Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

О. Н. ПАВЛОВА

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Лабораторный практикум

Владимир 2018

УДК 004.42
ББК 32.973-018
П12

Рецензенты:

Кандидат физико-математических наук
доцент кафедры функционального анализа и его приложений
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
И. А. Петренко

Кандидат технических наук
генеральный директор ООО «ФС Сервис»
Д. С. Квасов

Издается по решению редакционно-издательского совета ВлГУ

Павлова, О. Н. Основы программирования : лаб. практикум / О. Н. Павлова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2018. – 104 с.
П12
ISBN 978-5-9984-0901-1

Приведены задания с примерами реализации на языке программирования высокого уровня C++ для выполнения лабораторных работ по дисциплине «Основы программирования». В ходе выполнения представленных работ студенты научатся строить алгоритмы решения задач, разрабатывать программы, работать в среде Microsoft Visual Studio, а также отлаживать программы и оценивать корректность их работы.

Предназначен для бакалавров, обучающихся по направлениям 28.03.01 «Нанотехнологии и микросистемная техника», 12.03.05 «Лазерная техника и лазерные технологии».

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 18. Табл. 7. Библиогр.: 5 назв.

УДК 004.42
ББК 32.973-018

ISBN 978-5-9984-0901-1

ПРЕДИСЛОВИЕ

В современном мире компьютеры и различные электронные вычислительные устройства очень активно внедряются во все сферы человеческой жизни. Проведение экспериментов и обработка их результатов, контроль производственных процессов, в которых могут быть задействованы лазерные технологии и нанотехнологии, – это лишь небольшой спектр применения компьютеров в современной жизни и работе специалистов указанных областей.

Курс «Основы программирования» призван познакомить студентов с базовыми понятиями программирования на языке высокого уровня C++, с методами построения алгоритмов.

Курс готовит студентов для выполнения обработки и представления экспериментальных данных, автоматизации экспериментов, моделирования объектов профессиональной деятельности.

Существует большое количество вариантов учебных программ, существующих в различных университетах и колледжах. При разработке учебно-методического комплекса дисциплины «Основы программирования», в состав которого входит курс лабораторных занятий, учтены, прежде всего, федеральные государственные образовательные стандарты подготовки бакалавров по направлениям 28.03.01 «Нанотехнологии и микросистемная техника» и 12.03.05 «Лазерная техника и лазерные технологии». Стоит отметить, что практикум соответствует рабочим программам по дисциплине «Основы программирования» указанных направлений подготовки бакалавров.

Практикум может быть рекомендован в качестве основного для выполнения лабораторных работ по дисциплине «Основы программирования».

ВВЕДЕНИЕ

Практикум состоит из десяти лабораторных работ, соответствующих усвоению практических навыков использования лекционного материала. Предполагает решение типовых задач программирования. По результатам выполнения каждой лабораторной работы требуется оформление отчета.

Структура изложения материала следующая: каждая лабораторная работа сопровождается описанием поставленной цели, теоретическим материалом, также в теоретической части разбираются практические примеры, аналогичные индивидуальным задачам. В каждой работе приведены задания, которые требуется выполнить студенту. Задания подразделяются на выполняемые по индивидуальному варианту, исследовательские, для всех студентов, и задачи, направленные на оттачивание проектной деятельности студента и работы в коллективе исполнителей, которые выполняются минигруппами по два-три человека.

Для контроля каждая работа, выполненная студентом, проходит проверку на работоспособность и корректность программы. На данном этапе преподаватель проверяет умение студента работать в среде разработки и тестировать программы. После допуска к защите студент готовит отчет. При подготовке отчета студент получает навыки работы с дополнительной литературой, навыки подготовки отчетной документации согласно нормативным стандартам.

Заключительный этап – защита работы, для подготовки к которой требуется ответить на контрольные вопросы.

Для выполнения лабораторных работ рекомендуется использовать среду разработки программного обеспечения Microsoft Visual Studio 2008/2010/2012/2015/2017 компилятор языка C++.

ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ОТЧЕТА

Содержание отчета:

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Краткая теоретическая часть (две-четыре страницы связного самостоятельно написанного текста на основании теоретического материала и литературы).
5. Блок-схема алгоритма, выполненная на компьютере в графическом схемном редакторе, например Microsoft Visio.
6. Программа (листинг программы).
7. Окно с результатами работы программы.
8. Выводы по результатам работы.

Блок-схема – обязательная часть отчета для первых четырех лабораторных работ.

При защите каждой лабораторной работы требуется рабочий проект для демонстрации программ. Отчет предоставляется в распечатанном, скрепленном виде. Для защиты будет предложено ответить на вопросы по теоретической части и решить практические задачи, аналогичные заданиям.

1. При оформлении отчета текст набирают в формате MS Word.
2. Стандартная страница текста – страница формата А4:

✓ левое поле – 30 мм;
✓ правое – 10 мм;
✓ верхнее и нижнее – 20 мм;
✓ междустрочный интервал – полуторный;
✓ шрифт Times New Roman, кегль 14 пт;
✓ режим «выравнивание по ширине»;
✓ отступ в начале абзаца (красная строка) – 15 мм.

3. Все листы должны быть пронумерованы.

– номер «один» присваивается титульному листу, номер на нем не проставляют (прил. 1).

– номер страницы проставляют по центру листа, кегль 12 пт.

Лабораторная работа № 1
РЕАЛИЗАЦИЯ ЛИНЕЙНЫХ АЛГОРИТМОВ

Цель работы: научиться составлять блок-схемы линейных алгоритмов, по составленной блок-схеме реализовывать простые консольные приложения на языке С++.

Теоретические сведения

Общая схема решения задачи на ЭВМ

Решение задачи на ЭВМ одним этапом – кодированием с помощью языка программирования – можно выполнять, если она содержит незначительное число действий.

Более сложные задачи решаются на ЭВМ путем прохождения ряда этапов:

1) определение перечня входных, выходных и промежуточных данных и методов расчета;

2) разработка алгоритма решения задачи;

3) создание текста программы;

4) отладка программы;

5) тестирование программы.

Алгоритм

Алгоритм – точное предписание о порядке выполнения действий из заданного фиксированного множества для решения всех задач заданного класса.

Для построения алгоритма используется графический способ – ***блок-схема программы.***

Наиболее простым при описании алгоритма является линейный способ записи алгоритма.

Структура программы

1. Раздел директив.
2. Раздел описания типов и объектов.
3. Раздел глобальных переменных.
4. Раздел функций (объявление и определение).
5. Раздел основной функции main (_tmain) – всегда должен быть.

Любая программа на языке C++ состоит из одной и более функций, одна из которых должна иметь имя ***main (_tmain)***, с которой начинается выполнение программы.

Понятие «Переменная»

Переменная – именованная ячейка памяти. Переменные могут быть разных типов:

1) числовые (целые – int, short, long и вещественные – float, double);
2) символьные (одиночные символы – char и строки – string);
3) логические (булевы – bool);
4) объектные и др.

Для объявления переменной нужно указать тип данных и имя переменной, например int a. В зависимости от места объявления переменные могут быть глобальными и локальными. Глобальные переменные объявляются вне всех операторов блока {}, они создаются во время запуска программы и уничтожаются по завершении работы программы. Локальные переменные создаются внутри блока и доступны для использования только внутри него, по выходу из блока уничтожаются.

Вывод информации в C++

Первый способ – использование функции вывода информации. Требуется в разделе подключения библиотек описать следующие строки:

```
#include <stdio.h>
int printf(const char *format [, argument]... );
```

format – формат вывода (%d – целое число, %f – вещественное число, %s – строка, \n – вставка разрыва строки, \t – вставка табуляции)

argument – переменная или константы, которые выводятся на экран.

Перечень спецификаторов для ввода/вывода информации функциями printf/ scanf:

%c – чтение символа,
%d – чтение десятичного знакового целого,
%u – чтение десятичного беззнакового целого,
%i – чтение десятичного целого,
%e, %E, %f – чтение числа типа float,
%lf – чтение числа типа double,
%h – чтение short int,

%o – чтение восьмеричного числа,

%s – чтение строки,

%x – чтение шестнадцатеричного числа,

%p – чтение указателя,

%n – чтение указателя в увеличенном формате.

Пример 1:

```
float summ = 2.72;
printf("Summ is %f.\n", summ);// – вывод значения, хранящегося в ячейке, именованной summ, с последующим переводом курсора на следующую строку:
```

Summ is 2.72.

Второй способ – использование потокового вывода cout.

Требуется в разделе подключения библиотек описать следующие строки:

```
#include <iostream>
using namespace std;
```

Пример 2:

```
float summ = 2.72;
cout<<"Summ is "<< summ<<endl;// – вывод значения, хранящегося в ячейке, именованной summ, с последующим переводом курсора на следующую строку:
```

Summ is 2.72.

Ввод информации в C++

Первый способ – использование функции вывода информации. Требуется в разделе подключения библиотек описать следующие строки:

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

Функция scanf() – основная функция ввода с консоли. Она предназначена для ввода данных любого встроенного типа и автоматически преобразует введенное число в заданный формат.

```
int scanf (char *управляющая_строка, ...);
```

Управляющая строка содержит три вида символов: спецификаторы формата, пробелы и другие символы.

Пример 3:

```
int x;
printf("Введите значение переменной x: ");
```

```
scanf("%d", &x);
```

Второй способ – использование потокового ввода cin.

Пример 4:

```
int x;
cout<<"Введите значение переменной x: ";
cin>>x;
```

Допустимые операции языка С++

Для выполнения лабораторной работы приведенных в табл. 1 операций достаточно, но перечень операций языка С++ гораздо шире. Более подробно с ним можно познакомиться в лекционной курсе или рекомендованной литературе.

Таблица 1

Некоторые операции языка

Операция	**Действие**	**Пример**
new	Создание (размещение)	new type
delete	Уничтожение (освобождение)	delete pointer
delete[]	Уничтожение массива	delete [] pointer
~	Дополнение	~expression
!	Логическое НЕ	! expression
+	Унарный плюс	+1
-	Унарный минус	-1
()	Приведение типа	(type) expression
.*	Выбор элемента	object.*pointer
->	Выбор элемента	object->*pointer
*	Умножение	expression* expression
/	Деление	expression/ expression
%	Взятие по модулю	expression% expression
+	Сложение (плюс)	expression+ expression
-	Вычитание (минус)	expression - expression

Использование в программе математических функций и констант

При решении задач на нахождение значений функций в точках достаточно часто приходится вычислять значения математических функций, таких как sin(x), cos(x), модуль числа и т.д. Встроенных в язык функций нет, но есть функции из внешней библиотеки math.h (табл. 2).

Для работы требуется в разделе подключения библиотек описать следующие строки:

```
#define _USE_MATH_DEFINES
#include <math.h>
```

M_PI – число ПИ.

M_E – экспонента.

Таблица 2

Перечень основных функций математической библиотеки

Имя	**Описание**
abs	Возвращает абсолютную величину числа
acos	арккосинус
asin	арксинус
atan	арктангенс
atan2	арктангенс с двумя параметрами
ceil	округление до ближайшего большего целого числа
cos	косинус
cosh	гиперболический косинус
exp	вычисление экспоненты
fabs	абсолютная величина (числа с плавающей точкой)
floor	округление до ближайшего меньшего целого числа
fmod	вычисление остатка от деления нацело для чисел с плавающей точкой
ldexp	умножение числа с плавающей точкой на целую степень двух
log	натуральный логарифм
log10	логарифм по основанию 10
modf(*x*,*p*)	извлекает целую и дробную части (с учетом знака) из числа с плавающей точкой
pow(*x*,*y*)	результат возведения *x* в степень *y*, x^y
sin	синус
sinh	гиперболический синус
sqrt	квадратный корень
tan	тангенс
tanh	гиперболический тангенс

Ход работы

Решаем задачу «Рассчитать длину окружности и площадь круга радиуса *R*».

1. На входе алгоритма один параметр – радиус круга *R*. Для расчета длины окружности *l* и площади круга *S* можно применить две известные в математике формулы: $l = 2\pi R$ и $S = \pi R^2$ соответственно. Для решения задачи подходит линейный тип алгоритма.

2. Строим блок-схему алгоритма (рис. 1).

3. Запустите Microsoft Visual Studio 2010/2012/2013/2015/2017.

4. В меню выберите опцию File\New Project.

5. Далее должно появиться окно с перечнем файлов, выбираем Win32 Console Application.

6. Далее необходимо назвать проект (Name), путь для размещения (Location) и имя проектного решения (Solution name) и нажать клавишу «ОК».

7. Затем можно завершить создание проекта по шаблону, нажав кнопку «Finish», или создать пустой проект, нажав клавишу «Next» и выбрав в меню опцию «Empty Project».

8. Если мы выбрали создание проекта по шаблону, то на экране видим проект с готовым программным кодом.

9. Далее вводим в файл программный код. Листинг кода представлен ниже

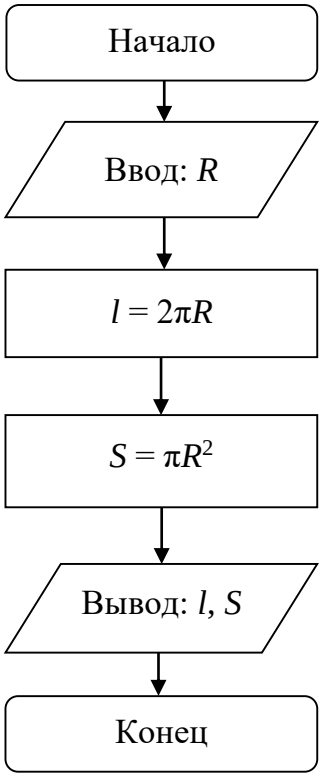

Рис. 1. Блок-схема

```
#include "stdafx.h"
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <locale.h>
#include <iostream>
using namespace std;

void main(void)
{
   setlocale(LC_ALL, "Russian");
   double R, l, S;
   cout<<"Введите радиус R: ";
```

```
    cin>>R;
    l=2*M_PI*R;
    S=M_PI*R*R;
    cout<<"Длина окружности l ="<<l<<endl;
    cout<<"Площадь круга S ="<<S<<endl;
    system("pause");
}
```

10. Запускаем программу, нажав «CTRL+F5», на вопрос «Создать проект?» отвечаем «Да» («Yes»).

11. Если в проекте нет никаких ошибок, то на черном экране видим результат выполнения программы (рис. 2).

```
Введите радиус R: 2.2
Длина окружности l =13.823
Площадь круга S =15.2053
Для продолжения нажмите любую клавишу . . .
```

Рис. 2. Результат запуска программы

12. В случае, если допущена ошибка, то отлаживаем проект и заново запускаем.

13. Выполняем по образцу первое задание согласно своему варианту.

14. Переходим к выполнению второго задания, решение которого должно выглядеть, как показано ниже.

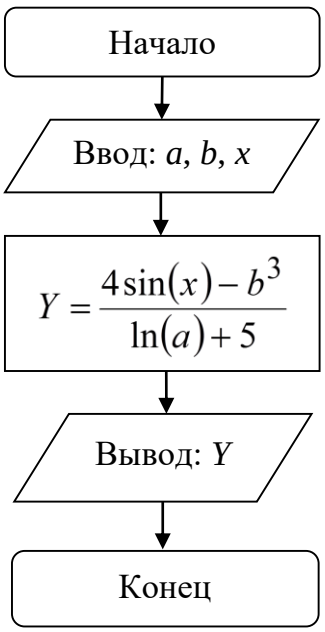

Рис. 3. Блок-схема

Пусть задана формула $Y=\frac{4\sin(x)-b^3}{\ln(a)+5}$. Требуется составить алгоритм для вычисления Y при заданных значениях параметров, введенных пользователем. Вариант исходных данных: x = 2.41, a = 24.02, b = 8.

Пояснения к решению задачи.

На вход алгоритма поступают три параметра x, a, b. Решение задачи заключается во вводе исходных данных, самого расчета значения функции и вывода результатов расчета. Блок-схема алгоритма приведена на рис. 3.

```
#include "stdafx.h"
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <locale.h>
#include <iostream>
using namespace std;

void main(void)
{
    setlocale(LC_ALL, "Russian");
    double x, a, b, Y;
    cout<<"Введите значения x a b: ";
    cin>>x>>a>>b;
    Y=(4*sin(x)-pow(b,3))/(log(a)+5);
    cout<<"Y ="<<Y<<endl;
    system("pause");
}
```

Результат запуска программы представлен на рис. 4.

```
Введите значения x a b: 2.41 24.02 8
Y =-62.2735
Для продолжения нажмите любую клавишу . . .
```

Рис. 4. Результат запуска программы

Задания для самостоятельного выполнения

Задача 1. Составить линейный алгоритм для вычисления:

1) длины медианы на сторону *a* в треугольнике со сторонами *a*, *b*, *c*;

2) длины медианы на сторону *b* в треугольнике со сторонами *a*, *b*, *c*;

3) длины медианы на сторону *c* в треугольнике со сторонами *a*, *b*, *c*;

4) длины биссектрисы на сторону *a* в треугольнике со сторонами *a*, *b*, *c*;

5) длины биссектрисы на сторону *b* в треугольнике со сторонами *a*, *b*, *c*;

6) длины биссектрисы на сторону *c* в треугольнике со сторонами *a*, *b*, *c*;

7) площади треугольника со сторонами *a*, *b*, *c*;

8) площади квадрата с диагоналями d;

9) площади ромба с диагоналями d_1 и d_2;

10) площади трапеции с высотой h и основаниями a и b;

11) площади вписанного четырехугольника со сторонами a, b, c, d (через полупериметры);

12) площади кольца с радиусами R и r;

13) диагонали параллелепипеда с ребрами a, b, c;

14) объема и поверхности параллелепипеда с ребрами a, b, c;

15) объема цилиндра высотой h и радиусом основания r.

Задача 2. Составить алгоритм вычисления Y по формуле с заданными значениями (табл. 3).

Таблица 3

Варианты индивидуальных заданий для задачи 2

Номер варианта	**Формула**	**Примеры исходных данных**		
		x	***a***	***b***
1	$Y=\dfrac{3b^4+a^3}{e^{-a}\left(3x^2-b\right)}$	–3.04	0.252	0.8
2	$Y=\dfrac{25-x^3}{\cos^2 a+b^4}$	5.7	7.8	10
3	$Y=\dfrac{\sqrt{4a}}{e^x-b^3}$	–2.04	16.2	4.3
4	$Y=\dfrac{\ln(x^3-7)}{\left(3a^4+2\right)\left(ab^2+\sqrt{x}\right)}$	11.4	4.3	20.3
5	$Y=\dfrac{x^4+\sin a}{b^3-2abx}$	2.01	–3.6	1.07
6	$Y=\dfrac{4\sin x-1}{\ln a+b^2}$	2.04	10.02	5.12
7	$Y=\dfrac{\sqrt{4a^2+bx}}{e^x+b^3}$	13.12	30.4	–7.3
8	$Y=\dfrac{0.5\sin x+1}{b^3-3a^2}$	–2.5	28.4	5.7
9	$Y=\dfrac{b^3-\cos x}{a^2+2}$	–2.4	3.05	8.02

Окончание табл. 3

Номер варианта	Формула	Примеры исходных данных		
		x	***a***	***b***
10	$Y = \frac{6e^x}{\cos a - b^3}$	31.7	–10.45	8.53
11	$Y = \frac{b^2 - e^x}{\sqrt{14a}}$	–4.53	0.41	2.43
12	$Y = \frac{3x^2 - 5.6}{2a^3 + b^2}$	2.4	0.2	–8
13	$Y = \frac{b^2 - x^3}{\cos(2a)}$	–8.04	0.25	7.5
14	$Y = \frac{x^2 - b}{(3a^3 + 5)\sqrt{2x^4 + b}}$	5.7	–7.8	0.2
15	$Y = \frac{b^2 - e^{5x}}{\sqrt{20a^{3/2}}(a + bx)}$	1.003	10.5	1.2

Срок выполнения

Одно занятие.

Контрольные вопросы и задания

1. Что такое алгоритм?
2. Что называется блок-схемой?
3. Какова функция блок-схемы при разработке алгоритма программы?
4. Опишите структуру программы на C++.
5. Что такое тип данных?
6. По каким принципам осуществляется выбор типа для конкретной переменной в программе?
7. Перечислите и охарактеризуйте основные типы данных языка C++.
8. Каково назначение функции main в проекте?
9. Каково назначение директивы #include?
10. Каково назначение операторов {}, “;”?

11. Как корректно применять функции расчета математических функций в программе?

12. Запишите формулу $Y = \dfrac{6e^{\cos x}}{\cos\ \ln a - b^5}$ согласно правилам языка C++.

Лабораторная работа № 2
РЕАЛИЗАЦИЯ РАЗВЕТВЛЕННЫХ АЛГОРИТМОВ

Цель работы: научиться составлять блок-схемы разветвленных алгоритмов, по составленной блок-схеме реализовывать простые консольные приложения.

Теоретические сведения

Разветвленный тип алгоритма – эта такая вычислительная схема, которая содержит не одну, а несколько возможных ветвей решения задачи, т.е. в структуре алгоритма есть хотя бы одна операция сравнения, порождающая две ветви последующего решения. Заранее нельзя сказать, какая ветвь алгоритма будет выполняться, все зависит от конкретных данных. Выбирается она автоматически, только в процессе исполнения алгоритма (остальные ветви для этого варианта данных останутся невостребованными). Тем труднее пользователю заранее (на этапе разработки) предусмотреть все возможные ветви решения и изобразить их, ничего не упустив.

Разветвленные алгоритмы бывают с полным ветвлением (рис. 5) и неполным ветвлением (рис. 6).

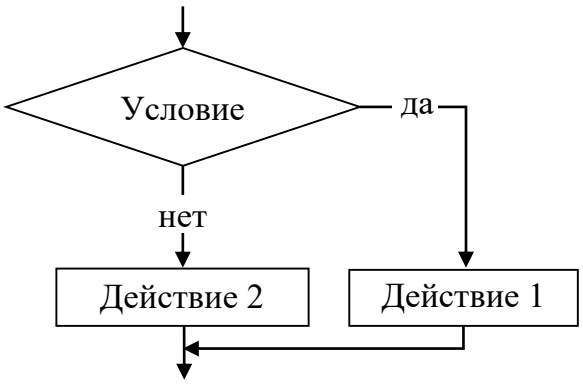

Рис. 5. Схема алгоритма с полным ветвлением

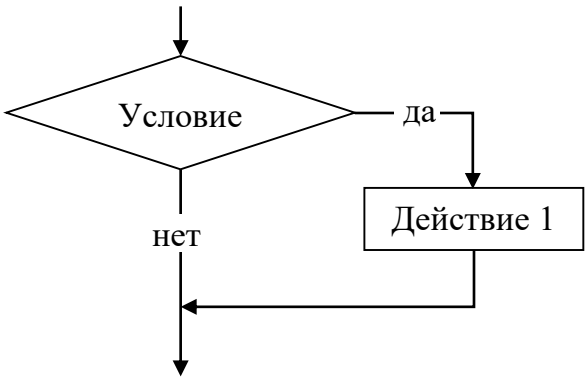

Рис. 6. Схема алгоритма с неполным ветвлением

Условный оператор if ... else

Для описания обеих схем разветвленного алгоритма применяется оператор *if … else* с учетом того, что для алгоритмов с неполным ветвлением будет отсутствовать часть условного оператора *else*.

Формат оператора:

if (Условие) Действие 1; [else Действие 2;]

Выполнение оператора *if* начинается с вычисления Условия.

Далее выполняем по следующей схеме:

– если Условие истинно (т.е. отлично от 0), то выполняется Действие 1;

– если Условие ложно (т.е. равно 0), то выполняется Действие 2;

– если Условие ложно и отсутствует Действие 2 (в квадратные скобки заключена необязательная конструкция), то выполняется следующий за *if* оператор.

После выполнения оператора *if* значение передается следующему оператору программы, если последовательность выполнения операторов программы не будет принудительно нарушена использованием операторов перехода.

Допускается использование вложенных операторов *if*. Оператор *if* может быть включен в конструкцию *if* или в конструкцию *else* другого оператора *if*. Чтобы сделать программу более читаемой, рекомендуется группировать операторы и конструкции во вложенных операторах *if*, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово *else* с наиболее близким *if*, для которого нет *else*.

Принцип работы логических операций

При построении выражения-условия для условного оператора могут применяться всевозможные операции сравнения на равенство (==), неравенство (!=), сравнения порядка (>, <, >=, <=), а также различные логические операции: логическое И (&&), логическое ИЛИ (||), отрицание (!).

Результаты логических операций выводятся на основе табл. 4.

Таблица 4

Правила выполнения логических операций

exp1	exp2	exp1&&exp2	exp1||exp2	!exp1
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Пример 1

Написать программу решения квадратного уравнения с коэффициентами *a*, *b*, *c*.

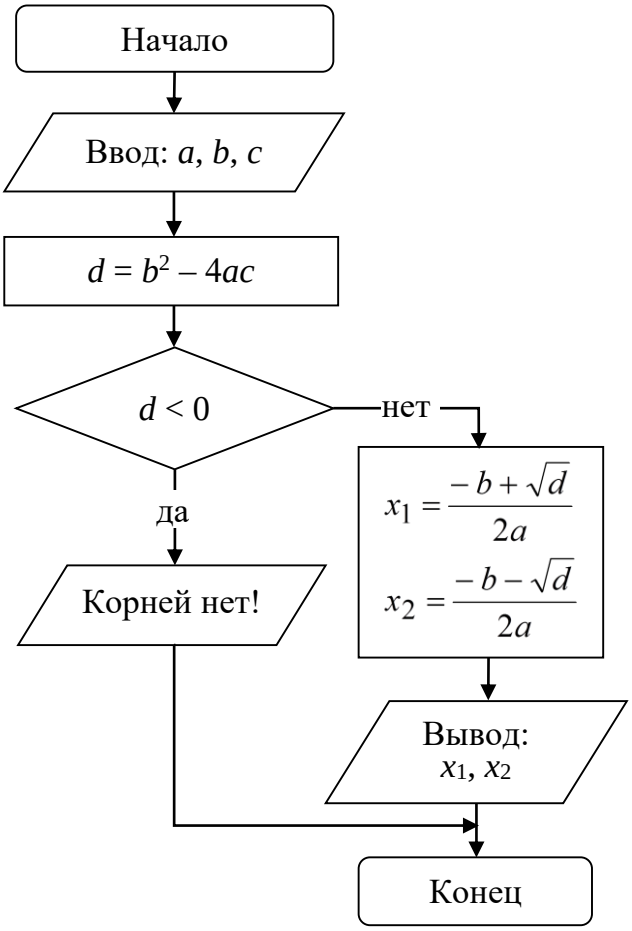

Рис. 7. Блок-схема алгоритма нахождения корней квадратного уравнения

Из школьного курса математики известно, что для нахождения корней квадратного уравнения требуется рассчитать дискриминант по формуле $d = b^2 - 4ac$, и в зависимости от его значения сделать выводы о наличии решения. Если дискриминант отрицательный, то решения уравнение не имеет, иначе существуют два корня, рассчитываемых по формуле $x_{1,2} = \frac{-b \pm \sqrt{d}}{2a}$.

Блок-схема алгоритма представлена на рис. 7. Ниже приведен код программы.

```
#include"stdafx.h"
#define _USE_MATH_DEFINES
#include<math.h>

int _tmain(int argc, _TCHAR* argv[])
{
  double a, b, c, x1, x2, d;
  printf("Введите коэффициенты уравнения a b c:");
```

```
    scanf("%lf %lf %lf", &a, &b, &c);
    d = b * b - 4 * a * c;
    if(d<0)
        printf("Корней нет!");
    else
    {
        x1 = (-b+sqrt(d))/(2*a);
        x2 = (-b-sqrt(d))/(2*a);
        printf("x1=%0.3lf\tx2=%0.3lf\n",x1, x2);
    }
    return 0;
}
```

Пример 2

Вычислить значение *Y* по формуле с учетом того, что *x* вводится пользователем

$$Y = \begin{cases} -1/\left(x^3+1\right) & \text{при} \quad x \le 1, \\ x^3/(5-x^3/5) & \text{при} \quad 1 < x \le 5. \end{cases}$$

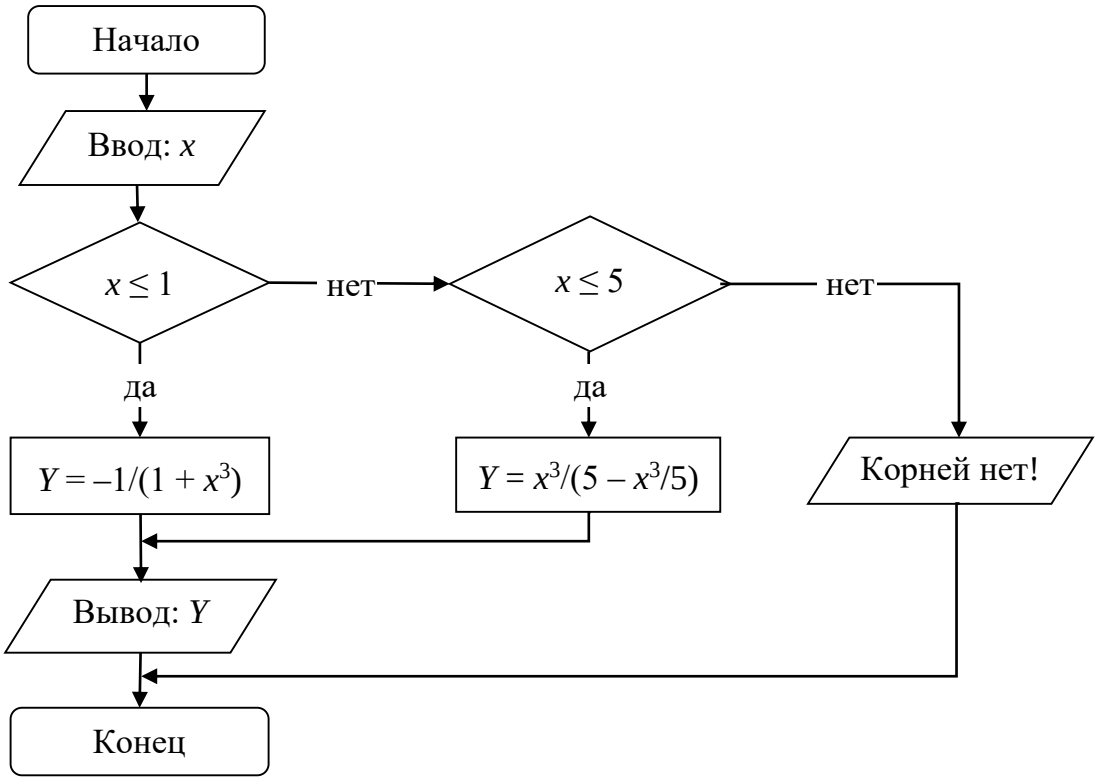

Рис. 8. Блок-схема решения примера 2

```
#include"stdafx.h"
#define _USE_MATH_DEFINES
#include<math.h>
int _tmain(int argc, _TCHAR* argv[])
{
    double x, Y;
    printf("Введите x:");
    scanf("%lf", &x);
    if(x<=1) Y=-1/(x*x*x+1);
    else if(x<=5) Y=x*x*x/(5-x*x*x/5); else
    {
        printf("Y не существует");
        return 0;
    }
    printf("Y=%lf", Y);
    return 0;
}
```

При тестировании программ, в коде которых есть разветвления, требуется создать тестовую таблицу, охватывающую все возможные результаты программы. Особое внимание при составлении таблицы стоит уделить граничным ситуациям. Пример такой таблицы для тестирования примера 2 приведен ниже (табл. 5).

Таблица 5

Правила выполнения логических операций

Входное значение	**Ожидаемый результат**	**Ответ программы**
$x = 0$	$Y = -1$	$Y = -1.000000$
$x = 1$	$Y = -0.5$	$Y = -0.500000$
$x = 2$	$Y = 40/17=2.3529$	$Y = 2.352941$
$x = 5$	$Y = -6.25$	$Y = -6.250000$
$x = 6$	Y не определен	Y не существует

Оператор switch

Оператор предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch (<выражение>)
{[объявления]
:
```

```
    [case константное-выражение1]:[операторы];
    [case константное-выражение2]:[операторы];
    :
    [default:[операторы]];
}
```

Выражение в круглых скобках может быть любым выражением, допустимым в языке C++, значение которого должно быть целым.

Отметим, что можно использовать явное приведение к целому типу.

Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора *switch* состоит из нескольких операторов, помеченных ключевым словом *case* с последующим *константным-выражением*. Следует отметить, что использование целого константного выражения – существенный недостаток, присущий рассмотренному оператору.

Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы. Все константные выражения в операторе *switch* должны быть уникальны. Кроме операторов, помеченных ключевым словом *case*, может быть, но только один фрагмент, помеченный ключевым словом *default*.

Список операторов может быть пустым либо содержать один или более операторов. Причем в операторе *switch* не требуется заключать последовательность операторов в фигурные скобки.

Отметим также, что в операторе *switch* можно использовать свои локальные переменные, объявления которых находятся перед первым ключевым словом *case*, однако в объявлениях не должна использоваться инициализация.

Схема выполнения оператора *switch* следующая:

1. Вычисляется выражение в круглых скобках.

2. Вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевым словом *case*.

3. Если одно из константных выражений совпадает со значением выражения, то управление передается оператору, помеченному соответствующим ключевым словом *case*.

4. Если ни одно из константных выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом *default*, а в случае его отсутствия управление передается следующему оператору после *switch*.

Отметим интересную особенность использования оператора *switch*: конструкция со словом *default* может быть не последней в теле оператора *switch*.

Ключевые слова *case* и *default* в теле оператора *switch* существенны только при начальной проверке, когда определяется начальная точка выполнения тела оператора *switch*. Все операторы – между начальным оператором и концом тела – выполняются вне зависимости от меток, если только какой-то из операторов не передаст управления из тела оператора *switch*. Таким образом, программист должен сам позаботиться о выходе из *case*, если это необходимо. Чаще всего для этого используется оператор *break*.

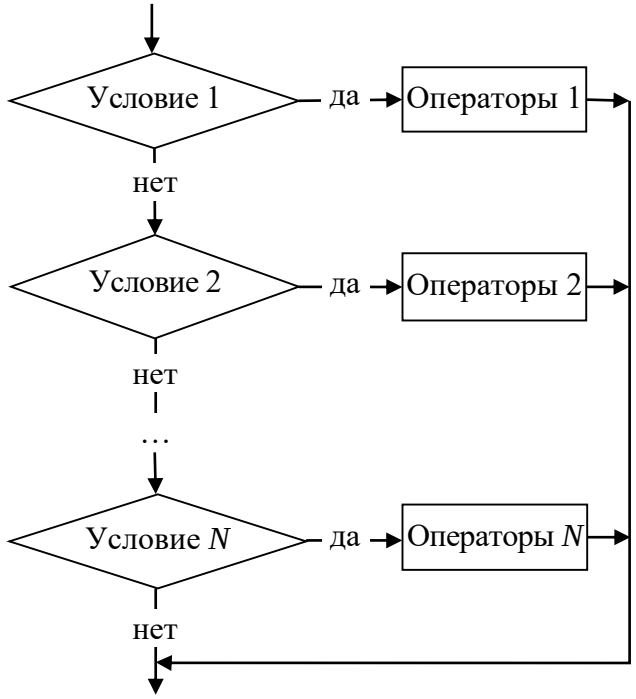

Рис. 9. Блок-схема с множественным выбором

Для того чтобы выполнить одни и те же действия для различных значений выражения, можно пометить один и тот же оператор несколькими ключевыми словами *case*. В блок-схемах множественный выбор обозначается, как представлено на рис. 9.

Пример 3 (*фрагмент кода, реализующий расчеты в зависимости от значения параметра ZNAK, суммы, разности, произведения или операции деления двух чисел*):

```
char ZNAC; //символ, обозначающий знак операции
int x, y, z;
switch (ZNAC)
{
    case '+': x = y + z; break;
    case '-': x = y - z; break;
    case '*': x = y * z; break;
    case '/': x = u / z; break;
}
```

Задания для самостоятельного выполнения

Первая задача выполняется каждым студентом, вторая и третья – согласно варианту.

Задача 1. Написать программу для решения биквадратного уравнения $ax^4 + bx^2 + c = 0$. На вход программы подаются три коэффициента уравнения a, b, c. Результатом решения уравнения могут быть следующие выводы:

- ✓ получены четыре различных решения;
- ✓ получены два решения;
- ✓ решений нет;
- ✓ уравнение не является биквадратным.

Задача 2. Вычислить значение Y по формуле, x вводится пользователем.

1. $$Y = \begin{cases} -1/x^2 & \text{при} \quad x \le -1, \\ x^2/(3 + x^2/5) & \text{при} \quad -1 < x \le 5, \\ 0.5 + \sin^2 x & \text{при} \quad 5 < x \le 6.28. \end{cases}$$

2. $$Y = \begin{cases} 58.4\sin^2 x & \text{при} \quad x \le -2.4, \\ \pi + \sin x & \text{при} \quad -2.4 < x \le 0.28, \\ \sin x - \cos^2 x & \text{при} \quad 0.28 < x \le 6.28. \end{cases}$$

3. $$Y = \begin{cases} 2.56x^2 - 12.6 & \text{при} \quad x \le 0.4, \\ x\sqrt{x} - 5x + 9 & \text{при} \quad 0.4 < x < 1.7, \\ tg(x) - 0.58 & \text{при} \quad x > 1.7. \end{cases}$$

4. $$Y = \begin{cases} x^2 - 21x + 63 & \text{при} \quad x \le -23, \\ x^3 - 3x^2 + 2tgx & \text{при} \quad -23 < x < -3, \\ 2.13 & \text{при} \quad -3 < x \le 0. \end{cases}$$

5. $$Y = \begin{cases} 2 - x^2 & \text{при} \quad x \le 4, \\ 5x - 7.5 & \text{при} \quad 4 < x < 20, \\ 1/(2 + x) & \text{при} \quad x > 20. \end{cases}$$

6. $Y=\begin{cases}1-x & \text{при} \quad x<1,\\ (1-x)(2-x) & \text{при} \quad 1\le x\le 2,\\ -(2-x) & \text{при} \quad x>2.\end{cases}$

7. $Y=\begin{cases}2+e^{x} & \text{при} \quad x\le 0,\\ 2+e^{x^2} & \text{при} \quad 0<x\le 1.5,\\ 1-e^{x} & \text{при} \quad 1.5<x\le 5.\end{cases}$

8. $Y=\begin{cases}1/(1-x) & \text{при} \quad x<-3,\\ 17.73 & \text{при} \quad -3\le x<-2,\\ 2-x\cos x & \text{при} \quad -2\le x<-1,\\ x^2-tg^2 x & \text{при} \quad -1<x<1.5.\end{cases}$

9. $Y=\begin{cases}125-\sin x & \text{при} \quad x\le 0.65,\\ 4.64+\sin^2 x & \text{при} \quad 0.65<x\le 1,\\ \sqrt{x^2+3x+0.125} & \text{при} \quad 1<x\le 6.3,\\ 0.43x^2-2x+0.5 & \text{при} \quad x>6.3.\end{cases}$

10. $Y=\begin{cases}e^{|x|} & \text{при} \quad x<-2,\\ 0 & \text{при} \quad |x|\le 2,\\ e^{x^2+x+1} & \text{при} \quad x>2.\end{cases}$

11. $Y=\begin{cases}arctg\sqrt{x^2+3} & \text{при} \quad x<-1,\\ \arccos x & \text{при} \quad -1\le x\le 1,\\ e^{-x^2} & \text{при} \quad 1<x\le 5,\\ x^2-3x+5 & \text{при} \quad 5<x\le 15,\\ \ln(x-15) & \text{при} \quad x>15.\end{cases}$

12. $Y=\begin{cases}564x-2x^{22} & \text{при} \quad x\le -5,\\ 1398 & \text{при} \quad -5<x\le 17,\\ 4x/(1-x^2) & \text{при} \quad 17<x<18,\\ \sin x & \text{при} \quad x\ge 18.\end{cases}$

13. $Y = \begin{cases} \sqrt{|\ln(0.2^2 - x^2)|} & \text{при} \quad 0.2 < x \le 0.3, \\ 1 & \text{при} \quad 0.3 < x \le 0.9, \\ 0.5\cos^2(x/2) & \text{при} \quad 0.9 < x \le 1, \\ tgx & \text{при} \quad x > 1. \end{cases}$

14. $Y = \begin{cases} \sqrt{|1 + x^3|} & \text{при} \quad x \le -12, \\ -0.95x + x^2 & \text{при} \quad -12 < x \le -0.5, \\ x/(2-x) & \text{при} \quad -0.5 < x \le 0, \\ x^2 & \text{при} \quad x > 0. \end{cases}$

15. $Y = \begin{cases} 163 - \cos^2 x & \text{при} \quad x \le -1.2, \\ \sqrt{-x} & \text{при} \quad -1.2 < x \le -0.5, \\ tgx - 4/tgx & \text{при} \quad -0.5 < x \le 0.2, \\ \sin x & \text{при} \quad x > 0.2. \end{cases}$

Задача 3. Составить программу с применением оператора множественного выбора.

1. Дан порядковый номер месяца, вывести на экран количество месяцев, оставшихся до конца года.

2. Дан порядковый номер дня месяца, вывести на экран количество дней, оставшихся до конца месяца.

3. Дан номер масти m ($1 \le m \le 4$), определить название масти. Масти нумеруются: «пики» – 1, «трефы» – 2, «бубны» – 3, «червы» – 4.

4. Дан номер карты k ($6 \le k \le 14$), определить достоинство карты. Достоинство определяется по следующему правилу: «туз» – 14, «король» – 13, «дама» – 12, «валет» – 11, «десятка» – 10, …, «шестерка» – 6.

5. Дан номер масти m ($1 \le m \le 4$) и номер достоинства карты k ($6 \le k \le 14$). Определить полное название соответствующей карты в виде «дама пик», «шестерка бубен» и т.д.

6. С 1 января 1990 года по некоторый день прошло n дней, определить название текущего месяца.

7. С 1 января 1990 года по некоторый день прошло m месяцев, определить название текущего месяца.

8. С некоторой даты по настоящий день прошло m месяцев, определить название месяца неизвестной даты.

9. С некоторой даты по настоящий день прошло *m* месяцев, найти неизвестную дату.

10. С некоторой даты по настоящий день прошло *n* дней, найти неизвестную дату.

11. С 1 января 1990 года по некоторый день прошло *m* месяцев и *n* дней, определить название текущего месяца.

12. Дано расписание приемных часов врача. Вывести на экран приемные часы врача в заданный день недели (расписание придумать самостоятельно).

13. Проведен тест, оцениваемый в целочисленный баллах от нуля до ста. Вывести на экран оценку тестируемого в зависимости от набранного количества баллов: от 90 до 100 – «отлично», от 70 до 89 – «хорошо», от 50 до 69 – «удовлетворительно», менее 50 – «неудовлетворительно».

14. Дан год. Вывести на экран название животного, символизирующего заданный год по восточному календарю.

15. Дан возраст мужчины в годах. Вывести на экран возрастную категорию: до года – «младенец», от года до 11 лет – «ребенок», от 12 до 15 лет – «подросток», от 16 до 25 лет – «юноша», от 26 до 70 лет – «мужчина», более 70 лет – «старик».

Срок выполнения

Одно занятие.

Контрольные вопросы и задания

1. Что такое разветвленный алгоритм?
2. Что означает неполное ветвление? Приведите пример.
3. Расскажите о синтаксисе условного оператора.
4. Какой блок в блок-схемах отвечает за отображение условия?
5. Какие значения являются истинными в C++?
6. Перечислите логические операции. Опишите, как они работают.
7. Опишите принцип работы оператора множественного выбора.
8. Что обозначает ключевое слово *default*?

Лабораторная работа № 3
РЕАЛИЗАЦИЯ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цель работы: научиться составлять блок-схемы циклических алгоритмов, по составленной блок-схеме реализовывать простые консольные приложения.

Теоретические сведения

Циклический тип алгоритма – это вычислительная схема, в которой некоторая последовательность операций алгоритма будет выполняться многократно (т.е. в цикле), образуя тело цикла (ТЦ). В крайнем случае тело цикла может состоять всего из одной операции. Для того чтобы результаты всякий раз получались новые, необходимо алгоритмически предусмотреть три момента:

а) надо организовать данные для первого прохождения тела цикла;

б) после каждого выполнения тела цикла надо обновлять данные для очередного прохождения цикла;

в) надо организовать условие выхода / продолжения цикла.

Типы циклических алгоритмов и соответствующие им операторы C++

1. Циклы с предусловием

Этот вид цикла работает по принципу: «Пока выполняется условие – происходит работа». Если условие цикла при первом входе в цикл ложно, то тело цикла ни разу не выполнится (рис. 10).

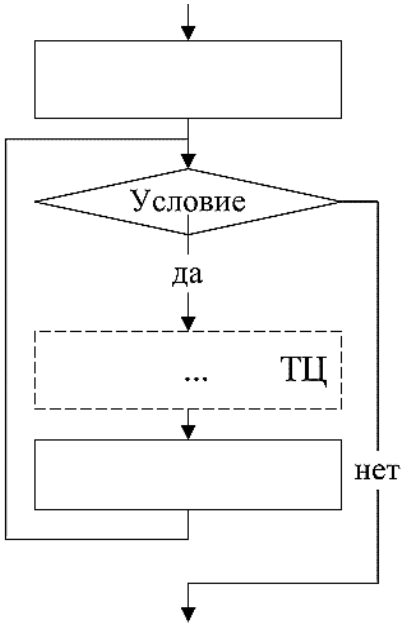

Рис. 10. Схема цикла с предусловием

```
//подготовка первого прохода цикла

while (условие)
{
    инструкция; //тело цикла
   // подготовка новой итерации
}
```

2. Циклы с постусловием

Бывают случаи, когда вы хотите проверять условие не в начале, а в конце цикла. В таком случае лучше использовать цикл *do ... while* (рис. 11). Особенностью цикла с постусловием является то, что один раз цикл обязательно выполнится, а дальнейшая работа будет зависеть от значения условия.

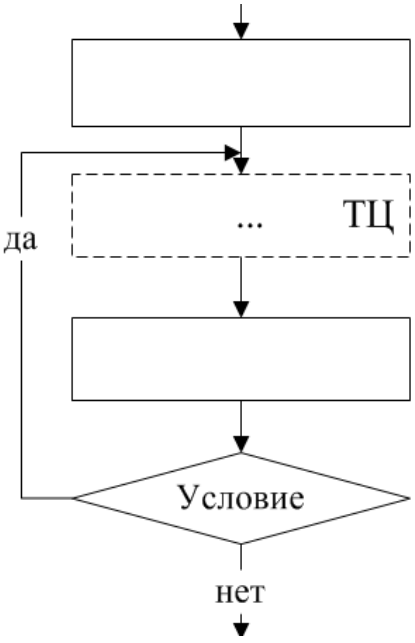

Рис. 11. Схема цикла с постусловием

```
// подготовка первого прохода цикла
do
{
      инструкция; //тело цикла
      // подготовка новой итерации

}
while (условие);
```

3. Циклы с параметром

Переменная, которая управляет циклом, называется параметром цикла. В блоке цикла указываются параметр цикла, знак присваивания (=), начальное значение параметра, конечное значение параметра, шаг изменения параметра (рис. 12). Например, *i* = 1, *n*, 1.

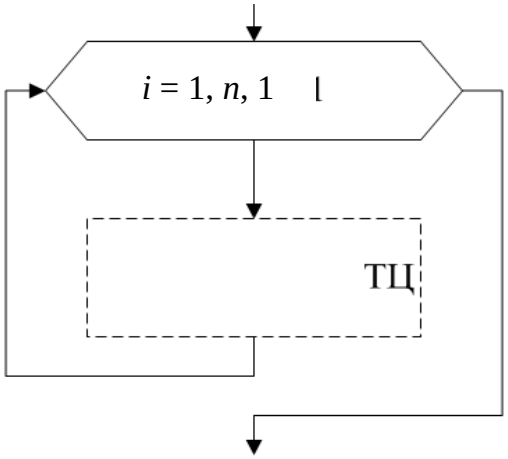

Рис. 12. Блок-схема цикла с параметром

```
for(i=1; i<=n; i++)
{

//тело цикла

}
```

Общий вид оператора for:

```
for (<инициализация>;
<условие>; <приращение>)
<оператор>;
```

Цикл *for* может иметь большое количество вариаций. *Инициализация* – это присваивание начального значения переменной, которая называется параметром цикла. *Условие* представляет собой условное выражение, определяющее, следует ли выполнять *оператор* цикла (тело цикла) в очередной раз. Оператор *приращение* изменяет параметр цикла при каждой итерации. Эти три оператора обязательно разделяются точкой с запятой. Цикл *for* выполняется, если выражение *условие* принимает значение ИСТИНА. Если оно хотя бы один раз принимает значение ЛОЖЬ, то программа выходит из цикла и выполняется оператор, следующий за телом цикла *for*.

Любое из указанных выражений может отсутствовать.

Операторы прерывания и пропуска текущей итерации

Оператор break обеспечивает прекращение выполнения самого внутреннего из включающих его операторов *switch, do, for, while*. После выполнения оператора *break* управление передается оператору, следующему за прерванным.

Оператор continue – оператор продолжения. Встречается внутри тела цикла и означает переход на следующую итерацию цикла.

Пример 1

Рассчитать значение функции в каждой точке *x*, *x*0 – координата начальной точки для расчета, Δ*x* – величина изменения координаты, *xk* – координата конечной точки, *n* – количество расчетных точек. Значения *x*0, Δ*x, xk, n*, приведенные в задании, вводятся с клавиатуры и могут меняться на усмотрение пользователя.

$Y = \cos x - \sin^2 x, \quad x0 = -0.5, \quad \Delta x = 0.2, \quad n = 10.$

Решение

В качестве исходных данных задано количество расчетных точек, начальная точка, шаг изменения параметра *x*, а также формула для вычисления функции.

Работа цикла прекращается, как только во всех требуемых точках произведен расчет ($i <= n$).

На каждой итерации цикла в данном случае требуется изменять значения двух параметров – координаты точки и ее номера.

Блок-схема алгоритма приведена на рис. 13.

В процессе преобразования блок-схемы в код программы можно выбрать цикл с предусловием или пошаговый. Напишем программу с пошаговым циклом *for*.

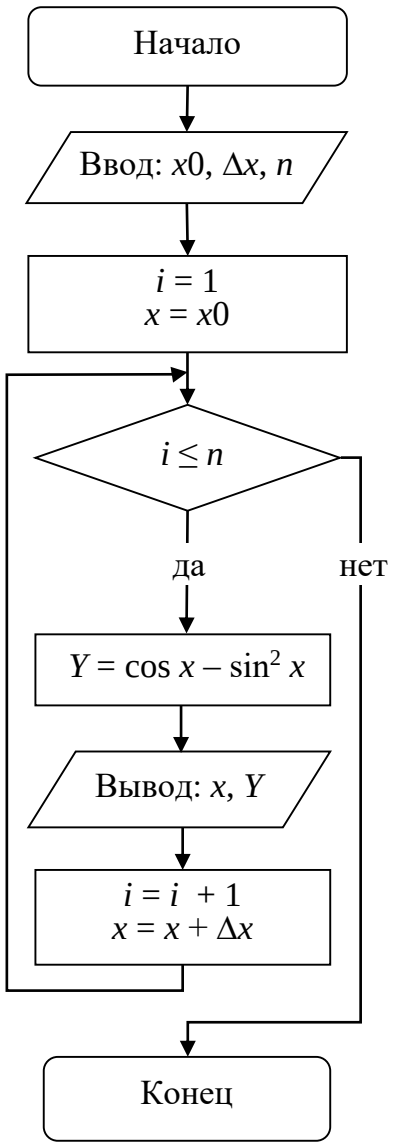

Рис. 13. Блок-схема алгоритма решения примера 1

```
#include "stdafx.h"
#include <math.h>
#include <stdio.h>
int _tmain(int argc, _TCHAR*
argv[])
{
  double x0,dx,x,y,n,i;
  printf("Введите x0 ");
  scanf_s("%lf",&x0);
  printf("Введите n ");
  scanf_s("%lf",&n);
  printf("Введите dx ");
  scanf_s("%lf",&dx);

  for(x=x0, i=1; i<=n; i++,
x+=dx)
  {
      y=cos(x)-sin(x)*sin(x);
      printf("x=%lf \ty=%lf
\n",x,y);
  }
  return 0;
}
```

Пример 2

Вычислить сумму натуральных четных чисел, не превышающих ***N***.

Решение

Входные данные: *N* – целое число.

Выходные данные: *S* – сумма четных чисел.

Промежуточные данные: *i* – переменная, принимающая значения от 2 до ***N*** с шагом 2, следовательно, также имеет целочисленное значение.

При сложении нескольких чисел необходимо накапливать результат в определенном участке памяти, каждый раз считывая из этого участка предыдущее значение суммы и прибавляя к нему следующее слагаемое. Для выполнения первого оператора накапливания суммы из участка памяти необходимо взять такое число, которое не влияло бы на результат сложения. То есть перед началом цикла переменной, предназначенной для накапливания суммы, необходимо присвоить значение **нуль**. Блок-схема решения этой задачи представлена на рис. 14.

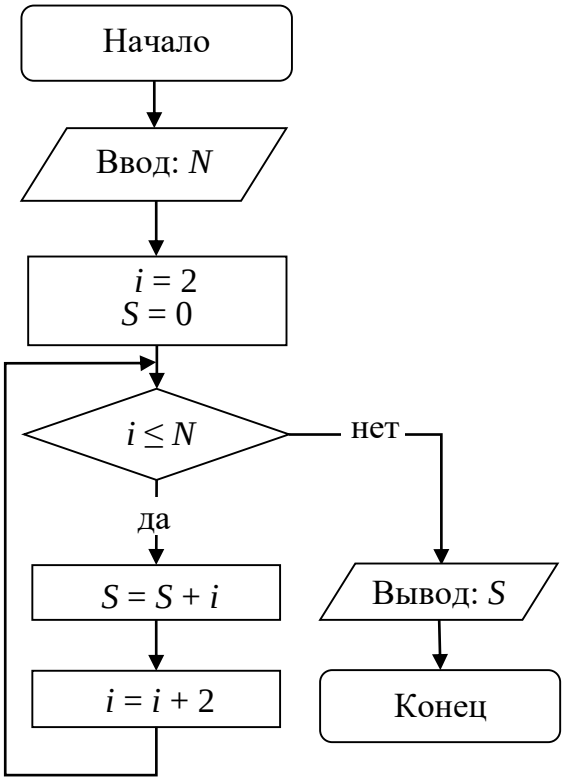

Рис. 14. Блок-схема задачи из примера 2

```
//Первый вариант программы с циклом с предусловием
    int main()
    {
        int N, S, i;
        S = 0;
        i = 2;
        while (i<=N)
        {
            S=S + i;
            i=i + 2;
        }
        printf("Сумма S=%d", S);
        return 0;
    }
```

```
// Второй вариант программы с применением цикла for
    int main()
    {
        int N, S, i;
        S = 0;
        for (i = 2; i <= N; i = i+2)
            S = S + i;
        printf ("Сумма S = %d\n", S);
        return 0;
    }
```

Задания для самостоятельного выполнения

Задача 1. Рассчитать значение функции в каждой точке x, $x0$ – координата начальной точки для расчета, Δx – величина изменения координаты, xk – координата конечной точки, n – количество расчетных точек. Значения $x0$, Δx, xk, n, приведенные в задании, вводятся с клавиатуры и могут меняться на усмотрение пользователя.

1) $Z = 2 - \sin^2(x)$ $x0 = 0.5$ $\Delta x = 0.1$ $n = 8$,
2) $Y = e^x + \sin(x)$ $x0 = 0.2$ $\Delta x = 0.05$ $n = 10$,
3) $Y = 5x^3 / \sqrt{2 + x}$ $x0 = -1$ $\Delta x = 0.2$ $n = 9$,
4) $S = x^2 e^x$ $x0 = -0.5$ $\Delta x = 0.5$ $n = 8$,
5) $P = (1 - x^2)/(1 + x^2)e^4$ $x0 = 0.2$ $\Delta x = 0.1$ $n = 7$,
6) $Y = 4.1 - \sqrt{|10x|} + 15.8x^2$ $x0 = 1.4$ $\Delta x = -0.2$ $n = 6$,
7) $Y = 4 \sin^2(x) + 1$ $x0 = 1.5$ $\Delta x = 0.5$ $xk = 4$,
8) $Z = 5.6(1 - x^2)$ $x0 = 17.6$ $\Delta x = -0.2$ $n = 9$,
9) $Y = 2 \sin(x)/x$ $x0 = 0.1$ $\Delta x = 0.05$ $n = 10$,
10) $Z = 0.51\sqrt{|x| - x}$ $x0 = -17.5$ $\Delta x = 0.5$ $n = 6$,
11) $S = 1 + 2x - x^2$ $x0 = 2.64$ $\Delta x = 0.01$ $n = 8$,
12) $S = 2x + 9.8x^2$ $x0 = 0$ $\Delta x = 10$ $n = 10$,
13) $Y = 12.4 / \sqrt{x} - x^2$ $x0 = 11.5$ $\Delta x = -0.2$ $n = 7$,
14) $V = 2x^2 - 0.625x + 0.7$ $x0 = 9.2$ $\Delta x = -0.2$ $n = 7$,
15) $Z = 12.65x^2 - 2.3\sqrt{|x|}$ $x0 = 0.5$ $\Delta x = -0.01$ $n = 5$.

Задача 2.

1. Напечатайте все точные квадраты натуральных чисел, не превосходящие данного числа n (например, при вводе 50 программа должна вывести 1 4 9 16 25 36 49).

2. Дано натуральное число n. Определите, является ли оно степенью числа два, и выведите слово YES, если является, и слово NO, если не является.

3. Для данного натурального числа n определите такое наименьшее целое k, что $2^k \geq n$. Например, при вводе числа «семь» программа должна вывести «три».

4. В первый день спортсмен пробежал x километров, а затем он каждый день увеличивал пробег на 10 % от предыдущего значения. По данному числу y определите номер дня, на который пробег спортсмена составит не менее y километров. Например, при вводе 10 20 программа должна вывести «девять». x и y – действительные числа, ответ – целое число.

5. В первый день спортсмен пробежал x километров, а затем он каждый день увеличивал пробег на 10 % от предыдущего значения. По данному числу y определите номер дня, на который суммарный пробег спортсмена составит не менее y километров. Например, при вводе 10 100 программа должна вывести «восемь».

6. Дано натуральное число n. Напишите программу, вычисляющую сумму цифр числа n. Выведите сумму цифр числа n.

7. Дано натуральное число n. Напишите программу, определяющую количество нулей среди всех цифр числа n. Выведите результат.

8. Дано натуральное число n. Напишите программу, определяющую наименьшую и наибольшую цифры данного числа. Выведите наименьшую и наибольшую цифры данного числа (например, при вводе 179 программа выводит 1 9).

9. Вводится последовательность целых чисел до тех пор, пока не будет введено число 0. После ввода числа 0 программа должна завершить свою работу и вывести сумму введенных чисел.

10. По данному натуральному числу n найдите сумму чисел 1 + + 1/1! + 1/2! + 1/3! + ... + 1/n!. Количество действий должно быть пропорционально n. Напишите программу, которая считывает значение n и выводит результат в виде действительного числа. К чему будет стремиться эта сумма при росте числа n?

11. По данному числу *n* выведите *n*-е число Фибоначчи. Использовать рекурсию нельзя.

12. Напишите программу, которая переставляет цифры числа в обратном порядке (например, ввод 179 – вывод 971). Напишите программу, которая по данному натуральному *n* печатает его цифры в обратном порядке.

13. Назовем число палиндромом, если оно не меняется при перестановке его цифр в обратном порядке. Напишите программу, проверяющую по данному числу *n*, является ли оно палиндромом. Напишите программу, которая по заданному числу *K* выводит количество натуральных палиндромов, не превосходящих *K*. Например, при вводе единицы программа выводит единицу, а при вводе 100 программа выводит 18.

14. Написать программу расчета факториала числа *n*.

15. Написать программу определения всех счастливых номеров билетиков автобуса.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Перечислите характеристики циклических алгоритмов.

2. Какая информация требуется для правильного построения циклического алгоритма?

3. Перечислите и охарактеризуйте виды циклов.

4. Как работает оператор *for*. Синтаксис и семантика оператора.

5. Как работает оператор *while*. Синтаксис и семантика оператора.

6. Как работает оператор *do…while*. Синтаксис и семантика оператора.

7. Опишите принципы работы операторов *break* и *continue*.

Лабораторная работа № 4

РЕАЛИЗАЦИЯ КОМБИНИРОВАННЫХ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цель работы: научиться составлять блок-схемы алгоритмов, содержащих вложенные циклы и циклы с разветвлениями в теле и досрочным выходом из тела цикла, по составленной блок-схеме реализовывать простые консольные приложения.

Теоретические сведения

Вложенные циклы

Вложенные циклы (цикл в цикле) – это циклические алгоритмы, имеющие несколько переменных (более 1), каждая из которых меняется по своему закону (одному из предложенных выше).

Рассмотрим функцию двух аргументов $Y = x^2/(x + a)$, где x и a меняются независимо друг от друга.

Исходные данные для x и a:

xн – начальное значение x;
aн – начальное значение a;
xк – конечное значение x;
aк – конечное значение a;
hx – шаг изменения x;
ha – шаг изменения a.

Точки, в которых вычисляется Y, можно изобразить некоторой областью D. Для построения алгоритма нельзя использовать в чистом виде ни один приведенный ранее простой цикл, так как он дает одновременное изменение x и a и вычисление Y лишь в диагональных точках области D (рис. 15).

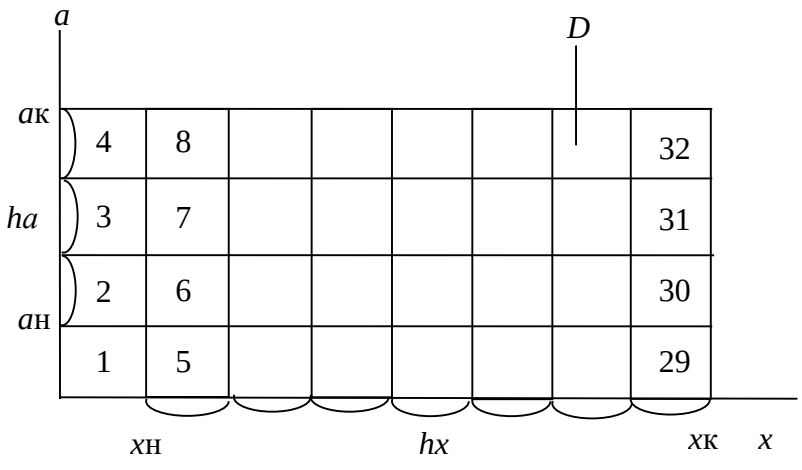

Рис. 15. Схема перебора точек для вложенных циклов

Необходимо построить два независимых цикла, вкладывая один в другой. Для правильной вложенности циклов следует предварительно определить: какой аргумент образует внутренний цикл, а какой – внешний.

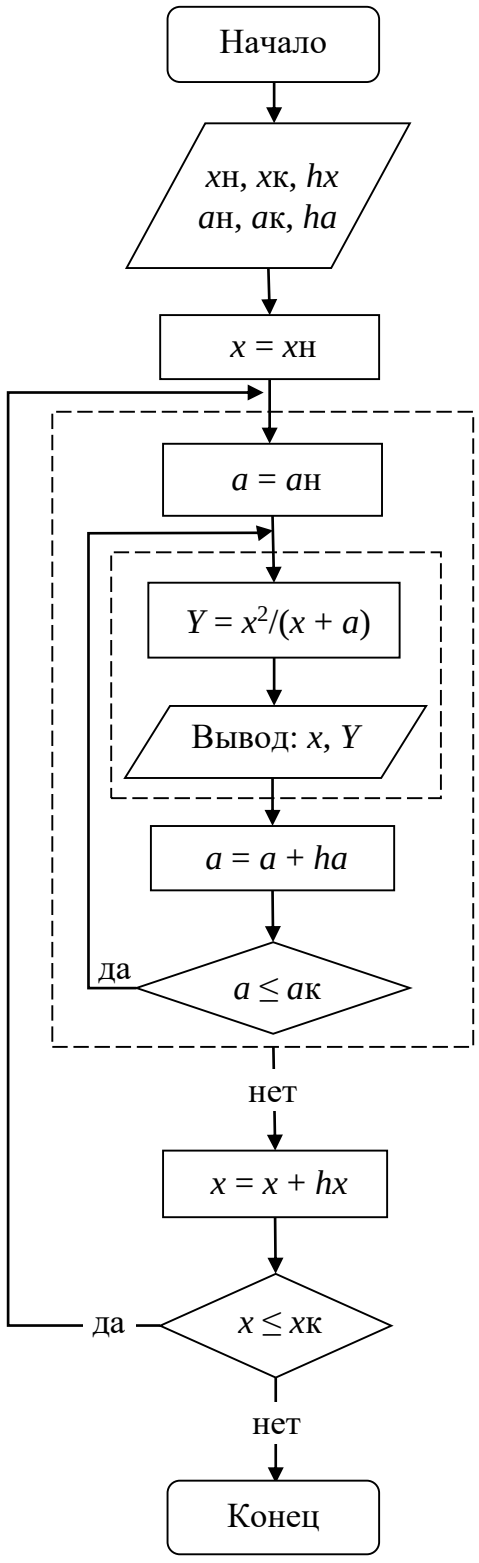

Рис. 16. Блок-схема алгоритма с вложенным циклом

С точки зрения решаемой задачи совершенно безразлично, как сделать эти назначения; меняется только порядок перебора расчетных точек в области *D* (но ни одна пропущена не будет). Пусть *a* – внутренний аргумент, т.е. меняется в первую очередь, тогда *x* образует внешний цикл.

Далее строим блок-схему алгоритма реализации поставленной задачи (рис. 16).

Переходим к написанию программы.

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

void main()
{
  double y,xn,hx,xk,an,
  ak, ha,x,a;
  cout<<"xn= "; //Просим
  пользователя ввести
  cin>>xn; //все требуе-
  мые для расчета
  cout<<"hx"; //параметры
  cin>>hx;
```

```
    cout<<"xk= "; cin>>xk;
    cout<<"an= ";
    cin>>an;
    cout<<"ha";
    cin>>ha;
    cout<<"ak= ";
    cin>>ak;
    x = xn;     //подготовка внешнего цикла
    do          // цикл по параметру x
    {
        a = an; //подготовка внутреннего цикла
        do      // цикл по параметру a
        {
            y=x*x/(x+a); // расчет функции

cout<<"y("<<a<<","<<x<<")="<<y<<endl;
            a += ha
        } while (a <= ak);
        x += hx;    //переход к новой итерации
    } while (x <= xk);
}
```

Разветвленное тело цикла, в том числе «досрочный» выход из тела цикла

В данном типе алгоритмов совмещаются схемы разветвленного алгоритма тела цикла. При этом стоит отметить, что при реализации данной схемы могут применяться циклические алгоритмы любых типов. Досрочный выход означает, что одна из ветвей выходит за пределы тела цикла, а другая продолжает цикл.

Пример. Вычислить значения заданной функции $Y(a_i)$ на последовательности неравноотстоящих узлов a_i:

$$Y = \begin{cases} -1/\left(a_i^3 + 1\right) & \text{при} \quad a_i \le 1, \\ a_i^3 / (5 - a_i^3 / 5) & \text{при} \quad 1 < a_i \le 5. \end{cases}$$

Исходные данные: n – длина массива a; массив $a = (a_1, a_2, \ldots, a_n)$.

Алгоритм представлен на рис. 17.

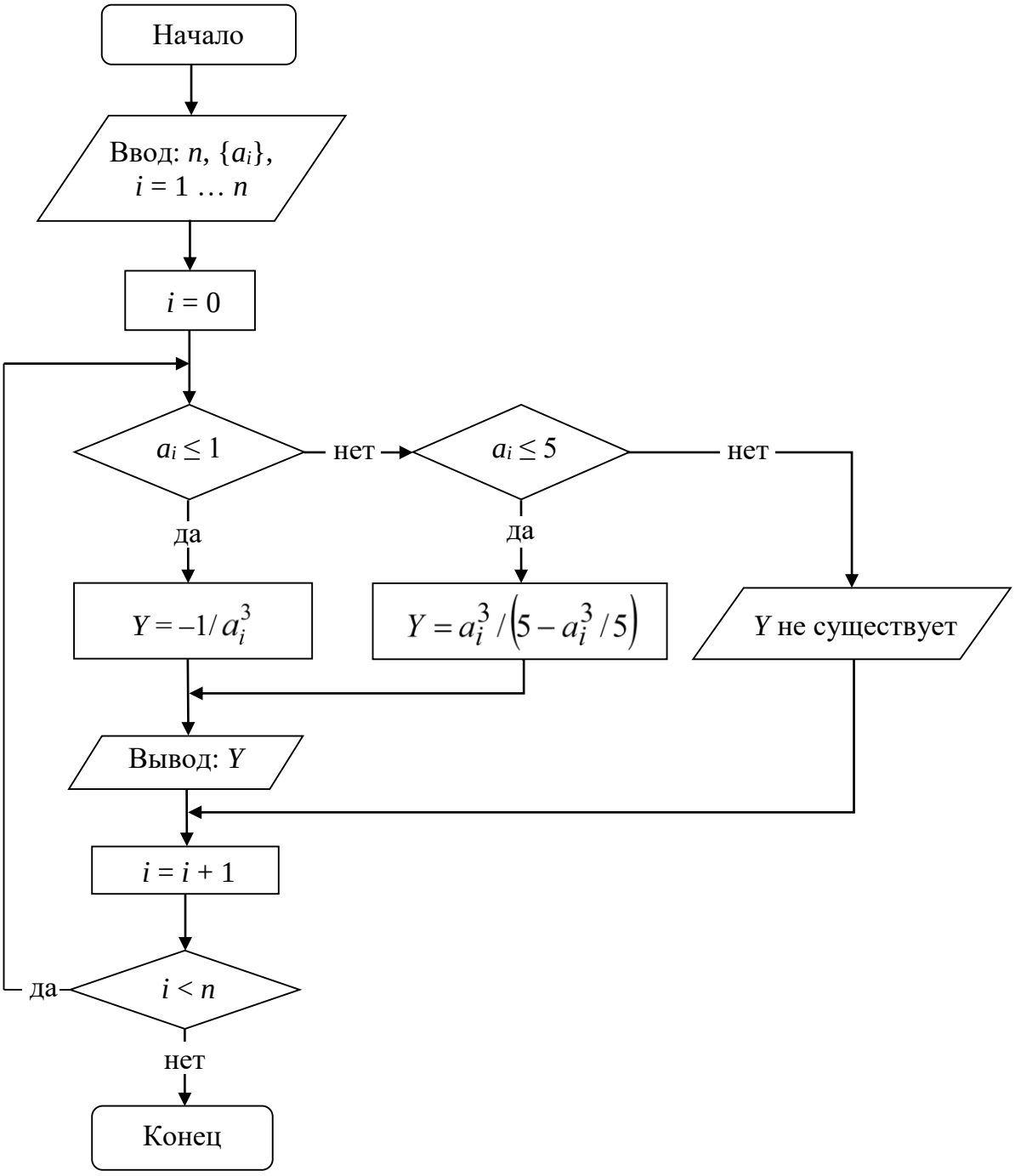

Рис. 17. Блок-схема циклического алгоритма с разветвлениями в теле цикла

На основе блок-схемы разрабатываем программу, текст которой представлен ниже.

```
#include"stdafx.h"
```

```
#define _USE_MATH_DEFINES
#include<math.h>
int _tmain(int argc, _TCHAR* argv[])
{
 int i, n;
 double *a, Y;
 printf("Введите количество расчетных точек:");
 scanf("%d", &n);
 a = new double[n]; //выделяем место в памяти
для хранения массива координат точек
 printf("Введите координаты расчетных точек:");
 for (i=0; i<n; i++)
     scanf("%lf", &a[i]);
 i=0;
do
{
   if(a[i]<=1) Y=-1/( pow(a[i],3)+1);
   else
   if(a[i]<=5) Y= pow(a[i],3)/(5-pow(a[i],3)/5);
   else
   {
        printf("Y не существует");
        return 0;
   }
   printf("a=%lf \tY=%lf\n ", a[i],Y);
   i++;
}
while (i<n);
return 0;
}
```

Задания для самостоятельного выполнения

Задача 1. Выполнить расчет функции от двух аргументов при заданных законах изменения аргументов. Все значения параметров даны для определения типов данных, могут быть использованы как вариант для тестирования программы. Все значения параметров вводятся с клавиатуры (табл. 6).

Таблица 6

Индивидуальные задания по вложенным циклам

Номер варианта	**Формула**	**Параметры**		
1	$Y = \frac{\cos(x)^2}{a^2 + 2}$	$a0 = 0.5$	$\Delta a = 0.4$	$ak = 2$
		$x = (0.1, 0.4, 1, 1.5, 2.7)$	$n = 5$	
2	$Y = \frac{\ln\lvert x+2\rvert}{a^2 + x}$	$a0 = -2$	$\Delta a = 0.8$	$n = 4$
		$x0 = -8$	$\Delta x = 0.5$	$xk = -4$
3	$Y = \sqrt{4x^2 + 3a}$	$a0 = 0.4$	$\Delta a = 0.6$	$n = 4$
		$x0 = 0$	$\Delta x = 1$	$xk = 7$
4	$Y = \text{arctg}(x - 2a)$	$a0 = 5$	$\Delta a = 0.2$	$ak = 6$
		$x = (8, 9.5, 14, 11.3, 13)$	$n = 5$	
5	$Y = \ln(a) + 5x$	$a0 = 0.5$	$\Delta a = 0.5$	$ak = 10$
		$x = (4.5, 4, 8.5, 6.3, 7.1)$	$n = 5$	
6	$Y = \frac{a^2 x}{\sqrt{x+1}}$	$a = (0.5, -1, 1.1, 2, 3.5, 4, 8.1)$	$n = 7$	
		$x0 = 0$	$\Delta x = 0.25$	$xk = 5$
7	$Y = a\ln(x) + 5$	$a0 = 2$	$\Delta a = 0.05$	$ak = 4.5$
		$x0 = 0.5$	$\Delta x = 0.25$	$n = 9$
8	$Y = a^2\text{tg}(x - 1)$	$a0 = 2$	$\Delta a = 0.5$	$n = 6$
		$x0 = -1$	$\Delta x = 1$	$xk = 5$
9	$Y = \ln\lvert x + a\rvert$	$a0 = 4$	$\Delta a = -0.5$	$n = 3$
		$x0 = 4$	$\Delta x = 0.4$	$xk = 6$
10	$Y = \frac{1}{\cos(\sqrt{x+a})}$	$a0 = 6$	$\Delta a = -0.4$	$ak = 4$
		$x0 = -0.5$	$\Delta x = 0.2$	$xk = 2.5$
11	$Y = e^{x^2} - a$	$a = (2.1, 2.7, 4, 5.5, -1, 0)$	$n = 6$	
		$x0 = 1.5$	$\Delta x = 0.5$	$xk = 4$
12	$Y = e^a - x$	$a0 = -2$	$\Delta a = 0.2$	$ak = 2$
		$x = (0.5, -2, 4.5, -5.3, 8.4)$	$n = 5$	
13	$Y = 2\sqrt{a+1}\cos(x)$	$a0 = 0$	$\Delta a = 0.25$	$ak = 2.5$
		$x0 = 0.1$	$\Delta x = 0.5$	$xk = 3.1$
14	$Y = \cos(a)^2 - x$	$a0 = 0$	$\Delta a = 0.6$	$ak = 2.4$
		$x0 = 3$	$\Delta x = -0.5$	$xk = 0.5$
15	$Y = \cos(a)^2 - x$	$a0 = 2$	$\Delta a = -0.5$	$ak = 4$
		$x0 = -4$	$\Delta x = 1$	$n = 6$

Задача 2.

Вычислить значение функции $y(x)=\begin{cases} x^2, & x \le 0, \\ x-58, & 0 < x \le 2, \\ \cos x, & 2 < x \le 5, \\ 0, & x > 5. \end{cases}$ в десяти точках, введенных с клавиатуры, и результат записать в массив.

Срок выполнения

Одно занятие.

Контрольные вопросы и задания

1. Что называется вложенным циклом? Каковы его функции в программе?
2. Что такое внешний цикл? Его функции в программе.
3. Внутренний цикл. Какова форма его записи в программе?
4. Назовите правила организации внешнего и внутреннего циклов.
5. Структура вложенного цикла: основные элементы и их описание.
6. Цикл с параметром. Его применение при создании вложенного цикла.
7. Цикл с предусловием. Его применение при создании вложенного цикла.
8. Цикл с постусловием. Его применение при создании вложенного цикла.

Лабораторная работа № 5
РАБОТА С МАССИВАМИ

Цель работы: изучить основные принципы и особенности работы с числовыми массивами.

Теоретические сведения

Объявление, определение и работа с массивами

Массив – это последовательность элементов одинакового типа, плотно расположенных в памяти друг за другом.

Массив может состоять из элементов любых типов, в том числе простых и составных, кроме функций, элементов типа *void*. Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве.

Объявление массива имеет два формата:

```
спецификатор_типа описатель [константное выражение];
    спецификатор_типа описатель [ ];
```

Описатель – это идентификатор массива. Описатель может быть простым идентификатором, если кроме идентификатора включает в себя признаки других типов [], (), *, то тогда тип элемента складывается из оставшейся части описателя и спецификации типа.

Пример

```
    int *a[10]; /* массив указателей на тип int; приоритет у скобок */
    int (*a)[10]; /*указатель на массив из 10 int*/
```

Спецификатор_типа задает тип элементов объявляемого массива.

Константное_выражение в квадратных скобках задает количество элементов массива.

Пример

```
    int a[10]; /* объявление целочисленного массива на 10 элементов */
    int a[10] = {0}; /* объявление целочисленного массива на 10 элементов и инициализация каждого элемента нулевым значением */
```

Константное_выражение при объявлении массива может отсутствовать, если:

1) при объявлении массив инициализируется:

```
    int a[]={1,2,3}; /* место в памяти отводится под три элемента целого типа*/
```

2) массив объявлен как формальный параметр функции;

3) массив объявлен как ссылка на массив, явно определенный в другом файле.

Если инициализация массива совмещается с объявлением, то элементов может быть меньше, чем в квадратных скобках. В этом случае место отводится для количества элементов, указанных в квадратных скобках, а проинициализированы будут от начала массива столько элементов, сколько реально указано в списке, остальные обнулятся.

Способ обращения к элементу массива:

```
<имя массива> [номер элемента]
```

Пример

```
a[1]; //обращение к первому элементу массива.
```

В C++ ни на стадии компиляции, ни на шаге запуска не отслеживается факт выхода за пределы массива.

Способы размещения массивов в памяти

Массивы в памяти программы можно размещать в статической и динамической памяти. В случае, когда на стадии разработки известны все сведения о массиве, количестве его элементов, такой массив можно размещать в статической памяти (`int a[10];`).

Если же программист знает, что в программе будет выполняться работа с массивом данных, но сами данные и их количество на момент разработки неизвестны, то в этом случае стоит воспользоваться размещением массива в динамической памяти.

Пример

```
#include <iostream>

int main() {
   using namespace std;
   int size, i = 0;

   cout<<"Введите количество элементов массива"<<endl;
   cin>>size; //ввод пользователем размера массива
   int* myarr = new int[size]; //выделение динамической памяти для хранения

   for (i = 0 ; i < size ; i++)
      myarr[i] = i;

   for (i = 0 ; i < size ; i++)
   printf_s("myarr[%d] = %d\n", i, myarr[i]);

   delete [] myarr;//освобождение памяти перед завершением работы программы
}
```

Многомерные массивы

В языке C++ определены только одномерные массивы, но поскольку элементом массива может быть массив, можно определить и многомерные массивы. Они формализуются списком *константных_выражений*, следующих за идентификатором массива, причем каждое *константное_выражение* заключается в свои квадратные скобки.

Каждое *константное_выражение* в квадратных скобках определяет число элементов по данному измерению массива, так что объявление двухмерного массива содержит два *константных_выражения*, трехмерного – три и т.д. Отметим, что в языке C++ первый элемент массива имеет индекс, равный ***нулю***.

Примеры

```
int a[2][3]; /* представлено в виде матрицы
a[0][0]a[0][1] a[0][2]
a[1][0]a[1][1] a[1][2]*/
double b[10]; /* массив из 10 элементов */
int w[3][3] = {{2, 3, 4},{3, 4, 8},{1, 0, 9 }};
```

В последнем примере объявление массива w[3][3] совмещено с заданием значений его элементам. Списки, выделенные в фигурные скобки, соответствуют строкам массива, в случае отсутствия скобок инициализация может быть выполнена неправильно.

В языке C++ можно использовать сечения массива, как и в других языках высокого уровня, однако на использование сечений накладывается ряд ограничений. Сечения формируются вследствие опускания одной или нескольких пар квадратных скобок. Пары квадратных скобок можно отбрасывать только справа налево и строго последовательно. Сечения массивов используются при организации вычислительного процесса в функциях языка C++, разрабатываемых пользователем.

Пример

```
int s[2][3];
```

Если в коде программы написать s[0], то это будет означать обращение ко всей нулевой строке массива (матрицы) *s*.

Пример

```
int b[2][3][4];
```

При обращении к массиву *b* можно написать, например b[1][2], и будет передаваться вектор из четырех элементов, а обращение b[1] даст двухмерный массив размерами 3 на 4. Нельзя написать b[2][4], подра-

зумевая, что передаваться будет вектор, потому что это не соответствует ограничению, наложенному на использование сечений массива.

Пример решения задания 1

Вычислить все элементы последовательности:

$a_1, a_1 + a_2, \ldots, a_1 + a_2 + \ldots + a_i + \ldots + a_n,$

если $a = (0.3, 0.2, 1.1, 1.5, -2.6, 2.2)$, $n = 6$.

```
#include <math.h>
#include <locale.h>
#include <iostream>
using namespace std;

int main()

{
    setlocale(LC_ALL,
"Russian");
    const int n = 6;
    float a[n] = {0.3, 0.2,
1.1, 1.5, -2.6, 2.2};
    float p = 0;
    for (int i=0;i<n;i++)
    {
        p += a[i];
        cout <<p <<",\t";
    }

    system("pause");
    return 0;
}
```

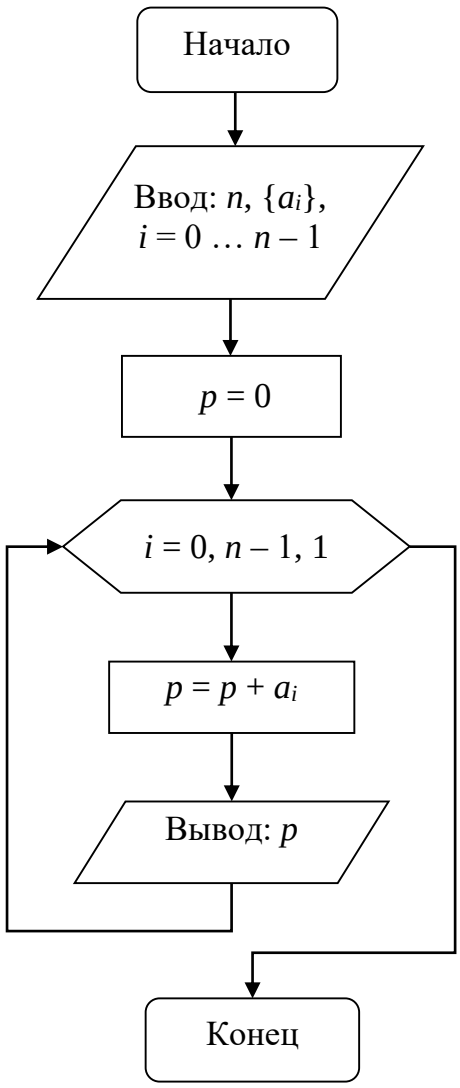

Рис. 18. Вычисление частичных сумм последовательности

Результаты вывода программы:

0.3, 0.5, 1.6, 3.1, 0.5, 2.7

Пример решения задания 2

Требуется найти количество отрицательных элементов в одномерном числовом массиве.

```
const int n = 10; // n - количество элементов в
массиве
float mas[n];
int i; // номер элемента массива, счетчик цикла
```

```
    int k=0; // количество отрицательных элементов
    for(i=0; i<n; i++) // цикл ввода элементов массива
    scanf("%f", &mas[i]);
    for(i=0; i<n; i++)
        {
            if(mas[i]<0) k++;
        }
        printf("Количество отрицательных элементов
%d\n", k);
```

Задания для самостоятельного выполнения

Задача 1. Составить комбинированный алгоритм, используя операции «накопления».

1. Вычислить все элементы последовательности:
a_1, $a_1 \cdot a_2$, …, $a_1 \cdot a_2 \cdot \ldots \cdot a_i \cdot \ldots \cdot a_n$, если a = (0.4, 0.9, 1.2, 1.7, 2.5, 2.1), n = 6.

2. Вычислить все элементы последовательности:
$|a_1|$, $|a_1 + a_2|$, $|a_1 + a_2 + a_3|$, …, $|a_1 + a_2 + \ldots + a_i + \ldots + a_n|$, если a = (–1.4, 0.9, 9.2, 7.7, –2.5, –2.1, 5.5), n = 7.

3. Вычислить
$B = a(a + 1)\,(a + 2) \cdot \ldots \cdot (a + i) \cdot \ldots \cdot (a + n)$, если n = 10, a = 4. Сумму каждой скобки следует вычислять накоплением, используя результат из предыдущей скобки, B – так же накапливать.

4. Вычислить, используя накопления числителя, знаменателя и всего результата:

$$A = \frac{\cos 1}{\sin 1}\frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} \cdot \ldots \cdot \frac{\cos 1 + \cos 2 + \ldots + \cos(i) + \ldots + \cos(n)}{\sin 1 + \sin 2 + \ldots + \sin(i) + \ldots + \sin(n)}.$$

5. Вычислить
$S = (1 + \sin 0.1)(1 + \sin 0.2) \cdot \ldots \cdot (1 + \sin n)$. Для вычисления аргумента функции sin использовать прием «накопление» суммы, и произведение так же накапливать.

6. Вычислить
$A = \frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} \cdot \ldots \cdot \frac{1}{\sin 1 + \sin 2 + \ldots + \sin(n)}$. Знаменатель каждой дроби и результат A получать накоплением суммы.

7. Вычислить $S = \sin(x) + \sin^2(x) + \ldots + \sin^n(x)$.

n, x – определяет пользователь. При решении операцию возведения в степень не использовать, а вычислять с помощью накопления произведения.

8. Вычислить $B = \frac{1}{a} + \frac{1}{a(a+1)} \cdot \ldots \cdot \frac{1}{a(a+1)\cdot \ldots \cdot (a+n)}$.

a, x – определяет пользователь. Знаменатель каждой дроби и результат *B* получать, используя прием «накопление».

9. Вычислить $S = \sum_{k=1}^{n} \left(\sum_{i=k}^{n} \frac{x+k}{i} \right)$.

n, x – определяет пользователь.

10. Вычислить $B = \sum_{k=1}^{n} \left(k(k+1) \cdot \ldots \cdot (2k) \right)$.

n – определяет пользователь.

11. Вычислить $B = \sin(x) + \sin(\sin(x)) + \ldots + \sin(\sin(\ldots(\sin(x))\ldots))$. Количество синусов в слагаемых равно их номерам.

12. Вычислить $S = \sum_{k=1}^{n} k^k$.

При решении операцию возведения в степень не использовать, а вычислять с помощью приема «накопление» произведения.

13. Вычислить $A = \sum_{k=1}^{n} x^{2k}$.

При решении операцию возведения в степень не использовать, а вычислять с помощью приема «накопление» произведения. *n, x* – определяет пользователь.

14. Вычислить $S = \sum_{k=1}^{n} \left(\prod_{i=k}^{n} \frac{x+k}{i} \right)$.

n, x – определяет пользователь.

15. Вычислить $B = \prod_{k=1}^{n} x^{2k}$. При решении операцию возведения в степень не использовать, а вычислять с помощью приема «накопление» произведения. *n, x* – определяет пользователь.

Задача 2. 1. Дана целочисленная прямоугольная матрица. Определить:

а) количество строк, не содержащих ни одного нулевого элемента;

б) максимальное из чисел, встречающихся в заданной матрице более одного раза.

2. Дана целочисленная прямоугольная матрица:

а) определить количество столбцов, не содержащих ни одного нулевого элемента.

б) характеристикой строки целочисленной матрицы назовём сумму её положительных чётных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.

3. Дана целочисленная прямоугольная матрица. Определить:

а) количество столбцов, содержащих хотя бы один нулевой элемент;

б) номер строки, в которой находится самая длинная серия одинаковых элементов.

4. Дана целочисленная квадратная матрица. Определить:

а) произведение элементов только в тех строках, которые не содержат отрицательных элементов;

б) максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

5. Дана целочисленная квадратная матрица. Определить:

а) сумму элементов в тех столбцах, которые не содержат отрицательных элементов;

б) минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

6. Дана целочисленная прямоугольная матрица:

а) характеристикой столбца целочисленной матрицы назовём сумму модулей его отрицательных нечётных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик;

б) найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

7. Дана целочисленная прямоугольная матрица. Определить:

а) сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент;

б) номера строк и столбцов всех седловых точек матрицы. Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-й строке и максимальным в j-м столбце.

8. Для заданной матрицы размерами 8 на 8:

а) найти такие k, что k-я строка матрицы совпадает с k-м столбцом;

б) найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

9. Соседями элемента A_{ij} в матрице назовём элементы A_{kl} с $i - 1 \le k \le i + 1,\ j - 1 \le l \le j + 1,\ (k, l) \ne (i, j)$. Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы:

а) построить результат сглаживания заданной вещественной матрицы размерами 10 на 10;

б) в сглаженной матрице найти сумму модулей элементов, разложенных ниже главной диагонали.

10. Дана матрица размерами 10 на 10:

а) элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы;

б) найти сумму модулей элементов, расположенных выше главной диагонали.

11. Дана прямоугольная матрица:

а) заданы коэффициенты системы линейных уравнений. С помощью допустимых преобразований привести систему к треугольному виду;

б) найти количество строк, среднее арифметическое элементов которых меньше заданной величины.

12. Дана прямоугольная матрица:

а) уплотнить заданную матрицу, удалив из неё строки и столбцы, заполненные нулями;

б) найти номер первой из строк матрицы, содержащих хотя бы один положительный элемент.

13. Выполнить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введённого режима). n может быть больше количества элементов в строке или столбце.

14. Выполнить циклический сдвиг элементов квадратной матрицы размерами $M \times N$ вправо на k элементов следующим образом:

элементы первой строки сдвигаются в последний столбец сверху вниз, из него – в последнюю строку; для остальных элементов – аналогично.

15. Дана целочисленная прямоугольная матрица:

а) определить номер первого из столбцов, содержащих хотя бы один нулевой элемент;

б) характеристикой строки целочисленной матрицы назовём сумму её отрицательных чётных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.

Срок выполнения

Три занятия.

Контрольные вопросы и задания

1. Дайте определение массива с точки зрения хранения.
2. Приведите общий принцип обращения к элементам массива.
3. С какого числа начинается нумерация элементов массива? Можно ли нумерацию изменить?
4. Каким образом можно задать размер массива?
5. Объявите трехмерный массив и изобразите его с точки зрения языка C++.
6. Объявите двумерный массив вещественных элементов размерами пять строк и четыре столбца. Далее:

a) проинициализируйте при объявлении;

b) организуйте заполнение массива с клавиатуры;

c) организуйте заполнение случайными числами (*rand*).

7. Посчитайте сумму элементов массива.
8. Может ли элементом массива быть массив?
9. Рассчитайте разность между максимальным и минимальным элементами массивов.
10. Как рассчитывается след матрицы?
11. Постройте алгоритм расчета среднего арифметического элементов одномерного вещественного массива данных.
12. Можно ли выполнить заполнение массива чисел случайными значениями? Если да, то приведите вариант программы.
13. Какой тип данных стоит использовать для описания матрицы чисел?

Лабораторная работа № 6

РАБОТА С СИМВОЛАМИ И СТРОКАМИ

Цель работы: изучить основные принципы программной обработки строковой и символьной информации и научиться применять для решения задач некоторые функций из библиотеки *string*.

Теоретические сведения

Символы и строки

Для представления символа используют тип *char*. Значение этого типа – код из таблицы ASCII (прил. 2). Сам тип интерпретируется как однобайтовая целая величина со знаком, диапазон значений которой равен от –128 до +127. Хотя только значения от 0 до +127 имеют символьные эквиваленты.

Символьные константы заключаются в апострофы, а символьные переменные могут быть либо символьными константами, которые можно вывести на печать, либо управляющими символьными константами в апострофах.

Строковая константа представляет собой последовательность символов, заключенных в двойные кавычки "Николенко".

В конце каждой переменной строкового типа компилятор добавляет символ конца строки '\0'.

Строковая переменная описывается как:

```
char имя [длина];
```

Строковая константа может располагаться в нескольких строках. Для их формирования используют обратную дробную черту «/» с последующим нажатием клавиши ENTER.

«Сибирская региональная /
школа бизнеса»

Для организации ввода/ вывода символьной и строковой информации кроме функций *printf* и *scanf* можно использовать следующие: `puts(); putchar(); gets(); getchar();`

Функция *putchar* выводит одиночный символ без перехода на новую строку. Функция *getchar* используется для ввода символа с клавиатуры.

```
char s;
s = getchar();
putchar(s);
```

Функция *puts* – используется для вывода строки символов, при этом автоматически дополняется символом '\n', т.е. переводит курсор на новую строку.

Функция *gets* – символы до тех пор, пока не нажата клавиша ENTER. Автоматически дополняют строку '\0'.

```
char a[15],*b;
gets (a); gets(b);
puts(a); puts(b);
```

Функции проверки символов и преобразования строк в другие типы данных

Перечисленные выше функции работают при помощи библиотеки *ctype.h*.

```
isalpha (s); // проверяет, является ли символ буквой.
isdigit (s); // проверяет, является ли символ цифрой.
islower (s); // проверяет, является ли символ строчной буквой.
isspace (s); // проверяет, является ли символ пустым (пробел).
isupper (s); // проверяет, является ли символ заглавной буквой.
isalnum (s); // проверяет, является ли символ алфавитно-цифровым.
isascii (s); // проверяет, является ли символ таблицей ASCII.
iscntrl (s); // проверяет, является ли символ управляющей константой.
ispunct (s); // проверяет, является ли символ знаком пунктуации.
```

При вводе информации иногда целесообразно использовать строку символов, которую можно преобразовать в соответствующее число. Для этого используют функции:

```
int atoi(строка); // преобразует строку в целое число int.
long atol(строка); // преобразует строку в целое число типа long.
double atof(строка); // преобразует строку в число с плавающей точкой.
```

Прототипы главной функции описаны в библиотеке *stdlib.h*; если строка не может быть преобразована, то функция возвращает ноль.

```
char c; int d;
gets(c);
d=atoi(c);
```

Функции работы со строками

Для использования специальных функций работы со строками необходимо подключить заголовочный файл *string.h*.

Приведем описание основных функций:

```
int strlen(const char *s) //Возвращает длину строки s.
char *strchr(const char *s, char c) //Отыскивает первое вхождение в строку s символа c. Возвращает указатель на символ c в строке s. В противном случае возвращает нулевой указатель.
char *strcpy(char * to, const char *from) //Копирует строку, на которую указывает from, в строку, на которую указывает to. Область, на которую указывает to, должна иметь достаточный размер для размещения копируемой строки.
char *strncpy(char *to, const char *from, int limit); //Аналогично предыдущей, но копирует не более чем limit символов.
int strcmp(const char *s1, const char *s2); //Функция сравнения двух строк. Возвращает -1,0 или +1, если s1 соответственно меньше, равна или больше s2.
int strncmp(const char *s1, const char *s2, size_t m); // Функция сравнения строк по первым m символам. Результат работы функции аналогичен strcmp
```

```
    int strcmpi(const char *s1, const char *s2);
//Функция сравнения строк без учетом регистра.
    int strncmpi(const char *s1, const char *s2,
size_t n); // Функция сравнения строк по первым m
символам без учета регистра.
    char *strcat(char *s1, const char *s2); // Функ-
ция конкатенации строк добавляет строку s2 в конец
строки s1.
```

Специальный класс string

Для его работы необходимо в начале программы подключить заголовочный файл *string*:

```
 #include <string>
```

В отличие от массива *char string* является классом. Более подробно о классах расскажем позднее, сейчас вам достаточно знать, что классы содержат в себе сразу несколько вещей: переменные, константы и функции для работы с переменными. Более подробно про классы и их объекты можно изучить в литературе по объектно-ориентированному программированию.

Для создания строки вам необходимо в начале программы написать *using namespace std*;

Далее, чтобы создать строку, достаточно написать:

```
string s;
```

Для записи в строку можно использовать оператор =

```
s="Hello";
```

Пример работы с классом *string*:

```
string name;
cout<<"Enter your name"<<endl;
cin.getline(name,256);
cout<<"Hi "<<s<<"!"<<endl;
```

Стоит отметить, что при работе со строкой *string* не требуется задавать размер строки. Но кроме этого существует множество функций для работы со строками.

```
    s.append(str); // добавляет в конец строки строку
str. Можно писать как s.append(переменная), так и
s.append("строка");
```

s.assign(str); // присваивает строке s значение строки str. Аналогично записи s=str.

int i=s.begin(); // записывает в i индекс первого элемента строки.

int i=s.end(); // аналогично, но последнего.

s.clear(); // как следует из названия, очищает строку, т.е. удаляет все элементы в ней.

s.compare(str); // сравнивает строку s со строкой str и возвращает 0 в случае совпадения (на самом деле сравнивает коды символов и возвращает их разность).

s.copy(куда, сколько, начиная с какого); // копирует из строки s в куда (там может быть как строка типа string, так и строка типа char). Последние два параметра необязательные (можно использовать функцию с 1,2 или 3 параметрами).

bool b=s.empty(); // если строка пуста, возвращает true, иначе false.

s.erase(откуда, сколько); //удаляет n элементов с заданной позиции.

s.find(str,позиция); // ищет строку str, начиная с заданной позиции.

s.insert(позиция,str, начиная, beg, count); // вставляет в строку s, начиная с заданной позиции, часть строки str, начиная с позиции beg и вставляя count символов.

int len=s.length(); // записывает в len длину строки.

s.push_back(symbol); // добавляет в конец строки символ.

s.replace(index, n,str); // берет n первых символов из str и заменяет символы строки s на них, начиная с позиции index.

str=s.substr(n,m); // возвращает m символов, начиная с позиции n.

s.swap(str); // меняет содержимое строк s и str местами.

s.size(); // возвращает число элементов в строке.

Пример

```
    string name, surname, text, fullname, s1, s2, s3,
user;
    user="Petya Petrov";
    cout<<"Enter your name"<<endl;
    cin>>name;
    cout<<"Enter your surname"<<endl;
    cin>>surname;
    fullname=name;
    fullname+=" "; // добавляем пробел
    fullname.append(surname);
    if (fullname.compare(user)==0)
//if(!(fullname.compare(user)))
        cout<<"Your are good user"<<endl;
    else
        cout<<"Bad user"<<endl;
    cout<<"enter s1"<<endl;
    cin>>s1;
    cout<<"enter s2"<<endl;
    cin>>s2;
    s1.swap(s2);
    cout<"new s1: "<<s1<<endl<<"new s2: "<<s2<<endl;
    cout<<"Enter big text with your name"<<endl;
    cin>>text;
    int i=0;
    i=text.find("name");
    while (i!=-1)
    {
        text.replace(i,name.length(),name);
        s3=text.substr(i,name.length());
        cout<<"Replaced: "<<s3<<endl;
        i=text.find("name");
    }
    cout<<"New text:"<<endl<<text<<endl;
    text.clear();
    cout<<"text: "<<text<<endl;
```

Задания для самостоятельного выполнения

При реализации воспользоваться двумя способами задания и работы со строками (массив символов с нулевым в конце и строка типа *string*).

1. Подсчитайте количество слов во введенной пользователем строке.

2. Поменяйте местами соседние слова в строках.

3. Переверните строку, т.е. последние символы должны стать первыми, а первые – последними.

4. Дана строка, состоящая из слов и чисел, отделенных друг от друга пробелами. Сформируйте три строки, одна из которых будет содержать только целые числа, встречающиеся в исходной строке, вторая – только вещественные числа, а третья – оставшиеся слова.

5. Добавьте в строку пробелы после знаков препинания, если они там отсутствуют. Удалите в строке все лишние пробелы (лишними считаются пробелы, следующие непосредственно за пробелами, т.е. между словами всегда должен находиться один пробел).

6. Определите, как часто встречается определенный символ в строке. Оставьте в строке только один символ из каждого встречающегося.

7. Найдите в строке определенную последовательность символов и замените ее другой.

8. Вставьте в заданную позицию строки другую строку.

9. Удалите из строки ее часть с заданной позиции и заданной длины.

10. Скопируйте часть строки с определенной позиции и определенной длины в другую строку.

11. Найдите в строке все заданные последовательности символов и замените их другой последовательностью.

12. Удалите из строки все слова, длина которых меньше пяти символов.

13. Требуется в произвольной строке удалить последнее слово, т.е. все символы после последнего пробела в строке.

14. Определите, является ли строка палиндромом. *Палиндром* – это число, слово или фраза, одинаково читающиеся в обоих направлениях.

15. Требуется выделить из строки (введенной пользователем) числа (только целые) и поместить их в массив, содержимое которого затем вывести на экран.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Что такое строка с точки зрения языка C++?
2. В чем отличие строки от массива символов?
3. Можно ли работать поэлементно со строкой?
4. Напишите программу, вычисляющую сумму цифр в строке вида "1ab3c405". Строку введите с клавиатуры.
5. Напишите фрагмент кода, выполняющий смену двух строк *str*1 и *str*2.

Лабораторная работа № 7
РАБОТА С УКАЗАТЕЛЯМИ

Цель работы: изучить назначение указателей, допустимые операции, способы работы с динамической памятью через указатели.

Теоретические сведения

Когда компилятор обрабатывает оператор определения переменной, например *int i* = 10, он выделяет память в соответствии с типом (*int*) и инициализирует ее указанным значением (10). Все обращения в программе к переменной по ее имени (*i*) заменяются компилятором на адрес области памяти, в которой хранится значение переменной. Можно определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями.

Так как указатель – это адрес некоторого объекта, то через него можно обращаться к этому объекту.

Пример:

```
y=&x /* переменной y присваивается адрес перемен-
ной x, указывающий, где в оперативной памяти нахо-
дится значение x */
    z=*y /* переменной z присваивается значение пе-
ременной, записанной по адресу y, если y=&x и z=*y,
то z=x */
```

В C++ различают три вида указателей:

✓ на объект;

✓ функцию;

✓ *void.*

Эти указатели отличаются свойствами и набором допустимых операций. Указатель, имеющий тип, соответствующий типу данных, хранящемуся в указателе, – типизированный указатель. Указатель на *void* – нетипизированный указатель.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного). Простейшее объявление указателя на объект имеет вид:

```
тип *имя;
```

где тип может быть любым – как базовым, так и пользовательским.

Модификатор звездочка при объявлении указателя относится непосредственно к имени переменной, поэтому для того чтобы объявить несколько указателей, требуется ставить ее перед именем каждого из них. Например, в операторе

```
int *a, b, *c;
```

описываются два указателя на целое с именами *a* и *c*, а также целая переменная *b*.

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов).

Указателю на *void* можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать тип.

Инициализация указателей

Указатели чаще всего используют при работе с динамической памятью, называемой «кучей». Это свободная память, в которой можно во время выполнения программы выделять место в соответствии с потребностями. Доступ к выделенным участкам динамической памяти, называемым *динамическими переменными,* производится только через указатели. Время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти. В C++

существуют два способа работы с динамической памятью. Первый использует семейство функций *malloc* и достался в наследство от Си, второй – операции *new* и *delete*.

При определении указателя надо стремиться выполнить его инициализацию, то есть присвоить начальное значение. Непреднамеренное использование неинициализированных указателей – распространенный источник ошибок в программах. Инициализатор записывается после имени указателя либо в круглых скобках, либо после знака равенства.

Существуют следующие способы инициализации указателя:

1. Присваивание указателю адреса существующего объекта:

с помощью операции получения адреса;

```
int a = 5; // целая переменная.
int* p = &a; //в указатель записывается адрес a.
int* p (&a); // то же самое другим способом;
```

с помощью значения другого инициализированного указателя:

```
int* u = p;
```

2. Присваивание указателю адреса области памяти в явном виде:

```
char* vp = (char *)OxB8000000;
```

Здесь 0xB8000000 – шестнадцатеричная константа, (char *) – операция приведения типа: константа преобразуется к типу «указатель на *char*» и будет программой трактоваться как адрес.

3. Присваивание пустого значения:

```
int* suxx = NULL;
int* rulez = 0;
```

пустой указатель можно использовать для проверки, ссылается указатель на конкретный объект или нет.

4. Выделение участка динамической памяти и присваивание ее адреса указателю:

а) с помощью операции new:

```
int* n = new int;    // 1
int* m = new int (10);   // 2
int* q = new int[10];    // 3
```

б) с помощью функции *malloc*, которая находится в библиотеке *stdlib.h*:

```
int* u1 = (int *)malloc(sizeof(int));// 4
int* u2 = (int *)malloc(5*sizeof(int));// 5
```

в) с помощью функции *calloc*, которая находится в библиотеке *stdlib.h*:

```
int* u3 = (int *)calloc(sizeof(int));// 6
int* u4 = (int *)calloc(5,sizeof(int));// 7
```

В операторе 1 операция *new* выполняет выделение достаточного для размещения величины типа *int* участка динамической памяти и записывает адрес начала этого участка в переменную *n*. Память под саму переменную *n* размером, достаточным для размещения указателя, выделяется на этапе компиляции.

В операторе 2, кроме описанных выше действий, производится инициализация выделенной динамической памяти значением 10.

В операторе 3 операция *new* выполняет выделение памяти под десять величин типа *int* (массива из десяти элементов) и записывает адрес начала этого участка в переменную *q*, которая может трактоваться как имя массива.

В операторе 4 делается то же самое, что и в операторе 1, но с помощью функции выделения памяти *malloc*, унаследованной из библиотеки Си. В функцию передается один параметр – количество выделяемой памяти в байтах. Конструкция (*int**) используется для приведения типа указателя, возвращаемого функцией, к требуемому типу. Если память выделить не удалось, функция возвращает *NULL*.

В операторе 5 происходит выделение участка памяти для хранения пяти целых чисел.

Операторы 6 и 7 эквивалентны операторам 4 и 5 соответственно.

Операцию *new* использовать предпочтительнее, чем функцию *malloc*, особенно при работе с объектами.

Освобождение памяти, выделенной с помощью операции *new*, должно выполняться с помощью *delete*, а памяти, выделенной функцией *malloc* или *calloc* – посредством функции *free*. При этом переменная-указатель сохраняется и может инициализироваться повторно. Приведенные выше динамические переменные уничтожаются следующим образом:

```
delete n;
delete m;
delete [] q;
free (u1); free (u2);
free (u3); free (u4);
```

Если память выделялась с помощью *new*[], для освобождения памяти необходимо применять *delete*[]. Размер массива при этом не указывается. Если квадратных скобок нет, то никакого сообщения об ошибке не выдается, но помечен как свободный будет только первый элемент массива, а остальные окажутся недоступны для дальнейших операций. Такие ячейки памяти называются «мусором».

Если переменная-указатель выходит из области своего действия, отведенная под нее память освобождается. Следовательно, динамическая переменная, на которую ссылался указатель, становится недоступной. При этом память из-под самой динамической переменной не освобождается. Другой случай появления «мусора» – когда инициализированному указателю присваивается значение другого указателя. При этом старое значение бесследно теряется.

Операции с указателями

С указателями можно производить следующие операции: присваивания и арифметические отношения.

Операция присваивания для указателей аналогична этой операции для других типов данных. В отличие от других типов при операции присваивания достаточно, чтобы оба операнда были типа «указатель», но типы этих указателей могут быть различными.

Пример:

```
int a;
int *p=&a, *p1;
*p = 8 /* значение 8 заносится по адресу p, т.е. значение переменой a=8*/
p1= p /* переменной p1 присваивается значение указателя p, которое равно адресу переменной a*/
```

Допустимые арифметические операции: +, – (соответственно +=, –=), ++, – –.

Арифметические действия с указателями учитывают тип переменной, на которую они указывают. Эти операции изменяют адрес указателя на величину, пропорциональную размеру типа данных.

Пример:

```
float f,*pf; /* указатель на вещественное число, в памяти вещественное занимает 4 байта */
pf=&f; /* например, переменная f хранится по адресу 200, тогда pf = 200 */
```

```
pf--; /* pf=196*/
pf+=4; /* pf=196+4*4=212*/
```

Арифметические операции с указателями (сложение с константой, вычитание, инкремент и декремент) автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например с массивами. *Инкремент* перемещает указатель к следующему элементу массива, *декремент* – к предыдущему. Фактически значение указателя изменяется на величину *sizeof(mun)*. Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа.

Выражение *(*p)++* инкрементирует значение, на которое ссылается указатель, при этом запись **p* означает доступ к значению объекта, на который настроен указатель *p*, и читается как «по указателю».

Пример

```
#include <stdio.h>
#include <conio.h>
void main()
{
   char *s, s1[10];
   scanf("%s", s1);
   s=s1;    /*настраиваем указатель на начало
массива s1 */
   printf("%c \n",*s);
   s += 4;  /*настраиваем указатель на четвер-
тый элемент массива */
   printf("%c", *s); /* выводим значение по
указателю */
   s++; //настраиваем указатель на пятый элемент
   printf("%c", *s); /* выводим значение по
указателю */
   s++; //настраиваем указатель на шестой элемент
   printf("%c", *s); /* выводим значение по
указателю */
   getch();
}
```

Допустимые операции отношений для указателей: ==, >=, <=, >, <, !=.

Пример

```
p1 == p2 /* результат операции ИСТИНА, если p1 и p2 указывают на один и тот же объект в оперативной памяти */
p1 >= p2 /* результат операции ИСТИНА, если переменная, на которую указывает p1, расположен в памяти правее (т.е. с большим адресом), чем p2 */
```

Унарная операция получения адреса & применима к величинам, имеющим имя и размещенным в оперативной памяти. Таким образом, нельзя получить адрес скалярного выражения, неименованной константы или регистровой переменной. Примеры операции приводились выше.

Указатель типа *void** используется для доступа к любым типам данных. Он указывает на место в оперативной памяти и не содержит информации о типе объекта, т.е. он указывает не на сам объект, а на его возможное расположение. К указателю типа *void* не применяются арифметические операции.

Указатели и константы

В операциях с указателями участвуют два объекта: сам указатель и объект, на который он ссылается. Помещение ключевого слова const перед объявлением указателя делает константным объект, а не указатель. Для объявления самого указателя в качестве константы используется оператор объявления * const, а не просто *.

Примеры:

```
const int *p1;// указатель на константу типа int
int const *p2;// указатель на константу типа int
int *const p3;/* константный указатель на объект типа int */
const int *const p4;/* константный указатель на константу типа int */
int const *const p4;/* константный указатель на константу типа int */
```

Первый и второй варианты записи являются синонимами и обозначают, что константа – объект, на который указывает указатель. То есть нельзя изменять значение, хранящееся в указываемом объекте.

Третий вариант указывает, что константен указатель, то есть его нельзя установить на другой объект – в него нельзя занести другой адрес в памяти.

Четвёртый и пятый варианты – синонимы, они указывают, что константен как указатель, так и указываемый объект.

Указатели на строки

Поскольку текстовая строка имеет тип const *char* [] и является массивом, к ней применимы все ранее приведённые соображения о массивах и указателях.

Строковый литерал можно присвоить переменной типа *char* *. Это сделано для совместимости с ранними версиями языка C, в которых не было ключевого слова const. Однако изменение строкового литерала через такой указатель является ошибкой:

```
void f() {
    char *p="text";
    p[3] = 'a'; // ОШИБКА
}
```

Память под строковые литералы выделяется статически, поэтому их свободно можно возвращать в качестве значения функции:

```
const char *access(int i)
{   …
    return "access denied";
}
```

Будут ли одинаковые литералы записываться в одно место памяти или нет – зависит от реализации.

Строки оканчиваются нуль-байтом (‘\0’), что делает их удобными для использования указателей. Рассмотрим копирование строк:

```
    const char src[]="Строка, которую надо скопировать";
    char dst[sizeof src];
    const char *p_src = src;
    char *p_dst = dst;
    while (*p_dst++ = *p_src++);
```

Создаются два указателя – константный указатель p_src, который хранит адрес копируемого элемента, и указатель p_dst, хранящий адрес, куда будет скопирован элемент. Копирование будет продолжаться до тех пор, пока не будет скопирован нуль-байт, завершающий строку.

Ссылки

Ссылка позволяет создать псевдоним (или второе имя) для переменных в вашей программе. Для объявления ссылки внутри программы укажите знак амперсанда (&) непосредственно после типа параметра. Объявляя ссылку, вы должны сразу же присвоить ей переменную, для которой эта ссылка будет псевдонимом, как показано ниже:

```
int& alias_name = variable; //Объявление ссылки
```

После объявления ссылки ваша программа может использовать или переменную, или ссылку:

```
alias_name = 1001;
variable = 1001;
```

В следующем примере создаем ссылку с именем *alias_name* и присваиваем псевдониму переменную *number*. Далее в коде используется как ссылка, так и переменная:

Пример

```
#include <iostream>
void main(void)
{
int number = 501;
int& alias_name = number; // Создать ссылку
cout << "Переменная number содержит" << number << endl;
cout << "Псевдоним для number содержит " << alias_name << endl;
alias_name = alias_name + 500;
cout << "Переменная number содержит " << number << endl;
cout << "Псевдоним для number содержит " << alias_name << endl;
}
```

Программа прибавляет 500 к ссылке *alias_name*. В итоге программа прибавляет 500 также и к соответствующей переменной *number*.

Результаты работы программы:

```
Переменная number содержит 501
Псевдоним для number содержит 501
Переменная number содержит 1001
Псевдоним для number содержит 1001
```

Использование ссылок значительно упрощает процесс изменения параметров внутри функции.

После своей инициализации ссылка не может быть изменена на ссылку на другой объект.

Для создания ссылки на постоянный объект необходимо выполнить следующие действия:

```
const double d = 10.5;
const double& rcd=d; /* объявление ссылки на неизменяемую переменную d */
```

Задания для самостоятельного выполнения

Для выполнения первого задания нужно написать тестовые программы для каждого задания с подробными комментариями по каждому пункту задания.

Разработать блок-схему алгоритма решения задания, написать и протестировать программы по обеим задачам. Оба задания выполняются каждым студентом.

Задание 1.

1. Объявите статические переменные, переменные ссылочных типов и указатели на *int, float, char, void*. Проинициализируйте указатели и ссылки. Выполните операции приведения на объявленных указателях и переменных ссылочных типов, используйте явное и неявное приведения типов.

2. Определите допустимые операции и объясните их смысл на указателях и переменных ссылочных типов.

3. Объявите константный указатель, константную ссылку, указатель на константу. Определите для всех объявленных переменных допустимые действия и приведите примеры допустимых и недопустимых использований.

4. Объявите указатель на указатель на переменную какого-либо базового типа, ссылку на ссылку на переменную какого-либо базового типа, проинициализируйте объявленные переменные. При инициализации используйте только статические переменные базовых типов. Обязательно используйте *void*.

5. Приведите пример всевозможного использования модификатора *const* в цепочках: указатель на указатель, ссылка на ссылку. Приведите примеры допустимого и недопустимого использования.

Задание 2.

Задана неквадратная матрица и два массива указателей. Первый массив указателей настройте на адреса максимальных элементов по строкам, второй массив указателей – на минимальные элементы по столбцам. Для доступа к элементам массива используйте:

1) косвенную адресацию;

2) индексные выражения.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Что такое указатель?

2. Перечислите проблемы, которые могут возникнуть при работе с неинициализированными указателями.

3. Каково назначение нетипизированного указателя? Каким образом можно объявить нетипизированный указатель?

4. Будет ли корректно работать следующий код? Если код работает некорректно, то внесите исправления.

```
int a = 5;
int *pf = &a;
float *p;
p = pf;
```

5. Объявите массив из трех указателей на вещественные переменные и задайте значения переменных через указатели.

6. Разместите в динамической памяти одномерный массив, двумерный массив.

7. Поясните, что объявлено, проинициализируйте все объявленные переменные и нарисуйте картинки в памяти и с точки зрения языка C++.

```
int (*pM)[3];
int *(*pMM)[2];
int m[2][3];
```

8. Напишите фрагмент программы, используя оператор выделения динамической памяти *new*. В программе должен выполняться захват памяти для пяти символов, ввод строки с клавиатуры и освобождение захваченной памяти.

9. Что такое ссылка?
10. Объявите ссылку на константу.

Лабораторная работа № 8
РАБОТА С ФУНКЦИЯМИ

Цель работы: изучить понятие «функция», правила описания, вызова функций, способы передачи параметров в функции.

Теоретические сведения

Функции позволяют программисту разбить программу на модули. Все переменные, объявленные в определениях функций, являются *локальными переменными* – они известны только той функции, в которой определены. Большинство функций имеют список *параметров.* Параметры позволяют функциям обмениваться информацией. Параметры функции – это также локальные переменные.

В программах, содержащих большое число функций, *main* (*_tmain*) должна быть реализована как группа обращений к функциям, выполняющим основную часть работы.

Каждая функция должна ограничиваться выполнением одной, точно определенной задачи, а имя функции должно отражать смысл данной задачи. Это облегчает абстракцию и способствует многократному использованию программного кода.

Существует несколько оснований для разбиения программы на функции. Подход «разделяй и властвуй» делает разработку программы более контролируемой. Другой мотив – это *повторное использование кода,* т.е. использование однажды написанных функций в качестве конструктивных блоков для создания новых программ. Многократное использование кода является основным определяющим фактором при переходе к объектно-ориентированному программированию. Имея хорошо продуманные имена и определения функций, можно создавать программы из стандартизованных модулей, не создавая специализированного программного кода. Эта методика известна как *абстракция.* Мы прибегаем к абстракции всякий раз, когда пишем программу, вызывающую функции стандартной библиотеки, например, *printf, scanf* и

pow. Третья причина – правило избегать дублирования кода в программе. Оформление кода в виде функции позволяет выполнять его в разных частях программы посредством простого вызова функции.

Прототип

Одна из наиболее важных особенностей ANSI Си – *прототипы функций*. Прототип функции сообщает компилятору тип данных, возвращаемых функцией, число параметров, получаемых функцией, тип и порядок следования параметров. Компилятор использует прототипы функций для проверки корректности обращений к функции. Прототип функции в общем виде:

```
тип_возвращаемого_значения имя_функции (список абстрактных типов);
```

Пример объявления прототипа функции, вычисляющей максимальное значение из трёх целочисленных значений.

```
int maximum (int, int, int);
```

Этот прототип объявляет, что *maximum* получает три параметра типа *int* и возвращает результат типа *int*. Обратите внимание, что прототип функции имеет тот же вид, что и первая строка определения функции *maximum*, за исключением того, что в него не включены имена параметров (*x*, *y* и *z*).

Имена параметров иногда включаются в прототип функции с целью документирования. Компилятор игнорирует эти имена. Пропуск точки с запятой в конце прототипа функции вызывает синтаксическую ошибку. Вызов функции, который не соответствует ее прототипу, вызывает синтаксическую ошибку. Ошибка генерируется также в том случае, если не согласованы прототип и определение функции. Например, если прототип функции записать в виде:

```
void maximum(int, int, int);
```

то компилятор зарегистрировал бы ошибку, поскольку возврат типа *void* в прототипе функции отличался бы от возврата типа *int* заголовка функции.

Другое важное следствие применения прототипов функций – *автоматическое приведение аргументов,* т.е. принудительное преобразование аргументов функции к соответствующему типу. Преобразование значений к более высоким типам происходит автоматически. Пре-

образование значений к более низким типам обычно приводит к неверному результату. Следовательно, значение может быть преобразовано в более низкий тип только посредством явного присваивания переменной более низкого типа или с помощью операции приведения. Значения аргументов функции преобразуются к типу параметров прототипа функции таким образом, как будто они непосредственно присваиваются переменным этого типа. Если прототип для данной функции не был включен в программу, компилятор формирует собственный прототип функции, используя ее первое вхождение в программу: или определение, или обращение к функции. По умолчанию компилятор предполагает, что функция возвращает тип *int*, ничего не предполагая относительно типа аргументов. Следовательно, если аргументы, переданные функции, некорректны, то компилятор не обнаружит ошибку.

Пропуск прототипа функции вызывает синтаксическую ошибку, если функция возвращает тип, отличный от *int*, а определение функции появляется в программе после вызова функции. В других случаях пропуск прототипа функции может вызвать ошибку во время выполнения программы или привести к непредвиденному результату. Прототип функции, размещенный вне любого определения функции, применяется ко всем вызовам, появляющимся в файле после прототипа функции. Прототип функции, помещенный в функцию, применяется только к тем вызовам, которые выполняются из этой функции.

Определение функции

Определение функции имеет следующий формат:

```
тип_возвращаемого_значения имя_функции(список_параметров)
{
    объявления и операторы;
};
```

В качестве имени функции *(имя_функции)* может использоваться любой допустимый идентификатор. Типом результата, возвращаемого вызывающей функции, является *тип_возвращаемого_значения.* Если *тип_возвращаемого_значения* задан ключевым словом *void*, то это означает, что функция ничего не возвращает. Если *тип_возвращаемого_значения* не указан, то компилятор будет предполагать тип *int*.

Если *тип_возвращаемого_значения* в определении функции опущен, то это вызовет синтаксическую ошибку в том случае, если прототип функции определяет, что возвращаемый тип не является целым (*int*).

Если функция должна возвращать некоторое значение, а в функции опущен оператор возврата, то это может привести к непредвиденным ошибкам. Стандарт *ANSI* гласит, что в этом случае результат не определен. Возврат значения из функции, для которой возвращаемый тип был объявлен как *void*, вызывает синтаксическую ошибку. Хотя опущенный возвращаемый тип по умолчанию расценивается как *int*, всегда задавайте возвращаемый тип явным образом. Однако для функции *main* возвращаемый тип обычно опускается.

Список_параметров – это список объявлений параметров (отделенных запятыми), получаемых функцией при ее вызове. Если функция не получает значения, *список_параметров* обозначается ключевым словом *void*. Тип каждого параметра должен быть указан явно за исключением параметров типа *int*. Если тип параметра не указан, то по умолчанию принимается тип *int*.

Замечания:

Ошибкой является запись *float x, y* вместо *float x, float y* при объявлении параметров функции, относящихся к одному типу. Такое объявление параметров, как *float x, y,* фактически присвоило бы параметру *y* тип *int*, поскольку *int* принимается по умолчанию.

Символ точки с запятой после правой круглой скобки, закрывающей список параметров в определении функции, является синтаксической ошибкой.

Повторное определение параметра функции как локальной переменной внутри функции – синтаксическая ошибка.

Объявления и *операторы* внутри фигурных скобок образуют *тело функции.* Тело функции называют *блоком.* Блок – это просто составной оператор, который включает в себя объявления. Переменные могут быть объявлены в любом блоке, а блоки могут быть вложены. *В любом случае функция не может быть определена внутри другой функции.* Определение функции внутри другой функции – синтаксическая ошибка.

Выбор осмысленных имен функций и параметров делает программы более удобочитаемыми и помогает избежать чрезмерного использования комментариев.

Программы должны составляться в виде совокупности небольших функций. Это упрощает создание программ, их отладку, поддержку и модификацию. Функция, требующая большого количества параметров, может выполнять слишком много задач. Рассмотрите возможность ее разделения на меньшие функции, которые выполняют выделенные задачи. Заголовок функции должен, по возможности, умещаться на одной строке.

Прототип функции, заголовок функции и вызов функции должны иметь взаимное соответствие по числу, типу и порядку следования аргументов и параметров, а также по типу возвращаемого значения.

Существуют три способа возвращения управления в ту точку программы, в которой была вызвана функция. Если функция не возвращает результат, управление возвращается просто при достижении правой фигурной скобки, завершающей функцию, или при исполнении оператора:

```
return;
```

Если функция возвращает результат, применяется оператор:

```
return выражение;
```

который возвращает вызывающей функции значение *выражения.*

Способы обмена данными

Существуют три способа передачи параметров в функции:

1) по значению;

2) по указателю;

3) по ссылке.

Приведём пример передачи параметров по значению:

Пример

```
#include <iostream>
double max(double x1, double x2){
    //передача параметров по значению
    if(x1>x2)return x1; //выход из функции и возврат значения
    else return x2; //выход из функции и возврат значения
}
```

При вызове функции указывается ее идентификатор и фактические параметры. Функция, возвращающая значение, обычно вызывается в выражении.

Пример

```
int main(){
    double a1=3;a2=2;a3;
    a3=max(a1,a2); //вызов функции,
    //a1 и a2 – фактические параметры
    cout<<a3;
    return 0;
}
```

Если функция не возвращает значения, то используется оператор `return` без параметров. В этом случае оператор `return` может быть опущен. Передача адреса переменной позволяет изменять значение внешних переменных внутри функции.

Пример

```
void swap(double *a, double *b){ //передача па-
раметров
    //по значению, но значениями являются адреса
a и b
    double c;
    c=*a;
    *a=*b;
    *b=c;
}
main(){
    double x=5,y=1;
    swap(&x,&y);
    return 0;
}
```

Передача параметра по ссылке реализуется как передача адреса параметра. Так как ссылка по определению есть синоним переменной, то при вызове функции формальный параметр становится синонимом фактического параметра. Поэтому изменения параметра, переданного по ссылке, вызовут изменения внешнего фактического параметра.

Пример

```
void swap(double &a, double &b){
    double c;
    c=a;
    a=b;
    b=c;
```

```
}
main() {
   double x=5, y=1;
   swap(x,y); //в качестве фактических параметров
   //передаются синонимы
   return 0;
}
```

В C++ отсутствует высокоуровневое понятие массива, поэтому передача массива в качестве параметра сводится к передаче указателя на начальный элемент массива. Следствие этого – необходимость передачи в качестве параметра функции размера массива.

Пример

```
double sum(int n, double *a) {
    double s=0.0;
    for(int i=0;i<n;i++)
    s=s+a[i];
    return s;
}
int main(){
    double  arr[3]={3,2,1};//arr  –  константный
указатель на массив
    cout<<sum(3,arr);
    return 0;
}
```

Для передачи адреса массива возможны следующие эквивалентные формы записи:

Пример

```
double sum(int n, double a[10]){...}
double sum(int n, double a[]){...}
double sum(int n, double *a){...}
```

Передача параметров по значению при вызове функции предполагает, что копия параметра помещается в программный стек. Кроме этого все локальные переменные функции размещаются в стеке при каждом входе в процедуру, что позволяет реализовать рекурсивный вызов функций, то есть вызов функции в своем же теле.

Пример

```
int fact(int k){
    if(k=1)
```

```
        return 1;
        else
        return k*fact(k-1); //рекурсивный вызов
    }
    main(){
        cout<<fact(3);
    }
```

Вызов функции

Во многих языках программирования существуют два способа вызова функций – *вызов по значению* и *вызов по ссылке.* Когда аргумент используется в вызове по значению, то вызываемой функции передается *копия* значения аргумента. Изменения, происходящие с копией, не отражаются на значении исходной переменной в вызывающей функции. Когда аргумент функции передается по ссылке, вызывающая функция фактически позволяет вызываемой функции изменять значение исходной переменной.

Передача аргумента по значению должна использоваться, когда вызываемой функции не нужно менять значение исходной переменной в вызывающей функции. Это предотвращает случайные *побочные эффекты,* которые мешают разработке правильных и надежных систем программного обеспечения. Передача аргумента по ссылке должна применяться только с «доверенными» вызываемыми функциями, которым необходимо менять первоначальную.

Пример

Найти сумму элементов по строкам в двумерном массиве, результат разместить в одномерном массиве.

```
#include "stdafx.h"
#include <stdlib.h>
#include <locale.h>
#include <time.h>

/*A, n, m - входные параметры, переданные по значению, b - выходной параметр, переданный по ссылке*/
   void S( int **const A, int n, int m, int*& b); //объявление функции
```

```
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "Russian");
 srand((unsigned)time(NULL)); //генератор раз-
броса случайных чисел
 int n, m; // n - количество строк, m - количе-
ство столбцов
 int **A, *b;
 inti, j; // номера элементов массива, счетчики
цикла
 printf("Введите количество строк в матрице: ");
 scanf("%d", &n);
 printf("Введите количество столбцов в матрице: ");
 scanf("%d", &m);
 // размещение в памяти матрицы A и вектора b
 b = new int[n];
 A = new int*[n];
 for(i=0; i<n; i++)
     A[i] = new int[m];
 // формирование матрицы
 for(i=0; i<n; i++)
 {
     for(j=0; j<m; j++)
     {
         A[i][j] = rand()%100; //заполнение слу-
чайными числами
         printf("%d ", A[i][j]);
     }
     printf("\n");
 }
 //вызов функции расчета сумм с передачей факти-
ческих параметров
 S(A, n, m, b);

 printf("Суммы по строкам:\n");
 for(i=0; i<n; i++) // цикл вывода результатов
подсчета сумм
     printf("%d %d\n", i+1, b[i]);
```

```
 return 0;
}

void S( int **const A, int n, int m, int*& b)
{
 for(int i=0; i<n; i++)
 {
     b[i]=0;
     for(int j=0; j<m; j++)
     {
         b[i]+=A[i][j];
     }
 }
}
```

Задания для самостоятельного выполнения

Для выполнения задания подгруппа разделяется на пары, совместно с преподавателем распределяются разрабатываемые функции. По результатам работы необходимо подготовить групповой отчет.

Для тестирования программы требуется создать меню, в котором необходимо предусмотреть возможности тестирования всех разработанных функций. Программа должна завершать свою работу по команде пользователя «Выход».

Для выполнения задания потребуется вспомнить правила выполнения операций над векторами и матрицами, изученные в курсе высшей математики.

Задача. Разработать и протестировать набор функций по работе с матрицами (двумерными массивами) и векторами (одномерными массивами). Должны быть реализованы функции для выполнения расчетов:

1) длины вектора;

2) скалярного произведения векторов;

3) умножения вектора на число;

4) умножения матрицы на число;

5) умножения матрицы на вектор;

6) умножения матрицы на матрицу, с отслеживанием возможности выполнения операции;

7) сложения матриц;
8) вычисления определителя;
9) функции ввода/вывода векторов и матриц.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Что такое функция?
2. Для чего применяют прототипы функций?
3. В чем заключаются отличия объявления и определения функции?
4. Что означает передача аргумента в функцию по ссылке? Приведите пример.
5. Верните из функции массив указателей на целочисленные переменные. Продемонстрируйте работу функции.
6. Каким образом можно передавать одномерные массивы в функции? Приведите примеры.
7. Поясните принцип передачи многомерных массивов в функции.
8. Напишите прототип функции *showCircle*, принимающей на вход координаты центра и радиус окружности.
9. Какое ключевое слово для параметра требуется указать, если мы хотим передать параметр по ссылке без возможности его изменения?

Лабораторная работа № 9
РАБОТА СО СТРУКТУРАМИ

Цель работы: изучить возможности языка C++ по созданию собственных типов данных на примере структурного типа.

Теоретические сведения

Структура – это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов. В языке C++ структура является видом класса и обладает

всеми его свойствами, но во многих случаях достаточно использовать структуры так, как они определены в языке Си:

```
struct [ имя_типа ]
{
    тип_1 элемент_1;
    тип_2 элемент_2;
    ...
    тип_n элемент_n;
} [ список_описателей ];
```

Элементы структуры называются *полями структуры* и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка.

Пример

```
struct {
    int year;
    char month;
    int day;
} date, date2;
```

Если список отсутствует, описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами.

Пример

```
struct student{ // описание нового типа student
char fio[30];
long int num_zac;
double sr_bal;
}; // описание заканчивается точкой с запятой

student gr[30], *p; /* определение массива типа
student и указателя на тип student.*/
```

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

Пример

```
struct {
    char fio[30];
```

```
    long int num_zac;
    double sr_bal;
}student1 = {"Necto", 011999,3.66};
```

При инициализации массивов структур следует заключать в фигурные скобки каждый элемент массива (учитывая, что многомерный массив – это массив массивов).

Пример

```
struct complex{
float re, im;
} compl [2][3] = {{{1, 1}, {1, 1}, {1, 1}},{{2, 2}, {2, 2}, {2, 2}}};
```

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции. Размер структуры не обязательно равен сумме размеров ее элементов, поскольку они могут быть выровнены по границам слова.

Доступ к полям структуры выполняется с помощью операций выбора . (точка) при обращении к полю через имя структуры и операции –> при обращении через указатель.

Пример

```
student student1, gr[30], *p;
student.fio = "Иванов";
gr[8].sr_bal=5;
p->num_zac = 012001;
```

Если элементом структуры является другая структура (вложенные структуры), то доступ к ее элементам выполняется через две операции выбора.

Пример

```
struct A {
    int a;
    double x;
};
struct B {
    A a;
    double x,
} x[2];
```

```
x[0].a.a = 1;
x[1]. x = 0.1;
```

Из примера видно, поля разных структур могут иметь одинаковые имена, поскольку у них разная область видимости. Более того, можно объявлять в одной области видимости структуру и другой объект (например, переменную или массив) с одинаковыми именами, если при определении структурной переменной использовать слово *struct*.

Пример

Описать структуру с именем *avto*, содержащую поля:

fam – фамилия и инициалы владельца автомобиля;

marka – марка автомобиля;

nomer – номер автомобиля.

Написать программу, выполняющую следующие действия:

ввод с клавиатуры данных в массив *gai*, состоящий из шести элементов типа *avto*; записи должны быть упорядочены в алфавитном порядке по фамилиям и именам владельцев;

вывод на экран информации о владельцах автомобиля, марка которого вводится с клавиатуры; если таковых автомобилей нет, то на экран дисплея вывести соответствующее сообщение.

Программа решения задачи будет иметь вид

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <windows.h>
char buf[256];

struct avto {
   char fam[25];
   char marka[20];
   char nomer[10];};

void output_gai(avto *,int );

void main(void)
{
   int i,n;
```

```
    bool flag;
    cout<< "Введите количество записей: ";
    cin>>n;
    avto *gai = new avto[n], temp;
    char m[20];
    for(i=0;i<n;i++)
    {// Ввод данных
       cout<< "Введите фамилию владельца "<<i+1;
       cout<< "-го автомобиля: ";
       cin >> gai[i].fam;
       cout<< "Введите марку "<<i+1;
       cout<< "-го автомобиля: ";
       cin >> gai[i].marka;
       cout<< "Введите номер "<<i+1;
       cout<< "-го автомобиля: ";
       cin >> gai[i].nomer;
    }
    cout<<  "Исходные  данные  до  сортировки"
<<endl;
    output_gai(gai,n);
    // Сортировка
    flag = true;
    while(flag)
    {
       flag = false;
       for(i=0;i<n-1;i++)
       if(strcmp(gai[i].fam,gai[i+1].fam)>0)
       {
         temp=gai[i];
         gai[i]=gai[i+1];
         gai[i+1]=temp;
         flag = true;
     }
}
cout<<endl<< "После сортировки"<<endl;
output_gai(gai,n);
cout<<endl<< "Введите марку автомобиля: ";
cin>>m;
```

```
cout<<endl<< "Искомые автомобили" <<endl;
flag = true;
for(i=0;i<n;i++)
if(!strcmp(m,gai[i].marka))
{
     output_gai(&gai[i],1);
     flag = false;
}
if(flag)
{
     cout<< "\nМарки "<<m;
     cout<< "нет в списках";}
     delete [] gai;
}
}
void output_gai( avto *gai, int n)
{
int i;
 for(i=0;i<n;i++)
 {
     cout.setf(ios::left);
     cout.width(15);
     cout<<gai[i].fam;// *(gai+i)->fam;
     cout.width(15);
     cout<<gai[i].marka;
     cout.setf(ios::right);
     cout.width(5);
     cout<<gai[i].nomer<<endl;
 }
}
```

Задания для самостоятельного выполнения

Требуется описать тип данных и набор функций (ввод данных для одной структуры, вывод информации одной структуры, сортировка, поиск) по работе с этим типом. В главной функции должен быть сценарий работы, описывающий выполнение задания по варианту, в котором осуществляется вызов описанных функций. Всю строковую информацию требуется хранить в динамической памяти.

Вариант 1.

Описать структуру с именем *STUDENT*, содержащую следующие поля:

- ✓ фамилия и инициалы;
- ✓ номер группы;
- ✓ успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

- ➢ ввод с клавиатуры данных в массив, состоящий из десяти структур типа *STUDENT*; записи должны быть упорядочены по возрастанию номера группы;
- ➢ вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0. Если таких студентов нет, вывести соответствующее сообщение.

Вариант 2.

Описать структуру с именем *STUDENT*, содержащую следующие поля:

- ✓ фамилия и инициалы;
- ✓ номер группы;
- ✓ успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

- ➢ ввод с клавиатуры данных в массив, состоящий из десяти структур типа *STUDENT*; записи должны быть упорядочены по алфавиту;
- ➢ вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку «2». Если таких студентов нет, вывести соответствующее сообщение.

Вариант 3.

Описать структуру с именем *AEROFLOT*, содержащую следующие поля:

- ✓ название пункта назначения рейса;
- ✓ номер рейса;
- ✓ тип самолета.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из семи элементов типа *AEROFLOT*; записи должны быть упорядочены по возрастанию номера рейса;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры. Если таких рейсов нет, выдать на экран соответствующее сообщение.

Вариант 4.

Описать структуру с именем *WORKER*, содержащую следующие поля:

- ✓ фамилия и инициалы работника;
- ✓ название занимаемой должности;
- ✓ год поступления на работу.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из десяти элементов типа *WORKER*; записи должны быть размещены по алфавиту;
- вывод на экран фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры. Если таких работников нет, выдать на экран соответствующее сообщение.

Вариант 5.

Описать структуру с именем *TRAIN*, содержащую следующие поля:

- ✓ название пункта назначения;
- ✓ номер поезда;
- ✓ время отправления.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *TRAIN*; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени. Если таких поездов нет, выдать на экран соответствующее сообщение.

Вариант 6.

Описать структуру с именем *TRAIN*, содержащую следующие поля:

- ✓ название пункта назначения;
- ✓ номер поезда;
- ✓ время отправления.

Написать программу, выполняющую следующие действия:

- ➢ ввод с клавиатуры данных в массив, состоящий из шести элементов типа *TRAIN*; записи должны быть упорядочены по времени отправления поездов;
- ➢ вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры. Если таких поездов нет, выдать на экран соответствующее сообщение.

Вариант 7.

Описать структуру с именем *MARSH*, содержащую следующие поля:

- ✓ название начального пункта маршрута;
- ✓ название конечного пункт маршрута;
- ✓ номер маршрута.

Написать программу, выполняющую следующие действия:

- ➢ ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *MARSH*; записи должны быть упорядочены по номерам маршрутов;
- ➢ вывод на экран информации о маршруте, номер которого введен с клавиатуры. Если таких маршрутов нет, выдать на экран соответствующее сообщение.

Вариант 8.

Описать структуру с именем *MARSH*, содержащую следующие поля:

- ✓ название начального пункта маршрута;
- ✓ название конечного пункта маршрута;
- ✓ номер маршрута.

Написать программу, выполняющую следующие действия:

- ➢ ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *MARSH*; записи должны быть упорядочены по номерам маршрутов;

- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры. Если таких маршрутов нет, выдать на экран соответствующее сообщение.

Вариант 9.

Описать структуру с именем *NOTE*, содержащую следующие поля:

- ✓ фамилия, имя;
- ✓ номер телефона;
- ✓ дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *NOTE*; записи должны быть упорядочены по датам рождения;
- вывод на экран информации о человеке, номер телефона которого введен с клавиатуры. Если такого нет, выдать на экран соответствующее сообщение.

Вариант 10.

Описать структуру с именем *NOTE,* содержащую следующие поля:

- ✓ фамилия, имя;
- ✓ номер телефона;
- ✓ дата рождения (массив из трех чисел);
- ✓ e-mail.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *NOTE*; записи должны быть упорядочены по трем первым цифрам номера телефона;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры. Если такого нет, выдать на экран соответствующее сообщение.

Вариант 11.

Описать структуру с именем *ZNAK*, содержащую следующие поля:

- ✓ фамилия, имя;
- ✓ знак зодиака;
- ✓ дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *ZNAK*; записи должны быть упорядочены по датам рождения;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры. Если такого нет, выдать на экран соответствующее сообщение.

Вариант 12.

Описать структуру с именем *ZNAK*, содержащую следующие поля:

- ✓ фамилия, имя;
- ✓ знак зодиака;
- ✓ дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *ZNAK*; записи должны быть упорядочены по знакам зодиака;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры. Если таких нет, выдать на экран соответствующее сообщение.

Вариант 13.

Описать структуру с именем *PRICE*, содержащую следующие поля:

- ✓ название товара;
- ✓ название магазина, в котором продается товар;
- ✓ стоимость товара в рублях.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *PRICE*; записи должны быть размещены в алфавитном порядке по названиям товара;
- вывод на экран информации о товаре, название которого введено с клавиатуры. Если таких товаров нет, выдать на экран соответствующее сообщение.

Вариант 14.

Описать структуру с именем *PRICE*, содержащую следующие поля:

- ✓ название товара;
- ✓ название магазина, в котором продается товар;
- ✓ стоимость товара в рублях.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *PRICE*; записи должны быть размещены в алфавитном порядке по названиям магазинов;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры. Если такого магазина нет, выдать на экран соответствующее сообщение.

Вариант 15.

Описать структуру с именем *ORDER*, содержащую следующие поля:

- ✓ расчетный счет плательщика;
- ✓ расчетный счет получателя;
- ✓ перечисляемая сумма в рублях.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа *ORDER*; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры. Если такого расчетного счета нет, выдать на экран соответствующее сообщение.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Назовите отличия структуры от массива.
2. Назовите операции доступа к полям структуры по указателю и через объект.
3. Каково назначение битовых полей? Особенности работы с ними.
4. Напишите функцию ввода с клавиатуры переменной структурного типа (структура STUDENT содержит поля: ФИО, дата рождения, успеваемость по пяти дисциплинам). Сформированные структуры из функции получите следующими способами:

а) в качестве параметра функции;

б) в качестве возвращаемого значения.

Продемонстрируйте работу функции.

Лабораторная работа № 10

РАБОТА С ФАЙЛАМИ

Цель работы: ознакомиться с понятием файла, изучить правила программной работы с файлами для манипуляции данными.

Теоретические сведения

Файлом называют способ хранения информации на физическом устройстве. Файл – это понятие, которое применимо ко всему – от файла на диске до терминала.

В C++ отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в *единое абстрактное логическое устройство*, называемое **потоком**.

Текстовый поток – это последовательность символов. При передаче символов из потока на экран часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток – это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Организация работы с файлами средствами C

Объявление файла

```
FILE *идентификатор;
```

Пример

```
FILE *f;
```

Открытие файла:

```
fopen(имя физического файла, режим доступа);
```

Режим доступа – строка, указывающая режим открытия и тип файла (табл. 7).

Типы файла: **бинарный** (b); **текстовый** (t).

Таблица 7

Режимы работы с файлами

Значение	Описание
r	Файл открывается только для чтения
w	Файл открывается только для записи. Если соответствующий физический файл существует, он будет перезаписан
a	Файл открывается для записи в конец (для дозаписи) или создается, если не существует
r+	Файл открывается для чтения и записи
w+	Файл открывается для записи и чтения. Если соответствующий физический файл существует, он будет перезаписан
a+	Файл открывается для записи в конец (для дозаписи) или создается, если не существует

Например,

```
f = fopen(s, "wb");
k = fopen("h:\ex.dat", "rb");
```

Неформатированный файловый ввод-вывод

Запись в файл:

```
fwrite(адрес записываемой величины, размер одного экземпляра, количество записываемых величин, имя логического файла);
```

Например,

```
fwrite(&dat, sizeof(int), 1, f);
```

Чтение из файла:

```
fread(адрес величины, размер одного экземпляра, количество считываемых величин, имя логического файла);
```

Например,

```
fread(&dat, sizeof(int), 1, f);
```

Закрытие файла:

```
fclose(имя логического файла);
```

Пример

Заполнить файл некоторым количеством целых случайных чисел.

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

```
using namespace std;

int main()
{
    FILE *f; int dat;
    srand(time(0));
    int n=rand()%30 + 1; /*генерируем количество
чисел*/
    cout << "File name? ";
    char s[20];
    cin.getline(s, 20); //вводим имя файла
    f=fopen(s, "wb");   /*открываем  файл  для
бинарной записи*/
    for (int i=1; i<=n; i++)
    {
        dat = rand()%101 - 50; /*генерируем слу-
     чайное число*/
        cout << dat << " "; /*выводим на экран
для контроля*/
        fwrite(&dat, sizeof(int), 1, f); /*за-
писываем число в файл*/
    }
    cout << endl;
    fclose(f); //закрываем файл
 system("pause");
 return 0;
}
```

Пример

Найти сумму и количество целых чисел, записанных в бинарный файл.

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

int main()
{
    FILE *f;
    int dat, n=0, sum=0;
```

```
    cout << "File name? ";
    char s[20];
    cin.getline(s, 20); /*вводим имя файла*/
    f=fopen(s, "rb"); /*открываем файл*/
   /*далее в цикле до тех пор, пока считываются
числа из файла*/
    while (fread(&dat, sizeof(int), 1, f))
   {
      n++; //считаем количество чисел
      cout << dat << " "; /*выводим на экран для
   проверки корректности расчетов*/
      sum+=dat;
   }
    cout << endl;
    cout << "Сумма: " << sum << "; Количество "
<< n << endl;
    fclose(f); //закрываем файл
    system("pause");
    return 0;
}
```

Форматированный файловый ввод-вывод

1. Функции *fgetc*() и *fputc*() позволяют соответственно осуществить ввод-вывод символа.

2. Функции *fgets*() и *fputs*() позволяют соответственно осуществить ввод-вывод строки.

3. Функции *fscanf*() и *fprintf*() позволяют соответственно осуществить форматированный ввод-вывод и аналогичный соответствующим функциям форматированного ввода-вывода, только делают это применительно к файлу.

Организация работы с файлами средствами C++

Файловый ввод-вывод с использованием потоков

Библиотека потокового ввода-вывода *fstream*.

Связь файла с потоком вывода:

```
ofstream имя логического файла;
```

Связь файла с потоком ввода:

```
ifstream имя логического файла;
```

Открытие файла:

```
имя логического файла.open(имя физического файла);
```

Закрытие файла:

```
имя логического файла.close();
```

Пример

Заполнить файл значениями функции $y = x\cos x$.

```
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

double fun(double x);

int main()
{
   double a, b, h, x; char s[20];
   cout << "Enter the beginning and end of the
segment, / /step-tabulation: ";
   cin >> a >> b >> h;
   cout << "File name? "; cin >> s;
   ofstream f;
   f.open(s);
   for (x=a; x<=b; x+=h)
   {
      f.width(10); f << x;
      f.width(15); f << fun(x) << endl;
   }
   f.close();
   system("pause");
   return 0;
}
double fun(double x)
{
```

```
   return x*cos(x);
}
```

Пример

Файл содержит несколько строк, в каждой из которых записано единственное выражение вида *a#b* (без ошибок), где *a, b* – целочисленные величины, # – операция +, –, /, *. Вывести каждое из выражений и их значения.

```
#include<cstdlib>
#include<iostream>
#include <fstream>
using namespace std;
int main()
{
long a, b;
char s[256], c;
int i;
cout << "File name? ";
cin >> s;
ifstream f;
f.open(s);
 while (!f.eof())
 {
 f.getline(s, 256);
i=0;
a=0;
while (s[i]>='0'&&s[i]<='9')
{
 a=a*10+s[i]-'0';
 i++;
}
 c=s[i++];
b=0;
 while (s[i]>='0'&&s[i]<='9')
{
 b=b*10+s[i]-'0';
 i++;
 }
 switch (c){
```

```
 case '+': a+=b; break;
 case '-': a-=b; break;
 case '/': a/=b; break;
 case '*': a*=b; break;
 }
 cout << s << " = " << a << endl;
}
 f.close();
 system("pause");
return 0;
}
```

Задания для самостоятельного выполнения

1. Напишите программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

2. Напишите программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.

3. Напишите программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

4. Напишите программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

5. Напишите программу, которая считывает текст из файла и определяет, сколько в нём слов, состоящих из не более чем четырёх букв.

6. Напишите программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключённые в кавычки.

7. Напишите программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

8. Напишите программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся или оканчивающиеся на гласные буквы.

9. Напишите программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

10. Напишите программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

11. Напишите программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один», … «девять», начиная каждое предложение с новой строки.

12. Напишите программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

13. Напишите программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.

14. Напишите программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нём введённое с клавиатуры слово.

15. Напишите программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.

Срок выполнения

Два занятия.

Контрольные вопросы и задания

1. Что такое файл?
2. Приведите отличия логического и физического файлов.
3. Перечислите типы файлов.
4. Приведите классификацию файлов по способам доступа к информации.
5. Какие действия необходимо совершить для работы с файлом?
6. Какая функция используется для открытия файла? Опишите параметры функции.
7. Каким образом можно определить, достигнут ли конец файла?
8. Напишите программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.
9. Если требуется быстро выполнить копирование файлов неизвестной структуры, какого типа файл нужно использовать?

ЗАКЛЮЧЕНИЕ

Подготовка специалистов в области лазерной техники и лазерных технологий, нанотехнологий предполагает выполнение работ по проведению различных экспериментов и обработке их результатов, контролю производственных процессов, в которых могут быть задействованы лазерные технологии и нанотехнологии. Это лишь небольшой спектр применения компьютеров в работе специалистов указанных областей.

Конечно, учебное пособие не включает всего объема знаний по программированию, однако материал пособия достаточно полно отражает все главные задачи дисциплины, демонстрирует основные способы их решения.

После изучения материала пособия студент получит практические навыки компьютерной обработки информации, необходимые для решения серьезных прикладных задач.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Воронова, Л. М.* Типовые алгоритмические структуры для вычислений : учеб. пособие / Л. М. Воронова ; Владим. гос. ун-т. – 2-е изд., перераб. и доп. – Владимир : Ред.-издат. комплекс ВлГУ, 2004. – 96 с. – ISBN 5-89368-503-2.

2. Алгоритмы: построение и анализ / Т. Х. Кормен [и др.]. – М. : Вильямс, 2016. – 1328 с. – ISBN 978-5-8459-2016-4.

3. *Прата, С.* Язык программирования C++. Лекции и упражнения : пер. с англ. / С. Прата. – М. : Вильямс, 2017. – 1248 с. – ISBN 978-5-8459-2048-5.

4. *Страуструп, Б.* Программирование. Принципы и практика с использованием C++ / Б. Страуструп. – М. : Вильямс, 2016. – 1328 с. – ISBN 978-5-8459-1949-6.

5. *Шилдт, Г.* Полный справочник по C++ : пер. с англ. / Г. Шилдт. – 4-е изд., стер. – М. : Вильямс, 2015. – 800 с. – ISBN 978-5-8459-2047-8.

ПРИЛОЖЕНИЯ

Приложение 1

Образец титульного листа

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Кафедра ФиПМ

ЛАБОРАТОРНАЯ РАБОТА № 2
по дисциплине «Основы программирования»
на тему «Реализация разветвленных алгоритмов на языке C++»

Вариант 1

Выполнил:
ст. группы НТ-117
Иванов И. М.
Принял:
ст. преподаватель каф. ФиПМ
Павлова О. Н.

Владимир 2018

Приложение 2

Таблица ASCII-кодов

Символы с кодами от 0 до 127

0 -	16 - ▶	32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
1 - ☺	17 - ◀	33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
2 - ☻	18 - ↕	34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
3 - ♥	19 - ‼	35 - #	51- 3	67 - C	83 - S	99 - c	115 - s
4 - ♦	20 - ¶	36 - $	52 - 4	68 - D	84 - T	100 - d	116 - t
5 - ♣	21 - §	37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
6 - ♣	22 - ▬	38 -&	54 - 6	70 - F	86 - V	102 - f	118 - v
7 -	23 - ↕	39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
8 -	24 - ↑	40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
9 -	25 - ↓	41 -)	57 - 9	73 - I	89 - Y	105 - i	121 - y
10 -	26 - →	42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
11 -	27 - ←	43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
12 -	28 - ∟	44 - ,	60 - <	76 - L	92 - \	108 - l	124 - |
13 -	29 - ↔	45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
14 - ♫	30 - ▲	46 - .	62 ->	78 - N	94 - ^	110 - n	126 - ~
15 - ☼	31 - ▼	47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ⌂
16 - ▶	32 -	48 -0	64 - @	80 - P	96 -	112 - p	

Символы с кодами от 128 до 255

128 -А	144 - Р	160 - а	176 - ▒	192 - └	208 - ╨	224 -р	240 - Ё
129 - Б	145 - С	161 - б	177 - ▒	193 - ┴	209 - ╤	225 -с	241- ё
130 -В	146 - Т	162 - в	178 - ▓	194 - ┬	210 - ╥	226 -т	242 - Є
131 - Г	147 - У	163 - г	179 - |	195 - ├	211 - ╙	227 -у	243 - є
132 -Д	148 - Ф	164 - д	180 - ┤	196 - —	212 - ╘	228 -ф	244 - Ї
133 -Е	149 - Х	165 - е	181 - ╡	197 - ┼	213 - ╒	229 - х	245 - ї
134 - Ж	150 - Ц	166 - ж	182 - ╢	198 - ╞	214 - ╓	230 - ц	246 - Ў
135 -З	151 - Ч	167 - з	183 - ╖	199 - ╟	215 - ╫	231 - ч	247 - ў
136 -И	152 -Ш	168 - и	184 - ╕	200 -	216 - ╪	232 - ш	248 - °
137 - Й	153 -Щ	169 - й	185 - ╣	201 - ╔	217 - ┘	233 - щ	249 - ●
138- К	154 - Ъ	170 - к	186 - ║	202 - ╩	218 - ┌	234 - ъ	250 - .
139 -Л	155 - Ы	171 - л	187 - ╗	203 - ╦	219 - █	235 - ы	251 - √
140 -М	156 - Ь	172 - м	188 - ╝	204 - ╠	220 - ▄	236 - ь	252- №
141 -Н	157 - Э	173 - н	189 - ╜	205 - ═	221 - ▌	237 - э	253 - ¤
142 -О	158 - Ю	174 - о	190 - ╛	206 - ╬	222 - ▐	238 - ю	254 - ■
143- П	159 - Я	175 - п	191 - ┐	207 - ╧	223 - ▀	239 - я	255 -
144 - Р	160 - а	176 - ▒	192 - └	208 - ╨	224 - р	240 - Ё	

ОГЛАВЛЕНИЕ

Учебное издание

ПАВЛОВА Ольга Николаевна

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Лабораторный практикум

Редактор А. А. Амирсейидова
Технический редактор А. В. Родина
Корректор Н. В. Пустовойтова
Компьютерная верстка Е. А. Герасиной
Выпускающий редактор Е. В. Невская

Подписано в печать 14.11.18.
Формат 60×84/16. Усл. печ. л. 6,05. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.