МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский государственный
гуманитарно-экономический университет

Алгоритмизация и программирование

Учебно-методическое пособие
для студентов, обучающихся по направлениям подготовки «Прикладная математика и информатика», «Прикладная информатика» и «Информатика и вычислительная техника»

Москва
2018

УДК 004.4(075.8)
ББК 32.973.202-018.2 я73
П 31

Рецензенты:

Б.Г. Миронов, доктор физико-математических наук, заведующий кафедрой «Высшая математика и естественные науки» ВГБОУ ВО «Российский университет транспорта (МИИТ)»;

Т.В. Истомина, доктор технических наук, профессор кафедры «Прикладная математика и информатика по областям» ФГБОУИ ВО «Московский государственный гуманитарно-экономический университет»;

И.Г. Благовещенский, доктор технических наук, доцент кафедры АСУБП МГУПП

Е.В. Петрунина

П 31 Алгоритмизация и программирование: *учебно-методическое пособие* / О.Н. Савельева, Э.В. Байрамов, Д.К. Печерский. – М.: МГГЭУ, 2018. – 122 с.

В учебно-методическом пособии приведены теоретические основы программирования на языке С++, рассмотрены основные принципы создания алгоритмов и написания программ на С++. Описаны основные линейные, разветвляющиеся и циклические структуры алгоритмов решения вычислительных задач, даны примеры решения типовых задач, а также задания для выполнения практических работ по рассматриваемым темам.

Для студентов, обучающихся по направлениям подготовки «Прикладная математика и информатика», «Прикладная информатика» и «Информатика и вычислительная техника», изучающих дисциплины «Алгоритмизация и программирование», «Языки и методы программирования», «Информатика и программирование».

ISBN 978-5-9799-0116-9

Краткое оглавление

Полное оглавление

Введение

Учебно-методическое пособие содержит теоретические и практические основы программирования на языке С++. Язык С++ является языком программирования со сложной структурой и синтаксисом. Данное пособие предназначено для начинающих программистов в качестве основы.

Решение прикладных вычислительных задач представляет собой сложный процесс, который включает основные этапы:

- постановка задачи;
- построение математической модели;
- формализация задачи;
- составление алгоритма решения задачи;
- написание программы на языке программирования;
- отладка и тестирование программы;
- составление инструкции для использования программы.

Важными этапами являются разработка алгоритма, написание программы и ее отладка. В данном пособии приведены основные алгоритмы решения вычислительных задач, рассмотрены алгоритмы и программы разветвляющейся структуры, циклической структуры, функции, указатели, массивы чисел, строки, динамические массивы.

Структура пособия построена в соответствии с последовательностью практических и лабораторных работ по дисциплине «Алгоритмизация и программирование», «Языки и методы програм-

мирования». Цели и задачи соответствуют программам указанных дисциплин и способствуют формированию соответствующих компетенций основной образовательной программы.

В результате освоения дисциплины студент должен

знать:

- основные понятия, приемы и методы алгоритмизации;
- принципы построения алгоритмов;
- основные типы вычислительных процессов: линейные, разветвляющиеся, циклические;
- язык программирования C++;
- среду программирования Microsoft Visual C++;

уметь:

- осуществлять постановку задачи;
- разрабатывать алгоритмы и кодировать их на языке программирования;
- использовать современные информационные технологии
- использовать инструментальные средства для решения различных задач;

владеть навыками:

- разработки алгоритмов и кодирования приложений для решения профессиональных задач;
- тестирования и отладки приложений;
- использования для решения задач современного программного обеспечения.

1

Разработка алгоритмов решения задач

Алгоритм — некоторая конечная последовательность правил (предписаний), однозначно определяющая процесс преобразования исходных и промежуточных данных в конечный результат решения задач.

Алгоритм линейной структуры (линейный алгоритм) — алгоритм, в котором правила выполняются последовательно друг за другом (рис. 1). Такой порядок выполнения называется естественным.

Для решения любой нетривиальной задачи можно составить несколько алгоритмов, приводящих к получению результатов. Из всех возможных алгоритмов необходимо выбрать наилучший по выбранному критерию качества.

Часто используют либо оценку точности решения, либо затраты времени на решение, либо некоторый интегральный критерий, включающий оценку точности и затраты времени.

Алгоритмы разветвляющейся структуры. На практике редко удается представить схему алгоритма решения задачи в виде линейной структуры. Часто в зависимости от каких-либо значений промежуточных результатов необходимо организовать вычисление либо по одним, либо по другим формулам, т.е. в зависимости от выполнения некоторого логического условия вычислительный процесс должен идти по одной или по другой ветви.

Алгоритм такого вычислительного процесса называется алгоритмом разветвляющейся структуры (рис. 2).

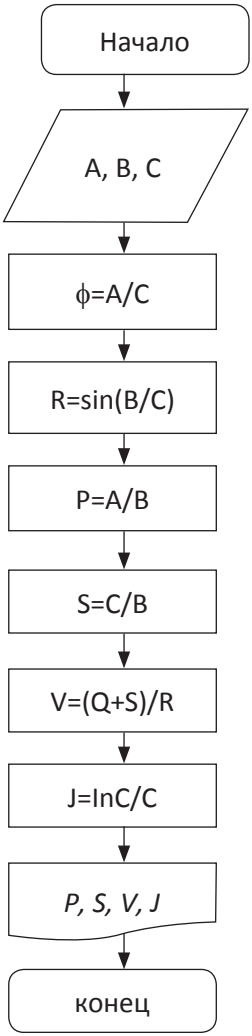

Рис. 1. Блок-схема линейного алгоритма

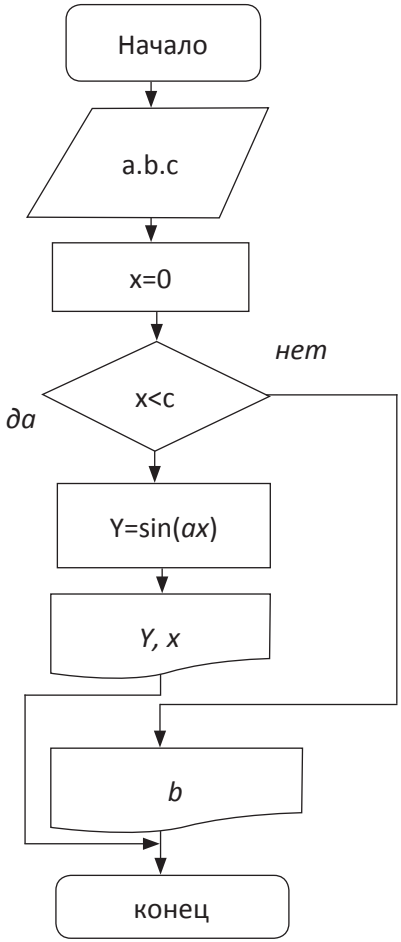

Рис. 2. Блок-схема алгоритма с одним разветвлением

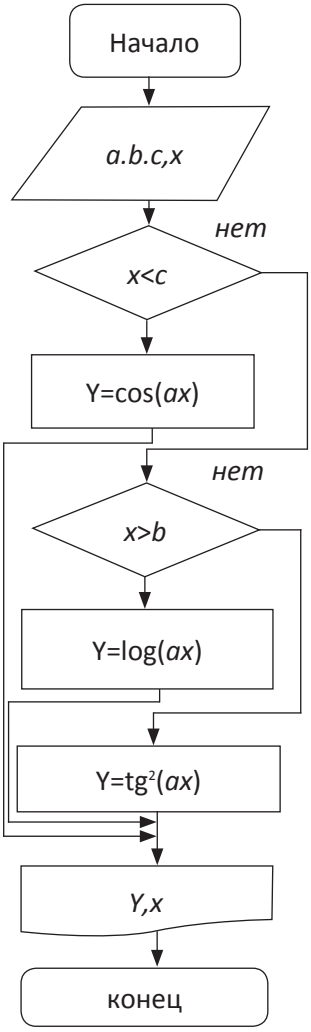

Рис. 3. Блок-схема алгоритма с двумя разветвлениями

В общем случае количество ветвей в таком алгоритме разветвляющейся структуры необязательно равно двум (рис. 3).

Алгоритмы циклической структуры. Часто при решении задач приходится многократно вычислять по одним и тем же математическим зависимостям при различных значениях входящих в них величин. Такие многократно повторяемые участки вычислительного процесса называются циклами. Использование циклов позволяет существенно сократить схему алгоритма и длину соответствующей ему программы.

Различают циклы с заданным и неизвестным числом повторений. К последним относятся итерационные циклы, характеризующиеся последовательным приближением к искомому значению с заданной точностью.

Примеры циклических алгоритмов решения простейших задач представлены на рис. 4.

Вычисления в цикле с несколькими одновременно изменяющимися параметрами. В рассмотренных ранее примерах алгоритмов циклической структуры в цикле изменялся только один параметр. На практике часто встречаются задачи, в которых необходимо использовать несколько параметров цикла, изменяющихся одновременно. Цикл с несколькими одновременно изменяющимися параметрами организуется по схеме, аналогичной схеме организации цикла

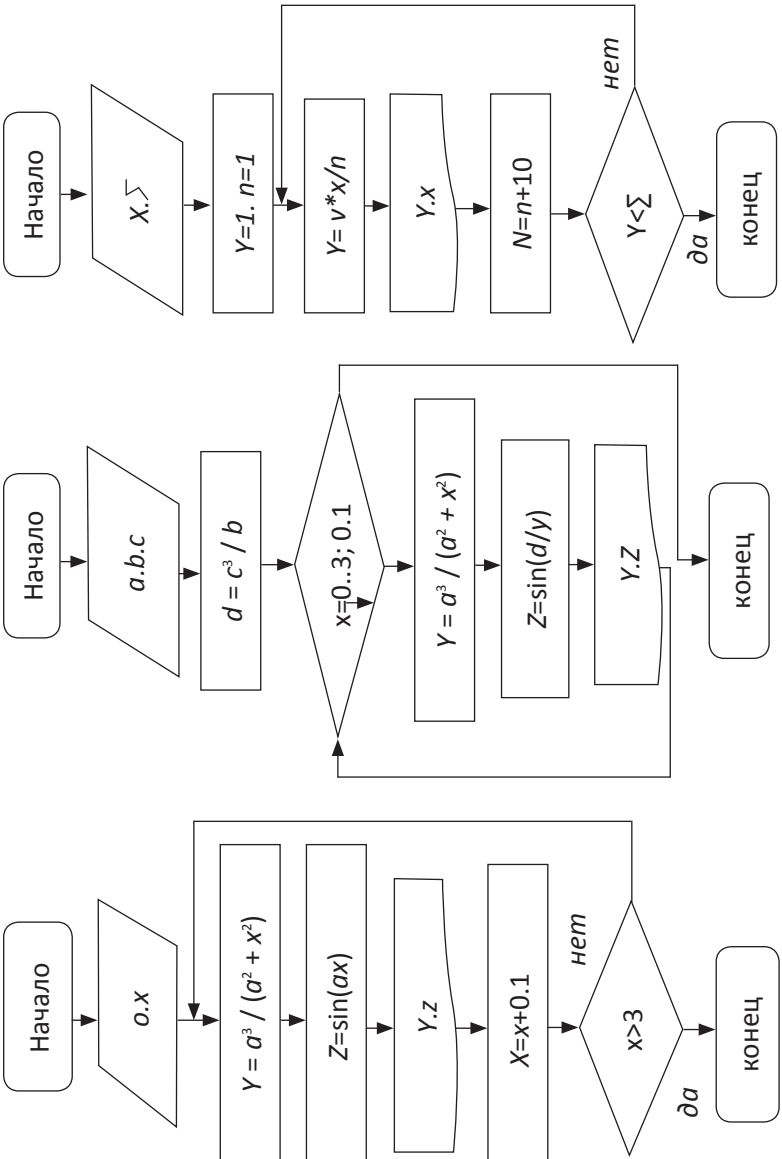

Рис. 4. Примеры блок-схем алгоритмов циклической структуры

с одним параметром. Для остальных параметров перед циклом необходимо задавать их начальные значения, а затем внутри его вычислять текущее.

Запоминание результатов. В рассмотренных выше задачах результатом вычислений было значение простой переменной, для записи которого в памяти ЭВМ выделяется одна ячейка. Если в процессе выполнения алгоритма значение переменной изменяется, то после окончания расчета в памяти ЭВМ останется лишь последнее значение результата. Чтобы записать все вычисленные значения, нужно выделить для их хранения необходимое количество ячеек памяти (массив), а текущий результат обозначить переменной с индексом.

Вычисление количества. Если необходимо вычислить количество каких-либо событий, например количество положительных или отрицательных значений функции $y = f(x)$, целесообразно организовать цикл, в котором надо предусмотреть накопление суммы путем прибавления единицы при наступлении какого-либо события (т.е. при выполнении условия). Формула для накопления количества (счетчик) будет иметь вид $N = N + 1$. Начальное значение $N = 0$.

Вычисление суммы и произведения. Если необходимо вычислить сумму значений некоторой функции $y = f(x)$ при различных значениях аргумента, целесообразно организовать цикл, в котором надо предусмотреть не только вычисление значений функции, но и накопление суммы путем прибавления полученных слагаемых к сумме всех предыдущих слагаемых. Формула, используемая для накопления суммы, имеет вид $z_n = z_{n-1} + y_n$. Поскольку нет надобности в запоминании значений всех промежуточных слагаемых, в качестве z и y нужно использовать простые переменные и накопление суммы вести в цикле по формуле $z := z + y$, где знак «:=» означает присваивание значения. Если начальное значение z предварительно приравнять к нулю, то после первого выполнения цикла значение z будет равно первому значению функции.

Аналогично накапливается и произведение с той лишь разницей, что для его накопления используется формула $z := z*y$, а начальное значение произведения должно быть равно единице.

Вычисление суммы членов бесконечного ряда. Задачи этого типа являются задачами, использующими итерационный цикл, так как заранее неизвестно, при каком члене ряда будет достигнута требуемая точность. Выход из цикла организуется по условию достижения требуемой точности. Для вычисления суммы членов ряда используется рассмотренный ранее прием накопления сумм.

Вычисление полинома. Для вычисления полинома *n*-й степени $y = a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}$ удобно использовать формулу Горнера $y = \left(\ldots\left(a_1x + a_2\right)x + a_3\right)x + \cdots + a_n\Big)x + a_{n+1}$. Если выражение, стоящее внутри скобок, обозначить y, то значение выражения в следующих скобках можно вычислить, используя рекуррентную формулу $y_{i+1} = y_ix + a_{i+1}$. Значение полинома у получается после повторения этого процесса в цикле n раз. Начальное значение y_1 целесообразно взять равным a_1, а цикл начинать с $i = 2$. Если обозначить у простой переменной, то схема примет вид, показанный на рис. 4. Все коэффициенты полинома и свободный член, как правило, сводятся в массив, состоящий из $n + 1$ элементов (n — порядок полинома). Если полином не содержит членов с некоторыми степенями x, то на соответствующем месте в массиве необходимо поместить коэффициент, равный 0.

Нахождение наибольшего и наименьшего значений. Нахождение наибольшего и наименьшего значения функции $y = f(x)$ выполняется в цикле, в котором вычисляется текущее значение функции и сравнивается с наибольшим или наименьшим из всех предыдущих значений этой функции. Если текущее значение функции окажется больше наибольшего из предыдущих значений, то его надо считать новым наибольшим значением. В противном случае наибольшее значение остается прежним. Сказанное можно описать условной математической формулой:

$$y_{\max} = \begin{Bmatrix} y_i, & \text{если } y_i > y_{\max}, \\ y_{\max}, & \text{если } y_i \le y_{\max}, \end{Bmatrix}$$

Аналогично для наименьшего значения (где $i = 1, 2, \ldots, n$):

$$y_{\min} = \begin{cases} y_i, & \text{если } y_i < y_{\min}, \\ y_{\min}, & \text{если } y_i \geq y_{\min}, \end{cases}$$

После первого выполнения цикла вычисляется y_1 и сравнивается с начальным значением y_{max} или y_{min}. После сравнения y_{max} или y_{min} принимают значение y_1. Тогда после вычисления y_2 будет находить наибольшее или наименьшее из этих первых двух значений функции. Необходимо в качестве начального значения y_{max} брать число порядка -10^{10}, чтобы наверняка выполнилось условие $y_1 > y_{max}$, а в качестве начального значения y_{min} — очень большое число, чтобы выполнилось условие $y_1 > y_{min}$.

Следует отметить, что здесь речь идет не о максимуме или минимуме функции, а о наибольшем или наименьшем из вычисленных значений функции. Это объясняется тем, что ЭВМ вычисляет дискретные значения функции и истинный максимум или минимум может находиться между ними.

Алгоритм со структурой вложенных циклов. Любой цикл, содержащий внутри себя один или несколько других циклов, называется внешним, а остальные циклы — внутренними.

Правила организации внешнего и внутреннего циклов — такие же, как и простого цикла. Параметры этих циклов изменяются не одновременно, т.е. при одном значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения.

2

Основы программирования на языке C++

Язык программирования C++ — достаточно сложный для изучения язык. Но он до сих пор держит лидирующие позиции по использованию в программных проектах, поскольку имеет огромное количество функций, обладает довольно высокой производительностью и постоянно улучшается [1, 2].

Для написания программ на языке C++ первоначально необходимо установить и настроить одну из интегрированных сред разработки — IDE. Установка и настройка среды разработки Netbeans приведена в [3], Microsoft Visual Studio в — [4], интегрированной среды разработки Qt — в [5]. Стандартным типом приложения будем считать консольное приложение (console application), а первоначальный код приложения должен соответствовать коду, представленному в листинге 1. Стандартным расширением файла для приложений на C++ является «.cpp».

Каждый из алгоритмических языков имеет свой набор символов, состоящий из букв, цифр и специальных символов. Наборы специальных символов включают в себя знаки операций, синтаксические знаки, специальные знаки и ключевые (служебные) слова [6, 7].

Ключевое слово — последовательность символов, имеющая особый, раз и навсегда установленный смысл в данном языке. В языках программирования ключевые слова используются в конструкциях только в определенных синтаксисом местах.

Имена, или идентификаторы, присваиваются элементам данных (константам и переменным), функциям и типам данных.

Комментарии служат для пояснения программы или отдельных ее частей. Наличие комментариев делает программу более понятной и удобной для чтения. При выполнении программы комментарии игнорируются и не влияют на решение задачи. Их можно свободно распределять по всей программе и вставлять в любое место, где допускается пробел. В коде программы могут присутствовать комментарии двух типов:

1) строчные — в начале строки комментария ставятся два символа «правый слеш» — «//»;
2) блочные — в начале блока комментариев ставятся символы «слэш» и «звездочка» — «/*», в конце блока комментариев ставится обратная последовательность — «*/».

Выражения — это слова, объединенные с помощью знаков операции. По своему типу выражения могут быть арифметическими, логическими, строковыми. Выражения предназначены для получения какого-то результата (промежуточного или окончательного).

Оператор — это минимальная структура (последовательности инструкций) в программе, производящая законченное действие. Оператор содержит ключевые слова, определяющие его смысл, а также числа, переменные и выражения, значения которых и определяют конкретное действие этого оператора.

Программа на языке C++ всегда начинает свою работу с функции «main» — данная функция в коде программы обязательна. Далее программа может содержать в себе функции, в которые выносятся логически законченные блоки кода для множественного использования либо для облегчения чтения программы (см. листинг 1).

Каждая строка программы, содержащая какие-либо законченные операции (присваивание, вызов функции и т.д.) должна оканчиваться точкой с запятой — «;». Если что-то на строке занимает большую длину и строку нужно перенести, в месте разрыва строки следует поставить левый слеш — «\».

Листинг 1. Первоначальный код программы

```
// Подключение библиотек
#include <cstdlib>

// Использование пространств имен
using namespace std;

// Основная функция, с которой начинается работа любой программы
int main() {
  // Код программы

  return 0;
}
```

2.1. Типы данных

Данные в C++ представлены двумя понятиями — литералы и переменные [8].

Литералы — данные, находящиеся в тексте программы в обычном виде и имеющие тип: например, цифры (1, 5, 123, 2e3 …), символы в одинарных кавычках (‘a’, ‘+’, ‘1’), строки в двойных кавычках (“мама”, “Hello World!”).

Переменные — данные, находящиеся в оперативной памяти компьютера, имеющие тип, имя и адрес в памяти.

Существуют простые и составные (пользовательские) типы данных. Простые типы данных доступны изначально, составные типы данных описываются в программе, либо подключаются.

Тип char служит для хранения отдельных символов и небольших целых чисел. Он занимает один машинный байт.

Типы short, int и long предназначены для представления целых чисел. Данные типы различаются только диапазоном значений, которые могут принимать числа, а конкретные размеры перечисленных типов зависят от реализации. Каждый из этих типов может быть знаковым (signed) и беззнаковым (unsigned). В знаковых

типах самый левый бит служит для хранения знака (0 — плюс, 1 — минус), а оставшиеся биты содержат значение. В беззнаковых типах все биты используются для значения. 8-битовый тип signed char может представлять значения от –128 до 127, а unsigned char — от 0 до 255.

Типы float, double и long double предназначены для чисел с плавающей точкой и различаются точностью представления (количеством значащих разрядов) и диапазоном.

Тип bool занимает 1 бит в памяти и содержит логическое значение (0 — ложь, 1 — истина).

2.1.1. Литералы

Когда в программе встречается некоторое число, например 1, то это число называется литералом, или литеральной константой.

Константа — это величина, значение которой не изменяется в программе.

Литерал является неадресуемой величиной: он хранится в памяти компьютера, но получить его адрес невозможно. Каждый литерал имеет определенный тип. Так, «0» имеет тип int, «3.14159» — тип double.

По умолчанию все целые литералы имеют тип signed int. Можно явно определить целый литерал как имеющий тип long, приписав в конце числа букву L. Буква U в конце определяет литерал как unsigned int, а две буквы — UL или LU — как тип unsigned long.

Литералы, представляющие действительные числа, могут быть представлены как с десятичной точкой, так и в научной (экспоненциальной) записи. По умолчанию они имеют тип double. Для явного указания типа float нужно использовать суффикс F, а для long double — L, но только в случае записи с десятичной точкой.

Слова true и false являются литералами типа bool.

Представимые литеральные символьные константы записываются как символы в одинарных кавычках.

Специальные символы (табуляция, возврат каретки) записываются как escape-последовательности. Escape-последовательность общего вида имеет форму «\ooo», где «ooo» — от одной до трех

восьмеричных цифр. Это число является кодом символа из таблицы ASCII. Пример escape-последовательности представлен в табл. 1.

Таблица 1

Escape-последовательности

Наименование	Escape-последовательность
новая строка	\n
горизонтальная табуляция	\t
забой	\b
вертикальная табуляция	\v
возврат каретки	\r
прогон листа	\f
звонок	\a
обратная косая черта	\\
вопрос	\?
одиночная кавычка	\'
двойная кавычка	\"

Строковый литерал — строка символов, заключенная в двойные кавычки. Такой литерал может занимать и несколько строк, в этом случае в конце строки ставится обратная косая черта. Специальные символы могут быть представлены своими escape-последовательностями.

Фактически строковый литерал представляет собой массив символьных констант, где по соглашению языков C и C++ последним элементом всегда является специальный символ с кодом 0 (\0). Литерал ‘A’ задает единственный символ A, а строковый литерал «A» — массив из двух элементов: ‘A’ и \0 (пустого символа).

2.1.2. Переменные

Переменная, или *объект* — это именованная область памяти, к которой существует доступ из программы; туда можно помещать значения и затем извлекать их. Каждая переменная C++ имеет опре-

деленный тип, который характеризует размер и расположение этой области памяти, диапазон значений, которые она может хранить, и набор операций, применимых к этой переменной.

Объявление переменной предполагает указание имени переменной (например, var) и ее типа (например, int):

```
int var1;
int var2.
```

Если при объявлении переменной одновременно выделяется память под нее, то происходит *определение переменной*. Пример объявления переменных представлен в листинге 2. Также существует понятие константы — переменная, объявленная с ключевым словом «const», которой при объявлении сразу же присваивается значение, которое не может быть изменено в процессе работы программы.

Листинг 2. Пример объявления переменных

```
// Целое число
int a = 2;
// Целочисленная константа
const int b = 5;
// Число с плавающей точкой одинарной точности
float tiny = 4.5;
// Число с плавающей точкой двойной точности
double more = 3e24;
// Символ
char symb = ‘и’;
// Логическое значение
bool flag = false.
```

Имя переменной, или идентификатор, может состоять из латинских букв, цифр и символа подчеркивания. Прописные и строчные буквы в именах различаются. Язык C++ не ограничивает длину

идентификатора, однако пользоваться слишком длинными именами типа «gosh_this_is_an_impossibly_name_to_type» неудобно.

Некоторые слова в C++ являются ключевыми и не могут быть использованы в качестве идентификаторов; в табл. 2 приведен их полный список.

Таблица 2

Ключевые слова языка C++

asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	export
extern	false	float	for	friend
goto	if	inline	int	long
mutable	namespace	new	operator	private
protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_cast
struct	switch	template	this	throw
true	try	typedef	typeid	typename
union	unsigned	using	virtual	void
volatile	wchar_t	while		

Для лучшей читабельности программы рекомендуется придерживаться общепринятых соглашений об именах объектов:

- имя переменной обычно пишется строчными буквами, например, index (для сравнения: Index — это имя типа, а INDEX — константа, определенная с помощью директивы препроцессора #define);
- идентификатор должен нести какой-либо смысл, поясняя назначение объекта в программе, например: birth_date или salary;
- если такое имя состоит из нескольких слов, как, например, birth_date, то принято либо разделять слова символом подчер-

кивания (birth_date), либо писать каждое следующее слово с большой буквы (birthDate).

2.2. Выражения и операции

Каждая строка рабочей программы содержит определенные выражения и/или операторы. Большинство операторов служит для управления последовательностью действий в программе — это операторы условий (п. 2.4), циклов (2.5), вызова функций (п. 4).

Выражение — строка присваивания какой-либо переменной какого-либо значения. Выражение может состоять из переменных, литералов и операций.

Существуют различные типы операций:

- операция присваивания;
- операция определения;
- арифметические операции;
- логические операции;
- символьные операции;
- бинарные операции.

Пример выражений представлен в листинге 3[1].

Листинг 3. Общий вид операции присваивания

```
// Выражение в общем виде:
// %переменная1% = %переменная2% %оператор% ↩
%переменная3%

// Арифметическое выражение
a = b + 5;
// Логическое выражение
Flag = true;
```

Все операции производятся по порядку, устанавливать порядок операций можно при помощи постановки круглых скобок.

[1] В некоторых листингах в конце строк может встречаться символ ↩. Он означает разрыв строки в книге. В коде же разрыва быть не должно.

2.2.1. Арифметические операции

Все арифметические операции стандартны. Порядок их выполнения в выражении представлен в табл. 3.

Таблица 3

Арифметические операции

Порядок	Символ	Название	Пример
1	* /	Умножение Деление	expr1 * expt2 expr1 / expt2
2	%	Деление по модулю	expr1 % expt2
3	+ –	Сложение Вычитание	expr1 + expt2 expr1 – expt2

Деление целых чисел дает в результате целое число. Дробная часть результата, если она есть, отбрасывается. Пример представлен в листинге 4.

Листинг 4. Результат деления дробных целых

```
int ivall = 21 / 6; // ival1 = 3
int ival2 = 21 / 7; // ival2 = 3
```

Операция остаток (%), называемая также делением по модулю, возвращает остаток от деления первого операнда на второй, но применяется только к операндам целого типа (char, short, int, long). Результат положителен, если оба операнда положительны. Если же один или оба операнда отрицательны, результат зависит от реализации. Пример представлен в листинге 5.

Листинг 5. Использование операции деления по модулю

```
3.14 % 3; // ошибка: операнд типа double
21 % 6;   // правильно: 3
```

```
21 % 7;   // правильно: 0
21 % –5;  // машинно-зависимо: –1 или 1
int iva1 = 1024;
double dval = 3.14159;
iva1 % 12;   // правильно: 4
iva1 % dval; // ошибка: операнд типа double
```

Иногда результат вычисления выражения может быть неправильным, либо неопределенным. Это может происходить как по математическим причинам (например, деление на 0), так и по машинным (переполнение, неправильное приведение данных).

Также при вычислениях с вещественными числами стоит учитывать ошибки округления — потеря значащих цифр после запятой из-за представления цифр в компьютере.

Существует понятие «машинный ноль» — такое близкое к арифметическому нулю число, после которого компьютер считает, что ноль достигнут. Значение машинного нуля для каждого типа данных и конкретной реализации может отличаться.

Помимо простых арифметических операций существуют еще сокращенные операции присваивания, представителями которых являются инкремент (увеличение значения переменной на 1) и декремент (уменьшение значения переменной на 1). Сокращенные операции позволяют экономить место в коде, а также выполняются процессором быстрее, чем обычные операции присвоения. Список сокращенных операций представлен в табл. 4.

Таблица 4

Сокращенные арифметические операции

Операция	Пример	Соответствие
++ (инкремент)	i++, ++i	i = i + 1
— (декремент)	i—, —i	i = i – 1
+=	i += 1	i = i + 1

Продолжение табл. 4

Операция	Пример	Соответствие
–=	i –= k	i = i – k
*=	i *= 5	i = i * 5

Так же, как и оператор %, оператор % = может использоваться только для переменных целого типа.

Операции инкремента и декремента имеют две формы:

- постфиксная (i++, i—), при использовании которой сначала производится чтение значения переменной, а уже потом ее изменение;
- префиксная (++I, —i), при использовании которой сначала производится изменение значения переменной, а уже после чтение.

Операции инкремента и декремента можно встраивать в операторы цикла и условия, и в зависимости от их формы результат выполнения программы будет разным.

2.2.2. Логические операции и операции сравнения

Все логические операции возвращают результат типа bool, то есть, либо «true», если выражение истинно, либо «false», если выражение ложно. Логические операторы следует применять только к значениям типа bool. Если логическое значение применяется в выражении с целыми числами, то true преобразуется в 1, а false в 0. Список логических операций и операций сравнения представлен в табл. 5.

Логическое И (&&) возвращает истину только тогда, когда истинны оба операнда.

Логическое ИЛИ (||) дает истину, если истинен хотя бы один из операндов.

Гарантируется, что операнды вычисляются слева направо и вычисление заканчивается, как только результирующее значение становится известно.

Таблица 5

Список логических операций и операций сравнения

Порядок	Символ	Название	Пример
1	!	Логическое НЕ	!expr
2	<	Меньше	expr1 < expt2
	<=	Меньше или равно	expr1 <= expt2
	>=	Больше или равно	expr1 >= expt2
	>	Больше	expr1 > expt2
	==	Равно	expr1 == expt2
	!=	Не равно	expr1 != expt2
3	&&	Логическое И	expr1 && expt2
4	||	Логическое ИЛИ	expr1 || expt2

2.3. Линейная программа. Подключение библиотек

Линейной называется программа, в которой все действия выполняются последовательно, то есть, в которой отсутствуют условия и циклы. Простейшая линейная программа — вывод в консоль фразы «Hello World».

Для написания многих программ на языке C++ недостаточно основных средств языка — требуется написание сложных алгоритмов или использование функций. Многие стандартные алгоритмы и функции уже определены в стандартной библиотеке C++, и для их использования достаточно подключить один из заголовочных файлов при помощи директивы препроцессора «include». Использование данной директивы сводится к двум случаям:

1) подключение частей стандартной библиотеки C++. В данном случае имя заголовочного файла пишется в угловых скобках <…>, а сам файл ищется в директории с компилятором;
2) подключение сторонних библиотек и заголовочных файлов. В данном случае имя заголовочного файла пишется в двойных кавычках, а сам файл ищется в директории с программой.

2.3.1. Потоковый ввод и вывод

Для написания программы вывода «Hello World» требуется один из заголовочных файлов стандартной библиотеки C++ — «iostream», который позволяет работать с потоками ввода и вывода. Пример кода программы вывода «Hello World» представлен в листинге 6.

Листинг 6. Программа «Hello World»

```
// Стандартная библиотека C++
#include <cstdlib>
// Библиотека потокового ввода и вывода
#include <iostream>

using namespace std;
int main() {

   cout << “Hello World!” << endl;
   cout << “Hello” << “ “ << “World!\n”;

   return 0;
}
```

В самом начале программы подключаются две стандартные библиотеки C++: cstdlib, которая отвечает за работу программы и стандартных операторов, и iostream, которая позволяет работать с потоками ввода и вывода.

Поток cout — поток вывода, в который при помощи оператора «<<» записываются данные, которые после выводятся в консоль. Записывать составные данные в поток можно несколькими способами. Как пример, в листинге 6 обе строки выведут в консоль одно и то же. Оператор «endl» записывает в конец строки с текстом символ «\n», который является переводом строки. Во второй строке этот символ прописан явно.

Поток cin — поток ввода, из которого при помощи оператора «>>» считываются данные и заносятся в переменные.

Простейшая консольная программа-калькулятор представлена в листинге 7. В листинге 8 представлен способ обмена значениями двух переменных.

Листинг 7. Код консольного калькулятора

```
// Стандартная библиотека C++
#include <cstdlib>
// Библиотека потокового ввода и вывода
#include <iostream>

using namespace std;

int main() {
   int a, b;
   double c, d;

   cout << “Hello user!” << endl;
   cout << “Please, write the number a: “;
   cin >> a;
   cout << “Please, write the number b: “;
   cin >> b;
   cout << “Please, write the number c and d: “;
   cin >> c >> d;
   cout << “You write: a = “ << a << “ b = “
        << b << “ c = “ << c << “ d = “ << d << endl;

   cout << “a + b = “ << a + b << endl;
   cout << “a / b = “ << a / b << endl;
   cout << “a % b = “ << a % b << endl;
   cout << “c – d = “ << c – d << endl;
   cout << “c / d = “ << c / d << endl;
```

```
    return 0;
}
```

Листинг 8. Код операции обмена значениями

```
// Стандартная библиотека C++
#include <cstdlib>
// Библиотека потокового ввода и вывода
#include <iostream>

using namespace std;

int main() {
    float a, b;

    cout << “Hello user!” << endl;
    cout << “Please, write the number a and b: “;
    cin >> a >> b;

    a = a + b;
    b = a – b;
    a = a – b;

    cout << “a = “ << a << “ b = “ << b << endl;

    return 0;
}
```

Оператор cin так же, как и оператор cout, может работать сразу с несколькими значениями. Если после оператора «>>» стоит одна переменная, то после ввода значения и нажатия в консоли клавиши «Enter» на клавиатуре программа продолжит свое выполнение, если же переменных несколько, то их значения следует разделить символом пробела либо нажатием клавиши Enter — программа не продолжит выполнение, пока все переменные не будут иметь какое-либо значение.

Таблица 6

Список математических функций

Функция	Описание
abs	Возвращает абсолютную величину целого числа
acos	арккосинус
asin	арксинус
atan	арктангенс
atan2	арктангенс с двумя параметрами
ceil	округление до ближайшего большего целого числа
cos	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
frexp	разбивает число с плавающей точкой на мантиссу и показатель степени.
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, т.е. x^y
sin	синус
sinh	гиперболический синус
sqrt	квадратный корень
tan	тангенс

Продолжение табл. 6

Функция	Описание
tanh	гиперболический тангенс
acosh	гиперболический ареакосинус
asinh	гиперболический ареасинус
atanh	гиперболический ареатангенс
cbrt	кубический корень
copysign(x,y)	возвращает величину, абсолютное значение которой равно x, но знак которой соответствует знаку y
erf	функция ошибок
erfc	дополнительная функция ошибок
exp2(x)	значение числа 2, возведенного в степень x
expm1(x)	значение функции $e^x - 1$
fdim(x,y)	вычисление положительной разницы между x и y
fma(x,y,z)	значение функции $(x * y) + z$
fmax(x,y)	наибольшее значение среди x и y
fmin(x,y)	наименьшее значение среди x и y
hypot(x,y)	гипотенуза, $\text{sqrt}(x^2 + y^2)$
ilogb	экспонента числа с плавающей точкой, конвертированная в int
lgamma	натуральный логарифм абсолютного значения гамма-функции
llround	округление до ближайшего целого в направлении от нуля (возвращает long long)
lround	округление до ближайшего целого в направлении от нуля (возвращает long)
log1p(x)	натуральный логарифм $1 + x$
log2	логарифм по основанию 2
logb	целочисленная часть логарифма x по основанию 2
nan(s)	возвращает нечисловое значение 'Not a Number'

Продолжение табл. 6

Функция	Описание
nearbyint	округление аргумента до целого значения в формате числа с плавающей точкой
nextafter(x,y)	следующее ближайшее представимое для *x* (по направлению к *y*)
nexttoward(x,y)	то же, что и nextafter, но *y* имеет тип long double
remainder(x,y)	вычисляет остаток от деления согласно стандарту IEC 60559
remquo(x,y,p)	то же, что и remainder, но сохраняет коэффициент по указателю *p* (как int)
rint	округление до целого (возвращает int) с вызовом ошибки inexact, если результат отличается от аргумента.
round	округление до целого (возвращает double)
scalbln(x,n)	x * FLT_RADIX^n (n is long)
scalbn(x,n)	x * FLT_RADIX^n (n is int)
tgamma	гамма-функция
trunc	отбрасывание дробной части

2.3.2. Математические операции и функции

Для работы с математическими вычислениями зачастую требуются сложные функции, которые труднореализуемы при помощи стандартных методов языка C++. Для решения этой проблемы в стандартной библиотеке C++ есть заголовочный файл «cmath», содержащий множество математических функций.

Список функций заголовочного файла «cmath» представлен в табл. 6, пример работы с математическими функциями представлен в листинге 9.

Листинг 9. Математические функции

```
#include <cstdlib>
#include <iostream>
// Подключение заголовочного файла с математическими ↩
функциями
#include <cmath>

using namespace std;

int main()
{
   // логарифм десятичный
   cout << “log10(10)     = “ << log10(10.0)   << endl;
   cout << “log10(1)      = “ << log10(1.0)    << endl;
   // натуральный логарифм (по основанию экспоненты) ↩
exp = 2.718281
   cout << “log(2.718281) = “ << log(2.718281) << endl;
   // корень квадратный
   cout << “sqrt(9)       = “ << sqrt(9.0)     << endl;
   // два в кубе
   cout << «pow(2,3)      = « << pow(2.0,3.0)  << endl;
   // модуль числа
   cout << «abs(0)        = « << abs(0.0)      << endl;
   cout << «abs(–5)       = « << abs(–5.0)     << endl;
  // округление 3.14 до наименьшего целого, но не меньше чем 3.14
   cout << “ceil(3.14)    = “ << ceil(3.14)    << endl;
  // округление –2.4 до наименьшего целого, но не меньше чем –2.4
   cout << “ceil(–2.4)    = “ << ceil(–2.4)    << endl;
   // округление 3.14 до наибольшего целого, но не больше чем 3.14
   cout << “floor(3.14)   = “ << floor(3.14)   << endl;
   // округление –2.4 до наибольшего целого, но не больше чем –2.4
   cout << “floor(–2.4)   = “ << floor(–2.4)   << endl;
   // остаток от деления 2.4/2
   cout << «fmod(2.4/2.0) = « << fmod(2.4,2.0) << endl;
```

```
  return 0;
}
```

2.4. Разветвляющаяся программа

Разветвляющейся называется программа, в которой в зависимости от условия может выполниться либо один кусок кода (ветвь условия), либо другой. Также возможен вариант, когда не выполнится ни одна ветвь, либо выполнится несколько (различные формы оператора switch).

2.4.1. Оператор IF

Основным условным оператором является оператор if. В общем виде он записывается следующим образом (листинг 10).

В операторе if, условие — всегда выражение типа bool, которое может быть либо истинным, либо ложным. Если выражение истинно, то выполнится код в ветке «да», если ложно, то код в ветке «нет». Примеры разветвляющейся программы с условным оператором if представлены в листингах 10, 11.

Листинг 10. Пример программы с разветвляющейся структурой

```
If (/*условие*/)
{
  // ветка «да»
} else {
  // ветка «нет»
}
```

Листинг 11. Пример разветвляющейся программы

```
if (2 < 5) {
  cout << “Yes, 2 < 5” << endl;
```

```
} else {
   cout << “No, 2 >= 5” << endl;
}
```

В данной программе всегда будет выводиться текст из первой ветки условия.

Каждый оператор условия может содержать внутри себя неограниченное количество вложенных условий, циклов, и прочих операторов. Условие выполнения может быть как простым, так и составным и содержать много частей, связанных между собою логическими операторами.

Также возможна укороченная и комбинированная запись условного оператора if, как это представлено в листинге 12. Подобная запись возможна за счет того, что язык C++ позволяет использовать блоки кода вместо операторов, которые находятся внутри фигурных скобок {}.

Листинг 12. Укороченная и комбинированная запись

```
// Укороченная запись
if (true) cout << “it’s true!” << endl;

// Комбинированная запись
if (2 > 5) {
   cout << “Yes, 2 > 5” << endl;
} else if (2 > 5 || 1 < 3){
   cout << “Yes, 2 > 5 or 1 < 3” << endl;
} else {
   cout << “This branch will not run”;
}
```

Существует встраиваемый аналог условного оператора, который называется тернарным оператором. Используется он для сокращения записи присваивания значения какой-либо переменной в зависимости от условия. В общем виде он записывается следующим образом:

```
/*переменная*/ = (/*условие*/) ? /*значение 1*/ : /*значение 2*/
```

В качестве первого и второго значения могут выступать целые блоки кода или функции. Главное условие — чтобы они так или иначе возвращали значение, которое будет присвоено переменной. Пример использования тернарного оператора показан в листинге 13.

Листинг 13. Тернарный оператор

```
i = (1 < 2) ? 4 : 3;
```

Расшифровывается данная запись следующим образом: если 1 меньше 2, то занести в переменную *i* значение 4, иначе занести в переменную *i* значение 3.

2.4.2. Оператор SWITCH

В языке C++ возможно написание кода, который будет выполнен в зависимости от значения какой-либо переменной, не создавая при этом множество операторов if. Для этого используется оператор выбора switch. В общем виде он записывается следующим образом:

```
switch (/*переменная или условие*/) {
   case /*значение 1*/:
      // ветка значения 1
   // ...
   default:
      /*ветка значения по умолчанию*/
}
```

Работа оператора switch заключается в поочередном сравнении значения из круглых скобок со значениями после ключевого слова case. Как только оба значения совпадут, будет выполнен код, написанный в соответствующей ветке. Если ни одно из заранее написанных значений не совпало со значением условия, тогда будет выполнен код из ветки default. Пример использования оператора switch представлен в листинге 14.

Листинг 14. Использование оператора выбора

```
int a = 1;
cout << “Please write the number of current month: “;
cin >> a;

switch (a) {
   case 1:
      cout << “Current month is January” << endl;
      break;
   case 2:
      cout << “Current month is Febrary” << endl;
      break;
   case 3:
      cout << “Current month is Marth” << endl;
   default:
      cout << “This is default branch” << endl;
}
```

В зависимости от введенного значения, будет выполнен тот или иной участок кода.

В первых двух ветках написан оператор прерывания break — как только программа доходит до его выполнения, последняя инструкция условия, выбора или цикла перестает работать и код продолжает выполнение после закрывающей фигурной скобки. В данном примере, если в переменную будет занесено значение «3», то выполнятся как код из ветки 3, так и код в следующей за ней ветке. Выполнение кода будет продолжаться либо до оператора break, либо до конца оператора выбора.

2.5. Программа с циклической структурой

Программой с циклической структурой называется программа, которая имеет блок операторов, выполняющийся более одного раза

подряд. Существует два вида циклов: циклы с известным числом повторений и циклы с неизвестным числом повторений.

2.5.1. Оператор FOR

Оператор for реализует арифметический цикл, в котором заранее возможно посчитать количество выполнений кода. Синтаксис оператора цикла for записывается следующим образом:

```
for (/*переменная-счетчик*/; /*условие цикла*/; /*изменение ↵
значения счетчика*/) {
   // выполняемый код
}
```

Переменная-счетчик нужна для подсчитывания количества совершённых итераций. Обычно ее объявляют и присваивают ей первоначальное значение тут же, в цикле, но можно использовать и объявленную заранее переменную. Условием цикла часто служит выражение сравнения типа «i <= 10». В качестве изменения значения переменной-счетчика может быть любое присваивание, но часто используют операцию инкремента или декремента. Пример арифметического цикла представлен в листинге 15.

Листинг 15. Арифметический цикл

```
for (int i = 0; i < 10; i++) {
   cout << i << endl;
}
```

В соответствии с заголовком цикла, его тело выполнится 10 раз, а в консоль будут выведены числа от 0 до 9.

Алгоритм работы арифметического цикла:

1. Объявление переменной I;
2. Проверка i < 10. Если истина, то п.3, иначе выход из цикла;

3. Изменение значения i = i + 1;
4. Выполнение тела цикла;
5. Переход к п. 2.

2.5.2. Операторы DO и WHILE

Операторы do и while реализуют два типа итерационных цикла — с предусловием и с постусловием. Разница в них только в том, когда производится проверка условия — до выполнения тела цикла или после. Часто подобное бывает полезно, так как если условие не выполняется, то цикл с постусловием гарантированно выполнится хотя бы один раз, а цикл с предусловием не выполнится ни разу. Пример итерационных циклов представлен в листинге 16.

Листинг 16. Пример итерационных циклов

```
int i = 0;

do {
   cout << i << endl;
   i += 2;
} while (i < 10);

while (i != 0){
   cout << – i << endl;
}
```

Код этой программы сначала последовательно выведет числа от 0 до 8 с шагом 2, а потом от 9 до 0 с шагом 1. Подобный порядок вывода обусловлен тем, что в первом цикле изменение значения переменной происходит после его вывода в консоль, а во втором цикле изменение происходит до чтения значения и его вывода в консоль (см. п. 2.2.1.)

Обязательным условием работы цикла является наличия выражения, которое изменяет значение условия, иначе цикл будет выполняться бесконечно.

Бесконечный цикл можно получить при помощи следующего кода:

```
do {
   // тело цикла
}
```

или:

```
while (true) {
   // тело цикла
}
```

Иногда это бывает необходимо, однако, зацикливание программы ведет к бесконечному увеличению требуемой для ее работы памяти и в итоге может привести либо к аварийному закрытию программы, либо к зависанию компьютера.

Для предотвращения подобной ситуации **внутри цикла следует всегда предусматривать условие выхода из цикла.** Оно может быть представлено в явном виде с помощью следующего кода:

```
while (true) {
   // тело цикла
   if (/*условие выхода из цикла*/) break;
}
```

Еще одним способом выхода из цикла является изменение параметров условия, установленного в блоке while.

Вопросы и задания для самоконтроля

1. Какие ограничения накладывает на переменную указанный для нее тип?
2. С каких символов могут начинаться имена переменных?
3. Поменяйте значения двух переменных, используя дополнительную переменную.
4. Для чего в коде приложения используются комментарии?
5. Какие формы имеют операции инкремента и декремента. Приведите примеры.
6. Чем отличаются операторы цикла с предусловием и с постусловием.
7. Назовите и опишите основные выражения, входящие в состав цикла for.

Варианты заданий

1. Найти действительные корни биквадратного уравнения. Выдать сообщения о возможных вариантах решения.
2. Решить систему двух линейных уравнений с двумя неизвестными. Предусмотреть выдачу сообщений: «Нет решения», «Бесконечное множество решений», «Одно решение», «Прямые параллельны или перпендикулярны».
3. Найти координаты точки пересечения прямой $y = kx + b$ и окружности заданного радиуса r с центром в начале координат. Переменные k, b, r вводятся с клавиатуры. Определить координатную четверть, в которой находятся точки пересечения.
4. На плоскости заданы координаты трех точек и последовательно соединяются между собой. Определить тип получаемого треугольника: прямоугольный, остроугольный, тупоугольный.
5. С клавиатуры вводятся четыре различных числа: a, b, c, d. Выбрать из них такие три числа, которые составляют стороны треугольника с наибольшей площадью.

6. Создайте цикл, который будет выводить на экран числа от 50 до 70.
7. При помощи цикла for создайте пирамиду из символов “X” на экране.

3

Указатели

Указатель (pointer) в C++ — обособленный тип данных. Переменные этого типа хранят не сами данные, а адрес области в оперативной памяти, где находятся данные указанного для переменной типа, причем, в самой переменной находится адрес начала выделенной области.

Указатели используются при выполнении следующих операций:

– доступ к элементам массива;
– передача аргументов в функцию, в которой необходимо изменить эти переменные;
– передача в функции массивов или строковых переменных;
– выделение памяти;
– создание сложных структур.

Указатели являются важной возможностью языка C++. Идея указателя относится к определению адреса ячейки. При загрузке программы в память она занимает определенное место, в соответствии с которым каждая переменная также имеет конкретный адрес. Получить адрес переменной можно с использованием *оператора получения адреса* &.

Оператор получения адреса

Реальные адреса, занятые переменными в программе, зависят от различных факторов: размер оперативной памяти, компьютер на

котором работает программа, наличие других программ в памяти и т.д.

Адрес переменной — это не то же самое, что ее *значение*.

Для работы с указателями предусмотрены два вида унарных операторов, позволяющих получить доступ к данным, на которые они ссылаются, — оператор разыменования и переменные-указатели.

Переменная-указатель — переменная, хранящая в себе значение адреса (или просто *указатель*).

Использование переменных-указателей необходимо для увеличения возможностей программирования, увеличения адресного пространства.

Для объявления переменной-указателя после наименования типа ставится символ «*». Звездочка означает *«указатель на»*. Синтаксис объявления указателя выглядит следующим образом:

```
/*тип данных*/ * /*имя переменной*/;
```

Указатели используются для прямой работы с оперативной памятью. Обычные переменные и объекты располагаются в статической области памяти — стеке — и занимают там определенный объем, который после компиляции программы изменить уже невозможно. Данные, на которые ссылаются указатели, располагаются в динамической области памяти — куче, и в любой момент могут занять больший объем, чем был указан до компиляции.

Подобные свойства указателей полезны для организации динамических массивов, передаче данных в функции без копирования самих данных, а также для построения динамических приложений, так как указатель может ссылаться не только на переменную, но и на функцию, класс, объект [9].

Пример работы с указателями представлен в листинге 17.

Листинг 17. Работа с указателями

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;

int main() {

   int ch = 5; // Обычная переменная типа int
   int* p_ch = &ch; // Указатель, проинициализированный ↵
адресом переменной ch

   // Получение адреса переменной ch в памяти
   cout << &ch << endl;
   cout << p_ch << endl;
   // Получение адреса самого указателя
   cout << &p_ch << endl;
   // Получение значения, расположенного по адресу
   cout << *p_ch << endl;

   // Изменение значения, расположенного по адресу
   *p_ch = *p_ch + 1;
   cout << *p_ch << endl;

   // Перемещение указателя в памяти на размер типа int
   p_ch++;
   cout << p_ch << endl;  // Будет выведен новый адрес
   cout << *p_ch << endl; // Значение по этому адресу не определено

   return 0;
}
```

Каждая переменная в C++ состоит из двух частей.

1. Адрес расположения этой переменной — его можно получить, обратившись к переменной с помощью оператора &.
2. Значение переменной, то, что записано в ячейке с адресом — для его получения достаточно указать имя переменной.

Каждый указатель состоит из трех частей.

1. Адрес самого указателя — его можно получить, используя оператор & с именем указателя.
2. Значение указателя — адрес ячейки памяти, где расположено значение заданного типа – его можно получить, написав только имя указателя.
3. Данные — их можно получить, используя оператор разыменования указателя * перед его именем.

Указатели обязательно должны иметь значения, т.е. они должны быть инициализированы определенным адресом до начала использования.

Доступ к ячейке, на которую указывает указатель, осуществляется при помощи операции *разыменования* — получения значения переменной, на которую указывает указатель.

Любые операции с *неразыменованным* указателем ведут к изменению адреса, на который он указывает. Так, в листинге 17 к *неразыменованному* указателю была добавлена 1, что привело к смещению указателя на 1 байт — стандартный размер переменной типа int.

Так же существуют двойные указатели и операция двойного *разыменования* — «**». Значением двойного указателя будет адрес в памяти, в котором также находится адрес в памяти. Подобную цепочку «указателей на указатели» можно продолжать бесконечно, но это сильно усложнит чтение кода.

Вопросы и задания для самоконтроля

1. Из каких частей состоит переменная в C++?
2. Приведите основные части указателя в C++.
3. Приведите примеры задач, в которых используются указатели.
4. Приведите примеры основных операций с указателями.
5. Указатель переменной pointer указывает на переменную var. Напишите выражение, которое позволит получить значение переменной var, не используя при этом ее имя.
6. Напишите определение для переменной указателя на float.

4

Массивы и строки

Часто в программе требуется хранить и обрабатывать довольно много данных, но прописывать десятки и сотни переменных невыгодно и долго. Для решения подобных задач в языке C++ существуют массивы.

Массив — набор однотипных данных, располагающихся рядом в памяти.

Простые массивы в C++ являются статическими — при объявлении массива сразу надо задать его объем, либо проинициализировать его элементы какими-либо значениями. Пока массив не проинициализирован, значения его элементов не определены — там может находиться что угодно, бывшее в ячейке памяти.

Размер массива всегда должен задаваться константой — будь то литерал, или константная переменная. Если попытаться задать размер массива не константной переменной, то компилятор выдаст ошибку.

Нумерация элементов массива всегда начинается с 0; так, если задан массив размера 5, первый его элемент будет иметь индекс 0, а последний — индекс 4.

Массив объявляется тем же способом, что и переменные, но после имени массива следует указать число его элементов в квадратных скобках — «[]».

Синтаксис определения массива:

```
/*тип данных массива*/ /*имя массива*/ [размер]
```

Пример объявления массива и заполнения его данными представлен в листинге 18.

Листинг 18. Пример объявления и заполнения массива

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {

   const int n = 5;
   int mass[n];

   for (int i = 0; i < n; i++){
      mass[i] = i * i;
      cout << mass[i] << endl;
   }

   return 0;
}
```

Для массивов в C++ не определены никакие операции. Узнать размер массива невозможно, так что для этого следует выносить его в отдельную константную переменную. Также невозможно скопировать один массив в другой, для этого следует поочередно скопировать все его элементы в цикле.

Пример копирования массива представлен в листинге 19.

Листинг 19. Пример копирования массива

```
const int n = 5;
int mass1[n], mass2[n + 2];

for (int i = 0; i < n; i++){
   mass1[i] = i * i;
}

for (int i = 0; i < n + 2; i++){
   mass2[i] = mass1[i];
   cout << mass2[i] << endl;
}
```

Массив можно явно проинициализировать при объявлении, записав значения его элементов в фигурных скобках, как это показано в листинге 20. Если массив инициализируется явно, то его размер можно не указывать, он будет равен количеству элементов в фигурных скобках. Если задать размер массива большим числом, чем явно проинициализированные элементы, то оставшиеся элементы массива будут проинициализированы значениями по умолчанию для данного типа данных.

Листинг 20. Явная инициализация массива

```
const int n = 5;
int mass[n] = {3, 2, 1};

for (int i = 0; i < n; i++){
   cout << mass[i] << endl; // Выведет: 3 2 1 0 0
}
```

Язык C++ не осуществляет контроля выхода индекса за рамки размера массива. Если создать массив из 5 элементов, и запросить шестой, то это не вызовет ошибки, а программа определит

содержимое ячейки памяти, следующей за последним элементом массива.

4.1. Многомерные массивы

Для работы с данными в табличном (матричном) виде C++ позволяет использовать многомерные массивы. При объявлении многомерного массива каждая его размерность пишется в квадратных скобках после его имени, как это показано в листинге 21.

Листинг 21. Пример работы с многомерным массивом

```
int mass[4][3];

  for (int i = 0; i < 4; i++){
    for (int j = 0; j < 3; j++){
      mass[i][j] = i + j;
      cout << mass[i][j] << " ";
    }
    cout << endl;
  }
```

Данный код выведет в консоль матрицу из четырех строк и трех столбцов.

Количество измерений у массива может быть неограниченным.

Многомерный массив, как и одномерный, можно явно проинициализировать. Группы значений можно отделять фигурными скобками. Пример явного задания многомерного массива и особенность подобного задания представлены в листинге 22.

Листинг 22. Пример явного задания многомерного массива

```
int mass1[3][2] = {{1,1}, {2,2}};
int mass2[3][2] = {1,2};
int mass3[3][2] = {{1},{2}};
```

Во всех случаях задается массив из трех строк и двух столбцов. В первом случае в первой строке будут единицы, во второй строке двойки, в третьей нули. Во втором случае единица будет в первом элементе первой строки, а двойка — во втором элементе первой строки. В третьем случае единица будет в первом элементе первой строки, а двойка — в первом элементе второй строки.

4.2. Массив и указатель

В языке C++ имя объявленного массива является указателем на первый элемент этого массива. Совершая с ним действия, возможные для указателей, можно получить упрощение работы с массивами, а также возможность создания динамического массива.

Пример работы с массивом как с указателем представлен в листинге 23.

Листинг 23. Проход по массиву при помощи указателей

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {

   int m_size = 5;
   int mass[m_size] = {1, 2, 3, 4, 5};
   // Присваиваем указателю адрес первой ячейки массива
   int *m_begin = mass;
   // Присваиваем указателю адрес последней ячейки массива
   int *m_end = mass + m_size; // Операция «+» двигает указатель

   cout << *(m_begin + 2) << endl; // Вывод 3-го элемента массива

   // Пока первый указатель не будет указывать на ячейку
```

```
    // за последним элементом массива
    while (m_begin != m_end) {
        cout << *m_begin << endl;
        ++m_begin;
    }

    return 0;
}
```

4.3. Динамический массив

Использование статических массивов плохо тем, что перед их созданием всегда надо обязательно знать, сколько элементов они будут содержать. Уйти от этого ограничения можно, размещая массив в динамической памяти. В языке C++ операции new и delete предназначены для динамического распределения памяти компьютера. Операция new выделяет память из области свободной памяти, а операция delete высвобождает выделенную память. Выделяемая память после ее использования должна высвобождаться, поэтому операции new и delete используются парами. Даже если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы.

Пример создания динамического массива представлен в листинге 24.

Листинг 24. Динамический массив

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {

    // Выделение динамической памяти под переменную типа int
```

```
   // и инициализация этой переменной числом 5
   int *count = new int (5);
   // Выделение динамической памяти под массив типа int из ↩
5 элементов
   // и инициализация его ранее заданным числом 5
   int *mass = new int[*count];

   for (int i = 0; i < *count; i++){
      // Заполнение массива данными методом смещения указателя
      *(mass + i) = i * i;
      // Вывод адреса ячейки памяти и занесенного в нее значения
      cout << (mass + i) << « >> « << *(mass + i) << endl;
   }

   // Высвобождение памяти из-под переменной и массива
   delete count;
   delete [] mass;

   return 0;
}
```

Многомерные динамические массивы создаются схожим образом, но несколько сложнее — для этого используется указатель второго порядка. Пример создания многомерного динамического массива представлен в листинге 25.

Листинг 25. Пример создания многомерного динамического массива

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {
```

```
   int row = 2; // Число строк
   int col = 4; // Число столбцов
   // Выделение памяти под два указателя (строки матрицы)
   int **mass = new int * [row];

   for (int i = 0; i < row; i++){
      // Запись в значение указателей адресов выделенной ↵
памяти под столбцы
      mass[i] = new int[col];
   }

   for (int r = 0; r < row; r++){
      for (int c = 0; c < col; c++){
       // Заполнение массива данными методом смещения указателя
         mass[r][c] = r + c;
         // Вывод адреса ячейки памяти и занесенного в нее значения
         cout << mass[r][c] << « «;
      }
      cout << endl;
   }

   // Высвобождение памяти
   for (int i = 0; i < row; i++)
      delete [] mass[i];

   return 0;
}
```

Создание массива с более чем двумя размерностями производится по той же схеме, за исключением вложенности указателей. Пример трехмерного массива приведен в листинге 26.

Листинг 26. Создание трехмерного массива

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {

   int d1 = 2, d2 = 3, d3 = 2;
   // Объявление трехмерного массива
   int ***mass;

   // Выделение памяти
   mass = new int ** [d1];
   for (int i = 0; i < d1; i++){
      mass[i] = new int * [d2];
      for (int j = 0; j < d2; j++){
         mass[i][j] = new int [d3];
      }
   }

   // Заполнение данными
   for (int r = 0; r < d1; r++){
      for (int c = 0; c < d2; c++){
         for (int d = 0; d < d3; d++){
            mass[r][c][d] = r + c + d;
            cout << mass[r][c][d] << endl;
         }
      }
   }
      // Высвобождение памяти
   for (int i = 0; i < d1; i++) {
      for (int j = 0; j < d2; j++) {
         delete [] mass[i][j];
```

```
    }
    delete [] mass[i];
  }
  delete [] mass;
    return 0;
}
```

Существуют три отличия динамических многомерных массивов от статических.

Общее количество памяти, выделяемой под динамический массив, больше. Если под статический массив размерности [2][3][4] выделяется 2 * 3 * 4 = 24 ячеек памяти, то под динамический массив той же размерности выделяется 24 ячейки памяти под заданный тип данных, а также 2 + 2 * 3 = 8 ячеек памяти под указатели.

Память, выделенная под динамический массив, не является непрерывной, так что использовать метод обхода массива из п. 4.2. нельзя.

Передача динамических массивов как аргумент функции будет отличаться от передачи статических массивов. Подробнее в п. 5.4.

4.4. Массив типа CHAR. Тип данных STRING

Любые строки в C++ являются массивами значений типа char. Строка символов, записанная в двойных кавычках — массив из n + 1 элементов, в котором каждому элементу до предпоследнего соответствует символ из строки, а последнему — нулевой символ «/0».

Если массив типа char инициализируется в строке объявления, то указывать его размер не обязательно.

Для доступа к строковому массиву используется константный указатель типа char. Со строкой также возможно работать, как с массивом.

В стандартной библиотеке C++ присутствует заголовочный файл «cstring», позволяющий более удобно работать со строками, а также включающий в себя несколько полезных функций. Список функций для работы со строками представлен в табл. 7.

Таблица 7

Функции для работы со строками

Функция	Описание
strlen(имя_строки)	Определяет длину указанной строки, без учета ноль-символа
Копирование строк	
strcpy(s1, s2)	Выполняет побайтное копирование символов из строки s2 в строку s1
strncpy(s1, s2, n)	Выполняет побайтное копирование n символов из строки s2 в строку s1 возвращает значения s1
Конкатенация строк	
strcat(s1, s2)	Объединяет строку s2 со строкой s1. Результат сохраняется в s1
strncat(s1, s2, n)	Объединяет n символов строки s2 со строкой s1. Результат сохраняется в s1
Сравнение строк	
strcmp(s1, s2)	Сравнивает строку s1 со строкой s2 и возвращает результат типа int: 0 — если строки эквивалентны, >0 — если s1<s2, <0 — если s1>s2 (с учетом регистра)
strncmp(s1, s2, n)	Сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 — если строки эквивалентны, >0 — если s1<s2, <0 — если s1>s2 (с учетом регистра)
stricmp(s1,s2)	Сравнивает строку s1 со строкой s2 и возвращает результат типа int: 0 — если строки эквивалентны, >0 — если s1<s2, <0 — если s1>s2 (без учета регистра)
	Сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 — если

Продолжение табл. 7

Функция	Описание
strnicmp(s1,s2,n)	строки эквивалентны, >0 — если s1<s2, <0 — если s1>s2 (без учета регистра)
Обработка символов	
isalnum(c)	Возвращает значение true, если с является буквой или цифрой, и false — в других случаях
isalpha(c)	Возвращает значение true, если с является буквой, и false — в других случаях
isdigit(c)	Возвращает значение true, если с является цифрой, и false — в других случаях
islower(c)	Возвращает значение true, если с является буквой нижнего регистра, и false — в других случаях
isupper(c)	Возвращает значение true, если с является буквой верхнего регистра, и false — в других случаях
isspace(c)	Возвращает значение true, если с является пробелом, и false — в других случаях
toupper(c)	Если символ с является символом нижнего регистра, то функция возвращает преобразованный символ с в верхнем регистре, иначе символ возвращается без изменений
Функции поиска	
strchr(s, c)	Поиск первого вхождения символа с в строке s. В случае удачного поиска возвращает указатель на место первого вхождения символа с. Если символ не найден, то возвращается ноль
strcspn(s1, s2)	Определяет длину начального сегмента строки s1, содержащего те символы, которые не входят в строку s2

Продолжение табл. 7

Функция	Описание
strspn(s1, s2)	Возвращает длину начального сегмента строки s1, содержащего только те символы, которые входят в строку s2
strprbk(s1,s2)	Возвращает указатель первого вхождения любого символа строки s2 в строке s1
Функции преобразования	
atof(s1)	Преобразует строку s1 в тип double
atoi(s1)	Преобразует строку s1 в тип int
atol(s1)	Преобразует строку s1 в тип long int
Функции стандартной библиотеки ввода/вывода <stdio>	
getchar(c)	Считывает символ с со стандартного потока ввода, Возвращает символ в формате int
gets(s)	Считывает поток символов со стандартного устройства ввода в строку s до тех пор, пока не будет нажата клавиша ENTER

Примеры работы со строками представлены в листингах 27–30.

Листинг 27. Объявление строк

```
#include <cstdlib>
#include <iostream>
// Подключение заголовочного файла для работы со строками
#include <cstring>

using namespace std;

int main() {

  // Объявление строки как массива
  // количество элементов на 1 больше, чем символов
```

```
   char str1[13] = «Hello user!»;
   // Объявление строки как константного указателя
   const char *str2 = «I am program writed by C++»;
   // Объявление строки встроенным типом данных string
   string str3 = «And I love you»;

   cout << str1 << endl;
   cout << str2 << endl;
   cout << str3 << endl;

   // Указывать размер массива не обязательно
   char out1[] = «Please, write the string: «;
   cout << out1;
   char in1[500]; // Символьный массив для ввода
   gets(in1);     // Функция gets() считывает введенные символы ↵
до Enter
   string out2 = «You write string with lenght «;
   // Функция strlen() возвращает размер строки
   cout << out2 << strlen(in1) << « : « << in1 << endl;

   return 0;
}
```

Листинг 28. Копирование строк

```
#include <cstdlib>
#include <iostream>
// Подключение заголовочного файла для работы со строками
#include <cstring>

using namespace std;

int main() {

   // Инициализация строки s2
```

```
   char s2[27] = «Counter–Strike 1.6 forever»;
   // Резервируем строку для функции strcpy()
   char s1[27];
   // Содержимое строки s2 скопировалось в строку s1,
   // указатель возвращается на s1
   cout << «strcpy(s1,s2) = « << strcpy(s1,s2) << endl;
   // Вывод содержимого строки s1
   cout << «s1=          « << s1          << endl;
   // Резервируем строку для следующей функции
   char s3[7];
   // Копируем 7 символов из строки s2 в строку s3
   cout << strncpy(s3, s2, 7) << endl;

   return 0;
}
```

Листинг 29. Конкатенация строк

```
#include <cstdlib>
#include <iostream>
// Подключение заголовочного файла для работы со строками
#include <cstring>

using namespace std;

int main() {

   char s1[30] = “I am “;
   char s2[] = “programmer on the C++!!!!”;
   // объединяем строки s1 и s2, результат сохраняется в строке s1
   cout << strcat(s1,s2) << endl;
   char s3[23] = «I am a good «;
   // объединяем 10 символов строки s2 со строкой s3
   cout << strncat(s3,s2,10) <<  «!!!» << endl;
```

```
   return 0;
}
```

Листинг 30. Обработка символов

```
#include <cstdlib>
#include <iostream>
// Подключение заголовочного файла для работы со строками
#include <cstring>

using namespace std;

int main() {

   char symbol    = ‘d’; // Буква
   char digit     = ‘9’; // Цифра
   char space     = ‘ ‘; // Пробел
   char character = ‘#’; // Знак
   // Функция isalnum() проверяет, является ли ее аргумент ⮐
буквой или цифрой
   cout << symbol    << « – it is digit or alpha?: «; isalnum(symbol)    ⮐
? cout << «true\n»: cout << «false\n»;
   // Функция isalpha() проверяет, является ли ее аргумент буквой
   cout << symbol    << « – it is alpha?:        «; isalpha(symbol)    ? cout ⮐
<< «true\n»: cout << «false\n»;
   // функция isdigit() проверяет, является ли её аргумент цифрой
   cout << digit     << « – it is digit?:        «; isdigit(digit)     ? cout ⮐
<< «true\n»: cout << «false\n»;
   // Функция isspace() проверяет, является ли её аргумент пробелом
   cout << space     << « – it is space?:        «; isspace(space)     ? cout ⮐
<< «true\n»: cout << «false\n»;
   // Функция islower() проверяет, является ли её аргумент ⮐
буквой нижнего регистра
   cout << symbol    << « – it is lower alpha?:   «; islower(symbol)    ? cout
<< «true\n»: cout << «false\n»;
```

```
  // Функция isupper() проверяет, является ли её аргумент ↩
буквой верхнего регистра
  cout << symbol   << « – it is upper alpha?:   «; isupper(symbol) ↩
? cout << «true\n»: cout << «false\n»;

  return 0;
}
```

Вопросы и задания для самоконтроля

1. Напишите выражение, которое определяет одномерный массив целого типа, состоящий из 50 элементов.
2. Как записываются индексы при доступе к многомерному массиву?
3. Напишите выражение для доступа к 5-му элементу 3-го подмассива двумерного массива doublearray.
4. Запишите выражение, которое объявляет двумерный массив darray типа float и инициализирует первый его подмассив значениями 23, 34, 54; второй — значениями 56, 47, 18; третий — значениями 87, 67, 82.
5. Напишите выражения для доступа к переменной sample структуры, являющейся 15-м элементом массива list.
6. Напишите выражения для объявления массива manyfruits, содержащего 40 элементов типа fruit.

Варианты заданий

1. В двумерном массиве Darray размером n*n определить местоположение максимального элемента, расположенного на главной диагонали.
2. В одномерном массиве Barray определить максимальный элемент среди четных элементов массива.
3. В двумерном массиве $A(m,n)$ в каждой строке найти минимальный элемент. Сформировать одномерный массив $b_1,b_2,...,b_m$ из этих элементов.

4. В одномерном массиве *A(n)* найти количество элементов, которые меньше среднего арифметического положительных элементов этого массива.

5

Функции

В больших программах, в которых используются сложные алгоритмы или алгоритмы, разрабатываемые целой командой программистов, часто невозможно вместить весь рабочий код в функцию main — какие-то участки кода требуется повторить много раз, какие-то участки кода вообще не смогут работать, если не сделать их самостоятельными. Для решения этих проблем, а также проблемы избыточности и нечитаемости кода, существуют функции [1,2].

Функция — обособленный самостоятельный участок кода, производящий какие-либо действия и/или возвращающий значение.

В общем случае любая функция выглядит следующим образом:

```
/*тип возвращаемого значения*/ /*имя*/ (/*список параметров*/)
{
   // Тело функции
}
```

Листинг 31. Функция и ее вызов

```
#include <cstdlib>
#include <iostream>

using namespace std;
```

```
// Функция, принимающая два аргумента типа int
// и возвращающая значение типа int
int sum(int a, int b) {
   return a + b;
}

// Функция, не возвращающая значение
void output(int str) {
   cout << str << endl;
}

int main() {

   // Объявление двух переменных типа int
   int p1 = 5, p2 = 7;
   // Вызов функции, суммирующей два числа и присвоение его ↵
переменной
   int s = sum(p1, p2);
   // Вызов функции, выводящей сумму в консоль
   output(s);

   return 0;
}
```

5.2. Прототип функции

Каждую функцию в C++, как и переменные, нужно объявлять, иначе компилятор не будет знать об их наличии [7,8]. Существует четыре возможных варианта объявления функции:

1) до функции main;
2) после функции main, с добавлением прототипа функции перед функцией main;
3) в отдельном файле .cpp;
4) в отдельных файлах .h и .cpp.

Первый случай рассмотрен ранее и является самым простым, но не самым удобным. Все функции должны быть написаны до основной функции main, иначе компилятор не будет их видеть.

Во втором случае функции пишутся после функции main, но при этом до нее требуется обязательно записать прототипы.

Прототип — функция без тела. В общем виде любой прототип функции выглядит следующим образом:

```
/*тип возвращаемого значения*/ /*имя*/ (/*список типов ↵
параметров*/)
```

В третьем случае функция вместе с телом выносится в отдельный файл .cpp, а в начале программы следует написать ее прототип, как представлено в листинге 32.

Листинг 32. Вынос функции в отдельный файл

```
// Файл func.cpp
#include <iostream>

using namespace std;

// Функция, принимающая два аргумента типа int
// и возвращающая значение типа int
int sum(int a, int b) {
   return a + b;
}

// Функция, не возвращающая значение
void output(int str) {
   cout << str << endl;
}
// Файл main.cpp
#include <cstdlib>
using namespace std;
```

```
// Прототипы функций, вынесенных в отдельный файл
int sum(int, int);
void output(int);

int main() {

   // Объявление двух переменных типа int
   int p1 = 5, p2 = 7;
   // Вызов функции, суммирующей два числа, и присвоение ↵
его переменной
   int s = sum(p1, p2);
   // Вызов функции, выводящей сумму в консоль
   output(s);

   return 0;
}
```

В четвертом случае функция пишется в отдельном файле .cpp, а прототип выносится в файл .h, как это показано в листинге 33.

Листинг 33. Вынос функции в отдельный файл и файл заголовка

```
// Файл func.h
// Обязательные директивы препроцессора
#ifndef FUNC_H
#define FUNC_H

// Прототипы функций
int sum(int, int);
void output(int);

// Обязательная директива препроцессора
#endif
// Файл func.cpp
```

```
#include <iostream>
// Подключение заголовочного файла
#include «func.h»

using namespace std;

// Функция, принимающая два аргумента типа int
// и возвращающая значение типа int
int sum(int a, int b) {
return a + b;
}

// Функция, не возвращающая значение
void output(int str) {
cout << str << endl;
}
// Файл main.cpp
#include <cstdlib>
// Подключение заголовочного файла
#include «func.h»

using namespace std;

int main() {

// Объявление двух переменных типа int
int p1 = 5, p2 = 7;
// Вызов функции, суммирующей два числа
int s = sum(p1, p2);
// Вызов функции, выводящей сумму в консоль
output(s);

return 0;
}
```

При подключении заголовочных файлов .h директива препроцессора include пишется с использованием двойных кавычек. Знаки «<» и «>» используются для подключения стандартных библиотек из директории компилятора.

5.3. Рекурсия

Для решения многих задач, использующих алгоритм с повторяющими участками удобно использовать функции и два основных подхода их использования: итерационный и рекуррентный.

Для итерационного подхода характерен множественный вызов одной и той же функции в цикле до достижения каких-либо результатов. Часто промежуточные результаты вычислений требуется хранить в отдельных переменных либо использовать указатели для передачи данных из итерации в итерацию.

Суть рекуррентного подхода заключается в том, что функция вызывает саму себя. При этом каждый экземпляр функции будет находиться в оперативной памяти до того момента, пока на какой-то итерации рекурсия не прекратится и последняя функция не вернет какое-либо значение, после чего это значение будет подниматься выше в вызвавшую функцию и так до самой верхней, которая и вернет итоговое значение.

Различия в написании итерационных и рекуррентных (рекурсивных) функций представлены в листинге 34 на примере вычисления факториала.

Листинг 34. Пример рекуррентной и итерационной функций

```
#include <cstdlib>
#include <iostream>

using namespace std;

int i_fact(int);
int r_fact(int);
```

```
int main() {
   int ch = 1;

   cout << “Please enter the number” << endl;
   cin >> ch;

   cout << “Iteration ans = “ << i_fact(ch) << endl;
   cout << “Recursion ans = “ << r_fact(ch) << endl;

   return 0;
}

// Итерационная функция
int i_fact(int ch){
   int f = 1;
   for (int i = 1; i <= ch; i++){
      f *= i;
   }
   return f;
}
// Рекурсивная функция
int r_fact(int ch){
   if (ch == 1){
      return 1;
   } else {
      return ch * r_fact(ch – 1);
   }
}
```

Рекуррентные функции работают быстрее, чем итерационные, но при этом потребляют намного больше оперативной памяти.

5.4. Передача массивов как аргументов

Передать массив элементов в функцию можно несколькими способами:

1) как массив без указания длины;
2) как указатель;
3) как адрес в памяти.

Все три способа показаны в листинге 35.

Листинг 35. Передача статического массива в функцию

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

void showText1 (char str[])//функция принимает строку как массив
{
   cout << str << endl;
}

void showText2 (char *str)//указатель *str будет указывать ⮐
на адрес первого символа в строке
{
   cout << str << endl;
}

void showText3 (char (&str)[150])// адрес строки из 150-ти символов
{
   cout << str << endl;
}

int main()
{
   // Установка параметра отображения кириллицы в консоли
   setlocale (LC_ALL, «rus»);
   cout << «Можно ввести строку в круглых скобках при вызове ⮐
```

```
функции: « << endl;

   showText1(«~~~ ~~~ ~~~ cppstudio.com ~~~ ~~~ ~~~»);
   cout << endl;

   char str1[] = «str1 — передаем как массив в функцию void ↵
showText1 (char str[]);»;
   showText1(str1);
   cout << endl;

   char str2[] = «str2 — передаем это значение в функцию @
void showText2 (char *str); используя указатель.»;
   showText2(str2);
   cout << endl;

   char str3[150] = «str3 — передаем это значение в функцию @
void showText3 (char &str[]);\nТут используем адрес строки.»;
   showText3(str3);
   cout << endl;

return 0;
}
```

Вопросы и задания для самоконтроля

1. Дайте определение прототипу функции.
2. Перечислите возможные варианты объявления функций.
3. Перечислите элементы программы, передача которых возможна в виде аргументов функции.
4. Напишите прототип функции Light(), которая возвращает значение типа char и принимает два аргумента. Первый аргумент имеет тип int, второй — float.
5. Приведите возможные варианты передачи массивов в качестве аргументов функции.
6. Приведите отличие локальных и глобальных переменных.

Варианты заданий

1. Напишите функцию, которая находит из трех переданных аргументов максимальный.
2. Напишите функцию, которая меняет местами значения своих аргументов типа int и передает их в основную программу.
3. Напишите вызывающую программу для предыдущего задания.
4. Напишите функцию, осуществляющую поиск максимального элемента массива.
5. Напишите функцию, которая при каждом обращении будет выводить целое число столько раз, сколько раз она вызывалась ранее из основной программы.
6. Напишите программу для задания 5, которая будет вызывать функцию не менее 10 раз.

6

Область видимости и время жизни переменных

У каждой статической переменной существует область видимости и время жизни.

Область видимости — участок в программе, для которого переменная существует [1,2]. В зависимости от места объявления, существуют две области видимости:

1) глобальная — переменная объявлена до начала функции main и доступна во всей программе;
2) локальная — переменная объявлена в любой функции и доступна только в этой функции.

Переменная, объявленная внутри функции, не доступна нигде, кроме этой функции, и называется локальной. Если в функции объявлена переменная, имеющая такое же имя, как ранее объявленная глобальная переменная, то при обращении к этому имени в функции будет доступна локальная переменная. Глобальная переменная изменена не будет.

После окончания выполнения функции все статические переменные уничтожаются. Динамические переменные остаются жить либо до момента высвобождения памяти оператором delete, либо до окончания выполнения программы.

Существует также распространенный вариант объявления константных значений — при помощи директивы препроцессора «#define». Многие считают, что таким образом они определяют глобаль-

ную константу, но это не верно. Директива препроцессора define указывает на то, что объявленная в ней переменная в ходе препроцессирования (процесс чтения кода до компиляции) будет заменена на следующее за ней значение. Исходя из этого, константа, объявленная в директиве define, будет занимать не одну ячейку памяти, как переменная с модификатором «const», а столько, сколько раз она используется в программе как литерал.

7

Примеры и полезные алгоритмы

Примеры далее написаны в универсальной нотации C++ — они работают без каких-либо сторонних подключаемых библиотек и изменения настроек. Но есть несколько изменений, которые касаются сред разработки (IDE).

- При работе в IDE Microsoft Visual Studio среда сама создает файл «stdafx.h», в котором прописываются все прочие заголовочные файлы для работы программы, а сам он подключается директивой препроцессора #include в файле «main.cpp». При копировании кода примеров в эту среду следует добавить строку «#include “stdafx”» в начало кода программы.
- При работе в IDE NetBeans следует заранее скачать компиляторы языков C и C++. Проще всего сделать это, используя программу Cygwin [5]. Из списка скачиваемых пакетов следует выбрать следующие:
  - gcc — компилятор для языка C;
  - g++ — компилятор для языка C++;
  - gdb — дебаггер;
  - make — программа для сборки в запускаемый файл.

Также при работе с русским языком в консоли следует перед первым выводом в программе прописывать строчку «setlocale (LC_ALL, «RUS»);», она устанавливает консоль в режим вывода кириллицы.

Все описанное выше справедливо для ОС семейства Windows. Для ОС семейства Linux подобного делать не надо — все компиляторы и настройки уже есть в системе.

7.1. Примеры линейных программ

Пример 1.

Запрограммировать следующее выражение:

$$\left(a+b-\frac{f}{a}\right)+f*a*a-(a+b)$$

Числа *a, b, f* вводятся с клавиатуры. Организовать пользовательский интерфейс таким образом, чтобы было понятно, в каком порядке должны вводиться числа.

Код решения представлен в листинге 36.

Листинг 36.

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

int main()
{
setlocale (LC_ALL, “RUS”);
int a, b, f, x;
cout<<”Введите цифры a, b и f: “<<endl;
cout<<”a=”;
cin>>a;
cout<<”b=”;
cin>>b;
cout<<”f=”;
cin>>f;
```

```
cout<<"Вычисляем по формуле: x=(a + b – f / a) + f * a * a – ↵
(a + b)»<<endl;
x=(a + b – f / a) + f * a * a – (a + b);
cout<<»x=»<<x<<endl;
system («pause>>void»);
return 0;
}
```

Пример 2.

Вывести в консоль домик из символов.

Код решения представлен в листинге 37.

Листинг 37.

```
#include <iostream>

using namespace std;

int main()
{
 cout << "\t   House:\n\n";
 cout << "\t    / \\\n";
 cout << "\t   /  \\\n";
 cout << "\t  /    \\\n";
 cout << "\t /———— \\\n";
 cout << "\t / |    |  \\\n";
 cout << "\t  |    | \n";
 cout << "\t  |    | \n";
 cout << "\t  |____| \n";

 return 0;
}
```

7.2. Примеры программ с разветвляющейся структурой

Пример 1.

Составить программу, которая будет считывать введённое пятизначное число, после чего каждую цифру этого числа необходимо вывести в новой строке.

Код решения представлен в листинге 38.

Листинг 38.

```
#include <iostream>
using namespace std;
int main()
{
   setlocale (LC_ALL, “RUS”);
   cout<<”\t\t\tДеление числа на разряды\n\n”;
   int a;
   cout<<”Введите пятизначное число: “;
   cin>>a;
   cout<<”\n”;
   if (a>=0&&a<=9999)
   {
      cout<<”Вы ввели неверное число. Число должно быть ↵
пятизначным.\n\n”;
   }
   else {
      if (a>=100000)
   {
      cout<<”Вы ввели неверное число. Число должно быть ↵
пятизначным.\n\n”;
   }
   else {

      cout<<”1 цифра равна “<<(a/10000)<<”\n”;
      cout<<”2 цифра равна “<<(a/1000)%10<<”\n”;
```

```
      cout<<”3 цифра равна “<<(a/100)%10<<”\n”;
      cout<<”4 цифра равна “<<(a/10)%10<<”\n”;
      cout<<”5 цифра равна “<<a%10<<”\n”;

   cout<<endl;
   return 0;
   }
   }
}
```

Пример 2.

Программа должна переводить число, введенное с клавиатуры в метрах, в километры.

Код решения представлен в листинге 39.

Листинг 39.

```
#include <iostream>
using namespace std;
int main()
{
   setlocale (LC_ALL, “RUS”);
   cout<<”\t\t\tПеревод числа из метров в километры\n\n\n”;
   float a;
   cout<<”Введите количество метров: “;
   cin>>a;
   cout<<»\n»;
   if (a==1)
   {
   cout<<a<<” метр будет “<<a/1000<<” километра “<<endl;
   }
   if (a>1&&a<=4)
   {
   cout<<a<<» метра будет «<<a/1000<<» километра «<<endl;
   }
```

```
  if (a>=5&&a<1000)
  {
  cout<<a<<» метров будет «<<a/1000<<» километра «<<endl;
  }
  if (a==1000)
  {
  cout<<a<<» метров будет «<<a/1000<<» километр «<<endl;
  }
  if (a>1000&&a<=4900)
  {
  cout<<a<<” метров будет “<<a/1000<<” километра “<<endl;
  }
  if (a>=5000)
  {
  cout<<a<<” метров будет “<<a/1000<<” километров “<<endl;
  }
  cout<<endl;
  return 0;

}
```

Пример 3.

Составить программу, которая требует ввести два числа. Если первое число больше второго, то программа печатает слово «больше». Если первое число меньше второго, то программа печатает слово «меньше». А если числа равны, программа напечатает сообщение «Эти числа равны».

Код решения представлен в листинге 40.

Листинг 40.

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

```
   setlocale (LC_ALL, “RUS”);
   int chislo1, chislo2;
   cout<<”\t\t\tСравнение чисел с использованием оператора ↩
ветвления\n”;
   cout<<endl;
   cout<<”Введите 1 число:  “;
   cin>>chislo1;
   cout<<endl;
   cout<<”Введите 2 число:  «;
   cin>>chislo2;
   cout<<endl;
   if (chislo1>chislo2)
   {
      cout<<»БОЛЬШЕ»<<endl;
   }
   else if (chislo1<chislo2)
   {
   cout<<»МЕНЬШЕ»<<endl;
   }
   else
   {
   cout<<”Эти числа равны”<<endl;
   }
   cout<<endl;
    return 0;
}
```

Пример 4.

Организовать ввод двухзначного натурального числа с клавиатуры. Программа должна определить наименьшую и наибольшую цифры, которые входят в состав данного натурального числа.

Код решения представлен в листинге 41.

Листинг 41.

```
#include <iostream>
using namespace std;
int main()
{
   setlocale (LC_ALL, “RUS”);
   int chislo;
   cout<<”\t\t\tБольшая и меньшая цифры числа\n”;
   cout<<endl;
   cout<<”Введите число, не большее 99:  “;
   cin>>chislo;
   cout<<endl;
   if ((chislo/10)>(chislo%10))
   {
      cout<<chislo/10<<” — большая цифра “<<endl;
      cout<<chislo%10<<” — меньшая цифра “<<endl;
   }
   else {
      if ((chislo/10)<(chislo%10))
   {
      cout<<chislo%10<<” — большая цифра “<<endl;
      cout<<chislo/10<<” — меньшая цифра “<<endl;
   }
   else
   {
      cout<<”цифры равны\n”;
   }
   }
      cout<<endl;
   return 0;
}
```

Пример 5.

Напишите программу, в которой по известной начальной скорости V и времени полета тела T определяется угол, под которым тело брошено по отношению к горизонту. Воспользуйтесь соотношением

$$\alpha = \arcsin(gT/2V).$$

Код решения представлен в листинге 42.

Листинг 42.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   double V, T, param;
   const double Pi = asin(1.0), g = 9.8;
   cout << " V = ";
   cin >> V;

   cout << endl << " T = ";
   cin >> T;

   if(1 < fabs( g * T /( 2 * V)))
      cout << "Incorrect" << endl;
   else {
      param = asin(g * T / (2 * V));
      cout << "Ugol = " << param *(90/Pi) << endl;
   }

   return 0;
}
```

Пример 6.

По введенному номеру дня недели вывести его название, используя оператор множественного выбора. Нумерация дней недели начинается с 1 — понедельник, 2 — вторник и т.д.

Код решения представлен в листинге 43.

Листинг 43.

```
#include <iostream>

using namespace std;

int main()
{
   int number;
   // Дни недели
   const char pn[] = "Понедельник";
   const char vt[] = "Вторник";
   const char sr[] = "Среда";
   const char ch[] = "Четверг";
   const char pt[] = "Пятница";
   const char sb[] = "Суббота";
   const char vs[] = "Воскресенье";

   cout << "1. " << pn << endl;
   cout << "2. " << vt << endl;
   cout << "3. " << sr << endl;
   cout << "4. " << ch << endl;
   cout << "5. " << pt << endl;
   cout << "6. " << sb << endl;
   cout << "7. " << vs << endl;

   cout << "Введите номер: ";
   cin >> number;
   cout << "Вы выбрали:" << endl;
```

```
  switch(number)
  {
    case 1:
      cout << pn;
      break;
    case 2:
      cout << vt;
      break;
    case 3:
      cout << sr;
      break;
    case 4:
      cout << ch;
      break;
    case 5:
      cout << pt;
      break;
    case 6:
      cout << sb;
      break;
    case 7:
      cout << vs;
      break;
    default:
      cout << "Ошибка";
  }
  cout << endl;
  return 0;
}
```

Пример 7.

Напишите программу, которая должна определить, пройдет ли кирпич в отверстие. Размеры отверстия (длина и высота) вводит пользователь. То же самое касается габаритов кирпича: пользователь вводит в программу значения длины, ширины и высоты кирпича.

Код решения представлен в листинге 44.

Листинг 44.

```
#include <iostream>

using namespace std;

int main()
{
   double a, b, c, d, l;
   cout << "Enter the length(a): "; // длина кирпича
   cin >> a;
   cout << "Enter the width(b): "; // ширина кирпича
   cin >> b;
   cout << "Enter the height(c): "; // высота кирпича
   cin >> c;
   cout << "Enter the length of the oppenings(d): "; // вводим ↵
длину отверстия
   cin >> d;
   cout << "Enter the height of the oppenings(l): "; // вводим ↵
высоту отверстия
   cin >> l;
      switch ( (b <= d) && (c <= l) ? 1 : (a <= d) && (c <= l) ? 2 : (a <= d) ↵
&& ( b <= l) ? 3 : –1 ) // составное условие
      {
      case 1  : {cout << "Brick went into the hole by the first party"  ↵
<< endl; break; } // кирпич прошёл через отверстие первой стороной
      case 2  : {cout << "Brick went into the hole by the second party"↵
<< endl; break; } // кирпич прошёл через отверстие второй стороной
      case 3  : {cout << "Brick went into the hole by a third party"   ↵
<< endl; break; } // кирпич прошёл через отверстие третьей стороной
      default : cout << "Brick did not pass through the hole"         ↵
<< endl;         // кирпич не прошёл через отверстие
```

```
    }
  return 0;
}
```

Листинг 45.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  int count;
  float min, max;

  cout << "Please, enter the count: ";
  cin >> count;
  float* arr = new float[count];

  for (int i = 0; i < count; i++){
    cout << "arr[" << i << "] = " ;
    cin >> arr[i];
  }

  min = max = arr[0];

  for (int i = 0; i < count; i++){
    if (min > arr[i]) min = arr[i];
    if (max < arr[i]) max = arr[i];
  }

  delete [] arr;
  cout << "min = " << min << " max = " << max << endl;
```

```
   return 0;
}
```

Пример 2.

Составить программу табулирования функции $y = \mathrm{ctg}(\ln(x))^2$ на интервале $[a, b]$ с шагом h.

Код решения представлен в листинге 46.

Листинг 46.

```
#include <iostream>
// заголовочный файл содержит прототипы математических функций
#include <cmath>
// заголовочный файл содержит прототипы манипуляторов вывода
#include <iomanip>

using namespace std;

int main()
{
   float h = 0.1,      // шаг табулирования
         a = 5.0 / 100, // левая граница интервала
         b = a + 0.5;   // правая граница интервала
   cout << "y = ";
   for ( a; a <= b; a+=0.1) // цикл табулирования функции
   {
      cout << setprecision(3) /*три знака после запятой*/
           << pow(cos(log(a)) / sin(log(a)), 2) << "; "; ↩
// запрограммированная формула
   }
   cout << endl;

   return 0;
}
```

Пример 3.
Вычислить $z = \sin x + \sin x^2 + \cdots + \sin x^n$ при $1 \leq x \geq 10$ $\Delta x = 0{,}2$.
Код решения представлен в листинге 47.

Листинг 47.

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   double z = 0, dx = 0.2;
   int n = 5;

   for (float x = 1; x <= 10; x = x + dx){
      for (int dn = 1; dn <= n; dn++){
         z += sin(pow(x,dn));
      }
      cout << "z[" << x << "] = " << z << endl;
      z = 0;
   }

   return 0;
}
```

7.4. Массивы

Массивы используются для решения множества задач, в которых требуется обработка большого количества данных. Часто с массивом требуется совершить стандартные операции: отсортировать массив, найти в массиве конкретное значение, обратить массив (изменить порядок элементов на противоположный). Часто многомер-

ные массивы используют для проведения математических операций с матрицами, в том числе матричное умножение и транспонирование. Примеры работы с массивами представлены далее.

Пример 1.

Найти Y, если $Y = x_1 + x_2 + ... + x_n$, $x_i = z_i^3 - b_i + a_i^2 / \text{tg}(\beta_1)^2$. Количество x вводится пользователем программы. Для каждого x значения z, b, a, различны (вводятся пользователем программы).

Код решения представлен в листинге 48.

Листинг 48.

```
#include <iostream>

using namespace std;

int main()
{
   setlocale(LC_ALL, "rus");
   cout << "Введите количество иксов: ";
   int number_x = 0; // количество иксов
   cin >> number_x;
   float *arrayPtrZ = new float [number_x]; // динамический ↵
массив для значений Z
   float *arrayPtrB = new float [number_x]; // динамический ↵
массив для значений B
   float *arrayPtrA = new float [number_x]; // динамический ↵
массив для значений A
   float *arrayPtrBe = new float [number_x]; // динамический ↵
массив для значений Betta

   // инициализируем динамические массивы введёнными ↵
с клавиатуры значениями
   for (int counter = 0; counter < number_x; counter++)
   {
```

```
      cout << "Введите значения Z, B, A, Betta для X" << (counter+1) ↩
<< ":\n";
      cout << "Z = ";
      cin >> arrayPtrZ[counter];
      cout << "B = ";
      cin >> arrayPtrB[counter];
      cout << "A = ";
      cin >> arrayPtrA[counter];
      cout << "Betta = ";
      cin >> arrayPtrBe[counter];
   }

   float x = 0, y = 0;
   for (int counter = 0; counter < number_x; counter++)
   {
      x = pow(arrayPtrZ[counter], 3) – arrayPtrB[counter] + ↩
pow(arrayPtrA[counter], 2) / pow(tan(arrayPtrBe[counter]),2); ↩
// формула нахождения икса
      y += x; // суммируем иксы
   }
   delete arrayPtrZ;    delete arrayPtrB;   delete arrayPtrA;    delete ↩
arrayPtrBe; // высвобождаем память
   cout << "\ny = " << y << endl; // ответ

   return 0;
}
```

Пример 2.

Дан одномерный массив, длину массива задает пользователь. Вычислить сумму квадратов тех чисел, модуль которых превышает значение 2,5.

Код решения представлен в листинге 49.

Листинг 49.

```
#include <iostream>
// функции работы с системным временем
#include <ctime>
// функции манипуляции потоками ввода/вывода
#include <iomanip>

using namespace std;

int main()
{
   // инициализация рандомизатора
   srand(time(NULL));

   setlocale(LC_ALL, "rus");

   int length_array = 0; // длина массива

   cout << "Введите длину массива: ";
   cin >> length_array;

   // объявление одномерного динамического массива на 10 ↩
элементов:
   float *ptrarray = new float [length_array];
   float sum = 0; // сумма квадратов чисел

   for (int counter = 0; counter < length_array; counter++)
   {
      ptrarray[counter] = (float)(rand() % 100) / (rand() % 100 + 1); ↩
// заполняем массив случайными числами в диапазоне [0;99]
      cout << fixed << setprecision(2) << ptrarray[counter] << "  ";
   }
   cout << endl;
```

```
   for (int counter = 0; counter < length_array; counter++)
      if (abs(ptrarray[counter]) > 2.5)
         sum += pow(ptrarray[counter],2);

   // высвобождение памяти, отводимой под одномерный ↵
динамический массив:
   delete [] ptrarray;

   cout << "Сумма = " << sum << endl;

   return 0;
}
```

Пример 3.

Отсортировать заданный массив по возрастанию.

Код решения представлен в листинге 50.

Листинг 50.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   int count;

   cout << "Please, enter the count: ";
   cin >> count;
   float* arr = new float[count];

   for (int i = 0; i < count; i++){
      cout << "arr[" << i << "] = " ;
      cin >> arr[i];
```

```
    }

    // Сортировка от меньшего к большему
    for (int i = 0; i < count – 1; i++){
        for (int j = 1; j < count; j++){
            if (arr[j – 1] > arr[j]){
                float temp = arr[j – 1];
                arr[j – 1] = arr[j];
                arr[j] = temp;
            }
        }
    }

    cout << "Sorted array:\n";
    for (int i = 0; i < count; i++){
        cout << "arr[" << i << "] = " << arr[i] << endl; ;
    }

    delete [] arr;

    return 0;
}
```

Пример 4.

Транспонировать матрицу $A = \begin{matrix} 0 & 1 & 2 \\ 2 & 3 & 1 \\ 4 & 1 & 0 \end{matrix}$.

Код решения представлен в листинге 51.

Листинг 51.

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

```
int main()
{
   int A[3][3] = {{0, 1, 2}, {2, 3, 1}, {4, 1, 0}};
   int B[3][3];

   cout << "Matrix A:" << endl;
   for (int i = 0; i < 3; i++){
      for (int j = 0; j < 3; j++)
         cout << A[i][j] << " ";
      cout << endl;
   }

   // Транспонирование
   for (int i = 0; i < 3; i++){
      for (int j = 0; j < 3; j++)
         B[i][j] = A[j][i];
   }
   cout << "Matrix B:" << endl;
   for (int i = 0; i < 3; i++){
      for (int j = 0; j < 3; j++)
         cout << B[i][j] << " ";
      cout << endl;
   }

   return 0;
}
```

Пример 5.

Написать программу для умножения матриц

$$A = \begin{matrix} 1 & 3 \\ 4 & 2 \\ 0 & 4 \end{matrix}; B = \begin{matrix} 1 & 3 & 1 \\ 6 & 1 & 2 \end{matrix}$$

Код решения представлен в листинге 52.

Листинг 52.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   const int l = 3, m = 2, n = 3;
   int A[l][m] = {{1, 3}, {4, 2}, {0, 4}};
   int B[m][n] = {{2, 3, 1}, {6, 1, 2}};
   int C[l][n] = {}; // Инициализация массива нулями

   cout << "Matrix A:" << endl;
   for (int i = 0; i < l; i++){
      for (int j = 0; j < m; j++)
         cout << A[i][j] << " ";
      cout << endl;
   }

   cout << "Matrix B:" << endl;
   for (int i = 0; i < m; i++){
      for (int j = 0; j < n; j++)
         cout << B[i][j] << " ";
      cout << endl;
   }

   // Умножение
   for (int i = 0; i < l; i++){
      for (int j = 0; j < n; j++)
         for (int r = 0; r < m; r++)
         C[i][j] += A[i][r] * B[r][j];
```

```
    }

    cout << "Matrix C:" << endl;
    for (int i = 0; i < l; i++){
        for (int j = 0; j < n; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Пример 6.

Найти след матрицы. След матрицы — сумма элементов главной диагонали. Размер матрицы вводит пользователь, матрицу заполнять случайными числами.

Код решения представлен в листинге 53.

Листинг 53.

```
#include <iostream>
#include <ctime>
#include <iomanip>

using namespace std;

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "rus");

    int number_rows,    // строки
        number_columns; // столбцы

    cout << "Введите количество строк матрицы: ";
```

```
   cin >> number_rows;
   cout << "Введите количество столбцов матрицы: ";
   cin >> number_columns;

   // динамическое создание двумерного массива
   float **ptrarray = new float* [number_rows];
   for (int count = 0; count < number_rows; count++)
      ptrarray[count] = new float [number_columns];

   for (int counter_rows = 0; counter_rows < number_rows; ↩
counter_rows++)
   {
      for (int counter_columns = 0; counter_columns < number_↩
columns; counter_columns++)
      {
         ptrarray[counter_rows][counter_columns] = rand() % 100; ↩
// заполнение массива случайными числами
         cout << setw(2) << ptrarray[counter_rows][counter_columns] ↩
<< "  "; // вывод на экран двумерного массива
      }
      cout << endl;
   }
   cout << endl;

   int trace = 0; // след матрицы
   for (int counter_rows = 0; counter_rows < number_rows; ↩
counter_rows++)
      for (int counter_columns = 0; counter_columns < number_↩
columns; counter_columns++)
         if (counter_rows == counter_columns)
            trace += ptrarray[counter_rows][counter_columns]; ↩
// считаем след матрицы

   // удаление двумерного динамического массива
   for (int count = 0; count < number_rows; count++)
```

```
    delete []ptrarray[count];

  cout << "След матрицы: " << trace << endl;
  return 0;
}
```

Пример 7.

Ввести динамический массив положительных чисел с клавиатуры. Количество элементов массива вводится с клавиатуры. Если введено отрицательное число, то закончить ввод.

Код решения представлен в листинге 54.

Листинг 54.

```
.#include <iostream>

using namespace std;

int main()
{
  setlocale(LC_ALL, "rus");

  int *p = new int[1];
  int *p2;// для копирования данных из массива p перед ⮐
удалением памяти
  int a = 0; // число, которое вводит пользователь
  int nSize = 0; // размер динамического массива, который ⮐
будет увеличиваться на 1 при вводе положительного числа

  while(a >= 0){

    cout << "\nВведите целое число: ";
    cin >> a;

    if(a < 0){
```

```
      break; // если введено отрицательное число, то выход из цикла
    }

    if(nSize == 0)
    {
      p[nSize] = a;
      cout << p[nSize];
      nSize++;
    }
    else
    {
      p2 = new int [nSize + 1];
      for(int i = 0; i < nSize; i++)
      {
        p2[i] = p[i]; // копируем массив
      }
      p2[nSize] = a;

      delete [] p; // очищаем память

      p = new int[nSize + 1]; // выделяем новую память

      for(int i = 0; i < (nSize+1); i++)
      {
        p[i] = p2[i]; // копируем данные из временного массива
       cout << p[i] << " "; // и отображаем все его элементы на экран
      }
      delete [] p2; // очищаем память временного массива

      nSize++; // увеличиваем на 1
    }
  }

    delete [] p;// очищаем память перед завершением ↵
программы
```

```
   return 0;
}
```

Пример 8.

Сформировать случайным образом элементы массива $A(n)$, где n — размерность массива, задаётся с клавиатуры и не превышает 80. Найти максимальный элемент массива и его номер.

Код решения представлен в листинге 55.

Листинг 55.

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

using namespace std;

int main()
{
   const int max_size = 80;

   setlocale(LC_ALL, "ru");
   int n;
   cout << "Введите размерность массива: ";
   cin >> n;
   if ((n > 0) && (n <= max_size)) {
         srand(time(0));
         int* A = new int[n];
         int max, number_max;
         for (int i = 0; i < n; i++) {
               A[i] = rand();
               cout << "A[" << i << "] = " << A[i] << endl;
         }
```

```
        max = A[0];
        number_max = 0;
        for (int i = 1; i < n; i++) {
            if (A[i] > max) {
                max = A[i];
                number_max = i;
            }
        }
        cout << "Максимальный элемент массива: " << max << endl;
        cout << "Номер максимального элемента: " << number_max ↵
<< endl;
    }
    else {
        cout << "Ошибка: размерность массива должна быть ↵
от 1 до " << max_size << endl;
    }

    return 0;
}
```

7.5. Функции

Программы с отдельными функциями нужны в тех случаях, когда код программы массивен, либо есть отдельные строки кода, которые выполняются несколько раз. К примеру, такими программами являются алгоритмы вычисления физических констант и рядов Тейлора. Примеры подобных задач приведены далее:

Пример 1.

Вычислить число «e» с точностью $d = 0.000001$, используя ряд Маклорена

$$e^x = \sum_{i=0}^{\infty} \frac{x^n}{x!}$$

Код решения представлен в листинге 56.

Листинг 56.

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int factorial(int n){
   if (n == 0)
      return 1;
   else
      return n * factorial(n – 1);
}

double member (double x, int n){
   return pow(x, n) / factorial(n);
}

int main()
{
   double y = 0; // Обнуление переменной для результатов ↵
вычисления
   const double e = 2.718282; // Число Эйлера
   const double d = 1e–6;   // Погрешность
   int n = 0; // Счетчик итераций

   while (fabs(y – e) > d) {
      y += member(1, n++);
      cout << "y[" << n << "] = " << y << endl;
   }

   cout << "Congrulations!";
   cout << "|y[" << n << "] – e| = " << fabs(y – e) << \
         " < d = 0.0001"<< endl;
```

```
   return 0;
}
```

Пример 2.

Вычислить, не используя функцию pow, значения функции $z(x,m = x^m \sin^m(xm)$ для значений аргументов:

1. x от –1.1 до 0.3 с шагом 0.2;
2. m от 1 до 5 с шагом 1.

Код решения представлен в листинге 57.

Листинг 57.

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

double step(float m, int e)
{
   float t = 1;
   for ( 0; e; e—)
      t = t *m;
   return t;
}

int main()
{
   setlocale(LC_ALL, "rus");
   const double x1 = –1.1, x2 = 0.3, dx = 0.2;
   double y;
   for (float x = x1; x < x2; x+=dx)
   {
      for (int m = 1; m < 5 ; ++m)
      {
         y = step(x,m)*step(sin(x*m),m);
```

```
        cout << "Значение функции = " << y << "\t\t\tПри m = " << m << ↵
endl;
      }
      cout << "\t\tВычислено при x = " << x << endl;
   }
   return 0;
}
```

8

Задания

1. Вычислить площадь поверхности $S = \pi(R + r)l + \pi R^2 + \pi r^2$ и объем усеченного конуса $v = \left(\frac{1}{3}\right)\pi h(R^2 + r^2 + Rr)$.
2. Используя один оператор вывода — «cout», — программа должна напечатать прямоугольный треугольник из символов знака плюс +.
3. Вычислить значения функции

$$y = \frac{e^{-x_1} + e^{x_2}}{2},\ z = \left(a\sqrt{x_1} + b\sqrt{x_2}\right),\ \text{где}\ x_1 = \left(b + \sqrt{|b^2 + 4ac|}\right)/(2a),$$
$$x_2 = \left(b + \sqrt{|b - 4ac|}\right)/(2a).$$

4. Составить алгоритм для расчета пройденного телом пути $S(t)$, если зависимость этого пути от времени имеет вид $S(t) = at + bt^2 + ct^3$.
5. Составить алгоритм для расчета натяжения перекинутого через блок каната T с подвешенными на концах грузами массами m_1 и m_2 и представленного в виде $T = 2m_1m_2g / (m_1 + m_2)$, где g — ускорение силы тяжести.
6. Производительность шлифовально-затирочной машины составляет A, M^2/ч. Составить алгоритм для вычисления площади обработанной поверхности за время t,ч.

7. Производительность работы уплотняющего катка трактора при уплотнении грунта составляет P, M^3/смена. Составить алгоритм для определения объема уплотненного грунта за N дней при работе в одну смену.
8. Составить алгоритм для расчета $V = 3b$, P = a/27b^2 и $T = 8a/27bR$, если a,b и R известны.
9. Составить алгоритм для нахождения $x = \lg y$, если $y = at^2 + bt + c$. Величины a, b, c и t известны.
10. Составить алгоритм для нахождения отношения стороны пятиугольника c_5 к стороне треугольника c_3, если площадь пятиугольника S_5 равна площади треугольника c_3, причем
$$S_5 = \frac{C_3^2}{4}\sqrt{25 + 2\sqrt{5}},\ S_3 = \frac{C_3^2}{4}\sqrt{3}.$$
11. Составить алгоритм для расчета площади поверхности S, обработанной тремя шлифовально-затирочными машинами за время t, ч работы, если производительность машин известна и равна A, B и C.
12. Для приготовления известково-цементного состава необходимо A,%, известкового теста, B,%, белого цемента, C,%, поваренной соли, D,%, пигмента, остальное составляет вода. Составить алгоритм для расчета необходимого количества воды F, кг при приготовления Q, кг состава.
13. Цех централизованного приготовления малярной продукции за одну смену производит: синтетической шпатлевки A, m, замазки меловой E, m, замазки суриковой F, m, масляных окрашивающих составов G, m, водно-масляной эмульсии H, m. Составить алгоритм для расчета количества выпускаемой продукции за месяц, если в первую декаду цех работал в одну смену, во вторую — в полторы смены, а в третью — в две смены. Результат вывести на печать.
14. Фундамент сооружения имеет в плане размеры A и B. Размеры искусственного основания F в плане, определяемые из выражения $F = (A + 2C) / (B + 2C)$, где $C = 0{,}1B$, должны быть не менее 0,5 мм. Составить алгоритм для расчета и вывода на печать значения F.

15. Моменты инерции J твердых тел с массой m относительно оси вращения выражаются следующими зависимостями:
    а) для сплошного однородного цилиндра с радиусом Rц момент инерции $J = 0.5mR_{ц}^2$;
    б) для полого цилиндра с внутренним радиусом R_1 и внешним радиусом $R_2 = 2R_1$ момент инерции $J = 1/2m\left(R_1 + R_2^2\right)$;
    в) для полого однородного шара с радиусом Rц момент инерции $J = 2/5mR_{ц}^2$.

    Составить алгоритм для определения геометрических размеров указанных тел, если J и m известны и y всех тел одинаковы.
16. Постоянная времени T электрической цепи равна $T = RC$, R и C — соответственно сопротивление и емкость цепи. Составить алгоритм для определения R и C, если при значениях $T < T_0 = R_0C_0$ выбирается только R, а C постоянна и равна C_0. При $T > T_0$ выбирается C, а R постоянна и равна R_0.
17. Составить алгоритм для вычисления функции

$$u = \begin{cases} \sqrt{\dfrac{0{,}3741g15{,}87}{21g\dfrac{2{,}5}{1{,}9R^3 - 1{,}5R + 2}}}, & \text{если} \quad 1{,}9R^3 - 1{,}5R + 2 > 0; \\ \dfrac{\lg\dfrac{R^2 + 2{,}5}{3{,}4}}{\lg\dfrac{8}{1{,}5R^2 + 2}}, & \text{если} \quad 1{,}9R^3 - 1{,}5R + 2 \le 0. \end{cases}$$

18. Составить алгоритм для вычисления членов ряда

$$S = \left(\frac{1}{k+1}\right)\sin kx + \frac{10}{(5k+1)\cos kx},$$

если k изменяется от 1 до 15 с шагом 0,1 при заданном x.

19. Составить алгоритм для расчета функции $y = \frac{10\sin(ax)}{1+ax}$, если x изменяется от 0,1 до 10 с шагом $dx = 0{,}13$, а a — от 1,2 до 5,4 с шагом $da = 1{,}1$
20. Дана функция $x = a\sin(kt + 2\cos kt)$. Составить алгоритм для расчета этой функции, если a изменяется от 5 до 7 с шагом 0,12, t — от 4,2 до 6,2 с.
21. Составить алгоритм для вычисления функции $y = (\sin x) / (2i + 1)$ при x, заданном с точностью до 10^{-3}, причем i изменяется от 1 до 7.
22. Составить алгоритм для вычисления членов ряда
23. $S = \left(\frac{1}{k+1}\right)\sin kx + \frac{10}{(5k+1)\cos kx}$, если k изменяется от 1 до 15 с шагом 0,1 при заданном x.
24. Дана функция $y = 0.5 - \sin 5x$, причем x изменяется от 0 до 2 с шагом $\overline{A}x = 0{,}01$. Составить алгоритм для печати сообщений ДА, если $y > 0$, и НЕТ, если $y < 0$.
25. Дана функция $y = -2x \wedge 2 + 3x + 1.5$. Составить алгоритм для поиска max значения y, если x изменяется в диапазоне от 0.1 до 1 с шагом 0,01.
26. Составить алгоритм для вычисления функции $y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$ при x, заданном с точностью до величины 10^{-3}.
27. Составить алгоритм для определения минимального числа из массива данных a_i.
28. Составить алгоритм для определения из массива данных a чисел, превышающих некоторое заданное число b, и вывести полученные числа на печать.
29. Составить алгоритм для вычисления функции $x = (1 + y^2) / (2.05(3y^3 - 7y + 4))$. Вывести на печать сообщение «Задача не решается» для случая $3y^3 - 7y + 4 \le 0$.
30. Даны числа x_1, x_2, x_3. Составить алгоритм для вывода на печать наибольшего числа.

31. Найти произведение всех значений $y < -1$ и всех значений $y > -1$, если $y = \begin{cases} x + \dfrac{1}{x} & \text{при } x > -1 \\ 2x + \dfrac{1}{x+12} & \text{при } x < -1 \end{cases}$. Значения переменной находятся в диапазоне $-24 \le x < 0$, назначения шага вычислений равно $\Delta x = 0{,}1$.

Вычислить:

32. $P = a(a-z)(a-2z)...(a-z^2)$, если $z = \sqrt{3+\sqrt{6+\cdots+\sqrt{3(n-1)+\sqrt{3n}}}}$. 33. $Z = \prod_{i=3}^{12} \dfrac{\sum_{j=1}^{15} X^{i+j}}{i!}$.

34. $Y = \prod_{i=2}^{100} \dfrac{i+1}{i+2}$. 35. $S = \sum_{i=1}^{n} \dfrac{x + \cos(ix)}{2}$. 36. $Z = \sum_{i=1}^{100} \dfrac{1}{i^2}$.

37. $Y = \sum_{i=3}^{10} \dfrac{x^i}{i + \prod_{j=1}^{i} x^{(i+j)}}$. 38. $Z = X * \prod_{i=2}^{9} \dfrac{i+1}{i!}$.

39. $Y = \sum_{i=-5}^{5} \left(i \prod_{j=1}^{5} \left(j^2 + 1 \right) \right)$. 40. $S = \sum_{i=-5}^{15} \dfrac{ai}{i^2+1} + \prod_{j=2}^{8} \dfrac{xj}{x^2 + j^2}$.

41. $Z = x * \prod_{12}^{9} \dfrac{i + \cos x}{i!}$. 42. $Y = \sqrt{\sum_{i=1}^{N} \sqrt[3]{i^2}} - \prod_{i=1}^{N} \sin i$.

43. $Y = \sum_{k=1}^{n} \dfrac{k1}{\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k+1}}$. 44. $Y = \prod_{k=1}^{n} \left(1 + \dfrac{\sin kx}{k!} \right)$.

45. $Y = \sum_{i=1}^{8} \dfrac{1}{i^2} + \sum_{i=2}^{10} \dfrac{k+1}{k+2}$. 46. $S = \sum_{i=1}^{128} \dfrac{2}{2i^2}$. 47. $Y = \sum_{i=1}^{8} \dfrac{x^i}{i! + \prod_{j=1}^{i} (i+j)}$.

48. $Z = x * i = \prod_{i=2}^{9} \dfrac{i+1}{i^2+1}$. 49. $Y = \sum_{i=1}^{N} i^2 - \prod_{I=1}^{N} j$. 50. $Y = \sum_{P=-N}^{N} \dfrac{\overline{|P|!}}{\sum_{i=0}^{m} i} P = 1$.

51. $S = \sum_{i=1}^{128} \dfrac{\ln i}{2i^2}$. 52. $S = \sum_{i=1}^{100} \dfrac{\tan \sqrt{i}}{i^2}$. 53. $Y = \sum_{i=1}^{N} \sqrt[3]{i^2} - \sqrt{\prod_{i=1}^{N} i}$.

54. $S=\sum_{i=1}^{10}\frac{ai}{i^2+x}+\prod_{j=2}^{5}\frac{xj}{x^2+j^2}$. 55. $Y=\sum_{k=1}^{n}\frac{\sin^k x}{\prod_{i=1}^{k} i}$. 56. $Y=\sum_{i=1}^{n}\left(\frac{1}{i!}+\sqrt{|x|}\right)$.

57. $Y=\sum_{i=1}^{10}\sum_{i=1}^{128}\frac{i^2+j^2}{i^3+j^3}$. 58. $S=\sum_{i=1}^{100}\frac{1}{i^2}$. 59. $y=\prod_{i=1}^{7}\frac{\sum_{j=1}^{N} x^j}{i!}$.

60. $S=\sum_{i=1}^{10}\frac{1}{i!}$. 61. $Z=\sum_{i=1}^{n}\frac{x^i}{i!}+\prod_{i=0}^{n}\frac{1}{\sin(i)}$. 62. $C_n^m=\frac{n!}{m!(n-m)!}$.

63. $Z=\frac{1}{a}+\frac{1}{a^2}+\frac{1}{a^4}+\cdots+\frac{1}{a^{2n}}$.

64. $y=\frac{1}{a}+\frac{1}{a(a+1)}+\cdots+\frac{1}{a(a+1)(a+n)}$.

65. $S=1\cdot 2+2\cdot 3\cdot 4+3\cdot 4\cdot 5\cdot 6+n\cdot(n+1)\cdot(n+2)\ldots 2\cdot n$.

66. $Z=\prod_{i=1}^{8}\frac{yi^2}{y^2+1}$, где $y=\sum_{j=2}^{15}\frac{j+1}{j-1}$.

67. $x=\begin{cases} x^2 & \text{при} \quad x<-10, \\ x^2+x^3 & \text{при} \quad -10\le x<1, \\ x^2+x^3+\frac{1}{x} & \text{при} \quad x\ge 1. \end{cases}$

Значения переменной в диапазоне $15\le x\le 5$, шаг изменения переменной $\Delta x=1$. Найти максимальное значение y.

68. Определить, при каком значении n сумма ряда

$$s=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}.$$

будет отличаться от числа e = 2,718282 на величину $\delta<0.001$.

69. $y=\begin{cases} 1+\sqrt[3]{x} & \text{если} \quad x<a, \\ e^{-\cos x}, & \text{если} \quad a\le x<b, \\ \arccos(x), & \text{если} \quad x\ge b. \end{cases}$

70. Найти максимальное из двух чисел и заменить минимальное число нулем $\min\{x, y\}:=0$. Вычислить $\delta:=\max\{x, y\}$ напечатать значения x,y,z и δ.

71. Дан массив чисел. Найти значение максимального элемента. Если таких элементов несколько, то определить, сколько их. Вывести на экран массив и максимальный элемент.

72. Дан массив чисел. Найти, сколько в нем пар одинаковых соседних элементов и вывести на экран.
73. Дан массив чисел. Найти наибольший элемент, поставить его первым. Вывести исходный и результирующий массивы на экран.
74. Дан массив чисел. Провести сортировку по убыванию. Вывести исходный и результирующий массивы на экран.
75. Имеются данные об успеваемости (в процентах) не более чем 24 учебных групп. Определить, на сколько баллов нужно повысить успеваемость в самой отстающей группе, чтобы достичь среднего уровня успеваемости.
76. Известны данные о среднемесячной температуре за год. Определить, какая была самая высокая температура летом и самая низкая зимой.
77. В коллекции нумизмата не более 90 монет всех возможных достоинств. Определить, сколько имеется монет достоинством в 20 и 50 рублей и какие у них порядковые номера.
78. Дан упорядоченный по убыванию массив чисел. Ввести число *N*. Вставить это число в упорядоченный массив так, чтобы массив-результат тоже был упорядочен по убыванию.
79. Даны два упорядоченных массива целых чисел M1 и M2. Слить их в один упорядоченный массив. Рассмотреть случаи, когда массивы M1 и M2 упорядочены одинаково и когда по-разному.
80. Дан массив чисел. Найти сумму трех самых меньших элементов.
81. Дан массив целых чисел. Вывести массив в обратном порядке на экран.
82. Дан массив целых чисел M1. Ввести массив M2, размерность которого значительно меньше, чем у M1. Определить, сколько раз массив M2 встречается в M1.
83. Дана матрица $A(n, m)$ состоящая из натуральных чисел. Найти в ней наименьший элемент и определить его местоположение. Если таких элементов несколько, то вывести на экран положение каждого из них.

84. Дана матрица $A(n, m)$ состоящая из натуральных чисел. Найти в строках самые правые наименьшие элементы и определить местоположение.
85. Дана матрица $A(n, m)$ состоящая из натуральных чисел. Выбрать в строках самые левые наименьшие элементы и поставить их в первый столбец.
86. Дана матрица $A(n, m)$, состоящая из латинских букв. Отсортировать каждую строку в алфавитном порядке.
87. Дана матрица $A(n, m)$, состоящая из натуральных чисел. Расставить строки таким образом, чтобы элементы в первом столбце были упорядочены по убыванию.
88. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Повернуть ее на 90 градусов по часовой стрелке и вывести результат на экран.
89. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Повернуть ее на 90 градусов против часовой стрелки и вывести результат на экран.
90. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Повернуть ее на 180 градусов и вывести результат на экран.
91. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Зеркально отразить ее элементы относительно горизонтальной оси симметрии. Вывести результат на экран.
92. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Зеркально отразить ее элементы относительно вертикальной оси симметрии. Вывести результат на экран.
93. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Зеркально отразить ее элементы относительно главной диагонали. Вывести результат на экран.
94. Дана квадратная матрица $A(n, m)$, состоящая из натуральных чисел. Зеркально отразить ее элементы относительно побочной диагонали. Вывести результат на экран.
95. Дана квадратная матрица $A(n, m)$. Найти все самые нижние максимальные элементы по столбцам и вывести на экран их значения и местоположение.

96. Даны две матрицы $A(n, m)$ и $A(n, l)$. Объединить их столбцы так, чтобы в матрице-результате в первой строке элементы получились упорядоченными по возрастанию.
97. Дана матрица $A(n, m)$. Определить, сколько пар одинаковых соседних элементов в этой матрице. Элементы считаются соседними, если их индексы в столбцах и/или в строках различаются не более чем на единицу.
98. Дана матрица $A(n, m)$. Вводится произвольная матрица $B(m, m)$. Известно, что m существенно меньше, чем n. Определить, сколько раз вторая матрица встречается в первой.
99. Дана матрица $A(n, m)$. Найти максимальный элемент среди элементов, индексы которых четные, определить сумму элементов главной диагонали.
100. Дан одномерный целочисленный массив $A(n)$ Определить количество элементов массива, которые при делении на 5 дают остаток 2 или 3, а также число элементов, которые делятся без остатка.
101. Дана квадратная матрица $A(n, m)$. Определить произведение элементов, расположенных на главной и побочной диагоналях.
102. В одномерном массиве $A(n, m)$ элементы расположить в порядке убывания.
103. Транспортировать матрицу $A(n, n)$ (поменять местами строки и столбцы), дополнительную матрицу не использовать.
104. Дан одномерный массив $a_1,...a_n$. Получить

$$s = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}.$$

105. Просуммировать элементы столбцов заданной матрицы $A(4–3)$. Результат получить в одномерном массиве $b_1,b_2,b_3,...$
106. Из одномерного массива $A(n)$ сформировать два массива C и B с элементами массива A, делящимися на 2 без остатка или нет.
107. Дан одномерный массив $a_1,...a_n$. Определить количество отрицательных элементов в этом массиве.
108. Дана матрица $A(n, m)$. Получить $b_1,b_2,...,b_m$, где b_1 — значения средних арифметических строк.

109. Дан одномерный массив $a_1,...a_n$. Получить

$$Z = \sqrt{|a_1 \times a_2 \times \ldots \times a_n|}.$$

110. Для матрицы $A(n, n)$ сформировать одномерный массив из положительных элементов, расположенных на главной диагонали.
111. Дан одномерный массив $a_1,...a_n$. Получить

$$P = |a_1| \times |a_2| \times \ldots \times |a_n|.$$

112. Дана матрица $A(n, m)$. Получить $b_1,b_2,...,b_m$, где b_1 — количество отрицательных элементов в каждой строке.
113. Дан одномерный массив. Получить

$$Pa_1,\ldots,a_n. = a_1 \times a_2 \times \ldots \times a_n.$$

114. Дана матрица $A(4, 4)$ и числа K и L ($K <= 4$, $L < K$). Из L строки вычесть K-ю строку.
115. Дан одномерный массив $a_1,a_2,...,a_m$. Получить сумму и количество тех элементов последовательности, которые кратны 5.
116. Элементы матрицы $A(4, 6)$ вычисляются по формуле

$$a_{ij} = i^2 + \cos j.$$

Найти сумму элементов каждой строки.
117. Дан одномерный массив $A(n)$. Найти элементы массива a, которые делятся на свой индекс без остатка.
118. Дан одномерный массив $a_1,a_2,...,a_n$. Определить max элемент среди положительных элементов.
119. Дана матрица $A(n, m)$. Получить вектор $b_1,b_2,...,b_m$, где b_1 является наименьшим значением чисел в соответствующих строках матрицы.
120. В одномерном массиве $A(n)$ поменять местами первый элемент со вторым, третий с четвертым и т.д.

121. В двумерном массиве $A(n, m)$ найти сумму элементов, расположенных ниже побочной диагонали.
122. Дан одномерный массив $a_1,a_2,...,a_n$. Получить

$$P = \left(\sqrt{|a_1| - a_1}\right)^2 + \cdots + \left(\sqrt{|a_n| - a_n}\right)^2$$

123. Дана матрица $A(n, n)$. Найти наибольшее из значений, находящихся на главной диагонали.
124. Элементы матрицы $A(4, 6)$ вычисляются по формуле

$$a_{ij} = i^2 + \sin j.$$

Найти сумму элементов каждой строки.

125. В двумерном массиве $A(n, n)$ найти максимальный элемент среди элементов, расположенных выше главной диагонали, и минимальный элемент среди элементов, расположенных ниже главной диагонали, поменять их местами.
126. Дан одномерный массив $a_1,a_2,...,a_n$. Определить min элемент среди отрицательных элементов массива.
127. Дана действительная матрица $A(m, n)$. Получить вектор $b_1,b_2,...,b_m$, где b_1 равен произведениям элементов строк.
128. Дан одномерный массив $b_1,b_2,...,b_m$. Заменить все члены последовательности, которые больше 7, числом 7. Вычислить количество таких членов.
129. Дана матрица $A(m, n)$. Найти номера строк, все элементы которых одинаковы.
130. Сформировать массивы PLUS и MINUS из положительных и отрицательных элементов массива $A(n)$.
131. В двумерном массиве $A(n, n)$ найти количество элементов, значения которых являются четными, а сами они находятся выше главной диагонали.
132. Дан одномерный массив $q_1,q_2,...,q_m$. Найти те члены q_1, которые при делении на 7 дают остаток 1, 2 или 5.
133. Дана целочисленная матрица $A(15, 15)$. Напечатать индекс первого элемента.

134. Дана матрица $A(m, n)$. Найти сумму элементов, расположенных по периметру матрицы.
135. Дан одномерный массив $a_1,a_2,...,a_n$. Получить среднеарифметическое $a_1,a_2,...,a_n$.
136. Дана матрица $A(m, n)$. Получить $b_1,b_2,...,b_m$. где b_1 — наибольшие из значений чисел в соответствующих строках матрицы.
137. В одномерном массиве $A(n)$ поменять местами четные и нечетные элементы массива, стоящие рядом (n — четное число).
138. Задан одномерный массив $A(n)$. Создать двумерный массив, элементы строк которого образованы из элементов массива, умноженных на номер строки.
139. В одномерном массиве $A(n)$ найти среднее арифметическое значение элементов. Заменить элементы, меньшие среднего арифметического числом 1, а большие — числом 2.
140. Составить одномерный массив $B(n)$ из максимальных элементов каждого столбца двумерного массива $A(n, n)$.
141. Дан одномерный массив $a_1,a_2,...,a_n$. Заменить все элементы массива, которые больше семи, на число 7. Вычислить количество таких элементов.
142. Дана квадратная матрица $A(n, n)$. Найти номера строк, элементы в каждой из которых одинаковы.
143. В двумерном массиве $A(m, n)$ в каждой строке найти минимальный элемент. Сформировать одномерный массив $b_1,b_2,...,b_m$ из этих элементов.
144. В одномерном массиве найти максимальный и минимальный элементы, поменять их местами.
145. В двумерном массиве $A(n, n)$ найти максимальный элемент. Среди элементов столбца, где расположен максимальный элемент, подсчитать количество положительных и отрицательных элементов.
146. В одномерном массиве $A(n)$ найти сумму положительных и отрицательных элементов, определить, какая сумма по модулю больше.
147. В двумерном массиве $A(n, m)$ найти сумму элементов между строками K и L и столбцами P и Q.

148. Дан одномерный массив $a_1,a_2,...,a_n$. Получить среднеарифметическое значение положительных элементов.
149. В одномерный массив $A(n)$ записать символы английского алфавита. Вывести на экран алфавит в обратном порядке.
150. В двумерном массиве $A(n, n)$ найти максимальный и минимальный элементы. Переставить столбцы, содержащие эти элементы. На экран вывести исходную и результирующую матрицы.
151. Дана действительная матрица $A(10, 10)$. В строках с отрицательным элементом на главной диагонали найти сумму всех элементов.
152. Дан одномерный массив $a_1,a_2,...,a_n$. Получить

$$S = a_1 - a_2 + a_3 - \cdots + (-1)^{n+1} a_n.$$

153. Дан одномерный целочисленный массив $A(n)$ Определить количество элементов массива, которые при делении на 5 дают остаток 2 или 3, а также количество элементов, которые делятся без остатка.
154. Дана квадратная матрица $A(n, n)$. Определить произведение элементов, расположенных на главной и побочной диагоналях.

Литература

1. *Липпман Стенли Б.* C++ для начинающих. – М.: Вильямс, 2017. – 1120 с.
2. *Лафоре Р.* Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2015. – 928 с.
3. CppStudio: Основы программирования на языках Си и C++ для начинающих. [Электронный ресурс]. URL: http://cppstudio.com (дата обращения 23.05.2017).
4. Code-Life.ru. Портал о программировании. [Электронный ресурс]. URL: https://code-live.ru (дата обращения 15.05.2017).
5. Cygwin. [Электронный ресурс]. URL: https://cygwin.com (дата обращения 12.05.2017).
6. *Прата С.* Язык программирования C++. Лекции и упражнения. 6 изд. – М.: Вильямс, 2017. – 1248 с.
7. *Страуструп Б.* Язык программирования C++ (стандарт C++11). Краткий курс. – М.: Бином, 2017. – 176 с.
8. *Страуструп Б.* Программирование. Принципы и практика с использованием C++. 2 изд. – М.: Вильямс, 2016. – 1328 с.
9. *Кениг Э., Му Б.* Эффективное программирование на C++. Практическое программирование на примерах. – М.: Вильямс, 2016. – 368 с.

Учебное издание

Елена Валерьевна **Петрунина**
Оксана Николаевна **Савельева**
Эльмин Вагифович **Байрамов**
Дмитрий Константинович **Печерский**

Алгоритмизация и программирование

Учебно-методическое пособие

Ответственный редактор	С.А. Бобко
Редактор/корректор	Ю.Ф. Кравчинская
Технический редактор	К.А. Антонов
Компьютерная верстка	К.А. Антонов

Подписано в печать 26.12.2018. Формат 60x84 $^{1}/_{16}$.
Бумага офисная. Гарнитура *Times New Roman*. Печ. лист 7,5.
Тираж 84 экз. Заказ № 45.

Московский государственный гуманитарно-экономический университет
107150, Москва, ул. Лосиноостровская, д. 49.
Отпечатано в типографии МГГЭУ по технологии CtP.