На содержание

ГОСУДАРСТВЕННАЯ СЛУЖБА ГРАЖДАНСКОЙ АВИАЦИИ МИНИСТЕРСТВА ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ

Кафедра вычислительных машин, комплексов, систем и сетей

Л.А.Надейкина, Н.И.Черкасова

ПОСОБИЕ

к выполнению лабораторных работ № 1,2

по дисциплине

«АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ»

для студентов I курса

специальности 22.01.00

дневного обучения

Москва – 2001

1. ВВЕДЕНИЕ

1.1. Организация выполнения лабораторных работ

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

Для разработки программ используется система программирования Borland C++3.1.

Продолжительность каждой лабораторной работы 4 учебных часа.

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 создает тексты файлов программы с помощью текстового редактора интегрированной среды, проводит отладку и тестирование программы, представляет преподавателю результаты отладки и выполнения программы, выводит на печать текстовые файлы программы, файлы с исходными данными и результатами выполнения программы;
3) *защита работы* – производится при наличии оформленного отчета по лабораторной работе, студент защищает алгоритм задания и текст программы, отвечает на вопросы по всем видам работы на ПК во время отладки программы, а также на ряд контрольных теоретических вопросов.

Каждый этап выполнения отмечается в журнале преподавателя. 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.

1.2. 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++ 3.1:

- 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

Часть 1

В основе решения любой задачи лежит понятие алгоритма. Понятие алгоритма является основным при составлении программ для ЭВМ.

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 ГОСТом [1].

Разработка алгоритма – один из самых трудных этапов решения задачи. Сравнительно простой является алгоритмизация линейных процессов вычислений. 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. У оператора нет значения. 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.

Ниже будут рассмотрены основные операторы языка, используемых при программировании алгоритмов линейной структуры.

Но сначала рассмотрим структуру однофайловой программы на языке С++.

Структура программы на С++.

Программа на С++ 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 по возможности 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
// тело функции
{
  объявления локальных (внутренних) объектов функции main;
  исполняемые операторы функции main;
 }
```

4) Как правило, процесс разработки программы сводится к разбиению задачи на ряд фрагментов, выполняющих некоторую законченную обработку данных, которые оформляются в виде функций. Все функции С++ внешние. Объявление неглавных функций:

тип результата имя_функции (список формальных параметров)
//тело функции
{ … // телом функции является блок – последовательность описаний,

```
        //  определений и операторов
    }
```

5) Комментарии – это последовательность символов, заключенная в скобки **/ * … */**, или строка символов, начинающаяся символами **//.** 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. А с помощью *оператора присваивания* переменные получают новые значения.

Структура оператора присваивания:

L–значение = выражение;

L-значение - это все возможные формы обращения к именованной области оперативной памяти (ОП), значение которой доступно изменениям. Имя переменной – частный случай L- значения. При начальном изучении будем пользоваться следующей формой оператора присваивания:

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. Переменные, входящие в выражение правой части оператора присваивания должны иметь значения к моменту выполнения оператора.

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;  …
```

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 С++ операции разбиты на 16 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*.

Рассмотрим арифметические и логические выражения.

Арифметическое выражение аналогично алгебраическому выражению математики.

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

Операции: **+ , - , * , / , % , = , *op* = , ++ , - -** (здесь ***op*** одна из операций + , - ,* , / , %) .

Рассмотрим более подробно *операцию присваивания*.

Символ **‘=’** в языке С++ означает бинарную операцию, у которой в выражении должно быть два операнда: левый – переменная и правый – выражение.

имя_ переменной = выражение

Если **x** – имя переменной, то

x= 2.5+5.2

есть выражение со значением **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 то же значение:

cout << (x=2.5+5.2) ;

Когда в конце выражения с операцией присваивания помещен символ **‘;’** , это выражение становится оператором присваивания. То есть,

x= 2.5+5.2 ;

есть оператор простого присваивания переменной **x** значения **7.7.** Значения оператор не имеет.

В языке С++ 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 op= выражение ;

где **op** – одна из операций ***, / , % , + , - , &, ^ , | , << , >> .**
Оператор

операнд_1 op= операнд_2;

эквивалентен оператору

операнд_1 = операнд_1 op операнд_2;

Например, операторам

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

эквивалентны такие операторы простого присваивания:

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 функций для арифметических типов параметров представлены в заголовочном файле math.h и подключаются к программе следующей директивой:

#include < math.h>

Примеры встроенных функций:
fabs(x) – модуль вещественного аргумента ,
atan(x) –арктангенс(радианы) ,
sin(x) – синус ,
exp(x) – экспонента и т. д.

В арифметических выражениях операндами могут быть и переменные символьного типа *char*, так как на машинном уровне эти переменные 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 **=1**, если результат операции – ложь, то результат операции **=0**.

Логические выражения используются:

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

К логическим 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;
|| - логическое ИЛИ;
первая операция имеет более высокий приоритет, чем две последние. Операнды любых скалярных типов преобразуются к логическим значениям по правилу: все значения, отличные от нуля трактуются как истина (1), если значение равно нулю, то это 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:

1) 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 (с клавиатуры), а также 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 также 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>

2) 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 библиотеку средств ввода-вывода, построенную на основе механизма классов.

Поток – это обмениваемая последовательность байт. Обмен 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 извлечение 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.

Форма оператора 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 оперативную память, выделенную под переменную.

Но не так все просто. Дело в том, что 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. В отличие от функции **scanf,** при использовании потока **cin** не надо указывать правила преобразования данных.

Форма оператора 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. Интерпретация выводимого значения производится автоматически (в отличие от функции **printf**).

Пример программы линейной структуры

```
//расчет площади круга
   #include <iostream.h>          // директива подключения библиотеки
 // #define pi   3.1416           //директива определения константы
   float r ;                      // объявление глобальной переменной
   const  float   pi = 3.1416 ;   // объявление глобальной константы
   void main ( )                  // заголовок функции main
   {
    float s;                      // объявление локальной переменной
    // операторы:
    cout<<"Введите радиус окружности";  // оператор вывода данных
    cin>> r;                      // оператор ввода данных с клавиатуры
    s = pi*r*r;                   //оператор присваивания
    cout<<"Площадь круга равна :" << s;  //оператор вывода данных
  }
```

Часть 2

Работа в интегрированной среде Borland C++3.1

Система **Borland C++** представляет собой компилятор языка с интегрированной интерактивной средой разработки программ. "Среда" предоставляет сервисные средства, которые не относятся к языку программирования, а служат для облегчения процесса разработки и отладки программ.

Для вызова системы Borland C++ нужно войти в каталог, в который загружена система (напр., **BC** или **BORLANDC** и т. п.) , найти в его подкаталоге **BIN** файл **bc.exe** и запустить его на выполнение.

После успешного вызова системы Borland C++3.1 экран примет вид, приведенный на рисунке 2.1.

Первая строка окна – строка основного меню, под основным меню находится окно редактора с текстом программы. Нижняя строка экрана – строка состояния, содержит список команд и соответствующих им клавиш, которые используются наиболее часто при работе с системой в данной ситуации. Это клавиши быстрого управления, перечень "горячих" клавиш приведен, напр., в [1, с.343].

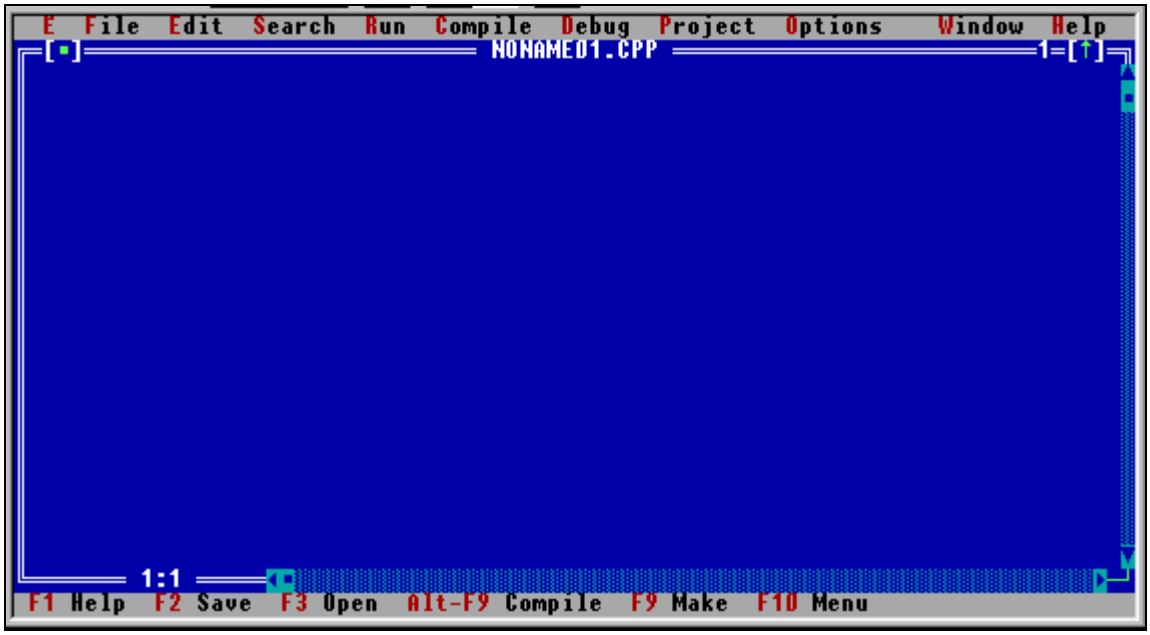

Рисунок 2.1 - Вид экрана после вызова системы

В различных ситуациях на основном окне могут появиться другие окна, напр.: окно просмотра, сообщений, помощи, компиляции, результатов работы программы, и т. д. Одновременно на экране можно видеть несколько окон, но в каждый момент времени активным может быть только одно окно. Активное окно имеет двойные линии рамок и всегда находится поверх других.

В процессе разработки программы используются окна: редактора, 2 отладочных (**Message -** окно сообщений и **Watch** - окно просмотра), диалоговые и экран пользователя с результатами работы программы.

Окна диалога широко используются в среде для просмотра и задания различных режимов работы или необходимых параметров. На рисунке 2.2 в качестве примера приведено окно диалога (окно открывается командой меню **File|Open),** с помощью которого можно выбрать существующий файл.

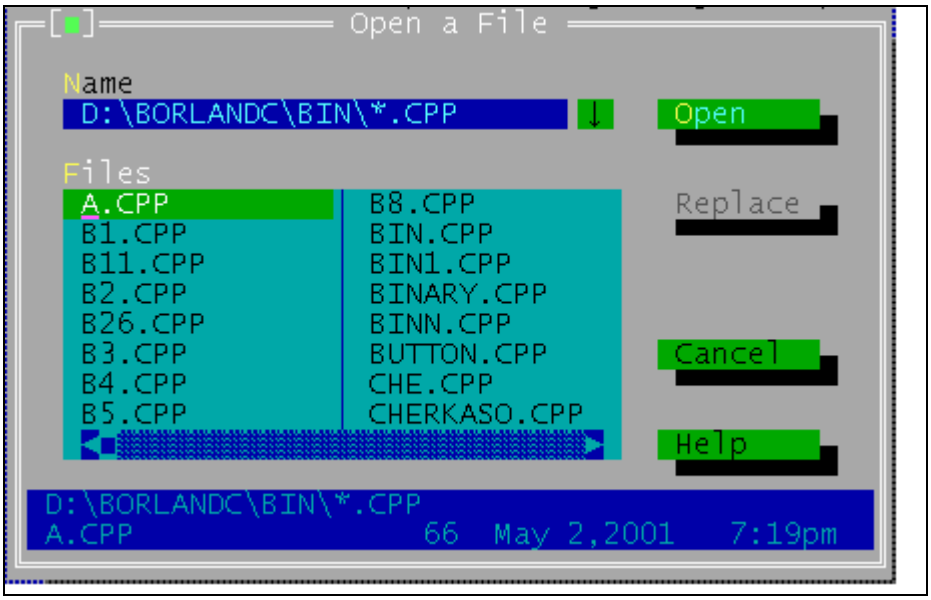

Рисунок 2.2 – Окно диалога для выбора файла

Окна диалога обычно содержат следующие элементы: кнопки управления, строки ввода, списки выбора, списки выключателей, списки переключателей, информационная часть и т.д.

Система меню Borland C++

Все управление в системе Borland C++ осуществляется в основном с помощью системы последовательно разворачивающихся меню.

Переход в основное меню из любого окна осуществляется либо щелчком мыши по соответствующему пункту меню или нажатием **F10**. При этом один из пунктов меню будет выделен, переход к другим пунктам осуществляется с помощью мыши или клавиш горизонтального перемещения курсора ←, →. После выбора пункта меню нажатие клавиши **Enter** , или щелчок мыши на соответствующем пункте меню вызовет разворачивание дополнительного меню команд данного пункта.

Далее при описании команд меню в скобках даны горячие клавиши вызова команд для оперативного управления средой.

Меню **File**. Операции с файлами.

Меню позволяет управлять файлами и выйти из системы.

Подменю **File** позволяет выполнять следующие операции с файлами:

1) **New** – создание нового файла; очищает память редактора и в редакторе создается новый файл с именем **NONAME.CPP (**этот режим изображен на рисунке 2.1);
2) **Open (F3)** – загрузка файла с магнитного диска (МД); открывается окно диалога для выбора файла и вывода его в окно редактора (окно изображено на рисунке 2.2);
3) **Save (F2)** – запоминание (запись) файла, отображенного в окне редактора, с прежним именем на МД (имя высвечивается в верхней рамке окна редактора), если к моменту записи имя файла было **NONAME.CPP**, редактор предложит переименовать файл;
4) **Save as** – создает копию файла с новым именем; открывается диалоговое окно, в которое надо поместить новое имя файла;
5) **Change dir** – смена текущего каталога;
6) **Print** – вывод файла на печать;
7) **Quit (Alt+X)** – выход из системы.

Меню **Edit**. Редактирование файлов.

Меню позволяет выполнять ряд операций над блоками текста файла программы. Блок – это прямоугольный фрагмент текста, выделенный цветом.

Для выделения можно использовать клавишу **Shift** и одновременно клавиши управления курсором или нажать левую клавишу мыши и, не отпуская ее, выделить нужный фрагмент.

Подменю включает следующие операции:

1) **Undo (Alt+BkSp)** – отменяет действие последней команды редактирования экрана;
2) **Redo (Shift+Alt+BkSp)** – повторяет действие последней команды редактирования;
3) **Cut (Shift+Del)**–удаляет блок и заносит его в текстовой буфер **clipboard**;
4) **Copy (Ctrl+Ins)** – копирует блок в текстовой буфер без удаления текста;

   после операций 3) и 4) в буфере текст остается выделенным и его можно вставить в любой файл, причем многократно.
5) **Paste (Shift+Ins) –** вставляет выделенный фрагмент из буфера**,** в указанное курсором место;
6) **Clear (Ctrl+Del) –** стирает выделенный блок, не занося его в буфер;
7) **Copy example-** копирует выделенный пример из окна подсказки;
8) **Show clipboard –**открывает окно буфера и показывает его содержимое.

Меню **Search**. Поиск и замены в тексте

Команды Меню позволяют решать следующие задачи

Find - поиск заданного текста, **Replace** - поиск и замену текста новым, **Locate function -** поиск местонахождения функций. Команды вызывают соответствующие окна диалога, в которые можно задавать объекты поиска (и замены) и параметры для поиска.

Команда **Search again (Ctrl+L)** повторяет действия последней команды **Find** или **Replace.**

Команды **Previous error (Alt+F7)** и **Next error (Alt+F8)** производят поиск места ошибок в тексте программы, обнаруженных во время компиляции, высвечивают строку в окне сообщения **Message** и помещают курсор в окне редактора к оператору, вызвавшему предыдущее или следующее сообщение об ошибке на этапе компиляции.

Команда **Go to line number** переводит курсор в строку, номер которой вводится в окне диалога, которое разворачивается при вызове команды.

Меню **Run**. Компиляция, компоновка и выполнение программы.

Меню позволяет решать следующие задачи:

1) **Run (Ctrl+F9)** – компиляция, компоновка и выполнение программы;

   - при выполнении программы и выводе данных на экран осуществляется переход из окна редактора в окно вывода результатов;

- прервать выполнение программы можно с помощью команды **Ctrl+Break;**

- при возникновении ошибки на любом этапе обработки программы активизируется окно сообщений и высвечивается первая строка окна с сообщением об ошибке;

- при возникновении ошибки на этапе выполнения программы выполнение прекращается, управление передается редактору и курсор устанавливается под оператором, который вызвал ошибку, а в окно результатов передается сообщение об ошибке;

- при нормальном завершении программы управление передается редактору;

2) **Program reset (Ctrl+F2) -** завершает сеанс отладки программы, возвращает программы в исходное состояние, закрывает все открытые в ней файлы;
3) **Go to cursor (F4)** – выполнение программы до строки, в которой располагается курсор;
4) **Trace into (F7) -** производит пошаговое выполнение программы, всех ее функций, включая главную и вызываемые функции, исключая библиотечные функции; при однократном вызове команды выполняется один оператор текущей строки, после чего выполнение программы приостанавливается, затем можно снова выполнить команду **Trace into** и т.д.
5) **Step over (F8) -** работает также как и предыдущая команда, но вызываемые функции выполняют как один оператор, в пошаговом режиме выполняются только операторы текста текущей функции, загруженной в редактор;
6) **Arguments -** позволяет передать в главную функцию **main** программы параметры **argv** и **argc** как это происходило бы при запуске программы из среды **MS-DOS** с передачей параметров программе; при вызове команды появляется диалоговое окно, позволяющее ввести строку параметров выполняемой программы.

Меню **Compile.** Компиляция программы.

Меню **Compile** реализует следующие виды обработки программы:

1) **Compile (Alt+F9) –** компилирует программу и из файла типа **.cpp**, находящегося в активном окне редактора, формирует объектный модуль с программным кодом -файл типа **.obj** с тем же основным именем.

После завершения компиляции выводится окно с сообщением о завершении компиляции, в которое выводится количество ошибок (**Errors –** ошибки**)** и некорректных конструкций

(**Warnings –** предупреждения) в синтаксисе программы; если компиляция прошла успешно, то количества ошибок и предупреждений выводятся равными нулю, а также выводится фраза:

Success: Press any key

что означает - Успех: Нажмите любую клавишу. Вид окна дан на рисунке 2.3.

При обнаружении ошибок или предупреждений на этапе компиляции формируется окно **Message** над окном редактора, с сообщениями об ошибках и предупреждениях. Если последовательно выделять строки сообщений об ошибках в окне **Message**, в окне редактора выделяются соответствующие ошибочные строки текста программы; попеременно активизируя окно сообщений и окно редактора, можно производить коррекцию текста программы.

Просмотреть сообщения можно также из окна редактора с помощью команд **Alt+F7 (Previous error)** и **Alt+F8 (Next error)**; Пример окна **Message** дан на рисунке 2.4.

2) **Make (F9)** - формирует исполняемый файл программы типа **.exe**, для этого проводится компиляция программы и компоновка программы с помощью редактора связей (**Linker)**. Если тексты отдельных модулей, которые используются основной программой или основным модулем при помощи директивы **#include**, были изменены после последней компиляции, соответствующие модули перекомпилируются, после чего компилируется файл, содержащий основную программу. Если текст включаемого модуля не будет найден, система использует существующий **obj-**файл без контроля изменения. Основная часть имени файла с исполняемой программой совпадает с основным именем файла, в котором находится функция **main**, а расширение имени всегда - **.exe**.

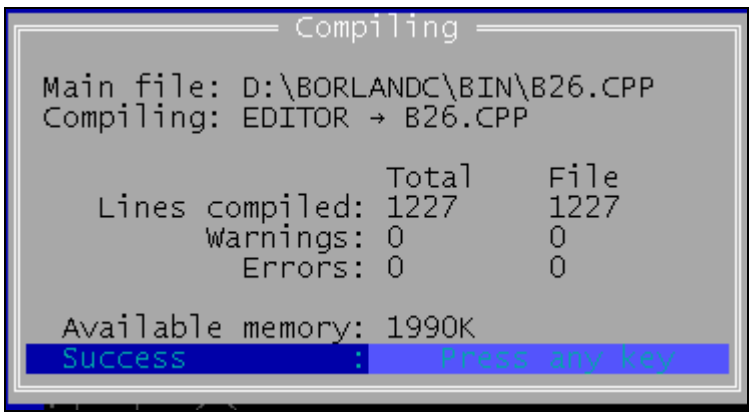

Рисунок 2.3 - Вид окна сообщения о результатах компиляции

Рисунок 2.4 - Вид окна сообщений **Message** об ошибках и предупреждениях

3) **Linker –** создает из объектных модулей (файлы типа **.obj**) файлы выполняемую программу (файл типа **.exe**).
   Команда **Linker** в отличие от команды **Make**, всегда использует текущие версии объектных модулей, без контроля их соответствия файлам исходного текста. Команду **Linker** надо использовать после формирования объектных модулей с помощью команды **Compile**.
4) **Build all –** работает аналогично команде **Make,** только все модули, которые используются основной программой перекомпилируются в любом случае, независимо от того, были ли в них изменения или нет.
5) **Information -** выводит окно диалога с информацией о текущем состоянии системы.
6) **Remove messages** - удаляет все сообщения из окна **Message**.

Меню **Debug.** Средства отладки программы.

Команды меню **Debug** позволяют решать следующие задачи по отладке программы:

1) **Evaluate/modify (Ctrl+F4) –** вычисляет значение выражения или переменной в процессе отладки программы. По этой команде разворачивается окно, позволяющее задать переменную или выражение (поле **Expression**), для которого следует вычислить значение. Результат выводится в поле **Result.** Третье поле окна – **New Value** позволяет задать новое значение переменной. Можно получить немедленное значение переменной (выражения) в поле **Result**, нажав на клавишу **Enter.** Можно получить значение, запустив программу в отладочном режиме до точки останова. Можно следить за изменением значения при пошаговом выполнении программы и т.д.
2) **Call stack (Ctrl+F3) –** позволяет в любой точке, в которой выполнение программы приостановлено просмотреть стек, а именно цепочку вызовов активных функций, их имена и списки фактических параметров. По этой команде открывается окно, в котором демонстрируется стек - вертикальный список вызовов функций, причем

самая нижняя в списке – это функция **main**, которая естественно была вызвана первой. В самом верху находится функция, которая вызвана последней и выполняется в момент вызова команды.

3) Команда **Watches –** открывает подменю из четырех команд:

- **Add watch (Ctrl+F7) –** добавляет выражение в диалоговое окно просмотра **Watch.** Открывается диалоговое окно, в котором имеется поле для ввода выражения (или переменной), значение которого будет контролироваться в процессе отладки. За одно обращение к команде вводится одно выражение, последовательно обращаясь к команде можно задать несколько переменных и выражений, значения которых нужно контролировать. Указанные переменные и выражения вместе с их текущими значениями будут содержаться в окне отладчика **Watch**. Для просмотра надо выполнять программу построчно или до требуемой точки - до курсора.
- **Delete watch –** удаляет выражение из окна просмотра, удаляет выделенную строку из окна **Watch**.
- **Edit watch –** позволяет редактировать выражение в окне просмотра. По команде открывается диалоговое окно, содержащее текст текущего выражения (выделенного в окне просмотра), который можно изменять и дополнять.
- **Remove all watches –** удаляет все переменные и выражения из окна просмотра **Watch**.

4) **Toggle breakpoint (Ctrl+F8)** - устанавливает или снимает точку прерывания программы (точки останова) в текущей строке, т.е. в строке с курсором в окне редактора. Прерывание программы происходит на первом операторе этой строки.

5) **Breakpoints** – открывает диалоговое окно, в котором показаны все точки останова в виде таблицы в составе:

список точек останова,
номер строки, в которой установлена данная точка,
условие останова,
количество прохождений программы через эту точку до останова.

Меню **Project.** Управление проектом.

Меню **Project** предназначено для формирования и редактирования проекта многофайловой программы, то есть программы, состоящей из модулей, расположенных в нескольких программных файлах. Назначение файла проекта – это объединение нескольких исходных объектных файлов в процессе создания многофайловой программы.

Меню **Options.** Управление системой **Borland C++.**

Команды меню **Options,** предназначены для настройки параметров среды. Вызовы команд приводят к разворачиванию соответствующих диалоговых окон. Меню **Options** содержит следующие команды:

1) **Application –** установка параметров приложения: характеристики программы с точки зрения операционной системы.
2) **Compiler –** установка параметров компилятора, напр., установка модели памяти, определение условий выдачи сообщений об ошибках и предупреждениях и т. д.
3) **Make –** устанавливает параметры системы при формировании объектного и выполняемого модуля.
4) **Linker –** установка параметров компоновщика.
5) **Directories -** установка имен каталогов для
   Include Directories – файлов, подключаемых с помощью #include;
   Library Directories – библиотечных файлов;
   Output Directory – выходных файлов;
   Source Directories – исходных программных файлов.
6) **Environment –** устанавливает параметры среды Borland C++, напр., можно определяет количество строк на экране (параметр **Size screen**), с помощью параметра **Auto-save** задать режим автоматического сохранения настроек среды, рабочего стола, проекта, задать параметры текстового редактора, мыши, рабочего стола, цветовые характеристики всех элементов экрана и т. д.
7) **Save –** позволяет установить необходимость запоминания параметров среды, рабочего стола и проекта в соответствующих файлах.

Более детально назначение каждой команды меню **Options** целесообразно осваивать самостоятельно по мере изучения языка С++.

Меню **Window**. Управление окнами среды.

Подменю **Window** позволяет выполнять следующие операции с окнами среды:

1) **Size/Move (Ctrl+F5) -**позволяет перемещать и изменять размеры окна: для перемещения окна надо выполнить команду **Ctrl+F5** и затем использовать клавиши ↑ и ↓, а для изменения размеров после команды **Ctrl+F5** надо нажать **Shift** и изменять размер окна с помощью тех же клавиш.
2) **Zoom (F5) –** распахивает окно на весь экран, повторное нажатие возвращает окну исходный размер.
3) **Cascade –** располагает окна каскадом.
4) **Tile –** располагает окна “мозаикой”.
5) **Next (F6) –**активизирует следующее окно.

6) Команда **Close (Alt+F3) –** закрывает окно. Команда **Close all –** закрывает все открытые окна.
7) Команды **Message (Сообщения), Output (Вывод), Watch (Просмотр), User screen (Alt+F5 - Окно пользователя), Project (Проект)** делают активными соответствующие окна.
8) **List all (Alt+0) –** открывает окно **Window list**, которое содержит список всех открытых окон с номерами и именами файлов и имена файлов, которые использовались в сеансе работы со средой, но окна которых уже закрыты.

Меню **Help.** Справочная система.

Меню **Help** содержит перечень разделов информационной помощи по всем аспектам системы Borland C++ и интегрированной среды:

1) **Contents –** содержание, открывает окно **Help**, содержащее перечень тем справочной системы. Выбрав нужный раздел, с помощью клавиши **Enter** можно перейти к следующему уровню подсказки.
2) **Index (Shift+F1) -** открывает окно **Help**, в котором дан перечень ключевых слов, названий директив, констант, операций, имен встроенных функций – всех объектов языка и системы, расположенных в алфавитном порядке.
3) **Topic search (Ctrl+F1) -** дает подсказки по операторам языка. Надо подвести курсор к нужному оператору и вызвать команду.
4) **Previous topic –** открывает окно предыдущей подсказки.
5) **Help on help -** открывает окно **Help** с правилами использования справочной системы.
6) **Active file –** позволяет выбрать один из файлов справочной системы:
   - IDE and C++Language – среда и язык C++;
   - Windows API – прикладной интерфейс Windows;
   - Object Windows API – объекты прикладного интерфейса Windows;
   - Turbo Vision API – прикладной интерфейс Turbo Vision.

Разработка текстов программ производится с помощью текстового редактора, который запускается при активизации любого из окон редактирования. В обычном режиме редактор позволяет выполнять большое количество команд, которые условно можно разбить на 5 групп: команды перемещения курсора, команды поиска фрагментов, команды вставки и удаления информации, команды работы с блоками информации и команды разнообразного назначения. Подробно команды редактора рассмотрены в [3,с.375-384].

2.3. Задание на выполнение лабораторной работы

Дома:
1) Проработать материал лекций: 2- Подготовка и решение задач на ЭВМ; 3 – Алгоритмы; 4 - Структура языка и программы на языке С++; 5 - Лексические основы языка. Операции С++; 6 - Концепция данных языка С++. Выражения. Материал лекций рассмотрен в [2, с.7-53 ; 3, с.5- 40].
2) Разработать алгоритм задачи вычисления арифметического и логического выражения согласно варианту;
В классе:
Разработать программу в соответствии с алгоритмом, используя, оператор присваивания и операторы ввода- вывода данных.

2.4. Порядок выполнения работы

1) Загрузить систему Borland C++3.1 (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;**
   - 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 значения в выходной поток **cout<<выражение;**
   - ввод с клавиатуры значений координат точки (x,y), используя функцию форматного ввода **scanf()** и поток **cin;**
   - 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) Вывести на печать текст программы из среды Borland C++ 3.1. (**File|Print).**
6) Вывести на печать с экрана результаты тестирования программы с помощью команды **Shift-PrtSc**.
7) Составить отчет.

2.5. Пример варианта лабораторной работы

Задание:

1) Дана формула для вычисления переменной z :

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

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 такого, что значение выражения – истина, если координаты точки попадают в затемненную область фигуры на рисунке 2.5.:

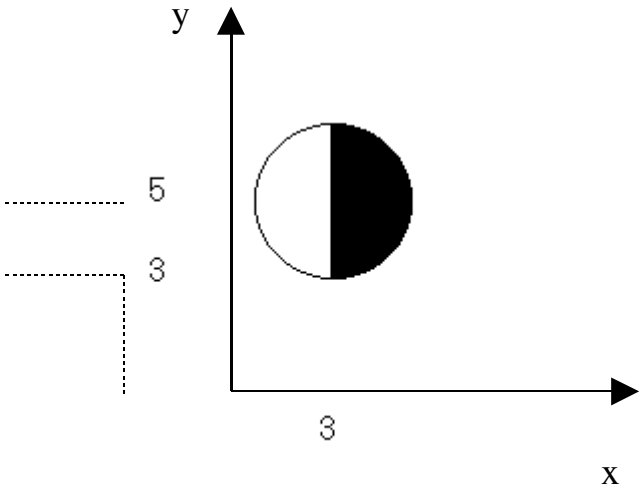

Рисунок 2.5 – Фигура для вычисления логического выражения

Значения координат точки x и y вводить с клавиатуры.

Ниже представлен текст программы примера:

```
//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:\nz="<<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:\nz=" <<z;
cout<<"\n\nРезультат с помощью выражения в операторе вывода :\nz="
<< (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:\nz=%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("\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 С++?
8) Целочисленные и вещественные константы .
9) Символьные константы и строки
10) Объявление констант и переменных в программе.
11) Операции С++. Приоритет операций .
12) Что определяет тип данных?
13) Каковы основные характеристики простых типов С++?
14) Каково назначение выражений, и из каких элементов они формируются?
15) Типы выражений.
16) Арифметические выражения – операнды, операции, примеры встроенных функции, 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?
21) Как отпечатать результаты отображенные на экране?

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:

1) организации диалога с пользователем с использованием алгоритмов разветвляющейся структуры;
2) объявления символьных массивов для хранения текстовых строк и массивов числовых данных (многомерных);
3) ввода данных целого типа и строк с клавиатуры и вывода таких данных на экран;
4) организации обработки числовых массивов с использованием алгоритмов циклической структуры;
5) 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.

Весь массив в целом характеризуется **именем**, и 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);
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**-ому измерениям,
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; // – 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]

– обращение к элементу **i –ой** строки и **j-го** столбца матрицы.

Первый индекс – это индекс строки, второй индекс – индекс столбца. Надо помнить, что нумерация индексов и в многомерных массивах идет от нуля .

Внутреннее представление массива

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

sizeof (type)* количество элементов массива,

где **sizeof(type)–** количество байт, выделяемое под один элемент массива.

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 под массив, можно определить также из следующего выражения:

sizeof (имя массива).

В оперативной памяти все, элементы 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 ⟶

Имя массива и 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 дает 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** воспринимается как **константный указатель,** значением которого является адрес первого элемента массива. Значение данного указателя нельзя изменять.

Рассмотрим одномерный массив. В соответствии с выше сказанным соблюдается равенство:

имя_ массива = = & имя_массива = = & имя_массива[0],

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

В соответствии с операцией сложения указателя с целым числом, если к имени массива 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, т.е. можно записать:

&имя_массива[i] = = имя_массива + i ,

применяя операцию разыменования к адресу можно получить доступ к значению по данному адресу. И соответственно обращаться к **i-му** элементу массива можно одним из эквивалентных способов:

имя_ массива [i] = = * (имя_массива + i)

Многомерные массивы рассмотрим на примере объявления двумерного массива:

type T [m][n]; **// m , n** – некоторые целочисленные константы, **type** – тип
// элемента массива

В массиве **m** строк по **n** элементов в строке **(n** столбцов**).**

Имя двумерного массива **T** отожествляется с его адресом, а также с адресом его первого элемента **T[0][0].**
Адрес любого элемента 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]** (**i** – индекс строки), имя этого массива является адресом первого элемента строки и адресом строки в целом.
Адреса строк массива равны **T[0] , T[1] ,…, T[m-1]** , что эквивалентно, как показано выше следующим выражениям: ***T , *(T+1) ,…, *(T+m-1).**
Таким образом,

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, проводится на этапе компиляции (формировании объектного кода программы).

Явная инициализация массива разрешена только при его определении.

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. Например, так:

int test [4] = { 10, 20 , 30 };

Только три элемента получили начальные значения, а элемент **test[3]** либо не определен, либо имеет нулевое значение, если массив внешний.

В втором случае инициализация - полная, 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, напр., 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) на случай, если массив будет программно заполняться другими значениями:

Байты

0	1	2	3	4	5	6	7	8	9
с	т	р	о	к	а	\0			

Можно объявлять символьный массив без указания его длины:

char stroka1 [ ] = “строка”;

Компилятор, выделяя память под этот массив, сам определит его длину, которая в данном случае будет равна 7 байт (6 байт под собственно строку и один байт под завершающий нуль):

Байты

0	1	2	3	4	5	6
с	т	р	о	к	а	\0

Инициализация двумерных числовых массивов:

int T [3][4]= { {1,2,3,4} , { 10,20,30,40}, {100,200,300,400} };

и символьных массивов:

char name [5] [18] = { “ Иванов” ,“Петров” ,“Розенбаум”,
“Цой”, ”Григорьев”};

При таком объявлении будет выделено памяти по 18 байт на каждую строку, в которые будут записаны символы строки и в конце добавлен байтовый ноль. Ниже показаны строки массива, которые в памяти будут располагаться последовательно одна за другой:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	И	в	а	н	о	в	\0											
1	П	е	т	р	о	в	\0											
2	Р	о	з	е	н	б	а	у	м	\0								
3	Ц	о	й	\0														
4	Г	р	и	г	о	р	ь	е	в	\0								

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

При определении многомерных массивов с инициализаций (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 в оперативной памяти первой, второй и третьей строки массива. В каждую из строк будет записана строковая константа из списка инициализации.

Ввод – вывод элементов массивов

Ввод – вывод числовых массивов

Ввод и 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 рисунке 3.1.

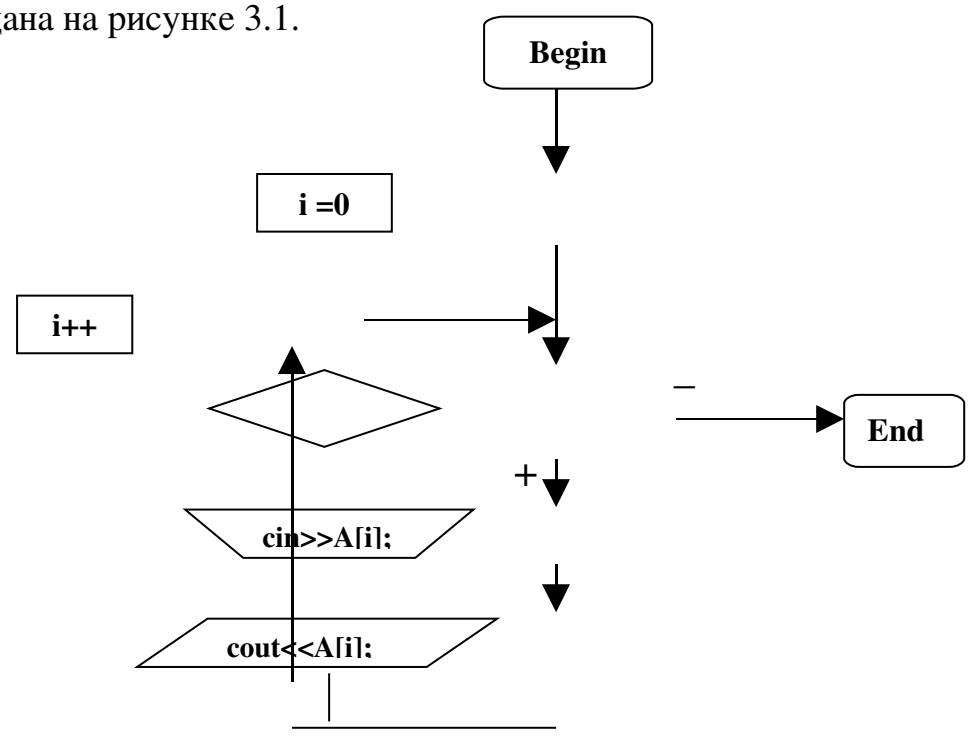

Рисунок 3.1 – Схема алгоритма ввода-вывода элементов одномерного массива.

Поясним схему алгоритма:

1) Устанавливается начальное значение индекса массива (**i= 0**) .

2) Проверяется условия, при котором выполняется обработка массива: значение индекса не должно превышать максимального значения индекса в массиве (**i<100**).
3) 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 опять 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 рисунке 3.2.

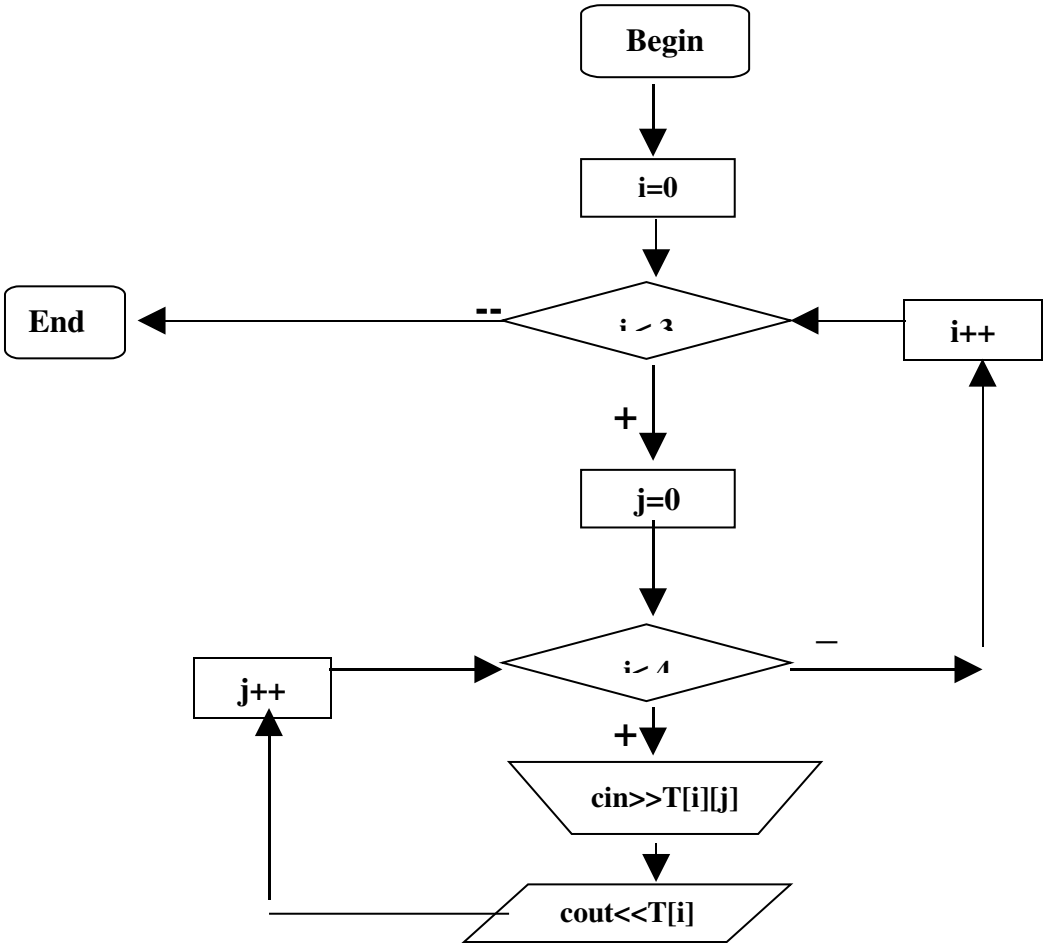

Рисунок 3.2 – Схема алгоритма ввода с клавиатуры значений элементов двумерного массива и вывода значений элементов на экран дисплея.

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.

Однако, обычно в программах символьные массивы используются для ввода и вывода текстовых строк при работе с текстовыми файлами. И в этом случае целесообразно чтобы символьный массив представлял связанную символьную строку, завершающуюся байтовым нулем. В этом случае синтаксис языка С++ допускает 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; -** извлекаются из стандартного входного потока **cin,** по умолчанию связанного с клавиатурой, символы 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.**
   Чтение 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 клавиатуры) и помещается по адресу **text** . Читаются все

символы до первого 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 вводит **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 программы до ввода строки.

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.
4) **puts(text) ; -** выводит строку в стандартный поток (по умолчанию - на экран) и добавляет в завершении символ **'\n' –** перехода на новую строку, т.е. на экране курсор переходит 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:
Дома:
1) Проработать материал лекций: 2- Подготовка и решение задач на ЭВМ; 3 – Алгоритмы; 4 - Структура языка и программы на языке С++; 5 - Лексические основы языка. Операции С++; 6 - Концепция данных языка С++. Выражения. Материал лекций рассмотрен в [2, с.7-53 ; 3, с.5- 40].
2) Разработать алгоритм задачи вычисления арифметического и логического выражения согласно варианту;
В классе:

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

1) Разработать алгоритм и программу диалога с пользователем согласно варианту.
2) Разработать алгоритм и программу обработки массива числовых данных согласно варианту.

3.4. Порядок выполнения работы

1) Разработать алгоритм диалога с пользователем:
   Из программы задаются 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 массив 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, напр., в [2,с.490-491].
   - Вывести на экран элементы массива в таблицу в формате с плавающей точкой и с фиксированной точкой в зависимости от номера столбца матрицы. 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 < =100, то
   фраза покупателя : « О!, цена, - это мне подходит. Покупаю!»
   Если 100< цена < =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 сумму элементов массива.

Текст программы представлен на рисунке 3.3.

```
 //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;
 }
}
```

Рисунок 3.3 – Текст программы примера лабораторной работы №2

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 С++.
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. ЛИТЕРАТУРА

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

СОДЕРЖАНИЕ

Стр.

Содержание