

## 1. ВВЕДЕНИЕ

Лабораторные работы выполняются на ПК во время занятий в классах ЦИТСО МГТУ ГА в присутствии преподавателя.

Для разработки программ используется система программирования Borland C++3.11.

Выполнение лабораторной работы состоит из трех этапов:

1) *подготовка к выполнению* – студент получает от преподавателя вариант задания на выполнение лабораторной работы, самостоятельно разрабатывает схему алгоритма задачи, получает допуск у преподавателя к выполнению работы;

2) *выполнение* - студент разрабатывает программы с помощью текстового редактора интегрированной среды, проводит отладку и тестирование программы, представляет преподавателю результаты;

3) *защита работы* – производится при наличии оформленного отчета по лабораторной работе, студент объясняет алгоритм задания и программу, отвечает на вопросы по всем видам работы на ПК во время отладки программы, а также на ряд контрольных теоретических вопросов.

Защита лабораторной работы фиксируется записью в журнале лабораторных работ и подписью преподавателя в отчете студента. В соответствии с положениями МГТУ ГА студент, не защитивший предыдущую лабораторную работу, не допускается к выполнению следующей.

### Оформление отчета

Отчет выполняется по каждой лабораторной работе в тетради для лабораторных работ.

Отчет должен содержать наименование лабораторной работы и ряд разделов:

- 1) цель лабораторной работы;
- 2) техническое задание и состав исходных данных для тестирования программы;
- 3) структуру программы, алгоритмы каждой функции программы (включая главную), кроме простейших;
- 4) таблицы глобальных переменных и локальных по каждой функции;
- 5) распечатку текстов программы и файлов с исходными данными;
- 6) распечатку протоколов тестирования программы.

Все распечатки должны быть аккуратно подклеены в соответствующий раздел отчета.

## 2. ЛАБОРАТОРНАЯ РАБОТА № 1

### Вычисление выражений с использованием алгоритмов линейной структуры

#### 2.1. Цель лабораторной работы

Целью лабораторной работы является, во-первых, освоение построения алгоритмов линейной структуры для вычисления выражений и, во-вторых, освоение интерфейса системы Borland C++ 3.11:

- структуры и состава главного меню системы;
- приемов работы с каждым пунктом меню;
- команд текстового редактора;
- запуска программы на трансляцию, выполнение;
- смены окон редактора, просмотра значений параметров программы и экрана результатов.

#### 2.2. Теоретические сведения

В основе решения любой задачи лежит понятие алгоритма.

*Алгоритм* – это конечная последовательность точно определенных элементарных действий для решения поставленной задачи при всех допустимых вариантах исходных условий задачи.

Основные свойства алгоритма:

- *конечность* – алгоритм всегда должен приводить к результату после конечного числа шагов;
- *определенность* – каждый шаг алгоритма должен быть точно и недвусмысленно определен;
- *эффективность* – все операции алгоритма должны быть достаточно простыми для их реализации;
- *массовость* – алгоритм должен приводить к правильному результату во всем диапазоне исходных условий задачи.

Решение задач на ЭВМ – это процесс обработки данных, ведущий от исходных данных к конечному результату.

Разработка алгоритма для ЭВМ включает в себя выделение этапов процесса обработки данных и представление их в определенной форме и последовательности, например, в виде схемы алгоритма. В схеме алгоритма этапы обработки представляются в виде структур алгоритма – графических элементов, соединенных линиями передачи управления. Каждому действию соответствует некоторая геометрическая фигура. Конфигурация графических элементов определена государственным стандартом (ГОСТ 19.701- 90) Единой системы программной документации (ЕСПД).

Разработка алгоритма – один из самых трудных этапов решения задачи. В алгоритмах линейной структуры этапы обработки данных (и соответственно графические элементы в схеме алгоритма) располагаются строго последовательно.

Разработанный алгоритм реализуется в виде программы для ЭВМ на одном из языков программирования.

Конструкции языка, в которых определены действия программы, называются операторами. Каждый исполняемый оператор определяет действия программы на очередном шаге ее исполнения. По характеру действий различают два типа операторов: операторы преобразования данных и операторы организации обработки данных.

Операторы присваивания, ввода и вывода данных, операторы вызова функций являются типичными операторами преобразования данных, и именно эти операторы используются при программировании линейных процессов вычислений.

### Структура программы на C++

Программа на C++ состоит из директив препроцессора, объявления глобальных объектов программы (переменных, констант, типов), определения одной главной функции (**main**), ряда неглавных функций и большого числа комментариев для пояснения текста программы.

Рассмотрим составные части программы:

1. Директивы препроцессора предназначены для организации обработки текста программы до ее компиляции. Каждая директива начинается с символа **#** и может располагаться в любом месте программы, однако рекомендуется располагать директивы в начале программы.

Форма использования директив **#include** и **#define** в программе:

а) директива **#include** используется для включения текстов из файлов:

**#include <имя файла>**           // - файл из стандартных библиотек

**#include "имя файла"**           // - файл пользователя

б) директива **#define** используется для замещений в тексте:

**#define имя значение**           // определение константы.

**#define имя(параметры) строка\_замещения** // определение макроса

2. Объявления глобальных (внешних) объектов программы (переменных, констант, типов) могут располагаться в любом месте программы вне функций. Форма объявления переменных и констант:

**тип\_данных имя\_переменной;**

**const тип\_данных имя константы = значение;**

3. Главная функция является обязательным элементом любой программы. Именно с нее начинается и в ней заканчивается выполнение программы. Определение главной функции:

**void main ()**                   // заголовок функции

{ телом функции является блок – последовательность объявлений локальных (внутренних) объектов функции **main** и исполняемых операторов функции, заключенная в фигурные скобки; }

4. Как правило, процесс разработки программы сводится к разбиению задачи на ряд фрагментов, выполняющих некоторую законченную обработку данных, которые оформляются в виде функций:

**тип\_возвращаемого\_результата имя\_функции (список параметров)**

{тело функции – блок, последовательность описаний, определений и операторов};

5. Комментарии – это последовательность символов, заключенная в скобки `/* ... */`, или строка символов, начинающаяся символами `/*` и заканчивающаяся символом перехода на новую строку. Комментарии воспринимаются компилятором как пробелы и используются для пояснения текста программы.

### Оператор присваивания

Часто решение поставленной задачи представляет собой процесс формирования результатов из исходных данных.

В программах данные фигурируют в виде программных объектов.

Данные, которые не изменяются в процессе выполнения программы, называются *константами*.

Данные, определенные в программе и изменяемые в процессе ее выполнения, называются *переменными*.

Правила формирования новых значений в программе задаются с помощью *выражений*.

С помощью *оператора присваивания* переменные получают новые значения. Структура оператора присваивания:

**L-значение = выражение;**

L-значением называют любую возможную форму обращения к именованной области оперативной памяти (ОП), значение которой доступно изменениям. Имя переменной – частный случай L-значения:

**имя\_переменной = выражение;**

Оператор присваивания выполняет следующее действие: “выражение” правой части *вычисляется* и его значение (в двоичной форме) помещается в представленную L-значением область памяти.

Примеры использования операторов присваивания:

**float x , y , z;**

**int i;**

**cin>>x>>y;**      // ввод значений с клавиатуры

**i= 0; z=x+y;      i= i+2;**      //операторы присваивания.

### Выражения

*Выражение* – это правило получения нового значения. Выражения формируются из операндов, операций и круглых скобок:

*операнды* – это константы, переменные и результаты функций;

*операции* – действия над операндами; выполняются в соответствии с приоритетом, в C++ операции разбиты на 18 рангов, операции одного ранга выполняются слева направо или справа налево в соответствии с ассоциативностью данной операции;

круглые скобки, как и в математических выражениях, задают порядок выполнения операций.

В зависимости от типов операндов и операций, выражения условно делят на *арифметические, логические и выражения над символами и строками*.

**Арифметическое выражение** аналогично алгебраическому выражению математики.

Операнды арифметических выражений: константы, переменные и результаты функций арифметических типов.

Операции:  $+$  ,  $-$  ,  $*$  ,  $/$  ,  $\%$  ,  $=$  , **ор** = ,  $++$  ,  $--$ .

Рассмотрим более подробно *операцию присваивания*.

Символ '=' в языке C++ означает бинарную операцию, у которой должно быть два операнда: левый – переменная и правый – выражение.

**имя\_переменной = выражение**

Например,  $x = 2.5 + 5.2$  есть выражение, имеющее значение 7.7. В тех конструкциях языка, в которых используется значение этого выражения, одновременно это значение присваивается и переменной **x**.

Например, в следующем операторе вывода будет выведено значение выражения 7.7 на экран и одновременно переменной **x** будет присвоено тоже значение: **cout << (x=2.5+5.2);**

Если после выражения с операцией присваивания поместить символ ';', это выражение становится оператором присваивания. То есть  $x = 2.5 + 5.2;$  - есть оператор простого присваивания переменной **x** значения 7.7.

В языке C++ существует целый набор "составных операций присваивания". Каждая из составных операций присваивания объединяет некоторую логическую или арифметическую операцию и собственно присваивание. Операция составного присваивания является основой для оператора составного присваивания:

**имя\_переменной ор = выражение;**

где **ор** – одна из операций  $*$  ,  $/$  ,  $\%$  ,  $+$  ,  $-$  , **&** ,  $^$  ,  $|$  ,  $<<$  ,  $>>$  .

Оператор:

**операнд\_1 ор = операнд\_2;**

эквивалентен оператору:

**операнд\_1 = операнд\_1 ор операнд\_2;**

Например, операторам:

**x \*= 4;      i += 2;      z /= 4\*i+x;**

эквивалентны такие операторы простого присваивания, как:

**x = x\*4;      i = i+2 ;      z = z / (4\*i+x) ;**

Арифметические выражения используются: в операторах присваивания, в качестве фактических параметров функций, в заголовках циклов, в операторах выбора.

Описания встроенных математических функций (sin, cos, tg, ln, lg и т.п.) представлены в заголовочном файле **math.h** и подключаются к программе следующей директивой: **#include <math.h>**.

В арифметических выражениях операндами могут быть и переменные символьного типа *char*, так как на машинном уровне эти переменные представлены целочисленным кодом и имеют такое же представление, как переменные целого типа. Поэтому переменным целого типа можно присваивать значение символа; символ может присутствовать в качестве операнда в любой арифметической операции.

**Логическое выражение** – синтаксическая конструкция для определения истинности или ложности каких-либо положений, элемент средств алгебры логики.

Логическое выражение – это несколько операндов, связанных логическими операциями, или один операнд и одноместная (унарная) логическая операция.

Тип результата логической операции (а также и всего логического выражения) – целочисленный, если результат логической операции – истина, то значение результата операции равно **1**, если результат операции – ложь, то значение результата операции равно **0**.

Логические выражения используются:

- в условных операторах, в условной операции;
- в операторах присваивания;
- в заголовках цикла;
- в качестве фактических параметров функций.

К логическим выражениям относят следующие операции:

1) *сравнения*:

<, <=, >, >= - меньше, меньше равно, больше, больше равно  
 ==, != - равно, не равно

первые четыре операции одного приоритета, две последние операции имеют более низкий приоритет;

2) *логические операции над данными любых скалярных типов* :

! - логическое отрицание НЕ, && - логическое И (логическая конъюнкция), || - логическое ИЛИ (логическая дизъюнкция).

Операции расположены в порядке убывания приоритета.

Операнды операций любых скалярных типов преобразуются к логическим значениям по правилу: все значения, отличные от нуля, трактуются как истина (1), значение равное нулю - как ложь (0);

3) *поразрядные (битовые) логические операции*:

~ - битовое инвертирование, & - поразрядное логическое умножение И, | - поразрядное логическое сложение ИЛИ, ^ - поразрядное исключающее ИЛИ;

4) *сдвиги* : <<, >> - битовые сдвиги влево и вправо.

### **Ввод – вывод данных**

Для ввода данных с клавиатуры и вывода данных на экран можно воспользоваться следующими средствами:

- использовать функции форматного ввода/вывода для работы со стандартными потоками, по умолчанию стандартные потоки связаны с клавиатурой и экраном дисплея:

**scanf()** – функция ввода данных из стандартного потока (**stdin**).

Аргументами функции **scanf()** являются *список адресов переменных* в ОП, значения которых должны быть введены из стандартного входного потока, но первым параметром функции является *строка с форматами ввода*, которые позволяют интерпретировать вводимые значения в соответствии с типами переменных;

**printf()** – функция вывода данных в стандартный поток (**stdout**).

Аргументами функции **printf()** являются *список выражений*, значения которых вычисляются и выводятся в стандартный выходной поток (на экран), а первым аргументом - *строка форматов для интерпретации выводимых значений*. Пример:

```
int i, k; float x;
```

```
scanf("%d%d%f", &i, &k, &x); //&-операция взятия адреса
```

```
printf("i = %d k = %d x = %f ", i, k, x);
```

Описания (прототипы) этих функций находятся в заголовочном файле **stdio.h**, который необходимо подключить к программе директивой:

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

- использовать непосредственно входные и выходные потоки из библиотеки классов входных и выходных потоков, описания которых находятся в заголовочном файле **iostream.h**.

Препроцессорная директива: **#include <iostream.h>** подключает к программе средства библиотеки ввода/вывода, построенной на основе механизма классов.

Поток – это обмениваемая последовательность байт. Обмен данными производится между оперативной памятью и внешними устройствами (файл на диске, принтер, клавиатура, дисплей, стример и т.п.) или между различными участками оперативной памяти.

**cin** – имя стандартного входного потока (по умолчанию связанного с клавиатурой);

**cout** – имя стандартного выходного потока (по умолчанию связанный с экраном дисплея);

>> - операция извлечения данных из потока или операция ввода;

<< - операция вставки данных в поток или операция вывода.

Операции извлечения данных из потока и вставки данных в поток являются основой для операторов ввода-вывода данных.

**Оператор ввода (ввод данных с внешнего устройства в ОП):**

```
cin >> L-значение;
```

L-значение – это обращение к именованному участку оперативной памяти, значение которого можно изменять, частный случай – имя переменной. В последнем случае формат оператора ввода таков:

**cin >> имя переменной;**

Из потока **cin** (с клавиатуры) извлекается значение и помещается в участок оперативной памяти данной переменной.

Но не так все просто. Визуальное представление данных не является формой хранения данных в ЭВМ. Внутри ЭВМ данные хранятся в виде двоичных кодов, которые регламентированы для каждого типа данных. При вводе выполняется преобразование символов из потока (с клавиатуры) в двоичные коды внутреннего представления данных, при этом происходит автоматическое распознавание типов вводимых данных. При использовании потока **cin** не надо указывать правила преобразования данных (в отличие от функции **scanf**).

**Оператор вывода (вывод данных из ОП на внешнее устройство):****cout << выражение;**

Из оперативной памяти извлекается значение выражения и помещается в выходной поток **cout** (на экран). При этом происходит преобразование двоичных кодов типизированного значения выражения в последовательность символов алфавита, изображающих значение на внешнем устройстве – на экране дисплея. Интерпретация выводимого значения производится автоматически (в отличие от функции **printf**).

**2.3. Задание на выполнение лабораторной работы****Домашнее**

1. Проработать материал лекций: Этапы решения задач на ЭВМ. Понятие алгоритма; Основные конструкции алгоритмического языка C++; Концепция данных языка C++. Выражения. Материал лекций рассмотрен в [1, с.7-89 ; 2, с. 5- 40].

2. Разработать алгоритм задачи вычисления арифметического и логического выражения согласно варианту.

**В компьютерных классах**

Разработать программу в соответствии с алгоритмом, используя оператор присваивания и операторы ввода- вывода данных.

**2.4. Порядок выполнения работы**

1. Загрузить систему Borland C++3.11 (файл **bc.exe**);
2. Создать новый файл для редактирования (**File|New**) и сохранить его на диске (**F2**) с некоторым именем;
3. Написать в окне редактирования программу, которая должна содержать:
  - объявление констант и переменных;
  - ввод с клавиатуры значений переменных, используя поток **cin** и операцию ввода данных **>>**;
  - вычисление значения арифметического выражения:
    - 1) в операторе присваивания:
      - а) используя выражение целиком и
      - б) разбив его на промежуточные переменные,



2) в параметрах функции **printf()**,

3) в операторе вывода: **cout<<выражение;**

– ввод с клавиатуры значений координат точки (x,y), используя функцию форматного ввода **scanf()** и поток **cin** и операцию ввода **>>;**

– вычисление значения логического выражения в операторе вывода в выходной поток **cout** и в операторе присваивания;

– комментарий - заголовок с фамилией исполнителя и наименованием лабораторной работы и пояснительные комментарии по тексту программы.

4. Провести компиляцию, отладку, тестирование программы, предварительно подготовив данные для тестирования.

5. Составить отчет.

## 2.5. Пример варианта лабораторной работы

Задание

1. Дана формула для вычисления переменной z:

$$z = \frac{a + y^b}{(e^{ay+1} - \sin^3(x)) \cdot \left(2,25 \cdot 10^2 - \frac{x \cdot y}{b}\right)}.$$

Разработать программы для вычисления значения z с использованием арифметического выражения, операторов присваивания и вывода на экран.

Значения переменных x, y ввести с клавиатуры, а константам a и b задать следующие значения a = 0.89, b = 7.56.

2. Разработать программу для вычисления логического выражения, значение которого есть истина, если координаты точки попадают в затемненную область фигуры на рис. 2.1, и ложь, если нет.

Значения координат точки x и y вводить с клавиатуры.

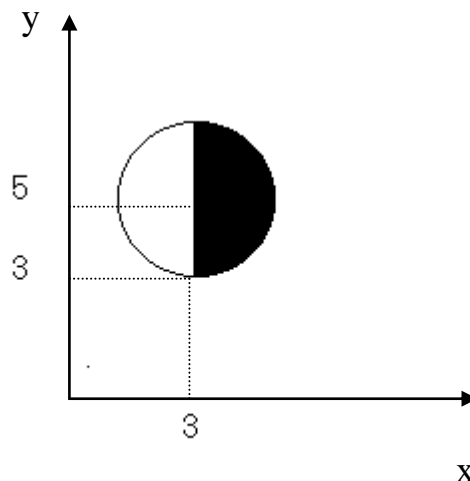


Рис. 2.1. Фигура для вычисления логического выражения

Текст программы

//Лабораторная работа №1 студента группы ЭВМ 1-1 Петрова Ивана

//Программирование алгоритмов линейной структуры

//Вычисление выражений

```

#include <iostream.h>          //директивы
#include <stdio.h>
#include <math.h>
#include <conio.h>             // препроцессора
const float a=0.89, b= 7.56;   // определение глобальных констант
//----- Главная функция-----
void main ()
{
float x, y, z, t, q;           //определение локальных переменных
clrscr();                     // функция очистки экрана
cout<< "Введите переменные\nx="; // вывод на экран строковой константы
cin>>x;                       //ввод значения с клавиатуры
cout<<"y="; cin>>y;
cout<<"\nПромежуточные переменные:";
//операторы присваивания:
t= a+pow(y, b);
q=(exp(a*y+1)-pow(sin(x),3))*(2.25e+02-x*y/b);
z=t/q;
cout<<"\nt="<<t<<"\nq="<<q
<<"\n\nРезультат с промежуточными переменными:\nз="<<z;
z= (a+pow(y,b))/(exp(a*y+1)-pow(sin(x),3))/(2.25e+02-x*y/b);
cout<<"\n\nРезультат с помощью одного выражения:\nз=" <<z;
cout<<"\n\nРезультат с помощью выражения в операторе вывода :\nз="
<< (a+ pow(y,b))/(exp(a*y+1)-pow(sin(x),3))/(2.25e+02-x*y/b);
printf("\n\nРезультат с помощью выражения- параметра функции \
printf:\nз=%12f", (a+pow(y,b))/(exp(a*y+1)-pow(sin(x),3))/(2.25e+02-x*y/b));

//Вычисление условного выражения
int i ;
cout<<"\n\nВведите координаты точки\nx=";
cin>>x;
cout<<"y="; cin>>y;
//Вычисление выражения в операторе вывода cout<<...;
cout<<"\n\nЗначение выражения:\n"
<<((pow(x-3,2)+pow(y-5,2)<=4) && (x >= 3));
//Использование условной операции для вывода слов true или false
((pow(x-3,2)+pow(y-5,2)<=4) && (x >= 3))? cout<<" - true":cout<<"-false";
// Вычисление выражения в операторе присваивания
printf("\n\nВведите координаты точки еще раз \nx= ");
scanf(" %f",&x );
printf("y="); scanf(" %f", &y );
i=((pow(x-3,2)+pow(y-5,2)<=4) && (x >= 3));
printf("\n\nЗначение выражения: %d", i );
}

```

## 2.6. Контрольные вопросы

1. Этапы обработки программы.
2. Что такое трансляция?
3. Какие виды работ можно выполнить с помощью текстового редактора?
4. Структура и состав меню системы Borland C++3.1.
5. Структура программы на языке C++.
6. Что такое константы и переменные программы языка C++?
7. Какие категории констант имеются в C++?
8. Целочисленные и вещественные константы.
9. Символьные константы и строки.
10. Объявление констант и переменных в программе.
11. Операции C++. Приоритет операций.
12. Что определяет тип данных?
13. Каковы основные характеристики простых типов C++?
14. Каково назначение выражений и из каких элементов они формируются?
15. Типы выражений.
16. Арифметические выражения (операнды, операции, вызовы встроенных функций, используемых в арифметических выражениях, порядок вычисления выражений, конструкции языка, в которых используются арифметические выражения).
17. Логические выражения - выражения сравнения, логические выражения с операндами любых типов и логические выражения с битовыми операциями.
18. Что такое операторы программы?
19. Оператор и операция присваивания.
20. Как произвести ввод данных с клавиатуры и вывод данных на экран?

## 3. ЛАБОРАТОРНАЯ РАБОТА № 2

### **Разработка алгоритмов разветвляющейся и циклической структуры. Разработка программ для работы в режиме диалога с пользователем**

#### 3.1. Цель лабораторной работы

Целью лабораторной работы является освоение:

- организации диалога с пользователем с использованием алгоритмов разветвляющейся структуры;
- объявления и использования символьных массивов для хранения текстовых строк и массивов числовых данных (многомерных);
- ввода/вывода данных числовых типов и символьных строк;
- организации обработки числовых массивов с использованием алгоритмов циклической структуры;
- использования операторов **if**, **for** и **switch** для организации обработки данных.

### 3.2. Теоретические сведения

#### Массивы

*Массив* – это совокупность данных одного типа, рассматриваемых как единое целое. Все элементы массива пронумерованы. Массив в целом характеризуется **именем**. Обращение к элементам массива выполняется по их номерам (**индексам**), которые всегда начинаются с **0**.

Массивы могут состоять из числовых данных, символов, строк, указателей и т. д. Символьные массивы, как правило, представляют в программе текстовые строки.

Если для обращения к какому-то элементу массива достаточно одного индекса, массив называется *одномерным*.

Если данные удобно представлять не в виде линейной последовательности, а в форме таблицы (матрицы), в которой данные занимают несколько строк, тогда для обращения к конкретному элементу надо задать два индекса: номер строки и номер элемента в этой строке (номер столбца). Такие массивы называются *двумерными*.

Массивы с числом индексов больше 1 называются *многомерными*.

Форма объявления одномерного массива (вектора):

**type имя массива [K];**

**K** – константное выражение, определяет *размер* массива (количество элементов в массиве);

**type** – тип элементов массива.

Например, **int A[10];** определяет массив из 10 элементов типа **int**, индексы которых принимают значения от 0 до 9.

Форма объявления многомерного массива:

**type имя массива [ K 1] [ K2] ...[K N];**

**type** – тип элементов массива;

**N** -размерность массива- количество индексов, необходимое для обозначения конкретного элемента;

**K1...KN** - количество элементов в массиве по 1...N-у измерениям, так в двумерном массиве **K1** – количество строк, а **K2** – количество столбцов;

Значения индексов по **i**-му измерению могут изменяться от **0** до **Ki – 1**;

**K1\*K2\*...\*KN** – размер массива (количество элементов массива).

Например, **float Z[13][6];** определяет двумерный массив, первый индекс которого принимает 13 значений от 0 до 12, а второй индекс принимает 6 значений от 0 до 5.

Обращение к элементам массива

С помощью операции **[]** (квадратные скобки) обеспечивается доступ к произвольному элементу массива.

Обращение к элементу одномерного массива:

**имя массива [индекс],**

где **индекс** – это не номер элемента, а его **смещение** относительно первого элемента с индексом **0**. Пример:

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

**A[5] = 0;** //A[5] – обращение к шестому (5+1) элементу массива.

Для обращения к элементам многомерного массива также используется имя массива, после которого в квадратных скобках стоит столько индексов, сколько измерений в массиве. Пример обращения к элементам двумерного массива:

**имя массива [ i ][ j ]**

это обращение к элементу **i** –ой строки и **j**-го столбца двумерного массива. Первый индекс – это индекс строки, второй индекс – индекс столбца.

### Внутреннее представление массива

При определении массива для его элементов выделяется участок памяти, размеры которого определяются количеством элементов массива и их типом:

**sizeof (type)\* количество элементов массива,**

где **sizeof(type)** – количество байт, выделяемое для одного элемент массива данного типа.

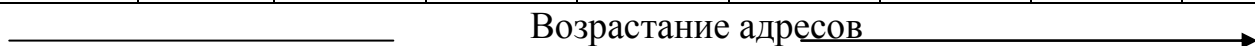
Операция **sizeof** имеет две формы: **sizeof(тип)** и **sizeof(объект)**. Учитывая это, а также то, что имя массива – это имя структурированной переменной, размер участка памяти, выделенного для всего массива, можно определить также из следующего выражения:

**sizeof (имя массива)**

В оперативной памяти все элементы массива располагаются подряд. Адреса элементов одномерных массивов увеличиваются от первого элемента к последнему. В многомерных массивах элементы следуют так, что при переходе от младших адресов к старшим наиболее быстро меняется крайний правый индекс массива. Так, при размещении двумерного массива в памяти сначала располагаются элементы первой строки, затем второй, третьей и т. д.

Например, элементы массива **int T [3][3]** будут располагаться так:

первая строка			вторая строка			третья строка		
T[0][0]	T[0][1]	T[0][2]	T[1][0]	T[1][1]	T[1][2]	T[2][0]	T[2][1]	T[2][2]


 Возрастание адресов →

### Еще один способ обращения к элементам массива

С одной стороны, **имя массива** следует рассматривать как имя структурированной переменной. И применение таких операций, как **sizeof** и **&** (получения адреса) к имени массива дают в качестве результата соответственно размер внутреннего представления в байтах всего массива и адрес первого элемента массива:

**sizeof(имя массива)** – длина в байтах внутреннего представления массива;

**&имя массива** - адрес массива в целом, равный адресу первого элемента массива.

С другой стороны, **имя массива** это **константный указатель**, значением которого является адрес первого элемента массива и значение данного указателя нельзя изменять.

Рассмотрим одномерный массив. В соответствии с вышесказанным соблюдается равенство:

**имя\_массива == &имя\_массива == &имя\_массива[0],**

то есть имя массива отождествляется с адресом массива в целом и с адресом его первого элемента.

В соответствии с операцией сложения указателя с целым числом, если к имени массива (указателю) прибавить целое число **i** :

**имя\_массива + i ,**

то на машинном уровне сформируется следующее значение адреса:

**имя\_массива + i \* sizeof(тип элемента),**

которое равно адресу **i** - го элемента массива, и следовательно можно записать:

**&имя\_массива[ i ] == имя\_массива + i .**

Операция разыменования адреса объекта предоставляет доступ к самому объекту. Применяя операцию разыменования для левой и правой части представленного выше уравнения, получаем результат:

**имя\_массива [ i ] == \* ( имя\_массива + i ),**

из которого следует, что обращаться к **i**-му элементу массива можно любым из этих эквивалентных способов.

*Многомерные массивы* рассмотрим на примере объявления двумерного массива:

**type T [m][n];**

**m, n** – целочисленные константы, **type** – тип элемента массива. В массиве **m** строк по **n** элементов в строке ( **n** столбцов).

Имя двумерного массива **T** также отождествляется с его адресом, а также с адресом его первого элемента **T[0][0]**.

Адрес любого элемента массива получается с помощью операции **&** (например, **&T[2][1]** – адрес второго элемента третьей строки массива).

Каждая строка двумерного массива - это одномерный массив с именем **T[i]**, где **i** – индекс строки и имя этого одномерного массива также является адресом первого элемента строки и адресом строки в целом.

Таким образом, адреса строк массива равны **T[0] , T[1] , ..., T[m-1]**, что эквивалентно, как показано выше, следующим выражениям: **\*T, \*(T+1) , ..., \*(T+m-1)**.

**T[ i ] == \*(T+i) == &T[i][0]** - адрес **i** –строки массива,

и, следовательно, обращаться к элементу **i**-ой строки **j**-го столбца массива можно одним из эквивалентных способов:

**T[ i ][ j ] == \*(\*(T+i)+j)**

### Инициализация массивов

Инициализация - это задание начальных значений объектов программы при их определении, проводится на этапе компиляции (формировании объектного кода программы).

Инициализация одномерных массивов возможна двумя способами: либо с указанием размера массива в квадратных скобках, либо без явного указания:

```
int test [ 4] = { 10, 20 , 30 ,40 };  
char ch [ ] = { 'a' , 'b' , 'c' , 'd' };
```

Список инициализации помещается в фигурные скобки при определении массива.

В первом случае инициализация могла быть не полной, если бы в фигурных скобках находилось меньше значений, чем четыре.

Во втором случае инициализация должна быть полной, и количество элементов массива компилятор определяет по числу значений в списке инициализации.

Чаще всего для инициализации символьных массивов используются строки.

*Строка или строковая константа*- это последовательность символов, заключенная в кавычки. Размещая строку в памяти, транслятор автоматически добавляет в ее конец байтовый ноль - символ с кодом равным нулю '\0'.

Инициация символьного массива может быть выполнена значением строковой константы, например, следующим образом:

```
char stroka [10] ="строка";
```

При такой инициализации компилятор запишет в память символы строковой константы и добавит в конце двоичный ноль '\0', при этом памяти будет выделено с запасом (10 байт).

Можно объявлять символьный массив без указания его длины:

```
char stroka1 [ ] = "строка";
```

Компилятор, выделяя память для массива, определит его длину, которая в данном случае будет равна 7 байт (6 байт под собственно строку и один байт под завершающий ноль):

Инициализация двумерных числовых и символьных массивов:

```
int T [3][4] = { {1,2,3,4} , { 10,20,30,40}, {100,200,300,400} };  
char name [ 5 ] [18] = { " Иванов" ,"Петров" ,"Розенбаум", "Цой",  
"Григорьев"};
```

При объявлении символьного массива будет выделено памяти по 18 байт на каждую строку, в которые будут записаны символы строки и в конце каждой строки добавлен байтовый ноль, всего 90 байт.

При таком объявлении в массиве остается много пустого места на случай программного изменения значений строк.

При определении многомерных массивов с инициализацией (в том числе и двумерных) значение первого индекса в квадратных скобках можно опускать.

Количество элементов массива по первому измерению компилятор определит из списка инициализации. Например, при определении:

```
char sh [ ] [40] = {“=====”,  

“| F | F | F | F | F |”,  

“=====”};
```

компилятор определит три строки массива по 40 элементов каждая, причем **sh[0]**, **sh[1]**, **sh[2]** – адреса этих строк массива в оперативной памяти. В каждую из строк будет записана строковая константа из списка инициализации.

#### Ввод/вывод элементов массивов

##### Ввод/вывод числовых массивов

Ввод/вывод значений арифметических массивов производится поэлементно. Следует организовать цикл (повторение обработки данных), в котором от итерации, к итерации, изменяя индексы элемента, производить ввод и вывод значений соответствующих элементов.

Например, для одномерного массива **int A[100]** схема алгоритма поэлементного ввода значений массива с клавиатуры и вывода значений на экран дана на рис. 3.1.

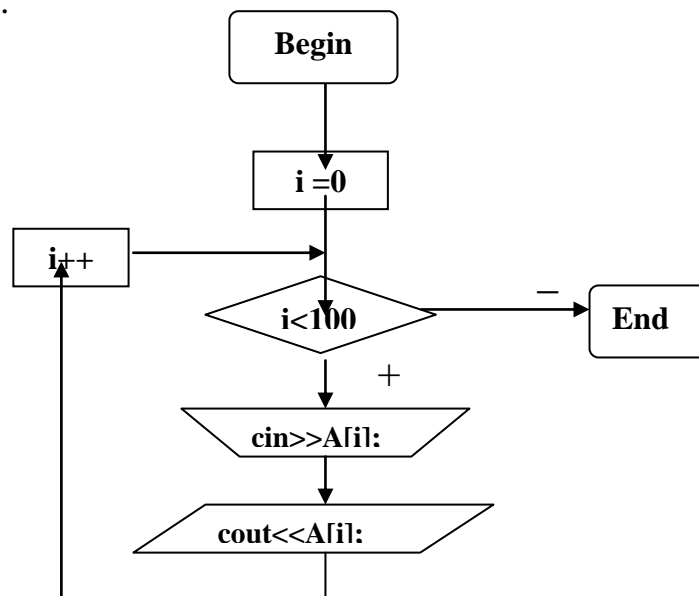


Рис. 3.1. Схема алгоритма ввода/вывода элементов одномерного массива

Поясним схему алгоритма:

- 1) устанавливается начальное значение индекса массива (**i = 0**).
- 2) проверяется условие, при котором выполняется обработка массива: значение индекса не должно превышать максимального значения индекса в массиве (**i < 100**).

Если условие истинно, то выполняется обработка:

- вводится с клавиатуры значение элемента массива с данным индексом (**cin >> A[i]**);
- значение элемента выводится на экран (**cout << A[i]**);
- значение индекса элемента увеличивается на единицу (**i++**);
- управление передается пункту 2.



Если условие ложно - обработка завершается.

При написании схемы алгоритма для двумерного массива надо учитывать, как элементы массива располагаются в оперативной памяти. Внешний цикл следует организовать по номерам строк. В теле этого цикла для каждого номера строки организовать внутренний цикл, в котором перебирать номера столбцов. Следуя такому алгоритму, мы обращаемся к элементам массива в той последовательности, в которой они располагаются в оперативной памяти.

Для двумерного массива **float T[3][4]** схема алгоритма представлена на рис. 3.2.

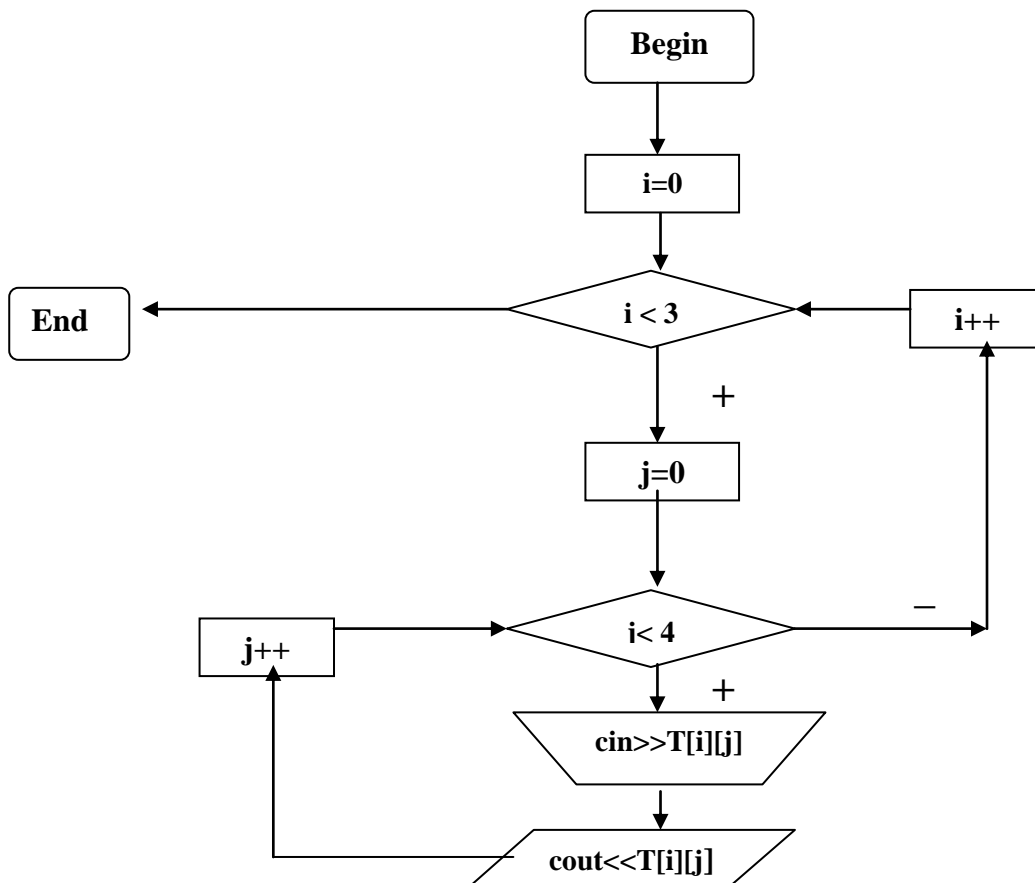


Рис. 3.2. Схема алгоритма ввода/вывода значений элементов двумерного массива

### Ввод/вывод символьных массивов

Ввод и вывод символьных массивов можно производить поэлементно, как и числовых массивов, т.е. рассматривать символьный массив как набор отдельных символов.

Синтаксис языка C++ допускает также обращение к символьному массиву целиком по его имени, а именно по адресу этого массива в оперативной памяти. При этом также допускается обращение к отдельным элементам – символам по их индексу в массиве.

Объявим некоторый символьный массив:

**char text [80];**

Следующие операторы позволяют произвести ввод символьных строк:

1) **cin>>text;** - символы извлекаются из стандартного входного потока **cin** и заносятся в оперативную память по адресу **text**, ввод начинается от первого символа, отличного от пробела до первого обобщенного пробела. В конце строки в память помещается двоичный ноль.

2) **cin.getline(text, n );** - извлекаются из стандартного входного потока **cin** любые символы, включая и пробелы, и заносятся в оперативную память по адресу **text**. Ввод происходит до наступления одного из событий: прочитан **n-1** символ или ранее появился символ перехода на новую строку **'\n'** (при вводе с клавиатуры это означает, что была нажата клавиша **Enter**). В последнем случае из потока символ **'\n'** извлекается, но в память не помещается, а помещается в память символ конца строки **'\0'**.

3) **gets(text);** - читается строка из стандартного потока (по умолчанию связанного с клавиатурой) и помещается по адресу **text**. Вводятся все символы до символа перехода на новую строку **'\n' (Enter)**, который в память не записывается, а в конце строки помещает двоичный ноль **'\0'**.

4) **scanf("%s", text);** –из стандартного потока читается строка символов до очередного пробела и вводит в массив **text**. В конце помещается байтовый ноль. Если строка формата имеет вид **"%ns"**, то считывается **n** непробельных символов.

5) **scanf("%nc", text);** – из стандартного потока вводятся **n** любых символов, включая и пробелы, и символ конца строки.

Если стандартный входной поток связан с клавиатурой, все приведенные выше операторы, в основе которых лежат вызовы функций, останавливают программу до ввода строки символов.

Вывод строки позволяют произвести следующие операторы:

1) **cout << text;** - выводит всю строку до байтового нуля в стандартный выходной поток **cout**, по умолчанию связанный с экраном дисплея;

2) **puts(text);** - выводит строку в стандартный поток и добавляет в завершении символ **'\n'** – перехода на новую строку;

3) **printf("%s", text);** - выводит в стандартный выходной поток всю строку;

**printf("%ws", text);** - выводит всю строку в поле **w**, где **w** – целое число, количество текстовых позиций на экране для вывода символов. Если **w** больше числа символов в строке, то слева (по умолчанию) или справа (формат **"%-ws"**) от строки выводятся пробелы. Если **w** меньше, чем количество выводимых символов, то выводится вся строка;

**printf("%w.ns", text);** - выводит **n** символов строки в поле **w**;

**printf("%.ns", text);** -выводит **n** символов строки в поле **w = n**;

### 3.3. Задание на выполнение лабораторной работы

#### Дома

1. Проработать материал лекций: Основные операторы языка C++; Адреса, указатели, ссылки; Индексированные данные. Материал лекций рассмотрен в [1, с.90-168; 2, с.40- 123].

2. Разработать алгоритм диалога с пользователем согласно варианту.

3. Разработать алгоритм обработки массивов числовых и символьных данных согласно варианту.

#### В компьютерном классе

Разработать программу диалога с пользователем и программу обработки массивов числовых и символьных данных. Использовать операторы организации обработки данных: вложенные условные операторы, оператор выбора варианта и операторы циклов.

### 3.4. Порядок выполнения работы

1. Разработать алгоритм диалога с пользователем.

В программе выводятся вопросы пользователю. На вопросы пользователь дает ответ, значения ответов вводятся с клавиатуры в переменные программы строковые и целочисленные. Затем значение целочисленной переменной анализируется с помощью оператора **if**. В зависимости от значения переменной на экран выводится та или иная фраза.

2. Открыть новый файл в редакторе среды Borland C++, сохранить его на диске с некоторым именем.

3. Написать в файле текст программы в соответствии с алгоритмом.

4. Провести отладку и тестирование программы.

5. Вывести на печать текст программы и текст результатов тестирования.

6. Разработать алгоритм обработки массива числовых данных:

- определить массив данных вещественного типа размером **5x6**;
- ввести значения массива с клавиатуры;
- вывести элементы массива на экран в виде матрицы через интервал;
- определить символьный массив, инициализировать его строками шапки таблицы.

Для воспроизведения в тексте программы строк шапки следует использовать символы псевдографики. Для этого надо установить режим работы с цифровой клавиатурой, нажав клавишу **Num Lock**. Затем надо нажать клавишу **Alt** и, не отпуская ее, набрать на цифровой клавиатуре (правая часть клавиатуры) код символа. Таблица кодов всех символов ПК, в том числе и символов псевдографики, приведена, например, в [1, с. 490-491];

- вывести на экран элементы числового массива в таблицу в формате с плавающей точкой и с фиксированной точкой в зависимости от номера столбца массива. Для выбора варианта использовать оператор **switch**;

- провести обработку массива согласно варианту.

7. Написать в файле текст программы в соответствии с алгоритмом.

8. Провести отладку и тестирование программы.

9. Распечатать текст программы и результаты выполнения программы.

### 3.5. Пример варианта лабораторной работы

#### Задание

1. Написать программы диалога пользователя- продавца с покупателем в книжном магазине:

Покупатель: «Что у Вас есть по Турбо-Паскалю?»

Ответ пользователя: «Учебное пособие Фаронова.»

Покупатель: «Учебное пособие Фаронова - это то, что я ищу. Сколько стоит книга?»

Ответ пользователя: вводит цену в рублях (цена - целое число).

Если цена  $\leq 100$ , то фраза покупателя: « О! цена, - это мне подходит. Покупаю!»

Если  $100 < \text{цена} \leq 150$ , то покупатель: «Да, цена, - это дорого, но книга мне нужна. Пожалуй, возьму»

Если цена  $> 150$ , то фраза покупателя: «Обойдусь без книги»

Значения ответов пользователя вводить в переменные:

**name** – имя символьного массива для ввода названия книги;

**price** – переменная типа **int** для введения цены книги.

2. Написать программу обработки числового массива размером 4x3: ввод элементов массива с клавиатуры; вывод элементов массива на экран в виде матрицы и в таблицу в формате с плавающей и фиксированной точкой; вычисление и вывод на экран суммы элементов массива.

#### Текст программы

//Лабораторная работа №2 студента группы ЭВМ 1-1 Иванова Петра

//Программа диалога с пользователем

#include <iostream.h>

#include <iomanip.h>

#include <conio.h>

void main()

{

clrscr(); // Очистка экрана пользователя

char name[40];

int price;

cout<<"\n\n Что у Вас есть по Турбо-Паскалю? \n";

//Ввод строки (название книги) с клавиатуры

cin.getline(name,40);

cout<<name<<"-это то, что я ищу.\n";

cout<<"Сколько стоит книга?\n";

cin>>price; // Ввод цены с клавиатуры

//анализ цены

if(price <= 100)

cout<<"О!, " << price <<" , - это мне подходит." "Покупаю!";

else

if(price <= 150)

```

cout<<"Да,"<< price <<" , - это дорого, но книга мне нужна. Пожалуй, возьму.";
else
cout<<"Обойдусь без книги";
getch();    // Задержка выполнения программы до нажатия любой клавиши
clrscr();
//Программа обработки массива арифметических данных
float M[4][3];
int i,j;
//Ввод/вывод элементов массива
//Нахождение суммы элементов массива
float s=0;    // переменная для нахождения суммы
cout<<"\n Массив М: ";
for(i=0;i<4;i++)
{cout<<"\n";
for(j=0;j<3;j++)
//ввод/вывод элементов и подсчет суммы
{cin >> M[i][j]; cout << M[i][j] << " "; s+=M[i][j];}
}
//вывод на экран суммы
cout<<"\nСумма массива ="<< s;
//Вывод массива в таблицу
//Массив строк шапки таблицы:
char sh[][41]={
"
                                Массив данных                                ",
"||=====||",
"||   Данные 1   ||   Данные 2   ||   Данные 3   ||",
"||=====||",
"||-----||",
"||=====||"};
//Вывод на экран строк шапки
for(i=0;i<4;i++)
cout<<sh[i]<<endl;
//Цикл for по строкам массива
for(i=0;i<4;i++)
{
cout<<"\272";
for(j=0;j<3;j++) //Цикл for по столбцам массива
//форматный вывод элементов, отличающийся для разных столбцов
switch(j)
{
case 0:case 1:cout.setf(ios::scientific);cout<<setw(12)<<M[i][j]<<"\272";break;
case 2:cout.setf(ios::fixed);cout<<setw(12)<<M[i][j]<<"\272\n";break;
}
}

```

```

if(i==3)
cout<<sh[5]<<endl;
else
cout<<sh[4]<<endl;
}
}

```

### 3.6. Контрольные вопросы

1. Классификация и характеристики основных типов данных.
2. Перечисляемый тип данных.
3. Определение нового типа с помощью **typedef**.
4. Форматы определения массивов. Инициализация.
5. Массивы арифметических данных. Формат внутреннего представления одномерных и многомерных массивов.
6. Ввод/вывод элементов массивов.
7. Символьные массивы.
8. Ввод/вывод строк.
9. Определение переменных и именованных констант в программе (с учетом спецификаторов и модификаторов).
10. Классы памяти и что они определяют.
11. Автоматические переменные.
12. Статические переменные.
13. Внешние переменные.
14. Классификация операторов C++.
15. Операторы обработки данных.
16. Операторы выбора.
17. Операторы циклов.
18. Оператор управления работой программы – пустой и составной.
19. Операторы передачи управления.
20. Как вывести в окно просмотра введенных в программе значений?

## 4. ЛАБОРАТОРНАЯ РАБОТА № 3

### **Разработка программ с использованием функций для обработки массивов арифметических и символьных данных**

#### 4.1. Цель лабораторной работы

Целью лабораторной работы является получение навыков программирования с использованием функций – основных программных единиц языка C++ и освоение:

- правил определения функций;
- назначения и состава параметров функции;
- передачи параметров по значению, адресу, ссылке;
- методов передачи в функцию массивов данных;
- правил вызова функций;
- построения функций обрабатывающих массивы.

## 4.2. Теоретические сведения

### Функции

Функции представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд частей, реализующих относительно самостоятельные смысловые части алгоритма.

Программа на языке C++ представляет собой совокупность произвольного количества функций, одна (и единственная) из которых обязательно главная функция с именем **main**. Выполнение программы начинается и заканчивается выполнением функции **main**. Выполнение неглавных функций иницируется в главной функции непосредственно или в других функциях, которые сами иницируются в главной.

Функции – это относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем.

Каждая функция существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы.

Упоминание имени функции в тексте программы называется *вызовом функции*. При вызове функции активизируется последовательность образующих ее операторов, а с помощью передаваемых функции параметров осуществляется обмен данными между функцией и вызывающей ее программой.

По умолчанию все функции внешние (класс памяти **extern**), доступны во всех файлах программы. При определении функции допускается класс памяти **static**, если надо чтобы функция использовалась только в данном файле программы.

### Определение функций

Определение функции – это программный текст функции. Оно может располагаться в любой части программы, кроме как внутри других функций. В языке C++ нет вложенных функций.

Определение состоит из заголовка и тела функции:

**<тип> <имя функции> (<список формальных параметров>)  
тело функции**

1. **тип** – тип возвращаемого функцией значения, с помощью оператора **return**, если функция не возвращает никакого значения, на место типа следует поместить слово **void**;

2. **имя функции** – идентификатор, уникальный в программе;

3. **список формальных параметров (сигнатура параметров)** – заключенный в круглые скобки список спецификаций отдельных формальных параметров, перечисляемых через запятую:

**<тип параметра> <имя параметра> ,**

**<тип параметра> <имя параметра> = <умалчиваемое значение> ,**

если параметры отсутствуют, в заголовке после имени функции должны стоять, либо пустые скобки **()**, либо скобки – **(void)**;

для формального параметра может быть задано, а может и отсутствовать умалчиваемое значение – начальное значение параметра;

4. **тело функции** – это блок или составной оператор, т.е. последовательность определений, описаний и операторов, заключенная в фигурные скобки.

### Переменные, доступные функции

#### 1. Локальные переменные:

- объявлены в теле функции, доступны и используются только в теле функции;
- при определении переменной ей выделяется память в *сегменте стека*, при завершении выполнения функции память освобождается;
- вне тела функции локальные переменные не существуют;
- обладают классом памяти **avto** (автоматические переменные);

#### 2. Формальные параметры:

- объявлены в заголовке функции и доступны только функции;
- формальные параметры за исключением параметров–ссылок являются локальными переменными, память им выделяется в стеке;
- параметр–ссылка доступен только функции, но он не является переменной, на него не выделяется память, это некоторая абстракция для обозначения внешней по отношению к функции переменной;

#### 3. Глобальные переменные:

- переменные объявлены в программе как внешние, т.е. вне всех функций, включая и главную функцию **main**;
- область действия таких переменных – вся программа от точки объявления переменной;
- память внешним переменным выделяется на этапе компиляции программы в *сегменте данных* и не освобождается до конца программы;
- чтобы глобальная переменная была доступна функции, функция не должна содержать локальных переменных и формальных параметров с тем же именем; локальное имя «закрывает» глобальное и делает его недоступным;

### Формальные и фактические параметры функции

Посредством формальных параметров осуществляется обмен данными между функцией и вызывающей ее программой. В функцию данные передаются для обработки. Функция, обработав эти данные, может вернуть в вызывающую функцию результат обработки.

В определении функции фигурируют формальные параметры, которые показывают, какие данные следует передавать в функцию при ее вызове, и как они будут обрабатываться операторами функции.

Список формальных параметров (список аргументов) функции указывает, с какими фактическими параметрами следует вызывать функцию.

Фактические параметры передаются в функцию при ее вызове, заменяя формальные параметры. Фактические параметры, по количеству, по типу (он



должен быть идентичным или совместимым), по расположению должны соответствовать формальным параметрам.

### Оператор **return**

Оператор **return** - оператор возврата **управления программой** и **значения** в точку вызова функции. С помощью этого оператора функция может вернуть одно скалярное значение любого типа. Форма оператора:

**return (выражение);**

- выражение определяет значение, возвращаемое функцией; выражение вычисляется, результат преобразуется к типу возвращаемого значения и передается в точку вызова функции;
- круглые скобки, ограничивающие выражение, не обязательны;
- если функция не возвращает результата, оператор может либо отсутствовать, либо присутствовать с пустым выражением: **return;**
- функция может иметь несколько операторов **return** с различными выражениями, если алгоритм функции предусматривает разветвление.

Функция завершается как только встречается оператор **return**. Если функция не возвращает результата и не имеет оператора **return**, она завершается по окончании тела функции.

### Вызов функции

Вызов функции передает ей управление, а также фактические параметры при наличии в определении функции формальных параметров.

Форма вызова функции:

**имя функции (список фактических параметров);**

Список фактических параметров может быть пустым, если функция без параметров: **имя функции ();**

Фактические параметры должны соответствовать формальным параметрам по количеству, типу и по расположению параметров.

Если функция возвращает результат, то ее вызов представляет собой выражение с операцией «круглые скобки». Операндами служат имя функции и список фактических параметров. Значением выражения является возвращаемое функцией значение.

Если функция не возвращает результата (тип – **void**), вызов функции представляет собой оператор.

При вызове функции происходит передача фактических параметров из вызывающей программы в функцию, и именно эти параметры обрабатываются в теле функции вместо соответствующих формальных параметров.

После завершения выполнения всех операторов функция возвращает управление программой в точку вызова.

## Умалчиваемые значения параметров

Формальный параметр может содержать умалчиваемое значение. В этом случае при вызове функции соответствующий фактический параметр может быть опущен и умалчиваемое значение используется в качестве фактического параметра. При задании умалчиваемых значений должно соблюдаться правило. Если параметр имеет умалчиваемое значение, то все параметры справа от него также должны иметь умалчиваемые значения.

### Описание функции (прототип)

При вызове функции формальные параметры заменяются фактическими, причем соблюдается строгое соответствие параметров по типам. При этом не предусматривается автоматическое преобразование типов (как в языке Си) в тех случаях, когда фактические параметры не совпадают по типам с соответствующими им формальными параметрами. В связи с этой особенностью языка Си++ проверка соответствия типов формальных и фактических параметров выполняется на этапе компиляции.

Строгое согласование по типам между параметрами требует, чтобы в модуле программы до первого обращения к функции было помещено либо ее определение, либо ее описание (прототип), содержащее сведения о типе результата и о типах всех параметров.

Прототип (описание) функции может внешне почти полностью совпадать с заголовком определения функции:

**<Тип функции > <имя функции>**

**( <спецификация формальных параметров> );**

Отличие описания от заголовка определения функции состоит в следующем:

- **наличие** ‘ ; ‘ в конце описания – это основное отличие и
- **необязательность** имен параметров, достаточно через запятые перечислить типы параметров.

### Передача фактических параметров

В С++ передача параметров в вызываемую функцию может осуществляться тремя способами:

- по значению, когда в функцию передается числовое значение параметра;
- по адресу, когда в функцию передается не значение параметра, а его адрес, что особенно удобно для передачи в качестве параметров массивов;
- по ссылке, когда в функцию передается не числовое значение параметра, а сам параметр и тем самым обеспечивается доступ из тела функции к участку памяти, выделенному для фактического параметра.

### Передача параметров по значению

Формальным параметром может быть только имя скалярной переменной стандартного типа или имя структуры, определенной пользователем.

При вызове функции формальному параметру выделяется память в стеке в соответствии с его типом. Фактическим параметром является выражение, **значение** которого копируется в стек, в область ОП, выделенную под формальный параметр. Фактическим параметром может быть просто неименованная константа нужного типа или имя некоторой переменной, значение которой будет использовано как фактический параметр.

Все изменения, происходящие с формальным параметром в теле функции, не передаются переменной, **значение** которой являлось фактическим параметром функции.

### **Передача параметров по адресу - по указателю**

Формальным параметром является указатель **type\*p** на переменную типа **type**. При вызове функции формальному параметру-указателю выделяется память в стеке 2 байта, если указатель ближний и 4 байта, если указатель дальний. Соответствующим фактическим параметром функции должно быть значение указателя, которое при вызове функции копируется в стек, в область ОП, выделенную под формальный параметр.

Фактическим параметром может быть либо адрес в ОП переменной типа **type**, либо значение другого указателя, типа **type\***, из вызывающей программы.

В область памяти (в стеке), выделенную для указателя **p** будет копироваться значение некоторого **адреса** из вызывающей функции.

В теле функции, используя операцию разыменования указателя **\*p**, можно получить доступ к участку памяти, адрес которого получил **p** при вызове функции, и изменить содержимое этого участка.

Если в теле функции изменяется значение **\*p**, при вызове функции эти изменения произойдут с тем объектом вызывающей программы, адрес которого использовался в качестве фактического параметра.

### **Передача параметров по ссылке**

В языке C++ ссылка определена как другое имя уже существующей переменной. При определении ссылки оперативная память не выделяется. Инициализатор, являющийся обязательным атрибутом определения ссылки, представляет собой имя переменной того же типа, имеющей место в памяти. Ссылка становится синонимом этой переменной.

#### **type & имя ссылки = имя переменной;**

Основные достоинства ссылок проявляются при работе с функциями.

Если определить ссылку **type & a** и не инициализировать ее некоторой переменной, то это равносильно созданию объекта, который имеет имя, но не связан ни с каким участком памяти. Это является ошибкой.

Однако такое определение допускается в спецификациях формальных параметров функций. Если в качестве формального параметра функции была определена неинициализированная ссылка - некоторая абстрактная переменная - которая имеет имя, но не имеет адреса, в качестве фактического параметра при вызове функции следует использовать имя переменной из вызывающей

программы того же типа, что и ссылка. Эта переменная инициализирует ссылку, т.е. связывает ссылку с участком памяти.

Таким образом, ссылка обеспечивает доступ из функции непосредственно к внешнему участку памяти той переменной, которая при вызове функции будет использоваться в качестве фактического параметра.

Все изменения, происходящие в функции со ссылкой, будут происходить непосредственно с переменной, являющейся фактическим параметром.

Фактическим параметром является имя переменной из внешней программы того же типа, что и ссылка. Функция изменяет значение фактического параметра.

Это наиболее перспективный метод передачи параметров, так как в этом случае вообще не происходит копирование фактического параметра в стек, будь то значение или адрес, функция непосредственно оперирует с внешними по отношению к ней переменными, используемыми в качестве фактических параметров.

### Формальные параметры – массивы

Массив в качестве фактического параметра может быть передан в функцию только по адресу, то есть с использованием указателя.

Если массив–параметр не есть символьная строка, то нужно либо использовать только массивы фиксированного, заранее определенного размера, либо передавать значение размера массива явным образом с помощью дополнительного параметра.

При работе со строками, т.е. с массивами данных типа **char**, последний элемент каждого из которых имеет значение **'\0'**, анализируется каждый элемент, пока не встретится символ **'\0'**, этот символ и считается последним элементом массива.

В качестве формального параметра массива данных можно использовать:

- 1) определение массива с фиксированными границами, например:

**float A[5];          int B[3][4];          char S [25];**

- 2) определение массива с открытой левой границей

**float A[ ];          int B[ ][4];          char S [ ];**

- 3) определение указателя на первый элемент массива

**float\*pA;          int\*pB;          char\*pS;**

Здесь и далее многомерные массивы будут рассматриваться на примере двумерного массива.

При всех этих определениях формального параметра в стеке выделяется оперативная память на один указатель для передачи в функцию адреса нулевого элемента массива – фактического параметра.

Массив, адрес которого будет использован при вызове функции, как фактический параметр, может быть изменен за счет выполнения операторов функции.

Если формальный параметр-массив определить как в первом случае, фактическим параметром функции будет имя массива из вызывающей программы той же размерности и с теми же фиксированными границами или же значение указателя на первый элемент такого фиксированного массива.

При использовании второго определения фактическим параметром также будет имя массива из вызывающей программы, но уже с произвольным количеством элементов по первому измерению. Количество элементов по другим измерениям массива должно быть фиксированным, таким же как в определении формального параметра – массива. В этом случае для числовых и многомерных символьных массивов в определении функции нужен дополнительный параметр для передачи размеров массива по первому измерению. Для одномерных символьных массивов, как было отмечено выше, нет необходимости передавать размеры массива.

Если формальный параметр определить как в третьем случае, фактическим параметром будет адрес первого элемента некоторого массива данных того же типа, что и тип указателя.

### Файловый ввод/вывод данных

Информация во внешней памяти сохраняется в виде файлов – именованных участков памяти. Файлы позволяют сохранять информацию при отключении компьютера.

Рассмотрим потоковый ввод/вывод верхнего уровня библиотеки классов.

Важнейшим моментом при операциях ввода/вывода является объявление потоков для обмена данными.

Поток – это обмениваемая последовательность байт. Обмен в данном случае производится между оперативной памятью и внешней памятью - файлом на диске.

Потоки для работы с файлами являются переменными следующих типов (классов):

**ofstream** - это тип выходного файлового потока;

**ifstream** - это тип входного файлового потока;

**fstream** - это тип двунаправленного файлового потока, предназначенного для чтения данных из файла и записи данных в файл.

Описание этих типов находится в файле **<fstream.h>**, который при работе с файлами необходимо подключить к программе директивой **include**.

Объявить файловых потоков:

**ofstream fout;** - выходной файловый поток; в этот поток можно только выводить данные.

**ifstream fin;** - входной файловый поток; из этого потока можно только извлекать (читать) данные.

**fstream fio;** - двунаправленный файловый поток; можно и извлекать данные из потока и помещать данные в поток.

Имена потоков **fout**, **fin**, **fio** – произвольные идентификаторы.

Объявив файловые потоки, нужно присоединить их к конкретным физическим файлам с помощью функции **open ()**.

Функция **open** открывает существующий файл или создает новый файл и связывает его с потоком для обмена данными. Форма вызова функции:

**имя потока.open(имя файла, режим, защита);**

Первый параметр - имя уже существующего или создаваемого заново файла. Это строка, определяющая полное имя файла в формате, регламентированном операционной системой.

Второй параметр определяет режим работы с открываемым файлом, третий определяет защиту файла. В простейшем случае форма вызова будет следующей: **имя потока. open (имя файла);**

Примеры вызовов **open()** для определенных выше потоков:

**fout. open (“A:\\USER\\ RESULT.DAT”);** - открыт новый файл для записи данных;

**fin. open (“DATA.TXT”);** - открывается существующий на диске файл для чтения данных, файл находится в текущем каталоге.

Тип потока определяет направление обмена данными при работе с файлом.

Если при создании файла **A:\\USER\\RESULT.DAT** нет достаточно места на диске **A** вызов функции **fout.open()** приведет к ошибке.

Аналогично завершится неудачей вызов функции **fin.open()** при открытии несуществующего на диске файла **DATA.TXT** для чтения данных.

Для проверки удачности завершения функции **open()** следует проверить значение выражения **!имя потока**. Если значение выражения **!имя потока** равно нулю, то ошибок при вызове функции не было.

Таким образом, поток не должен оставаться нулевым при успешном открытии файлов.

Следующий фрагмент программы позволяет проверить результат выполнения функции **open()**:

**if(!fin) { cout<<” ошибка при открытии файла”<<endl; exit(0);}**

Входные файловые потоки происходят от стандартного входного потока **cin** , а выходные файловые потоки – от стандартного выходного потока **cout**. И операции вставки в поток **>>** и извлечения из потока **<<**, а также все функции для чтения данных из потока и для их вывода в поток файловые потоки наследуют от стандартных потоков.

### **Чтение данных из файла**

Рассмотрим ввод данных из файла с помощью операции извлечения **>>**.

**Ввод данных из файла в ОП:**

**fin >> имя переменной;**

Операция извлечения **>>** может быть использована для ввода целых, вещественных чисел, символов и строк.

*При использовании операции извлечения >> игнорируются ведущие пробелы, и считывание идет до пробела или до первого недопустимого символа.*

Таким образом, если надо прочесть строку символов, содержащую пробелы, то с помощью операции >> это сделать непросто – каждое чтение выполняется до пробела, а ведущие левые пробелы игнорируются.

В этом случае следует использовать альтернативные функции ввода – функции бесформатного двоичного чтения, например, функцию **getline()**.

Функция **getline()** выполняет извлечение последовательности байтов из потока и перенос их в символьный массив.

При выполнении следующих операторов:

**char text [255];            fin.getline(text, n );**

из файлового потока **fin** извлекаются любые символы, включая и пробелы, и заносятся в оперативную память по адресу **text**. Чтение происходит до наступления одного из событий: прочитано **n-1** символов или если ранее появился символ перехода на новую строку ‘\n’ (символ-ограничитель). В последнем случае из потока символ ‘\n’ извлекается, но в память не помещается, а помещается в память символ конца строки ‘\0’. Если из входного потока извлечено ровно (**n-1**) символов, и не встретился символ-ограничитель, то концевой символ (‘\0’) также помещается после введенных символов.

#### **Вывод данных в файл**

Форма оператора вывода (вывод данных из ОП в файл):

**fout<< выражение;**

Из оперативной памяти извлекается значение выражения и помещается в выходной поток **fout**, связанный с файлом. При этом происходит преобразование двоичных кодов типизированного значения выражения (форма хранения значения в оперативной памяти) в последовательность символов алфавита, изображающих значение в текстовом файле.

При применении операций ввода/вывода к файловым потокам по умолчанию устанавливаются стандартные форматы внешнего представления пересылаемых данных. Например, при выводе данные занимают ровно столько позиций, сколько надо для их представления. Форматы представления выводимой информации могут быть изменены, использованием специальных функций для форматирования потоков.

Рассмотрим вызовы некоторые из них.

Чтобы установить ширину поля вывода, используется функция **width()**:

**fout.width(15);**

устанавливает значение ширины поля вывода 15 позиций в соответствии со значением параметра. Значение ширины поля надо устанавливать отдельно для каждого выводимого значения, даже если это одно и то же значение.

Чтобы установить точность вывода вещественных чисел используется функция **precision()**:

**fout.precision (5);**

устанавливает точность представления вещественных чисел в соответствии со значением параметра, т.е. количество цифр дробной части при выводе будет равно 5.

Использование функции **setf()** позволяет установить несколько форматов вывода, рассмотрим лишь два из них:

**fout.setf (ios::left);** вывод данных с левым выравниванием в поле вывода.

**fout.setf (ios::right);** вывод данных с правым выравниванием (это значение устанавливается по умолчанию).

**fout.setf (ios::fixed);** установка формата вывода вещественных чисел с фиксированной точкой.

**fout.setf (ios::scientific);** вывод вещественных чисел в формате с плавающей точкой.

Функция **setf()** может устанавливать несколько форматов одновременно, если при вызове функции нужные параметры связать побитовой логической операцией – дизъюнкцией ('|'):

**fout.setf (ios::scientific | ios::left );**

### Закрытие файла

По окончании работы с файлом или чтобы изменить режим доступа к файлу, его надо закрыть с помощью функции **close()**, а затем при необходимости открыть вновь в нужном режиме.

Вызовы

**fin. close();          fout.close();**

закроют соответствующие файлы.

## 4.3. Задание на выполнение лабораторной работы

### Дома

1. Проработать материал лекций: Адреса, указатели, ссылки; Индексированные данные - массивы; Функции. Функции и массивы.

Материал лекций рассмотрен в [1, с. 109-204 ; 2, с. 62- 169].

2. Разработать схемы алгоритмов функций ввода исходных числовых и символьных данных из файла данных в оперативную память и форматного вывода данных в файл результатов в виде таблицы.

3. Разработать схему алгоритма функции формирования массива из элементов массива исходных данных в соответствии с вариантом задания.

### В компьютерном классе

Разработать программу обработки данных с использованием функций, реализующую алгоритмы.

## 4.4. Порядок выполнения работы

1. Сформировать файл с исходными данными (набить с клавиатуры). Сначала в файле должны располагаться символьные строки – строки шапки таблицы. Как и в лабораторной работе №2 для строк таблицы использовать



символы псевдографики. Затем должны располагаться числа – положительные и отрицательные значения, целые, дробные и в виде дроби с мантиссой и порядком.

2. В соответствии с количеством данных в файле определить внешний символьный массив для хранения строк шапки таблицы и арифметический массив для хранения исходных числовых значений. Последний массив можно определить как локальный массив главной функции.

3. Разработать алгоритм и определить функцию ввода данных из файла в оперативную память, выделенную числовому и символьному массивам.

4. Разработать алгоритм и определить функцию вывода данных в файл результатов форматно в виде таблицы чисел. Причем в разные столбцы таблицы числа выводить по-разному: с разной точностью, с фиксированной или с плавающей точкой.

Обе функции должны иметь параметрами числовой массив, границы которого следует задать как внешние константы.

5. Разработать алгоритм и определить функцию формирования арифметического массива из элементов массива с исходными данными в соответствии с вариантом задания.

Функция должна иметь как минимум два параметра-массива: первый - для передачи в функцию массива с исходными данными для обработки и второй - для передачи в вызывающую функцию сформированного массива.

6. В главной функции определить формируемый массив, то есть выделить участок оперативной памяти для хранения элементов массива, представить прототипы и вызовы функций ввода, вывода данных, а также функции формирования нового массива. В главной функции произвести вывод элементов нового массива в файл результатов в виде матрицы.

7. Провести отладку и тестирование программы.

8. Вывести на печать тексты файлов с исходными данными и результатами выполнения программы, а также текст файла программы.

#### 4.5. Пример лабораторной работы №3

##### Задание

1. Составить файл данных, хранящий символьные и числовые данные.

2. Написать программу ввода данных из файла в оперативную память и вывода данных из оперативной памяти в файл результатов форматно в виде таблицы.

3. Написать программу формирования нового массива из массива исходных данных, элементами которого будут произведения элементов каждой строки, попадающих в некоторый интервал (первая строка нового массива), и количества таких элементов в каждой строке исходного массива (вторая строка нового массива). Если нужных элементов в строке нет, соответствующие элементы нового массива следует обнулить.

На рис. 4.1 приведен файл с исходными данными, на рис. 4.2 – файл с результирующими данными.

ИСХОДНЫЕ ДАННЫЕ					
ДАННЫЕ 1	ДАННЫЕ 2	ДАННЫЕ 3	ДАННЫЕ 4	ДАННЫЕ 5	ДАННЫЕ 6
1.34	33.5	312	44	-1.0E-3	46
56.89	4.32	7.e-2	67	450	14.85
67	4	-3.879	3.2	-436	67
0.2	6.4	-44	1.23	0.55	1.2E-3
1.578	0.25	4.5	0.45	-5.44	2.36

Рис. 4.1. – Исходные данные – содержимое файла lr1.dat

#### Текст программы

```
//Лабораторная работа №3 студента группы ЭВМ 1-1 Иванова Петра
//Разработка программы с использованием функций ввода/вывода данных
//числовых и символьных массивов и функции формирования
//арифметического массива
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
#include <iomanip.h>
const int m=5, n=6 ;
ifstream fin ;
ofstream fout;
char c[6][81];
void main()          //определение главной функции
{clrscr();
  fin.open("lr1.dat" );
  if (!fin) {cout<<"Ошибка при открытии файла данных"; exit(0);}
  fout.open("lr1.res" );
  if (!fout) {cout<<"Ошибка при открытии файла результатов"; exit(0);}
  float a[m][n];      //определение числового массива для хранения
                      //исходных данных
  float b[2][m];      //определение формируемого массива
  int i,j;
  void vvod (float a[m][n]);    // прототип функции vvod()-ввода данных
  void p ( float a[m][n] );    // прототип функции p()-вывода данных
  void obr(float a[m][n],float b[2][m], int , int );    // прототип функции obr()
  // вызовы функций:
```

```

vvod(a);
p(a);
obr(a,b, 3, 5);    //3, 5 -границы интервала значений нужных элементов
//вывод в файл результатов сформированного в функции obr() массива b:
fout<<"\n\n\nNew array:\n";
for(i=0;i<2;i++)    // в новом массиве две строки
{ fout<<"\n";        //каждая строка значений массива выводится
                        //с новой строки файла
  for(j=0;j<m;j++)
  fout<<setw(12) <<b[i][j];
}
// setw ( int n) - манипулятор вывода с параметром для
//установки поля вывода выводимого значения,
//прототипы манипуляторов находятся в файле iomanip.h
fout.close();
}
void vvod (float a[m][n])    // определение функции vvod()
{ int i,j;
for(i=0;i<6;i++)
{ fin.getline(c[i], 82,'\n');
  if(i<4) fout<<c[i]<<endl;
}
for(i=0;i<m;i++)
for(j=0;j<n;j++)
fin >> *(( a+i)+j);
fin.close();
}
void p ( float a[m][n] )    // определение функции p()
{ int i,j;
for(i=0;i<m;i++)
{fout<<"\272';
for(j=0;j<n;j++)
switch(j)
{case 0: case 1: case 3:case 4: fout.width(10); fout.setf(ios::left|ios::fixed);
fout.precision(5); fout<<a[i][j]<<"\263'; break;
case 2: fout.width(10); fout.setf(ios::scientific); fout.precision(4);
fout<<a[i][j]<<"\263'; break;
case 5: fout.width(10); fout.precision(4); fout<<a[i][j]<<"\272\n";break;
}
if (i==m-1) fout<<c[5]<<endl;
else fout<<c[4]<<endl;
}}
//определение функции obr() – функции формирования массива b

```

```

//первые два параметра это параметры-массивы для передачи в функцию
//массива с исходными данными и формируемого массива
//параметры A и B определяют границы интервала нужных значений
void obr (float a[m][n],float b[2][m], int A, int B)
{ int i, j, k; //k -переменная для подсчета количества нужных элементов
float pr; //переменная для подсчета произведений
for(i=0;i<m;i++) // для каждой строки массива
{ pr=1; // начальные установки для произведения
k=0; // и для количества нужных элементов
for(j=0;j<n;j++) // перебираем элементы столбцов в поисках элементов
if((A < a[i][j]) && (a[i][j] < B)) // если элемент попал в интервал
{pr=pr*a[i][j]; // формируем произведение
k++; // увеличиваем количество
}
if(k==0) //если в данной строке нет нужных элементов,
{b[0][i]=0; b[1][i]=0;} //элементы нового массива обнуляются
else {b[0][i]=pr; b[1][i]=k;} //или им присваиваются значения
}
}

```

ИСХОДНЫЕ ДАННЫЕ					
ДАННЫЕ 1	ДАННЫЕ 2	ДАННЫЕ 3	ДАННЫЕ 4	ДАННЫЕ 5	ДАННЫЕ 6
1.34	33.5	3.12E+02	44	-0.001	46
56.89	4.32	7E-02	67	450	14.85
67	4	-3.879E+00	3.2	-436	67
0.2	6.4	-4.4E+01	1.23	0.55	0.0012
1.578	0.25	4.5E+00	0.45	-5.44	2.36

New array :

0.0000	4.3200	12.8000	0.0000	4.5000
0.0000	1.0000	2.0000	0.0000	1.0000

Рис. 4.2 – Содержимое файла результатов lr1.res

#### 4.6. Контрольные вопросы

1. Что такое функция?
2. Определение, описание и вызов функции.
3. Переменные, используемые в функции.
4. Какими способами можно вернуть из функции результат?
5. Оператор **return**.
6. Что такое формальные и фактические параметры функции?
7. Умалчиваемые значения параметров.

8. Как нужно объявить формальный параметр, если фактическим параметром должно быть выражение?

9. Как нужно объявить формальный параметр, если посредством этого параметра должен быть возвращен скалярный результат выполнения функции?

10. Как нужно объявить формальный параметр, если посредством этого параметра должен быть возвращен массив как результат выполнения функции?

11. Как обращаться в теле функции с формальным параметром–указателем на скалярную переменную?

12. Поясните специфику использования ссылок при работе с функциями.

13. Как происходит обмен данными при передаче параметров по значению, по адресу и по ссылке?

14. Формальные параметры-массивы.

15. Основные средства отладки программ, использующих функции, разработанные пользователем.

#### 4.7. Варианты заданий лабораторной работы

1. а) сформировать двумерный массив из:

- сумм положительных элементов каждой четной строки исходного массива (первая строка нового массива);

- количества таких элементов в каждой четной строке (вторая строка нового массива);

б) определить произведение сумм сформированного массива и общее количество положительных элементов четных строк;

в) определить максимальное и минимальное значения сумм и поменять местами строки исходной матрицы, в которых они найдены, если номера строк разные.

2. Обработка совпадает с вариантом 1, кроме того, что для поиска элементов использовать нечетные столбцы исходной матрицы и в последнем пункте обработки переставлять столбцы исходной матрицы с минимальным и максимальным значением сумм;

3. а) сформировать двумерный массив из:

- произведений отрицательных элементов каждой строки исходной матрицы (первая строка нового массива);

- количества таких элементов в каждой строке (вторая строка нового массива); если таких элементов в строке нет, соответствующие элементы формируемого массива обнуляются;

б) определить сумму произведений сформированного массива и общее количество отрицательных элементов строк;

в) определить максимальное и минимальное значения произведений и поменять местами строки исходной матрицы, в которых они найдены, если номера строк разные.

4. Обработка совпадает с вариантом 3, кроме того, что для поиска элементов использовать нечетные строки исходной матрицы.

5. Обработка совпадает с вариантом 3, кроме того, что для поиска элементов использовать четные столбцы исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальным и максимальным произведением.

6. Задать A и B как параметры функции:

а) сформировать двумерный массив из:

- сумм элементов каждой строки исходной матрицы a, находящихся в пределах:  $A \leq a[i][j] < B$  - (первая строка нового массива);
- количества таких элементов в каждой строке (вторая строка нового массива);

б) определить произведение сумм сформированного массива и общее количество таких элементов строк;

в) определить максимальное и минимальное значения сумм и поменять местами строки исходной матрицы, в которых они найдены, если номера строк разные.

7. Обработка совпадает с вариантом 6, кроме того, что для поиска элементов использовать четные строки исходной матрицы.

8. Обработка совпадает с вариантом 6, кроме того, что для поиска элементов использовать каждый столбец исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальной и максимальной суммой.

9. Обработка совпадает с вариантом 6, кроме того, что для поиска элементов использовать нечетные столбцы исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальной и максимальной суммой.

10. Задать A и B как параметры функции:

а) сформировать двумерный массив из:

- произведений элементов каждого столбца исходной матрицы a, находящихся в пределах:  $A < a[i][j] \leq B$  - (первая строка нового массива);
- количества таких элементов в каждом столбце - (вторая строка нового массива);

б) определить сумму произведений сформированного массива и общее количество таких элементов столбцов;

в) определить максимальное и минимальное значения произведений и поменять местами столбцы исходной матрицы, в которых они найдены, если номера столбцов разные.

11. Обработка совпадает с вариантом 10, кроме того, что для поиска элементов использовать нечетные строки исходной матрицы и в последнем пункте обработки поменять местами строки исходной матрицы с минимальным и максимальным произведением.

12. Обработка совпадает с вариантом 10, кроме того, что для поиска элементов использовать четные столбцы исходной матрицы.

13. а) сформировать двумерный массив из:

- первых попавшихся отрицательных элементов каждой строки исходного массива (первая строка нового массива);

- их индексов (вторая и третья строки нового массива); если отрицательных элементов в строке нет, соответствующий элемент формируемого массива обнуляется, а в качестве индексов помещаются значения -1;

- б) определить сумму отобранных элементов и их количество;

- в) определить максимальное и минимальное значения из отобранных элементов и их индексы в исходной матрице и поменять местами строки исходной матрицы, в которых они найдены, если номера строк разные.

14. Обработка совпадает с вариантом 13, кроме того, что для поиска элементов использовать четные строки исходной матрицы.

15. Обработка совпадает с вариантом 13, кроме того, что для поиска элементов использовать столбцы исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальным и максимальным элементом.

16. Обработка совпадает с вариантом 13, кроме того, что для поиска элементов использовать нечетные столбцы исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальным и максимальным элементом.

17. Задать A и B как параметры функции:

- а) сформировать двумерный массив из:

- первых попавшихся элементов каждой строки исходной матрицы, находящихся в пределах:  $A \leq a[i][j] \leq B$  - (первая строка нового массива);

- их индексов (вторая и третья строки нового массива); если таких элементов в строке нет, соответствующий элемент формируемого массива обнуляется, а в качестве индексов помещаются значения -1;

- б) определить произведение отобранных элементов и их количество;

- в) определить максимальное и минимальное значения из отобранных элементов и их индексы в исходной матрице и поменять местами столбцы исходной матрицы, в которых они найдены, если номера столбцов разные.

18. Обработка совпадает с вариантом 17, кроме того, что для поиска элементов использовать нечетные строки исходной матрицы.

19. Обработка совпадает с вариантом 17, кроме того, что для поиска элементов использовать столбцы исходной матрицы и в последнем пункте обработки поменять местами строки исходной матрицы с минимальным и максимальным элементом.

20. Обработка совпадает с вариантом 17, кроме того, что для поиска элементов использовать четные столбцы исходной матрицы и в последнем пункте обработки поменять местами строки исходной матрицы с минимальным и максимальным элементом.

21. а) сформировать двумерный массив из:

- минимальных и максимальных элементов каждой строки исходной матрицы (первая строка нового массива);

- их индексов (вторая и третья строки нового массива); поменять местами минимальное и максимальное значение в каждой строке исходной матрицы;

б) определить произведение минимальных значений и сумму максимальных значений;

в) определить максимальное из максимальных значений (максимальный элемент матрицы) и минимальное из минимальных значений (минимальный элемент матрицы) и поменять местами строки исходной матрицы, в которых они найдены, если номера строк разные.

22. Обработка совпадает с вариантом 21, кроме того, что для поиска элементов использовать нечетные строки исходной матрицы.

23. Обработка совпадает с вариантом 21, кроме того, что для поиска элементов использовать столбцы исходной матрицы и в последнем пункте обработки поменять местами столбцы исходной матрицы с минимальным и максимальным элементом.

24. Обработка совпадает с вариантом 21, кроме того, что для поиска элементов использовать четные столбцы исходной матрицы и в последнем пункте обработки поменять местами четные столбцы исходной матрицы с минимальным и максимальным элементом четных столбцов.

25. а) сформировать двумерный массив из:

- двух наименьших элементов каждой строки исходной матрицы (первая строка нового массива);

- их индексов (вторая и третья строки нового массива);

б) определить произведение отрицательных минимальных значений и сумму положительных минимальных значений;

в) определить два наименьших значения исходной матрицы и поменять местами строки исходной матрицы, в которых они найдены, если номера строк - разные.

26. Обработка совпадает с вариантом 25, кроме того, что для поиска элементов использовать нечетные строки исходной матрицы и в последнем пункте обработки два наименьших элемента определять из нечетных строк исходной матрицы и поменять местами строки исходной матрицы, в которых они найдены, если номера строк - разные.

27. Обработка совпадает с вариантом 25, кроме того, что для поиска элементов использовать четные столбцы исходной матрицы и в последнем пункте обработки два наименьших элемента определять из четных столбцов исходной матрицы и поменять местами столбцы исходной матрицы, в которых они найдены, если номера столбцов - разные.

28. а) Сформировать двумерный массив из:

- двух наибольших элементов каждого столбца исходной матрицы (первая строка нового массива);



- их индексов (вторая и третья строки нового массива);

б) определить произведение больших элементов каждого столбца и сумму вторых по величине значений;

в) определить два наибольших значения исходной матрицы и поменять местами столбцы исходной матрицы, в которых они найдены, если номера столбцов – разные;

29. Обработка совпадает с вариантом 28, кроме того, что для поиска элементов использовать нечетные столбцы исходной матрицы и в последнем пункте обработки два наибольших элемента определять из нечетных столбцов исходной матрицы и поменять местами столбцы исходной матрицы, в которых они найдены, если номера столбцов - разные.

30. Обработка совпадает с вариантом 28, кроме того, что для поиска элементов использовать четные строки исходной матрицы и в последнем пункте обработки два наибольших элемента определять из четных строк исходной матрицы и поменять местами строки исходной матрицы, в которых они найдены, если номера строк - разные.

## 5. СПИСОК ЛИТЕРАТУРЫ

1. Подбельский В.В. Язык C++. - М: Финансы и статистика , 1999.
2. Климова Л.М. Основы практического программирования на языке C++. - М: Приор, 1999.

## СОДЕРЖАНИЕ

1. Введение .....	3
2. Лабораторная работа № 1	
Вычисление выражений с использованием алгоритмов линейной структуры.....	4
2.1. Цель лабораторной работы.....	4
2.2. Теоретические сведения .....	4
2.3. Задание на выполнение лабораторной работы.....	10
2.4. Порядок выполнения работы.....	10
2.5. Пример варианта лабораторной работы	11
2.6. Контрольные вопросы.....	13
3. Лабораторная работа № 2.	
Разработка алгоритмов разветвляющейся и циклической структуры.	
Разработка программ для работы в режиме диалога с пользователем..	13
3.1. Цель лабораторной работы.....	13
3.2. Теоретические сведения.....	14
3.3. Задание на выполнение лабораторной работы .....	21
3.4. Порядок выполнения работы.....	21
3.5. Пример варианта лабораторной работы.....	22
3.6. Контрольные вопросы.....	24
4. Лабораторная работа № 3	
Разработка программ с использованием функций для обработки массивов арифметических и символьных данных.....	24
4.1. Цель лабораторной работы.....	24
4.2. Теоретические сведения.....	25
4.3. Задание на выполнение лабораторной работы .....	34
4.4. Порядок выполнения работы.....	34
4.5. Пример лабораторной работы №3 .....	35
4.6. Контрольные вопросы.....	38
4.7. Варианты заданий лабораторной работы.....	39
5. Список литературы.....	43