

Сергей Истрати

Примечание ! Содержание этой книги переведено с румынского на русский язык не совсем компетентными “переводчиками”. Прошу не считать этот перевод официально-правильным. Советую использовать румынский оригинал с сайта или из библиотеки.

Программирование

Задания и методические указания по
выполнению курсовой работы



Кишинёв 2005

Данные методические указания предназначены для студентов I и II курса дневного и заочного отделений Технического Университета Молдовы, изучающих программирование, а в частности, для студентов Факультета Радиоэлектроники и Телекоммуникаций Кафедры Оптоэлектронных систем, специальности 1871 Инженерия и Менеджмент в телекоммуникациях и 2105 Оптоэлектронные системы.

Методические указания включают в себя условия заданий и описание методов и средств выполнения курсовой работы по предмету Программирование.

Автор: Старший лектор Сергей Истрати

Ответственный редактор: конф. унив. дк. Павел Нистирюк

Рецензент: академик, дк. хабилитат Виктор Боршевич

© Т .У.М. 2005

Содержание

Введение	4
1. Варианты заданий курсовой работы	5
1.1.1 Обработка двумерных массивов. Общие указания	5
1.1.2 Пример решения одного задания	6
1.1.3 Варианты	12
1.2.1 Обработка баз данных. Общие указания	15
1.2.2 Пример решения задания	17
1.2.3 Варианты	23
2. Указания по содержанию курсовой работы	27
3. Приложение 1. Массивы языка С.	28
4. Приложение 2. Структуры языка С	31
5. Приложение 3. Файлы в языке С	37
6. Список рекомендуемой литературы	49

Введение

Цель данной работы заключается в демонстрации методов выполнения и инструментов, с помощью которых будет выполнено задание курсовой работы по предмету Программирование.

Выполнение курсовой работы предполагает решение задач на языке С. Для достижения этой цели предлагается несколько типов задач, которые содержат главные аспекты программирования на языках С++ и Pascal. Таким образом, предложены разные варианты обработки двумерных массивов и баз данных, решение которых предполагает использование самых разнообразных методов и инструментов, которыми располагают вышеназванные языки программирования.

Обработка двумерных массивов предполагает вычисление некоторого поля, или периметра этого поля, вычисление минимальных и максимальных элементов в различных строках, колонках, заштрихованных областей в возрастающем порядке, убывающем или обратном и др.

Обработка некоторой базы данных предполагает сбор разной информации и распределение её в базе данных, для составления которой будет использован массив записи типа статьи. Полученная информация будет распределена по-разному, руководствуясь разными показателями. Дополнительно к этому необходимо присутствие диалога между программой и пользователем во время обработки данных, для реализации которого предлагается создание некоторого меню, основанного на использовании указания - comutator switch (case).

Помимо этого для более эффективного решения условия задач предлагается использование некоторых вспомогательных инструментов, например, использование подпрограмм: процедур или функций, хранение информации в папках, оформление интерфейса программ с помощью цветов и звуков.

Представленные методические указания представляют собой часть из сюиты дидактико-методических работ, составленных старшим лектором Сергеем Г. Истрати, направленных на оптимизацию процесса обучения студентов предмету Программирование. Были составлены следующие работы:

- Лекционный курс по Программированию. Язык Pascal.
- Лекционный курс по Программированию. Язык С.

- Методические указания для выполнения лабораторных работ.
- Методические указания для выполнения индивидуальных работ.
- Методические указания для выполнению курсовых работ (данная работа).

Все эти работы доступны в Internet по адресу www.istrati.com

1. Варианты заданий курсовых работ

Выполнение курсовой работы подразумевает выполнение одной задачи в языке C++ или Pascal. Предлагаются 2 типа задач:

- 1) обработка двумерных массивов;
- 2) обработка баз данных;

решая которые необходимо использовать самые разные методы программирования и инструменты, которыми располагают данные языки программирования.

1.1.1. Обработка двумерных массивов. Общие указания

Задания из каждого варианта включают в себя по одному двумерному массиву $x [n,n]$, который содержит заштрихованную в определённом порядке область для каждого варианта. Решение задачи сводится к обработке элементов из заштрихованной и не заштрихованной области согласно условию варианта.

Первым шагом в решении задания служит вывод условия задания на монитор. Заполнение массива, то есть присвоение значения элементам массива, которое будет выполнено двумя способами:

- 1) вручную;
- 2) автоматически;

После заполнения массива, он должен будет быть объявлен в удобной для чтения форме, то есть элементы должны быть расположены по строкам и столбцам. Более того, массив должен быть выведен в двух цветах: элементы закрашенной области одного цвета, а элементы не закрашенной области другого цвета.

Затем следует выполнение каждого из трёх пунктов условия. Для этого будут созданы несколько подпрограмм, каждая из которых решает определённое условие. Количество подпрограмм будет зависеть от условия задачи и от желания исполнителя курсовой работы.

После выполнения всех трёх пунктов условия и нахождения результатов, финальный массив будет выведен на монитор.

С целью сделать программу более удобной для использования, в ней будет создано меню, с помощью которого будет легко переходить от одной стадии вычисления к другой. Таким образом, меню должно содержать следующие пункты:

- 1) Выведение условия задачи.
- 2) Ручное заполнение массива.
- 3) Автоматическое заполнение массива.
- 4) Выведение массива (в 2 цветах).
- 5) Выполнение пункта а) из условия.
- 6) Выполнение пункта б) из условия.
- 7) Выполнение пункта с) из условия.
- 8) Выведение финального массива (в 2 цветах).
- 9) Выход

После выполнения каждого из 3 пунктов, соответствующий результат должен быть выведен на монитор. Пункты 5,6,7 из меню соответствуют выполнению пунктов а,б,с из условия, они могут быть объединены в меньшее число пунктов или же разбиты на большее число пунктов из меню в зависимости от условия задачи и желания исполнителя курсовой работы.

1.1.2. Пример выполнения одного условия

Дан двумерный массив $x[n][n]$ с заштрихованной областью



- a) *Найти значения и позиции максимальных элементов из обеих заштрихованных областей*
- b) *Записать в файл сумму чётных элементов из заштрихованной области*
- c) *Упорядочить каждый ряд из заштрихованной области в возрастающем порядке*

Листинг программы:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
//выведение условия задачи на монитор
void uslovie (void) {
int i,j,n=10,k; clrscr();
```

```

gotoxy(20,2);textcolor(11);
printf("Курсовая работа по Программированию");
gotoxy(29,4);textcolor(15); printf("Условие задачи:");
gotoxy(10,5);textcolor(11);
printf("Дан двумерный массив x[N][N] С заштрихованной областью
(красная)");
gotoxy(5,6); printf("a) Найдите значения и позиции максимальных
элементов");
gotoxy(5,7); printf(" из обеих частей заштрихованной области");
gotoxy(5,8);
printf("b) Записать в файл сумму чётных элементов из
заштрихованной области");
gotoxy(5,9);
printf("c) Упорядочить каждую строку в возрастающем порядке");
k=floor(float(n)/float(2));
for(i=0;i<n;i++) {
for(j=0;j<n;j++) {
if( (i<k) && ( j>=i) && (j<(n-i)) )
{gotoxy(15+(j*3),11+i);textcolor(12); printf("* ");}
else if( (i>=k) && ( j>=(n-i-1)) && (j<=i) )
{gotoxy(15+(j*3),11+i);textcolor(12);printf("* ");}
else {gotoxy(15+(j*3),11+i);textcolor(10);printf("* ");} } }
textcolor(15);gotoxy(10,wherey()+2);
printf("Для выхода в меню нажмите <ENTER>"); getch(); }
//вывод исходного массива в двух цветах на монитор
void vivod(int y[50][50],int m) {
int i,j,k; clrscr();gotoxy(15,2);textcolor(15);
printf("Массив x[%d][%d] :\n",m,m); k=floor(float(m)/float(2));
for(i=0;i<m;i++) {
for(j=0;j<m;j++) {
if( (i<k) && ( j>=i) && (j<(m-i)) )
{gotoxy(15+(j*4),4+i);textcolor(12); printf("%3d ",y[i][j]);}
else if( (i>=k) && ( j>=(m-i-1)) && (j<=i) )
{gotoxy(15+(j*4),4+i);textcolor(12);printf("%3d ",y[i][j]);}
else {gotoxy(15+(j*4),4+i);textcolor(10);printf("%3d ",y[i][j]);} } }
textcolor(15);gotoxy(10,wherey()+2);
printf("Для выхода в меню нажмите <ENTER>"); getch(); }
//определение значений и позиций максимальных элементов из 2
заштрихованных областей
void maximum (int y[50][50],int m) {
int i,j,k,max1,rmax1,сmax1,max2,rmax2,сmax2;

```

```

//верхняя часть заштрихованной области.
k=floor(float(m)/float(2));
max1=y[0][0]; rmax1=0; cmax1=0;
for(i=0;i<k;i++) {
for(j=i;j<(m-i);j++) {
if (y[i][j]>=max1) {max1=y[i][j]; rmax1=i; cmax1=j;} } }
//нижняя часть заштрихованной области.
max2=y[m-1][0]; rmax2=m-1; cmax2=0;
for(i=k;i<m;i++) {
for(j=(m-i-1);j<=i;j++) {
if (y[i][j]>=max2) {max2=y[i][j]; rmax2=i; cmax2=j;} } }
//вывод результатов
clrscr();gotoxy(15,2);textcolor(15);
printf("Максимальные элементы из заштрихованной области:");
for(i=0;i<m;i++) {
for(j=0;j<m;j++) {
if( (i<k) && ( j>=i) && (j<(m-i)) )
{gotoxy(15+(j*4),4+i);textcolor(12);cprintf("%3d ",y[i][j]);}
else if( (i>=k) && ( j>=(m-i-1)) && (j<=i) )
{gotoxy(15+(j*4),4+i);textcolor(12);cprintf("%3d ",y[i][j]);}
else {gotoxy(15+(j*4),4+i);textcolor(10);cprintf("%3d ",y[i][j]);}
if( ((i==rmax1)&&(j==cmax1)) || ((i==rmax2)&&(j==cmax2)) )
{gotoxy(15+(j*4),4+i);textcolor(15);cprintf("%3d ",y[i][j]);} } }
textcolor(15);gotoxy(10,wherey()+2);
cprintf("Максимальный элемент из верхней заштрихованной части
x[%d][%d] = %d",rmax1,cmax1,y[rmax1][cmax1]);
gotoxy(10,wherey()+1);
cprintf("Максимальный элемент из нижней заштрихованной части
x[%d][%d] = %d",rmax2,cmax2,y[rmax2][cmax2]);
gotoxy(10,wherey()+2);
cprintf("Для выхода в меню нажмите <ENTER>");
getch(); }
//вычисление суммы чётных элементов и их запись в файл
void file (int y[50][50],int m) {
int i,j,k,s=0; FILE *f;
k=floor(float(m)/float(2));
//верхняя часть заштрихованной области.
for(i=0;i<k;i++) {
for(j=i;j<(m-i);j++) {
if (fmod(y[i][j],2)==0) s+=y[i][j]; } }
//нижняя часть заштрихованной области.

```

```

for(i=k;i<m;i++) {
for(j=(m-i-1);j<=i;j++) {
if (fmod(y[i][j],2)==0) s+=y[i][j]; } }
if((f=fopen("c:/suma.txt", "w"))==NULL) {
clrscr();gotoxy(15,2);textcolor(15);
sprintf("Файл не может быть открыт"); goto exit;}
fprintf(f,"Сумма чётных элементов из заштрихованной области
S=%d",s); fclose(f);
clrscr();gotoxy(15,2);textcolor(15);
sprintf("В файл c:\\\\suma.txt были записаны следующие данные:");
textcolor(11);gotoxy(15,wherey()+2);
sprintf("Сумма чётных элементов из заштрихованной области
S=%d",s);
exit: textcolor(15);gotoxy(15,wherey()+2);
sprintf("Для выхода в меню нажмите <ENTER>"); getch(); }
// упорядочение значений заштрихованной области в возрастающем
порядке
void uporeadocenie (int(*k)[50],int z) {
int i,j,g,c,min,aux,w;
w=floor(float(z)/float(2));
//упорядочение в возрастающем порядке верхней части
for(i=0;i<w;i++) {
for(j=i;j<(z-i);j++) {
min=*(k[i]+j);c=j;
for(g=j;g<(z-i);g++) {
if (*(k[i]+g)<min) {min=*(k[i]+g);c=g;}}
aux=*(k[i]+j); *(k[i]+j)=*(k[i]+c); *(k[i]+c)=aux;}}
//упорядочение в возрастающем порядке нижней области
for(i=w;i<z;i++) {
for(j=(z-i-1);j<=i;j++) {
min=*(k[i]+j);c=j;
for(g=j;g<=i;g++) {
if (*(k[i]+g)<min) {min=*(k[i]+g);c=g;}}
aux=*(k[i]+j); *(k[i]+j)=*(k[i]+c); *(k[i]+c)=aux;}}
clrscr();gotoxy(15,2);textcolor(15);
sprintf("Элементы каждого ряда из заштрихованной области ");
textcolor(15);gotoxy(15,wherey()+1);
sprintf("были упорядочены в возрастающем порядке");
textcolor(11);gotoxy(15,wherey()+2);
sprintf("Чтобы увидеть результат выберите 8 пункт меню");
textcolor(15);gotoxy(15,wherey()+2);

```

```

printf("Для выхода в меню нажмите <ENTER>"); getch(); }
//обработка исключения, когда элементы массива не имеют значений
void osibka (int q) {
if (q==0) { clrscr();gotoxy(15,2);textcolor(12);
printf("Массив не может быть обработан. Элементы не имеют
значений");
textcolor(15);gotoxy(15,wherey()+2);
printf("Для выхода в меню нажмите <ENTER>"); getch(); } }
//главная программа
void main(void) {
int x[50][50]={0},t[50][50]={0};
int n,i,j,z=0,h=0; char w,v;
//вывод минимума
m0:clrscr(); textcolor(15); gotoxy(15,2);
printf("Выбери из меню:\n"); gotoxy(15,4);
printf("1: Вывод исходных условий\n"); gotoxy(15,5);
printf("2: Ручное заполнение массива\n"); gotoxy(15,6);
printf("3: Автоматическое заполнение массива\n"); gotoxy(15,7);
printf("4: Вывод исходного массива\n"); gotoxy(15,8);
printf("5: Вычисление максимальных элементов из заштрихованных
областей\n"); gotoxy(15,9);
printf("6: Запись суммы в файл\n"); gotoxy(15,10);
printf("7: Упорядочение элементов заштрихованной области\n");
gotoxy(15,11);
printf("8: Вывод финального массива\n"); gotoxy(15,12);
printf("9: Выход\n"); gotoxy(15,13);
// выбор из меню
w=getch();
switch (w) {
case '1': goto m1;
case '2': goto m2;
case '3': goto m3;
case '4': goto m4;
case '5': goto m5;
case '6': goto m6;
case '7': goto m7;
case '8': goto m8;
case '9': goto m9;
default : goto m0;}
m1:условия(); goto m0;
m2: //ручное заполнение массива

```

```

clrscr(); z=1; gotoxy(15,2); textcolor(15);
printf("Введите размер массива x N=");
scanf("%d",&n); gotoxy(15,3);
printf("Введите элементы массива x[%d][%d]\n",n,n);
for(i=0;i<n;i++) {
for(j=0;j<n;j++) {
gotoxy(15+(j*4),5+i);
scanf("%3d",&x[i][j]); } }
textcolor(15);gotoxy(10,wherey()+1);
printf("Для выхода в меню нажмите <ENTER>"); getch(); goto m0;
m3: //автоматическое заполнение массива
clrscr(); z=1; gotoxy(15,2); textcolor(15);
printf("введите размер массива x N=");
scanf("%d",&n); randomize();
for(i=0;i<n;i++) {
for(j=0;j<n;j++) {
x[i][j]=random(100)-50; }}
// вывод исходного массива
gotoxy(15,4); textcolor(15);
printf("Исходный массив x[%d][%d] \n",n,n);
for(i=0;i<n;i++) {
for(j=0;j<n;j++) {
textcolor(15); gotoxy(15+(j*4),6+i);
printf("%3d ",x[i][j]); } }
gotoxy(10,wherey()+2);
printf("Для выхода в меню нажмите <ENTER>");
getch(); goto m0;
m4: if (z==1) {vivod(x,n); goto m0;};
osibka(z); goto m0;
m5: if (z==1) {maximum(x,n); goto m0;};
osibka(z); goto m0;
m6: if (z==1) {file(x,n); goto m0;};
osibka (z); goto m0;
m7: if (z==1) { for(i=0;i<n;i++) for(j=0;j<n;j++) t[i][j]=x[i][j];
uporeadocenie(t,n); h=1; goto m0; };
osibka (z); goto m0;
m8: if (h==1) {vivod(t,n); goto m0;};
if (h==0) { clrscr();gotoxy(15,2);textcolor(12);
printf("Массив не может быть выведен. Элементы не были
упорядочены");
textcolor(15);gotoxy(15,wherey()+2);





```


```






printf("Для выхода в меню нажмите <ENTER>");
getch(); goto m0;}
m9: clrscr();
gotoxy(15,2); textcolor(15);
printf("Вы действительно хотите выйти ? y/n");
v=getch(); if ((v=='n')||(v=='N')) goto m0;
gotoxy(15,4); textcolor(15);
printf("Для выхода из программы нажмите <ENTER>");
getch(); }

```


1.1.3. Варианты

Вар.	Условие	Массив
1.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Вычислить среднее арифметическое нечётных элементов из заштрихованной области.</p> <p>б) Найти позиции и числа из массива, равных цифре пользователя.</p> <p>с) Если элемент, равный цифре пользователя в массиве не существует следует записать соответствующий комментарий и издать звуковой сигнал.</p>	
2.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти максимальные элементы между заштрихованной областью и соседней областью.</p> <p>б) Вычислить среднее арифметическое чётных элементов из всей заштрихованной части.</p> <p>с) Заменить все элементы из не заштрихованной части на ноль.</p>	
3.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Вычислить сумму первого и последнего чётного элемента из заштрихованной области и их позиции.</p> <p>б) Найти сколько равных этой сумме элементов находится в не заштрихованной области.</p> <p>с) Записать в файл значения и позиции всех элементов, граничащих с самым маленьким положительным элементом из заштрихованной области.</p>	
4.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Вычислить среднее арифметическое всех элементов делящихся на 4 из заштрихованной области.</p> <p>б) Приравнять все элементы из не заштрихованной</p>	

	<p>области к нулю жёлтого цвета.</p> <p>с) Найти значение и позицию максимальных элементов из каждой строки заштрихованной области.</p>	
5.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Показать чётные элементы с их соседними элементами из заштрихованной области (указать значения и позиции)</p> <p>б) Заменить отрицательный элемент из не заштрихованной области на "8".</p> <p>с) Записать в файл количество нечётных элементов из заштрихованной области.</p>	
6.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти среднее арифметическое элементов >5 из заштрихованной области.</p> <p>б) Записать в файл количество элементов равных числу пользователя из заштрихованной области.</p> <p>с) Вычислить сумму элементов, расположенных по периметру не заштрихованной области.</p>	
7.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти среднее арифметическое чётных элементов из заштрихованной области и записать в файл.</p> <p>б) Найти максимальный элемент из каждого чётного столбца заштрихованной области.</p> <p>с) Найти сколько элементов из заштрихованной области равны числу пользователя.</p>	
8.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти среднее арифметическое элементов по периметру заштрихованной области.</p> <p>б) Найти количество чётных элементов из нечётных столбцов заштрихованной области.</p> <p>с) Найти самый большой элемент из каждой не заштрихованной области и записать его в файл.</p>	
9.	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти значение и позицию каждого чётного элемента из заштрихованной области.</p> <p>б) Упорядочить в убывающем порядке элементы каждого столбца заштрихованной области.</p> <p>с) Найти количество чётных элементов из области " *".</p>	

10	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Упорядочить в возрастающем порядке элементы каждой строки из заштрихованной области.</p> <p>б) Найти максимальный элемент каждого столбца из заштрихованной области.</p> <p>с) Найти количество элементов из не заштрихованной области, равные числу пользователя и запишите его в файл.</p>	
11	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти позицию и количество элементов, равных числу пользователя.</p> <p>б) Найти максимальный элемент из каждого нечётного столбца из заштрихованной области.</p> <p>с) Запишите в файл сумму отрицательных элементов из массива.</p>	
12	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Вычислить произведение нечётных элементов из заштрихованной области.</p> <p>б) Найти максимальный элемент по периметру заштрихованной области.</p> <p>с) Запишите в файл значения и позиции элементов из каждого угла заштрихованной области.</p>	
13	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти среднее арифметическое элементов, делящихся на 3 из заштрихованной области.</p> <p>б) Найти количество нечётных элементов из каждого чётного столбца заштрихованной области.</p> <p>с) Записать в файл количество нулей из не заштрихованной области.</p>	
14	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти сколько элементов из заштрихованной области равны числу пользователя.</p> <p>б) Найти значение и позицию минимальных элементов из каждого ряда заштрихованной области.</p> <p>с) Записать в файл сумму чётных элементов из не заштрихованной области</p>	

15	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Упорядочить в возрастающем порядке элементы каждого столбца из заштрихованной области.</p> <p>б) Найти количество элементов, равных числу пользователя из не заштрихованной области</p> <p>с) Записать в файл произведение нечётных элементов из не заштрихованной области.</p>	
16	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти среднее арифметическое чётных элементов из заштрихованной области.</p> <p>б) Показать чётные элементы с их соседними элементами из заштрихованной области (указать значения и позиции)</p> <p>с) Сравнить суммы элементов из заштрихованной и не заштрихованной области.</p>	
17	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти сумму и позицию первого и последнего чётного элемента из заштрихованной области.</p> <p>б) Найти сумму элементов по периметру не заштрихованной области.</p> <p>с) Сравнить произведения нечётных элементов из заштрихованной и не заштрихованной области.</p>	
18	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Найти сумму и позицию первого и последнего не чётного элемента из заштрихованной области.</p> <p>б) Найти сумму элементов по периметру не заштрихованной области.</p> <p>с) Сравнить произведения нечётных элементов из заштрихованной и не заштрихованной области.</p>	
19	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Записать в файл значения и позиции всех соседних элементов самого большого положительного элемента из заштрихованной области.</p> <p>б) Найти среднее арифметическое всех элементов по периметру заштрихованной области.</p> <p>с) Сравнить среднее арифметическое положительных элементов из заштрихованной и не заштрихованной области.</p>	

20	<p>Дан двумерный массив $x[n,n]$.</p> <p>а) Упорядочить в убывающем порядке элементы каждой строки из заштрихованной области.</p> <p>б) Приравнять все элементы из не заштрихованной области к нулю жёлтого цвета.</p> <p>с) Сравнить среднее арифметическое положительных элементов из заштрихованной и не заштрихованной области.</p>	
----	--	---

1.2.1. Обработка баз данных. Общие указания.

Условия из каждого варианта включают по одной базе данных, которая содержит специальную информацию для каждого варианта в отдельности. Решение задачи сводится к решению базы данных, согласно условию варианта, которое включает в себя поиск в базе данных по любому полю, упорядочение записей и другое.

Первым шагом к решению задачи послужит выведение условия задачи на монитор. Потом пользователь получит запрос, чтобы указать необходимое число записей, из которых и будет составлена база данных. Заполнение базы данных, то есть присвоение значений каждому полю из записи будет производиться вручную, путем введения данных с клавиатуры. После этого, база данных будет записана в папку с именем “baza.txt” для того, чтобы потом можно было её читать, выводить и обрабатывать.

После заполнения базы данных и записи её в папку “baza.txt”, она будет прочитана из папки и выведена на монитор в удобной для чтения форме, то есть вся выведенная информация будет расположена в таблице, которая будет содержать заглавие с названием полей и с конкретным числом записей. Более того, база данных должна быть выведена в нескольких цветах, таким образом, чтобы две соседние записи были разных цветов.

После выведения базы данных, пользователь получает запрос на введение информации в ещё одну запись. Если пользователь согласится с запросом, тогда эта запись будет записана прямо в папку “baza.txt” и расположена в конце базы данных. После введения этой информации запрос будет повторен. После этого заполненная база данных будет выведена на монитор.

Потом следует решение всех условий задачи. Для этой цели будут созданы несколько подпрограмм, каждая из которых решает конкретное условие. Так же для вышеописанных действий, то есть

заполнения базы данных, запись в папку и её вывод на монитор, так же могут быть использованы отдельные подпрограммы. Количество подпрограмм будет зависеть от условия программы и желания исполнителя курсовой работы.

После выполнения условия и вывода полученных результатов на монитор, финальная база данных будет заново выведена на монитор в удобной для чтения форме.

Если условие работы предполагает упорядочение записи в определённом порядке, тогда финальная база с упорядоченными записями будет записана в другой папке с именем "newbaza.txt".

С целью сделать программу более удобной в применении, в ней будет создано меню, с помощью которого будет легко переходить от одного этапа вычисления к другому. Таким образом, программа должна содержать следующие пункты меню:

1. Вывод условия задачи.
2. Заполнение базы данных и её запись в папку "baza.txt".
3. Чтение базы данных из папки "baza.txt" и вывод её на монитор.
4. Введение новой записи и вывод новой базы на монитор.
5. Поиск в базе данных.
6. Другие условия, согласно условию.
7. Упорядочение базы данных и запись её в папку "newbaza.txt".
8. Вывод финальной упорядоченной базы данных.
9. Выход.

После каждого пункта меню, программа снова вернётся к выводу меню. После решения каждого пункта, соответствующий результат будет выведен на монитор. Пункты 5,6,7 из меню, которые соответствуют решению условию варианта, могут быть объединены в меньшее число пунктов или разделены на большее число пунктов меню, в зависимости от условия задачи и от желания исполнителя курсовой работы.

1.2.2. Пример решения одного условия.

Составить базу данных с N записями, которые должны содержать информацию о компьютерах в кабинете для лабораторных работ по программированию.

- a) Организовать поиск компьютеров по имени и операционной системе.*
- b) Найти компьютеры с памятью ROM в интервале, предложенном пользователем.*
- c) Упорядочить записи в убывающем порядке памяти RAM.*

Listing программы:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
//глобальные переменные
int z=0;
struct pc {
char name[15],so[22];
float rom,ram;}aux;
struct pc x[50];

//вывод исходного условия
void uslovie (void) {
clrscr(); gotoxy(20,2);textcolor(11);
printf("Курсовая работа по программированию");
gotoxy(29,4);textcolor(15); printf("Условия задачи:");
gotoxy(10,5);textcolor(11);
printf("Составить базу данных с N записями, которые содержали
бы информацию");
gotoxy(10,6);
printf("о компьютерах в компьютерном классе.");
gotoxy(5,7);
printf("a)Организовать поиск компьютера по имени и операционной
системе");
gotoxy(5,8);
printf("b)Найти компьютеры с памятью ROM в интервале,
предложенном пользователем");
gotoxy(5,9);
printf("c)Упорядочить записи в убывающем порядке памяти
RAM");gotoxy(5,wherey()+2);
printf("┌──────────────────────────────────────────────────────────────────────────────────┐");
gotoxy(5,wherey()+1);
printf("┆      Имя      ┆          O C          ┆      ROM (Gb)      ┆      RAM (Mb)      ┆");
gotoxy(5,wherey()+1);
printf("┌──────────────────────────────────────────────────────────────────────────────────┐");
gotoxy(5,wherey()+1);
printf("┆      ┆      ┆      ┆      ┆");
gotoxy(5,wherey()+1);
```



```

clrscr();gotoxy(15,2);textcolor(15);
sprintf("Файл не может быть открыт"); z=1; goto exit;} i=0; z=0;
while( fread(&x[i],sizeof(x[i]),1,f)==1 ) i++; fclose(f);
exit: return(i);}

```

//вывод базы на монитор

```

void vivod (int k, char namef[20]) { int i;
if (z==1) {clrscr(); gotoxy(5,2); textcolor(15);
sprintf ("База данных не была прочитана из файла"); goto exit;}
else { clrscr(); gotoxy(5,2); textcolor(15);
sprintf("База данных, прочитанная из файла %s
:",namef);gotoxy(5,wherey()+2);
textcolor(15);
printf(" ┌──────────────────────────────────────────────────────────────────────────────────┐");
gotoxy(5,wherey()+1);
printf(" │      Имя      │          О С          │  ROM (Gb)  │  RAM (Mb)  │");
gotoxy(5,wherey()+1);
printf(" └──────────────────────────────────────────────────────────────────────────────────┘");
gotoxy(5,wherey()+1); for(i=0;i<k;i++){
if (fmod(i,2)==0) {textbackground(12); textcolor(15);} else
{textbackground(BLUE); textcolor(10);}
sprintf(" │ %15s │ %22s │ %12.3f │ %12.3f │ ",x[i].name,x[i].so,x[i].rom,x[i]
.ram);
gotoxy(5,wherey()+1);} textbackground(BLACK); textcolor(15);
printf(" ┌──────────────────────────────────────────────────────────────────────────────────┐");
} exit: gotoxy(5,wherey()+2);
sprintf("Для выхода в меню нажмите <ENTER>"); getch();}

```

```

void add (char namef[20]) {
struct pc s; float a,b; FILE *f; int i; char q;
if((f=fopen(namef,"a"))==NULL) { clrscr();gotoxy(5,2);textcolor(15);
sprintf("Файл не может быть открыт"); goto exit;}
ad: clrscr();gotoxy(5,wherey()+2);textcolor(15);
sprintf("Хотите ли вы добавить еще одну запись в базу данных? у/н
");
q=getch(); if ((q=='y')||(q=='Y')) {
gotoxy(10,wherey()+2); textcolor(15);
sprintf("Назовите имя станции "); scanf("%s",s.name);
gotoxy(10,wherey()+1);
sprintf("Назовите операционную систему "); scanf("%s",s.so);
gotoxy(10,wherey()+1);

```

```

printf("Укажите величину памяти ROM (в Gb) ");scanf("%f",&a);
s.rom=a;
gotoxy(10,wherey()+1);
printf("Укажите величину памяти RAM (в Mb) ");scanf("%f",&b);
s.ram=b;
f=fopen(namef,"a"); fwrite(&s,sizeof(s),1,f); fclose(f); goto ad;}
exit: gotoxy(5,wherey()+2); textcolor(15);
printf("Для выхода в меню нажмите <ENTER>"); getch(); }

// поиск по имени, операционной системе и интервалу памяти ROM
void poisk(int k,char poisk[22]) {
int i,g; char Z[22],B[22];float niz,verh,R;
if (z==1) {clrscr(); gotoxy(5,2); textcolor(15);
printf("База данных не была прочитана из файла"); goto exit;}
else { clrscr(); gotoxy(5,2); textcolor(15);
if (strcmp(poisk,"Name")==0) {g=1;
printf("Назовите имя искомой станции "); scanf("%s",Z);}
if (strcmp(poisk,"Операционная Система")==0) {g=2;
printf("Введите искомую Операционную Систему "); scanf("%s",Z);}
if (strcmp(poisk,"память ROM")==0) {g=3;
printf("Введите нижний предел памяти ROM (Gb) ");
scanf("%f",&niz);
gotoxy(5,wherey());
printf("Введите верхний предел памяти ROM (Gb) ");
scanf("%f",&verh);}
gotoxy(5,wherey()+1); textcolor(15);
printf("Результат поиска по %s :",poisk);gotoxy(5,wherey()+2);
printf(" ┌──────────────────────────────────────────────────────────────────────────────────┐");
gotoxy(5,wherey()+1);
printf(" │      Имя      │          O C          │ ROM (Gb) │ RAM (Mb) │");
gotoxy(5,wherey()+1);
printf(" └──────────────────────────────────────────────────────────────────────────────────┘");
gotoxy(5,wherey()+1); for(i=0;i<k;i++){ textcolor(15);
if (g==1) {strcpy(B,x[i].name); if ( strstr(strlwr(B),strlwr(Z))!=0 ) {
printf(" │ %15s │ %22s │ %12.3f │ %12.3f │ ",x[i].name,x[i].so,x[i].rom,x[i]
.ram);
gotoxy(5,wherey()+1);} }
if (g==2) {strcpy(B,x[i].so); if ( strstr(strlwr(B),strlwr(Z))!=0 ) {
printf(" │ %15s │ %22s │ %12.3f │ %12.3f │ ",x[i].name,x[i].so,x[i].rom,x[i]
.ram);
gotoxy(5,wherey()+1);} } }

```



```

printf("      и запись базы данных в файл  c:\\newbaza.txt");
gotoxy(5,wherey()+1);
printf("9: Вывод финальной базы данных на монитор");
gotoxy(5,wherey()+1);
printf("0: выход");
w=getch();
switch (w) {
case '1': goto m1;
case '2': goto m2;
case '3': goto m3;
case '4': goto m4;
case '5': goto m5;
case '6': goto m6;
case '7': goto m7;
case '8': goto m8;
case '9': goto m9;
case '0': goto m0;
default : goto meniu;}

m1: uslovie(); goto meniu;
m2: n=zapolnenie(); zap_papka (n,baza); goto meniu;
m3: n=ctenie_file(baza); vivod(n,baza); goto meniu;
m4: add(baza); goto meniu;
m5: n=cteniee_file(baza); poisk(n,"Имя"); goto meniu;
m6: n= cteniee_file (baza); poisk (n,"Операционная Система"); goto
meniu;
m7: n= cteniee_file (baza); poisk (n,"память ROM"); goto meniu;
m8: n= cteniee_file (baza); uporeadocenie(n); zap_papka(n,newbaza);
goto meniu;
m9: n= cteniee_file (newbaza); vivod(n,newbaza); goto meniu;
m0: clrscr();
gotoxy(15,2); textcolor(15);
printf("Вы действительно хотите выйти ? y/n");
v=getch(); if ((v=='n')||(v=='N')) goto meniu;
gotoxy(15,4); textcolor(15);
printf("Для выхода из программы нажмите <ENTER>");
getch(); }

```

1.2.3. Варианты

Вар.	Условие
21	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о дате регистрации автомобиля.</p> <p>а) Расположить все записи в убывающем порядке по числам (день, месяц, год) регистрации.</p> <p>б) Организовать поиск на основании любого автомобиля по дате, числу и марке автомобиля.</p>
22	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о студентах какой-либо группы.</p> <p>а) Вычислить стипендию каждому студенту, исходя из результатов какой-либо сессии.</p> <p>б) Расположить записи по величине стипендии в убывающем порядке.</p> <p>с) Записать в файл число студентов, получающих стипендию и студентов, не получающих стипендию.</p>
23	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о паспортных данных группы лиц.</p> <p>а) Организовать поиск какой-либо личности по номеру паспорта, имени личности, году рождения.</p> <p>б) Расположить запись в алфавитном порядке по фамилиям.</p>
24	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о производственном процессе на каком-либо заводе, где изготавливаются 3 вида деталей.</p> <p>а) Организовать поиск рабочих по фамилиям.</p> <p>б) Вычислить рабочих, которые изготавливают минимальное и максимальное количество деталей, отдельно по всем 3 видам деталей.</p> <p>с) Расположить запись в алфавитном порядке по фамилиям.</p>
25	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о троллейбусных маршрутах и времени пребывания на каждую остановку.</p> <p>а) Чтобы была возможность узнать какие троллейбусы доедут до указанной остановки в установленное время.</p> <p>б) Самое позднее время прибытия на остановку каждого троллейбуса должно быть записано в файл.</p> <p>с) Расположить записи в возрастающем порядке по времени прибытия на первую остановку.</p>

26	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о работниках какой-либо организации.</p> <p>а) Расположить записи в убывающем порядке по величине зарплаты.</p> <p>б) Организовать поиск работников по фамилиям и занимаемой должности.</p> <p>с) Информация о директоре должна быть в отдельном файле.</p>
27	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о расписании лекций студента в каждый день недели.</p> <p>а) Чтоб была возможность узнать расписание на любой день.</p> <p>б) Чтобы была возможность узнать будет ли иметь лекции студент в данной аудитории в день, указанный пользователем.</p> <p>с) Узнать какие лекции имеет студент в указанное время в среду.</p> <p>д) Узнать предметы с минимальным и максимальным количеством часов на протяжении недели.</p>
28	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о номерах телефонов какой-либо группы лиц.</p> <p>а) Расположить записи в возрастающем порядке по телефонным станциям (первые 2-цифры номера).</p> <p>б) Чтобы была возможность узнать всю информацию о владельце номера телефона, указанного пользователем.</p> <p>с) Последняя запись в файле должна быть записана в отдельном файле.</p>
29	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о какой-то группе студентов.</p> <p>а) Чтоб было возможно узнать список студентов, рождённых в о дном году, месяце и дня, предложенном пользователем.</p> <p>б) Чтоб можно было узнать список студентов, рождённых в определённый период лет, указанных пользователем.</p> <p>с) Расположить записи в убывающем порядке по возрасту студентов.</p>
30	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о какой-то группе сотрудников.</p> <p>а) Чтоб можно было узнать сколько сотрудников получают зарплату, равной величине, предложенной пользователем.</p> <p>б) Расположить сотрудников в алфавитном порядке соответственно первой букве алфавита.</p>

	с) Чтоб можно было узнать возраст каждого сотрудника по году рождения.
31	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о пользователях телефонной станции и о дате последней оплаты услуг.</p> <p>а) Чтоб можно было найти пользователя по номеру телефона и по фамилии.</p> <p>б) Расположить запись в убывающем порядке по дате последней оплаты(день, месяц, год).</p>
32	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о телепередачах какого-то TV канала на протяжении одной недели.</p> <p>а) Вывести программу телепередач на день, указанный пользователем.</p> <p>б) Чтоб можно было узнать какая передача будет транслироваться в определённый день в указанное время и сколько времени ещё будет длиться эта передача.</p> <p>с) Определить передачи с минимальной и максимальной длительностью на каждый день.</p>
33	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о расписании приёма к какому-то врачу.</p> <p>а) Чтоб можно было узнать какой пациент будет на приёме в указанное время.</p> <p>б) Чтоб был возможен поиск пациентов по фамилии.</p> <p>с) Расположить записи в возрастающем порядке по времени приёма (часы и минуты).</p>
34	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о расположении книг на полках библиотеки: Книга-Отдел-Полка.</p> <p>а) Чтоб можно было узнать названия всех книг определённого отдела (например, Техника).</p> <p>б) Чтоб можно было выбрать любую книгу и узнать на какой полке она находится.</p> <p>с) Чтоб можно было найти книгу по автору.</p> <p>д) Расположить запись в алфавитном порядке по названию отдела, в котором находится.</p>
35	Составить базу данных с n количеством записей, которая бы содержала информацию о читателях библиотеки.

	<p>a) Найти любого читателя по фамилии и номеру читательского билета.</p> <p>b) Расположить записи в возрастающем порядке по дате записи в библиотеку (день, месяц, год).</p> <p>c) Чтоб была доступна информация о просроченных читательских билетах.</p>
36	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о пользователях информационной сети, доступной через пароль.</p> <p>a) Чтоб войти в систему, пользователь должен набрать имя и личный пароль.</p> <p>b) Расположить пользователей в алфавитном порядке по имени.</p> <p>c) Войдя в систему, пользователь должен получить информацию о своих правах в системе (чтение, редактирование, создание, удаление папок).</p> <p>d) Чтоб можно было найти пользователя по имени (без демонстрации пароля).</p>
37	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о студентах факультета.</p> <p>a) Чтоб можно было найти студента по кафедрам, специальностям и фамилии.</p> <p>b) Вычислить средний балл по результатам сессии для каждого студента и найти стипендию для любого студента по желанию пользователя.</p> <p>c) Расположить записи в убывающем порядке по среднему баллу.</p>
38	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о группе владельцев автомобилей.</p> <p>a) Чтоб был возможен поиск по регистрационному номеру автомашины.</p> <p>b) Расположить записи в алфавитном порядке по маркам автомобилей.</p> <p>c) Указать фамилии владельцев автомобилей самых популярных марок.</p>
39	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о расписании прибытия и отправления автобусов на автостанции.</p> <p>a) Чтоб был возможен поиск по наименованию маршрута и по времени отправления.</p> <p>b) Расположить записи по времени отправления (часы и</p>

	минуты) в возрастающем порядке.
40	<p>Составить базу данных с n количеством записей, которая бы содержала информацию о курсе лея в какой-либо валютной кассе.</p> <p>а) Чтоб можно было узнать курс покупки и продажи валюты, необходимой клиенту.</p> <p>б) Дать возможность клиенту продать или купить валюту в любом количестве.</p> <p>с) Расположить записи по стоимости валюты в убывающем порядке.</p>

2. Указания по содержанию курсовой работы.

Курсовая работа выполняется и оформляется соответственно установленным стандартам для выполнения документов soft.

Курсовая работа должна состоять из 9 частей:

1. Содержание.
2. Введение.
Включает описание цели курсовой работы, вкратце историю языка, в котором выполнена работа (1-2 стр.)
3. Задание.
Каждый студент получает задание на листе А4, подписанном заведующим кафедрой и руководителем работы (1стр.)
4. Логическая схема программы.
Блок-схема программы должна быть выполнена соответственно действующим стандартам. И состоять из блок-схемы главной программы и блок-схем других подпрограмм, использованных в программе, выполненных отдельно (объём зависит от алгоритма)
5. Листинг программы.
Собственно текст программы на языке, на котором выполнена работа (объём зависит от алгоритма)
6. Описание программы.
Содержит подробное описание вычислительных алгоритмов, отдельное описание главной программы и вспомогательных подпрограмм. Дополнительно может включать описание всех инструментов, типов данных, инструкций и стандартных функций языков, использованных в программе. (объём 10-15 стр.).
7. Получение результата.
Включает описание данных вывода и финальных данных, полученных в результате выполнения программы (объём 2-4 стр.).

8. Вывод.

Содержит краткий анализ программы, анализ начальных и финальных результатов (1-2стр.).

9. Список использованной литературы. (1-2 стр.)

Курсовая работа должна быть выполнена и защищена в сроки, указанные в задании, полученном каждым студентом.

Приложение 1. Массивы в языке С

Описание массивов

Массив – упорядоченная последовательность элементов одного и того же типа. Факт, что массив это единое целое, состоящее из нескольких элементов, позволяет нам назвать величину массива – сложной величиной.

Массив характеризуется именем, типом, размером. Общий формат описания массивов следующий:

tip name[d1][d1]...[dn]; где :

tip – это общий тип для всех элементов массива, то есть тип массива. Тип какого-либо массива может быть любым типом определённых данных: целым, вещественным, строковым и т.д.

name – это имя массива. В качестве имени массива используется любой идентификатор, более того, поскольку имя массива есть идентификатор, то на него распространяется всё, что указано в разделе “Название переменных (идентификаторов)”, *d1, d2, dn*-размеры массива. Размер массива может быть постоянным (константным) выражением с целым результатом. В зависимости от размеров, массивы разделяются на :

1. одномерные массивы (1 измерение);

одномерный массив – представляет собой ряд элементов, расположенных в одном ряду. Каждый элемент определённого массива имеет 1 координату: порядковый номер элемента в ряду.

2. двумерные массивы (2 измерения);

двумерный массив представляет собой структуру, состоящую из рядов и колонок. Каждый элемент двумерного массива имеет 2 координаты: номер ряда и номер столбца.

3. трёхмерные массивы (3 измерения);

трёхмерный массив представляет собой эквивалентную структуру с объёмом куба, имеющего 3 измерения: длину, ширину, высоту. Каждый элемент трёхмерного массива имеет 3 координаты: номер

ряда (в длину), номер колонки (в ширину) и номер высоты (в глубину).

4. многомерные массивы.

Пример описания массивов:

```
int vector[20];      vector – одномерный массив из 20 целых чисел;  
float x[10];        x – массив из 10 целых элементов;  
float matrix[7][9];  matrix – двумерный массив из 63 (7*9)  
переменных чисел;  
char fraza[25];     fraza – массив (предложение) из 25 знаков ;  
int spase[15][30][18]; spase – трёхмерный массив целых чисел  
(одномерный массив, состоящий из двумерных массивов);
```

Доступ к элементам массива.

При всём том, что массив это единое целое, мы не можем говорить о значении целого массива. Массивы содержат элементы, значениями которых оперируют в программе. Каждый элемент в массиве имеет свои показатели и своё значение. В качестве показателя одного элемента используется целое число, которое указывает порядковый номер элемента в массиве. Отсчёт элементов в массиве соответствует порядку исчисления, начиная от нуля. Это означает, что показатель одного элемента может иметь значение от нуля до $d-1$, где d – размер массива.

В качестве значения одного элемента из массива может служить любое число указанное в описании типа массива, то есть тип присваиваемого значения любому элементу из массива, должен соответствовать типу массива. Синтаксис доступа к любому элементу из массива следующий: *name*[*i1*][*i2*..*in*]. Где *name* это имя массива, *i1* – показатель элемента из размера 1, *i2* – показатель элемента из размера 2, *in* – показатель значения из размера *n*. Наиболее часто обрабатывают одномерные и двумерные массивы. Доступ к любому элементу из массива производят следующим образом: *name*[*i*]; где *name* – имя массива, *i* – порядковый номер элемента в массиве.

Пример:

vector[5]; доступ к элементу с порядковым номером 5 из массива *vector*.

fraza[20]; доступ к элементу с порядковым номером 20 из массива *fraza*.

Доступ к любому элементу из любого двумерного массива состоит в следующем *name*[*i*][*j*]; где *i* номер ряда, в котором находится элемент; *j* номер столбца, в котором находится элемент.

Например:

matrix[4][7]; указывает на элемент 4 ряда 7 столбца массива *matrix*.

y[0][0]; указывает на первый элемент в массиве, то есть 0 ряд, 0 столбец;

В случае, когда массив простого типа, присвоение значения элементу массива производится, как и в случае присвоения значения простой переменной:

x[0]=7.125;

vector[19]+=1;

matrix[1][1]=5.5;

fraza[3]='b';

space [3][5][2]=8;

В случае, когда массив структурного типа, присвоение значений и доступ к любому элементу производится соответственно правилам присвоения и доступа к структурированным переменным.

Элемент в массиве может появляться в любом выражении, где позволено присутствие переменной типа, одинакового с типом значения элемента.

Инициализация массивов

Часто необходимо, чтоб элементы массива имели значения, даже в момент описания массива.

Процесс присвоения значения элементам массива во время его описания называется инициализацией массива. Синтаксис инициализации любого одномерного массива следующий:

tip name[*d*]={*v*0,*v*1,*v*2,...,*v**n*-1}; где *tip* тип массива, *name* имя массива, *v*0,*v*1,*v*2,*v**n*-1 соответствующие значения элементов массива *name*[0],*name*[1] и т.д. Например:

int x[8]={1,3,15,7,19,11,13,5};

В этом случае элементы массива будут иметь следующие значения: *x*[0]=1; *x*[1]=3; *x*[2]=15; *x*[3]=7; *x*[4]=19; *x*[5]=11; *x*[6]=13; *x*[7]=5;

Необходимо отметить факт, что показатели массива меняются, начиная с нуля, то есть при описании массивов максимальное значение показателя массива совпадает с числом элементов в массиве минус один.

При инициализации массива необязательно указывать размеры массива, компилятор определит число элементов после описания массива и оформит массив соответствующей величины. Например:

int x[]={1,3,15,7,19,11,13,5};

Элементы массива примут значения, как и в предыдущем случае.

Проверим ещё несколько элементов инициализации массивов:

float vector[4]={1.2,34.57,81.9,100.77}; // *vector* – массив из 4 элементов типа *float*;

int digit[5]={1,2,3}; // *digit* – массив целого типа из 5 элементов, двум последним элементам присваивается значение 0.

char m[5]={'A','B','C','D'}; // *m* – массив из 5 знаков, последнему элементу присваивается значение 0-символ;

float const y[4]={25,26,17,18}; // инициализация массива *y*[4], элементы которого постоянны, тип *float* и не могут быть изменены в процессе выполнения программы.

Проверим инициализацию двумерного массива:

```
int a[3][3]={ {1,4,2},  
              {7,5,3},  
              {8,6,9} };
```

Инициализация двумерного массива производится по рядам. Элементы этого массива имеют следующие значения: $a[0][0]=1$; $a[0][1]=4$; $a[0][2]=2$; $a[1][0]=7$; $a[1][1]=5$; $a[1][2]=3$; $a[2][0]=8$; $a[2][1]=6$; $a[2][2]=9$;

При инициализации данного двумерного массива каждый ряд заключается в фигурную скобку. Если в указанных нами рядах не будет достаточно элементов для заполнения рядов, в этом случае на месте элементов, для которых не хватило значений, появятся нули. Если в этом случае опустить внутренние фигурные скобки, элементам массива придаются значения последовательного типа, взятые из списка. Заполнение массива производится по рядам. Элементы массива, для которых в списке не хватило значений, получают значение 0. Если в списке есть больше значений, чем элементов, тогда такой список считается неверным. Всё выше сказанное относится ко всем видам массивов.

Например, инициализация двумерного массива:

```
int a[3][3]={ 1,4,2,7,5,3,8,6,9};
```

Три эквивалентных метода инициализации трёхмерного массива:

```
int p[3][2][2]={ { {1,2},{3,4} }, { {5,6},{7,8} }, { {9,10},{11,12} }  
};
```

```
int p[3][2][2]={ {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

```
int p[3][2][2]={1,2,3,4,5,6,7,8,9,10,11,12};
```

Приложение 2. Структуры в языке C

До настоящего момента были изучены типы сложных данных, элементы которых принадлежат одному и тому же типу данных (простому или сложному). В случае некоторого массива, его элементы были одного и того же типа: целого, вещественного, символьного, без возможности присвоения любым элементам массива значений различного типа. Но часто бывают ситуации, когда необходима обработка и хранение более полной информации, таких как расписание занятий, успеваемость студентов и т.д. Если рассмотрим случай с успеваемостью студентов, тогда просто предположить, что будет необходима следующая информация: имя студента, группа, оценка, средний бал. Эти данные связаны между собой тем, что принадлежат одному и тому же лицу. Как следствие, было бы оправдано понимание их как одно сложное значение. Но эти типы данных отличаются между собой: имя и группа будут типа «строка», оценки по экзаменам целого типа, а средний балл вещественного типа (*float*). Группировка этих составляющих в одну сложную переменную – возможна посредством нового типа данных, названного в языке C *structura*.

Первый шаг в группировании составляющих разных типов в одну сложную переменную – это объявление и описание структуры. Объявив структуру, создается новый тип данных пользователя, о которых до настоящего времени не знал компилятор. Объявление структуры производится с объявления типов, перед началом главной функции *main()*.

Объявление некоторой структуры начинается с ключевого слова “*struct*”, после которого следует имя структуры, которое ещё называется типом регистрации. Элементы одной переменной зарегистрированного типа записываются после имени структуры в фигурных скобках. Синтаксис записи элементов структуры аналогичен синтаксису объявления переменных: указывается значение и тип элемента в структуре, разделённые знаком “;”. Описание структуры так же заканчивается символом “;” Синтаксис описания некоторой структуры в общем случае следующий: *struct name {tip_1 name_1; tip_2 name_2;.....; tip_n name_n;};*

Список элементов структуры называется шаблоном. Одна структура, вообще-то не объявляет ни одной переменной. Элементы одной структуры – это не отдельные переменные, они являются составляющими одной или нескольких переменных. Такие переменные называются структурированными и объявляются типом соответствующей структуры. Соответствующий шаблон описывает

эти составляющие, таким образом будет определён объём необходимой памяти необходимого для каждой структурной переменной регистрационного типа. Рассмотрим пример с успеваемостью какого-нибудь студента, объявление некоторой структуры будет следующим:

```
struct stud {char name [20] ; int ex1, ex2; float med; char grup [10];};
```

Объявление переменных типа структура

Объявление структур выделяет объём необходимой памяти. Перед использованием любого типа данных необходимо объявить соответствующую переменную. Невозможно использовать структуру в программе без объявления о переменной регистрационного типа, точно так же, как невозможно использование некоторого значения типа *float* перед описанием переменной типа *float*

Синтаксис объявления некоторой переменной – структуры следующий:

```
struct name_structura name_peremennaya;
```

Здесь ключевое слово *struct* указывает компилятору, что идёт речь о структуре, а тип регистрации *stud* определяет шаблон, по которому будет составлена переменная. За типом регистрации следует наименование переменной, которая будет использована в программе.

Например, для получения доступа к данным об успеваемости некоторого студента необходимо объявить о переменной: *struct stud a;* Теперь имеем переменную *a*, составленную из 5 полей, для которых была забронирована память.

Если в программе необходимо использовать несколько переменных одного и того же типа запись, возможно использование следующего синтаксиса:

```
struct stud a, b, c; Здесь объявлены 3 переменные типа запись stud: a, b, c. В случае если необходимо использование переменных типа запись, они объявляются отдельно.
```

Существует возможность объявления переменной типа запись одновременно с описанием структуры. Для этого значение переменной помещают между закрывающейся фигурной скобкой и символом “;” в конце объявления структуры:

```
struct stud { char name [20]; char grup [10]; int ex1, ex2; float med;}a;
```

Здесь *stud* – это тип запись и значение нового типа данных, который называется структура, ещё называются структурными полями, *a*-это

значение переменной, которая будет использована в программе и составлена согласно шаблону из 5 составляющих.

Инициализация переменных типа запись

В случае если исходные значения некоторой переменной – структуры известны, возможно присвоение их во время объявления переменной. В случае, если объявляется простая переменная типа запись, инициализация будет частью объявления структуры:

```
struct stud { char name [20]; char grup [10];  
int ex1, ex2; float med;}  
a={"Ivanov", "SOE-022", 8,7,7.5};
```

Здесь была описана структура *stud* и соответственно объявлена переменная *a* с инициализацией значений для своих составляющих.

Другой вариант инициализации составляющих структур это инициализация их в типе данных запись некоторой переменной типа запись. Например:

```
struct stud { char name [20]; char grup [10];  
int ex1, ex2; float med;}  
main () { struct stud a={"Ivanov", "SOE-022", 8,7,7.5}; }
```

Структура называется глобальной, если объявлена перед главной функцией *main()*, и называется локальной, если объявлена внутри функции *main()*, или внутри другой функции. Но если необходима инициализация некоторой структуры, которая содержит строки, её необходимо объявить перед функцией *main()* или как постоянную переменную: *static struct stud;*

Использование структур

Структура может быть обработана в программе, только если она объявлена переменной типа запись. Эта переменная состоит из нескольких элементов, каждый из которых имеет значение различного типа. Но обращение к значениям, которые находятся в структурных полях становится невозможным используя только имя соответствующего поля. Так же невозможен доступ к значениям типа запись, используя только её название. Соответственно правилам синтаксиса языка C++, для доступа к элементу структуры необходимо указать название переменной типа запись и название соответствующего поля, используя следующий синтаксис: *name_var.name_pole* , где *name_var* имя переменной типа запись *name_pole* имя поля соответствующей переменной.

Примечание: При обработке структурных полей, будут использованы все функции, указания и действия, относящиеся к типу, которому принадлежат структурные поля.

Пример: Составить структуру, которая содержала бы информацию о студенте (имя, группа) и вычисляла бы средний бал по результатам двух экзаменов:

```
struct stud {char name [20], group[10];
int ex1, ex2; float med;};
void main (void) { struct stud a;
puts(“Введите имя и группу”);
gets(a.name); gets(a.group);
puts(“Введите оценки по двум экзаменам”);
scanf(“%d%d”, &a.ex1, &a.ex2);
a.med=(a.ex1+a.ex2)/2;
printf(“media=%f”, a.med);}
```

Структуры чешуйчатые

Тип структура это сложный тип, это означает, что переменная этого типа может состоять из нескольких простых и сложных элементов, которые в свою очередь, могут содержать другие переменные. Этот факт даёт возможность использования некоторых структур в качестве полей для других структур. Такие структуры называются чешуйчатыми. Количество уровней чешуйчатых структур может быть бесконечным, но не рекомендуется использовать очень много уровней из-за неудобства синтаксиса.

В случае если структура А в своём составе имеет одно поле, которое в свою очередь является структурой В, тогда структура А объявляется только после того, как будет объявлена структура В. Следующий случай использует переменную *a* типа структура *stud* – чешуйчатая, которая содержит информацию о студенте и результаты сессии. Информация о результатах сессии собрана в отдельную структуру, называемой *sesia* и использована в качестве поля *otsenka* в структуре *stud*:

```
struct sesia {int ex1, ex2, ex3;
float med;};
struct stud {char name [20], group[10];
struct sesia otsenka;};
void main (void) { struct stud a;
puts(“Введите имя и группу”);
gets(a.name); gets(a.group);
puts(“Введите оценки 3 экзаменов”);}
```

```
scanf(“%d%d%d”, &a.otsenka.ex1, &a.otsenka.ex2,
&a.otsenka.ex3);
a.nota.med=( a.otsenka.ex1+a.otsenka.ex2+a.otsenka.ex3)/3;
printf(“\nmedia=%f”,a.otsenka.med);}
```

Здесь был вычислен средний бал по результатам 3 экзаменов. Этот пример демонстрирует синтаксис обращения к переменной из любого поля некоторой чешуйчатой структуры: используя название каждого поля, отделяемого точкой и записанного в убывающем порядке до поля назначения. Синтаксис *a.nota.med* демонстрирует обращение к полю *med*, которое принадлежит структуре (*sesia*), в которой в свою очередь есть поле (*otsenka*), входящее в состав другой структуры (*stud*) вызываемая при помощи переменной *a*.

Массивы структур

Во время объявления переменной типа запись, в памяти регистрируется объём для сохранения и обработки данных, которые будут содержаться в полях структуры соответственно шаблону на одну запись: один студент, одна личность и т.д. Чаще всего необходима обработка информации о группе лиц, в данном случае группы студентов. В этом случае необходимо объявление нескольких переменных типа запись, каждая из которых представлена конкретной записью на каждого студента. Для систематизации сохранённой информации во множестве записей – уместно объявление массива структур. В этом случае, каждый элемент одномерного массива типа структура сохранит вам информацию об одной личности и максимальном числе лиц, зарегистрированных таким образом, будет равно количеству элементов массива.

Одномерный массив структур можно объявить следующим образом:

struct name_structura nume_masiv [N]; где N – число элементов массива.

Пример: *struct stud x[10];* Таким образом, были забронированы 10 областей памяти, каждая из которых имеет объём, необходимый для сохранения целой структуры для обработки информации об успеваемости студента.

Обращение в программе к элементу массива структур будет выполнено, как и в случае простого массива: будет указано название массива с порядковым номером элемента в квадратных скобках. Например: *x[3]*. Но такое обращение к структуре будет неверным. Обработка структуры производится посредством обработки отдельных полей из неё. Доступ к полю из переменной типа запись,

которая в то же время принадлежит массиву будет возможен используя синтаксис: *name_masiv[k].Name_pole*, где *k* – порядковое число необходимой записи.

Пример: Для записи в поле “*name*” из структуры “*stud*” имя студента с порядковым номером 2, используем следующий синтаксис:

```
struct stud x[10]; gets(x[2].name).
```

Примечание: Необходимо помнить, что отсчёт элементов массива начинается с индекса 0. В данном случае составление массива из 10 структур с порядковым номером от 0 до 9.

Пример: Дана база данных с *n* количеством записей, которая содержит информацию об успеваемости группы студентов. По оценкам, полученным во время экзаменов любой сессии, вычислить средний балл для каждого студента. Необходимые поля: Имя, Группа, Оценки на экзаменах, Средний балл:

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
struct stud {char name[20], grupa[10];
int ex1,ex2; float med;};
void main (void){ clrscr();
struct stud x[50]; int i,N;
printf("Введите количество студентов\n");
scanf("%d",&N);
printf("Введите информацию о %d студентах:\n",N);
for (i=0; i<N; i++){
printf("Введи имя студента %d\n",i);scanf("%s",x[i].name);
printf("Введи группу студента %d\n",i); scanf("%s",x[i].grupa);
printf("Введите 2 оценки студентов %d\n",i);
scanf ("%d%d",&x[i].ex1,&x[i].ex2);
x[i].med=(x[i].ex1+x[i].ex2)/2;}
printf("\nФинальный массив:\n");
printf("*****\n");
printf("** Имя ** Группа ** Оценка1 ** Оценка2 ** Средняя
**\n");
printf("*****\n");
for (i=0;i<N;i++) {
printf("**%10s**%9s**%7d**%7d**%9.2f**\n",
```

```

x[i].name,x[i].grupa,x[i].ex1,x[i].ex2,x[i].med);
printf("*****\n");} getch(); }

```

Приложение 3. Папки в языке C

Во время выполнения программ, в большинстве случаев, существует необходимость вывода результатов. Но возможности простого вывода данных на монитор очень малы. Даже использование временной остановки выполнения программы, с целью дать возможность пользователю прочитать всю выведенную информацию, не решает до конца нашу задачу. Если выведенное сообщение исчезает с монитора, повторное выведение на монитор будет возможным только при повторном запуске программы в целом. Более того, значения переменных из программы сохраняются только на протяжении выполнения программы. Одновременно с окончанием выполнения программы, вся введённая информация теряется

Для сохранения введенной информации, с целью дальнейшего использования необходима её запись на диск в форме специальных структур, которые называются файлами. Использование файлов позволяет сохранить любую информацию на длительное время, перенесение данных с одного носителя информации на другой, чтение и выведение данных в случае надобности. Вся информация, касающаяся работы с файлами в языке C/C++ сохраняется в библиотеке `stdio.h`. Поэтому перед началом обработки папок в C/C++, необходимо включить эту библиотеку в программу с помощью директивы `#include<stdio.h>`, которая даёт возможность записи информации в файл и её дальнейшее чтение. Во время ввода данных в программу, в файл с диска, имеет место копирование их в оперативную память компьютера, а информация из файла, которая находится на жёстком диске остаётся неизменной на протяжении выполнения программы. При записи данных из программы на диск, в файл записываются данные, которые хранились до этого момента в оперативной памяти.

Запись или чтение информации из файла выполняется с помощью указателя на файл. Во время записи или чтения информации из файла, компилятор использует посреднический уровень записи между программой и жёстким диском, где сохраняется файл. Этот уровень представлен зоной памяти, названной «зона тампон», которая имеет место для временного сохранения информации с целью её записи, а потом чтения её из папки.

Для отправки или чтения информации из «зоны тампон», компилятор пользуется специальной структурой, названной «структура - файл». В этой структуре сохранена информация, необходимая компьютеру для осуществления записи/чтения данных из файла, включительно адрес зоны памяти, куда помещен файл. Синтаксис объявления какого-либо указателя файла следующий:

```
FILE *file_pointer;
```

Где ключевое слово *FILE* указывает компилятору, что объявленная переменная это указатель файла, а *file_pointer* это имя указателя. В случае если программа предполагает работу с несколькими файлами, необходимо объявление нескольких указателей файлов в последовательности:

```
FILE *f1, *f2, *f3;
```

где *f1, f2, f3* названия указателей.

Обработка данных в языке C/C++ предполагает выполнение следующих этапов:

- 1) Открытие файла (для того, чтоб была возможна запись или чтение информации из файла, её надо открыть.)
- 2) Обработка файла (действия чтения/записи).
- 3) Закрытие файла. Для того, чтобы записанная информация была сохранена в файле, её надо закрыть.

Открытие файлов

Открытие какого-либо файла производится с помощью функции *fopen()*, которая имеет следующий синтаксис:

```
pointer=fopen("name_f", "mod");
```

где *pointer* это название указателя файла, *name_f* это название реального файла с жёсткого диска, *mod* это метод доступа к файлу.

Результат записи этой функции это присвоение адреса структуры типа

файл указателю файла. Первый параметр это название файла, которое обычно имеет следующую структуру: *name.ext*, где *ext* это протяжение файла, состоящее из 3 знаков. В качестве второго параметра, функция принимает метод доступа к файлу, то есть информацию о действиях, выполненных с файлом.

Существуют 3 главных метода открытия файлов:

- 1) Метод "w" позволяет открыть файл с целью записи в неё информации или передачи информации принтеру, а затем её распечатки. Если указанный файл не существует, его можно заново создать. Если файл уже существует, вся существующая в нём информация будет уничтожена.

- 2) Метод “*r*” указывает компилятору, что файл будет открыт с целью чтения с него информации. Если файл не существует, то в момент его открытия появится ошибка выполнения.
- 3) Метод “*a*” позволяет открыть файл с целью дополнения, то есть записи информации в его конце. В случае если файл не существует, его надо создать. Если указанный файл существует, тогда записанная информация будет помещена в конце файла без уничтожения уже существующей в файле информации.

Например, для создания нового файла с названием *info.txt* будет использован следующий синтаксис:

```
FILE *file;  
file=fopen(“info.txt”, “w”);
```

В случае если необходимо прочесть некоторые данные, нужно пользоваться методом доступа “*r*”:

```
FILE *file;  
file=fopen(“info.txt”, “r”);
```

Для отпечатывания информации из файла на бумагу с помощью печатающего устройства, название файла будет *PRN*, а метод доступа “*w*”:

```
FILE *file;  
file=fopen(“PRN”, “w”);
```

Надо отметить, что и название файла и выбранный метод доступа к нему, выделены двойными кавычками. Это обусловлено фактом, что параметры в функции *fopen()* передаются в качестве ряда знаков. Используя эти возможности можно набрать на клавиатуре название файла, которое желает пользователь:

```
char name [12];  
FILE *f;  
printf(“Выведите название файла\n”);  
gets(name);  
f=fopen(name, “w”);
```

Во время работы с файлами в C/C++ пользуются специальными указателями, в котором сохраняется информация о настоящем положении чтения из файла. Во время чтения данных из файла, указатель определяет следующую порцию данных, которые необходимо прочесть с диска.

Если файл открыт впервые режимом доступа *r*, указатель помещается на первом символе файла. Во время выполнения следующих действий чтения, указатель помещается на начало следующей порции данных. Величина шага чтения из файла зависит от величины считываемой из файла информации. Если за один шаг

считывают только 1 символ, тогда указатель помещают на следующий символ, если читается структура, указатель помещают на следующую структуру. Тогда, когда вся информация из файла прочтена, указатель попадает на специальный код, названный концом папки: (*eof*). Попытка чтения из файла после окончания файла ведёт к ошибке.

Если файл открыт в режиме доступа “*w*”, указатель также помещается на начало файла, таким образом первые данные, записанные в файл, будут помещены в начале. Во время закрытия файла, в её конце будет записан символ окончания паки (*eof*). Если во время открытия в режиме ввода “*w*” файл уже существует, вся существующая информация в ней будет удалена, и поверх неё будет записана новая информация. Любые предшествующие данные, которые могут остаться не удаленными в файле – будут помещены после кода окончания файла (*eof*), и доступ к ним будет закрыт. Таким образом вся информация из файла, открытого в режиме доступа “*w*” будет удалена, даже в случае если файл будет открыт без записи в него каких-либо данных.

Если файл открывается в режиме доступа “*a*”, указатель помещается на символ конца файла (*eof*). Любая информация, записанная таким образом в файл, будет помещена после уже существующих данных, а после записи в конце файла добавляется код конца файла (*eof*).

В некоторых случаях бывает такая ситуация, что оперативная система не может открыть файл, указанный в функции *fopen()*. Это может быть обусловлено отсутствием места на жёстком диске, или фактом, что указанный файл просто-напросто не существует. Возможна так же и такая ситуация, когда необходима распечатка данных на принтере, а он не включён, или отсутствует бумага. В случае попытки пользования файлом, который не открывается, программа будет закрыта в результате ошибки в работе. Для предотвращения аварийной остановки программы, можно проверить состояние файла с помощью указания *if*, которое остановит программу в случае ошибки. Здесь использована особенность оперативной системы, которая возвращает значение *NULL* в случае появления ошибки и, когда файл не может быть открыт. В этом случае код *NULL* возвращается на место адреса структуры – файл и программа останавливаются. Условие для предотвращения аварийного выхода из программы, будет иметь вид:

```
if ( (file=fopen("info.txt", "w"))==NULL )  
puts ("Файл не может быть открыт");
```

exit();

После того как вся информация записана в файл или прочитана из него, необходимо закрыть файл, то есть прервать связь между файлом и программой. Закрытие файла осуществляется с помощью функции *fclose()*, которая имеет следующий синтаксис:

fclose(f_pointer); где *f_pointer* это название указателя в файле.

Одновременно с закрытием файла, мы получаем гарантию того, что вся информация из «зоны тампон» действительно записана в файле. Если программа заканчивается до закрытия файла, возможна ситуация, когда часть информации, которая не была записана на диске, остаётся в «зоне тампон», и в результате чего – теряется. Кроме этого, если файл не открывается правильно, в конце него не будет записан в обязательном порядке код конца файла (*eof*) и следующее открытие файла будет сопровождаться ошибкой.

Более того, закрытие файла освобождает указатель, после чего его можно использовать для доступа к другому файлу, или для выполнения других действий с файлом. Например, необходимо создать файл, записать в нём информацию, затем прочесть её из файла. Отметим факт, что после записи данных, файл надо закрыть и только после этого открыть его для чтения, таким образом, имея соответствующий доступ к данным из файла:

```
#include<stdio.h>
void main (void) {
FILE *file;
if ((file=fopen("info.txt", "w"))==NULL) {
puts ("Файл не может быть открыт\n");
exit(); }
//Здесь будут помещены указания для записи информации в
файл.
fclose (file);
if ((file=fopen("info.txt", "r"))==NULL) {
puts ("Файл не может быть открыт\n");
exit(); }
// Здесь будут помещены указания по чтению информации из
файла.
fclose (file); }
```

Здесь файл открывается внутри условия *if*, то есть во время открытия проверяется значение, возвращенное оперативной системой к открытием файла и если возвращено значение *NULL* создаётся соответствующее сообщение и программа останавливается.

Отметим факт, что некоторые компиляторы позволяют запись данных в файл путём очистки «зоны тампон» с помощью функции *flush()*. Эта функция позволяет без того, чтоб закрыть файл, записать всю информацию из «зоны тампон» в файл, затем эта зона освобождается от некоторых данных.

Функции записи/чтения из файла

Язык C/C++ содержит большие возможности передачи данных в файл и чтения из файла в зависимости от использованной функции:

- Функции *fputc()* и *putc()* используются в случае записи некоторого знака в файл или распечатке на принтере.
- Функции *fgetc()* и *getc()* используются для чтения некоторых знаков из файла.
- Для записи ряда знаков в файл или распечатки его на принтере, используют функцию *fputs()*.
- Для чтения ряда знаков из файла пользуются функцией *fgets()*.
- Функция *fprintf()* используется в случае вывода форматированных знаков, рядов знаков и цифр на диск или принтер.
- Функция *fscanf()* используется для чтения форматированных знаков или цифр из файла.
- Запись некоторой структуры в файл возможна, используя функцию *fwrite()*.
- Чтение некоторой структуры из файла осуществляется с помощью функции *fread()*.

Запись/чтение знаков

Запись/чтение знаков из файла это основная форма работы с папками. Хотя эти действия и не пользуются большой популярностью, они хорошо отображают основные принципы работы с файлами.

Любой знак может быть записан в файл, как уже было сказано при помощи функции *fputc()*, используя следующий синтаксис:

fputc(v,fp); где *v* это переменная символьного типа (*char*), и *fp* – это имя указателя файла.

Следующий пример делает возможной запись в файл целого набора символов, до тех пор, пока не будет нажата клавиша Enter:

```
#include <stdio.h>
#include <conio.h>
void main (void) {
    FILE *f;
    char lit; clrscr();
    f=fopen("info.txt","w");
```

```

printf (“введите несколько символов \n”);
do {
lit=getch();
putch (lit);
fputc(lit,f);} while(lit!='\ r’);
fclose (f); }

```

Здесь файл открыт в режиме доступа w. И если в момент открытия файл с названием “into.txt” не существовал, он будет создан. В цикле “do” имеет место последовательное чтение знаков с клавиатуры с помощью функции *getch()* и запись их в файл с помощью функции *fputc()*. Необходимо отметить факт, что с таким же успехом здесь можно было бы использовать функцию *putch()* с теми же параметрами. Цикл “do” будет продолжаться до тех пор, пока не будет нажата клавиша *ENTER* и только после этого файл закроется.

Для чтения знаков из файла пользуются функцией *getc()* и *fgetc()* со следующим синтаксисом:

ch_var=getc(fp); где *ch_var* это переменная символьного типа, а *fp*—указатель на файл.

Следующий пример демонстрирует, как может быть прочитана информация из файла, созданного по предыдущему примеру:

```

#include <stdio.h>
#include <conio.h>
void main (void) {
FILE *f1; char lit;
clrscr();
f1=fopen(“info.txt”,”r”);
printf(“прочитанная информация:\n”);
while( (lit=fgetc(f1))!=EOF)
printf (“%c”,lit);
fclose(f1); getch(); }

```

Файл открыт типом “r”, следовательно, возможно чтение информации из него. Цикл *while*, в котором имеет место последовательное чтение знаков из файла – будет выполняться, пока не будет определён символ конца файла *EOF*, который записывается в конец каждого файла в момент его закрытия.

Функция чтения *fgetc(f1)* использует другой указатель для того же файла—*f1*. Этот факт обусловлен тем, что даже если для действий записи и чтения из файла используется тоже название файла, запись или чтение желательно выполнять с помощью различных указателей.

Запись/чтение строк

В случае, когда необходимо записать в файл набор символов, то есть строк – рекомендуется использовать функцию *fputs()*; которая имеет следующий синтаксис:

fputs (s_var, fp); где *s_var* это переменная строки символов, а *fp* – индикатор на файл.

Функция *fputs()* осуществляет запись строки в файл или запись их на бумаге без, ввода знака *конец линии*. Для каждой строки, записанной таким образом в файл, для начала с нового ряда необходимо вручную вводить символ *конца строки*.

Следующий пример делает возможной запись в файл целого набора фамилий:

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main (void) {
FILE *k; char fam[15];
clrscr();
printf (“введите фамилию\n”);
gets (fam);
k=fopen(“familia.txt”, ”w”);
while(strlen(fam)>0){
fputs(fam,k); fputs(“\n”,k);
printf(“введите следующую фамилию\n”);
gets (fam);}
fclose (k);}
```

Здесь цикл *while* будет повторяться до тех пор, пока не будет введён ряд длины 0. Функция *fputs()* записывает в файл ряды каждый с нового ряда, благодаря вводу *fputs(“\n”,k)*; В конце своей обработки, файл *familia.txt* обязательно будет закрыт *fclose(k)*; Необходимо отметить, что для распечатки на бумагу набранных фамилий на принтере, необходимо указать название файла “*prn*” следующим образом: *k=fopen (“prn”, ”w”)*; Для правильной распечатки на принтере необходимо использовать строки длиной в 81 символ, с целью того, чтоб ряд поместился полностью по ширине монитора, перед тем, как нажмём клавишу ENTER.

Чтение строк знаков из файла осуществляется посредством функции *fgets()*, которая имеет следующий синтаксис:

fgets(s_var, l, fp); где *s_var* это переменная типа строка, *l* – это переменная или постоянная целого типа, которая указывает на

максимальное возможное количество символов в ряду, *fp* – указатель на файл.

Следующий пример делает возможным чтение фамилий из файла “familia.txt”, созданного по предыдущему примеру:

```
# include <stdio.h>
# include <conio.h>
void main (void) {
FILE *r; char name[15];
clrscr();
r=fopen(“familia.txt”,”r”);
printf(“Информация прочитанная из файла:/n”);
while ( fgets(name, 15, r)!=NULL )
printf(“%s”name);
fclose(r); getch(); }
```

Здесь цикл *while* будет повторяться, пока не будет определён код «конец файла». В случае чтения информации из файла на уровне строк, для указания конца файла, пользуются кодом *NULL*, а код *EOF* используется для чтения знаков. Функция *fgets()* прочитает строку до кода «новая линия», если его длина не превышает значение “*l-1*“, указанное в параметрах функции.

Отметьте факт того, что функция *printf(“%s”,name);* не использует код “/n” для передвижения по новому ряду, потому что каждый ряд, прочитанный из файла, уже содержит код “\n” – новая строка, записанная в файл в предыдущем примере с помощью функции *fputs(“\n”,k)*.

Форматированный ввод/вывод

Функция для обработки символов и строк предназначена для записи/чтения из файла только текстовой информации. В случае если необходимо записать в файл данные, которые содержат числовые значения, пользуются функцией *fprintf()* со следующим синтаксисом:

fprintf(fp, format, data); где *fp* указатель на файл, в который записывается, *format* это контрольный ряд формата, записанных данных и *data* это список переменных или значений, которые надо записать в файл. Пример:

```
fprintf(f, “%d”, cost);
```

Следующий пример демонстрирует метод записи информации в файл о наборе продуктов. В файле будут записаны названия, стоимость и количество продуктов, хранящихся на складе:

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

```

#include<string.h>
void main (void) {
FILE *f; clrscr();
char name[20], otvet=' y' ;
float tsena; int unit;
f=fopen( "product.txt", "w");
while (otvet==' y' ){
printf( "Введите название продукта\n");
scanf( "%s", name);
printf( "Введите стоимость продукта %s\n", name);
scanf( "%f", &tsena);
printf( "Введите количество продукта %s\n", name);
scanf( "%d", &unit);
fprintf( f, "%s %f %d\n", name, tsena, unit);
printf( "Желаете продолжить? y/n\n");
otvet=getch();}
fclose(f);}

```

В результате выполнения этой программы в файл будет записана информация о некоторых продуктах, например:

```

discheta 4.500000 100
Mouse 140.000000 3
Monitor 2000 1

```

Обратите внимание, что символ “\n”- конец строки в конце контрольного форматированного ряда функция *fprintf()*, благодаря ему, информация о каждом продукте записывается с нового ряда файла.

Форматированное чтение информации из файла осуществляется при помощи функции *fscanf()*, которая имеет те же ограничения, что и функция *scanf()*, и использует следующий синтаксис: *fscanf(fp, format, data);* который идентичен синтаксисам функции *fprintf()* при описании параметров.

В следующем примере осуществляется чтение информации из файла “*produs.txt*” о продуктах со склада, записанных в файл в предыдущем примере:

```

#include<stdio.h>
#include<conio.h>
void main (void) {
clrscr();
FILE *a; char name[20];
float tsena; int unit;
a=fopen( "produs.txt", "r");

```

```

while( fscanf( a, "%s%f%d", name, &tsena, &unit)!=EOF) {
printf( "Название: %s\n", name);
printf( "цена:      %f\n", tsena);
printf( "количество: %d\n", unit); }
fclose(a); getch(); }

```

Здесь чтение данных из файла происходит одновременно с проверкой условий продолжения цикла *while*. Цикл *while* будет выполняться, пока не будет определён код конца файла – в данном случае *EOF*.

Данные из файла прочитаны переменными *name*, *tsena* и *unit*, а затем выведены на монитор.

Файлы и структуры

Для записи переменной типа структура (запись) в файле используют функцию *fwrite()*, которая имеет следующий синтаксис:

fwrite (&struct_var, struct_size, n, fp); где:

- *&struct_var* это название переменной типа структура с адресным оператором, который сообщает компилятору адрес стартовой ячейки в памяти, куда помещена структура.
- *struct_size* это величина структуры. Для определения величины структуры используется функция *sizeof(s)*; где *s* это название переменной типа структура.
- *n* это целое число, которое определяет количество структур, которые будут записаны в файл с первой попытки. Здесь же рекомендуется использовать одно значение. Значения больше, чем 1 используют в случае записи в файл массива структур с первой попытки.
- *fp* это название указателя файла.

Пример: *fwrite(&a,size(a),1,f);*

В следующем примере создан массив структур с информацией о группе студентов, который содержит имя студента, год рождения и средний бал. Сначала массив структур заполняется информацией, а затем по одной, с помощью цикла *for* и функции *fwrite()* записываются в файл. Название файла даётся пользователем и сохраняется в переменной "*filename*" типа строка символов:

```

#include<stdio.h>
#include<conio.h>
struct stud {
char nume [15];
int an; float med;};
void main (void) { clrscr();

```

```

struct stud x[10]; int i,n;
FILE *f; char filename[12];
float m;
printf("Введите количество студентов \n");
scanf("%d" &n);
for (i=0; i<n; i++) {
printf("Введите имя студента\n");
scanf("%s", x[i].name);
printf("Введите год рождения \n");
scanf("%d", x[i].god);
printf("Введите средний бал\n");
scanf("%f, &m); x[i].med=m; }
printf("введите название файла \n");
scanf("%s",filename);
f=fopen(filename,"w");
for(i=0; i<n; i++) fwrite (&x[i], sizeof (x[i]), 1, f);
fclose(f); getch(); }

```

В результате выполнения этой программы будет создан файл с названием, которое было дано переменной *filename*, где были записаны структуры с информацией о студентах. Если откроем файл с помощью редактора обычного текста, заметим в нём непонятный смысл. В действительности информация (точнее сказать структуры), которая находится в этом файле непонятна компилятору и может быть прочитана с помощью функции *fread()*, которая имеет тот же синтаксис, что и функция *fwrite()*, при описании параметров:

```
fread (&struct_var, struct_size, n, fp);
```

В следующем примере осуществляется чтение всех записей (структур) из созданного файла в предыдущем случае и вывод данных на монитор. Необходимо отметить такой момент, что функция *fread()*, в результате своего выполнения возвращает значение, которое соответствует количеству структур, с успехом прочитанных из файла. В данном случае имеет место чтение структур по одной из файла, если функция в случае успеха вернёт значение 1. В случае создания или определения конца папки, функция вернёт значение 0.

```

#include<stdio.h>
#include<conio.h>
struct stud {
char name[15];
int god; float med;};
void main (void) {clrscr();

```

```

struct stud y [10];
FILE *k; char fn[12]; int i=0;
printf (“введите название файла \n”);
scanf (“%s”, fn);
k=fopen (fn, ”r”);
printf (“Информация прочитанная из файла:\n”);
while ( fread (&y[i], sizeof(y[i]), 1, k)==1) {
printf (“Имя студента: %s\n”,y[i].name);
printf (“Год рождения: %d\n”,y[i].god);
printf (“Средний бал: %f \n”,y[i].med); i++;}
fclose(k); getch(); }

```

Следующая таблица содержит описание всех возможностей данных относительно файла, включительно значения возвращённые функцией в случае ошибочного чтения:

Тип данных	Функции вывода	Функции ввода	Значение, возвращенное при чтении
Символы	<i>putc(); fputc();</i>	<i>getc(); fgetc();</i>	<i>EOF</i>
Строка символов	<i>fputs();</i>	<i>fgets();</i>	<i>NULL</i>
Форматные данные	<i>fprintf();</i>	<i>fscanf();</i>	<i>EOF</i>
Структуры	<i>fwrite();</i>	<i>fread();</i>	<i>0</i>

Список рекомендуемой литературы

Данные методические указания можете найти в Internet по адресу www.istrati.com

Nr.	Название работы (книга, методический материал, указатель)	Автор	Год издания
1.	Pascal și Turbo Pascal	Bălănescu T. ș.a	1992
2.	Inițiere în Turbo Pascal	Kalisz E.	1997
3.	Pascal pe interesul tuturor	Anghel,Florin Stînga	1992
4.	Turbo Pascal 6.0 Ghid de utilizare	Sandur Covax	1993
5.	Turbo Pascal 6.0 Programe	Lucian Vasiu ș.a.	1994
6.	Îndrumare metodică de laborator N510	FCIM	1996
7.	Calculatoare personale	Gremalschi A.	1997
8.	Structura calculatoarelor numerice	Gremalschi A.	1996
9.	Limbajele C și C++ pentru începători	Negrescu L.	1996
10.	Turbo C++	Cojocaru C, O.	1994
11.	Введение в язык Паскаль	Абрамов В.Г.	1988
12.	Введение в программирование на языке Паскаль	Эрбс, Хайнц-Эрих, Штольц, Отто.	1989
13.	Вычислительная техника и программирование	Алексеев В.Е.	1991
14.	Язык программирования Си	Керниган Б. Ритчи Д.	1992
15.	Программирование на языке Си для персональных компьютеров	Григорьев А.	1990
16.	Программирование на языке Си, справочное пособие	Котлинская Г.П.	1991