ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Л.А. Надейкина

ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие
по выполнению лабораторных работ № 1, 2, 3, 4

для студентов I курса
направления 09.03.01
очной формы обучения

Москва
ИД Академии Жуковского
2021

УДК 004.42
ББК 6ф7.3
Н17

Рецензент:
Черкасова Н.И. – канд. физ.-мат. наук, доцент каф. ВМКСС

Надейкина Л.А.

Н17 Программирование [Текст] : учебно-методическое пособие по выполнению лабораторных работ № 1, 2, 3, 4 / Л.А. Надейкина. – М.: ИД Академии Жуковского, 2021. – 48 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Программирование» по учебному плану для студентов I курса направления 09.03.01 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 23.03.2021 г. и методического совета 23.03.2021 г.

УДК 004.42
ББК 6ф7.3

1. ЛАБОРАТОРНАЯ РАБОТА № 1

Вычисление выражений с использованием алгоритмов линейной структуры.

1.1 Цель лабораторной работы

Целью лабораторной работы является, во-первых, освоение построения алгоритмов линейной структуры для вычисления выражений и, во-вторых, знакомство с основами программирования в системе программирования Microsoft Visual Studio.

1.2 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.

Интегрированная среда разработки

«IDE» от *«Integrated Development Environment»* — это программное обеспечение, которое содержит все необходимые средства для разработки, компиляции, линкинга и отладки кода программы. Для написания программ на языке C++ мы будем использовать Visual Studio от Microsoft (для пользователей Windows).

Во-первых, несмотря на то, что код программ находится в файлах с расширением .cpp, эти файлы добавляются в проект. Проект содержит все необходимые файлы программы, а также сохраняет указанные пользователем настройки IDE. При компиляции программы, проект “указывает” компилятору и линкеру, какие файлы нужно скомпилировать, а какие связать. Стоит отметить, что файлы проекта одной IDE не будут работать в другой IDE, придется создать новый проект.

Во-вторых, есть разные типы проектов. При создании нового проекта, нужно будет выбрать его тип. Все проекты, которые мы будем создавать на лабораторных работах, будут консольного типа. Это означает, что они запускаются в консоли (аналог командной строки). По умолчанию, консольные приложения не имеют графического интерфейса пользователя — GUI (сокр. от «Graphical User Interface») и компилируются в автономные исполняемые файлы. Это идеальный вариант для изучения языка C++, так как он сводит всю сложность к минимуму.

В-третьих, при создании нового проекта большинство IDE автоматически добавят ваш проект в рабочее пространство. Рабочее пространство — это своеобразный контейнер, который может содержать один или несколько связанных проектов. Несмотря на то, что можно добавить несколько проектов в одно рабочее пространство, все же рекомендуется создавать отдельное рабочее пространство для каждой программы. Это намного упрощает работу для новичков.

Традиционно, первой программой на новом языке программирования является всеми известная программа «Hello, world!». Мы не будем нарушать традиции.

Для создания нового проекта в Visual Studio 2019, нужно сначала запустить эту IDE, затем выбрать ***"Файл" > "Создать"> "Проект"*** (рис. 1). Дальше появится диалоговое окно, где нужно будет выбрать ***"Консольное приложение Windows"*** из вкладки ***"Visual C++"*** и нажать ***"OK"*** (рис. 2). Там же нужно указать имя проекта (любое) и его расположение в соответствующих полях.

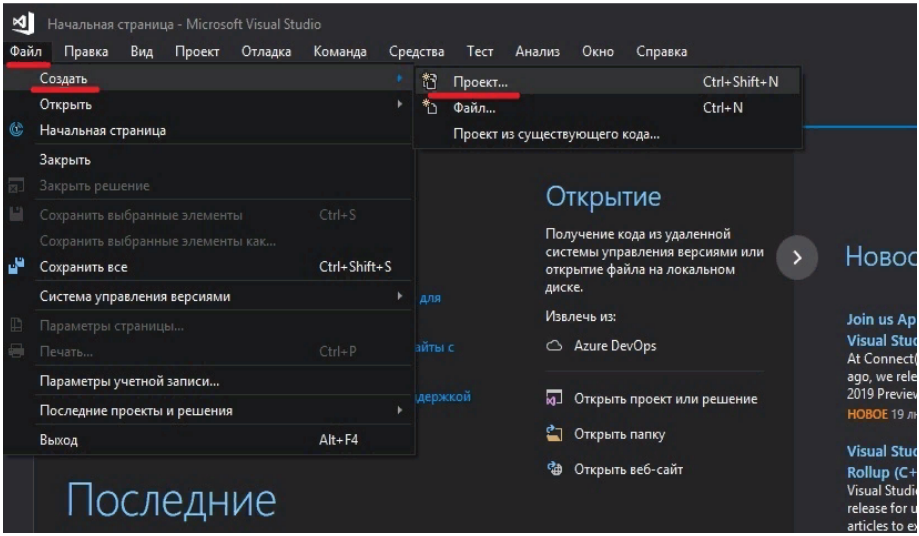


Рисунок 1. Вид первой страницы.

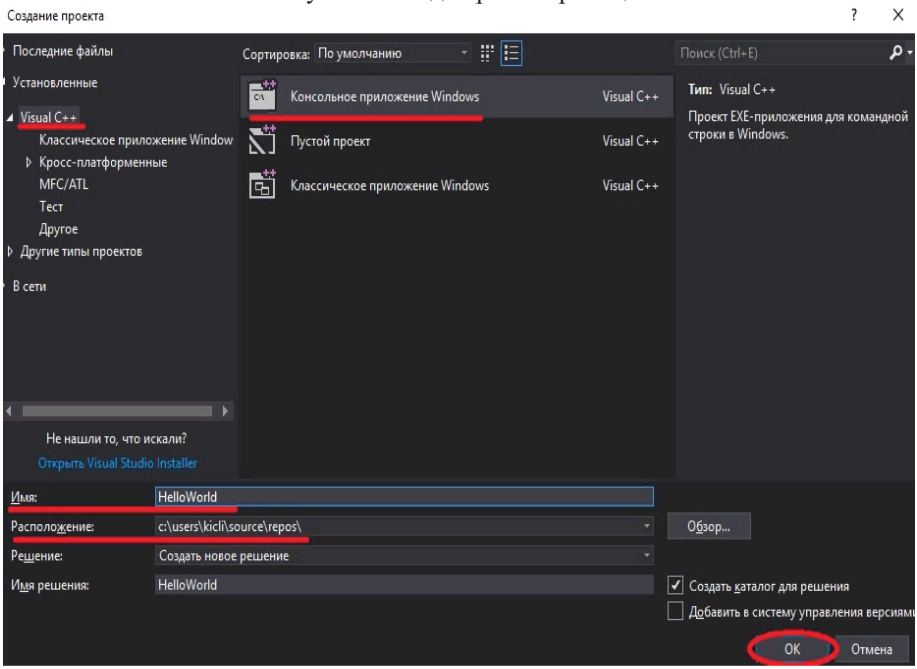


Рисунок 2. Окно выбора типа проекта.

В текстовом редакторе вы увидите, что уже есть некоторый текст и код — его нужно удалить, а затем напечатайте или скопируйте следующий код:

```
#include <iostream>
int main () {
   std::cout<<"Hello, world"<<std::endl;
   return 0;
   }
```

Вот, что должно получиться (рис. 3):

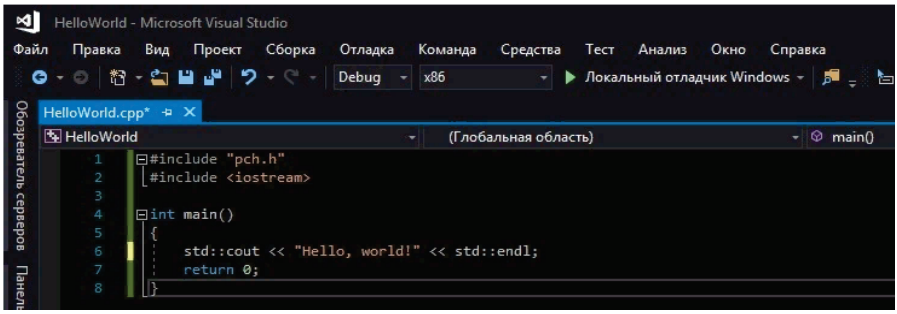
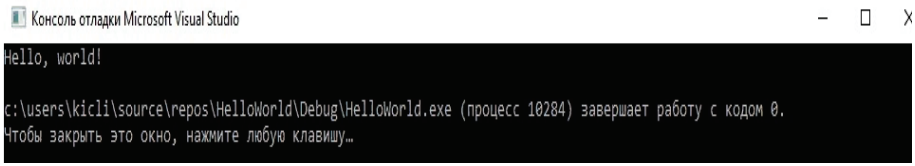


Рисунок 3. Текст программы.

Строка #include "pch.h" требуется только для пользователей Visual Studio 2017. Если вы используете Visual Studio 2019 (или более новую версию), то не нужно писать эту строку вообще.

Запустить программу в Visual Studio можно, нажав комбинацию Ctrl+F5. Если всё хорошо, то увидите следующее:

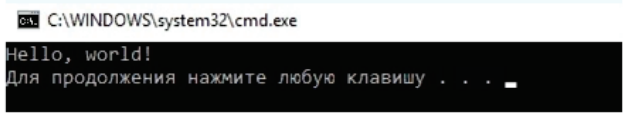


Это означает, что компиляция прошла успешно и результат выполнения программы, следующий:

Hello, world!

Чтобы убрать строку «…завершает работу с кодом 0…», нужно перейти в ***"Отладка" > "Параметры"*** (рис. 4). ***Затем "Отладка" > "Общие"*** и поставить галочку возле ***"Автоматически закрыть консоль при остановке отладки"*** и нажать ***"ОК"*** (рис. 5)

Тогда ваше консольное окно будет выглядеть следующим образом:



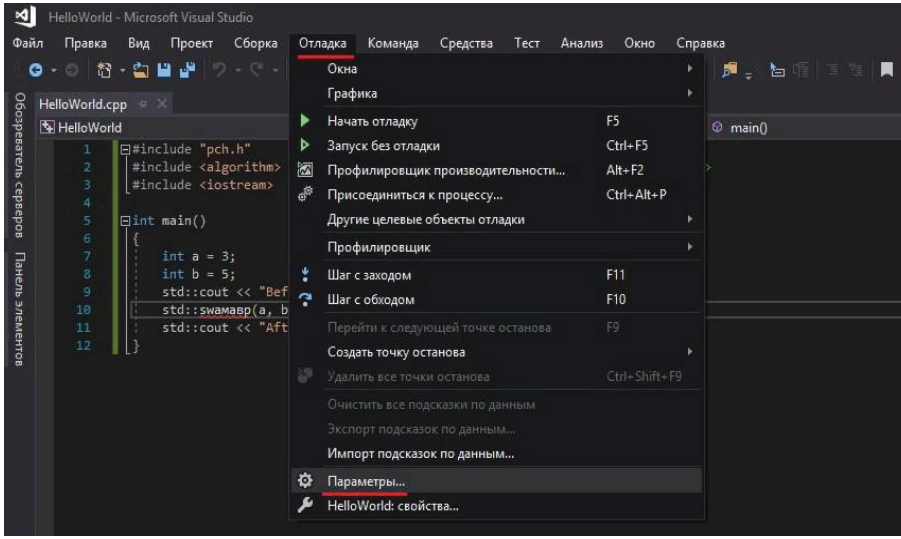


Рисунок 4. Отладка.

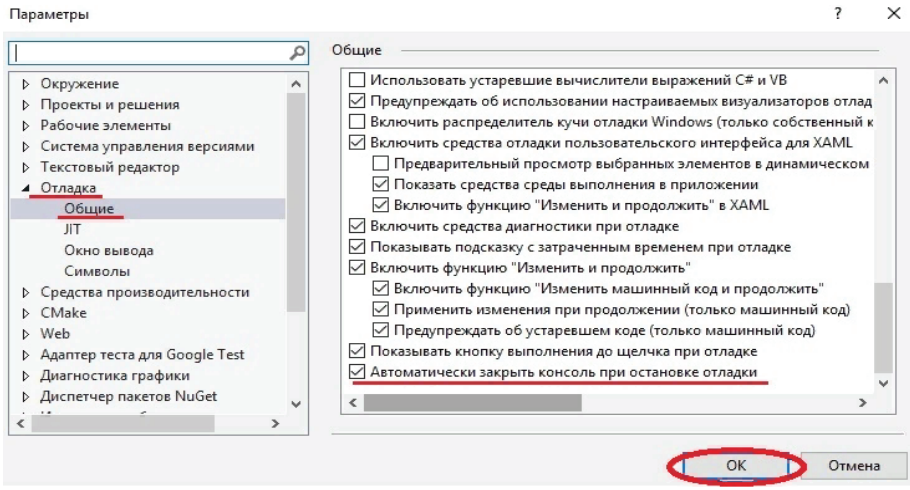


Рисунок 5. Параметры

Режимы конфигурации «Debug» и «Release»

Конфигурация сборки (англ. «*build configuration*») — это набор настроек проекта, которые определяют принцип его построения.

Конфигурация сборки состоит из:

- имени исполняемого файла;
- имени директории исполняемого файла;
- имён директорий, в которых IDE будет искать другой код и файлы библиотек;

– информации об отладке и параметрах оптимизации вашего проекта.

Интегрированная среда разработки имеет две конфигурации сборки: «*Release*» (Релиз) и «*Debug*» (Отладка).

Конфигурация «Debug» предназначена для отладки программы.

Эта конфигурация отключает все настройки по оптимизации, включает информацию об отладке, что делает программы больше и медленнее, но упрощает проведение отладки. Режим *«Debug»* обычно используется в качестве конфигурации по умолчанию.

Конфигурация «Release» используется во время сборки программы для её дальнейшего выпуска. Программа оптимизируется по размеру и производительности и не содержит дополнительную информацию об отладке.

Переключение между режимами «Debug» и «Release» в Visual Studio — выбрать соответствующую из выпадающего списка на панели быстрого доступа (рис. 6):

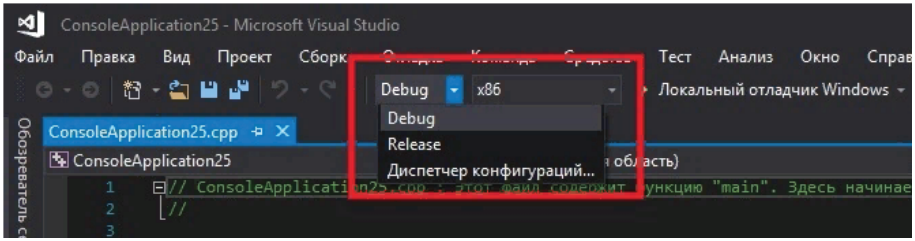


Рисунок 6. Конфигурации сборки проекта

Структура программы на С++.

Процесс разработки программ на С++ предполагает разбиение процесса решения задачи на ряд этапов, выполняющих функционально законченную обработку данных и формирование соответствующих функций.

В результате программа представляет собой совокупность функций, одна из которых главная, называемая ***main***.

Главная функция может располагаться в любом месте программы, но где бы она не находилась выполнение программы начинается и заканчивается именно в главной функции.

Определение любой функции, в том числе и главной в С++ состоит из заголовка и тела функции:

<тип возвращаемого функцией результата> <имя> (список параметров)

{тело функции – последовательность действий функции}

Приведем пример простой программы:

```
#include <iostream>     //директивы
using namespace std;
int main () {cout<<"Программа стартовала"<<endl;
return 0;
}
```

В результате выполнения программы в консольном окне экрана выведется фраза: ***Программа стартовала***.

В первой строке – команда (директива) препроцессора, обеспечивающая включение в программу средств работы со стандартными потоками ввода/вывода данных.

Эти средства подключаются к программе при использовании заголовочного файла с именем ***iostream***.

Стандартным потоком вывода по умолчанию является вывод на экран дисплея. Стандартный поток ввода обеспечивает чтение данных с клавиатуры.

Вторая строка – это заголовок функции ***main***.

В общем случае функция C++ вызывается другой функцией, а заголовок функции описывает интерфейс между ней и той функцией, которая ее вызывает.

Слово, стоящее перед именем, описывает информацию, которую функция передает в вызывающую функцию.

Заголовок главной функции описывает интерфейс между функцией ***main*** и операционной системой.

Стандарт языка C++ требует, чтобы определение функции ***main*** начиналось со следующего заголовка:

int main ()

- слово ***int*** указывает, что функция ***main ()*** возвращает целое значение.

Возвращаемое функцией ***main ()*** значение должно быть равно нулю, если выполнение программы прошло успешно.

Круглые скобки после ***main*** требуются в соответствии с синтаксисом заголовка любой функции.

В них помещается необязательный для главной функции список параметров. В данном примере список пуст.

Тело функции – это заключенная в фигурные скобки последовательность описаний, определений и операторов функции.

В теле данной программы описаний и определений нет, а есть только два оператора.

Первый из них:

cout <<"Программа стартовала"<<endl;

где ***cout*** - имя стандартного выходного потока.

Данные для вывода передаются потоку с помощью операции <<. То, что нужно вывести, помещается от операции << справа. В данном случае это строка – ***"Программа стартовала"***, заключенная в кавычки последовательность символов.

Вслед за строкой помещается еще одна операция вывода <<, а затем манипулятор ***endl*** (сокращение от ***"end of line"*** – "конец строки"). Его роль - очистить буфер выходного потока и поместить в выходной поток символ перехода на новую строку.

Второй оператор в программе ***return 0;*** – оператор возврата. Он завершает выполнение программы и передает управление программой в точку ее вызова, а также значение выражения, стоящего в операторе ***return***.

Так как программа "запускается" на исполнение по команде операционной системы, то возврат будет выполнен к операционной системе.

Простая программа на С++ состоит из следующих элементов:

1) препроцессорные директивы;

2) объявление глобальных объектов программы (типов, переменных, констант);

3) объявление одной главной функции ***main ();***

4) объявление ряда неглавных функций;

5) комментарии.

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 констант и переменных:

const тип_данных имя_ константы = значение;
тип_данных имя переменной;

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

int main () // заголовок функции
{ //тело функции
}

Телом функции является блок – последовательность объявлений локальных (внутренних) объектов функции ***main*** и исполняемых операторов функции, заключенная в фигурные скобки, последний оператор функции - оператор ***return***.

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 символом '//' и заканчивающаяся символом перехода на новую строчку**.** 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 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.

Выражения

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

операнды – это константы, переменные и результаты вызовов функций;

операции – действия над операндами, выполняются в соответствии с приоритетом, в С++ 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 и логические***.

Арифметическое выражение аналогично алгебраическому выражению математики. Операнды арифметических выражений: константы, переменные и результаты функций арифметических типов.

Операции: **+ , - , * , / , % , = , *op* = , ++ , - -.**

Рассмотрим более подробно ***операцию присваивания*.**

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

имя_ переменной = выражение

Если после выражения с операцией присваивания поместить символ ***';'***, это выражение становится ***оператором простого присваивания***.

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;

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 <cmath>***

Логическое выражение – синтаксическая конструкция для определения истинности или ложности какого-либо положения, 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– истина, то значение результата операции равно ***true (1)***, если результат операции – ложь, то значение результата - ***false (0)***.

Рассмотрим логические переменные. ***Логические переменные*** — это переменные, диапазон которых состоит только из двух возможный значений: ***true (1)*** и ***false (0).*** Для объявления переменной используется тип ***bool***:

bool b;

Инициализировать логическую переменную или выполнить операцию присваивания можно с помощью ключевых слов ***true*** или ***false***:

```
bool b1 = true; // копирующая инициализация
bool b2(false); // прямая инициализация
bool b3 {true}; // uniform-инициализация (C++11)
b3 = false; // операция присваивания
```

С помощью операции логического отрицания ***НЕ (!)*** можно изменить ***true*** на false и наоборот (***false*** на ***true***):

```
bool b1 = !true; // значение b1 - false
bool b2 (!false); // значение b2 - true
```

На самом деле, логические значения не сохраняются как ***true*** или ***false***. Они обрабатываются в виде целых чисел: ***true*** — единица, вместо ***false*** — ноль.

Следовательно, при попытке вывести логические значения с помощью ***std::cout***, увидим либо 0, либо 1:

```
#include <iostream>
 int main () {
   std::cout << true << std::endl; // вместо true единица
   std::cout << !true << std::endl; // вместо !true ноль
```

```
   bool b(false);
   std::cout << b << std::endl;   // b - false (0)
   std::cout << !b << std::endl; // !b - true (1)
   return 0;}
```

Результат выполнения программы:

```
1
0
0
1
```

Чтобы в ***std::cout*** выводилось ***true*** или ***false*** (вместо целых чисел), надо использовать манипулятор форматирования потока ***std::boolalpha***:

```
#include <iostream>
int main () {
   std::cout << true << std::endl;
   std::cout << false << std::endl;
   std::cout << std::boolalpha; // выводим логические значения как "true"
                                // или "false"
   std::cout << true << std::endl;
   std::cout << false << std::endl;
   return 0;}
```

Результат выполнения программы:

```
1
0
true
false
```

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 равные нулю, - как ложь (0).

3) *поразрядные (битовые) логические операции*:

~ - битовое инвертирование, **&** поразрядное логическое умножение И; **^ -** поразрядное исключающее ИЛИ; **| -** поразрядное логическое сложение 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, 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), и также первым аргументом - *строка форматов для интерпретации выводимых значений.*

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>*** 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, 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.

Оператор ввода (ввод данных с внешнего устройства в ОП):
cin >> имя переменной;

Из потока **cin** (с клавиатуры) извлекается значение и помещается в участок оперативную память 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**).

1.3 Задание на выполнение лабораторной работы

Вариант задания получить у преподавателя. Разработать алгоритм задачи вычисления арифметического и логического выражения согласно варианту.

1.4 Порядок выполнения работы

1. Создать новый проект в Visual Studio 2019.
2. 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 и защитить отчет.

1.5. Пример варианта лабораторной работы

Задание:

1. Дана формула для вычисления значения выражения:

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

Разработать программы для вычисления значения формулы 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. 7. и – ложь, если нет.

Значения координат точки x и y вводить с клавиатуры.

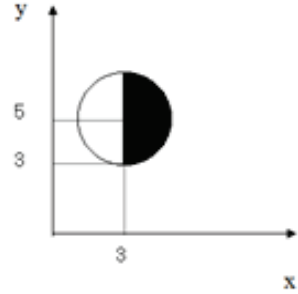


Рисунок 7. – Фигура для вычисления логического выражения

Текст программы

```
//Программирование алгоритмов линейной структуры
//Вычисление выражений
#include <iostream>     //директивы
#include <windows.h>
#include <cmath>        // препроцессора
using namespace std;
const double a = 0.89, b = 7.56;          // определение глобальных констант
//----------------------- Главная функция--------------------------------------
int main () {
SetConsoleCP (1251);  // задаем кодировку для вывода символов на экран
SetConsoleOutputCP (1251); //задаем кодировку для ввода символов с
//клавиатуры
//Вычисление арифметического выражения
double x, y, z, t, q;                    //определение локальных переменных
cout <<"Введите переменные\nx= "; // вывод на экран строковой константы
cin>> x;      cout <<"y="; cin>> y;     //ввод значения с клавиатуры
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 = %
12lf", (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:\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 (-5, 2) <= 4) && (x >= 3))? cout << "- true": cout << "-false";
// Вычисление выражения в операторе присваивания
printf ("\n\nВведите координаты точки\nx= ");
scanf_s (" %lf", &x);       printf("y="); scanf_s (" %lf", &y);
i = ((pow (x - 3, 2) + pow (y - 5, 2) <= 4) && (x >= 3));
printf ("\nЗначение выражения: %d", i); cout << endl;
system("pause");
return 0;}
```

Результат выполнения программы представлен на рис. 8.

```
Введите переменные
x= 8
y=4

Промежуточные переменные:
t=35611
q=20887.9

Результат с промежуточными переменными:
z=1.70486

Результат с помощью одного выражения:
z=1.70486

Результат с помощью выражения в операторе
z=1.70486

Результат с помощью выражения- параметра ф
z =  1.704863e+00

Введите координаты точки
x=3.4
y=5.1


Значение выражения:
1 - true

Введите координаты точки еще раз
x= 2.9
y=5

Значение выражения: 0
Для продолжения нажмите любую клавишу . .
```

Рисунок 8. Результат тестирования программы

1.6. Контрольные вопросы

1) Этапы обработки программы.

2) Что такое трансляция?

3) Структура языка С++.

4) Этапы создания исполняемого кода программы.

5) Структура программы на языке С++.

6) Константы и переменные программы языка С++. Объявление констант и переменных.

7) Какие категории констант имеются в С++?

8) Лексемы языка. Целочисленные и вещественные константы.

9) Лексемы языка. Символьные константы и строки.

10) Операции С++. Приоритет операций.

11) Что определяет тип данных?

12) Каковы основные характеристики простых типов С++?

13) Каково назначение выражений, и из каких элементов они формируются?

14) Арифметические выражения (операнды, операции, вызовы встроенных функций, используемых в арифметических выражениях, порядок вычисления выражений, конструкции языка, в которых используются арифметические выражения).

15) Логические выражения - выражения сравнения, логические выражения с операндами любых типов и логические выражения с битовыми операциями.

16) Что такое операторы программы?

17) Оператор и операция присваивания.

18) Как произвести ввод данных с клавиатуры и вывод данных на экран?

2. ЛАБОРАТОРНАЯ РАБОТА № 2

Разработка алгоритмов разветвляющейся структуры. Разработка программ для работы в режиме диалога с пользователем.

2.1. 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
- ввода/вывода данных числовых типов и символьных строк;
- использования вложенных условных операторов для организации обработки данных.

2.2. Теоретические сведения

Условный оператор

Решение большинства задач не удается описать с помощью алгоритмов линейной структуры. Так в алгоритме ветвления (или развилки), в зависимости от проверки некоторого условия выполняется та или иная последовательность действий. Алгоритм ветвления имеет две формы, представленные на рис. 9.

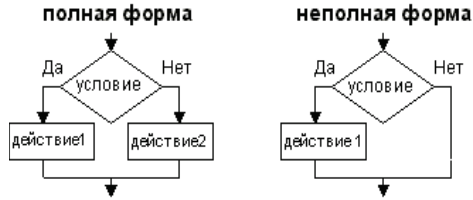


Рисунок 9. Схемы алгоритма ветвления

Условный оператор относится к операторам управления работой программы и реализует алгоритмическую схему развилка и соответственно имеет две формы.

Полная форма: ***if (условное выражение) оператор 1; else оператор 2;***

Неполная форма: ***if (условное выражение) оператор;***

В качестве выражения может быть любое скалярное выражение, которое автоматически приводится к логическому типу. Каждый из операторов - по синтаксису один оператор простой или составной.

Если выражение истинно (то есть его значение не равно нулю), то выполняется ***оператор 1*** (прямая ветвь алгоритма), в противном случае выражение – ложно и выполняется ***оператор 2*** (альтернативная ветвь).

Условный оператор может иметь сокращенную форму. И тогда, если условное выражение истинно, то ***оператор*** выполняется. В случае ложности выражения никаких действий не выполняется.

Вложенные условные операторы

Если ***оператор 1*** или ***оператор 2*** в полной форме или ***оператор*** в сокращенной форме являются также условными операторами, то эти операторы называются вложенными условными операторами, которые также имеют прямую и альтернативную ветвь.

При определении, к какому условному оператору какая относится альтернативная ветвь, существует правило: рассматриваются слева направо каждый ***else***. Очередная альтернативная ветвь ***else*** принадлежит к ближайшему к ней, свободному (не связанному с другим ***else***) оператору ***if***.

Рассмотрим пример:

if (x = =1) if (y = =1) cout << "x =1 и y =1"; else cout<< "x! = 1";

Условный оператор составлен неправильно!

Действительно, при значениях ***x=1*** и ***y! =1***, будет выведена не правильная фраза ***"x! =1"***

Ниже представлены два варианта правильно составленных операторов:

if (x = = 1) {if (y = = 1) cout<< "x =1 и y =1";} else cout<<"x!= 1";

или

if (x= =1) if (y= =1) cout<<" x =1 и y =1 "; else; else cout<<"x! = 1";

Ввод/вывод символьных массивов

Ввод и вывод символьных массивов можно производить поэлементно, то есть рассматривать символьный массив как набор отдельных символов.

Синтаксис языка С++ 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 клавиатуры) 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***;

2.3. Задание на выполнение лабораторной работы

Разработать алгоритм и реализовать программу диалога с пользователем согласно варианту. Вариант получить у преподавателя.

2.4. Порядок выполнения работы

1) Разработать алгоритм диалога с пользователем:

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

2) Создать новый проект в Visual Studio 2019.

3) Написать в файле с расширением cpp текст программы в соответствии с алгоритмом. Программа должна содержать объявление строковых и числовых переменных. В ответ на запрос проводить ввод значений переменных с клавиатуры и анализировать их значения, с использованием вложенных условных операторов, для реализации нужной ветви алгоритма.

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

5) Составить и защитить отчет.

2.5. Пример варианта лабораторной работы

Задание:

Разработать алгоритм и написать программу диалога между тур. оператором и клиентом:

Клиент: <<Здравствуйте, у Вас есть горячие туры?>>

Тур. оператор вводит название тура: <<Да, например, **Кипр>>**

Клиент: <<**Кипр?** Очень интересно! А сколько стоит тур?>>

Тур. оператор вводит цену (цена – целое число)

Если Цена> 500, то

Клиент: <<**Цена** — это очень дорого для меня>>

Если Цена> 200, то

Клиент: <<**Цена** — это дорого для такой короткой поездки!>>

Если Цена> 100, то

Клиент: <<**Цена** - нормальная цена, но у меня не хватает денег. Приду завтра>>

Если Цена <= 100, то Клиент: <<**Цена** — это мне подходит. Покупаю>>

На рис. 10 представлена схема алгоритма диалога между тур. оператором и клиентом.

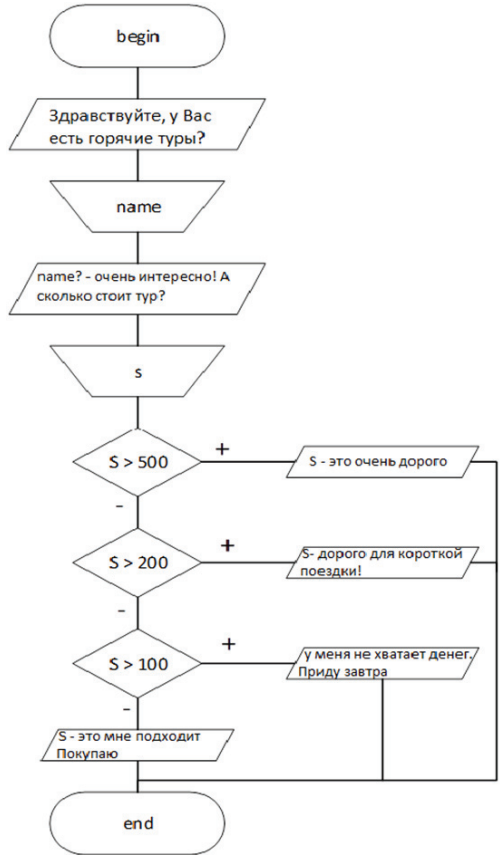


Рисунок 10. Схема алгоритма

Текст программы

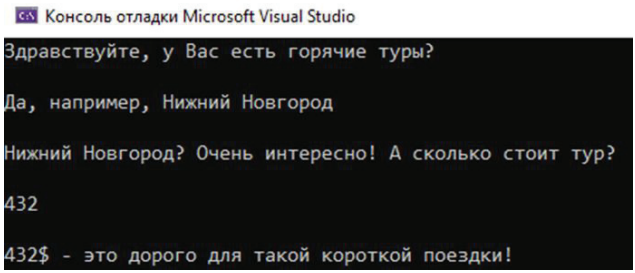
```
#include <iostream>
#include <windows.h>
using namespace std;
int main () {
   SetConsoleCP (1251);
   SetConsoleOutputCP (1251);
   char name [50];
   int s;
   cout << "Здравствуйте, у Вас есть горячие туры?\n";
   cout << "\nДа, например, ";
   cin.getline (name, 50);
   cout << "\n" <<name << "? Очень интересно! А сколько стоит тур?\n\n";
   cin >> s; cout << "\n";
```

```
if (s > 500) cout << s << "$ — это          очень дорого для меня!";
else if (s > 200) cout << s << "$ — это дорого для такой короткой поездки!";
else if (s > 100)
cout << s << "$ - нормальная цена, но у меня не хватает денег. Приду завтра.";
else cout << s << "$ — это мне подходит. Покупаю!";
return 0;}
```

2.6. Контрольные вопросы

1) Классификация и характеристики основных типов данных.
2) Перечисляемый тип данных.
3) Символьные массивы.
4) Ввод/вывод строк.
5) Классы памяти и что они определяют?
6) Автоматические переменные.
7) Статические переменные.
8) Внешние переменные.
9) Классификация операторов С++.
10) Операторы обработки данных.
11) Условный оператор. Вложенные условные операторы.
12) Что такое диалоговая программа?

Результат выполнения программы



3. ЛАБОРАТОРНАЯ РАБОТА № 3.

Программирование циклических алгоритмов для обработки массивов числовых и символьных данных.

3.1. Цель лабораторной работы

Целью лабораторной работы является освоение

- объявления и использования символьных массивов и массивов числовых данных (одномерных и многомерных);
- ввода/вывода данных числовых типов и символьных строк;
- организации обработки числовых массивов с использованием алгоритмов циклической структуры;

- использования операторов **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 [K1] [K2] …[KN];

type – тип элементов массива;

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

K1…KN – константные выражения, определяющие 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.

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 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 (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 отожествляется 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.

Имеются еще две формы инициализации массива:

int array [5] = { }; - инициализируем все элементы массива значением 0.

int array [5] {4, 5, 8, 9, 12}; - используем ***uniform-инициализацию*** для инициализации фиксированного массива.

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 нуль):

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.

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 этих строк массива в оперативной памяти.

Ввод/вывод числовых массивов

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

Схема алгоритма поэлементного ввода значений двумерного массива ***float T [3][4]*** с клавиатуры и вывода значений на экран дана на рис. 11.

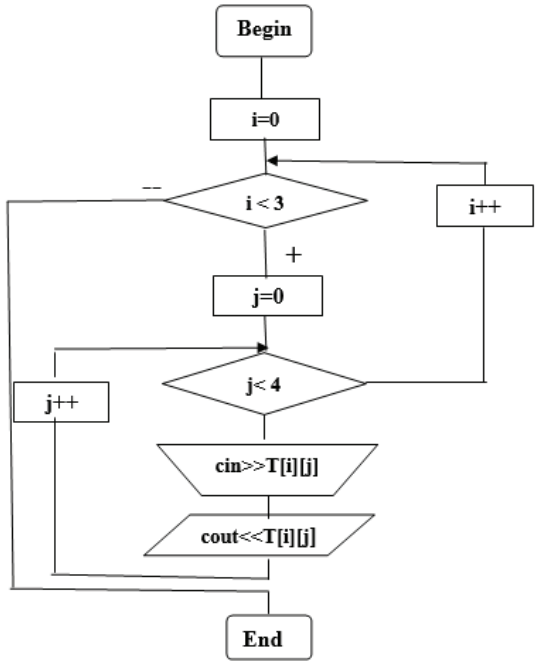


Рисунок 11. – Схема алгоритма ввода/вывода значений двумерного массива.

3.3. Задание на выполнение лабораторной работы

Разработать алгоритм и реализовать программу обработки массивов числовых и символьных данных согласно варианту. Вариант получить у преподавателя.

3.4. Порядок выполнения работы

1) Разработать алгоритм обработки массивов числовых данных и символьных данных.

2) Создать новый проект в среде Visual Studio 2019.

3) Написать в файле с расширением cpp текст программы в соответствии с алгоритмом. В программе следует:

- определить массив данных вещественного типа размером 5x6.
- ввести значения массива с клавиатуры или заполнить массив случайными числами.

- вывести элементы массив на экран в виде матрицы через интервал.
- определить символьный массив, инициализировать его строками шапки таблицы,
- 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.

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

5) Составить и защитить отчет.

3.5. Пример варианта лабораторной работы

Задание:

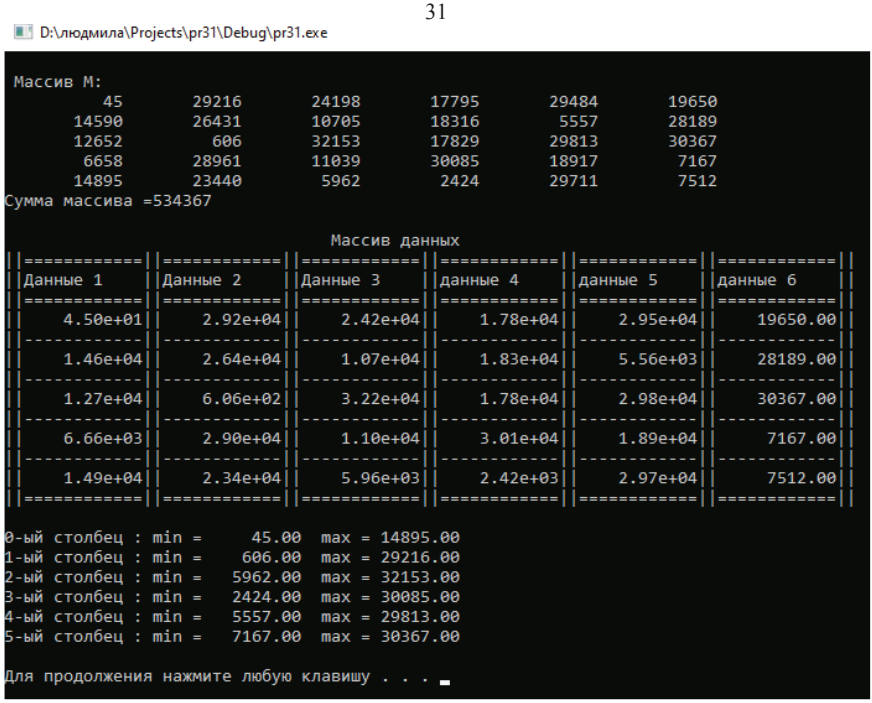
Написать программу обработки числового массива размером 5x6: ввод элементов массива с клавиатуры; вывод элементов массива на экран в виде матрицы и в таблицу в формате с плавающей и фиксированной точкой; вычисление и вывод на экран суммы элементов массива и минимальных и максимальных элементов каждого столбца.

Текст программы

```
#include<iostream>
#include<iomanip>
#include<windows.h>
using namespace std;
int main () {
SetConsoleCP (1251);
SetConsoleOutputCP (1251);
srand (time (0));
double M [5][6];
int i, j;
//-----------Ввод/вывод элементов массива-----------------------------------
double s = 0;                  // переменная для нахождения суммы элементов
cout << "\n Массив M: ";
for (i = 0; i<5; i++) {
    cout << "\n";
    for (j = 0; j<6; j++) {        //ввод/вывод элементов и подсчет суммы
       M[i][j] = rand ();          //cin >> M[i][j];
        cout << setw (12) << M[i][j];  s += M[i][j];
       }
}
cout << "\nСумма массива =" << s<<endl;
//------------------Вывод массива в таблицу------------------------------
//Массив строк шапки таблицы:
```

```
char sh [][90] = {"                    Массив данных              ",
"||==========||=========||=========||==========||========||=========||",
"||Данные 1   ||Данные 2   ||Данные 3   ||данные 4    ||данные 5  ||данные 6    ||",
"||==========||=========||=========||==========||========||=========||",
"||----------||---------||---------||----------||--------||---------||",
"||==========||=========||=========||==========||========||=========||"};
for (i = 0; i <4; i++)              //Вывод на экран строк шапки
     cout << sh[i] << endl;
for (i = 0; i<5; i++) {             //Цикл for по индексам строк массива
     cout << "||";
     for (j = 0; j<6; j++)          //Цикл for по индексам столбцов массива
     //форматный вывод элементов, отличающийся для разных столбцов
               switch (j) {
                 case 0: case 1: case 2: case 3: case 4: cout.unsetf (ios::fixed);
                              cout.setf(ios::scientific); cout <<setprecision(2) << setw(12)
                               << M[i][j] << "||";
                              break;
                 case 5: cout.unsetf(ios::scientific); cout.setf(ios::fixed);
                              cout << setprecision(2) << setw(12) << M[i][j] << "||\n";
                              break;
                 }
               if (i == 4)    cout<< sh[5] << endl;
                       else   cout << sh [4] << endl;
 }
double min, max;
for (j = 0; j < 6; j++) {
    min = max = M[0][j];
    for (i = 0; i < 5; i++)
    if (M[i][j] < min) min = M[i][j];
    else if (M[i][j] > max) max = M[i][j];
    cout << endl << j << "-ый столбец: " << "min = " << min << "  max = " << max;
  }
cout << endl<<endl;
system("pause");
return 0;}
```

Результат выполнения программы представлен на рис. 13



3.6. Контрольные вопросы

1) Массивы. Форматы определения массивов. Инициализация.

2) Массивы арифметических данных. Формат внутреннего представления одномерных и многомерных массивов.

3) Ввод/вывод элементов массивов.

4) Классы памяти и что они определяют?

5) Автоматические, внешние, статические переменные.

6) Классификация операторов С++. Операторы обработки данных.

7) Операторы организации обработки данных. Операторы выбора.

8) Операторы циклов.

9) Операторы передачи управления.

4. ЛАБОРАТОРНАЯ РАБОТА № 4

Разработка функций ввода и форматного вывода элементов числовых и символьных массивов.

4.1. Цель лабораторной работы

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

- правил определения функций;
- назначения и состава параметров функции;
- передачи параметров по значению, адресу, ссылке;

- методов передачи в функцию массивов данных;
- правил вызова функций;

4.2. Теоретические сведения

Функции

Программа на языке С++ представляет собой совокупность произвольного количества функций, одна (и единственная) из которых - главная функция с именем ***main***.

Выполнение программы начинается и заканчивается выполнением функции ***main.*** Выполнение неглавных функций инициируется в главной функции непосредственно или в других функциях, которые сами инициируются 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. Определение функции может располагаться в любой части программы, кроме как внутри других функций. В языке С++ нет вложенных функций.

Определение состоит из заголовка и тела функции:

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

1) ***тип*** – это 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, то есть, последовательность определений, описаний и операторов, заключенная в фигурные скобки.

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. Если функция не возвращает результата, 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.

Описание функции (прототип)

При вызове функции формальные параметры заменяются фактическими, причем соблюдается строгое соответствие параметров по типам. В связи с этой особенностью языка С++ 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.

Переменные, доступные функции

1) ***локальные переменные***:

- объявлены в теле функции, доступны только в теле функции;

- при определении переменной ей выделяется память в сегменте стека, при завершении выполнения функции память освобождается;

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;
- 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 не доступным;

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;
- 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 окончанию тела функции.

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***.

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.

Передача фактических параметров

В C++ передача параметров 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 объекту, являющемуся фактическим параметром.

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 4 байта. 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 С++ 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, 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 является ***l-значением,*** и она инициализирует ссылку, то есть связывает ссылку со своим 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.

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 является символьной строкой, то нужно, либо использовать только массивы фиксированного, заранее определенного размера, либо передавать значение размера массива явным образом с помощью дополнительного параметра.

При работе 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 любой мерности и второй параметр – общее количество элементов в массиве:***

type*p, int n.

Фактическими параметрами в этом случае будут – указатель типа ***type****, значение которого - адрес первого элемента массива и второй параметр - значение общего количества элементов в массиве**.**

При этом надо помнить, что имя массива любой мерности – это константный указатель, значением которого является адрес первого элемента массива, однако только для одномерного массива имя – есть указатель на элемент массива, для двумерного массива имя массива – это указатель на строку массива и так далее.

Чтобы получить указатель на элемент массива для двумерного массива, надо разыменовать имя массива, для трехмерного массива – два раза разыменовать имя массива и так далее. Или решать проблему явным приведением типов указателей.

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 - тип выходного файлового потока;

ifstream - тип входного файлового потока;

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

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

Примеры объявления 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.

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

void open (const char* filename, int mode= умалчиваемые значения,
int protection= умалчиваемые значения)

Функция открывает существующий файл или создает новый файл и связывает его с потоком для обмена данными. Вызов компонентной функции осуществляется с помощью уточненного имени:

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, то 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 правильности завершения функции ***open ()*** следует проверить значение выражения ***!имя потока***. Если значение 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 потока происходят от стандартного входного потока ***cin***, а выходные файловые потока – 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, а также функции форматирования файловые потоки наследуют от стандартных потоков. Рассмотрим вызовы некоторые из них.

Чтобы установить ширину поля вывода используется функция width():

fout.width (15);

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); - левое 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 ***unsetf ()*** - используется для отмены формата – параметра.

Закрытие файла

По окончанию 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 ();*** и ***fout.close ();*** - закроют соответствующие файлы.

4.3. Задание на выполнение лабораторной работы

Разработать алгоритм и реализовать программу обработки массивов числовых и символьных данных - ввода данных из файла данных в оперативную память и вывода данных из оперативной памяти в файл результатов форматно в виде таблицы.

4.4. Порядок выполнения работы

1) Сформировать файл с исходными данными (набить с клавиатуры). Сначала в файле должны располагаться символьные строки – строки шапки таблицы. 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) Провести отладку и тестирование программы.

4.5. Пример выполнения лабораторной работы

Задание:

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

2) Составить алгоритм и написать программу ввода данных из файла в оперативную память и вывода данных из оперативной памяти в файл результатов форматно в виде таблицы.

Схемы алгоритмов функций

На рис.12 представлена схема алгоритма главной функции ввода данных.

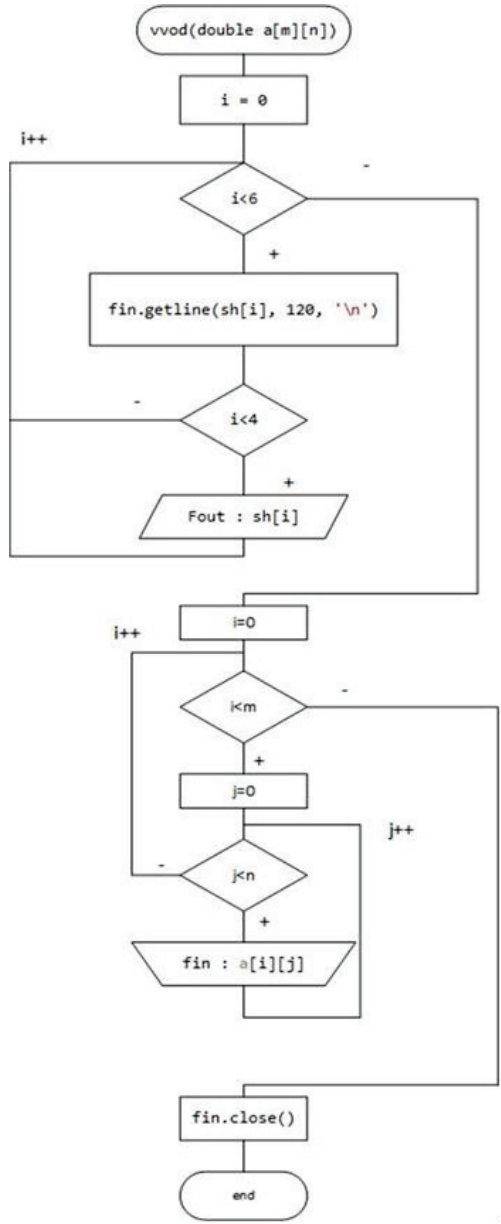


Рисунок 12. Схема алгоритма функции vvod ()

На рис.13 представлена схема алгоритма функции вывода данных.

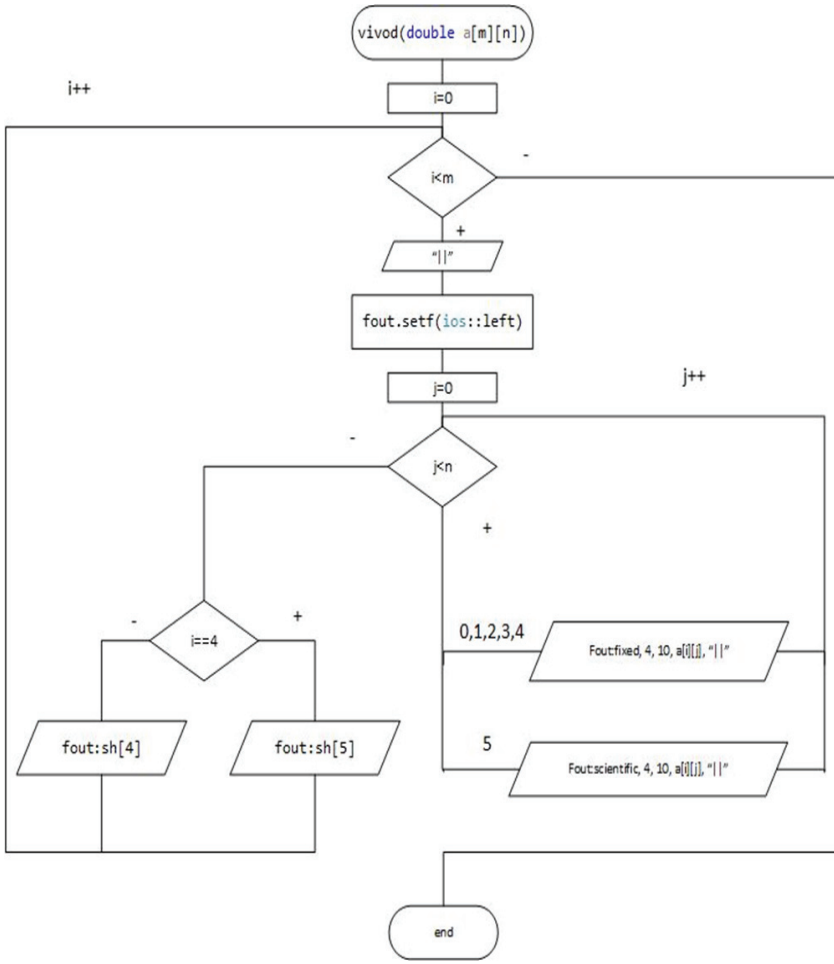


Рисунок 13. Схема алгоритма функции vivod ()

На рис.14 представлена схема алгоритма главной функции.

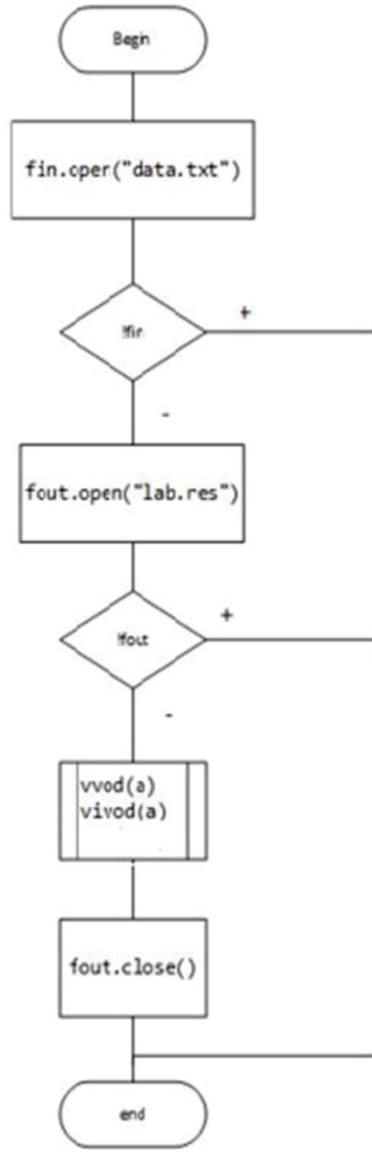


Рисунок 14. Схема алгоритма функции main ()

Текст программы

```
#include <iostream>
#include <windows.h>
#include <fstream>
#include <iomanip>
using namespace std;
ofstream fout;
ifstream fin;
const int m = 5, n = 6;
char sh [6][120];
//---------------------функция ввода данных------------------------------
void vvod (double a[m][n]) {
   int i, j;
   for (i = 0; i < 6; i++) {
      fin.getline (sh [i], 120, '\n');
      if (i < 4)
         fout << sh [i] << endl;
   }
   for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
         fin >> a[i][j];
   fin.close ();
}
//------------------------функция вывода данных----------------------------------
void vivod (double a[m][n]) {
   int i, j;
   for (i = 0; i < m; i++) {
      fout << "||";
      fout.setf (ios::left);
      for (j = 0; j < n; j++)
         switch (j) {
         case 0:
         case 1:
         case 2:
         case 3:
         case 4:
            fout.width (10);
            fout.unsetf (ios::scientific);
```

```
            fout.setf (ios::fixed);
            fout.precision (2);
            fout << a[i][j] << "||";
            break;
         case 5:
            fout.width(10);
            fout.unsetf(ios::fixed);
            fout.setf(ios::scientific);
            fout.precision(2);
            fout << a[i][j] << "||\n";
            break;
         }
      if (i = = 4)
         fout << sh [5] << endl;
      else
         fout << sh [4] << endl;
   }
   fout.unsetf (ios::scientific);
}
//----------------------------Главная функция-------------------------------
int main () {
   SetConsoleCP (1251);
   SetConsoleOutputCP (1251);
   double a[m][n];
   fin.open ("data.txt");
   if (!fin) {
      cout << "Ошибка при открытии файла данных!";
      exit (0);
   }
   fout.open ("l4.res");
   if (!fout) {
      cout << "Ошибка при открытии файла результатов!";
      exit(0);
   }
   vvod(a);
   vivod(a);
   fout.close ();
   return 0;
}
```

На рис. 15 показан файл с исходными данными.

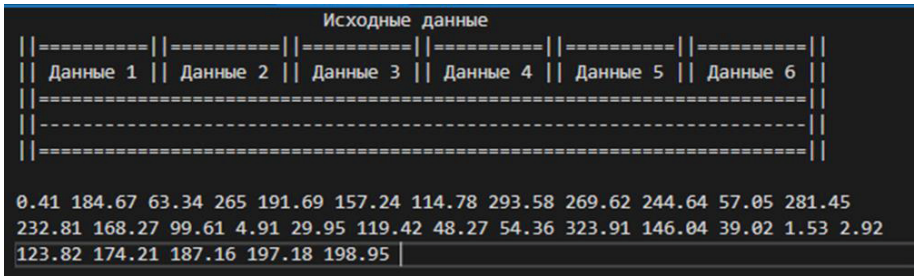


Рисунок 15. Файл данных data.txt

На рис. 16 показан файл результатов.

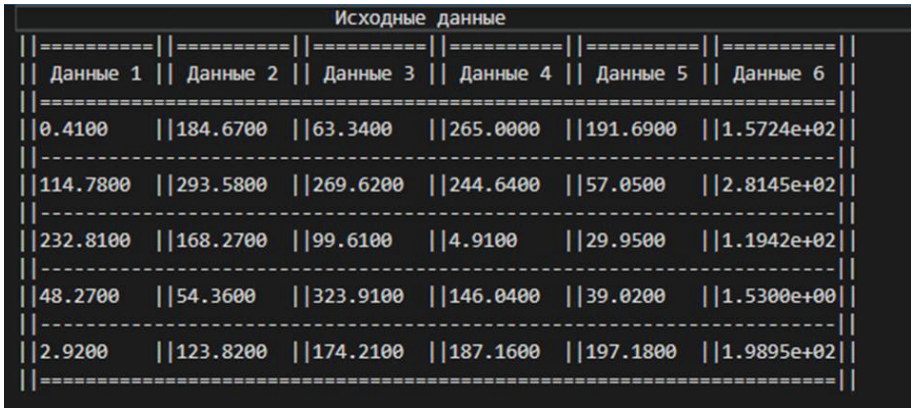


Рисунок 16. Файл данных "l4.res"

4.6. Контрольные вопросы

1. Что такое функция?
2. Определение, описание и вызов функции.
3. Переменные, используемые в функции.
4. Какими способами можно возвратить 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 использовать в теле функции с формальные параметры – указатели на скалярные данные?

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.

5. СПИСОК ЛИТЕРАТУРЫ

1) Надейкина Л.А. Программирование на языке высокого уровня. Часть 1. Учебное пособие. - М: МГТУ ГА, 2012, 84 с.

2) Надейкина Л.А. Программирование. Часть 2. Учебное пособие. - М: МГТУ ГА, 2017, 84 с.

3) Подбельский В.В. Стандартный Си++. М.: Финансы и статистика, 2008, 688с.

4) Павловская Т.А. С/ С++. Программирование на языке высокого уровня - СПб: Питер, 2011. – 461 с.

СОДЕРЖАНИЕ

Л.А. Надейкина

Программирование

Учебно-методическое пособие

В авторской редакции

Подписано в печать 20.05.2021 г.
Формат 60х84/16 Печ. л. 3 Усл. печ. л. 2,79
Заказ № 761/0429-УМП29 Тираж 30 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru