которое должно быть равно одной из констант, определенных в <***stdio.h***>:

SEEK_CUR – от текущей позиции указателя (1);

SEEK_END – от конца файла (2);

SEEK_SET – от начала файла (0).

Параметр ***off*** задает количество байтов, на которое необходимо сместить указатель соответственно параметру ***org***. Величина смещения может быть как положительной, так и отрицательной, но нельзя сместиться за пределы начала файла.

Такой доступ к данным в файле называется произвольным.

Прототип функции ***fsetpos***:

***int fsetpos (File *f, const fpos_t *pos)*;**

Функция перемещает текущую позицию в файле, связанном с потоком ***f***, на позицию *pos, предварительно полученную с помощью функции ***fgetpos***.

Примечание. Функции ***fseek*** и ***fsetpos*** нельзя использовать для стандартных потоков.

Функция ***rewind*** очищает флаги ошибок при работе с потоками и устанавливает указатель на начало файла. Ее прототип:

void rewind(File *f);

4.3.3. Функции чтения и записи потока байтов fread и fwrite

Прототип функции ***fread***:

size_t fread (***void *buffer***, ***size_t size***, ***size_t count***, ***FILE *stream***);

Функция ***fread*** считывает ***count*** элементов длиной ***size*** байтов в область, заданную указателем ***buffer***, из потока ***stream***. Возвращает количество прочитанных элементов, которое может быть меньше ***count***, если при чтении произошла ошибка или встретился конец файла.

Прототип функции ***fwrite***:

size_t fwrite (const ***void *p***, ***size_t size***, ***size_t n***, ***FILE *f***);

Функция записывает ***n*** элементов длиной ***size*** байтов из буфера, заданного указателем ***p***, в поток ***f***. Возвращает число записанных элементов.

4.3.4. Функции чтения символа из потока (getc, fgetc, getchar)

Функция ***getc***, имеющая прототип

int getc (FILE *f);

возвращает очередной символ в формате ***int*** из стандартного потока ***f***. Если символ не может быть прочитан, то возвращает значение ***EOF***.

Функция ***fgetc***, имеющая прототип

int fgetc (FILE *f);

возвращает очередной символ в формате ***int*** из потока ***f***. Если символ не может быть прочитан, то возвращает значение ***EOF***.

Функция ***getchar***, имеющая прототип

int getchar (void);

возвращает очередной символ в формате ***int*** из стандартного потока (*stdin*). Если символ не может быть прочитан, то возвращает значение ***EOF***.

4.3.5. Функции записи символа в поток (putc, fputc, putchar)

Функция ***putc*** записывает символ ***ch*** в поток ***f***. При ошибке возвращает значение ***EOF***. Ее прототип:

int putc (***int ch***, ***FILE *f***);

Выполняется ***fputc*** аналогично функции ***putc***. Ее прототип:

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

Функция ***putchar()*** выводит символ ***ch*** на стандартное устройство вывода, добавляя в конце символ новой строки. Возвращает неотрицательное значение при успехе или EOF – при ошибке.

4.3.6. Функции чтения строки из потока (fgets, gets)

Прототип функции ***fgets***:

char *fgets (***char *s***, ***int n***, ***FILE *f***);

Функция читает не более ***n-1*** байт из потока ***f*** и помещает прочитанные байты в строку ***s***, прекращая чтение при обнаружении символа новой строки или конца файла. Символ новой (**'\n'**) строки при чтении не отбрасывается, помещается в конец строки ***s***. Прочитанная строка дополняется ограничителем строки (**'\0'**). При обнаружении ошибки или конца файла возвращает ***NULL***, в противном случае – указатель на строку ***s***.

Функция ***gets*** читает символы с клавиатуры до появления символа новой строки и помещает их в строку ***s***. Сам символ новой строки в строку не включается. Возвращает указатель на ***char***. Прототип функции ***gets***:

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

4.3.7. Функции записи строки в поток (fputs, puts)

Прототип функции ***fputs***:

int fputs (***const char * s***, ***FILE *f***);

Функция ***fputs*** записывает строку символов ***s*** в поток ***f***. Символ конца строки в файл не записывается. При ошибке возвращает значение ***EOF***, иначе – неотрицательное число.

Функция ***puts*** выводит строку ***s*** на стандартное устройство вывода, добавляя в конце символ новой строки. Возвращает неотрицательное значение при успехе или ***EOF*** – при ошибке. Прототип функции ***puts***:

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

4.3.8. Функции форматированного ввода (чтения) из потока (fscanf, scanf, sscanf)

Прототип функции ***fscanf***:

int fscanf (***FILE *f***, ***const char *fmt*** [, ***par1***, ***par2***, …]);

Эта функция вводит (читает) строку параметров ***par1***, ***par2***, … в формате, определенном строкой ***fmt***, из потока (файла) ***f***. Возвращает число переменных, которым присвоено значение.

Прототип функции ***scanf***:

int scanf (***const char *fmt*** [, ***par1***, ***par2***, …]);

Функция ***scanf*** вводит (читает) строку параметров ***par1***, ***par2***, … в формате, определенном строкой ***fmt***, со стандартного устройства ввода (обычно с клавиатуры (***stdin***)). Возвращает число переменных, которым присвоено значение.

Прототип функции ***sscanf***:

int sscanf (***const char *buf***, ***const char *fmt*** [, ***par1***, ***par2***, …]);

Функция ***sscanf*** вводит данные аналогично функции ***scanf***, но не с клавиатуры, а из строки символов, переданной ей первым параметром. Аргумент ***buf*** – строка символов, из которой вводятся значения, ***fmt*** – строка формата, в соответствии с которой происходит преобразование данных. Многоточие указывает на наличие необязательных аргументов, соответствующих адресам вводимых значений.

4.3.9. Функции форматированного вывода в поток (fprintf, printf, sprintf)

Прототип функции ***fprintf***:

int fprintf (***FILE *f***, ***const char *fmt***, ***…***);

Функция записывает в поток ***f*** значения переменных, список которых обозначен многоточием (**…**), в формате, указанном строкой ***fmt***. Возвращает число записанных значений.

Прототип функции ***printf***:

int printf (***const char *fmt***, **…**);

Функция выводит на стандартное устройство вывода (***stdout***) значения переменных, перечисленных в списке, обозначенном многоточием (**…**), в формате, определенном строкой ***fmt***. Возвращает число выведенных значений.

Прототип функции ***sprintf***:

int sprintf (***char *buf***, ***const char *fmt***, **…**);

Функция выводит в строку ***buf*** значения переменных, перечисленных в списке, обозначенном многоточием (**…**), в формате, определенном строкой ***fmt***.

Примечание. Спецификации формата ***fmt*** были рассмотрены в разделе «Консольный ввод-вывод».

4.4. Обработка ошибок

Функции работы с потоками возвращают значения, которые рекомендуется анализировать в программе и обрабатывать ошибочные ситуации, возникающие, например, при открытии файлов или чтении из потока. При работе с файлами часто используют функции ***feof*** и ***ferror***.

Прототип функции ***feof***:

int feof (***FILE *f***);

Функция возвращает ***EOF*** или значение, отличное от 0, если достигается конец файла; в противном случае 0.

Прототип функции ***ferror***:

int ferror (***FILE *f***);

Возвращает не равное нулю целое значение, означающее код ошибки, если обнаружена ошибка ввода/вывода; 0 – в противном случае.

4.5. Пример обработки текстового файла

В текстовом файле "dbase.txt" хранятся данные о сотрудниках фирмы. В каждой строке файла указана фамилия, год рождения, ок-

лад сотрудника. Для простоты обработки файла данные записаны единообразно: первые 15 символов занимает фамилия, следующие 5 – год рождения, последние 10 – оклад.

Требуется, интерпретируя структурами строки файла, вывести на экран или принтер содержимое файла в виде таблицы, создать из строк файла с данными о сотрудниках фирмы, моложе 20 лет, массив структур. Вывести на экран или в текстовый файл полученный массив или вывести сообщение о том, что такие молодые сотрудники не работают в фирме.

Структура записи файла изображена на рис. 4.2.

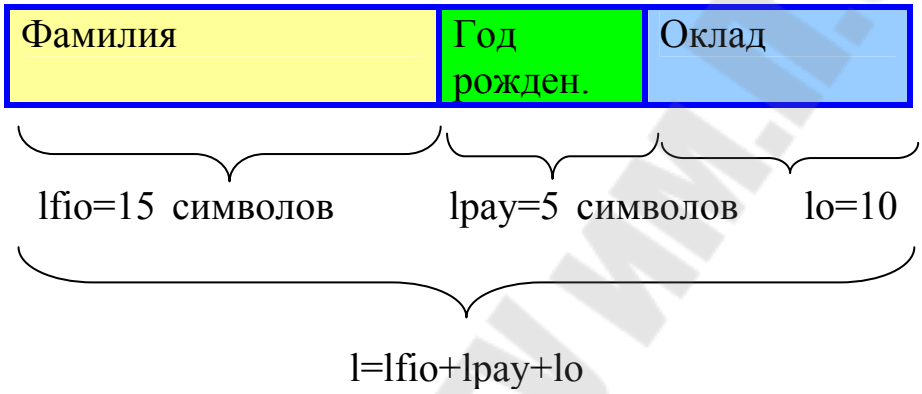

Рис. 4.2

Записи файла могут иметь вид:

Иванов И.П. **1982 453120**
Авчинникова **1999 578320**
Васильков **1955 456780**
Чернов **1967 1345600**

Требуется на экран, на принтер или в текстовый файл вывести список в форме таблицы 4.1. Например:

Таблица 4.1

Список сотрудников фирмы

Фамилия	Год рождения	Оклад
Иванов И.П.	1982	453120
Авчинникова	1999	578320
Васильков	1955	456780
Чернов	1967	1345600

```
/*   Поиск в массиве структур, читаемых из текстового файла */
#include <stdio.h>    //с записью рез-тов в новый текстовый файл
#include <string.h>
#include <stdlib.h>
//#include <windows.h>
main()
{
 const int lfio=15,            //длина поля фио,
     lpay=5,                   // длина поля г.рожд
     lo=10,                    //длина поля оклада
     l=lfio+lpay+lo;           //длина записи в файле
 struct Man
     {  char   fio[lfio];      //фио
 int    year;             //год рожд.
 float  pay;              //оклад
     };
 Man db;
 char  s[l];                   //строка для записи файла
 FILE *fin,                    //Исх. файл
     *fo,*f1 ;                 //Вых. Файлы
 fin =fopen("dbase.txt", "r");
 if  (fin==NULL)
puts("Ошибка открытия файла\n");
else
{
puts("Файл открыт");
fo=fopen("dbout.txt","w");
f1=fopen("dbout1.txt","w");
while (!feof(fin))
  { fgets(s,l,fin);   puts(s);
    strncpy(db.fio,s,lfio-1);
    db.fio[lfio-1]='\0';
    db.year=atoi(&s[15]);
    db.pay=atof(&s[20]);
    if(db.year>2000)
      fprintf(fo,"%-15s %10.1f\n", db.fio, db.pay);
    fputs(s,f1);
  }
 fclose(fin); fclose(fo); fclose(f1);//
```

```
}
 fflush(stdin); getchar();
 return 0;
}
```

4.6. Пример обработки текстового и бинарного файла

/* Построчное считывание данных из текстового файла "***dbase.txt***" в буферную переменную ***s***, формирование из них структуры ***db*** и запись ее в двоичном режиме в выходной файл "***dbout.dat***".

Считывание из двоичного файла записи с номером i и вывод ее на экран.

Считывание из двоичного файла записей и вывод на экран только тех записей, для которых фамилия есть "***ivanoff***". */

В таблице 4.2 приведены значения переменных.

Таблица 4.2

Таблица соответствия

№	Идентификатор	Тип	Комментарий
1	lfio	const int	Длина поля фио
2	lyear	const int	Длина поля г. рожд.
3	lo	const int	Длина поля оклада
4	l	const int	Длина записи
5	db	Запись	Запись
6	s	строка	Строка с содерж. записи
7	fin	Текстовый файл	Исходный текстовый файл
8	fo	Двоичный файл	Двоичный файл, получ. из fin
9	kol	int	Кол-во записей файла fin
10	i	int	Номер записи файла fin

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
main()
{
 const int lfio=15, lpay=5, lo=10,//длины полей: фио, г.рожд., оклада в т. файле
 l=lfio+lpay+lo;   //длина записи в т. файле
 struct Man
    {  char  fio[lfio];   //фио
       int    year;       //год рожд.
```

```
        float  pay;         //оклад
    };
  Man db;
  char  s[l];         //строка для записи в файл
  FILE *fin,          //Исх. файл
       *fo ;       //Вых. файлы
  if ((fin=fopen("dbase.txt", "r"))==NULL)
        {
          puts("Ошибка открытия файла\n");
        fflush(stdin); getchar();  return (1);
        }
  puts("Файл открыт");
  fo=fopen("dbout.dat","w+b");
  int kol=0;  //кол. зап. в текст. файле
  while (!feof(fin))    //пока не конец файла
   {
    fgets(s,l,fin);      //читается строка
    puts(s);
    strncpy(db.fio,s,lfio-1);    //из строки формир. структура db
    db.fio[lfio-1]='\0';
    db.year=atoi(&s[lfio]);
    db.pay=atof(&s[lfio+lpay]);
    fwrite(&db, sizeof (db) ,1, fo);     //структура зап. в бинарный файл
    kol++;
   }
  fclose(fin);
  //чтение зап. с ном. i из бин. файла и вывод ее на экран
  int i;
  printf("Введите номер записи (0-%d)", kol-1);
  cin>>i;
  if(i>=kol || i<0) { cout<<"Запись не существует"; fclose(fo); return (1);}
  fseek(fo, sizeof (db)*i, SEEK_SET);   //установ. указ. на зап. с ном. i
  fread(&db, sizeof(db), 1, fo);        //чтение зап. из файла
  cout << db.fio << " "<< db.year<<" " << db.pay; //вывод зап. на экран

//Вывод записей с фам. ivanoff
  fseek(fo,0,SEEK_SET);  //указ. на нач. файла
  for(i=0; i<kol;i++)
    {  printf("\n");
       fread(&db,sizeof (db),1,fo);
       if(!strcmp (    db.fio," ivanoff      "))
```

```
            printf("% -15s % 5d %/.0f \n",  db.fio, db.year, db.pay);
    }
  fclose(fo);
  fflush(stdin); getchar();
  return (0);
}
```

5. ПОРАЗРЯДНЫЕ ЛОГИЧЕСКИЕ ОПЕРАЦИИ

5.1. Размещение данных в памяти ПЭВМ

Данные и программы во время работы ПЭВМ размещаются в оперативной памяти, которая представляет собой последовательность пронумерованных ячеек. По указанному номеру процессор находит нужную ячейку, поэтому ***номер ячейки называется ее адресом***. Минимальная адресованная ячейка состоит из 8 двоичных позиций, т. е. в каждую позицию может быть записан 0 или 1.

Для удобства работы введены следующие термины, обозначающие совокупности двоичных разрядов. Эти термины обычно используются в качестве единиц измерения объемов информации, хранимой или обрабатываемой в ЭВМ.

Объем информации, который помещается в одну двоичную позицию, называется ***битом***.

Объем информации, равный 8 битам, называется ***байтом***.

В одной ячейке из 8 двоичных разрядов помещается объем информации в 1 байт, поэтому объем памяти принято оценивать количеством байт:

2^{10} байт = 1024 байт = 1 Кб;

2^{10} Кб = 1048576 байт = 1 Мб;

2^{10} Мб = 1 Гб;

2^{10} Гб = 1 Тб.

При размещении данных производится их запись с помощью нулей и единиц – ***кодирование***, при котором каждый символ заменяется последовательностью из 8 двоичных разрядов в соответствии со стандартной кодовой таблицей (ASCII). Например, символ *D* (код – 68) имеет двоичный код 01000100; символ *F* (код – 70) имеет двоичный код 00100110; символ 4 (код – 52) имеет двоичный код 00110100.

При кодировании числа (коды) преобразуются в двоичное представление, например,

$$2 = 1*2^1 + 0*2^0 = 10_2;$$
$$5 = 1*2^2 + 0*2^1 + 1*2^0 = 101_2;$$
$$256 = 1*2^8 = 100000000_2.$$

С увеличением числа количество разрядов для его представления в двоичной системе резко возрастает, поэтому для размещения большого числа выделяется несколько подряд расположенных байт. В этом случае ***адресом ячейки является адрес первого байта***, один бит которого выделяется под знак числа.

Последовательность нескольких битов или байтов часто называют ***полем*** данных. ***Биты в числе (в слове, в поле и т. п.) нумеруются справа налево, начиная с 0-го разряда***.

В ПК могут обрабатываться поля постоянной и переменной длины.

Поля постоянной длины:

слово – 2 байта;
полуслово – 1 байт;
слово длиной 10 байт – 10 байт;
двойное слово – 4 байта;
расширенное слово – 8 байт.

Числа с фиксированной запятой чаще всего имеют формат слова и полуслова, числа с плавающей запятой – формат двойного и расширенного слова.

Поля переменной длины могут иметь любой размер от 0 до 256 байт, но обязательно, равный целому числу байтов.

5.2. Побитовые операции

Побитовые или поразрядные логические операции предназначены для обработки значений на машинном (битовом или двоичном) уровне их представления и позволяют манипулировать индивидуальным битом в интегрированном примитивном типе данных (данным целого типа). Побитовые операции выполняются по законам Булевой алгебры с битами как с самостоятельными величинами. Обычно побитовые операции используются в программах, реализующих доступ к аппаратуре, для логической обработки битовых данных, требующих операций над битами: их выделения из значения, анализа, замены, сдвига и т. п. В таблице 5.1 приведены побитовые операции в порядке убывания их приоритета. Чтобы правильно использовать эти операции, надо знать машинное (битовое) представление обрабатываемых значений.

Таблица 5.1

Побитовые операции языка С

Операции	Результат	Оператор	Результат
~	побитовое унарное отрицание (обращение) (NOT)	–	–
>>	сдвиг вправо	>> =	сдвиг вправо с присваиванием
<<	сдвиг влево	<<=	сдвиг влево с присваиванием
&	побитовое И (AND) – поразрядное логическое умножение	&=	побитовое И (AND) с присваиванием
^	побитовое исключающее ИЛИ (XOR) – сложение разрядов по модулю 2	^=	побитовое исключающее ИЛИ (XOR) с присваиванием
|	побитовое ИЛИ (OR)	|=	побитовое ИЛИ (OR) с присваиванием

Побитовые операции выполняются только над скалярными типами данных, т. е над данными, принимающими только целочисленные значения: char, short int, int, long int, а также их signed и unsigned модификации.

Побитовые операции можно комбинировать со знаком = для соединения побитовой операции и операции присваивания: **&=**, |= и ^= являются допустимыми. Так как ~ – это унарная операция, то она не может комбинироваться со знаком =.

Результаты побитовых операций **&**, ^, |, ~ идентичны результатам соответствующих им логических операций. Результаты побитовых операций **&**, ^, |, ~ приведены в таблице истинности 5.2.

Таблица 5.2

Таблица истинности побитовых операций &, ^, |, ~

A	B	A&B	A^B	A|B	~ A
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	0	1	0

Побитовые операции &, ^, | сравнивают каждый бит своего первого операнда с соответствующим битом второго операнда и в зависимости от их комбинации устанавливают соответствующий бит результата в 0 или 1.

Операция ~ изменяет каждый бит своего операнда на противоположное значение: 0 – на 1, 1 – на 0.

Пример 1. ~

```
int a=2;    //0000…0010 – двоичное представление числа 2
int r=~a;   //1111…1101  – результат ~
```

Пример 2. &

```
int a=10;      //0000…01010
int b=12;      //0000…01100
int r=a&b;     //0000…01000 – число 8
```

Пример 3. ^

```
int a=10;     //0000…01010
int b= 12;    //0000…01100
int r=a^b;    //0000…00110 – число 6
```

Пример 4. |

```
int a=10;     //0000…01010
int b= 12;    //0000…01100
int r=a|b;      //0000…01110 – число 14
```

Операция **<<** объединяет два операнда (например, a<<b) и выполняет сдвиг влево всех битов своего левого операнда (операнд a) на число позиций, заданное правым операндом (операнд b). При этом часть битов в левых разрядах операнда **a** выходит за границы и теряется, а соответствующие правые позиции заполняются нулями.

Операция **>>** также объединяет два операнда (a>>b) и означает сдвиг вправо. Она перемещает все биты своего левого операнда вправо на число позиций, заданное правым операндом. Когда биты левого операнда выдвигаются за самую правую позицию слова (16 бит), то они теряются. При сдвиге вправо освобождающиеся старшие (левые) разряды сдвигаемого числа заполняются предыдущим содержимым знакового разряда. Такое поведение называют ***расширением знакового разряда***.

Результат операции сдвига не определен, если второй операнд отрицательный. Результат операции теряется, если результат сдвига не может быть представлен типом первого операнда после преобразования.

Пример сдвига влево

$0000000010001010_2 << 2 = 0000001000101000_2$ (<< 2 эквивалентно умножению на 4).

Покажем, что сдвиг влево на два бита эквивалентен умножению на 4:

$0000000010001010_2=2^7+2^3+2^1 = 128+8+2 = 138_{10}$;
$0000001000101000_2=2^9+2^5+2^3 =512+32+8=552_{10}$.
138*4=552.

Пример сдвига вправо

0000000010001010_2 >> 2 = 0000000000100010_2 (>> 2 эквивалентно целочисленному делению на 4).

Покажем, что сдвиг вправо на два бита эквивалентен делению на 4:
$0000000010001010_2=2^7+2^3+2^1 = 128+8+2 = 138_{10}$;
$0000000000100010_2 =2^5+2^1=32+2=34_{10}$;
[138/4]=34.

Операции сдвига могут применяться для деления или умножения целого значения на число, равное степени 2:

x << n – эквивалентно умножению числа x на 2^n.

x >> n – эквивалентно целочисленному делению числа x на 2^n.

Например, чтобы разделить целое число x на 8 ($8=2^3$), можно записать оператор:

x=x>>3;

5.3. Примеры работы с битами

В приведенных ниже примерах x – целое число (2 байта): bitno – номер бита; maska – число, которое может быть записано в виде двоичного числа.

1. **Установка заданного бита или группы битов** (значение бита или группы битов сделать равным 1):

x|= (1 << bitno) или x |= maska //1

Строку //1 можно записать:

x=x|(1 << bitno) или x= x|maska

Например, надо в числе x установить 5-й бит. Для этого запишем оператор **x=x|(1<<5);**

Покажем на числовом примере, что это верно;

$x=15_{10} =0000000000001111_2$
$1_{10} =0000000000000001_2$
$1<<5 =0000000000100000_2$
$x|(1<<5) =0000000000101111_2$

Например, в числе требуется установить каждый четный бит, тогда используем «маску» maska =0b0101010101010101;

Если x=0b0110010011110001; (серым цветом выделены биты, которые требуется установить), то после выполнения оператора x=x|maska; получим значение x, равное

0111010111110101

В самом деле:

0110010011110001
0101010101010101

После | получаем

0111010111110101

Здесь цветом выделены установленные биты. Остальные биты остались без изменения.

Если надо сформировать значение результата как совокупности всех двоичных единиц операндов, можно использовать операцию |. Например,

int x=10; //0000…1010_2
int y= 12; //0000…1100_2
int z=x|y; //0000…1110_2 – число 14_{10}

Если в качестве результата надо сохранить только те разряды, в которых только одна из слагаемых есть единица, надо применить операцию ^. Например,

int x=3; //0000…000011_2
int y= 26; //0000…011010_2
int z=x^y; //0000…011001_2 – число 25

2. **Сброс (очистка) заданного бита или группы битов** (значение бита или группы битов сделать равным 0):

x &= ~(1 << bitno) или x &= ~maska //2

Строку //2 можно записать:

x =x &(~(1 << bitno)) или x =x&(~maska)

Например, надо в числе x сбросить 5-й бит. Для этого запишем оператор **x=x&(~(1<<5));**

Покажем на числовом примере, что это верно;

x = 0000000011110011_2
1 = 0000000000000001_2
1<<5 = 0000000000100000_2
~(1<<5) = 1111111111011111_2
x&(~(1<<5)) = 0000000011010011_2

Например, в числе требуется сбросить биты 2-й, 10-й и 13-й, тогда используем «маску»

maska =0b0010010000000100;

Если x= 0b0110010011110001; (серым цветом выделены биты, которые требуется сбросить), то после выполнения оператора

x =x&(~maska) ;

получим последовательно:

x = 0110010011110001_2

maska = 0010010000000100_2

~maska = 1101101111111011_2

x&(~maska) = 0100000011110001_2

В программе, таким образом, надо записать два оператора:

maska =0b0011010000000100; // Задание маски в виде двоичного числа

x =x&(~maska) ;

3. **Инвертирование заданного бита**:

x ^= (1 << bitno) //3

Строку //3 можно записать:

x=x ^ (1 << bitno)

Например, требуется инвертировать 5-й бит числа x. В программе запишем оператор

x=x ^ (1 <<5);

Покажем на числовом примере, что это так для $x=0101010000001001_2$.

1 = 0000000000000001_2

1<<5 = 0000000000100000_2

x = 0101010000001001_2.

x ^ (1 <<5) = 0101010000101001_2.

4. **Для выделения заданных битов используют «маски», которые обнуляют ненужные биты**:

Дано число A. Выделить из числа 1, 3 10 и 12 биты:

A&0b0001010000001010.

Здесь маска записана в двоичной системе счисления, число A имеет тип int.

Таким образом, чтобы «замаскировать» или выделить для обработки заданные разряды (биты) двоичного числа, надо использовать в константе выделения («маске») значения: для выделения разрядов – единицы, а для «маскировки» разрядов – нули.

Задачи

Даны два целых положительных числа A и B:

1. Определить значение каждого бита числа A.

2. Обнулить все четные биты числа A.

3. Выполнить проверку заданного (вводится с клавиатуры) бита числа B, и если заданный бит имеет значение «ЛОЖЬ», т. е. равен 0, установить его.

4. Найти $A \& B$, $A \mid B$, $A \wedge B$, $\overline{A}$.

5. Рассчитать $A >> 1$, $A >> 2$, $A >> 3$, $A >> 8$, $B << 1$, $B << 2$, $B << 3$, $B << 8$ и показать, что результаты выполнения указанных операций совпадают с делением или умножением на степень двойки, соответствующий количеству сдвигаемых разрядов вправо или влево.

Переменные A и B должны иметь тип unsigned int.

Исходные данные и результат вывести в десятичной, шестнадцатеричной системах счисления и в виде двоичного числа.

Решение

```
/* Определение каждого бита целого числа A */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,  //Заданное целое число
    A1,  //Рабочее целое число
    B,   //Маска
    C,   //Значение бита числа
    i;   //параметр цикла

 puts(" Введите целое число");
 scanf("%d",&A);
 printf("  A=%d\n",A);
 B=1;  //в двоичной сист. 0000000000000001
 A1=A;
 puts(" Двоичное представление числа A:");
 for(i=1;i<=16;i++)
    {
     C=A1&B;//Определение i-1-го бита
     printf("  бит %2d равен  %d\n",i-1,C);
```

```
    A1=A1>>1;
  }
 printf("\n");
 getch();
 return(0);
}
```

Запустим программу на выполнение (BIT1.EXE)

Решение

```
/* Обнуление четных битов целого числа A */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,  //Заданное целое число
    A1,  //Рабочее челое число
    B,   //Маска
    C,C1,   //Значение бита числа
    i;   //параметр цикла
 int y1[16],y2[16];
 puts(" Введите целое число");
 scanf("%d",&A);
 printf("  A=%d\n",A);
 B=1;  //в двоичной сист. 0000000000000001
 A1=A;
 /* Двоичное представление исходного  A: */
 for(i=0;i<=15;i++)
   {
    C=A1&B;
    y1[i]=C;
    A1=A1>>1;
   }
 printf("\n");
 /* Обнуление четных битов числа A  */
 for(i=0;i<=15;i=i+2)
   {
    B=~(1<<i);
    A=A&B;
   }
```

```
/* Двоичное представление измененного числа  A:*/
B=1;A1=A;
for(i=0;i<=15;i++)
  {
    C=A1&B;
    y2[i]=C;
    A1=A1>>1;
  }
printf(" Исходное и измененное число A\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y1[i]);
printf("\n ");

for(i=15;i>=0;i--)
  printf(" %d ",y2[i]);
printf("\n");
getch();
return(0);
}
```

Решение

```
/* Найти A&B, A|B, A^B, ~A для целых A и B */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 int A,B,  //Заданные числа
            A1,  //Рабочее целое число
   C,C1,C2,C3,C4,   //Значение бита числа  и результаты
   i,    //параметр цикла
   k;   //номер бита
 int y1[16],y2[16],y[16];
 puts(" Введите два целых числа");
 scanf("%d%d",&A,&B);
 printf("  A=%d B=%d\n",A,B);
 getch();
 //1 -  в двоичной сист. 0000000000000001
 A1=A;
 /* Двоичное представление исходного  A: */
```

```
for(i=0;i<=15;i++)
  {
   C=A1&1;
   y1[i]=C;
   A1=A1>>1;
  }
printf("\n");
A1=B;
/* Двоичное представление исходного  B: */
for(i=0;i<=15;i++)
  {
   C=A1&1;
   y2[i]=C;
   A1=A1>>1;
  }
printf("\n");
C1=A&B;
/* Двоичное представление A&B:*/
A1=C1;
for(i=0;i<=15;i++)
  {
   C=A1&1;
   y[i]=C;
   A1=A1>>1;
  }
printf(" A:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y1[i]);
printf("\n ");
printf(" B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y2[i]);
printf("\n");
printf(" A&B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
 /* Двоичное представление A|B:*/
C2=A|B;
```

```
A1=C2;
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y[i]=C;
    A1=A1>>1;
  }
printf(" A|B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
/* Двоичное представление A^B:*/
C3=A^B;
A1=C3;
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y[i]=C;
    A1=A1>>1;
  }
printf(" A^B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
/* Двоичное представление ~A:*/
C4=~A;
A1=C4;
for(i=0;i<=15;i++)
  {
    C=A1&1;
    y[i]=C;
    A1=A1>>1;
  }
printf(" ~A:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");
getch();
return(0);
}
```

Решение

```
/* Если в A установлены 1 2 5 8 биты в B установлены 3 4 9 биты в C 1 бит, то вычислить 2A&B/4+4C
   Если в A установлены 6 9  биты в B установлены 13 15 биты в C 2 бит, то вычислить (A|B/2)C */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
{
 unsigned int A,B,  //Заданные числа
              AR,BR,R,  //Рабочее челое число
              s,   //цифра числа
    A1,A2,A5,A8,B3,B4,B9,A6,A9,B13,B15;    //Значение бита числа  и результаты
    char C,CR,C1,C2;
   int i,    //параметр цикла
    k;   //номер бита
 int y1[16],y2[16],y[16];
 puts(" Введите три целых числа");
 scanf("%d%d%d",&A,&B,&C);
 printf("  A=%d B=%d C=%d\n",A,B,C);
 getch();
 //1 -  в двоичной сист. 0000000000000001
 AR=A;
 /* Двоичное представление исходного  A: */
 for(i=0;i<=15;i++)
   {
     s=AR&1;
     y1[i]=s;
     AR=AR>>1;
   }
 printf("\n");
 BR=B;
 /* Двоичное представление исходного  B: */
 for(i=0;i<=15;i++)
   {
     s=BR&1;
     y2[i]=s;
```

```
    BR=BR>>1;
  }
printf("\n");
/* Двоичное представление C:*/
CR=C;
for(i=0;i<=15;i++)
  {
    s=CR&1;
    y[i]=s;
    CR=CR>>1;
  }
printf(" A:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y1[i]);
printf("\n ");
printf(" B:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y2[i]);
printf("\n");
printf(" C:\n ");
for(i=15;i>=0;i--)
  printf(" %d ",y[i]);
printf("\n");

/* Определение значений битов 1 2 5 8 числа A
 битов 3 4 9 числа B и бита 1 числа C */
 AR=A;  A1=(AR>>1)&1;
 AR=A;  A2=(AR>>2)&1;
 AR=A;  A5=(AR>>5)&1;
 AR=A;  A8=(AR>>8)&1;
 BR=B;  B3=(BR>>3)&1;
 BR=B;  B4=(BR>>4)&1;
 BR=B;  B9=(BR>>9)&1;
 CR=C;  C1=(CR>>1)&1;
 if(A1&&A2&&A5&&A8&&B3&&B4&&B9&&C1)
   {
    puts("ДА1");
    AR=A; BR=B; CR=C;
    R=(AR<<1)&(BR>>2)+CR<<2;
```

```
    printf("R=%d\n",R);
   }
    /* Определение значений битов 6 9 числа A
  битов 13 15 числа B и бита 2 числа C */
  else
   {
    AR=A;  A6=(AR>>6)&1;
    AR=A;  A9=(AR>>9)&1;
    BR=B;  B13=(BR>>13)&1;
    BR=B;  B15=(BR>>15)&1;
    CR=C;  C2=(CR>>2)&1;
    if(A6&&A9&&B13&&B15&&C2)
     {
       puts("Да2");
       AR=A; BR=B; CR=R;
       R=(AR|(BR>>1))<<3;
       printf("R=%d\n",R);
     }
    else
     {
       puts("Очистка A, B, C");
       A=A&0; B=B&0;  C=C&0;
       printf("A=%d B=%d C=%d\n",A,B,C);
     }
   }
 R=0z12;
 printf("R=%z\n",R);

 getch();
 return(0);
}
```

Литература

1. Язык С++ : учеб. пособие / И. Ф. Астахова [и др.]. – Минск : Новое знание, 2003. – 203 с.

2. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – Москва ; Санкт-Петербург : БИНОМ – Невский диалект, 1999.

3. Павловская, Т. А. С/С++. Программирование на языке высокого уровня / Т. А. Павловская. – Санкт-Петербург : Питер, 2002. – 464 с.

4. Мовшович, С. М. Методические указания к лабораторным занятиям по теме «Основы алгоритмизации» / С. М. Мовшович. – Гомель : Ротапринт ГПИ, 1988. – 25 с.

5. Касаткин, А. И. Профессиональное программирование на языке С++: От Turbo C к Borland C++ : справ. пособие / А. И. Касаткин, А. Н. Вальвачев ; под общ. ред. А. И. Касаткина. – Минск : Выш. шк., 1992. – 240 с.

6. Кравченко, О. А. Программирование ввода-вывода данных и линейных вычислительных алгоритмов на языке С : практ. пособие к выполнению лаборатор. и контрол. работ по дисциплине «Вычислительная техника и программирование» для студентов техн. специальностей днев. и заоч. форм обучения / О. А. Кравченко, А. М. Мартыненко. – Гомель : ГГТУ им. П. О. Сухого, 2005. – 33 с.

7. Кравченко, О. А. Программирование разветвляющихся и циклических алгоритмов на языке С : пособие по выполнению лаборатор. и контрол. работ по дисциплине «Вычислительная техника и программирование» для студентов техн. специальностей днев. и заоч. форм обучения / О. А. Кравченко, Е. В. Коробейникова. – Гомель : ГГТУ им. П. О. Сухого, 2005. – 33 с.

8. Информатика. Базовый курс : учеб. пособие / под ред. С. В. Симоновича. – 2-е изд. – Санкт-Петербург : Питер, 2007. – 639 с.

9. Павловская, Т. А. С/С ++. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. – Санкт-Петербург : Питер, 2006. – 460 с.

10. Павловская, Т. А. С#. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. – Сант-Петербург : Питер, 2007. – 432 с.

11. Острейковский, В. А. Информатика : учеб. для вузов / В. А. Острейковский. – Москва : Высш. шк., 2000. – 511 с.

12. Информатика : учебник / под ред. Н. В. Макаровой. – Москва : Финансы и статистика, 1998.

13. Топп, У. Структуры данных в С++ / У. Топп ; пер. с англ. У. Форд. – Москва : БИНОМ, 1994. – 816 с.

14. Крячков, А. В. Программирование на С и С++. Практикум : учеб. пособие для вузов / А. В. Крячков, И. В. Сухина, В. К. Томшин. – Москва : Горячая линия – Телеком, 2000 – 344 с.

15. Страуструп, Б. Язык программирования Си++ / Б. Страуструп. – Москва : Радио и связь, 1991. – 352 с.

Содержание

1. Строки и символы 3
1.1. Работа с символами 3
1.2. Строки 10
1.2.1. Понятие и описание строки 10
1.2.2. Ввод-вывод строк 11
1.2.3. Операции над строками 12
1.3. Примеры решения задач по обработке строк 21
2. Записи (структуры) 27
2.1. Определение записи (структуры) 27
3. Сортировка 53
3.1. Общие сведения 53
3.2. Классификация методов сортировки 54
3.3. Алгоритм сортировки методом извлечения 58
3.4. Алгоритм сортировки методом обменов 60
3.5. Минимизация числа просмотров при сортировке методом «пузырька» 61
3.6. Сортировка методом обменов за один просмотр «с возвращением» 62
3.7. Алгоритм сортировки методом слияния 64
3.8. Алгоритм сортировки распределением 67
3.9. Пример перестановки строк матрицы в порядке невозрастания сумм ее строк 70
3.10. Применение алгоритма сортировки методом «пузырька» для расположения записей в массиве записей в лексикографическом порядке 72
3.11. Использование встроенной функции qsort из библиотеки stdlib для сортировки массива записей 74
4. Файлы в языке С 76
4.1. Общие понятия 76
4.2. Работа с файлами (потоками) 79
4.2.1. Этапы работы с файлами (потоками) 79
4.3. Ввод-вывод в поток 83
4.3.1. Основные понятия 83
4.3.2. Позиционирование в файле 84
4.3.3. Функции чтения и записи потока байтов fread и fwrite 85

4.3.4. Функции чтения символа из потока (getc, fgetc, getchar) 85
4.3.5. Функции записи символа в поток (putc, fputc, putchar) 86
4.3.6. Функции чтения строки из потока (fgets, gets) 86
4.3.7. Функции записи строки в поток (fputs, puts) 86
4.3.8. Функции форматированного ввода (чтения) из потока (fscanf, scanf, sscanf) 87
4.3.9. Функции форматированного вывода в поток (fprintf, printf, sprintf) 87
4.4. Обработка ошибок 88
4.5. Пример обработки текстового файла 88
4.6. Пример обработки текстового и бинарного файла 91
5. Поразрядные логические операции 93
5.1. Размещение данных в памяти ПЭВМ 93
5.2. Побитовые операции 94
5.3. Примеры работы с битами 97
Литература 108

Учебное электронное издание комбинированного распространения

Учебное издание

Кравченко Ольга Алексеевна
Мовшович Семен Михайлович
Коробейникова Евгения Васильевна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Курс лекций по одноименной дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения

Электронный аналог печатного издания

Редактор *Н. И. Жукова*
Компьютерная верстка *Н. Б. Козловская*

Подписано в печать 24.05.10.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 6,51. Уч.-изд. л. 6,2.
Изд. № 198.
E-mail: ic@gstu.by
http://www.gstu.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.