



МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный технический университет

Кафедра «Машины и технология литейного производства»

ИНФОРМАТИКА

Лабораторный практикум

**Минск
БНТУ
2024**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Машины и технология литейного производства»

ИНФОРМАТИКА

Лабораторный практикум
для студентов специальности
6-05-0714-03 «Инженерно-техническое проектирование
и производство материалов и изделий из них»
профилизации «Машины и технология литейного производства»,
«Аддитивные технологии в литейном производстве»

*Рекомендовано учебно-методическим объединением по образованию
в области металлургического оборудования и технологий*

Минск
БНТУ
2024

УДК 004. (076.5)
ББК 32.81я7
И74

Составители:

Е. В. Телешова, С. А. Мацинов, С. Л. Ровин

Рецензенты:

декан механико-технологического факультета ГГТУ
им. П. О. Сухого, доцент кафедры «Металлургия и технологии
обработки материалов», канд. техн. наук *И. Б. Одарченко*;
заведующий отделом комплексных программ
и формовочного оборудования ОАО «БЕЛНИИЛИТ»,
канд. техн. наук *Д. М. Голуб*

И74 Информатика : лабораторный практикум для студентов специальности 6-05-0714-03 «Инженерно-техническое проектирование и производство материалов и изделий из них» профилизации «Машины и технология литейного производства», «Аддитивные технологии в литейном производстве» / сост. : Е. В. Телешова, С. А. Мацинов, С. Л. Ровин. – Минск : БНТУ, 2024. – 64 с.
ISBN 978-985-583-999-7.

Практикум разработан в соответствии с учебной программой по курсу «Информатика» (раздел «Программирование»), содержит требования и необходимую информацию для выполнения лабораторных работ по указанной дисциплине.

Издание предназначено для закрепления теоретических знаний, полученных при изучении дисциплины, а также приобретения практических навыков, необходимых для освоения информационных технологий и использования их в профессиональной инженерной деятельности.

УДК 004 (076.5)
ББК 32.81я7

ISBN 978-985-583-999-7

© Белорусский национальный
технический университет, 2024

Оглавление

Введение	4
Лабораторная работа № 1. Типовая структура программы.	
Вывод текста в консоль.....	5
Лабораторная работа № 2. Понятие алгоритма	10
Лабораторная работа № 3. Арифметические операции	13
Лабораторная работа № 4. Основные типы данных. Арифметические выражения	19
Лабораторная работа № 5. Программирование разветвляющихся алгоритмов. Оператор if	23
Лабораторная работа № 6. Оператор организации циклов for	28
Лабораторная работа № 7. Операторы циклов while и do while	32
Лабораторная работа № 8. Обработка одномерных массивов. Сортировка массивов	36
Лабораторная работа № 9. Обработка двумерных массивов	42
Лабораторная работа № 10. Обработка матриц в С++	46
Лабораторная работа № 11. Объявление, определение и вызов функции	51
Лабораторная работа № 12. Строки и символы.....	58
Литература	64

Введение

Цель изучения учебной дисциплины «Информатика» – научить студентов применять современные информационные технологии в практической деятельности инженера.

Изучение основ информатики и программирования дает возможность эффективно использовать современные методы численного решения инженерных задач, знакомит со структурами хранения данных, возможностями их эффективной обработки, пополнения и использования. Знание основ программирования дает возможность самостоятельной автоматизации доступа к данным, их наглядному представлению в локальных и глобальных сетях.

Основными задачами изучения учебной дисциплины во 2-м семестре являются:

- изучение основ алгоритмизации инженерных задач;
- изучение языка программирования C++;
- освоение навыков формирования баз данных;
- изучение основных методов решения математических задач, описывающих инженерные проблемы.

Лабораторная работа № 1

ТИПОВАЯ СТРУКТУРА ПРОГРАММЫ. ВЫВОД ТЕКСТА В КОНСОЛЬ

Цель работы: освоение простейшей структуры программы на языке C++; получение навыков в организации ввода-вывода на языке C++.

Общие сведения

Язык C++ является компилируемым языком. Это означает, что программа, написанная на этом языке, должна быть преобразована в байт-код специальными средствами в исполняемый файл (.exe), который в свою очередь содержит инструкции, понятные операционной системе, для которой эта программа была написана. Такие средства называются компиляторами.

Для удобства разработки текстовый редактор кода, компилятор, средства отладки и т. д. могут быть объединены в единое программное обеспечение, которое называют интегрированной средой разработки (IDE). Удобство заключается в автоматизации большинства настроек компилятора, подсказках во время написания кода, графическом интерфейсе, возможностях пошагового выполнения программы и т. д. Примерами таких сред являются Visual Studio, Dev C++.

Создание проекта. Чтобы создать проект, нажмите «Файл», а затем «Создать проект». В появившемся меню выберите «Пустой проект. C++», а затем назовите свой проект в поле «Имя проекта». Теперь в окне «Обозреватель решений» правой кнопкой мыши кликните на «Исходные файлы», затем выберите «Добавить», «Создать элемент», «Файл C++ (.cpp)» – и введите имя файла (оно может совпадать с названием проекта).

Рассмотрим структуру программы на языке C++ на типичном примере «Hello World». Задача: вывести в консоль заранее заданный текст («Hello world»).

```
#include <iostream>
using namespace std;
int main()
```

```

{
cout<<"Hello World";
return 0;
}

```

Основа каждой программы – это набор упорядоченных команд, которые необходимо выполнить для решения поставленной задачи. В случае описываемой программы необходима была только одна команда: **вывести на экран строку** «Hello World». И на языке C++ она записывается: `cout << "Hello World";`. Остальные строки программы являются стандартными для большинства задач данного пособия и будут составлять каркас будущих программ.

Каждая команда должна оканчиваться специальным знаком «;». Именно по этому знаку происходит разделение кода на команды (переносы строк между командами не обязательны, но желательны для поддержания «читаемости» кода).

Разберем подробнее строку кода, отвечающую за вывод в консоль: «`cout << "Hello world";`».

Представим, что есть некоторое виртуальное представление сущности «консоль». Также есть две очереди: первая – элементы, которые ожидают вывода в консоль; вторая – элементы, которые были введены пользователем в консоль с клавиатуры и ожидают обработки.

Первая очередь именуется – *cout*, вторая – *cin*. В данной работе работа будет вестись только с *cout*, *cin* будет рассмотрен позднее.

«<<» – в случае cout является оператором вывода.

После этого оператора могут идти любые элементы, которые позволено выводить в консоль (для которых определено, как именно происходит вывод в консоль). В нашем случае это строка (любая последовательность символов, записанная в двойных кавычках). Для упрощения можно запоминать следующим образом: **«консоль (cout) вывести (<<) строка ("Hello world")».**

«<<» является бинарным оператором (работает с двумя аргументами). Любая операция имеет результат (например, для операции «+» и выражения «2 + 2» результат будет целым числом 4). Это позволяет записывать сложные выражения (например, «(2 + 2) + 2»). Операция «<<» работает точно так же, а в качестве результата всегда возвращает саму сущность «cout». Это означает, что можно записывать такие составные выражения, как: «((cout << "Hello") << " ") << "World";».

При этом, так как все операторы одинаковые и имеют одинаковый приоритет, то, как и в случае с « $(2 + 2) + 2 = 2 + 2 + 2$ », скобки можно опустить и записать «`cout << "Hello" << " " << "World";`».

Для использования библиотеки необходимо ее подключить. Это делается с использованием команды «`#include <iostream>`», где вместо «`iostream`» может быть имя другой библиотеки. Стоит заметить, что эта команда является особенной. Во-первых, она находится за пределами функции **main** и любых других функций, во-вторых, она не оканчивается «;». Это связано с тем, что данная команда является директивой препроцессора.

Все команды, которые мы рассматривали ранее, так или иначе выполнялись в процессе действия программы. Т. е. это были команды самой программы. Команды, начинающиеся с «`#`», являются директивами препроцессора (командами для компилятора). Эти команды выполняются на одном из этапов сборки программы, и в конечной программе их уже нет. Так, директива «`#include`» помещает на место, где она записана, код из указанного файла. Так можно разбивать свою программу на множество файлов, если код становится слишком объемным. При этом, если имя библиотеки записано в треугольные скобки («`#include <iostream>`»), то файл ищется в стандартных файлах языка C++, а если в двойных кавычках («`#include "myfile.cpp"`»), то ищется в папке с файлом, где эта директива указана.

Разбиение программы на файлы в рамках данного пособия производиться не будет. Стандартные библиотеки, которые будут использоваться в ближайших работах: *iostream* – библиотека для работы с очередями (потоками) ввода/вывода (*input output stream*; включает `cin`, `cout`). *math.h* – библиотека для работы с математическими функциями (`sqrt` – квадратный корень, `abs` – модуль). Строка «`using namespace std;`» необходима для сокращенного написания имен стандартных сущностей и функций (таких, как `cin`, `cout`).

Например, полные имена для *cin*, *cout* – `std::cin`, `std::cout`. Конструкция «`using namespace имя_пространства;`» позволяет указать системе, что если используется сущность и ее объявление не найдено в основном файле, то следует искать ее так же в указанном пространстве имен. В приведенном примере объявление «`cout`» сначала будет искаться в основном файле, а затем в пространстве имен «`std`».

При описании в коде выводимого текста открывающаяся и закрывающаяся двойные кавычки должны находиться на одной строке. Для вывода переноса строки и подобного используются специальные символы. Каждый такой символ начинается с обратного слеша «\». Примеры специальных символов: «\n» – перенос строки, «\t» – табуляция. Для вывода самого обратного слеша его необходимо записать дважды «\\». Также для обозначения переноса строки можно использовать специальную сущность из пространства имен «std» – «endl».

Например, для вывода «Hello» и «World» на разных строках код может выглядеть следующим образом:

```
cout << "Hello\nWorld";  
cout << "Hello" << endl << "World";
```

Также стоит отметить конструкции для пояснительных комментариев в коде. **Комментарии** представляют собой специально обозначаемые фрагменты программы (любая совокупность знаков), содержание которых компилятор игнорирует.

В C++ поддерживаются два типа комментариев: однострочные и многострочные. *Однострочный комментарий* начинается с пары символов // и продолжается только до конца строки. *Многострочный комментарий* должен начинаться символами /* и заканчиваться */ , может занимать несколько строк. Например,

```
// Это однострочный комментарий  
/* Этот комментарий  
уже  
многострочный */
```

Правила оформления кода:

1. После открывающейся фигурной скобки добавляется отступ в начале строки, на строке с закрывающейся фигурной скобкой отступ убирается.

2. Все бинарные операции (+, -, *, /, %, =, <<, >>) окружаются пробелами.

3. После унарного минуса пробел не ставится (-5 нужно писать слитно).

4. Перед знаками препинания (запятая и точка с запятой) пробел не ставится, после – ставится.

5. После открывающейся и перед закрывающейся круглой скобкой пробел не ставится.

6. Если в условии задачи сказано «на вход даются два числа A и B », то переменные, в которые считываются эти числа, должны называться так же, но маленькими буквами (a и b соответственно).

Задание

Написать программу, которая выведет в консоль ситуацию из игры в крестики-нолики следующего вида:

```
X | O | X
O | X | X
O | X | O
```

При этом сделать разные варианты реализации:

1. С использованием конструкции `using namespace std`.
 2. Без использования конструкции `using namespace std`.
 3. С использованием `endl`.
 4. Без использования `endl`.
 5. Чтобы в коде весь вывод был записан в одну строку (одной командой).
 6. Чтобы в коде также читалось указанное изображение. Т. е. чтобы крестики и нолики, находящиеся друг под другом на изображении, также находились друг под другом в нужной позиции и в коде.
- В данном задании ввод и вывод в консоль осуществлять только латинскими символами (английский алфавит).

Содержание отчета

1. Тема и цель работы.
2. Краткие теоретические сведения.
3. Задание. Текст программы.
4. Результаты выполнения программы.

Лабораторная работа № 2

ПОНЯТИЕ АЛГОРИТМА

Цель работы: освоение способов описания алгоритма.

Общие сведения

Понятие алгоритма занимает центральное место в современной математике и программировании.

Алгоритм – строгий и четкий набор правил, определяющий последовательность действий, приводящих к достижению поставленной цели.

Существует несколько способов описания алгоритмов. Наиболее распространенные способы – это словесное и графическое описания алгоритма.

При *словесной записи* алгоритм описывается с помощью естественного языка с использованием следующих конструкций:

- 1) шаг (этап) обработки (вычисления) значений данных – « \Rightarrow »;
- 2) проверка логического условия: если (условие) истинно, то выполнить действие 1, иначе – действие 2;
- 3) переход к определенному шагу (этапу) N .

Для примера рассмотрим алгоритм решения квадратного уравнения вида $a \cdot x^2 + b \cdot x + c = 0$:

- 1) ввод исходных данных a, b, c ($a, b, c \neq 0$);
- 2) вычислить дискриминант $D = b^2 - 4 \cdot a \cdot c$;
- 3) если $D < 0$, то перейти к п. 6, сообщив, что действительных корней нет;
- 4) иначе, если $D \geq 0$, вычислить $x_1 = (-b \pm \sqrt{D}) / (2 \cdot a)$;
- 5) вывести результаты x_1 и x_2 ;
- 6) действительных корней нет.

Графическое изображение алгоритма – это представление его в виде схемы, состоящей из последовательности блоков, каждый из которых отображает содержание очередного шага алгоритма. А внутри фигур кратко записывают действие, выполняемое в этом блоке. Такую схему называют блок-схемой. Она позволяет сделать

алгоритм более наглядным и выделяет в нем основные алгоритмические структуры (линейная, ветвление, выбор и цикл). Элементы блок-схемы размещены в табл. 2.1.

Таблица 2.1

Элементы блок-схемы

Элемент блок-схемы	Назначение элемента блок-схемы
	Применяется для обозначения начала или конца алгоритма
	Данные ввода-вывода
	Для описания линейной последовательности команд
	Для обозначения условий в алгоритмических структурах «ветвление» и «выбор», имеет один вход сверху и два выхода (налево – истинно, и направо – ложно)
	Для вызова отдельно описанного алгоритма (подпрограммы)
	Для объявления переменных или ввода комментариев
	Комментарий
	Соединитель – используется при обрыве линии и продолжении ее в другом месте (необходимо присвоить название)

Пример простейшего линейного процесса. Необходимо организовать расчет арифметического выражения (2.1) при различных исходных данных:

$$z = \frac{\operatorname{tg}^2 x}{\sqrt{x^2 + m^2}} + x^{(m+1)} \sqrt{x^2 + m^2}, \quad (2.1)$$

где $x > 0$ – вещественное, m – целое.

Разработка алгоритма обычно начинается с составления схемы. Продумывается оптимальная последовательность вычислений, при которой, например, отсутствуют повторения. При написании алгоритма рекомендуется переменным присваивать те же имена, которые фигурируют в заданном арифметическом выражении либо иллюстрируют их смысл.

Для того чтобы не было «длинных» операторов, исходное выражение полезно разбить на ряд более простых. В примере предлагается схема вычислений, представленная на рис. 2.1.

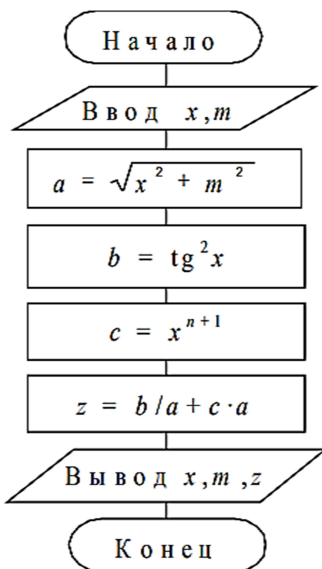


Рис. 2.1. Схема линейного процесса

Задание

Для задач из лабораторной работы № 1 записать словесный и составить графический алгоритм.

Содержание отчета

1. Тема и цель работы.
2. Схемы алгоритмов программ.

Лабораторная работа № 3

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Цель работы: получение навыков в организации ввода-вывода на языке C++; использование операций присвоения и арифметических операций.

Общие сведения

Выражения используются для вычисления значений (определенного типа) и состоят из операндов, операций и скобок. Каждый операнд может быть, в свою очередь, выражением или одним из его частных случаев – константой или переменной. Операнды задают данные для вычислений. **Выражение** – это последовательность операндов, разделителей (круглых скобок) и знаков операций, задающая вычисление. **Знак операции** – это один или более символов, определяющих действие над операндами, то есть операции задают действия, которые необходимо выполнить. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные и тернарные – по количеству участвующих в них операндов, и выполняются в соответствии с приоритетами. Большинство операций выполняются слева направо.

Арифметические операции служат для описания *арифметических действий*:

- вычитание;
- + сложение;
- * умножение;
- / деление ($10 / 3$ равно 3; $-7 / 3$ равно -1)
- % вычисление остатка от деления первого аргумента на второй ($10 \% 3$ равно 1; $8 \% 4$ равно 0; $-10 \% 3$ равно -1);
- ++ увеличение на единицу (increment);
- уменьшение на единицу (decrement).

Пример. Демонстрация использования арифметических операций.

```
#include<iostream>
using namespace std;
int main()
```

```

{
int n; // Объявление переменной n типа int
int variable; // Объявление переменной variable типа int
cout << "Vvedite n "; // Вывод сообщения
cin >> n; // Ввод числа с клавиатуры в переменную n
cout << "n "<<" variable" << endl; // Вывод строки
/* Замена знака*/
variable = -n; /* Присваивание переменной variable отрицатель-
ного значения n = -10 */
cout << n <<" " << variable << endl; /* Вывод переменных n
и variable, разделенных пробелами */
// Инкремент
variable = ++n; // После выполнения n = 11, variable = 11
cout << n <<" " << variable << endl; variable = n++; /*После вы-
полнения n = 12, variable = 11 */
cout << n <<" " << variable << endl; // Декремент
variable = --n; // После выполнения n = 11, variable = 11
cout << n << " " << variable << endl; variable = n--; /*После вы-
полнения n = 10, variable = 11 */
cout << n << " " << variable << endl;
// Сложение
variable = n + 7; // После выполнения n = 10, variable = 17
cout << n << " " << variable << endl;
// Вычитание
variable = n - 5; // После выполнения n = 10, variable = 5
cout << n << " " << variable << endl;
// Умножение
variable = n*4; //После выполнения n = 10, variable = 40
cout << n << " " << variable << endl;
// Деление
variable = n / 3; // После выполнения n = 10, variable = 3
cout << n << " " << variable << endl;
// Вычисление остатка от деления первого аргумента на второй
variable = n % 3; // После выполнения n = 10, variable = 1
cout << n << " " << variable << endl;
return 0;
}

```

Операция присваивания может быть простой и составной.

Простая – двухаргументная операция вида: *переменная* = *выражение*. В этой операции тип выражения всегда преобразуется в тип переменной. Выполнение операции присваивания приводит к присваиванию переменной значения выражения, стоящего с правой стороны оператора присваивания. Результатом операции является значение правой стороны оператора, что позволяет составлять цепочки присваиваний, например $x = y = z = 100$; (результатом выполнения "=" является значение выражения, стоящего с правой стороны, то есть все переменные получают значение, равное 100).

Пример 1:	Смысл записи	Пример 2:	Значения
<i>int i, j, k;</i>		<i>int n, a, b, c, d;</i>	
<i>float x, y;</i>		$n = 2; a = b = c = 0;$	
...		$a = ++n;$	$n = 3, a = 3$
$x *= y;$	$x = x * y;$	$a += 2;$	$a = 5$
$i += 2;$	$i = i + 2;$	$b = n++;$	$b = 3, n = 4$
$x /= y + 15;$	$x = x / (y + 15);$	$b -= 2;$	$b = 1$
$-k;$	$k = k - 1;$	$c = --n;$	$n = 3, c = 3$
$k--;$	$k = k - 1;$	$c *= 2;$	$c = 6$
$j = i++;$	$j = i; \quad i = i + 1;$	$d = n--;$	$d = 3, n = 2$
$j = ++i;$	$i = i + 1; \quad j = i;$	$d \% = 2;$	$d = 1$

Рис. 3.1. Операции присваивания

В данной работе будем работать с целочисленными переменными. Если мы хотим присвоить значение переменной, то слева следует написать имя присваиваемой переменной, затем знак равно, а справа – арифметическое выражение, в котором могут использоваться числа и другие переменные. Например:

```
x = 2 + 3;
y = x * 4.
```

В результате выполнения этих операций в переменной x окажется число 5, а в переменной y – число 20.

Переменные также можно считывать с клавиатуры. Вот пример программы, которая считывает два числа a и b и выводит их сумму:

```
#include <iostream>
using namespace std;
int main() {
```



```

int a, b;
cin >> a >> b;
cout << a + b;
return 0;
}

```

Обратите внимание, что при чтении из *cin* стрелочки направлены в противоположную по сравнению с *cout* сторону. Мы «забираем» данные из потока ввода *cin* и «кладем» в *cout*. Если мы хотим считать несколько переменных, то при перечислении их следует разделять стрелочками. Строку

```
cin >> a >> b;
```

можно заменить строками

```
cin >> a;
```

```
cin >> b;
```

которые будут делать то же самое.

В *cout* также можно класть несколько значений, причем числа и строки могут идти вперемешку. Рассмотрим это на примере. Пусть нам нужно считать два числа и вывести их разность в виде арифметического выражения. Решение этой задачи выглядит так:

```

#include <iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a << " - " << b << " = " << a - b;
    return 0;
}

```

Если ввести числа 1 и 2, то на экран будет выведено:

```
1 - 2 = -1.
```

Все различные значения, которые мы хотим вывести с помощью *cout*, следует разделять стрелочками <<. На место переменных и арифметических выражений будет подставлено их значение, а все, что выводится в кавычках, останется без изменения.

Если вы выводите несколько чисел, то обязательно добавляйте между ними пробел, иначе они склеятся и ответ будет неправильным.

Пример. Предположим, с самолета сбросили груз. Приземлился он через t секунд, и нужно определить, на какой высоте летел самолет.

Ускорение свободного падения нам известно (поскольку наши числа целые, то мы возьмем его равным 10). Мы посчитаем скорость (v), на которой груз достиг земли, затем среднюю скорость (vm , это конечная скорость, поделенная на 2) и, зная среднюю скорость и время, легко рассчитаем расстояние.

```
#include <iostream>
using namespace std;
int main() {
    int t, v, g = 10;
    cin >> t;
    v = g * t;
    int vm = v / 2;
    int s = vm * t;
    cout << s;
    return 0;
}
```

В этом решении мы заводили переменные там, где они нам понадобились впервые, а также сразу клали в переменную заданное число при ее создании.

Пример. Даны два числа a и b , причем $b > 0$. Надо посчитать целую часть от деления a на b , округленную вверх. Напомним, что при делении C++ округляет результат вниз, не так, как нам нужно.

Решение: прибавить к числу что-нибудь и затем разделить его с округлением вниз. Если число a делится на b нацело, то результат не должен изменяться, значит, нельзя прибавлять к числу a что-либо большее $b - 1$ (если прибавить больше, то результат деления получится уже больше правильного). Можно ли прибавить что-нибудь меньшее $b - 1$? Рассмотрим «худший» случай, когда остаток от деления a на b равен единице, например, $a = 11$, $b = 5$. Тогда мы сложим a и $b - 1$ (получим 15) и разделим на 5 – получится правильный ответ 3.

Наше решение будет работать и для отрицательных чисел за счет особенностей деления на C++. Полный код решения выглядит так:

```
#include <iostream>
using namespace std;
int main() {
```

```
int a, b;  
cin >> a >> b;  
cout << (a + b - 1) / b;  
return 0;  
}
```

Задание

1. N студентов получили K заданий и решили разделить их поровну. Определите, сколько заданий останется после того, как все студенты возьмут себе равное количество заданий. На вход дается два целых положительных числа N и K , каждое из которых не превышает 1000.

2. Дано целое трехзначное число. Найдите сумму его цифр. На вход дается число от 100 до 999.

3. Посчитайте произведение, сумму и разность трех чисел.

4. Дано четырехзначное число, вывести на экран в обратном порядке цифры, из которых это число состоит. Подсказка: чтобы взять из числа отдельные цифры, надо применять деление по модулю на 10.

Содержание отчета

1. Тема и цель работы.
2. Краткие теоретические сведения.
3. Схема алгоритма решения.
4. Текст и результаты выполнения программы

Лабораторная работа № 4

ОСНОВНЫЕ ТИПЫ ДАННЫХ. АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ

Цель работы: освоение способов описания алгоритма; изучение основных типов данных; получение навыков в программировании математических задач.

Общие сведения

Операторы языка C++ манипулируют переменными и константами в виде выражений. Имена, которые используются для обозначения переменных, функций, меток и других определенных пользователем объектов, называются *идентификаторами*. Идентификатор может содержать латинские буквы, цифры и символ подчеркивания, и начинаться обязан с буквы или символа подчеркивания. Идентификатор не должен совпадать с ключевыми словами.

Рассмотрим понятие *переменной* – это именованная область памяти, используемая для хранения информации.

Данные различного типа хранятся и обрабатываются по-разному. В любом языке каждая переменная должна иметь определенный тип, который необходимо указать до использования переменной. Эта операция называется *объявлением переменной* и имеет синтаксис:

тип_данных *имя_переменной*.

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- множество допустимых значений, которое может принимать переменная этого типа;
- операции и функции, которые можно применять к величинам этого типа. В табл. 4.1 представлены некоторые типы данных C++.

При определении переменной в языке C++ необходимо предоставить компилятору информацию о ее типе. С помощью одной инструкции можно объявить сразу несколько переменных, например: **int** d1, d2, d3;

В языке C++ выражение, после которого стоит точка с запятой, считается оператором, то есть законченным действием. Перечень

описываемых переменных одного типа (тип указывается в начале перечня) обязательно должен заканчиваться точкой с запятой.

Таблица 4.1

Основные типы данных

Тип данных	Размер в байтах	Название	Диапазон
bool	1	Логический тип	true или false
char	1	Символьный тип	От 0 до 255
int	4	Целые числа	От -2147483648 до 2147483647
float	4	Действительные числа	От 1,8E - 38 до 1,8E + 38
double	8	Действительные числа двойной точности	От 2,2E - 308 до 1,8E + 308

Вместе с идентификаторами типов могут использоваться так называемые модификаторы типа. *Модификаторы типа* – это специальные ключевые слова, которые указываются перед идентификатором типа и позволяют изменять базовый тип. В C++ используются модификаторы: **signed** (значения со знаком), **unsigned** (значения без знака), **short** (укороченный тип), **long** (расширенный тип).

Все четыре модификатора могут использоваться для типа **int**. Модификаторы **signed** и **unsigned**, кроме этого, используются с типом **char**. Модификатор **long** используют с типом **double**.

В C++ предусмотрен сокращенный способ объявления **unsigned**-, **short**- и **long**-значений целочисленного типа. Это значит, что при объявлении **int**-значений достаточно использовать слова **unsigned**, **short** и **long**, не указывая тип **int**, то есть тип **int** подразумевается. Например, следующие две инструкции объявляют целочисленные переменные без знака: **unsigned x**; **unsigned int y**;

Тип **char** предназначен для хранения символов. Чтобы задать символ, необходимо заключить его в одинарные кавычки. Например, **char ch**;

```
ch='x';
```

Тип **bool** предназначен для хранения булевых (то есть ИСТИНА/ЛОЖЬ) значений. В C++ определены две булевы константы: **true** и **false**, являющиеся единственными значениями, которые могут иметь переменные типа **bool**.

Константы – именованные ячейки памяти, значения которых фиксируются на начальном этапе выполнения программы и не могут быть изменены программой. В этом смысле константы – это те же переменные, но только с фиксированным, определенным единожды значением. Константы могут иметь любой базовый тип данных.

Объявляются константы при помощи спецификатора **const**:

const тип_константы имя_константы=значение;

Например, так можно объявить константу с именем **max**:

const int max=9;

Таблица 4.2

Стандартные математические функции

Математическая функция	В C++	Математическая функция	В C++
\sqrt{x}	sqrt(x)	$\arcsin(x)$	asin(x)
$ x $	fabs(x)	$\arccos(x)$	acos(x)
e^x	exp(x)	$\arctg(x)$	atan(x)
x^y	pow(x,y)	$\arctg(x / y)$	atan2(x)
$\ln(x)$	log(x)	$\tgh(x)$	tanh(x)
$\lg_{10}(x)$	log10(x)	остаток от деления x на y	fmod(x,y)
$\sin(x), \cos(x)$	sin(x), cos(x)	наименьшее целое $\geq x$	ceil(x)
$\tgg(x)$	tan(x)	наибольшее целое $\leq x$	floor(x)

Задание

1. Дано положительное действительное число X . Выведите его дробную часть. Пример: дано число 17,9, выводом ответа должно быть 0,9.

2. Даны длины сторон треугольника. Вычислите площадь треугольника.

3. Напишите программу для расчета по двум формулам. Предварительно подготовьте тестовые примеры по второй формуле с помощью калькулятора (результат вычисления по первой формуле должен совпадать со второй). Игнорировать возможность деления на ноль.

1. $z_1 = \cos \alpha + \sin \alpha + \cos 3\alpha + \sin 3\alpha$
 $z_2 = 2\sqrt{2} \cos \alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$
2. $z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$
 $z_2 = \cos^2 \alpha + \cos^4 \alpha$
3. $z_1 = \frac{x^2 + 2x - 3 + (x+1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x-1)\sqrt{x^2 - 9}}$
 $z_2 = \sqrt{\frac{x+3}{x-3}}$
4. $z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$
 $z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$
5. $z_1 = \frac{\sqrt{(3m+2)^2 - 24m}}{3\sqrt{m} - \frac{2}{\sqrt{m}}}$
 $z_2 = -\sqrt{m}$
6. $z_1 = \left(\frac{a+2}{\sqrt{2a}} - \frac{a}{\sqrt{2a}+2} + \frac{2}{a-\sqrt{2a}}\right) \cdot \frac{\sqrt{a}-\sqrt{2}}{a+2}$
 $z_2 = \frac{1}{\sqrt{a}+\sqrt{2}}$
7. $z_1 = \cos \alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$
 $z_2 = 4 \cos \frac{\alpha}{2} \cdot \cos \frac{5}{2}\alpha \cdot \cos 4\alpha$

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 5

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ. ОПЕРАТОР IF

Цель работы: изучение условного оператора **if**; получение навыков в программировании разветвляющихся алгоритмов.

Общие сведения

Под *условными* обычно подразумевают операторные конструкции, с помощью которых в программе реализуются точки ветвления. В C++ используются условные операторы **if** и **else**.

Оператор **if** используется для разветвления процесса вычислений на несколько направлений, позволяет выполнять разные блоки операторов в зависимости от того, выполняется ли некое условие.

Вариант а

if (*выражение*) *инструкция 1*; **else** *инструкция 2*;

Вариант б

if (*выражение*)

{*последовательность инструкций 1*}

else

{*последовательность инструкций 2*}

Условие указывается в круглых скобках после ключевого слова **if**. Если его значение истинно, то выполняется *инструкция 1* (для варианта а) или *последовательность инструкций 1* (для варианта б), в противном случае выполняется *инструкция 2* (для варианта а) или *последовательность инструкций 2* (для варианта б).

После выполнения условного оператора управление передается оператору, следующему после него.

Пример. В данной программе вычисляется частное чисел x и y . Если делитель равен нулю, то выдается соответствующее сообщение, иначе вычисляется частное.


```

#include<iostream>
using namespace std;
int main()
{
float x,y,z; /* Объявление трех переменных типа float */
cout<<"Enter x: "; /* Вывод текстового сообщения */
cin>> x;      /* Ввод с клавиатуры значения переменной x */
cout<<"Enter y: "; // Вывод текстового сообщения
cin>> y;      /* Ввод с клавиатуры значения переменной y. Если
y = 0, то вывод сообщения о невозможности деления на ноль */
if (y == 0) cout <<"You can not divide by zero!"<< endl;
else //Иначе, т. е. если y не равен нулю
{z = x/y;    // Вычисление значения переменной z
cout <<"z = "<< z << endl;} /* Вывод значения z на экран */
return 0; }

```

```

Enter x: 8.26
Enter y: 0
You can not divide by zero!
Enter x: 9.44
Enter y: 3.22
z = 2.93168

```

Рис. 5.1. Результаты выполнения программы при разных x и y

Если требуется проверить несколько условий, их объединяют знаками логических операций. Например,

```

if (a < b && (a > d || a == 0))b++;
else
{b *= a;
a = 0;}

```

Выражение в примере будет истинно в том случае, если выполняются одновременно условие $a < b$ и одно из условий в скобках. Если опустить внутренние скобки, будет выполнено сначала логическое И, а потом ИЛИ.

Операции сравнения служат для описания следующих действий:

```

> (<)    больше (меньше);
>= (<=) больше или равно (меньше или равно);
==       проверка на равенство;
!=       неравенство.

```

Вторая форма оператора *if*

Нередко на практике используют комбинацию из нескольких вложенных условных операторов:

if (*выражение*) {операторы 1}

else if (*выражение*) {операторы 2}

...

else

{операторы N}

Если условный оператор содержит другие условные операторы, то слово **else** соотносится с ближайшим ключевым словом **if**, еще не связанным ни с каким ключевым словом **else**.

На рис. 5.2 представлена структурная схема работы блока из условных операторов.

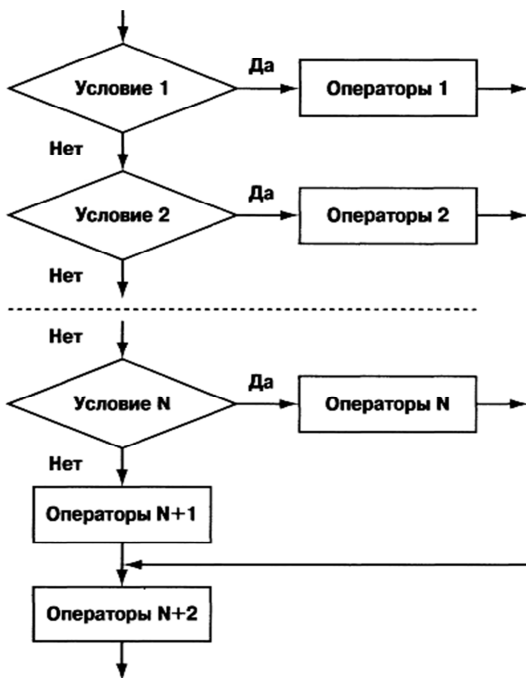


Рис. 5.2. Блок-схема

Пример. Программа, в которой вычисляется значение y в зависимости от переменной x :

$$y = \begin{cases} x + \frac{2}{x}, & x > 0 \\ \frac{4}{x}, & 0 \leq x \leq 1 \\ 2 - x, & \text{если } x < 0 \end{cases} \quad (5.1)$$

```

1  #include <iostream>
2  using namespace std;
3  int main()
4  {float y,x;
5   cout << "Enter x: ";
6   cin >> x; //Ввод с клавиатуры переменной x
7   if (x < 0) //Проверка: если x<0
8   { y = 2-x; } // Этот оператор будет выполнен, если x<0
9   //Следующий оператор выполняется если x больше либо равен 0
10  else if (x <= 1) // Проверка: x<=1
11  { y = 4/x; } // Этот оператор будет выполнен, если x находится
12     // в диапазоне (0,1)
13  else
14  { y = x+2/x; } // Этот оператор будет выполнен, если x>1
15  cout << "y= " << y << endl;
16  return 0; }

```

Рис. 5.3. Код программы

```

Enter x: 0.2 Enter x: -10.55 Enter x: 6.5
y= 20          y= 12.55          y= 6.80769

```

Рис. 5.4. Результат выполнения программы

Пример. Реализация предыдущей задачи с использованием вложенных операторов `if`.

```

1  #include<iostream>
2  int main()
3  { float y,x;
4   std::cout << "Enter x, please:" << std::endl;
5   std::cin >> x;
6   if (x>= 0)
7   { if (x<=1)
8   { y = 4/x; }
9   else
10  { y=x+2/x; }
11  }
12  else
13  { y = 2-x; }
14  std::cout << "y= " << y << std::endl;
15  return 0; }

```

Рис. 5.5. Код программы

Задание

1. Даны два целых числа. Выведите значение наибольшего из них. Если числа равны, выведите любое из них.

2. Даны три натуральных числа A, B, C . Определите, существует ли треугольник с такими сторонами. Если треугольник существует, выведите строку YES, иначе выведите строку NO. Треугольник – это три точки, не лежащие на одной прямой.

3. Написать программу для вычисления значения функции. Протестировать программу при различных значениях аргументов.

$$1. \quad z = \begin{cases} \ln(x), & \text{при } x < -\pi \\ \sin x + \cos 2x, & \text{при } -\pi \leq x < \pi \\ x^3 + 1, & \text{при } \pi \leq x < 10 \\ \frac{x+1}{x^2+8}, & \text{при } 10 \leq x < 100 \\ \ln x, & \text{в остальных случаях} \end{cases}$$

$$2. \quad y = \begin{cases} 0, & \text{при } x \leq 0 \\ 1/x, & \text{при } 0 < x \leq 1 \\ x^2, & \text{при } 1 < x \leq 4 \\ 14 + \log_2 x, & \text{при } x > 4 \end{cases}$$

$$3. \quad y = \begin{cases} \sin x, & \text{при } x \leq 0 \\ \operatorname{arctg} x, & \text{при } 0 < x \leq \pi/4 \\ \log_2 x, & \text{при } \pi/4 < x \leq 32 \\ 1/x, & \text{в остальных случаях} \end{cases}$$

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 6

ОПЕРАТОР ОРГАНИЗАЦИИ ЦИКЛОВ FOR

Цель работы: изучение оператора организации циклов **for**; получение навыков в написании программ с использованием цикла.

Общие сведения

Мы можем организовать повторяющееся выполнение одной и той же последовательности инструкций с помощью специальной конструкции, именуемой *циклом*. Синтаксис оператора **for** имеет вид:

```
for (инициализация переменных; условие; инкремент-изменение переменных)  
  { операторы }
```

Элемент *инициализация* обычно представляет собой инструкцию присваивания, которая устанавливает управляющую переменную цикла равной некоторому начальному значению. Эта переменная действует в качестве счетчика, который управляет работой цикла. Элемент *условие* представляет собой условное выражение, в котором тестируется значение управляющей переменной цикла. По результату этого тестирования определяется, выполнится цикл **for** еще раз или нет. Элемент *инкремент* – это выражение, которое определяет, как изменяется значение управляющей переменной цикла после каждой итерации (т. е. каждого повторения элемента *инструкция*). Цикл **for** будет выполняться до тех пор, пока вычисление элемента *условие* дает истинный результат. Как только это условное выражение станет ложным, цикл завершится, а выполнение программы продолжится с инструкции, следующей за циклом **for**. Схема работы оператора цикла **for** представлена на рис. 6.1

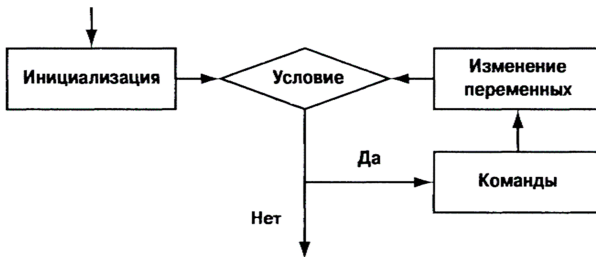
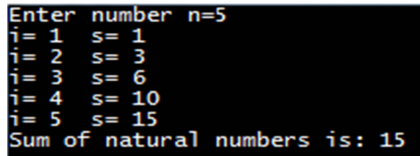


Рис. 6.1. Схема работы оператора цикла **for**

Пример. Вычислить сумму n целых чисел.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  { // Объявление переменных,
5  int n, i, s=0; // причем s инициализируется значением 0
6  cout << "Enter number n=" ; /* Вывод текстового сообщения */
7  cin >> n; // Ввод значения в переменную n
8  for (i=1; i<=n; i++) // Цикл будет выполняться n раз
9  {
10     {s+=i; // Увеличение переменной s на текущее значение i
11     cout << "i= " << i ; // Вывод текущего значения переменной i
12     cout << " s= " << s << endl; //Вывод текущего значения переменной s
13     }
14 // Вывод итогового значения s
15 cout << "Sum of natural numbers is: " << s<< endl;
16 return 0;}
```

Рис. 6.2. Код программы



```
Enter number n=5
i= 1  s= 1
i= 2  s= 3
i= 3  s= 6
i= 4  s= 10
i= 5  s= 15
Sum of natural numbers is: 15
```

Рис. 6.3. Результат выполнения программы

Пояснение к программе. Основу программы составляет оператор цикла, который содержит в первом блоке команду инициализации индексной переменной $i = 1$ начальным единичным значением. Второй блок – проверяемое условие $i \leq n$. Это означает, что оператор цикла выполняется до тех пор, пока индексная переменная i не превышает значения переменной n (значение переменной предварительно вводится с клавиатуры). В третьем блоке указана инструкция $i++$, в силу чего значение индексной переменной на каждом шаге увеличивается на единицу. Наконец, в основном блоке оператора цикла (в фигурных скобках) использована команда $s+=i$, согласно которой на каждом шаге целочисленная переменная s увеличивается на значение индексной переменной i и выводится на экран. Процесс будет продолжаться до тех пор, пока значение индексной переменной не

превысит значения переменной n . Таким образом, после выполнения оператора цикла значение переменной s определяется суммой натуральных чисел от 1 до n .

Управляющая переменная цикла **for** может изменяться как с положительным, так и с отрицательным приращением, причем величина этого приращения может быть любой.

Пример. Вывести в столбец числа от 50 до -50 с шагом 10.

```
1  #include<iostream>
2  int main()
3  {
4  for (int i=50;i>=-50;i-=10)
5  std::cout <<i<< std::endl;
6  return 0;}
```

Рис. 6.4. Код программы

Задание

1. Написать программу, которая будет показывать на экране квадрат числа, введенного пользователем. Пользователь должен сам решать – выйти из программы или продолжить ввод. (Подсказка – необходимо запустить бесконечный цикл, в котором предусмотреть его прерывание при наступлении определенного условия).

2. На складе имеется определенное количество ящиков. Когда подъезжает машина для погрузки, попросить пользователя ввести, сколько ящиков загрузить в первую машину, во вторую и так далее, пока не закончатся ящики. Предусмотреть тот случай, когда пользователь введет количество ящиков больше, чем есть на складе.

3. Вычислить и вывести на экран в виде таблицы значения функции F на интервале от $X_{\text{нач.}}$ до $X_{\text{кон.}}$ с шагом dX . Где a, b, c – действительные числа.

$$1. \quad F = \begin{cases} ax^2 + b, & \text{при } x < 0 \text{ и } b \neq 0 \\ \frac{x-a}{x-c}, & \text{при } x > 0 \text{ и } b = 0 \\ \frac{x}{c}, & \text{в остальных случаях} \end{cases}$$

$$2. \quad F = \begin{cases} ax^2 + b + c, & \text{при } a < 0 \text{ и } c \neq 0 \\ \frac{-a}{x - c}, & \text{при } a > 0 \text{ и } c = 0 \\ a(x + c), & \text{в остальных случаях} \end{cases}$$

$$3. \quad F = \begin{cases} -ax^2, & \text{при } c < 0 \text{ и } a \neq 0 \\ \frac{a - x}{xc}, & \text{при } c > 0 \text{ и } a = 0 \\ \frac{x}{c}, & \text{в остальных случаях} \end{cases}$$

$$4. \quad F = \begin{cases} -ax^2 - b, & \text{при } x < 5 \text{ и } c \neq 0 \\ \frac{x - a}{x}, & \text{при } x > 5 \text{ и } c = 0 \\ \frac{-x}{c}, & \text{в остальных случаях} \end{cases}$$

Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dX$ ввести с клавиатуры.

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программ.

Лабораторная работа № 7

ОПЕРАТОРЫ ЦИКЛОВ WHILE И DO WHILE

Цель работы: изучение оператора организации циклов **while** и **do while**; получение навыков в написании программ с использованием цикла.

Общие сведения

Синтаксис оператора **while** имеет следующий вид:

```
while (условие)  
{ операторы }
```

Сначала проверяется условие, указанное в круглых скобках после ключевого слова **while**. Если условие справедливо, поочередно выполняются операторы, указанные в фигурных скобках. Если оператор один, фигурные скобки можно не указывать. Операторы выполняются до тех пор, пока условие истинно. Значение выражения условия вычисляется перед каждым выполнением операторов цикла, который, таким образом, будет выполняться ноль или более раз (не выполнится ни разу в случае ложности условия при входе в цикл). Схема выполнения оператора цикла **while** показана на рис. 7.1.

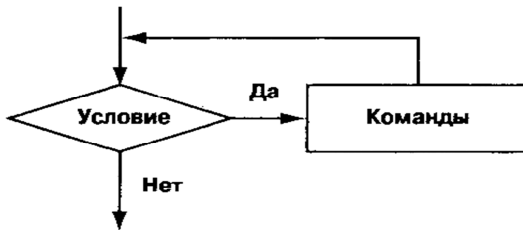


Рис. 7.1. Схема выполнения оператора цикла **while**

Пример. Программа для вычисления суммы целых чисел с помощью оператора цикла **while**.

Результат выполнения программы будет иметь вид:

```

1 | #include <iostream>
2 | using namespace std;
3 | int main () {
4 |     int n, i=1,s=0;
5 |     cout << "Enter number n=";
6 |     cin >> n;
7 |     // Операторы цикла будут выполняться пока i<=n
8 |     while (i<=n){
9 |         s+=i;
10 |        i++;
11 |        cout<< "Sum of natural numbers is: "<< s;
12 |        return 0;}

```

Рис. 7.2. Код программы

Синтаксис оператора **do while**:

```

do {
    операторы
}
while (условие)

```

В операторе цикла **do while** выполняемые операторы (заключенные в фигурные скобки) указываются после ключевого слова **do**. Далее проверяется условие, указанное в круглых скобках. Если условие выполняется, снова выполняются команды после ключевого слова **do** и т. д. Операторы выполняются до тех пор, пока условие истинно. Значение выражения условия вычисляется и анализируется после каждого выполнения операторов цикла, которые будут выполнены один или более раз. Схема выполнения оператора цикла **do while** показана на рис. 7.3.

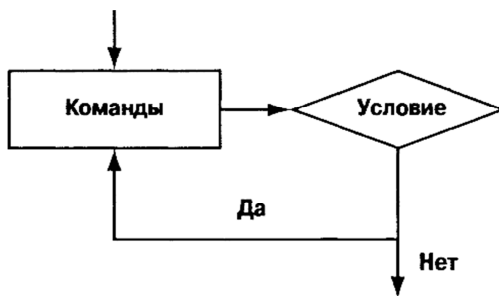


Рис. 7.3. Схема выполнения оператора цикла **do while**

Принципиальная разница между операторами **while** и **do while** состоит в том, что в первом случае сначала проверяется условие, а затем (если верно условие) выполняются операторы. Во втором случае сначала по крайней мере один раз выполняются операторы, а затем проверяется условие. По сравнению с оператором цикла **for**, операторы **while** и **do while** требуют от программиста большей ответственности в первую очередь в плане детальной проработки механизма изменения значения проверяемого условия в процессе выполнения команд основного блока оператора. Программный код должен быть составлен корректно, чтобы не получить бесконечный цикл.

Пример. Программа для вычисления суммы целых чисел с помощью оператора цикла **do while**.

```
1  #include <iostream>
2  using namespace std;
3  int main () {
4      int n,i=1,s=0;
5      cout << "Enter number n=";
6      cin >> n;
7      do {
8          s+=i;
9          i++; }
10     while(i<=n);
11     cout<< "Summa of natural numbers is: "<< s;
12     return 0; }
```

Рис. 7.4. Код программы

```
Enter number n=5
Sum of natural numbers is: 15
```

Рис. 7.5. Результат выполнения программы

Пояснение к программам. Разница между двумя программами с циклами **while** и **do while** все же есть. Если пользователь введет, например, отрицательное число (значение переменной n), то первая программа в качестве значения суммы укажет 0, в то время как во втором случае будет выведена 1. Причина в том, что в цикле **while** при ложном проверяемом условии команды оператора цикла не выполняются и в качестве значения суммы возвращается начальное нулевое значение переменной s . Во втором случае сначала выпол-

няется один цикл и уже после этого проверяется условие. За этот один выполненный цикл значение переменной s увеличивается на 1, и в результате программой для суммы натуральных чисел возвращается единичное значение.

Задание

1. Напишите программу, которая будет считать сумму всех чисел от 1 до 1000.

2. По данному целому числу N распечатайте все квадраты натуральных чисел, не превосходящие N , в порядке возрастания. Пример: на ввод дано число 50, следовательно, программа должна вывести числа 1 4 9 16 25 36 49.

3. Программа получает на вход последовательность целых неотрицательных чисел, каждое число записано в отдельной строке. Последовательность завершается числом 0, при считывании которого программа должна закончить свою работу и вывести количество членов последовательности (не считая завершающего числа 0). Числа, следующие за числом 0, считывать не нужно. Пример: на ввод дается последовательность чисел 1 7 9 0 5, следовательно, ответом будет число 3.

4. Вычислить приближенное значение бесконечной суммы с точностью до $\varepsilon = 0,0001$. $1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} + \dots$

5. Вычислить приближенное значение бесконечной суммы с точностью до $\varepsilon = 0,001$. $\frac{1}{1^2} + \frac{1}{3^2} + \dots + \frac{1}{(2n+1)^2} + \dots$

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 8

ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ. СОРТИРОВКА МАССИВОВ

Цель работы: получение практических навыков в работе с одномерными массивами; знакомство с алгоритмами упорядочения.

Общие сведения

Массив – это последовательность переменных одного типа, использующая одно имя. Обращение к элементам массива осуществляется с помощью индексов. **Индекс** – ссылка на конкретное значение в массиве, то есть индекс определяет позицию элемента внутри массива. В C++ все массивы используют ноль в качестве индекса своего первого элемента. Массивы удобны для хранения большого количества взаимосвязанных значений.

Объявление одномерного массива имеет следующий вид:

```
тип_массива имя_массива [размер]
```

```
int sample[10];
```

Здесь *тип* объявляет базовый тип массива и является типом каждого элемента массива. Параметр *размер* определяет, сколько элементов содержит массив.

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива. В C++ первый элемент массива имеет нулевой индекс. Поскольку массив `sample` содержит 10 элементов, его индексы изменяются от 0 до 9. Чтобы получить доступ к элементу массива по индексу, достаточно указать нужный номер элемента в квадратных скобках. Так, первым элементом массива `sample` является `sample [0]`, а последним – `sample [9]`. Например, следующая программа помещает в массив `sample` числа от 0 до 9.

```
#include <iostream>
using namespace std;
int main()
{
    int sample[10]; // Эта инструкция резервирует область
```

```

// памяти для 10 элементов типа int.
int t;
// Помещаем в массив значения.
for(t = 0; t < 10; ++t) sample[t] = t; // Обратите внимание
// на то, как индексируется
// массив sample.
// Отображаем содержимое массива.
for(t = 0; t < 10; ++t)
cout << "Это элемент sample[" << t << "]: "
<< sample[t] << "\n" ;
return 0;
}

```

Результаты выполнения этой программы выглядят так.

```

Это элемент sample[0]
Это элемент sample[1]
Это элемент sample[2]
.....
Это элемент sample[9]

```

Элементы массива в памяти расположены последовательно друг за другом. Ячейка с наименьшим адресом относится к первому элементу массива, а с наибольшим – к последнему. Например, после выполнения этого фрагмента кода

```

Int nums[5];
int i;
for(i = 0; i < 5; i++) nums[i] = i;
массив nums будет выглядеть следующим образом 0; 1; 2; 3; 4;
/*

```

*Вычисление среднего и определение минимального и максимального из набора значений. */*

```

#include <iostream>
using namespace std;
int main()
{
int i, avg, min_val, max_val;
int nums[10];
nums[0] = 10; nums[1] = 18; nums[2] = 75; nums[3] = 0; nums[4] = 1;
nums[5] = 56; nums[6] = 100; nums[7] = 12; nums[8] = -19; nums[9] = 88;
// Вычисление среднего значения.

```

```

avg = 0;
for(i = 0; i < 10; i++)
avg += nums[i]; // ← Суммируем значения массива nums. j

avg /= 10; // ← Вычисляем среднее значение.
cout << "Среднее значение равно " << avg << "\n";
// Определение минимального и максимального значений.
min_val = max_val = nums[0];
for(i = 1; i < 10; i++) {
if(nums[i] < min_val) min_val = nums[i]; // минимум
if(nums[i] > max_val) max_val = nums[i]; // максимум
cout << "Минимальное значение: " << min_val << "\n";
cout << "Максимальное значение: " << max_val << "\n";
return 0;
}

```

При выполнении эта программа генерирует такие результаты.

Среднее значение равно 34

Минимальное значение: -19

Максимальное значение: 100

Инициализация одномерных массивов

В С++ предусмотрено несколько способов инициализации массивов. **1-й способ – инициализация при объявлении.** Он предусматривает следующие варианты:

а) инициализация при объявлении с указанием размерности, например:

```
int my_array[3] = {12, 1245, 18094};
```

```
//или
```

```
const int n = 3;
```

```
int my_array [n] = {12, 1245, 18094};
```

б) Объявление с инициализацией без указания размерности, например:

```
/* компилятор сам определяет размерность массива в процессе компиляции */
```

```
int my_array []={12, 1245, 18094};
```

в) Объявление с инициализацией не всех элементов, например:
 /* в данном случае компилятор инициализирует нулями два оставшихся элемента массива*/

```
int my_array [5] = {12, 1245, 18094};
```

2-й способ – инициализация в цикле. Для этого используются следующие конструкции:

```
тип имя_массива[размер];
```

```
for (i = 0; i < размер; i++)
```

```
cin >> имя_массива[i]; или
```

```
тип имя_массива[размер];
```

```
for (i = 0; i < размер; i++) имя_массива[i] = значение;
```

Первая позволяет вводить элементы массива с клавиатуры, вторая – присваивать значения, полученные, например, в процессе вычисления некоторого выражения.

Сортировка.

Рассмотрим массив целых или вещественных чисел a_1, a_2, \dots, a_n . Пусть требуется переставить элементы этого массива так, чтобы после перестановки они были упорядочены по неубыванию $a_1 \leq a_2 \leq \dots \leq a_n$ или по невозрастанию $a_1 \geq a_2 \geq \dots \geq a_n$. Эта задача называется задачей сортировки или упорядочения массива. Для решения этой задачи можно воспользоваться, например, следующими алгоритмами:

1. Найти элемент массива, имеющий наименьшее (наибольшее) значение, переставить его с первым элементом. Затем проделать то же самое, начав со второго элемента, и так далее (сортировка выбором).

2. Последовательным просмотром чисел a_1, a_2, \dots, a_n найти наименьшее i , такое, что $a_i > a_{i+1}$ или $a_i < a_{i+1}$. Поменять a_i и a_{i+1} местами, возобновить просмотр с элемента a_{i+1} и так далее. Тем самым наибольшее или наименьшее число передвинется на последнее место. Следующие просмотры следует начинать опять сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы (сортировка обменами).

3. Просматривать последовательно a_2, \dots, a_n и каждый новый элемент вставлять на подходящее место в уже упорядоченную последовательность a_1, \dots, a_{i-1} . Это место определяется последовательным сравнением a_i с упорядоченными элементами a_1, \dots, a_{i-1} (сортировка простыми вставками).

4. Сравнить элементы a_1 и a_2 и, если $a_1 > a_2$ (или $a_1 < a_2$), то эти элементы переставить. Далее сравнить элементы a_2 и a_3 и, если $a_2 > a_3$ (или $a_2 < a_3$), то их переставить. Далее сравнить элементы a_3

и a_4 и так далее до элементов a_{n-1} и a_n включительно. Далее эти действия повторить, начиная опять с первого элемента. Последним является контрольный проход, при котором не будет перестановок элементов (**сортировка по методу пузырька**).

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main() {
int nums[10]; int a, b, t; int size;
size = 10; // Количество сортируемых элементов.
// Помещаем в массив случайные числа.
for(t = 0; t < size; t++) nums[t] = rand();
// Отображаем исходный массив.
cout << "Исходный массив:\n ";
for(t = 0; t < size; t++) cout << nums[t] << ' ';
cout << "\n";
// Реализация алгоритма пузырьковой сортировки.
for(a = 1; a < size; a++)
for (b = size-1; b >= a; b--) {
if(nums[b-1] > nums[b]) { // Если значения элементов массива
расположены не по порядку, то меняем их местами.
t = nums[b-1];
nums[b-1] = nums[b];
nums[b] = t; } } // Конец пузырьковой сортировки
// Отображаем отсортированный массив
cout << "\n Отсортированный массив:\n ";
for (t = 0; t < size ; t++) cout << nums[t] << ' ';
return 0;}
```

Задание

1. Выведите все элементы массива с четными индексами (то есть $A[0]$, $A[2]$, $A[4]$, ...).

2. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

- сумму отрицательных элементов массива;
- произведение элементов массива, расположенных между максимальным и минимальным элементами.

3. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

– сумму положительных элементов массива;

– произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами. Упорядочить элементы массива по убыванию.

4. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

– сумму элементов массива с нечетными номерами;

– сумму элементов массива, расположенных между первым и последним отрицательными элементами.

5. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

– количество элементов массива, равных 0;

– сумму элементов массива, расположенных после минимального элемента. Упорядочить элементы массива по возрастанию модулей элементов.

6. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

– произведение положительных элементов массива;

– сумму элементов массива, расположенных до минимального элемента.

Содержание отчета

1. Тема и цель работы.

2. Схема алгоритма решения.

3. Текст и результаты выполнения программ.

Лабораторная работа № 9

ОБРАБОТКА ДВУХМЕРНЫХ МАССИВОВ

Цель работы: получение практических навыков в работе с двухмерными массивами.

Общие сведения

Массив – это совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива может осуществляться с помощью индекса.

Простейшим видом многомерного массива является двухмерный массив. **Двухмерный массив** – это массив одномерных массивов. Двухмерный массив объявляется следующим образом:

тип имя_массива[размер1][размер2];

Следовательно, для объявления двухмерного массива целых чисел с размером 10 и 20 следует написать: **int d[10][20];**

Для доступа к элементу в 3 строке 5 столбце массива **d** следует использовать **d[2][4]** В массивах индексация начинается с нуля. Двухмерные массивы можно представить в виде матрицы, где первый индекс отвечает за строку, а второй – за столбец.

Пример. В двумерный массив помещаются последовательные числа от 1 до 12.

```
#include <iostream>
using namespace std;
int main()
{
    int t, i, nums [3] [4] ;
    for(t = 0; t < 3; ++t) {
        for(i = 0; i < 4; ++i) {
            nums[t][i] = (t * 4) + i + 1; // Для индексации массива
            // nums требуется два индекса.
            cout << nums[t][i] << ' ';
        }
        cout << '\n' ;
    }
    return 0;
}
```

В этом примере элемент numS [0] [0] получит значение 1, элемент numS [0][1] – значение 2, элемент numS [0] [2] – значение 3 и т. д. Значение элемента numS [2] [3] будет равно числу 12.

В C++, помимо двумерных, можно определять массивы трех и более измерений. Вот как объявляется многомерный массив.

тип имя[размер1] [размер2] ... [размеры];

Многомерные массивы **инициализируются** так же, как и одномерные.

1-й способ – инициализация при объявлении:

а) объявление с указанием размерности, например:

```
int my_array [4] [3] = { {12, 1245, 18094}, {2, 518, 45},  
{45, 67, 45}, {34, 562, 98} };
```

/ если опустить внутренние фигурные скобки, то результат не изменится. В любом случае, сначала будут заполняться строки, а затем столбцы */*

//или

```
const int n = 4;
```

```
const int m = 3;
```

```
int my_array [n] [m] = { {12, 1245, 18094}, {2, 518, 45},  
{45, 67, 45}, {34, 562, 98} };
```

б) объявление с инициализацией без указания первой размерности, например:

```
int tu_array [ ][3] = { {12, 1245, 18094}, {2, 518, 56},  
{45, 67, 45}, {34, 562, 98} };
```

/ Можно не указывать размерность основного массива. При этом размерности вложенных массивов должны быть указаны. */*

в) объявление с инициализацией не всех элементов, например:

```
int my_array [4] [3] = { {12, 1245, 18094}, {2, 518},  
{45, 67, 45}, {34, 98} };
```

/ Можно пропустить инициализацию значений в любой строке, при этом значение автоматически будет инициализировано нулем */*

Пример. Объявить массив размерностью 3×4 и ввести его значения с клавиатуры.

Обратите внимание на использование вложенных циклов **for** при инициализации многомерных массивов. Массив инициализации рисуется построчно.

```

1  #include<iostream>
2  #include<iomanip>
3  int main()
4  {
5      const int row = 3;
6      const int column = 4;
7      int data [row][column];
8      for (int i = 0; i < row; i++)
9      {
10         for (int j = 0; j < column; j++)
11         { std::cout<< "Enter data["<<i<<"][" << j<< "]=";
12           std::cin >> data [i][j]; }
13     }
14     for (int i = 0; i < row; i++)
15     {
16         for (int j = 0; j < column; j++)
17         {
18             std::cout << std::setw(6) << data [i][j]; }
19             std::cout << std::endl;
20         }
21     return 0; }

```

Рис. 9.1. Код программы

Задание

1. Программа получает на вход размеры массива n и m , затем n строк по m чисел в каждой. n и m не превышают 100. Выведите два числа: номер строки и номер столбца, в которых стоит наибольший элемент в двумерном массиве. Если таких элементов несколько, то выводится тот, у которого меньше номер строки, а если номера строк равны, то тот, у которого меньше номер столбца.

2. Дана целочисленная прямоугольная матрица. Определить количество строк, не содержащих ни одного нулевого элемента;

3. Найти наибольший элемент матрицы $A(5 \times 3)$ и номер строки и столбца, в котором он находится.

4. Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента.

5. Дана целочисленная квадратная матрица. Определить произведение элементов в тех строках, которые не содержат отрицательных элементов.

6. Дана целочисленная квадратная матрица. Определить сумму элементов в тех столбцах, которые не содержат отрицательных элементов.

7. Найти наибольший элемент главной диагонали матрицы $A(4 \times 4)$ и вывести на экран всю строку, в которой он находится.

8. Дана целочисленная прямоугольная матрица. Определить:

– количество строк, содержащих хотя бы один нулевой элемент;

– номер столбца, в котором находится самая длинная серия одинаковых элементов.

9. Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 10

ОБРАБОТКА МАТРИЦ В C++

Цель работы: получение навыков программирования некоторых операций над матрицами.

Общие сведения

Матрица – это двумерный массив, каждый элемент которого имеет два индекса: номер строки и номер столбца, поэтому для работы с элементами матрицы необходимо использовать два цикла. Если значениями параметра первого цикла будут номера строк матрицы, то значениями параметрами второго – столбцы (или наоборот). Обработка матрицы заключается в том, что вначале поочередно рассматриваются элементы первой строки (столбца), затем второй и т. д. до последней.

Перед тем, как приступить к изучению алгоритмов обработки матриц, рассмотрим, как описываются матрицы в C++. Двумерный массив можно объявить так:

тип имя_переменной [n] [m];

Здесь ***тип*** определяет тип элементов массива, ***имя_переменной*** – имя матрицы, ***n*** – количество строк, ***m*** – количество столбцов. Строки нумеруются от 0 до $n - 1$, столбцы от 0 до $m - 1$.

Например, **`int h[10] [15];`**

Выше описана матрица целых чисел ***h***, состоящая из 10 строк и 15 столбцов (строки нумеруются от 0 до 9, столбцы от 0 до 14).

Для обращения к элементу матрицы необходимо указать ее имя и в квадратных скобках номер строки, затем номер столбца. Например, **`h[2] [5]`**.

Ввод-вывод матриц.

Матрицы, как и одномерные массивы, нужно вводить (выводить) поэлементно. Блок-схема ввода элементов матрицы $A[n] [m]$ изображена ниже:

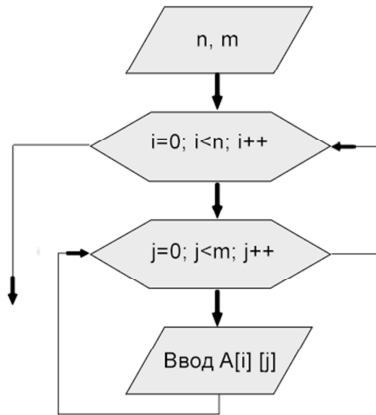


Рис. 10.1. Блок-схема

Код программы на Visual C++ ввода-вывода матрицы будет иметь примерно такой вид:

```

#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, "RUS");
    int i,j,N,M,a[20][20];
    cout<<"N ="; // ввод количества строк
    cin>>N;
    cout<<"M ="; // ввод количества столбцов
    cin>>M;
    cout<<"Input matrix A \n";
    // цикл по переменной i, в которой перебираем строки матрицы
    for (i = 0; i < N; i++)
        // цикл по переменной j, в котором перебираем элементы внутри
        строки
        for (j = 0; j < M; j++)
            cin>>a[i][j]; //ввод очередного элемента матрицы
    cout<<"matrix A \n";
    for (i = 0; i < N; i++)
    {

```



```

// цикл по переменной i, в котором перебираем строки матрицы
for (j = 0; j < M; j++)
cout<<a[i][j]<<"\t"; // вывод очередного элемента матрицы
cout<<endl; // переход на новую строку после вывода всех эле-
ментов строки
}
system("pause");
return 0;
}

```

```

N=3
M=4
Input matrix A
32
65
5
-5
45
76
4
3
2
6
7
8
matrix A
32      65      5      -5
45      76      4      3
2       6       7      8
Для продолжения нажмите любую клавишу . . . Kvodo.ru

```

Рис. 10.2. Результат работы программы

Некоторые свойства матриц:

- если номер строки элемента совпадает с номером столбца ($i = j$), это означает, что элемент лежит на главной диагонали матрицы;
- если номер строки превышает номер столбца ($i > j$), то элемент находится ниже главной диагонали;
- если номер столбца больше номера строки ($i < j$), то элемент находится выше главной диагонали;
- элемент лежит на побочной диагонали, если его индексы удовлетворяют равенству $i + j + 1 = n$;
- неравенство $i + j + 1 < n$ характерно для элемента, находящегося выше побочной диагонали;
- элементу, лежащему ниже побочной диагонали, соответствует выражение $i + j + 1 > n$;

– матрица называется *единичной*, если все элементы нули, а на главной диагонали единицы;

– матрица называется *диагональной*, если все элементы нули, кроме главной диагонали;

– матрица *нулевая*, если все элементы нули;

– матрица называется *симметричной*, если $A = AT$.

Для проверки, что матрица B является обратной матрице A , нужно проверить условие $A \times B = E$ (E – единичная матрица).

Пример. Найти сумму элементов матрицы, лежащих выше главной диагонали.

Алгоритм решения данной задачи можно построить следующим образом: обнулить ячейку для накапливания суммы (переменная S). Затем с помощью двух циклов просмотреть каждый элемент матрицы, суммировать элементы при условии, что элемент находится выше главной диагонали. Текст решения задачи:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    setlocale (LC_ALL, "RUS"); // Отображение кириллицы
    int S, i, j, N, M, a[20][20];
    cout<<"N ="; cin>>N;
    cout<<"M ="; cin>>M;
    cout<<"Введите матрицу A \n";
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            cin>>a[i][j];
    for (S = i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            // если элемент лежит выше главной диагонали, то нарачиваем
            // сумму
            if (j > i) S += a[i][j];
    cout<<"S ="<<S<<endl;
    system("pause");
    return 0;
}
```

Задание

1. Вычислить количество положительных элементов квадратной матрицы, расположенных по ее периметру и на диагоналях.

2. Проверить, является ли заданная квадратная матрица единичной. Единичной называют матрицу, у которой элементы главной диагонали – единицы, а все остальные – нули.

3. Поменять местами элементы главной и побочной диагонали матрицы $A(k, k)$. Алгоритм сводится к обмену элементов на главной и побочной диагоналях в каждой строке

4. Преобразовать исходную матрицу $A(N, M)$ так, чтобы нулевой элемент каждой строки был заменен средним арифметическим элементов этой строки. Необходимо в каждой строке матрицы найти сумму элементов, разделить на количество элементов в строке и полученный результат записать в первый элемент строки.

5. Преобразовать исходную матрицу так, чтобы нулевой элемент каждой строки был заменен средним арифметическим элементом этой строки.

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 11

ОБЪЯВЛЕНИЕ, ОПРЕДЕЛЕНИЕ И ВЫЗОВ ФУНКЦИИ

Цель работы: получение практических навыков в работе с функциями.

Общие сведения

Функция – это именованный программный код, выполняющий какое-либо законченное действие, который может многократно вызываться в программе. Любая программа на C++ состоит из функций, одна из которых должна иметь имя **main** (с нее начинается выполнение программы). Функция начинает выполняться в момент *вызова*. Любая функция должна быть *объявлена* и *определена*. Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова. Структура программы с использованием функций:

#include подключение библиотек, файлов *объявление функции*

int main() {

объявление локальных переменных

...

вызов функции

...

}

определение функции

Объявление функции (*прототип, заголовок, сигнатура*) задает ее имя, тип возвращаемого значения и список передаваемых параметров.

тип имя ([список_параметров]);

Определение функции содержит, кроме объявления, *тело* функции, представляющее собой последовательность операторов и операций в фигурных скобках:

тип имя ([список_параметров])

{ тело функции:

1) объявление локальных переменных

2) операторы

3) return [выражение]

}

Рассмотрим составные части определения.

– тип возвращаемого функцией значения может быть любым. Если функция не должна возвращать значение, указывается тип **void**;

– список параметров определяет величины, которые требуется передать в функцию при ее вызове. Элементы списка параметров разделяются запятыми, и для каждого параметра указывается его тип и имя. Функция также может не иметь параметров, тогда указываются просто пустые скобки ();

– **return** служит 1) для выхода из функции и 2) возврата значения в вызвавшую ее функцию. Если функция описана как **void**, выражение не указывается. Выражение, указанное после **return**, преобразуется к типу возвращаемого функцией значения и передается в точку вызова.

Для **вызова** функции необходимо указать ее имя, и в круглых скобках через запятую передать ей набор аргументов в соответствии с параметрами, указанными в заголовке функции.

имя (список аргументов);

Если тип возвращаемого функцией значения не **void**, она может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать.

Пример. Создание и использование в программе функции *sumfunct*, вычисляющей произведение двух чисел.

```
1  #include <iostream>
2  using namespace std;
3  float sumfunct(int a, int b) // Объявление функции sumfunct
4  {                               // с параметрами a и b типа int
5      int y = b*a;
6      return y; // функция возвращает значение y
7  }
8  int main()
9  { int rez;
10     rez = sumfunct (78,5);/* Переменной rez присваивается значение,
11        возвращаемое функцией sumfunct, вызванной с аргументами 78 и 5 */
12        // Число 78 присваивается параметру a функции
13        // Число 5 присваивается параметру b функции
14     cout << "rez = " << rez;
15     return 0;}
```

Рис. 11.1. Код программы

Инструкция **return** имеет две формы применения: первая позволяет возвращать значение из функции, вторая – нет.

Версия инструкции return, которая не возвращает значение.

Если тип значения, возвращаемого функцией, определяется ключевым словом **void** (то есть функция не возвращает значения вообще), то для выхода из функции достаточно использовать такую форму инструкции **return**:

return;

При обнаружении инструкции **return** управление программой немедленно передается инициатору ее вызова. Любой код в функции, расположенный за инструкцией **return**, игнорируется.

Пример. Функция *power()*, показанная в следующей программе, отображает результат возведения целочисленного значения в положительную целую степень. Если показатель степени окажется отрицательным, инструкция **return** немедленно завершит эту функцию еще до попытки вычислить результат.

```
1  #include <iostream>
2  void power(int base, int exp); // Прототип функции power
3  int main()
4  {int a =10; int b =2;
5   power(a, b); // Вызов функции power
6   power(a, -b) ; // Вызов функции power
7   return 0;
8  }
9  void power (int base, int exp)
10 { if (exp<0) return; /* Если показатель степени отрицательный,
11 |                 то - выход из функции */
12 | /* Если показатель степени положительный, то будут выполнены
13 |    следующие операторы */
14   int y=1;
15   while (exp!=0)
16   {y*=base; // Возводим целое число в положительную степень
17   exp--;}
18   std::cout << "Rezultat = " << y; }
```

Рис. 11.2. Код программы

Если значение параметра *exp* отрицательно (как при втором вызове функции *power*), вся оставшаяся часть функции опускается.

Функция может содержать несколько инструкций **return**. В этом случае возврат из функции будет выполнен при обнаружении одной

из них. Однако следует иметь в виду, что слишком большое количество инструкций **return** может ухудшить ясность алгоритма и ввести в заблуждение тех, кто будет в нем разбираться. Несколько инструкций **return** стоит использовать только в том случае, если они способствуют ясности функции

Функция может *возвращать значение* инициатору своего вызова. Таким образом, возвращаемое функцией значение – это средство получения информации из функции. Для возврата функцией значения используется вторая форма инструкции **return**:

return значение;

Эту форму инструкции **return** нельзя использовать с **void**-функциями.

Функция, которая возвращает значение, должна определить тип этого значения. Тип возвращаемого значения должен быть совместимым с типом данных, используемых в инструкции **return**. В противном случае неминуема ошибка во время компиляции программы. Функция может возвращать данные любого допустимого в C++ типа, за исключением массива.

Пример. Перепишем уже известную функцию *power*. В этой версии функция *power* возвращает значение.

```
1  #include <iostream>
2  int power(int base, int exp); // Прототип функции power
3  int main()
4  {int a =10; int b =2; int answer;
5   |answer = power(a, b); // Переменной answer присваивается
6   |                        //значение, возвращаемое функцией
7   |std::cout << "Answer is " << answer;
8   |return 0;}
9  int power (int base, int exp) // Описание функции
10 { if (exp<0)
11  |{ return 0; /* Если показатель степени отрицательный,
12  |             то - выход из функции */
13  |}
14  |/* Если показатель степени положительный, то будут выполнены
15  |   следующие операторы */
16  |int y=1;
17  |while (exp!=0)
18  |{y*=base; // Возводим целое число в положительную степень
19  |exp--;}
20  |return y;}
```

Рис. 11.3. Код программы

Факториал. Факториалом числа N называется произведение всех чисел от 1 до N . Например, $4! = 1 \times 2 \times 3 \times 4 = 24$.

Пример. Написать программу для вычисления числа вычитаний:

$$C_n^m = \frac{n!}{m!(n-m)!}$$

```
#include <iostream.h>
int fact(int );
int main()
{
int n, m,c; cin >>n;
cin >>m;
cout <<"C = "<< fact(n)/(fact(m)*fact(n-m)); // вызов функции
return 0;
}
int fact(int a) // определение функции
{
int i, p;
if(a == 0|| a == 1) return 1;
for(i = 1; i <= n; i++)
p = p*i;
return p; // Возврат значения p в точку вызова
}
```

Параметры функции. Использование параметров является основным способом обмена информацией между вызываемой и вызывающей функциями. Параметры, перечисленные в заголовке описания функции, называются *формальными*, а записанные в операторе вызова функции – *фактическими*.

При вызове функции в первую очередь вычисляются выражения, стоящие на месте фактических параметров; затем выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего фактического параметра. Существует два способа передачи параметров в функцию: по значению и по адресу.

Передача по значению.

Синтаксис при вызове имя_ф(имя_фактического_параметра);
при определении и объявлении тип имя_ф(тип имя_формального_параметра);

При передаче по значению в стек заносятся копии значений фактических параметров, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, и поэтому при изменении формальных параметров фактические параметры не изменяются.

Используется два синтаксиса.

1. С помощью ссылки.

при вызове имя_ф(имя факт. парам);

при определении и объявлении тип имя_ф(тип &имя форм параметра);

2. С помощью указателя.

при вызове имя_ф(&имя факт. парам);

при определении и объявлении тип имя_ф(тип *имя форм параметра);

При передаче по адресу в стек заносятся копии адресов фактических параметров, а функция осуществляет доступ к ячейкам памяти по этим адресам, т. е. при изменении значений формальных параметров значения фактических параметров также изменяются.

При передаче по адресу в качестве фактических параметров нельзя использовать выражения – только имена переменных.

```
#include <iostream.h>
```

```
void f(int , int* . int& ):
```

```
int main(){
```

```
int i = 1. j = 2, k = 3;
```

```
cout <<"i j k\n":
```

```
cout << i <<' '<< j <<' '<< k <<\n'; //1 2 3
```

```
f(i, &j, k);
```

```
cout << i <<' '<< j <<' '<< k <<\n'; //1 3 4
```

```
return 0;}
```

```
void f(int i. int* j. int& k){
```

```
i++; (*j)++; k++;}
```

Задание

1. Даны отрезки a , b , c и d . Для каждой тройки этих отрезков, из которых можно построить треугольник, вычислить площадь этого треугольника и вывести на экран.

2. Даны действительные числа x , y , z . Вычислить выражение:

$$\frac{\max(x, y, z)}{\max(x^2, y^2, z^2)} + \frac{\min(x, y, z)}{\min(x^2, y^2, z^2)}.$$

3. Даны действительные числа a , b , c . Вычислить:

$$\frac{\max(a, a \cdot b) + \max(a \cdot b, c)}{1 + \max(a + bc, 1, 15)}.$$

4. Даны длины a , b и c сторон некоторого треугольника. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. (Замечание: длина медианы, проведенной к стороне a , равна $0,5\sqrt{2b^2 + 2c^2 - a^2}$).

5. Даны действительные числа a_0, a_1, a_2, a_3 . Получить для $x = 1, 2, 3, 4$ значения $p(x+1) - p(x)$, где $p(y) = a_3y^3 + a_2y^2 + a_1y + a_0$.

6. Напишите функцию $\min(a, b)$, вычисляющую минимум двух чисел. Затем напишите функцию $\min4(a, b, c, d)$, вычисляющую минимум 4 чисел с помощью функции \min . Считайте четыре целых числа и выведите их минимум.

7. Даны четыре действительных числа: x_1, y_1, x_2, y_2 . Напишите функцию $\text{distance}(x_1, y_1, x_2, y_2)$, вычисляющую расстояние между точкой (x_1, y_1) и (x_2, y_2) . Считайте четыре действительных числа и выведите результат работы этой функции.

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

Лабораторная работа № 12

СТРОКИ И СИМВОЛЫ

Цель работы: получение практических навыков в работе со строками; изучение функций в работе со строками

Общие сведения

Символ – элементарная единица, некоторый набор которых несет определенный смысл. В языке программирования C++ предусмотрено использование символьных констант. Символьная константа – это целочисленное значение (типа int), представленное в виде символа, заключенного в одинарные кавычки, например 'a'.

Пример объявления символьной переменной:

```
char symbol = 'a';
```

// где symbol – имя переменной типа char

char – тип данных для хранения символов

Символьные строки состоят из набора символьных констант, заключенных в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-терминатора и отводить дополнительный байт под него.

\0 – символ нуль-терминатора.

Пример объявления строки

```
char string[10];
```

// где string – имя строковой переменной, 10 – размер массива, то есть в данной строке может поместиться 9 символов, последнее место отводится под нуль-терминатор.

Строка при объявлении может быть инициализирована начальным значением, например, так: `char string[10] = "abcdefghf";`

Посимвольная инициализация строки:

```
char string[10] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'f', '\0'};
```

При объявлении строки не обязательно указывать ее размер, но при этом обязательно нужно ее инициализировать начальным значением. Тогда размер строки определится автоматически.

Инициализация строки без указания размера:

```
char string[] = "abcdefghf";
```

Строка может содержать символы, цифры и специальные знаки. В C++ строки заключаются в двойные кавычки. Имя строки является константным указателем на первый символ. Разработаем программу с использованием строк.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
    char string[] = "this is string - "; // объявление и инициализация
    строки
    cout << "Enter the string: ";
    char in_string[500]; // строковый массив для ввода
    gets(in_string); // функция gets() считывает все введенные сим-
    волы с пробелами до тех пор, пока не будет нажата клавиша Enter
    cout << string << in_string << endl; // вывод строкового значения
    system("pause");
    return 0;
}
```

С помощью функции *gets()* считаются все введенные символы с пробелами до тех пор, пока во вводимом потоке не встретится код клавиши Enter. Если использовать операцию *cin*, то из всего введенного считается последовательность символов до первого пробела.

Таблица 12.1

Функции для работы со строками и символами

Функция	Пояснение
<code>strlen(имя_строки)</code>	определяет длину указанной строки, без учета нуль-символа
<code>strcpy(s1, s2)</code>	выполняет побайтное копирование символов из строки <i>s2</i> в строку <i>s1</i>
<code>strncpy(s1, s2, n)</code>	выполняет побайтное копирование <i>n</i> символов из строки <i>s2</i> в строку <i>s1</i> , возвращает значения <i>s1</i>
<code>strcat(s1, s2)</code>	объединяет строку <i>s2</i> со строкой <i>s1</i> . Результат сохраняется в <i>s1</i>
<code>strncat(s1, s2, n)</code>	объединяет <i>n</i> символов строки <i>s2</i> со строкой <i>s1</i> . Результат сохраняется в <i>s1</i>
<code>strcmp(s1, s2)</code>	сравнивает строку <i>s1</i> со строкой <i>s2</i> и возвращает результат типа <code>int</code> : 0 – если строки эквивалентны, > 0 – если <i>s1</i> < <i>s2</i> , < 0 – если <i>s1</i> > <i>s2</i> (с учетом регистра)

Функция	Пояснение
strncmp(s1, s2, n)	сравнивает n символов строки $s1$ со строкой $s2$ и возвращает результат типа <code>int</code> : 0 – если строки эквивалентны, > 0 – если $s1 < s2$, < 0 – если $s1 > s2$ (с учетом регистра)
stricmp(s1, s2)	сравнивает строку $s1$ со строкой $s2$ и возвращает результат типа <code>int</code> : 0 – если строки эквивалентны, > 0 – если $s1 < s2$, < 0 – если $s1 > s2$ (без учета регистра)
strnicmp(s1, s2, n)	сравнивает n символов строки $s1$ со строкой $s2$ и возвращает результат типа <code>int</code> : 0 – если строки эквивалентны, > 0 – если $s1 < s2$, < 0 – если $s1 > s2$ (без учета регистра)
isalnum(c)	возвращает значение <code>true</code> , если c является буквой или цифрой, и <code>false</code> в других случаях
isalpha(c)	возвращает значение <code>true</code> , если c является буквой, и <code>false</code> в других случаях
isdigit(c)	возвращает значение <code>true</code> , если c является цифрой, и <code>false</code> в других случаях
islower(c)	возвращает значение <code>true</code> , если c является буквой нижнего регистра, и <code>false</code> в других случаях
isupper(c)	возвращает значение <code>true</code> , если c является буквой верхнего регистра, и <code>false</code> в других случаях
isspace(c)	возвращает значение <code>true</code> , если c является пробелом, и <code>false</code> в других случаях
toupper(c)	если символ c является символом нижнего регистра, то функция возвращает преобразованный символ c в верхнем регистре, иначе символ возвращается без изменений
strchr(s, c)	поиск первого вхождения символа c в строке s . В случае удачного поиска возвращает указатель на место первого вхождения символа c . Если символ не найден, то возвращается ноль
strspn(s1, s2)	определяет длину начального сегмента строки $s1$, содержащего те символы, которые не входят в строку $s2$
strspn(s1, s2)	возвращает длину начального сегмента строки $s1$, содержащего только те символы, которые входят в строку $s2$
atof(s1)	преобразует строку $s1$ в тип <code>double</code>
atoi(s1)	преобразует строку $s1$ в тип <code>int</code>
atol(s1)	преобразует строку $s1$ в тип <code>long int</code>
getchar(c)	считывает символ c со стандартного потока ввода, возвращает символ в формате <code>int</code>
gets(s)	считывает поток символов со стандартного устройства ввода в строку s до тех пор, пока не будет нажата клавиша ENTER

Рассмотрим несколько программ, используя функции для работы со строками и символами.

Копирование строк.

```
#include "stdafx.h"  
#include <iostream>  
using namespace std;
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    char s2[27] = "Counter-Strike 1.6 forever"; // инициализация  
    строки s2
```

```
    char s1[27]; // резервируем строку для функции strcpy()
```

```
    cout << "strcpy(s1,s2) = " << strcpy(s1,s2) << endl; // содержимое  
    строки s2 скопировалось в строку s1, возвращается указатель на s1
```

```
    cout << "s1= " << s1 << endl; // вывод содержимого строки s1
```

```
    char s3[7]; // резервируем строку для следующей функции
```

```
    cout << strncpy(s3, s2, 7) << endl; // копируем 7 символов из  
    строки s2 в строку s3
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

Конкатенация строк. Использование функций *strcat()* и *strncat()*, для объединения строк.

```
#include "stdafx.h"  
#include <iostream>  
using namespace std;
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    char s1[30] = "I am ";
```

```
    char s2[] = "programmer on the C++!!!!";
```

```
    cout << strcat(s1,s2) << endl; // объединяем строки s1 и s2,  
    результат сохраняется в строке s1
```

```
    char s3[23] = "I am a good ";
```

```
    cout << strncat(s3, s2,10) << "!!!" << endl; // объединяем 10 сим-  
    волов строки s2 со строкой s3
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

Сравнение строк. Рассмотрим работу функции *strcmp()*, остальные функции используются аналогично.

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char* argv[])
{
    char s1[] = "www.cppstudio.com";
    char s2[] = "http://www.cppstudio.com";
    cout << " s1 == s1 -> " << setw(2) << strcmp(s1,s1) << endl; //
строка s1 = s1
    cout << " s1 <  s2 -> " << setw(2) << strcmp(s1,s2) << endl; //
строка s1 < s2
    cout << " s2 >  s1 -> " << setw(2) << strcmp(s2,s1) << endl; //
строка s2 > s1
    system("pause");
    return 0;
}
```

Задание

1. По введенному символу определите, является ли он цифрой. Выведите «YES», если символ является цифрой и «NO» в противном случае. Обратите внимание, что слова нужно выводить маленькими буквами.

2. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество слов в данной строке.

3. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество слов в строке, начинающихся с буквы «с».

4. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого короткого слова.

5. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество букв «а» в каждом слове данной строки.

6. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество слов из трех букв в данной строке.

7. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество слов в строке, заканчивающихся на буквы «а».

8. Дана строка, состоящая из n символов. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого длинного слова.

Содержание отчета

1. Тема и цель работы.
2. Схема алгоритма решения.
3. Текст и результаты выполнения программы.

ЛИТЕРАТУРА

1. Шилдт, Г. С++: базовый курс: пер. с англ. / Г. Шилдт. – 3-е изд. – М. : Издательский дом «Вильямс», 2010. – 624 с.
2. Шилдт, Г. Полный справочник по С++: пер. с англ. / Г. Шилдт. – 4-е изд. – М. : Издательский дом «Вильямс», 2008. – 800 с.
3. Павловская, Т. А. С/ С++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб. : Питер, 2009. – 461 с.
4. Шилдт, Г. С++: для начинающих: пер. с англ. / Г. Шилдт. – М. : ЭКОМ Паблишерз, 2007. – 640 с.
5. Павловская, Т. А. С/ С++. Структурное программирование: практикум / Т. А. Павловская, Ю. А. Щупак. – СПб. : Питер, 2007. – 239 с.
6. Шилдт, Г. С++: руководство для начинающих: пер. с англ. / Г. Шилдт. – 2-е изд. – М. : Издательский дом «Вильямс», 2005. – 672 с.
7. Либерти, Д. Освой самостоятельно С++ за 21 день / Д. Либерти, Б. Джонс. – М. : Издательский дом «Вильямс», 2007. – 784 с.
8. Либерти, Д. Освой самостоятельно С++ за 24 часа / Д. Либерти, Д. Хорват. – М. : Издательский дом «Вильямс», 2007. – 448 с.
9. Васильев, А. Н. Самоучитель С++ с примерами и задачами / А. Н. Васильев. – СПб. : Наука и техника, 2010. – 480 с.

Учебное издание

ИНФОРМАТИКА

Лабораторный практикум
для студентов специальности
6-05-0714-03 «Инженерно-техническое проектирование
и производство материалов и изделий из них»
профилизации «Машины и технология литейного производства»,
«Аддитивные технологии в литейном производстве»

С о с т а в и т е л и:

ТЕЛЕШОВА Елена Владимировна
МАЦИНОВ Сергей Алексеевич
РОВИН Сергей Леонидович

Редактор *А. С. Козловская*
Компьютерная верстка *Н. А. Школьниковой*

Подписано в печать 11.06.2024. Формат 60×84 ¹/₁₆. Бумага офсетная. Ризография.
Усл. печ. л. 3,72. Уч.-изд. л. 2,16. Тираж 100. Заказ 973.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.