

Содержание

Введение	3
Глава 1. Основные операторы	4
1.1. Алфавит языка	5
1.2. Типы данных	7
1.3. Переменные	10
1.4. Константы	10
1.5. Операции	13
1.6. Стандартные математические функции.	15
1.7. Порядок выполнения операций в выражениях	16
1.8. Ввод-вывод данных	18
1.9. Примеры решения задач	23
1.10. Упражнения	25
1.11. Контрольные вопросы	27
Глава 2. Базовые операторы	29
2.1. Выражения, блоки и пустые операторы	29
2.2. Операторы ветвления	31
2.3. Операторы цикла	43
2.4. Упражнения	56
2.5. Контрольные вопросы	64
Глава 3. Математические задачи	66
3.1. Рекуррентные соотношения	66
3.2. Вычисление сумм ряда	73
3.3. Упражнения	80
3.4. Контрольные вопросы	83
Глава 4. Простые функции	85
4.1. Простые функции	86
4.2. Рекурсивные функции	97

4.3. Перегрузка функций	100
4.4. Шаблоны функций	103
4.5. Упражнения	107
4.6. Контрольные вопросы	112
Глава 5. Массивы	113
5.1. Указатели	113
5.2. Одномерные массивы	116
5.3. Двумерные массивы	130
5.4. Функции и массивы	136
5.5. Упражнения	158
5.6. Контрольные вопросы	168
Литература	170
Литература	170

Введение

Данное учебное пособие представляет собой первую часть методической разработки для изучения основ программирования на языке C++ [1]. Оно предназначено для студентов младших курсов, обучающихся на специалистов в области информационных технологий.

Пособие состоит из 5 глав.

Первая глава посвящена осуждению базовых элементов языка, обсуждаются вопросы, связанные с переменными, типом данных. Рассмотрены операторы ввода-вывода, как унаследованные от языка C, так и разработанные специально для языка C++.

Во второй главе рассматриваются основные операторы языка с подробными примерами их применения. Вводится понятие блока, или вложенного оператора, операторов ветвления и операторов цикла.

Третья глава посвящена изучению примеров использования основных операторов для решения некоторых математических задач, таких как работа с рекуррентными соотношениями или нахождение сумм функциональных рядов.

Четвертая глава посвящена изучению основного компонента языка C++ — функциям. Рассмотрены разные типы функций: возвращающие значения, не возвращающие значения, рекурсивные. Обсуждаются также вопросы перегрузки и шаблонов функций.

В пятой главе рассматриваются вопросы использования массивов, вводится понятие динамических переменных. Рассмотрены одномерные и двумерные массивы, статические и динамические, приведены примеры использования массивов.

Учебное пособие структурировано следующим образом. В начале каждой главы представлен теоретический материал с примерами программ и подробными пояснениями. Каждая глава завершается набором упражнений и списком контрольных вопросов по теме.

Материалы этого пособия используются на лекционных и практических занятиях со студентами факультета компьютерных наук и информационных технологий и механико-математического факультета Саратовского государственного университета им. Н.Г. Чернышевского.

Глава 1

Основные элементы языка C++

В любом естественном языке существуют символы, слова, словосочетания и предложения. Аналогичные элементы содержит и любой язык программирования, в том числе и C++. Только эти элементы называются иначе

- алфавит языка — множество символов, допустимых в языке;
- лексема, или элементарная конструкция, — минимальная единица языка, имеющая самостоятельный смысл;
- выражение — правило вычисления некоторого значения;
- оператор задает описание некоторого действия.

Каждый элемент языка определяется синтаксисом и семантикой. Синтаксис — это правила построения элементов языка, а семантика — их смысловое содержание.

Операторы бывают двух типов: исполняемые и неисполняемые. Неисполняемые операторы служат для описания данных.

Совокупность операторов образует программу на алгоритмическом языке. Для того чтобы программа была выполнена, требуется перевести ее в машинный код — язык, понятный процессору, и создать исполняемый файл (с расширением `.exe`). Эти действия происходят в несколько этапов:

- выполняются директивы препроцессора, например, подключаются заголовочные файлы;
- компилятор проверяет программу на наличие синтаксических ошибок и при их отсутствии строит объектный модуль;
- компоновщик строит исполняемый файл (`.exe`), добавляя к объектному файлу текст, подключаемых в программе элементов (например, функций).

Рассмотрим основные элементы языка C++.

1.1. Алфавит языка

Алфавит языка C++ содержит [2]:

- буквы латинского алфавита **A...Z**, **a...z**;
- арабские цифры 0, 1, ... , 9;
- пробельные символы: пробел, символ табуляции, символ перехода на новую строку;
- специальные символы: `. , ; : ? ! - + * / % () [] < > { } @ # ~ ^ ' | = & _`

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы).

Границы лексем определяются другими лексемами, такими, как разделители или знаки операций.

Идентификатор — это имя программного объекта (например, переменной, константы, функции). Идентификатор в C++ может содержать буквы, цифры и символ подчеркивания, начинается либо с буквы, либо с символа подчеркивания. В C++ различаются символы верхнего и нижнего регистра, т. е. **MyName**, **myname** и **myName** — три разных идентификатора. Идентификатор не должен совпадать с ключевым словом. Не рекомендуется начинать идентификатор с символа подчеркивание, т. к. он может совпасть с именем системного объекта.

Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Например, **main**, **int**, **bool** и другие.

Комментарии в C++ существуют двух типов:

`//... — текст будет закомментирован до конца строки;`

`/* ... */ — комментируется текст, расположенный между указанными символами.`

Каждая программа на языке C++ состоит из одной или нескольких функций. В каждой программе должна быть функция с именем **main** и только одна. Выполнение программы начинается именно с этой функции. Любая функция имеет следующую структуру:

- заголовок;
- тело функции — операторы, выполняемые в функции.

Заголовок функции состоит из

- типа результата функции;
- имени функции;
- списка параметров.

Пример 1.1. Рассмотрим пример программы нахождения суммы двух чисел.

Листинг 1.1.

```

1 #include <iostream> /*подключаем заголовочный файл iostream, в котором
   содержатся описание классов для ввода-вывода данных*/
2 using namespace std; //используем пространство имен std
3 int main(){           //заголовок главной функции
4     int a, b, s;       //описываем три целых переменных a,b,s
5     cout << "Введите два целых числа"; //вывод на экран
6     cin >> a >> b;     //ввод двух чисел a,b
7     s = a + b;         //в переменную s записываем сумму чисел a и b
8     cout << "Сумма двух чисел равна" << s << endl; //вывод на экран
9     return 0;
10 }
```

Строка 1: с помощью директивы препроцессора `include` подключается заголовочный файл `iostream`. Заголовочные файлы обычно имеют расширение `".h"`. Существует несколько способов подключения заголовочных файлов:

- `#include "name.h"` — этот вариант предполагает, что заголовочный файл `name.h` лежит рядом с текущим файлом.
- `#include <name.h>` — данный вариант предполагает, что заголовочный файл `name.h` необходимо искать рядом в директориях, содержащих заголовочные файлы.
- `#include <name>` — этот способ подключения обычно используется для заголовочных файлов из состава STL. В новых версиях имя заголовочного файла

указывается без расширения `".h"` и необходимо указывать пространство имен `std`.

Строка 2: будет использоваться пространство имен `std`.

Строка 3: заголовок функции `main`. Функция `main` может возвращать значение в вызвавшую систему и принимать параметры из внешнего окружения. Возвращаемое значение должно быть целого типа `int`. Если функция `main()` ничего не возвращает или возвращает 0, вызвавшая система получит значение, означающее успешное завершение. Ненулевое значение означает аварийное завершение. В данном случае список параметров у функции `main` пуст (пустые круглые скобки).

Строка 4: объявление трех переменных `a`, `b`, `s` целого типа `int`. После каждого оператора ставится `;`.

Строка 5: вывод на экран строки «Введите два целых числа» при помощи `cout` и операции направления потока данных `<<`.

Строка 6: вводятся значения переменных `a` и `b` с помощью `cin`.

В операторах ввода-вывода знаки `<<` и `>>` указывают направление потока данных.

Строка 7: переменной `s` присваивается значение выражения `a + b`.

Строка 8: вывод на экран фразы «Сумма двух чисел равна», значения переменной `s` и перевод курсора на новую строку (`endl`).

Строка 9: оператор `return` возвращает значение 0 как результат функции `main`.

Строка 10: конец функции `main`.

□

1.2. Типы данных

Тип данных у объекта (переменной, значения и т.д.) определяет

- объем памяти, выделяемой для хранения данных;
- возможные значения;
- структуру хранения данных;
- набор функций и операций, применяемых к данным.

Типы данных можно разделить на простые и составные. К простым стандартным типам данных относятся

- целые,
- вещественные,
- символьные,
- логический.

Существуют простые типы данных, определяемые пользователем — перечислимые типы.

К составным типам относятся

- массивы,
- структуры,
- файлы,
- объекты.

Существует один целочисленный тип данных `int` и четыре спецификатора `signed` (знаковый), `unsigned` (беззнаковый), `short`, `long`. Спецификаторы `short` и `long` могут использоваться для описания короткого или длинного целого типов данных. При этом название типа `int` может быть опущено. То есть, `short int` соответствует `short`, а `long int` — `long`. Объем памяти, выделяемый под целочисленные типы, различается в зависимости от компьютера и компилятора (Таблица 1.1).

Таблица 1.1. Объем памяти для целого типа данных

Тип данных	16-разрядная система	32-разрядная система	64-разрядная система
<code>short</code>	2 байта	2 байта	2 байта
<code>int</code>	2 байта	4 байта	4 или 8 байт
<code>long</code>	4 байта	4 байта	4 или 8 байт

Для 64-разрядной системы объем памяти отличается для разных моделей данных.

Спецификаторы `signed` (знаковый) и `unsigned` (беззнаковый) можно добавлять к любому имени типа. По умолчанию все целочисленные типы являются знаковыми, поэтому спецификатор `signed` можно опускать.

В C++ существует два символьных типа данных: `char` и `wchar_t`. Под хранение данных типа `char` отводится 1 байт, что достаточно для хранения 256 символов, закодированных в таблице ASCII. Объекты типа `char` можно использовать для хранения целочисленных значений в диапазоне от -128 до 127 . Тип `wchar_t` предназначен для работы с набором символов, для хранения которых недостаточно 1 байта типа данных `char`. Размер этого типа зависит от реализации, но, как правило, соответствует типу `short`.

Для хранения логических значений используется тип `bool`. Величины типа `bool` могут принимать одно из двух значений `true` (истина) или `false` (ложь) и занимают 1 байт памяти. В логических выражениях 0 (ноль) истолковывается как ложное значение (`false`), а любое другое ненулевое значение соответствует логическому значению истина (`true`). При преобразовании к целому типу `true` интерпретируется как 1, а `false` — как 0.

Для описания вещественных чисел (чисел с плавающей точкой) можно использовать один из трех типов: `float`, `double` и `long double`. Структура чисел с плавающей точкой отличается от целочисленных. Для вещественных чисел хранится мантисса и порядок. Диапазон значений и объем памяти, отводимый для хранения вещественных чисел приведен в таблице 1.2.

Таблица 1.2. Диапазон значений и объем памяти для вещественных типов данных

Тип данных	Диапазон значений	Объем памяти в байтах
<code>float</code>	$3.4E - 38 \dots 3.4E + 38$	4
<code>double</code>	$1.7E - 308 \dots 1.7E + 308$	8
<code>long double</code>	$3.4E - 4932 \dots 3.4E + 4932$	10 (в некоторых компиляторах совпадает с типом <code>double</code>)

При указании диапазона значений для вещественных чисел указываются самое минимальное и самое максимальное по модулю числа.

В языке C++ существует еще один основной тип данных — `void`. Этот тип данных используется только для описания типа результата функции, которая ничего не возвращает.

1.3. Переменные

Переменная — это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием любая переменная должна быть описана.

При описании переменной указывается ее тип и имя. Пример описания:

```
int x, y; //описаны две переменные x и y типа int
float z; //описана переменная z типа float
```

При описании нескольких переменных одного типа их можно указывать через запятую. При описании переменных их можно инициализировать, то есть присваивать начальные значения. Пример:

```
1 int a = 5, b(6); //переменная a описывается и ей присваивается значение 5,
    переменная b описывается и ей присваивается значение 6
2 float d(-7.5); // переменная d описывается и ей присваивается значение
    -7.5
```

Инициализация может быть произведена либо с помощью знака равенства = либо с помощью скобок (...).

1.4. Константы

Константа — это величина, которая не изменяется в течении всей программы. Константы бывают двух видов: именованные и неименованные. Неименованные константы — это обычные значения. Например, в выражении $5 * x + 7.2 * y$ неименованными константами являются 5 и 7.2.

Неименованные **целочисленные константы** могут быть записаны в одной из трех форм:

- десятичная, например, 15, 654;
- восьмеричная — слева к записи числа добавляется 0, например, 061, 0251;
- шестнадцатеричная — слева к записи числа добавляется 0x или 0X, например, 0x15, 0X6A.

Неименованные **вещественные константы** могут быть записаны в одном из двух видов

- число с фиксированной точкой. Например, 0.000015, -14000.08. Дробная часть отделяется от целой точкой.
- число с плавающей точкой (в экспоненциальной форме). Например, 1.5E-5, -1.400008E+4, -8.7e+7.

Число с плавающей точкой имеет следующий вид mEp или meP , где m — это вещественное число, называемое мантиссой, p — целое знаковое число, называемое порядком. Для перевода числа с плавающей точкой к привычному нам числу с фиксированной точкой необходимо мантиссу умножить на десять в степени порядка, т. е.

$$mEp = m * 10^p.$$

По умолчанию вещественные константы имеют тип **double**. Можно явно указать тип констант с помощью суффиксов **F**, **f** для типа **float** и **L**, **l** для типа **long double**. Например,

в выражениях $x = 5.3 * 10$, $y = 8.4 - 123$ константы 5.3 и 8.4 будут иметь тип **double**;

в выражениях $x = 5.3f * 10$, $y = 8.4F - 123$ константы 5.3 и 8.4 будут иметь тип **float**;

в выражениях $x = 5.3l * 10$, $y = 8.4L - 123$ константы 5.3 и 8.4 будут иметь тип **long double**.

Символьные константы представляют собой одни или несколько символов, заключенных в апострофы. Например, `'a'`, `'-'`, `'w'`, `'\n'`, `'\t'`.

Символьные константы, состоящие из одного символа, занимают в памяти один байт и имеют стандартный тип **char**. Двухсимвольные константы занимают два байта и имеют тип **int**, при этом первый символ размещается в байте с меньшим адресом.

Символ обратной косой черты используется для представления:

- кодов, не имеющих графического изображения (например, `'\a'` — звуковой сигнал, `'\n'` — перевод курсора в начало следующей строки);
- символов апострофа (`'`), обратной косой черты (`\`), знака вопроса (`?`) и кавычки (`"`);

- любого символа с помощью его шестнадцатеричного или восьмеричного кода, например, `\073`, `\0xF5`. Числовое значение должно находиться в диапазоне от 0 до 255.

Последовательности символов, начинающиеся с обратной косой черты, называют управляющими, или «escape»-последовательностями. В таблице 1.3 приведены их допустимые значения.

Таблица 1.3. Управляющие последовательности символов

Управляющая последовательность	Значение
<code>\a</code>	Звуковой сигнал
<code>\b</code>	Удаление предыдущего символа
<code>\f</code>	Перевод страницы
<code>\n</code>	Новая строка
<code>\t</code>	Горизонтальная табуляция
<code>\v</code>	Вертикальная табуляция
<code>\r</code>	Возврат каретки
<code>\'</code>	Апостроф (одионая кавычка)
<code>\"</code>	Двойная кавычка
<code>\\</code>	Обратная косая черта
<code>\?</code>	Символ вопросительного знака
<code>\0ddd</code>	Восьмеричный код символа
<code>\0xdddd</code>	Шестнадцатеричный код символа

Строковые константы заключаются в кавычки. Внутри строковых констант могут использоваться управляющие последовательности. Примеры:

```
"Hello, my friend! \n"
```

```
"How are you? \n"
```

Для описания именованных констант используется ключевое слово `const`.

Формат объявления констант выглядит следующим образом

```
const <тип константы> <имя константы> = <значение>;
```

Например,

```
const int a = 10000;
```

```
const float pi = 3.1415;
```

1.5. Операции

Арифметические операции

Арифметические унарные операции: ++ (инкремент), -- (декремент), + (унарный плюс), - (унарный минус).

Операции инкремент (увеличение на единицу) и декремент (уменьшение на единицу) имеют две формы префиксную и постфиксную. В префиксной форме сначала выполняются операции инкремента и декремента для соответствующих переменных, а затем измененные переменные используются в выражении. В постфиксной форме сначала значения переменных используются в выражении, а затем значения переменных изменяются на единицу. Рассмотрим примеры применения этих операций.

После выполнения фрагмента программы

```
int a = 1, b = 2, c, d;  
c = ++a; d = --b;
```

переменные примут следующие значения $a = 2$, $b = 1$, $c = 2$, $d = 1$.

После выполнения фрагмента программы

```
int a = 1, b = 2, c, d;  
c = a++; d = b--;
```

переменные примут следующие значения $a = 2$, $b = 1$, $c = 1$, $d = 2$.

Арифметические бинарные операции: + (сложение), - (вычитание), * (умножение), / (деление), % (остаток от деления).

Операция деления к целым числам применяется как целочисленное деление, а к вещественным как обычное вещественное деление. Операция взятия остатка от деления применима только к целым числам.

Пример 1.2. Применение операций деления и остатка от деления

Листинг 1.2.

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

```

4  cout << 5 / 2 << endl;    //будет выведено 2
5  cout << 5.0 / 2 << endl;  // будет выведено 2.5
6  cout << 5 % 2 << endl;    // будет выведено 1
7  }

```

□

Логические операции

Логической унарной операцией является ! (отрицание).

Логические бинарные операции: || (логическое или), && (логическое и).

Логические операции применяются к операндам логического типа данных. Результат операции || будет **true** (истина), если хотя бы один из операндов будет равен **true**. Результат операции && будет **true** (истина), тогда и только тогда, когда оба операнда будут равны **true**. Значения логических операций представлены в таблице 1.4.

Таблица 1.4. Логические операции

x	y	!x	x y	x && y
true	true	false	true	true
true	false	false	true	false
false	true	true	true	false
false	false	true	false	false

Операции отношения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), == (сравнение на равенство), != (сравнение на неравенство). Операндами для данных операций могут быть арифметические выражения или указатели. Результат операций отношения — это значение логического типа (**true** или **false**). Например, результатом операции $3 < 5$ будет значение **true**, а результатом выражения $5 <= -5$ будет значение **false**.

Операции присваивания:

- простая операция присваивания =;
- сложные операции присваивания += (присваивание со сложением), -= (присваивание с вычитанием), *= (присваивание с умножением), /= (присваивание с делением), %= (присваивание с остатком от деления).

Рассмотрим равноценные выражения с использованием сложных операций присваивания и простой операции присваивания.

С использованием сложных операций	С использованием простой операции	Результат операции
<code>int a = 1, b = 2; a += b;</code>	<code>int a = 1, b = 2; a = a + b;</code>	<code>a = 3</code>
<code>int a = 10, b = 2; a -= b;</code>	<code>int a = 10, b = 2; a = a - b;</code>	<code>a = 8</code>
<code>int a = 10, b = 2; a *= b;</code>	<code>int a = 10, b = 2; a = a * b;</code>	<code>a = 20</code>
<code>int a = 10, b = 2; a /= b;</code>	<code>int a = 10, b = 2; a = a / b;</code>	<code>a = 5</code>
<code>float a = 5, b = 2; a /= b;</code>	<code>float a = 5, b = 2; a = a / b;</code>	<code>a = 2.5</code>
<code>int a = 10, b = 2; a %= b;</code>	<code>int a = 10, b = 2; a = a % b;</code>	<code>a = 0</code>

Еще одной унарной операцией является операция **sizeof** — определение размера выражения или типа в байтах. Операндом для данной операции может быть как выражение так и тип данных. Например, после выполнения фрагмента

```
int x = 100, y = -6;
cout << "Размер целого типа = "<< sizeof (int)<< endl;
cout << "Размер выражения 2*x = "<< sizeof(2 * x)<< endl;
cout << "Размер выражения x*y+0.5 = "<< sizeof(x*y + 20)<< endl;
```

На экран будет выведено

```
Размер целого типа = 4
Размер выражения 2*x = 4
Размер выражения x*y+0.5 = 4
```

В C++ существует одна тернарная операция — **логическая операция**. Формат записи логической операции

$$\langle \text{операнд } 1 \rangle ? \langle \text{операнд } 2 \rangle : \langle \text{операнд } 3 \rangle$$

где *операнд 1* — это выражение логического или арифметического типа. Если *операнд 1* — истина (**true**), то результат операции равен *операнду 2*, иначе *операнду 3*. Если *операнд 1* — арифметическое выражение, то оно считается истинным при любом ненулевом значении.

1.6. Стандартные математические функции.

В языке C стандартные математические функции содержались в заголовочном файле `"math.h"`. В C++ также можно использовать этот заголовочный файл. Кроме

того в C++ можно подключать заголовочный файл `"cmath.h"`, в котором содержатся более разнообразные математические функции. В таблице 1.5 приведены некоторые математические функции из заголовочного файла `"cmath"`.

Таблица 1.5. Математические функции

Функция	Описание
<code>abs(x)</code>	Абсолютная величина x . В заголовочном файле <code>math.h</code> необходимо использовать функция <code>fabs(x)</code> для вещественного x
<code>acos(x)</code>	Арккосинус x
<code>asin(x)</code>	Арсинус x
<code>atan(x)</code>	Арктангенс x
<code>ceil(x)</code>	Округление до ближайшего целого больше x
<code>cos(x)</code>	Косинус x
<code>exp(x)</code>	e^x
<code>floor(x)</code>	Округление до ближайшего целого меньше x
<code>log(x)</code>	Натуральный логарифм x
<code>pow(x, y)</code>	x^y , где x — вещественное, y — любое числовое
<code>sin(x)</code>	Синус x
<code>sqrt(x)</code>	Корень квадратный из x
<code>tan(x)</code>	Тангенс x

Примечание: все тригонометрические функции работают с радианами.

Математические константы не определены в стандарте C++. Чтобы использовать их, необходимо сначала определить `_USE_MATH_DEFINES`, а затем включить `cmath` или `"math.h"`. Пример:

```
#define _USE_MATH_DEFINES
#include <cmath>
```

В месте использования константы указывается ее имя. В таблице 1.6 приведены некоторые математические константы.

1.7. Порядок выполнения операций в выражениях

Операции в выражениях выполняются в соответствии с их приоритетом:

Таблица 1.6. Математические константы

Имя константы	Выражение	Значение
M_E	e	2.71828182845904523536
M_LOG2E	$\log_2 e$	1.44269504088896340736
M_LOG10E	$\log_{10} e$	0.434294481903251827651
M_LN2	$\ln(2)$	0.693147180559945309417
M_LN10	$\ln(10)$	2.30258509299404568402
M_PI	π	3.14159265358979323846
M_PI_2	$\pi/2$	1.57079632679489661923
M_PI_4	$\pi/4$	0.785398163397448309616
M_1_PI	$1/\pi$	0.318309886183790671538
M_2_PI	$2/\pi$	0.636619772367581343076
M_2_SQRTPI	$2/\sqrt{\pi}$	1.12837916709551257390
M_SQRT2	$\sqrt{2}$	1.41421356237309504880
M_SQRT1_2	$1/\sqrt{2}$	0.707106781186547524401

1. выражения в скобках;
2. унарные операции $-$, $+$, $--$ (префиксный декремент), $++$ (префиксный инкремент), $!$, `sizeof`;
3. значения функций;
4. операции $*$, $/$, $\%$;
5. бинарные операции $+$, $-$;
6. операции сравнения: $<$, $<=$, $>$, $>=$;
7. операции сравнения $==$, $!=$;
8. операция $\&\&$;
9. операция $||$;
10. тернарная операция;
11. операции присваивания.

Если в выражении встречаются несколько операций с одинаковым приоритетом, то они выполняются слева направо. Исключения составляют унарные операции, условная операция и операции присваивания.

Выражение `a = b = c` означает, что переменной `b` присваивается значение переменной `c`, а переменной `a` присваивается значение переменной `b`.

В выражении `a = b + c + d` сначала выполняются операции сложения слева направо, а затем операция присваивания.

При вычислении выражений возможно преобразование типов. Преобразование типов может быть неявным и явным.

Рассмотрим пример:

```
float a = 1, b;  
b = a + 5;
```

Неименованная константа 5 по умолчанию имеет тип `int`. При выполнении операций в выражении происходит неявное преобразование типа `int` в тип данных `float`.

После выполнения фрагмента программы

```
float a = 16/5;
```

значение переменной `a` = 3, так как если константы 16 и 5 целые, то и деление будет целочисленным. Для того чтобы деление было вещественным, необходимо провести явное преобразование типов. Например, предыдущее выражение может быть записано

```
float a = float (16)/ 5;
```

1.8. Ввод-вывод данных

В C++ существует два способа консольного ввода-вывода данных. Один из способов унаследован из языка C. В этом случае ввод данных производится с помощью функции `scanf()`, а вывод с помощью функции `printf()`. Прототипы этих функций содержатся в заголовочном файле `stdio.h`.

Прототип функции `printf`:

```
int printf(char *управляющая строка , ...);
```

С помощью этой функции можно вывести на экран монитора строку символов, число, значение переменной и т. д. Функция `printf` возвращает количество выведенных символов.

Управляющая строка содержит два типа информации: символы, которые непосредственно выводятся на экран, и спецификаторы формата, определяющие, как выводить аргументы.

Функция `printf()` — это функция форматированного вывода. Это означает, что в параметрах функции необходимо указать формат данных, которые будут выводиться. Формат данных указывается спецификаторами формата. Спецификатор формата начинается с символа `%`, за которым следует код формата (Таблица 1.7).

Таблица 1.7. Спецификаторы формата

Код	Формат
<code>%c</code>	Символ
<code>%d</code>	Десятичное целое число со знаком
<code>%i</code>	Десятичное целое число со знаком
<code>%e</code>	Экспоненциальное представление числа (в виде мантиссы и порядка) (<code>e</code> на нижнем регистре)
<code>%E</code>	Экспоненциальное представление числа (в виде мантиссы и порядка) (<code>E</code> на верхнем регистре)
<code>%f</code>	Десятичное число с плавающей точкой
<code>%g</code>	Использует более короткий из форматов <code>%e</code> или <code>%f</code>
<code>%G</code>	Использует более короткий из форматов <code>%E</code> или <code>%F</code>
<code>%o</code>	Восьмеричное число без знака
<code>%s</code>	Символьная строка
<code>%u</code>	Десятичное целое число без знака
<code>%x</code>	Шестнадцатиричное без знака (строчные буквы)
<code>%X</code>	Шестнадцатиричное без знака (прописные буквы)
<code>%p</code>	Выводит указатель
<code>%n</code>	Соответствующий аргумент должен быть указателем на целое число.
<code>%%</code>	Выводит знак процента

В спецификаторе формата, после символа `%` может быть указана точность. Точность

- указывает на минимальное количество символов, которое должно появиться при

обработке типов `d`, `i`, `o`, `u`, `x`, `X`;

- указывает на минимальное количество символов, которое должно появиться после десятичной запятой (точки) при обработке типов `e`, `E`, `f`, `F`;
- максимальное количество значащих символов для типов `g` и `G`;
- максимальное число символов, которые будут выведены для типа `s`;

Пример 1.3. Фрагмент кода программы, использующего оператор вывода `printf`.

Листинг 1.3.

```
1 ...
2 float x = 10.3563;
3 printf("Переменная x = %.3f");
4 int a = -3;
5 printf("Переменная a = %.6d");
6 ...
```

Результат выполнения строки 3 листинга 1.3:

`x = 10.356.`

Результат выполнения строки 5 листинга 1.3:

`a = -3.`

Под вывод целочисленного числа (переменная `a`) отведено 6 позиций, выравнивание происходит по правому краю. □

Функция `scanf()` — функция форматированного ввода. С ее помощью можно ввести данные со стандартного устройства ввода (клавиатуры). Вводимыми данными могут быть целые числа, числа с плавающей точкой, символы, строки и указатели. Прототип функции `scanf`:

```
int scanf(const char *⟨управляющая строка⟩, ...);
```

Функция `scanf` возвращает количество удачно считанных переменных. Управляющая строка содержит три вида символов: спецификаторы формата, пробелы и другие символы. Спецификаторы формата

`%c`, `%d`, `%i`, `%e`, `%h`, `%o`, `%s`, `%p`, `%n`,

значения которых приведены в таблице 1.7.

Пример 1.4. Пример вызова функций `printf` и `scanf`:

```

1 ...
2 scanf("%i",&a); // считывается значение целочисленной переменной a
3 scanf("%i %f",&b,&c); /*считываются значения переменных b (целый тип) и c (
    вещественный тип)*/
4 printf("Result\n a=%d b=%d c=%f", a, b, c); /*вывод на экран значений
    переменных*/
5 ...

```

□

Второй способ консольного ввода-вывода данных — это потоковый ввод-вывод данных. Для применения этого способа необходимо подключить заголовочный файл `iostream.h`.

`cin` — входной поток; `cout` — выходной поток.

Прототипы этих потоков описаны в пространстве имен `std`, поэтому при подключении заголовочного файла `iostream` необходимо еще указывать строку

`using namespace std;`

Пример 1.5. Пример поиска среднего арифметического двух целых чисел

Листинг 1.4.

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     int a, b; float c;    /*описание переменных*/
5     cin >> a >> b;        /*ввод значений переменных a и b*/
6     c = float (a + b) / 2; /*подсчет среднего арифметического*/
7     cout << "с = " << c; /*вывод на экран результата */
8     return 0;
9 }

```

□

Можно не писать строку `using namespace std;`, но в этом случае перед каждым использованием `cin` и `cout` необходимо прописывать `std::`.

Пример 1.6. Пример поиска среднего арифметического двух целых чисел.

Листинг 1.5.

```

1 #include <iostream>
2 int main(){

```

```

3  int a, b; float c;    /*описание переменных*/
4  std::cin >> a >> b;  /*ввод значений переменных a и b*/
5  c = float (a + b) / 2; /*подсчет среднего арифметического*/
6  std::cout << "с = " << c; /*вывод на экран результата */
7  return 0;
8  }

```

Результат работы программы:

<i>a</i>	<i>b</i>	результат
1	9	5
−3	9	3

□

Символы << и >> указывают направление потока данных.

В потоковом вводе-выводе возможно форматировать вывод данных на экран с помощью манипуляторов. Для этого необходимо подключить заголовочный файл `iomanip.h`. Манипуляторы встраиваются непосредственно в операторы ввода-вывода. В таблице 1.8 приведены основные манипуляторы форматирования.

Таблица 1.8. Манипуляторы

Манипулятор	Описание
setw(n)	Определяет ширину поля вывода в <i>n</i> символов
setprecision (n)	Определяет количество цифр (<i>n</i> – 1) в дробной части числа
endl	Переход на новую строку
left	Выравнивание по левой границе (по умолчанию)
right	Выравнивание по правой границе
boolalpha	Вывод логических величин в текстовом виде
noboolalpha	Вывод логических величин в числовом виде
dec	Вывод величин в десятичной системе счисления (по умолчанию)
oct	Вывод величин в восьмеричной системе счисления (для этого нужно снять флаг вывод в десятичной)
hex	Вывод величин в шестнадцатеричной системе счисления (для этого нужно снять флаг вывод в десятичной)
scientific	Экспоненциальная форма вывода вещественных чисел
fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)

Пример 1.7. Использование манипуляторов:

Листинг 1.6.

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int main(){
5     int a = 20;
6     cout << "a10 = " << a << endl;
7     cout << "a8 = " << oct << a << endl;
8     cout << "a16 = " << hex << a << endl;
9     float b = -0.00012;
10    cout << fixed << "b = " << b << endl;
11    cout << scientific << "b = " << b << endl;
12    cout << setprecision(3) << "b = " << b << endl;
13    return 0;
14 }
```

На экран будет выведено

```
a10 = 20
a8 = 24
a16 = 14
b = -0.000120
b = -1.200000e-004
b = -1.200e-004
```

□

1.9. Примеры решения задач

Пример 1.8. Необходимо написать программу для подсчета значения $y = \sqrt{x^5 + 5x}$, значение x вводится с клавиатуры.

Приведем два варианта программы для решения этой задачи с использованием `scanf` и `printf` (листинг 1.7) и с использованием `cin` и `cout` (листинг 1.8)

Листинг 1.7.

```
1 #include <stdio.h>
2 #include <cmath>
```

```

3 int main(){
4     float x, y;           //объявление переменных x и y
5     printf ("x = ");      // вывод строки "x="
6     scanf ("%f", &x);     //ввод значения переменной x
7     y = sqrt(pow(x, 5) + 5*x); //вычисление y
8     printf ("y=%3f", y);  //вывод значения y
9     return 0;
10 }

```

Листинг 1.8.

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 int main(){
5     float x, y;           //объявление переменных x и y
6     cout << "x = ";      // вывод строки "x="
7     cin >> x;             //ввод значения переменной x
8     y = sqrt(pow(x, 5) + 5*x); //вычисление y
9     cout << "y=" << y << endl; //вывод значения y
10    return 0;
11 }

```

Результат работы программы:

x	y
5	56.125
-2	4.690

□

Пример 1.9. Необходимо написать программу для вычисления площади равнобедренного треугольника, две стороны которого равны a , третья равна b . Значения a и b вводятся с клавиатуры.

Для решения этой задачи вспомним формулу площади для равнобедренного треугольника $S = \frac{1}{2}bh$, где h — высота, опущенная на основание. Высота вычисляется по формуле $h = \sqrt{a^2 - \frac{b^2}{4}}$.

Приведем два варианта программы для решения этой задачи с использованием `scanf` и `printf` (листинг 1.9) и с использованием `cin` и `cout` (листинг 1.10)

Листинг 1.9.


```

1 #include <stdio.h>
2 #include <cmath>
3 int main(){
4     float a, b, h, S;           /*объявление переменных a,b,h,S*/
5     printf ("a = "); scanf("%f", &a); /*вывод приглашения "a=", ввод значения
        переменной a*/
6     printf ("b = "); scanf("%f", &b); /*вывод приглашения "b=", ввод значения
        переменной b*/
7     h = sqrt(a*a - b*b / 4)      /*вычисление длины высоты h*/
8     S = b*h / 2;                /*вычисление площади S*/
9     printf("S = %.3f\n", S);     /*вывод на экран значения площади S*/
10    return 0;
11 }

```

Листинг 1.10.

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 int main(){
5     float a, b, h, S;           /*объявление переменных a,b,h,S*/
6     cout << "a = "; cin >> a;   /*вывод приглашения "a=", ввод значения
        переменной a*/
7     cout << "b = "; cin >> b;   /*вывод приглашения "b=", ввод значения
        переменной b*/
8     h = sqrt(a*a - b*b/4);       /*вычисление длины высоты h*/
9     S = b*h/2;                  /*вычисление площади S*/
10    cout << "S = " << S << endl; /*вывод на экран значения площади S*/
11    return 0;
12 }

```

Результат работы программы:

a	b	S
3	4	4.472
3	3	3.897

□

1.10. Упражнения

Номера и количество заданий, а также способ отчетности определяется преподавателем.

I. Написать программу, которая вычисляет значение функции $y(x)$, значение x вводится с клавиатуры.

- | | |
|---|--|
| 1) $y = \sin 2x^5 - 10$ | 2) $y = \frac{x^3 - 5}{x^2 + 0.1}$ |
| 3) $y = \sqrt{x^2 + 100} - x$ | 4) $y = \ln(25 + x^4)$ |
| 5) $y = e^{16-x} + \cos x$ | 6) $y = \operatorname{tg} 5^{x+10} + 15$ |
| 7) $y = \sin \pi(x + 5) - 0.25$ | 8) $y = \cos 2x^3$ |
| 9) $y = \frac{x - 2.5}{3 + x^4}$ | 10) $y = \frac{\sqrt{6x^6 + 10}}{6 + x^2}$ |
| 11) $y = \sqrt{\frac{ \sin 2x }{25}}$ | 12) $y = \sin \frac{100}{x^4 + 3}$ |
| 13) $y = \ln 2^{x+0.5}$ | 14) $y = \frac{2x^3}{3^{2x}}$ |
| 15) $y = \sqrt[3]{2x + 5}$ | 16) $y = \frac{2^x}{\sin 2x + \cos 2x}$ |
| 17) $y = \sqrt{\frac{\ln(x^4 + 16)}{x^4 + 16}}$ | 18) $y = \sin \frac{2x + 5}{6} + 0.5$ |
| 19) $y = 5x^5 - \frac{2x^3}{15}$ | 20) $y = \frac{\sin x - 15x^2}{3}$ |

II. Написать программу, которая вычисляет некоторую величину по известной.

Известная величина вводится с клавиатуры. Найденную величину необходимо вывести на экран.

- 1) Найти площадь квадрата S по длине диагонали d .
- 2) Найти периметр квадрата P по длине диагонали d .
- 3) Найти площадь квадрата S по периметру P .
- 4) Найти площадь прямоугольника S по длинам сторон a и b .
- 5) Найти длину гипотенузы по длинам катетов a и b .
- 6) Найти площадь прямоугольного треугольника S по двум катетам a и b .
- 7) Найти площадь прямоугольного треугольника S по катету a и гипотенузе.
- 8) Найти длину окружности l по радиусу r .

- 9) Найти площадь круга S по радиусу r .
- 10) Найти площадь круга S по длине окружности l .
- 11) Найти расстояние между двумя точками по их координатам на плоскости.
- 12) Найти расстояние между двумя точками по их координатам в пространстве.
- 13) Найти площадь круга S , вписанного в равносторонний треугольник со стороной a .
- 14) Найти площадь круга S , описанного около прямоугольного треугольника с гипотенузой .
- 15) Найти площадь треугольника по длинам трех сторон.
- 16) Найти объем шара V по радиусу r .
- 17) Найти объем куба V по ребру a .
- 18) Найти длину высоты в равностороннем треугольнике по длине стороны a .
- 19) Найти длину биссектрисы в равностороннем треугольнике по длине стороны a .
- 20) Найти площадь равностороннего треугольника S по длине стороны a .

1.11. Контрольные вопросы

1. Какие символы входят в алфавит языка C++?
2. Что такое идентификатор? По каким правилам он строится?
3. Каким образом выделяются комментарии в C++?
4. С какой функции начинается выполнение программы в C++?
5. Какие существуют простые стандартные типы данных в C++?
6. Назовите целый тип данных в C++ и четыре спецификатора. Опишите назначение спецификаторов.
7. Назовите логический тип данных, его возможные значения и объем памяти, выделяемый под значения данного типа.

8. Назовите символьные типы данных, опишите их.
9. Назовите вещественные типы данных, опишите их.
10. Для чего используется тип `void`?
11. Каким образом описываются переменные?
12. Каким образом описываются именованные константы?
13. Какие существуют арифметические операции?
14. Какие существуют логические операции?
15. Какие существуют операции отношения?
16. Какие существуют операции присваивания и как они работают?
17. Каким образом осуществляется консольный ввод-вывод данных с помощью функций `scanf`, `printf`?
18. Каким образом осуществляется консольный ввод-вывод данных с помощью потоков `cin`, `cout`?

Управление порядком выполнения инструкций программы.

Основные операторы

Оператор является основной конструкцией императивного языка программирования. Это единая и неделимая инструкция, выполняющая какое-либо действие. Любая инструкция заканчивается точкой с запятой (;).

Программа представляет собой набор операторов, управляющих ходом ее выполнения. В простейших случаях операторы выполняются последовательно но, как правило, этого недостаточно для решения реальных задач. Большинство задач требуют принятия решения в зависимости от различных ситуаций. Для этого в C++ используются специальные управляющие конструкции позволяющие менять порядок действий в зависимости от некоторых условий и повторять инструкции определенное количество раз.

Вообще, в C++ можно условно выделить четыре основных типа операторов [1]:

- условные операторы (**if**, **if-else** и **switch**) обеспечивают выполнение определенных блоков программы в зависимости от условий;
- операторы цикла (**while**, **do-while** и **for**), обеспечивают повторное выполнение блоков программы, в зависимости от условий;
- операторы перехода (**break**, **continue**, **return**, **goto**), позволяющие прервать ход выполнения программы и передать управление в другую ее часть;
- другие операторы (оператор «выражение», пустой оператор, составной оператор);

Ниже подробно рассмотрены все перечисленные операторы.

2.1. Выражения, блоки и пустые операторы

Самый простой тип оператора — *выражение*, за которым следует точка с запятой. Это может быть вычисление некоторого значения определенного типа, вызов функции или выполнение некоторого законченного действия.

Примеры выражений:

```
d = b*b - 4*a*c; cout << sin(x); x++;
```

Результаты следующих двух операторов различны! Префиксная и постфиксная формы операций инкремента и декремента различаются, если используются в составе некоторого выражения:

```
x = 5;  
y = (x++) * 2; //результат y=10;  
x = 5;  
y = (++x) * 2; //результат y=12;
```

C++ допускает несколько операций присваивания в одном операторе:

```
c = (a = 2) * b; //равносильно a=2; c=a*b;
```

Частным случаем оператора является *пустой оператор*, состоящий только из точки с запятой.

```
; //пустой оператор
```

Он используется там, где синтаксис C++ требует употребления оператора, а логика программы — нет. Например, пустой оператор может потребоваться в операторе ветвления **if**, когда по какой-либо ветви ничего не требуется выполнять, а иногда — для пустого тела цикла, т. к. операторы подобные **do**, **for**, **while** требуют наличия выполняемого оператора в теле.

Примеры использования пустого оператора будут приведены позже в этой главе.

Составной оператор или блок, дает возможность использовать несколько операторов в том месте, где предполагается использование одного, т. е. если некоторая последовательность действий должна выполняться как единое целое, но реализуется несколькими различными операторами, то они объединяются в блок — последовательность операторов, заключенных в фигурные скобки. Составной оператор воспринимается компилятором как один оператор и может использоваться всюду, где синтаксис требует одного оператора, а алгоритм — нескольких. Он может содержать один оператор или быть пустым. У блока собственная область видимости: объявленные внутри блока переменные доступны только внутри данного блока или блоков, вложенных в него.

2.2. Операторы ветвления

Эти операторы позволяют выполнять или пропускать те или иные блоки программы в зависимости от условий, меняя тем самым последовательный ход работы программы.

2.2.1. Условный оператор **if**

Условный оператор позволяет организовать выбор выполнения того или иного блока программы, т. е. ветвление алгоритма в двух направлениях, в зависимости от условия. Условие задается в виде выражения, результат вычисления которого должен иметь логическое значение.

Если требуется проверить несколько условий, их объединяют с помощью логических операций И (**&&**), ИЛИ (**||**).

При создании условных выражений следует помнить о том, что в C++ любое значение, отличное от нуля, воспринимается как истинное, а равное нулю — как ложное.

Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок, используя операторные скобки **{ }**. Блок может содержать любые операторы, в том числе описания и другие условные операторы, но не может состоять из одних описаний.

Оператор **if** имеет две формы: сокращенную и полную.

Сокращенная форма:

```
if( логическое выражение ) { блок операторов }
```

Если значение *логического выражения* равно **true**, то будет выполнен *блок операторов*, следующих за этим выражением, если значение равно **false**, то операторы выполнены не будут и управление будет передано оператору, следующему за оператором **if**.

Полная форма:

```
if( логическое выражение ) { блок операторов_1 };  
[ else { блок операторов_2 } ];
```

Сначала вычисляется *логическое выражение*, стоящее в скобках. Если результат вычислений имеет значение истина (**true**), выполняется *оператор* или *блок операторов_1* программы, стоящий после скобок. Если результатом является значение ложь (**false**), и присутствует ключевое слово **else**, то выполняется *оператор* или *блок операторов_2*, следующий за ключевым словом **else**. Перед словом **else** ставится точка с запятой! После этого управление передается оператору, следующему за оператором **if**.

В общем случае ключевое слово **else** связывается с ближайшим ключевым словом **if**, которое еще не связано с ключевым словом **else**.

Структурная схема оператора ветвления приведена на рисунке 2.1.

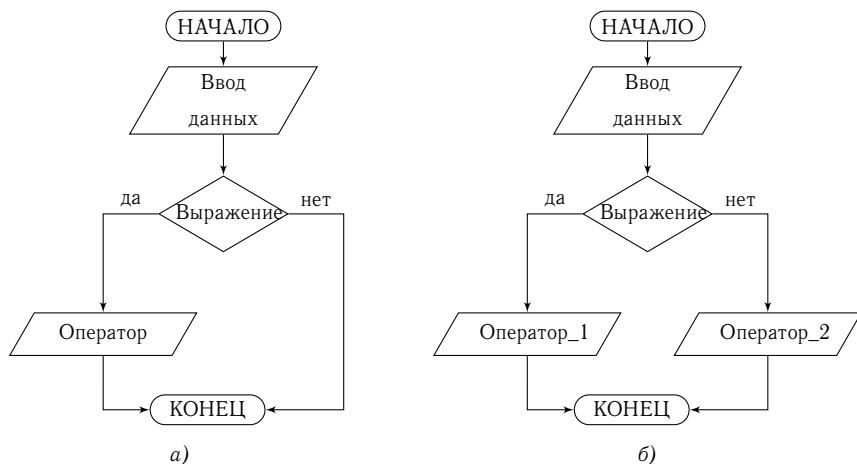


Рис. 2.1. Оператор ветвления, представленный в а) полной и б) неполной форме

Пример 2.1. Вывести на экран цифры числа, если оно является трехзначным.

Листинг 2.1.

```

1  ...
2  int x;
3  cout << "Введите число: ";
4  cin >> x;
5  if (x > 99 && x < 1000)
6      cout << x/100 << ' ' << (x%100)/10 << ' ' << x%10;
7  else
8      cout << "x не является трехзначным числом";

```


Результат работы программы:

x	$x/100$	$(x\%100)/10$	$x\%10$
10	x не является трехзначным числом		
251	2	5	1

□

В листинге 2.2, помимо вывода цифр трехзначного числа, добавлено нахождение суммы и произведения этих цифр. Таким образом, после проверки условия, необходимо выполнить блок операторов, заключенных в фигурные скобки:

Пример 2.2. Если число является трехзначным, вывести на экран его цифры, сумму и произведение этих цифр.

Листинг 2.2.

```
1 ...
2 int x, a, b, c;
3 cout << "Введите число: ";
4 cin >> x;
5 if (x > 99 && x < 1000){
6     a = x / 100; b = (x % 100) / 10; c = x % 10;
7     cout << "Цифры числа: " << a << ' ' << b << ' ' << c << endl;
8     cout << "Сумма цифр числа равна: " << a + b + c << endl;
9     cout << "Произведение цифр числа равно: " << a * b * c;
10 }
11 else
12     cout << "x не является трехзначным числом";
13 ...
```

Результат работы программы:

x	a	b	c	сумма	произведение
10	x не является трехзначным числом				
253	2	5	3	10	60

□

Пример 2.3. Ниже демонстрируется алгоритм, позволяющей определить в какой четверти координатной плоскости находится точка с заданными координатами. При условии, что $x \cdot y \neq 0$. Для решения используются вложенные условные операторы:

Результат работы программы:

```
float x,y;
cout<<"Введите два числа: ";
cin>>x>>y;
if (x*y!=0)
    if (x>0) if (y>0) cout<<"1 квадрант"; } 3 } 2 } 1
            else cout<<"4 квадрант"; } 3
    else     if (y>0) cout<<"2 квадрант"; } 3
            else cout<<"3 квадрант"; }
```

x	y	результат
5	9	1 квадрант
9	-7	4 квадрант
-9	7	2 квадрант
-9	-7	3 квадрант

□

В данном случае, в каждом условном операторе **if** выполняется только один оператор (также условный оператор **if**), поэтому ставить фигурные скобки не обязательно.

2.2.2. Примеры использования операторов ветвления для решения задач

Пример 2.4. В листинге 2.3 приведена программа для нахождения действительных корней биквадратного уравнения $ax^4 + bx^2 + c = 0$, где a, b, c — действительные числа и $a \neq 0$.

Биквадратное уравнение приводится к квадратному уравнению при помощи подстановки $y = x^2$.

После выяснения существования корней квадратного уравнения, путем подстановки находятся все возможные корни исходного биквадратного уравнения.

Листинг 2.3.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main() {
6     setlocale ( LC_ALL, "Russian" );
```

```

7  double a, b, c, D, y1, y2;
8  cout << "Введите коэффициенты уравнения:" << endl;
9  cout << "a= "; cin >> a;
10 cout << "b= "; cin >> b;
11 cout << "c= "; cin >> c;
12 D = b * b - 4 * a * c;
13 if (D > 0){
14     y1 = ( -b + sqrt (D)) / (2 * a);
15     y2 = ( -b - sqrt (D)) / (2 * a);
16     if (y1 > 0)
17         cout << "x1 = " << sqrt(y1) << "x2 = " << -sqrt(y1) << endl;
18     if (y2 > 0)
19         cout << "x3 = " << sqrt(y2) << "x4 = " << -sqrt(y2);
20 }
21 else
22     if (D == 0){
23         y1 = -b / (2 * a);
24         if (y1 > 0)
25             cout << "x1 = " << sqrt(y1) << "x2 = " << -sqrt(y1);
26         else
27             cout<<"Уравнение не имеет корней";
28     }
29     system("pause");
30     return 0;
31 }

```

Результат работы программы:

a	b	c	x_1	x_2	x_3	x_4
1	-5	6	2.449	-2.449	1.414	-1.414
1	5	6	1.414	-1.414		
1	-8	16	2	-2	$= x_1$	$= x_2$
1	1	4	Уравнение не имеет корней			

□

Для правильного вывода текста на кириллице в консольных приложениях, в строке 6 листинга 2.3, используется функция `setlocale()`, которая задает локализацию для текущей программы. Первый параметр — константа, говорящая какие категории нужно локализовать, второй параметр функции — идентификатор локали. Локаль содержит информацию о том, как интерпретировать и выполнять определен-

ные операции ввода/вывода и преобразования с учетом географического расположения и специфики языков в определенных условиях.

В листинге 2.4 приведена программа вычисления площади треугольника по его сторонам с помощью формулы Герона $s = \sqrt{p(p-a)(p-b)(p-c)}$, где p — полупериметр треугольника $p = \frac{a+b+c}{2}$. Прежде чем вычислять площадь, необходимо проверить принципиальную возможность существования треугольника с заданными сторонами. Делается это с помощью аксиомы неравенства треугольника: каждая сторона треугольника должна быть меньше суммы двух других сторон, т. е. $a < b + c$, $b < a + c$, $c < a + b$.

Пример 2.5. Вычислить площадь треугольника по его сторонам.

Листинг 2.4.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main() {
6     setlocale (LC_ALL,"Russian" );
7     double a, b, c, p;
8     cout << "Введите длины сторон треугольника : " << endl;
9     cout << "a= "; cin >> a;
10    cout << "b= "; cin >> b;
11    cout << "c= "; cin >> c;
12    if (a < b + c)
13        if (b < a + c)
14            if (c < a + b){
15                p = (a + b + c) / 2;
16                cout << "Площадь треугольника равна: " << sqrt(p * (p - a) * (p - b) * (p -
17                    c));
18            }
19            else
20                cout << "Треугольника с такими сторонами не существует!";
21            else
22                cout << "Треугольника с такими сторонами не существует!";
23            else
24                cout << "Треугольника с такими сторонами не существует!";
25    system("pause");
```

```
25  return 0;
26 }
```

<i>a</i>	<i>b</i>	<i>c</i>	<i>S</i>
3	4	5	6
4	4	4	6.928
4	8	2	Треугольник не существует

□

2.2.3. Оператор выбора switch

Иногда, в программе требуется проверить значение выражения или переменной на равенство одному из значений в списке и, в зависимости от результата, выполнить те или иные действия. Конечно, все эти проверки можно выполнить с помощью многократных конструкций `if-else`, но получившийся при этом код будет громоздким и сложным для восприятия.

Поэтому, для организации подобных вычислений в C++ имеется оператор множественного выбора `switch`, который проверяет совпадение значения переменной (или выражения) с каждым значением из списка констант символов или целых чисел и, в случае совпадения, выполняет оператор или блок операторов, ассоциированный с данным значением.

Оператор `switch` имеет следующий формат:

```
switch (⟨выражение⟩) {
    case ⟨константное_выражение_1⟩ : [ ⟨блок_операторов_1⟩ ; break ; ]
    case ⟨константное_выражение_2⟩ : [ ⟨блок_операторов_2⟩ ; break ; ]
    ...
    case ⟨константное_выражение_N⟩ : [ ⟨блок_операторов_N⟩ ; break ; ]
    [ default : ⟨операторы⟩ ]
}
```

Структура оператора `switch` состоит из ряда меток `case` (выбор) и необязательной метки `default` (умолчание).

Оператор работает следующим образом: сначала вычисляется значение выражения, затем это значение последовательно сравнивается со значениями констант, стоящих после меток `case`, и, в случае совпадения с одним из значений, выполняются оператор или блок операторов стоящих после двоеточия.

Если значение выражения или переменной не совпадает ни с одной из констант, и оператор `switch` содержит метку `default`, то выполняются инструкции, стоящие после нее. Бывают ситуации, в которых никакой обработки по метке `default` не требуется и при ее отсутствии управление передается следующему за `switch` оператору.

Оператор `break` необходим для того, чтобы выполнить выход из оператора `switch`. Если он не указан, то будут выполняться все последующие операторы из списка, несмотря на то, что значение, которым они помечены, не совпадает со значением выражения. Это происходит потому, что в C++ если один вариант является истинным, то и все последующие варианты рассматриваются как истинные. При этом оператор `break` выполняет выход из самого внутреннего из объемлющих его операторов `switch`.

В операторе `switch` для проверки на совпадение могут применяться только константные выражения целого типа, т.е. любая комбинация символьных и целочисленных констант, которая имеет целое постоянное значение. Символьная константа представляется как соответствующий символ, заключенный в одиночные кавычки, например, `'A'`. Целая константа — просто целое число. Все константные выражения должны иметь разные значения, но быть одного и того же типа.

Если нужно выполнить одно и то же действие или блок операторов для нескольких меток, то они могут следовать друг за другом через двоеточие. Это означает, что операторы, помеченные этим списком констант будут выполняться при совпадении значения выражения с любым из значений констант из этого списка.

Оператор `switch` в C++ не поддерживает диапазоны.

Структурная схема оператора приведена на рисунке 2.2

Пример 2.6. Следующий фрагмент программы демонстрирует работу оператора `switch` для программирования упрощенного арифметического калькулятора:

Листинг 2.5.

```
1 ...
2 char sign;
3 double x, y;
4 cout << "Введите операнды" << endl;
5 cout << "x= "; cin >> x;
6 cout << "y= "; cin >> y;
7 cout << "Введите знак операции"; cin >> sign;
```

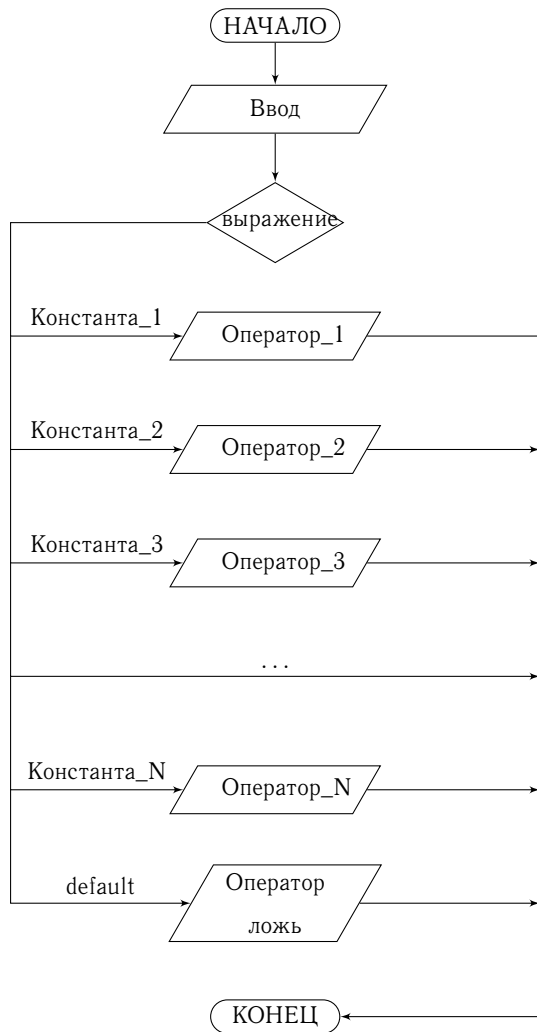


Рис. 2.2. Структурная схема оператора switch

```

8  switch (sign){
9    case '+': cout << "x+y= " << x + y; break;
10   case '-': cout << "x-y= " << x - y; break;
11   case '*': cout << "x*y= " << x * y; break;
12   case '/': cout << "x/y= " << x / y; break;
13   case '^': cout << "x^y= " << pow(x,y); break;
14   default : cout << "Такая операция не предусмотрена";
15 }

```

x	y	знак	результат
1	5	*	5
2	5	+	7
1	3	!	Такая операция не предусмотрена

□

Если в листинге 2.5 убрать операторы `break`, то результат работы будет следующий:

```

x=0.5
y=2
Введите знак операции *
x*y= 1x/y= 0.25x^y=0.25Такая операция не предусмотрена

```

Следующий пример демонстрирует случай когда одно и тоже действие выполняется для нескольких меток.

Пример 2.7. Дан порядковый номер месяца. Вывести на экран сколько в нем дней.

Листинг 2.6.

```

1  ...
2  int x;
3  cout << "Введите номер месяца "; cin >> x;
4  switch (x){
5  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
6    cout << " в этом месяце 31 день"; break;
7  case 4: case 9: case 11:
8    cout << " в этом месяце 30 день"; break;
9  case 2:
10   cout << "Если год високосный, то 29 дней, если нет, то 28 дней "; break;
11  default: cout << "Такого месяца нет";
12 }

```

x	результат
1	в этом месяце 31 день
8	в этом месяце 30 дней
13	Такого месяца нет

□

2.2.4. Пример использования оператора выбора для решения задач

В листинге 2.7 приведена программа, позволяющая в зависимости от сделанного пользователем выбора (введенного значения), запросить необходимые данные и найти общий член и сумму первых n членов арифметической или геометрической прогрессий.

Пример 2.8. Найти сумму первых n членов арифметической или геометрической прогрессии.

Листинг 2.7.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main() {
6     setlocale ( LC_ALL, "Russian" );
7     int pr;
8     cout << "Для нахождения суммы арифметической прогрессии введите 1" <<
9         endl;
10    cout << "Для нахождения суммы геометрической прогрессии введите 2" <<
11        endl;
12    cin >> pr;
13    int n;
14    switch (pr){
15    case 1:
16        float a1, d, an;
17        cout << "Введите значение первого члена прогрессии ";
18        cin >> a1;
19        cout << "Введите значение разности прогрессии ";
20        cin >> d;
21        cout << "Введите количество членов прогрессии ";
22        cin >> n;
23        an = a1 + (n - 1) * d;
24        cout << "Общий член арифметической прогрессии находится по формуле: "
25            << endl;
26        cout << "an=a1+(n-1)*d и равен " << an << endl;
27        cout << "Сумма первых n членов прогрессии находится по формуле: " <<
28            endl;
29        cout << "sn=((a1+an)/2)*n и равна " << ((a1 + an) / 2) * n << endl;
```

```

26     break;
27 case 2:
28     float b1, q, bn;
29     cout << "Введите значение первого члена прогрессии b1<>0 ";
30     cin >> b1;
31     if (!b1){
32         cout << "Значение не должно быть равным нулю! ";
33         goto metka;
34     }
35     cout << "Введите значение знаменателя прогрессии q<>0 ";
36     cin >> q;
37     if (!q){
38         cout << "Значение не должно быть равным нулю! ";
39         goto metka;
40     }
41     cout << "Введите количество членов прогрессии ";
42     cin >> n;
43     bn = b1 * pow(q, n - 1);
44     cout << "Общий член геометрической прогрессии находится по формуле: "
45         << endl;
46     cout << "bn=b1*pow(q,n-1) и равен " << bn << endl;
47     cout << "Сумма первых n членов прогрессии находится по формуле: " <<
48         endl;
49     cout << "sn=(b1*(pow(q,n)-1))/(q-1), если q не равно 1 и " << endl;
50     cout << "sn=n*b1, если q=1 и равна ";
51     if (q == 1)
52         cout << n * b1;
53     else
54         cout << (b1 * (pow(q, n) - 1)) / (q - 1);
55     cout << endl;
56     break;
57 metka: default : cout << "Вы ввели ошибочные данные!";
58 }
59 system("pause");
60 return 0;
61 }

```

Вид прогрессии	a_1	$d(q)$	n	a_n	b_n
1	5	2	4	11	32
2	2	2	4	16	30
2	2	0	Вы ввели ошибочные данные		

□

В 33 и 39 строках программы используется оператор безусловного перехода `goto`, позволяющий прервать выполнение кода и передать управление в другое место программы, помеченное меткой. Таким образом, выполняется проверка введенных значений, и если они введены неверно, то выполнение операторов прерывается и управление передается на строку 55, на помеченный меткой оператор `default`, сообщающий об ошибке!

2.3. Операторы цикла

В ситуации, когда необходимо многократно повторить какое-то действие или их некоторую последовательность, незаменимыми являются операторы цикла, которые позволяют выполнять один или несколько операторов определенное число раз в зависимости от некоторых условий. Любой цикл имеет точку входа, проверочное условие и (необязательно) точку выхода. Цикл, не имеющий точки выхода, называется бесконечным. Для бесконечного цикла проверочное условие всегда принимает истинное значение.

2.3.1. Оператор цикла `while`

В C++ цикл `while` обычно используется в тех случаях, когда число повторений цикла заранее неизвестно. Поэтому, в зависимости от начального условия, цикл может выполниться несколько раз или не выполниться вообще.

Цикл `while` называется циклом *с предусловием* и имеет следующий формат:

```
while (⟨логическое выражение⟩) { ⟨блок_операторов⟩; }
```

Блок операторов, выполняемый внутри цикла называют — телом цикла.

Схема выполнения оператора `while` выглядит следующим образом: сначала вычисляется *логическое выражение*, стоящее в скобках, затем, если выражение ложно, то выполнение оператора `while` прекращается и выполняется следующий по по-

рядку оператор, идущий за циклом. Если выражение истинно, то выполняется тело оператора `while`.

Оператор цикла выполняется повторно до тех пор, пока логическое выражение не примет значение ноль («ложь»). Следовательно, во избежании «зацикливания» (бесконечного выполнения цикла) и зависания программы необходимо предусмотреть в теле цикла изменение операндов логического выражения, для того чтобы условие выполнения цикла `while` в итоге стало ложным!

Ниже приведена структурная схема оператора:

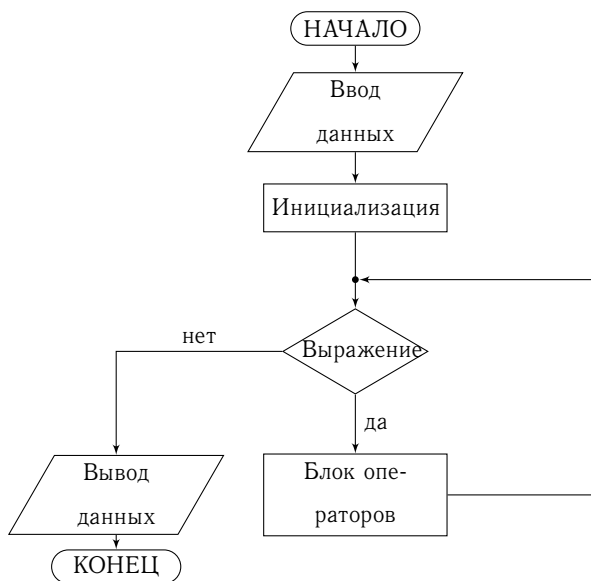


Рис. 2.3. Структурная схема оператора цикла с предусловием

Пример 2.9. Дано целое число $N > 0$. Найти сумму его цифр.

Листинг 2.8.

```
1 ...  
2 int N, s = 0;  
3 cin >> N;  
4 while (N){  
5     s += N % 10;  
6     N /= 10;  
7 }
```

```
8 cout << s;
```

Результат работы программы при $N = 12345$:

N	$N \% 10$	$N / 10$	s
12345	5	1234	5
1234	4	123	9
123	3	12	12
12	2	1	14
1	1	0	15

В цикле `while` последовательно выделяются и суммируются цифры исходного числа, начиная с младшего разряда. Для выделения цифры в младшем разряде применяется операция взятия остатка от деления числа N на 10 (операция `\%`), затем эта цифра прибавляется к сумме и отбрасывается от исходного числа.

При делении двух целых чисел дробная часть отбрасывается, поэтому после операции $N /= 10$ получается новое число, равное исходному без младшей цифры. Так происходит до тех пор пока, наконец, число не станет равным нулю. □

Пример 2.10. Дано целое число $N > 0$. С помощью операций деления нацело и взятия остатка от деления определить, имеются ли в записи числа N нечетные цифры.

Листинг 2.9.

```
1 ...
2 int N;
3 cin >> N;
4 while (N && !(N % 2))
5     N /= 10;
6 if (N)
7     cout << "Нечетные цифры в числе есть";
8 else
9     cout << "Нечетных цифр в числе нет";
```

В ходе работы программы текущее число каждый раз проверяется на нечетность. Если число нечетное (остаток от деления на два больше нуля), значит в его записи должна быть нечетная цифра и цикл завершает свою работу т. к. второе условие в логическом выражении становится ложным.

Если текущее число четное, то оно уменьшается путем отбрасывания цифры в младшем разряде за счет операции $N /= 10$ и снова проверяется на нечетность.

Если N становится равным нулю — это означает, что были просмотрены все разряды исходного числа и оно не содержит нечетных цифр, и цикл заканчивает свою работу, т. к. первое условие в логическом выражении ложно. \square

Пример 2.11. Пример использования цикла `while`, приводящий к заикливанию программы.

Листинг 2.10.

```
1 ...
2 int i = 1, s = 0;
3 while (i > 0)
4     s += i;
```

Переменная `i`, используемая в условии цикла не меняет своего значения и остается равной единице, что приводит к заикливанию программы. \square

Следующий фрагмент программы содержит цикл `while` с изначально ложным значением логического выражения в условии. Следовательно цикл не выполнится ни разу.

Пример 2.12. Необходимо найти сумму целых чисел меньших числа `a`.

Листинг 2.11.

```
1 ...
2 int s = 0, a;
3 cin >> a;
4 i = a;
5 while (i < a){
6     s += i;
7     i++;
8 }
9 cout << s;
```

\square

Так как перед началом работы цикла переменной `i` присвоено значение `a`, то цикл не сработает ни разу и сумма будет равна нулю. Это происходит потому что условие выполнения цикла проверяется в начале, в отличие от цикла с *постусловием* о котором речь пойдет далее.

2.3.2. Оператор цикла `do while`

Цикл `do while` также используется в тех случаях, когда число повторений цикла заранее неизвестно, но, в отличие от цикла `while` условие завершения цикла проверяется не в начале, а в конце после выполнения тела цикла. Поэтому, независимо от значения логического выражения цикл будет выполняться хотя бы один раз! Если условие, проверяемое в конце, будет истинным, то цикл продолжит свою работу, если ложным, закончит свое выполнение.

Цикл `do while` называется циклом с постусловием и имеет следующий формат:

```
do {⟨блок_операторов;⟩ } while (⟨логическое выражение⟩)
```

Схема выполнения оператора `do while` выглядит следующим образом: сначала выполняется тело оператора, затем вычисляется логическое выражение, стоящее в скобках. Если выражение ложно, то выполнение оператора `do while` прекращается и выполняется следующий по порядку оператор, идущий за циклом. Если выражение истинно, то тело цикла выполняется еще раз.

Оператор цикла выполняется повторно до тех пор, пока логическое выражение не примет значение ноль («ложь»). Аналогично циклу `while` необходимо предусматривать в теле цикла изменение операндов логического выражения, для того чтобы условие выполнения цикла `do while` в итоге стало ложным и не произошло заикливание.

Ниже приведена структурная схема оператора:

Рассмотрим вариант применения цикла `do while` для решения предыдущей задачи нахождения суммы целых чисел меньших числа `a`. Так как цикл `do while` выполняется один раз всегда, независимо от условия, то в результате сумма будет равна числу `a`, а не нулю как в примере 2.12.

Пример 2.13. Найти сумму целых чисел меньших числа `a`.

Листинг 2.12.

```
1 ...
2 int s = 0, a;
3 cin >> a;
4 i = a;
5 do{
```

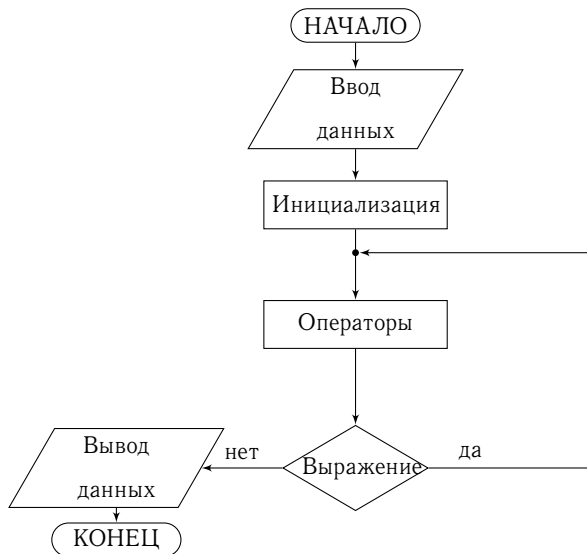


Рис. 2.4. Оператор цикла с постусловием

Рис. 2.5. Структурная схема оператора цикла с постусловием

```

6   s += i;
7   i++;
8 }
9 while (i < a);
10 cout << s;

```

□

Цикл `do while` часто имеет смысл использовать, если программа должна сначала запрашивать у пользователя данные, а затем их проверять.

Например, в следующем цикле суммируются вводимые пользователем числа и, каждый раз, запрашивается нужно ли продолжать выполнения программы:

Пример 2.14. Найти сумму чисел, вводимых пользователем.

Листинг 2.13.

```

1 ...
2 float x,s = 0;
3 char answer;
4 do{

```



```

5  cout << "Введите очередное суммируемое число";
6  cin >> x;
7  s += x;
8  cout << "Нужно ли продолжать ввод? Y / N";
9  cin>>answer;
10 } while (answer != 'N');
11 cout << s;

```

Результат работы программы:

answer	<i>x</i>	<i>s</i>
Y	5	5
Y	10	15
N		

□

2.3.3. Оператор цикла for

Оператор **for** это наиболее общий способ организации цикла. Его называют циклом с параметром.

Цикл **for** имеет следующий формат:

```

for ( ( < инициализация; логическое выражение; блок_модификаций> )
      { <блок_операторов>; }

```

Инициализация, проверка логического выражения и изменение параметров составляют три части управляющего раздела, заключенного в круглые скобки. Каждая часть является выражением, отделяемым от других частей точкой с запятой. Оператор или блок операторов, заключенный в фигурные скобки, следующий за управляющим разделом, выполняется до тех пор, пока логическое выражение остается истинным.

Любой из блоков цикла **for** может отсутствовать, но точку с запятой, обозначающую позицию блока необходимо ставить.

Схема выполнения оператора **for** выглядит следующим образом: сначала выполняется блок инициализации, используемый для объявления и присвоения начальных значений переменным цикла. В этом блоке можно инициализировать несколько параметров, разделяя их запятой. Если переменные объявлены в блоке инициализации, то область их видимости цикл и вложенные блоки. Инициализация выполняется один раз в начале работы цикла. После этого вычисляется логическое выражение и,

если полученный результат принял истинное значение, выполняется тело цикла. В противном случае выполнение цикла прекращается и осуществляется переход к оператору, следующему непосредственно за телом цикла. Эта проверка осуществляется перед каждой новой итерацией. Уже после выполнения операторов тела цикла выполняется блок модификаций, который обычно служит для изменения параметров цикла. Затем снова осуществляется переход к вычислению логического выражения. Ниже приведена структурная схема оператора:



Рис. 2.6. Структурная схема цикла for

Пример 2.15. Найти сумму четных чисел из диапазона от **a** до **b**.

Листинг 2.14.

```

1 ...
2 int a, b, s = 0;
3 cout << "Введите границы диапазона a и b" << endl;
4 cout << "a= "; cin >> a;
5 cout << "b= "; cin >> b;
6 for(int i = a; i <= b; i++)
7     if (i % 2 == 0) s += i;
8 cout << s;
  
```

Результат работы программы при $a = 5$ до $b = 10$:

i	строка 7	s
5	false	0
6	true	6
7	false	6
8	true	14
9	false	14
10	true	24

В блоке инициализации объявляется переменная i , область ее видимости ограничена циклом. Ей присваивается начальное значение интервала. Затем задается условие, при истинности которого, цикл выполняется. Если пользователь введет значение для переменной a большее чем для b , цикл не сработает ни разу. В этом смысле цикл **for** является циклом с предусловием.

Переменная i меняет свое значение при каждой итерации увеличиваясь на единицу. В теле цикла, на каждой итерации, для очередного значения i проверяется четность, путем взятия остатка от деления на два и в случае если значение i четно, оно прибавляется к сумме. □

В некоторых случаях бывает удобно сознательно создать бесконечный цикл, т. е. чтобы условие выполнения цикла всегда было истинным. Для управления такой структурой используется оператор **break**, позволяющий прекратить выполнение цикла.

Бесконечный цикл можно задать для любого из трех операторов:

```
for (; ;)  
    { /*действия; ...*/break;... }
```

...

```
while (1){  
    /*действия; ...*/break;...  
}  
...  
do {  
    /*действия; ...*/break;...  
} while(1)
```

Пример 2.16. Пример цикла, который будет работать до тех пор, пока на клавиатуре не будет набрана буква **A**.

Листинг 2.15.

```
1 ...
2 char ch;
3 for ( ; ; ){
4     cin >> ch;
5     if (ch == 'A') break; // выход из цикла
6 }
7 cout << "Вы ввели букву А";
```

□

Любые типы циклов `for`, `while`, `do while` могут быть вложенными друг в друга.

Например, с использованием вложенных циклов нужно вывести на экран следующую таблицу значений:

```
1  2  3  4  5
2  4  6  8 10
3  6  9 12 15
4  8 12 16 20
5 10 15 20 25
```

```
for(int i=1; i<=5; i++,cout<<endl)
    for(int j=1; j<=5; j++)
        cout<<setw(3)<<right<<i*j; }
```

} Тело внешнего цикла

} Тело внутреннего цикла

Во внешнем цикле определяется количество строк выводимых на экран. В данном случае их пять. В блоке модификаций помимо приращения параметра `i`, выполняется перевод выходного потока на новую строку. Количество итераций внутреннего цикла определяет сколько чисел выводить в каждой строке, а в теле внутреннего цикла выводится нужное значение с использованием манипуляторов форматирования, которые встраиваются непосредственно в операторы `cin` и `cout` и позволяют управлять отображением данных на экране: `setw(n)` — устанавливает максимальную ширину поля вывода, `right` — выравнивает вывод по правому краю. Для использования этих манипуляторов нужно подключить заголовочный файл `iomanip`.

2.3.4. Примеры использования операторов цикла для решения задач

Пример 2.17. Найти все трехзначные числа, в старшем разряде которых стоит четная цифра, а в младшем нечетная.

Задача решается с использованием всех трех видов циклов и для удобства предусмотрена возможность выбора вида цикла с помощью оператора `switch`.

Листинг 2.16.

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main(){
6     setlocale ( LC_ALL,"Russian" );
7     char a;
8     int i, b;
9     do{
10         cout << "Выберите вид цикла, которым следует решить задачу" << endl;
11         cout << "w - while; d - do while; f - for" << endl;
12         cin >> a;
13         i = 201;
14         switch (a){
15             case 'w':
16                 while ( i < 1000){
17                     if ( ( i % 10 % 2) && ( !(i / 100 % 2) ) )
18                         cout << i << ' ';
19                     i++;
20                 }
21                 break;
22             case 'd':
23                 do{
24                     if ( ( i % 10 % 2) && ( !(i / 100 % 2) ) )
25                         cout << i << ' ';
26                     i++;
27                 }
28                 while (i < 1000);
29                 break;
30             case 'f':
31                 for (; i < 1000; i++)
```

```

32         if ( ( i % 10 % 2) && ( !(i / 100 % 2) )
33             cout << i << '  ';
34         break;
35     default: cout << "Вы ошиблись";
36     }
37     cout << "решить другим способом? 1 - да, 0 - нет" << endl;
38     cin >> b;
39 }
40 while (b);
41 system("pause");
42 }

```

□

Пример 2.18. Вывести на экран следующую таблицу, используя вложенные циклы:

```

1
1
2 2
2 2
3 3 3
3 3 3
4 4 4 4
4 4 4 4
5 5 5 5 5
5 5 5 5 5

```

Листинг 2.17.

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main(){
6     int i, j;
7     for (i = 1; i <= 5; i++, cout << endl){
8         for (j = 1; j <= i; j++)
9             cout << i << '  ';
10        cout << endl;
11        for(j = 1; j <= i; j++)
12            cout << i << '  ';

```

```

13     cout << endl;
14 }
15 system ("pause");
16 }

```

□

Пример 2.19. Вычислить значение заданной функции в точках заданного отрезка $x \in [a, b]$ с указанным шагом h . Результат выводится на экран в виде таблицы. Если в некоторых точках заданного интервала функция не определена, то в соответствующей графе таблицы выводится сообщение: «функция не определена».

$$F(x) = \begin{cases} \sqrt{x^2 - 1}, & \text{при } x < 1.5, \\ x^2, & \text{при } x \geq 1.5, \end{cases} \quad x \in [0.5; 3], h = 0.1$$

Листинг 2.18.

```

1  #include <iostream>
2  #include <cmath>
3  #include <iomanip>
4  using namespace std;
5
6  int main() {
7      setlocale( LC_ALL, "Russian" );
8      double a, b, h, x;
9      cout << "Введите границы отрезка" << endl;
10     cout << "a= "; cin >> a;
11     cout << "b= "; cin >> b;
12     cout << "Введите шаг" << endl;
13     cout << "h= "; cin >> h;
14     int i;
15     for( i = 1, x = a; x <= b; x += h, i++, cout << endl){
16         cout << i;
17         cout << setw(25) << right << x;
18         if (x >= 1.5)
19             cout << setw(25) << right << x * x << '  ';
20         else if (abs(x) < 1)
21             cout << setw(25) << right << "Функция не определена";
22         else
23             cout << setw(25) << right << sqrt( (x * x) - 1.0);
24     }

```

25 system ("pause");

26 }

□

2.4. Упражнения

Номера и количество заданий, а также форма отчетности определяется преподавателем.

I. Необходимо проверить выполнение того или иного условия, и если выполнение невозможно, выдать сообщение об этом. Задания содержат материал из школьного курса геометрии, в частности, необходимо знание свойств простейших геометрических фигур, операций с векторами.

- 1) Проверить является ли треугольник с заданными сторонами a, b, c равнобедренным. Если да, то найти высоту (медиану и биссектрису) этого треугольника, опущенную на основание.
- 2) Найти радиус вписанной окружности в треугольник со сторонами a, b, c . Необходимо проверить существование треугольника с заданными сторонами.
- 3) Найти радиус описанной окружности около треугольника со сторонами a, b, c . Необходимо проверить существование треугольника с заданными сторонами.
- 4) Даны координаты трех точек на плоскости (x_i, y_i) , $i = 1..3$. Определить какая из точек наиболее удалена от начала координат.
- 5) Даны координаты трех точек на плоскости (x_i, y_i) , $i = 1..3$. Определить какая из точек наименее удалена от начала координат.
- 6) Даны три точки на плоскости: A, B и C , заданные своими координатами. Нужно выяснить, лежат ли они на одной прямой.
- 7) Даны две прямые на плоскости, заданные уравнениями $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$. Проверить совпадают ли эти прямые.
- 8) Даны две прямые на плоскости, заданные уравнениями $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$. Проверить параллельны ли эти прямые.

- 9) Даны две прямые на плоскости, заданные уравнениями $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$. Проверить пересекаются ли они.
- 10) Дана прямая на плоскости, заданная уравнением $Ax + By + C = 0$ и точка $M(x, y)$. Определить взаимное расположение точки и прямой: точка лежит слева от прямой, на прямой, справа от прямой.
- 11) Вычислить площадь треугольника, заданного координатами его вершин. Прежде чем вычислять площадь, необходимо проверить, можно ли построить треугольник по заданным точкам.
- 12) Заданы два вектора на плоскости $\vec{a}(x_1, y_1)$ $\vec{b}(x_2, y_2)$. Выяснить какой угол они образуют (тупой, острый или они ортогональны).
- 13) Прямая задана координатами двух точек $P_1(x_1, y_1), P_2(x_2, y_2)$. Определить взаимное расположение точки $M(x, y)$ и прямой: точка лежит слева от прямой, на прямой, справа от прямой.
- 14) Определить принадлежит ли точка $M(x, y)$ отрезку с координатами $P_1(x_1, y_1), P_2(x_2, y_2)$.
- 15) Прямая задана координатами двух точек $P_1(x_1, y_1), P_2(x_2, y_2)$. Определить взаимное расположение двух точек $M(x_3, y_3)$ и $N(x_4, y_4)$ относительно прямой.
- 16) Прямая и окружность могут иметь одну или две точки пересечения или не пересекаться. Даны окружность с центром $O(x_1, y_1)$ и радиусом R и прямая, заданная координатами двух точек $P_1(x_2, y_2), P_2(x_3, y_3)$. Определить количество общих точек прямой и окружности.
- 17) Даны две прямые на плоскости, заданные уравнениями $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$. Найти ориентированный угол между ними, в случае если они не перпендикулярны.
- 18) Выяснить, будут ли перпендикулярными отрезки KL и MN , если $K(x_1, y_1), L(x_2, y_2), M(x_3, y_3)$ и $N(x_4, y_4)$.
- 19) Два отрезка заданы своими координатами. Выяснить, пересекаются ли эти отрезки, не находя точку пересечения.
- 20) Проверить, образуют ли базис два вектора плоскости $\vec{a}(x_1, y_1)$ $\vec{b}(x_2, y_2)$.
- 21) Даны две прямые на плоскости, заданные уравнениями с угловыми коэффициентами $y_1 = k_1x + b_1$ и $y_2 = k_2x + b_2$. Найти ориентированный угол между ними, в случае если они не перпендикулярны.
- 22) Проверить коллинеарны ли два вектора плоскости $\vec{a}(x_1, y_1), \vec{b}(x_2, y_2)$.

- 23) Четырехугольник задан координатами своих вершин $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3), D(x_4, y_4)$. Проверить является ли он параллелограммом.
- 24) Четырехугольник задан координатами своих вершин $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3), D(x_4, y_4)$. Проверить является ли он трапецией.
- 25) Проверить, является ли четырехугольник, заданный координатами вершин $(x_i, y_i), i = 1..4$ выпуклым или нет.

II. Для решения всех заданий, предполагается использование оператора выбора **switch**.

- 1) Задан порядковый номер дня недели. Вывести на экран его название и количество дней оставшихся до конца недели.
- 2) Создайте меню для выбора материков в виде пронумерованного списка. В зависимости от выбора пользователя, необходимо выводить на экран площадь материка и сведения о его расположении.
- 3) Задан порядковый номер месяца. Вывести на экран номер соответствующего квартала.
- 4) Задана дата в формате: число, номер месяца, год. Вывести на экран дату в формате: число, название месяца, год.
- 5) Создайте меню для выбора названия кафедр в виде пронумерованного списка. В зависимости от выбора пользователя, необходимо выводить на экран название факультета, которому принадлежит кафедра.
- 6) Задан номер дня недели. Вывести на экран его название и расписание группы на этот день.
- 7) Задана буква русского алфавита. Определить является она гласной, согласной или разделительным знаком.
- 8) В зависимости от заданного номера месяца вывести на экран время года.
- 9) Задан порядковый номер планеты солнечной системы в зависимости от удаленности от солнца (1 – 8). Вывести на экран название планеты и расстояние от солнца.
- 10) Задан номер дня недели. Вывести на экран его название на английском, немецком и французском языках.
- 11) Определить достоинство и масть карты по введенным номеру карты и номеру масти соответственно. (Номер масти 1 – «пики», 2 – «трефы», 3 – «бубны», 4 – «червы». Достоинство карты 14 – «туз», 13 – «король», 12 – «дама», 11 – «валет», 10 – «десятка» и т. д. до «шестерки»).

- 12) Задан символ. Вывести на экран является ли он буквой, цифрой, знаком препинания, знаком арифметической операции, скобкой или служебным символом.
- 13) Задан порядковый номер дня недели и месяца. Вывести на экран название дня недели и месяца на английском языке.
- 14) Дано целое число в диапазоне 10 – 99. Вывести число прописью, например: 25 — «двадцать пять».
- 15) Написать программу, осуществляющую перевод рублевых денежных средств в основные валюты (EUR, USD, GBP, JPY, CHF) по текущему курсу. Входными данными являются код валюты и сумма денежных средств в рублях.
- 16) Создайте меню для выбора геометрических фигур (круг, прямоугольник, квадрат, ромб, треугольник) в виде пронумерованного списка. В зависимости от выбора пользователя, необходимо выводить на экран формулы периметра и площади соответствующих фигур.
- 17) Создать меню для выбора тригонометрических функций ($\sin(x)$, $\cos(x)$, $\operatorname{tg}(x)$, $\operatorname{ctg}(x)$, $\operatorname{sec}(x)$) в виде пронумерованного списка. Пользователь задает значение аргумента и выбирает из списка нужную функцию. Необходимо вывести на экран значение функции.
- 18) Написать программу, вычисляющую стоимость междугородного телефонного разговора. Входными данными являются код города и длительность разговора. Цена одной минуты разговора зависит от города, куда звонит абонент.
- 19) Написать программу перевода сантиметров в различные национальные системы измерений длины (дюйм, фут, ярд, хенд). Входными данными являются количество сантиметров и код национальной системы измерений.
- 20) Задан год. Вывести на экран название животного, символизирующего этот год по восточному календарю начиная с 1924 года по 2063.
- 21) Написать программу перевода литров в различные национальные системы измерений объема (пинта, галлон, баррель (нефтяной), кварта). Входными данными являются количество литров и код национальной системы измерений.
- 22) Написать программу перевода градусов Цельсия в градусы Фаренгейта, Реомюра, Кельвина, Ранкина. Входными данными являются количество градусов Цельсия и код единицы измерения температуры.

- 23) Написать программу перевода граммов в тройскую систему мер весов (фунт, унция, пеннивейт, карат). Входными данными являются количество граммов и код тройской системы измерений.
- 24) Написать программу перевода ар(соток) в различные национальные системы измерений площади (акр, перч, квадратный ярд, квадратный фут). Входными данными являются количество соток и код национальной системы измерений.
- 25) Создайте меню для выбора типа треугольника (равносторонний, равнобедренный, прямоугольный) в виде пронумерованного списка. В зависимости от выбора пользователя, необходимо выводить на экран основные свойства (такие как высота, медиана, биссектриса, площадь) для данного типа треугольника.

III. Каждая задача должна быть решена тремя способами, с использованием всех трех типов циклов!

- 1) Составьте программу, выводящую на экран квадраты целых чисел от a до b .
- 2) Составьте программу, выводящую на экран все четные числа из диапазона от a до b по убыванию.
- 3) Составьте программу, вычисляющую сумму квадратов всех нечетных чисел от 1 до N .
- 4) Составьте программу, выводящую на экран все целые числа оканчивающиеся на цифру X из диапазона от a до b .
- 5) Составьте программу, выводящую на экран отрицательные четные числа оканчивающиеся на цифру X из диапазона от a до b .
- 6) Составьте программу, выводящую на экран все числа кратные пяти от 1 до N .
- 7) Составьте программу, выводящую на экран все трехзначные числа которые начинаются и заканчиваются на одну и ту же цифру.
- 8) Составьте программу, выводящую на экран все четные числа кратные трем из диапазона от a до b .
- 9) Составьте программу, вычисляющую сумму кубов всех четных чисел от 1 до N .
- 10) Составьте программу, вычисляющую сумму чисел кратных пяти в интервале от a до b .

- 11) Составьте программу, выводящую на экран все нечетные числа кратные пяти из диапазона от a до b по убыванию.
- 12) Составьте программу, выводящую на экран все двухзначные числа, в записи которых цифры разные.
- 13) Составьте программу, выводящую на экран все трехзначные числа в записи которых любые две цифры различны.
- 14) Составьте программу, вычисляющую сумму отрицательных чисел кратных пяти в интервале от a до b .
- 15) Составьте программу, выводящую на экран все четные двухзначные числа у которых разница между старшей и младшей цифрой равна пяти.
- 16) Составьте программу, выводящую на экран все четырехзначные числа сумма цифр младших разрядов которых равна сумме цифр в старших разрядах.
- 17) Дано натуральное число n , действительные числа a_1, a_2, \dots, a_n , вводимые с клавиатуры. Составьте программу, вычисляющую следующие суммы: $a_1, a_1 + a_2, \dots, a_1 + a_2 + \dots + a_n$.
- 18) Вычислить значение многочлена $x^4 - 12x^2 + 5x - 17$ для $x = 0, 1, \dots, n$.
- 19) Дано натуральное число n , действительные числа a_1, a_2, \dots, a_n , вводимые с клавиатуры. Составьте программу, вычисляющую следующие суммы: $|a_1|, |a_1 + a_2|, \dots, |a_1 + a_2 + \dots + a_n|$.
- 20) Составьте программу, выводящую на экран все трехзначные числа у которых сумма младшей и старшей цифр равна средней цифре.
- 21) Составьте программу, выводящую на экран все трехзначные числа у которых произведение младшей и старшей цифр равно средней цифре.
- 22) Составьте программу, выводящую на экран суммы цифр всех трехзначных чисел.
- 23) Составьте программу, выводящую на экран произведение цифр всех трехзначных чисел.
- 24) Составьте программу, выводящую на экран все двузначные числа в записи которых обе цифры четные.
- 25) Составьте программу, выводящую на экран все трехзначные числа в записи которых все цифры различны.

IV. Вывести на экран данные в следующем виде:

	-5	-4	...	4	5		0.2	0.4	0.6	0.8	1	1					
	-5	-4	...	4	5		0.3	0.5	0.7	0.9	1.1	2	2				
1)	-5	-4	...	4	5	2)	0.4	0.6	0.8	1	1.2	3	3	3			
	-5	-4	...	4	5		0.5	0.7	0.9	1.1	1.3	4	4	4	4		
	-5	-4	...	4	5		0.6	0.8	1	1.2	1.4	5	5	5	5	5	
	1						0	0	0	0	0	<i>a</i>	<i>c</i>	<i>e</i>	<i>g</i>	<i>i</i>	
	1	2					1	1	1	1		<i>c</i>	<i>e</i>	<i>g</i>	<i>i</i>		
4)	1	2	3			5)	2	2	2			6)	<i>e</i>	<i>g</i>	<i>i</i>		
	1	2	3	4			3	3					<i>g</i>	<i>i</i>			
	1	2	3	4	5		4						<i>i</i>				
	0	1	2	3	4		25	25	25	25	25	10					
	0	1	2	3			20	20	20	20		8	8				
7)	0	1	2			8)	15	15	15			9)	6	6	6		
	0	1					10	10					4	4	4	4	
	0						5						2	2	2	2	2
	0.1	0.2	0.3	0.4	0.5		<i>a</i>					<i>a</i>					
	0.1	0.2	0.3	0.4			<i>b</i>	<i>b</i>				<i>a</i>	<i>b</i>				
10)	0.1	0.2	0.3			11)	<i>c</i>	<i>c</i>	<i>c</i>			12)	<i>a</i>	<i>b</i>	<i>c</i>		
	0.1	0.2					<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>			<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	
	0.1						<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>e</i>		<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>
	4	3	2	1	0		<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	2					
	3	2	1	0			<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>		4	4				
13)	2	1	0			14)	<i>a</i>	<i>b</i>	<i>c</i>			15)	6	6	6		
	1	0					<i>a</i>	<i>b</i>					8	8	8	8	
	0						<i>a</i>						10	10	10	10	10
	<i>z</i>	<i>z</i>	<i>z</i>	<i>z</i>	<i>z</i>		10	10	10	10	10	10	8	6	4	2	
	<i>y</i>	<i>y</i>	<i>y</i>	<i>y</i>			8	8	8	8			8	6	4	2	
16)	<i>x</i>	<i>x</i>	<i>x</i>			17)	6	6	6			18)	6	4	2		
	<i>w</i>	<i>w</i>					4	4					4	2			
	<i>v</i>						2						2				
	10						<i>a</i>					6					
	8						<i>a</i>					5					
	9	9					<i>b</i>	<i>b</i>				5	5				
	7	7					<i>b</i>	<i>b</i>				4	4				
19)	8	8	8			20)	<i>c</i>	<i>c</i>	<i>c</i>			21)	4	4	4		
	6	6	6				<i>c</i>	<i>c</i>	<i>c</i>				4	4	4		
	7	7	7	7			<i>c</i>	<i>c</i>	<i>c</i>				3	3	3		
	7	7	7	7			<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>			3	3	3	3	
	5	5	5	5			<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>			2	2	2	2	

V. Вычислить значения функции $F(x)$ в точках заданного отрезка $x \in [a, b]$ с указанным шагом h и вывести на экран в виде таблицы:

ε	Значение аргумента	Значение функции
1
2
...

Если в некоторых точках заданного интервала функция не определена, то в соответствующей графе таблицы вывести сообщение: «функция не определена».



$$1) F(x) = \begin{cases} \frac{1}{1-x^2}, & \text{при } x < 2 \\ \sqrt{x^2-1}, & \text{при } x \geq 2 \end{cases}$$

$x \in [-2; 4], h = 0.5$



$$2) F(x) = \begin{cases} \ln(8-x^2), & \text{при } x < 3 \\ x^3, & \text{при } x \geq 3 \end{cases}$$

$x \in [0; 3.5], h = 0.5$

$$3) F(x) = \begin{cases} \frac{1+x^2}{2x^2}, & \text{при } x < 1.1 \\ 0, & \text{при } x \geq 1.1 \end{cases}$$

$x \in [-0.1; 1.3], h = 0.1$

$$4) F(x) = \begin{cases} \frac{\sqrt{x-1}}{x-2}, & \text{при } x < 1.5 \\ \frac{1}{x-2}, & \text{при } x \geq 1.5 \end{cases}$$

$x \in [0.5; 4], h = 0.5$

$$5) F(x) = \begin{cases} \ln(2x-1), & \text{при } x < 1 \\ x^2, & \text{при } x \geq 1 \end{cases}$$

$x \in [0; 1.2], h = 0.1$

$$6) F(x) = \begin{cases} \frac{1}{x^2-4}, & \text{при } x < 2.2 \\ \frac{1}{x-3}, & \text{при } x \geq 2.2 \end{cases}$$

$x \in [1.9; 3], h = 0.1$

$$7) F(x) = \begin{cases} \frac{\sqrt{x-1}}{\sqrt{x^2-1}}, & \text{при } x < 1.5 \\ x+1, & \text{при } x \geq 1.5 \end{cases}$$

$x \in [-1.5; 3], h = 0.5$

$$8) F(x) = \begin{cases} \ln \frac{x}{x-4}, & \text{при } x < 4.2 \\ \sqrt{x-5}, & \text{при } x \geq 4.2 \end{cases}$$

$x \in [3.8; 5.2], h = 0.2$

$$9) F(x) = \begin{cases} \frac{1}{|x^2-1|}, & \text{при } x < 1.5 \\ \sqrt{x-2}, & \text{при } x \geq 1.5 \end{cases}$$

$x \in [0; 3], h = 0.5$

$$10) F(x) = \begin{cases} \frac{|x-2|}{|x^3+8|}, & \text{при } x < -1 \\ \sqrt{1-x}, & \text{при } x \geq -1 \end{cases}$$

$x \in [-2.5; 1.5], h = 0.5$

$$11) F(x) = \begin{cases} \frac{1}{7-x^3}, & \text{при } x < 2 \\ \sqrt{x-3}, & \text{при } x \geq 2 \end{cases}$$

$x \in [1.2; 3], h = 0.2$

$$12) F(x) = \begin{cases} \frac{1}{|2x+6|}, & \text{при } x < -2 \\ \ln(x+1), & \text{при } x \geq -2 \end{cases}$$

$x \in [-3.5; 1], h = 0.5$

$$13) F(x) = \begin{cases} \operatorname{tg}(x), & \text{при } x < \pi \\ \frac{1}{\sin(x)}, & \text{при } x \geq \pi \end{cases}$$

$x \in [0; \frac{3\pi}{2}], h = \pi/4$

$$14) F(x) = \begin{cases} \frac{1}{|1-4x|}, & \text{при } x < 0.5 \\ \ln(x-1), & \text{при } x \geq 0.5 \end{cases}$$

$x \in [0; 1.2], h = 0.1$

$$\begin{array}{ll}
15) F(x) = \begin{cases} \frac{\sqrt{2x-3}}{x-2}, & \text{при } x < 2.2 \\ \ln(x^2-8), & \text{при } x \geq 2.2 \end{cases} & 16) F(x) = \begin{cases} \ln(|x|-2), & \text{при } x < 2.5 \\ \sqrt{x-3}, & \text{при } x \geq 2.5 \end{cases} \\
x \in [1.4; 2.4], h = 0.2 & x \in [-3; 4], h = 0.5 \\
17) F(x) = \begin{cases} \sqrt{\frac{4x-1}{1-x}}, & \text{при } x < 1.5 \\ \ln(2-x), & \text{при } x \geq 1.5 \end{cases} & 18) F(x) = \begin{cases} \ln(\cos(x)), & \text{при } x < \frac{3\pi}{2} \\ \frac{1}{\cos(x)}, & \text{при } x \geq \frac{3\pi}{2} \end{cases} \\
x \in [0; 2.5], h = 0.25 & x \in [0; 2\pi], h = \pi/4 \\
19) F(x) = \begin{cases} \sqrt{\cos(x)}, & \text{при } x < \frac{3\pi}{2} \\ \sqrt{\sin(x)}, & \text{при } x \geq \frac{3\pi}{2} \end{cases} & 20) F(x) = \begin{cases} \ln(\frac{1}{\sin(x)}), & \text{при } x < \frac{3\pi}{2} \\ \ln(\sin(x)), & \text{при } x \geq \frac{3\pi}{2} \end{cases} \\
x \in [0; 2\pi], h = \pi/4 & x \in [0; 2\pi], h = \pi/4 \\
21) F(x) = \begin{cases} \ln(\frac{x-2}{x-3}), & \text{при } x < 3.5 \\ \sqrt{2x-8}, & \text{при } x \geq 3.5 \end{cases} & 22) F(x) = \begin{cases} \cot(x), & \text{при } x < \frac{3\pi}{2} \\ 0, & \text{при } x \geq \frac{3\pi}{2} \end{cases} \\
x \in [1.5; 5], h = 0.5 & x \in [0; 2\pi], h = \pi/4 \\
23) F(x) = \begin{cases} \sqrt{\frac{3x-5}{x-3}}, & \text{при } x < 3.5 \\ \ln(x-4), & \text{при } x \geq 3.5 \end{cases} & 24) F(x) = \begin{cases} \frac{\ln(x-2)}{\ln(3x-9)}, & \text{при } x < 3.2 \\ \sqrt{2x-7}, & \text{при } x \geq 3.2 \end{cases} \\
x \in [1; 4], h = 0.5 & x \in [1.8; 3.6], h = 0.2 \\
25) F(x) = \begin{cases} \frac{1}{\cos(x)}, & \text{при } x < \pi \\ \operatorname{tg}(x), & \text{при } x \geq \pi \end{cases} & 26) F(x) = \begin{cases} \sqrt{\frac{1}{1-\cos(x)}}, & \text{при } x < \frac{3\pi}{2} \\ \operatorname{tg}(x), & \text{при } x \geq \frac{3\pi}{2} \end{cases} \\
x \in [0; 2\pi], h = \pi/4 & x \in [0; 2\pi], h = \pi/4
\end{array}$$

2.5. Контрольные вопросы

1. Как можно объединить несколько действий для выполнения их в составе какого-либо оператора?
2. Какие формы оператора ветвления **if** вам известны?
3. Чем отличается оператор ветвления **switch** от оператора **if**?
4. Какие типы данных можно использовать в операторе **switch**?
5. Для чего используется оператор **break** в операторе **switch**?
6. Какой оператор применяется для многократного повторения некоторых действий?
7. Чем отличаются операторы цикла **while** от **do while**?

8. Может ли отсутствовать один из блоков цикла `for`?
9. Что означает выражение «бесконечный цикл»?
10. Могут ли различные типы циклов быть вложенными друг в друга?
11. Если переменная объявлена в блоке инициализации цикла `for` какова ее область видимости?

Применение C++ для решения математических задач

В данном разделе будут рассмотрены некоторые алгоритмы для решения прикладных математических задач, в частности, рекуррентные соотношения, суммы функциональных рядов.

3.1. Рекуррентные соотношения

В математике очень часто приходится иметь дело с вычислениями значений функций в какой-то момент времени, прогнозировании поведения какой-либо системы и т. д., другими словами, определять каким будет поведение системы в какой-либо момент времени, если известно ее состояние на текущий момент времени. Другим словами, есть последовательность чисел a_0, a_1, \dots, a_{n-1} и необходимо определить a_n . Для этого используются рекуррентные соотношения.

Рекуррентные соотношения — это формулы вида

$$a_n = f(a_{n-1}, \dots, a_{n-p}),$$

позволяющие определить n -ый член последовательности a_1, a_2, \dots, a_n по p ее предыдущим членам.

Рекуррентные соотношения встречаются во многих областях науки: в математике, физике, биологии и т. д.

Самый простой пример рекуррентной последовательности — это арифметическая или геометрическая прогрессия. Например, если известна разность d арифметической прогрессии, то рекуррентное соотношение для нее записывается в виде

$$a_k = a_{k-1} + d. \quad (3.1)$$

Как видно, для того, чтобы найти n -ый член прогрессии, необходимо знать $n - 1$ -ый член прогрессии и ее разность. Следовательно, зная первый член прогрессии и разность, можно последовательно определить все последующие члены прогрессии.

Пример 3.1. Рассмотрим пример программы, вычисляющей первые n членов арифметической прогрессии:

Листинг 3.1. Арифметическая прогрессия

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     setlocale(LC_ALL,"Russian");           // русская клавиатура
5     int a, d, n;
6     cout << "Введите первый член прогрессии\n";
7     cout << "a1="; cin >> a;               //первый член прогрессии
8     cout << "Введите разность прогрессии\n";
9     cout << "d="; cin >> d;                 //разность
10    cout << "Введите количество членов прогрессии\n";
11    cout << "n="; cin >> n;                  //количество членов прогрессии
12    cout << "a" << 1 << "=" << a << endl;  //вывод 1-ого члена прогрессии
13    for (int i = 2; i <= n; i++){           //вычисление i-ого члена
14        a += d;
15        cout << "a" << i << "=" << a << endl; //вывод на экран
16    }
17    return 0;
18 }
```

Результат работы программы при $a_1 = 5$, $d = 4$, $n = 5$:

i	a_{i-1}	$a_i = a_{i-1} + d$
1		5
2	5	9
3	9	14
4	14	19
5	19	24

□

Рассмотрим теперь в качестве примера старинную задачу, описанную итальянским математиком Леонардо Пизанским, более известным под прозвищем Фибоначчи, в XIII веке. [3]

Пара новорожденных кроликов находится в некоем замкнутом пространстве (нет хищников, питания достаточно, за год ни один кролик не умирает). Сколько пар кроликов родится при этом в течении года, если природа кроликов такова, что каждый месяц пара кроликов производит на свет другую пару, а способность к производству потомства у них появляется по достижению двухмесячного возраста.

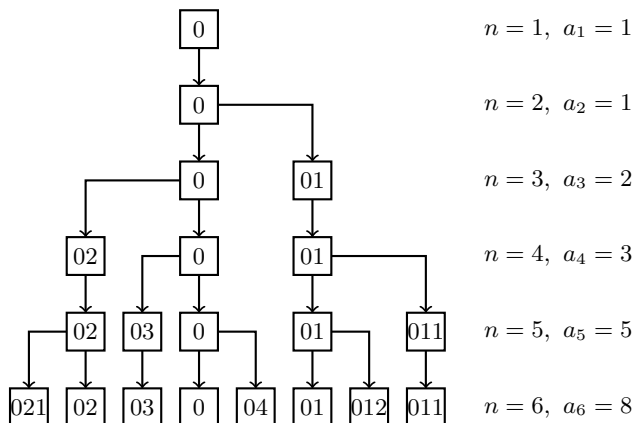


Рис. 3.1. Графическая интерпретация последовательности чисел Фибоначчи

Как видно из рисунка 3.1, первые два месяца будет одна пара кроликов, обозначенная на рисунке 0 ($a_1 = a_2 = 1$). В третий месяц крольчиха 0 произведет пару кроликов (01 на рисунке 3.1), т. е., будет уже две пары кроликов 0 и 01 ($a_3 = a_1 + a_2$). На четвертый месяц родители 0 произведут на свет еще одну пару крольчат (02), соответственно, будет три пары кроликов (родители и две пары приплода). На следующий месяц приплод принесут родители 0 (новорожденная пара обозначена 03) и первая пара крольчат 01 (дети — 011), следовательно, будет две пары приплода и три уже имеющих пары, т. е., пять пар кроликов. Получается, что в пятый месяц приплод принесут те пары кроликов, которые существовали в третий месяц. Количество пар в пятый месяц — это приплод (равный количеству пар кроликов в третий месяц (a_3)) + пары, уже бывшие в предыдущем месяце (a_4). Тогда $a_5 = a_3 + a_4$.

Таким образом, в n -ый месяц приносят приплод все пары кроликов, которые существовали в $n - 2$ -ом месяце. Тогда, общее число пар кроликов в n -ый месяц — это количество пар кроликов в $n - 1$ месяце + приплод, равный количеству пар в $n - 2$ месяце.

То есть, число пар кроликов в n -ый месяц можно определить через рекуррентное соотношение вида:

$$a_n = a_{n-1} + a_{n-2}, \quad (3.2)$$

где $n = 3, 4, \dots$ и $a_1 = a_2 = 1$.

В данном случае, для того, чтобы найти n -ый член последовательности Фибоначчи, необходимо хранить в памяти значения двух предыдущих чисел, поэтому в программе будем использовать 3 переменных, $a1$, в которой будем хранить значение $n-2$ -ого члена последовательности, $a2$ — для $n-1$ -ого числа и a — для текущего значения.

Например, при $i = 4$, значение $a1 = 1$ ($i - 2$ итерация), значение $a2 = 2$ ($i - 1$ итерация), после выполнения соотношения (3.2) значение $a = 3$.

После выполнения 4-ой итерации, значение переменной $a1$ соответствует 2-ой итерации, значение переменной $a2$ соответствует 3-ей итерации, значение переменной a соответствует 4-ой итерации,

На следующем шаге необходимо найти сумму значений переменных, соответствующих 3-ей итерации и 4-ой итерации. Ввод новых переменных невозможен, поэтому надо переопределить уже имеющиеся переменные.

Таблица 3.1. Числа Фибоначчи

Номер итерации	Шаг итерации		
	$a1$	$a2$	a
i	$i - 2$	$i - 1$	i
$i + 1$	$i - 3$	$i - 2$	$i - 1$

Как видно из таблицы 3.1, после вычисления i -ого значения числа Фибоначчи, необходимо переопределить переменные для $i + 1$ -ого шага следующим образом:

$$a1 = a2$$

$$a2 = a$$

Например, после 4-ой итерации значения переменных

$$a1 = 1, a2 = 2, a = 3.$$

После переопределения переменных значения переменных:

$$a1 = 2, a2 = 3.$$

В итоге, на 5-ой итерации значение переменной a :

$$a = a1 + a2 = 5.$$

Пример 3.2. Рассмотрим пример программы, вычисляющей n -ое число Фибоначчи:

Листинг 3.2. Вычисление числа Фибоначчи

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     setlocale(LC_ALL,"Russian");
5     int a1 = 1, a2 = 1, a, n;
6     cout << "Введите номер числа Фибоначчи\n";
7     cout << "n="; cin >> n;
8     if (n > 2){                                     //первые два числа уже известны
9         for ( int i = 3; i <= n; i++){
10             a = a1 + a2;                             //вычисляем i-ое число Фибоначчи
11             a1 = a2;                                 //a1 - i-2-ое число на следующем шаге,
12             a2 = a;                                  //a2 - i-1-ое число на следующем шаге,
13         }
14         cout << "a" << n << "=" << a << endl; //n-ое число Фибоначчи
15     }
16     else cout << "a" << n << "=1\n";
17     return 0;
18 }

```

Результат работы программы при $n = 6$:

i	$a1$	$a2$	a
1	1		
2	1	1	
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8

□

Также в виде рекуррентного соотношения можно представить такие математические понятия, как факториал и возведение в степень натурального числа.

По определению, факториал числа $n!$ — это произведение чисел от единицы до n , причем $0! = 1$. Т. е., $5! = 1 * 2 * 3 * 4 * 5 = 4! * 5$.

Тогда факториал числа n можно представить в виде следующего рекуррентного соотношения

$$n! = (n - 1)! * n. \quad (3.3)$$

или

$$a_n = a_{n-1} * n, \quad a_0 = 1. \quad (3.4)$$

Факториал числа вычисляется для целочисленного типа, так как для вещественного типа возможно накопление ошибок, и является быстрорастущей функцией, поэтому выполнять вычисления можно только до значения $n < 13$. При больших n не хватает памяти, выделенной для переменной целого типа и результат будет непредсказуем.

Пример 3.3. Рассмотрим пример программы, вычисляющей факториал числа n :

Листинг 3.3. Вычисление числа Фибоначчи

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     setlocale(LC_ALL,"Russian");
5     long a = 1;                //начальное значение факториала
6     int n;
7     cout << "n="; cin >> n;    //ввод n
8     for (int i = 2; i <= n; i++)
9         a *= i;                //i!=(i-1)!*i;
10    cout << n << "!=" << a << endl;
11    return 0;
12 }
```

Результат работы программы при $n = 6$:

i	a_{i-1}	a
1		1
2	1	2
3	2	6
4	6	24
5	24	120
6	120	720

В случае, если введено значение $n > 12$, результат непредсказуем.

Например,

$n = 12$

$12! = 479001600$

Этот результат корректный.

При $n = 13$ результат некорректный:

$n = 13$

$13! = 1932053504$

в то время как умножение $12!$ на 13 дает следующий результат $13! = 6227020800$, данное число уже превосходит максимально возможное значение переменных типа `long`. \square

Возведение в степень натурального числа также можно представить в виде рекуррентного соотношения. По определению,

$$x^n = \underbrace{x \times x \times x \times \dots \times x}_{n \text{ раз}} = x^{n-1} \times x$$

$$x^0 = 1$$

Тогда для возведения в степень можно воспользоваться следующим рекуррентным соотношением:

$$a_n = a_{n-1} * x, \quad a_1 = x. \quad (3.5)$$

Пример 3.4. Рассмотрим возведение числа x в степень n .

Листинг 3.4. Возведение в степень числа

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     setlocale(LC_ALL,"Russian");
5     int a = 1;                //начальное значение
6     int n, x;
7     cout << "n="; cin >> n;
8     cout << "x="; cin >> x;
9     for (int i = 1; i <= n; i++){
10         a *= x;               //возводим в степень
11     cout << x << "^" << n << " = " << a << endl; //выводим x^n = a
12     return 0;
13 }
```

Результат работы программы при $n = 6$, $x = 2$:

i	a_{i-1}	a_i
1	1	2
2	2	4
3	4	8
4	8	16
5	16	32
6	32	64

□

В данном разделе были рассмотрены простейшие рекуррентные соотношения. Однако с помощью рекуррентных соотношения можно решать и более сложные задачи. Примеры таких задач приведены в следующем разделе.

3.2. Вычисление сумм ряда

Рассмотрим теперь задачи, связанные с вычислением сумм или произведений функциональных рядов. Сумма ряда — это выражение вида

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

На каждом шаге к сумме добавляется значение a_k , где k — номер шага. Тогда сумму ряда на k -ом шаге можно представить в виде рекуррентного соотношения

$$S_k = S_{k-1} + a_k.$$

3.2.1. Простые ряды

Под простым рядом понимается нахождение суммы рядов, каждый член которого не зависит от предыдущего значения.

Например, необходимо найти сумму следующего ряда:

$$S = 1 + 2^2 + 3^2 + 4^2 + 5^2 + \dots + n^2.$$

На каждом шаге итерации к сумме добавляется i^2 . Алгоритм для решения подобных задач похож на алгоритм нахождения арифметической прогрессии, только на каждом шаге итерации к сумме добавляются различные числа, а не одинаковые, как в случае арифметической прогрессии.

Например, для $n = 6$ сумма ряда определяется следующим образом:

i	$a_i = i * i$	$S_+ = a_i$
1	1	1
2	4	5
3	9	14
4	16	30
5	25	55
6	36	91

Алгоритм действий следующий:

1. На каждом шаге определяется значение a .
2. Добавляется полученное значение к значению переменной S .

Пример 3.5. Например, найдем сумму ряда

$$S = \frac{1}{\sqrt{|\sin 1|}} + \frac{1}{\sqrt{|\sin 2|}} + \frac{1}{\sqrt{|\sin 3|}} + \dots + \frac{1}{\sqrt{|\sin n|}}.$$

Сокращенно, эту сумму можно представить в виде

$$S = \sum_{k=1}^n \frac{1}{\sqrt{|\sin k|}}.$$

Тогда, $a_k = \frac{1}{\sqrt{|\sin k|}}$ и для этой суммы на k -ом шаге можно записать рекуррентное соотношение вида

$$S_k = S_{k-1} + \frac{1}{\sqrt{|\sin k|}}.$$

Начальное значение суммы $S_0 = 0$.

Листинг 3.5. Вычисление суммы ряда (пример 3.5)

```

1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4 int main(){
5     setlocale (LC_ALL,"Russian");
6     float S=0, n;                //начальное значение
7     cout << "n="; cin >> n;
8     for (float i = 1; i <= n; i++)
9         S += 1 / sqrt (abs (sin (i))); //сумма ряда
10    cout << "S=" << S << endl;
11    return 0;
12 }
```

Результат работы программы при $n = 6$:

i	a	S
1	1.09014	1.09014
2	1.04869	2.13883
3	2.66199	4.80081
4	1.1495	5.95031
5	1.02119	6.9715
6	1.8918	8.8633

□

3.2.2. Сложные ряды

В случае, когда члены функционального ряда представляют собой простые функции, проблем с вычислением суммы или произведения ряда не возникает. Более сложной является задача, когда члены ряда — это функции, содержащие либо факториалы, либо возведение в степень натурального числа. Например, необходимо найти сумму следующего ряда:

$$\sum_{n=1}^k \frac{(-1)^n x^{2n+1}}{(2n+1)!} \quad (3.6)$$

Факториал числа и возведение в степень, как было рассмотрено в предыдущих разделах, являются рекуррентными соотношениями. Каждое из этих слагаемых необходимо вычислять через рекуррентные соотношения, это, во-первых, загромождает программу, во-вторых, нерационально, так как три цикла — это затраты времени, в-третьих, факториал — быстрорастущая функция, определенная только для $n \leq 12$. Однако, выражение $\frac{x^n}{n!}$ определено для $n > 12$, если сразу вычислять отношение, а не определять по отдельности сначала числитель, потом знаменатель, а потом вычислять их частное. Поэтому рассмотрим прием, который облегчает решение таких задач.

Распишем сумму (3.6):

$$\sum_{n=1}^k \frac{(-1)^n x^{2n+1}}{(2n+1)!} = -\frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Рассмотрим по шагам как будут вычисляться члены ряда.

Как видно из таблицы 3.2 на каждом шаге член ряда умножается на один и тот же параметр, соответственно, член ряда на текущем шаге можно рекуррентно

Таблица 3.2. Вычисление суммы ряда (3.6)

i	a_i	соотношение (число)	соотношение (символ.)
1	$-\frac{x^3}{3!}$		
2	$\frac{x^5}{5!}$	$-\frac{x^3}{3!} \cdot \frac{-x^2}{4 \cdot 5}$	$a_1 \cdot \frac{-x^2}{2i(2i+1)}$
3	$-\frac{x^7}{7!}$	$\frac{x^5}{5!} \cdot \frac{-x^2}{6 \cdot 7}$	$a_2 \cdot \frac{-x^2}{2i(2i+1)}$
4	$\frac{x^9}{9!}$	$-\frac{x^7}{7!} \cdot \frac{-x^2}{8 \cdot 9}$	$a_3 \cdot \frac{-x^2}{2i(2i+1)}$
...			
$k-1$	$\frac{(-1)^{k-1} x^{2k-1}}{(2k-1)!}$	$\frac{(-1)^{k-2} x^{2k-3}}{(2k-3)!} \cdot \frac{-x^2}{2(k-1) \cdot (2(k-1)+1)}$	$a_{k-2} \cdot \frac{-x^2}{2i(2i+1)}$
k	$\frac{(-1)^k x^{2k+1}}{(2k+1)!}$	$\frac{(-1)^{k-1} x^{2k-1}}{(2k-1)!} \cdot \frac{-x^2}{2k \cdot (2k+1)}$	$a_{k-1} \cdot \frac{-x^2}{2i(2i+1)}$

выразить через предыдущий по следующей формуле

$$a_n = a_{n-1} * d \quad (3.7)$$

Для суммы ряда (3.6) значение параметра $d = -\frac{x^2}{2n(2n+1)}$ для $n = 1, 2, \dots, k$. Значение этого параметра можно определить, не вычисляя члены ряда, из формулы (3.7) $d = \frac{a_n}{a_{n-1}}$. Рассмотрим, как определить значение этого параметра для вычисления суммы ряда (3.6):

$$\begin{aligned}
 d &= \frac{a_n}{a_{n-1}}, \\
 a_n &= \frac{(-1)^n x^{2n+1}}{(2n+1)!}, \\
 a_{n-1} &= \frac{(-1)^{n-1} x^{2(n-1)+1}}{(2(n-1)+1)!} = \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}.
 \end{aligned}$$

Тогда

$$d = \frac{(-1)^n x^{2n+1} (2n-1)!}{(-1)^{n-1} x^{2n-1} (2n+1)!} = \frac{-x^2 (2n-1)!}{(2n-1)! * 2n * (2n+1)} = -\frac{x^2}{2n(2n+1)}.$$

Мы получили результат, аналогичный определенному из таблицы (3.2).

Таким образом при вычислении сумм ряда, подобного (3.6), можно воспользоваться двумя рекуррентными соотношениями:

$$\begin{aligned} a_n &= a_{n-1} * d \\ S_n &= S_{n-1} + a_n \end{aligned} \quad (3.8)$$

Так как рекуррентных соотношений два, то и начальные значения необходимо определять как для переменной a , так и для S .

Итак, для решения задач, подобных (3.6), необходимо воспользоваться следующим алгоритмом:

1. Из формулы (3.7) определить параметр d .
2. Определить начальные значения S и a .
3. По формулам (3.8) определить сумму ряда.

Пример 3.6. Рассмотрим пример программы, вычисляющей сумму ряда (3.6). Замечание. Для наглядности будем выводить значения переменных на каждом шаге итерации:

Листинг 3.6. Вычисление суммы ряда (3.6)

```
1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4 int main(){
5     setlocale (LC_ALL,"Russian");
6     int n;
7     cout << "n="; cin >> n;
8     float x;
9     cout << "x="; cin >> x;
10    float a = -x*x*x / 6, S = a;           //начальные значения
11    cout << "i\ta\tS\n";                  //заголовок таблицы
12    cout << 1 << "\t| " << a << "\t| " << S << endl; //вывод 1-ого шага
13    for (int i = 2; i <= n; i++){
14        a *= -x*x / ( 2*i * (2*i + 1));    //i - ый член ряда
```

```

15     S += a;                                     // сумма
16     cout << i << "\t| " << a << "\t| " << S << endl;
17 }
18 return 0;
19 }

```

Результат работы программы при $n = 5$, $x = 2$:

i	a	S
1	-1.33333	-1.33333
2	0.266667	-1.06667
3	-0.0253968	-1.09206
4	0.00141093	-1.09065
5	-5.13067e-005	-1.0907
6	1.8918	8.8633

□

3.2.3. Бесконечные суммы

Очень часто в математике возникают задачи нахождения бесконечных сумм. Аналитически можно определить предел, к которому сходится подобный ряд. Компьютер должен работать с конечными задачами, соответственно, необходимо определить условие прекращения вычисления суммы.

Из математического анализа известно, что ряд является сходящимся, если существует такое $0 < \varepsilon \ll 1$, что $|S_n - S_{n-1}| < \varepsilon$. Из рекуррентного соотношения относительно суммы известно, что $S_n = S_{n-1} + a_n$. Тогда условие прекращения цикла имеет следующий вид:

$$|S_n - S_{n-1}| = |S_{n-1} + a_n - S_{n-1}| = |a_n| < \varepsilon \quad (3.9)$$

Возможна ситуация, когда ряд будет расходящимся. В этом случае получится бесконечный цикл, тогда для того, чтобы решить такую задачу, необходимо добавить проверку на количество итераций, то есть, если прошло N итераций, а выражение (3.9) ложное, то вычисление суммы прекращается.

Таким образом, условие завершения вычисления бесконечной суммы имеет следующий вид:

$$\text{abs}(a) > \text{eps} \ \&\& \ n < N$$

где a — n -ый член ряда.

Итак, для решения задач, связанных с бесконечными суммами, необходимо воспользоваться следующим алгоритмом:

1. Из формулы (3.7) определить параметр d .
2. Определить точность вычислений (переменная `eps`) и условие выхода из цикла (если ряд расходящийся) (переменная `N`).
3. Определить начальные значения S и a .
4. По формулам (3.8) определить сумму ряда до тех пор, пока $a \geq \text{eps}$ и $i < N$.

Пример 3.7. Найти сумму

$$S = \sum_{k=1}^{\infty} \frac{x^k}{k!}.$$

Решение задачи ищется по следующему алгоритму:

1. Сначала определим

$$d = \frac{a_n}{a_{n-1}} = \frac{x^n(n-1)!}{x^{n-1}n!} = \frac{x}{n}$$

2. Определяем начальные условия $a_1 = x$; $S_1 = a_1$
3. Суммы вычисляем с помощью следующих рекуррентных соотношений:

```
for (int i = 2; abs(a)< eps && i < N; i++){  
    a *= x / i;  
    S += a;  
}
```

Листинг 3.7. Вычисление суммы ряда (пример 3.7)

```
1 #include <iostream>  
2 #include <math.h>  
3 #include <iomanip>  
4 using namespace std;  
5 int main(){  
6     setlocale(LC_ALL,"Russian");  
7     int N = 100;  
8     float x, eps;
```

```

9  cout << "x="; cin >> x;
10 cout << "eps="; cin >> eps;
11 float a = x, S = a; //начальные условия
12 cout << left << setw(3) << "i\t" << setw(10) << "a\t"; //заголовок
13 cout << setw(10) << "S" << endl;
14 cout << left << setw(3) << 1 << "\t" << setw(10) << a; //1 шаг
15 cout << "\t" << setw(10) << S << endl;
16 for (int i = 2; abs(a) > eps && i <= N; i++){ //i-ый шаг
17     S += a;
18     a* = x/i;
19     cout << left << setw(3) << i << "\t" << setw(10) << a;
20     cout << "\t" << setw(10) << S << endl;
21 }
22 return 0;
23 }

```

Результат работы программы для $x = 2$, $N = 100$, $eps = 0.01$:

i	a_i	S_i
1	2	2
2	2	4
3	1.33333	5.3333
4	0.666667	6
5	0.266667	6.26667
6	0.0888889	6.35556
7	0.0253968	6.38095
8	0.00634921	6.3873

Видно, как программа завершает работу как только значение переменной **a** становится меньше значения переменной **eps**. □

3.3. Упражнения

Номер и количество заданий, которые необходимо выполнить студенту, а также форму отчета, выбирает преподаватель.

I. Разработать программу, которая вычисляет n -ый член последовательности

$$1) \ y_n = y_{n-1} + \frac{y_{n-2}^2}{n+1},$$

$$y_1 = 1, y_2 = -1$$

$$2) \ y_n = \frac{1}{2}y_{n-1} + \frac{1}{3}y_{n-2},$$

$$y_1 = 0.5, y_2 = 0.4$$

$$3) \quad y_n = \frac{1}{6}(y_{n-1} + \frac{1}{y_{n-2}}),$$

$$y_1 = -2, y_2 = 1$$

$$4) \quad y_n = \frac{1}{y_{n-1}^2} + y_{n-2}^3,$$

$$y_1 = 3, y_2 = 2$$

$$5) \quad y_n = \frac{1}{6}(\frac{y_{n-1}}{n-2} + \frac{y_{n-2}}{n-1}),$$

$$y_1 = 2, y_2 = 3$$

$$6) \quad y_n = \frac{1}{n}y_{n-1} + \frac{1}{n^2}y_{n-2},$$

$$y_1 = 0.9, y_2 = 1.1$$

$$7) \quad y_n = \frac{y_{n-1}}{y_{n-2}} + \frac{1}{3},$$

$$y_1 = 1.5, y_2 = 1$$

$$8) \quad y_n = \frac{n}{y_{n-1}} + \frac{y_{n-2}}{n-1},$$

$$y_1 = -0.5, y_2 = 0.5$$

$$9) \quad y_n = \frac{n}{3}y_{n-2}^2 + \frac{n+1}{2}y_{n-1}^2,$$

$$y_1 = 0.1, y_2 = -0.1$$

$$10) \quad y_n = \frac{y_{n-1}}{n+1} + \frac{y_{n-2}}{n-1},$$

$$y_1 = 0.45, y_2 = 0.55$$

$$11) \quad y_n = y_{n-2}^2 + y_{n-1} + \frac{1}{n},$$

$$y_1 = 1, y_2 = -1$$

$$12) \quad y_n = \frac{y_{n-2}}{n^2} + \frac{y_{n-1}}{n} + \frac{1}{2},$$

$$y_1 = 0.5, y_2 = -0.5$$

$$13) \quad y_n = 5 - \frac{y_{n-2}}{(n+2)^2} - \frac{y_{n-1}^2}{(n+1)},$$

$$y_1 = 1, y_2 = 3$$

$$14) \quad y_n = y_{n-1}^2 + \frac{y_{n-1}}{y_{n-2}} + y_{n-2},$$

$$y_1 = 2, y_2 = 0.2$$

$$15) \quad y_n = \frac{1}{6}(y_{n-2} - y_{n-1}),$$

$$y_1 = 1, y_2 = -1$$

$$16) \quad y_n = \frac{1}{3}(\frac{y_{n-1}}{2} + \frac{y_{n-2}}{3}),$$

$$y_1 = 2, y_2 = 1$$

$$17) \quad y_n = \frac{1}{5}(2y_{n-2} + 3y_{n-1}),$$

$$y_1 = 4, y_2 = -4$$

$$18) \quad y_n = \frac{1}{n}((n-1)y_{n-1} + (n-2)y_{n-2}),$$

$$y_1 = 1, y_2 = 0.5$$

$$19) \quad y_n = \frac{y_{n-2}^2}{y_{n-1}},$$

$$y_1 = 1, y_2 = 2$$

$$20) \quad y_n = \frac{1}{n-1}y_{n-2} + \frac{1}{n-2}y_{n-1},$$

$$y_1 = 0.5, y_2 = 0.4$$

$$21) \quad y_n = \frac{y_{n-1}}{n+1} + \frac{y_{n-2}}{n-1},$$

$$y_1 = 1, y_2 = -1$$

$$22) \quad y_n = \frac{1}{n}(y_{n-1} + ny_{n-2}),$$

$$y_1 = 5, y_2 = 4$$

$$23) \quad y_n = \frac{n}{3}y_{n-2} + \frac{n-1}{2}y_{n-1},$$

$$y_1 = -1, y_2 = -2$$

$$24) \quad y_n = \frac{1}{5}y_{n-2} + \frac{1}{4}y_{n-1} + \frac{1}{n},$$

$$y_1 = 0.5, y_2 = 0.25$$

II. Разработать программу, вычисляющую сумму функционального ряда

$$1) \quad S = 1 + 2^2 + 3^2 + \dots + n^2,$$

$$2) \quad S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n},$$

- 3) $S = 1 + 2^3 + 3^3 + \dots + n^3$, 4) $S = 1 + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n}$,
5) $S = 1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \dots + \frac{1}{\sqrt{n}}$, 6) $S = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$,
7) $S = 1 + e^2 + e^3 + \dots + e^n$, 8) $S = 1 + \frac{1}{e^2} + \frac{1}{e^3} + \dots + \frac{1}{e^n}$,
9) $S = 1 + \sin 2 + \sin 3 + \dots + \sin n$, 10) $S = 1 + \frac{1}{\sin 2} + \frac{1}{\sin 3} + \dots + \frac{1}{\sin n}$,
11) $S = \frac{1}{2} + \frac{2}{2} + \frac{3}{2} + \dots + \frac{n}{2}$, 12) $S = 1 + \frac{1}{\sqrt{e^2}} + \frac{1}{\sqrt{e^3}} + \dots + \frac{1}{\sqrt{e^n}}$,
13) $S = 1 + \cos 2 + \cos 3 + \dots + \cos n$, 14) $S = 1 + \frac{1}{\cos 2} + \frac{1}{\cos 3} + \dots + \frac{1}{\cos n}$,
15) $S = 1 + \cos^2 2 + \cos^2 3 + \dots + \cos^2 n$, 16) $S = 1 + \cos^2 2 + \cos^2 3 + \dots + \cos^2 n$,
17) $S = 1 + 4 + 6 + \dots + 2n$, 18) $S = 1 + \frac{1}{\cos^2 2} + \frac{1}{\cos^2 3} + \dots + \frac{1}{\cos^2 n}$,
19) $S = 1 + \ln 2 + \ln 3 + \dots + \ln n$, 20) $S = 1 + \frac{1}{4} + \frac{1}{6} + \dots + \frac{1}{2n}$,
21) $S = 1 + \cos 4 + \cos 6 + \dots + \cos 2n$, 22) $S = 1 + \sin 6 + \sin 9 + \dots + \sin 3n$,
23) $S = 1 + \cos^2 4 + \cos^2 6 + \dots + \cos^2 2n$, 24) $S = 1 + \frac{1}{3} + \frac{1}{6} + \dots + \frac{1}{3n}$,

III. Разработать программу, вычисляющую сумму функционального ряда

- 1)
$$S = \sum_{n=1}^k \frac{x^{n+1}}{(2n+1)!}$$
, 2)
$$S = \sum_{n=1}^k \frac{(-1)^{n+1} x^{2n-1}}{2n!}$$
,
3)
$$S = \sum_{n=1}^k \frac{x^{2n-1} 2^{2n+1}}{(n+1)!}$$
, 4)
$$S = \sum_{n=1}^k \frac{(-1)^n x^{2n+1}}{(2n)!}$$
,
5)
$$S = \sum_{n=1}^k \frac{(-1)^{n-1} x^{2n+2}}{(2n+1)}$$
, 6)
$$S = \sum_{n=1}^k \frac{(-1)^n x^{2n}}{3n!}$$
,
7)
$$S = \sum_{n=1}^k \frac{(-1)^{n-1} x^{2n+2}}{3^n (n+1)}$$
, 8)
$$S = \sum_{n=1}^k \frac{(-1)^{n-1} x^{n-1}}{(2n-1)!}$$
,

9)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} x^2}{2^{2n+1} (2n-1)!}$$

,

11)

$$S = \sum_{n=1}^k \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

,

13)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} x^{n-1}}{3^n 2(n+1)!}$$

,

15)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} x^{2n+1} 2^{n+1}}{(2n+1)!}$$

,

17)

$$S = \sum_{n=1}^k \frac{x^{2n+1} 3^{n-1} (n+1)!}{2^{n+1}}$$

,

19)

$$S = \sum_{n=1}^k \frac{x^{2n+1}}{2^{n+1} (2n+1)!}$$

,

21)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} x^{2n-2}}{(2n-1)!}$$

,

23)

$$S = \sum_{n=1}^k \frac{(-1)^n x^{n+1}}{2(n-1)!}$$

,

10)

$$S = \sum_{n=1}^k \frac{x^{2n-1} 3^{2n+1}}{2n!}$$

,

12)

$$S = \sum_{n=1}^k \frac{3^{2n-1}}{x^{2n+1} 2n}$$

,

14)

$$S = \sum_{n=1}^k \frac{(-1)^{n-1} x^{2n+2} n!}{n^2 + n + 1}$$

,

16)

$$S = \sum_{n=1}^k \frac{(-1)^n x^{2n+2}}{(2n+2)!}$$

,

18)

$$S = \sum_{n=1}^k \frac{(-1)^{n-1} x^{2n} 3^{n+2}}{(2n+1)!}$$

,

20)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} (2n)! x^{2n-1}}{2^{n-1}}$$

,

22)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} 3^{2n-1} x^n}{(2n+1)!}$$

,

24)

$$S = \sum_{n=1}^k \frac{(-1)^{n+1} 3^{2n-1} (2n+1)!}{x^{2n+1}}$$

,

3.4. Контрольные вопросы

1. Что такое рекуррентные соотношения?
2. Как можно представить сумму n чисел через рекуррентные соотношения?

3. Что такое последовательность Фибоначчи?
4. Представьте факториал с помощью рекуррентных соотношений.
5. Представьте число x^n через рекуррентные соотношения.
6. Что такое сумма ряда?
7. Как найти бесконечную сумму ряда?
8. Как определить множитель для представления члена ряда через рекуррентные соотношения?

Пользовательские функции в C++

Несложно сразу написать код программы, находящей среднемесячную температуру или подсчитывающую расстояние, пройденное человеком при равномерном движении за определенное время. Но на практике часто приходится решать гораздо более сложные задачи. Одним из подходов решения таких задач является метод «сверху вниз». Он состоит в разбиении решаемой задачи на некоторую совокупность более простых задач. После того, как такое разбиение состоялось для каждой простой подзадачи разрабатывается соответствующий вспомогательный алгоритм. При этом бывает не просто не запутаться в переменных, правильно организовать передачу информации между подзадачами и т. п. Огромную помощь в такой работе оказывают функции. Оформление каждой подзадачи в виде отдельной функции позволяет облегчить процессы программирования, тестирования и отладки кода.

Математическое понятие функции выражает интуитивное представление о том, как одна величина полностью определяет значение другой величины. Так значение переменной x однозначно определяет значение выражения x^2 , а значение месяца однозначно определяет значение следующего за ним месяца, также любому человеку можно сопоставить другого человека — его отца. Аналогично, некоторый задуманный заранее алгоритм по варьируемым входным данным выдает определенные выходные данные.

В математике часто используется следующее интуитивное определение функции.

Функция f — это закон или правило, согласно которому каждому элементу x из множества X ставится в соответствие единственный элемент y из множества Y .

Записывается функция следующим образом: $y = f(x)$.

Функционирование программ, написанных на языке программирования C++, основано на использовании функций. Без функций в нем нельзя сделать ничего. Любая программа на C++ имеет по крайней мере одну функцию — это функция `main()`. Ввод-вывод можно организовывать с помощью функций `printf()` и `scanf()`, находить модуль числа x функцией `abs(x)`, синус — функцией `sin(x)` и т. д.

Но C++ позволяет не только использовать стандартные, но и создавать соб-

ственные функции. Этому вопросу и посвящена данная глава.

4.1. Простые функции

4.1.1. Начальные сведения

Для того чтобы использовать функцию в C++, требуется выполнить следующие шаги:

- Представить определение функции.
- Представить прототип функции.
- Вызвать функцию.

Если используется библиотечная функция, то она уже определена и скомпилирована. Все что остается — правильно вызвать эту функцию. Например, перечень стандартных библиотечных функций C++ включает функцию `sqrt()` для нахождения квадратного корня вещественного числа. Стандартный заголовочный файл `math.h` содержит прототип функции для `sqrt()` и ряда других математических функций. Поэтому можно использовать эту функцию, не заботясь об ее реализации. При создании собственных функций, приходится самостоятельно обработать все три аспекта — определение, прототипирование и вызов.

Пример 4.1. Демонстрация всех три шагов создания функции:

Листинг 4.1.

```
1 #include<iostream>
2 using namespace std;
3
4 void simple();           //прототип функции
5
6 int main() {
7     cout << "main() вызовет функцию simple():\n";
8     simple();             //вызов функции
9     return 0;
10 }
11
12 void simple(){           //определение функции
```

```

13 cout << "Я - функция simple.\n";
14 }

```

Ниже показан вывод программы из листинга 4.1:

```

main() вызовет функцию simple():
Я - функция simple.

```

□

Рассмотрим эти три шага подробнее.

4.1.2. Определение функции

Все функции можно разбить на две категории: те, которые не возвращают значений, и те, которые их возвращают. Функции, не возвращающие значений, имеют специальный тип **void** и следующую общую форму:

```

void<имя функции> ( [ <список формальных параметров> ] )
{
    <оператор(ы)>
    return //не обязательно
}

```

Здесь *список формальных параметров* указывает типы и количество аргументов (параметров), передаваемых функции. Он может и отсутствовать, но наличие скобок при этом обязательно. Необязательный оператор **return** отмечает конец функции. При его отсутствии функция завершается на закрывающей фигурной скобке [4].

Функция с возвращаемым значением отдает генерируемое ею значение функции, которая ее вызвала. Другими словами, если функция возвращает квадратный корень из 9.0 (`sqrt(9.0)`), то вызывающая ее функция получает значение 3.0.

Такая функция объявляется, как имеющая тип — такой, как у возвращаемого ею значения. Вот ее общая форма:

```

<тип возвращаемого значения> <имя функции> ( [ <список формальных параметров> ] )
{
    <оператор(ы)>
    return <значение>
}

```

Функции с возвращаемыми значениями требуют использования оператора `return` таким образом, чтобы вызывающей функции было возвращено значение. Само значение может быть константой, переменной либо более общим выражением. Единственное требование — чтобы выражение по типу совпадало с *типом возвращаемого значения* либо могло быть преобразовано в этот тип. (Если, скажем, объявлен возвращаемый тип `double`, а функция возвращает выражение `int`, то `int` неявно приводится к `double`.) Затем функция возвращает конечное значение в ту функцию, которая ее вызвала. Язык C++ накладывает ограничения на типы возвращаемых значений: возвращаемое значение не может быть массивом. Все остальное допускается — целые числа, числа с плавающей точкой, указатели и даже структуры и объекты! (Интересно, что несмотря на то, что функция C++ не может вернуть массив непосредственно, она все же может вернуть указатель на первый элемент массива.)

Функция завершается после выполнения оператора `return`. Если функция содержит более одного оператора `return`, например, в виде альтернатив разных выборов `if else`, в этом случае она прекращает свою работу по достижении первого оператора `return`. Например, в следующем листинге `else` излишен:

Листинг 4.2.

```
1  int max(int x, int y){
2  if (x > y)
3      return x;
4  else
5      return y;
6  }
```

4.1.3. Прототипирование и вызов функции

Прототип описывает интерфейс функции для компилятора. Т. е. он сообщает компилятору, каков тип возвращаемого значения, если он есть у функции, а также количество и типы аргументов данной функции.

Возникает вопрос. Зачем нужны прототипы и не проще ли обойтись без них? В простой программе с небольшим кодом обычно так и поступают. Однако функция может и не находиться в том же файле, где находится стартовая функция `main()`. C++ позволяет разбивать программу на множество файлов, которые компилируются независимо и позднее собираются вместе в одну исполняемую программу. В таких

случаях без прототипов не обойтись. Кстати, в прототипе не обязательно даже указывать имена аргументов достаточно только типы. Например `int mult(int, int)`.

При задании нескольких аргументов требуется описать тип каждой переменной отдельно. Например недопустимо объявление: `int mult(int x, y)`.

В коде, приведенном в листинге 4.3 присутствуют и прототипы и определения и вызовы двух функций. В дальнейшем из-за простоты рассматриваемых задач прототипы использоваться не будут.

Пример 4.2. Пример использования прототипов, вызовов и определения функций

Листинг 4.3.

```
1 #include<iostream>
2 using namespace std;
3 int mult(int x, int y);    //Прототип: возвращает int
4 void print_odd(int x);     //Прототип: не возвращает значение
5
6 int main(){
7     int a, b;
8     cout << "Введите 2 целых числа:";
9     cin >> a >> b;
10    int z = mult(a, b);     //Вызов функции
11    cout << z << " - ";
12    print_odd(z);           //Вызов функции
13    return 0;
14 }
15 //---Определение функций---
16 int mult (int x, int y){
17     return x * y;
18 }
19
20 void print_odd (int x){
21     if (x%2 == 0)
22         cout << "четное";
23     else
24         cout << "нечетное";
25 }
```

Обратите внимание, что `main()` вызывает функцию `print_odd()` типа `void`, используя имя функции и аргумент, за которыми следует точка с запятой. Это

пример оператора вызова функции. Но поскольку `mult()` возвращает значение, `main()` может использовать его в правой части оператора присваивания:

```
int z = mult(a, b).
```

□

4.1.4. Аргументы функций, передача по значению и по ссылке

Теперь рассмотрим аргументы функций. C++ обычно передает их по значению. Это означает, что числовое значение аргумента передается в функцию, где присваивается новой переменной.

Пример 4.3. Найти площадь треугольника по трем его сторонам.

Листинг 4.4.

```
1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4
5 double square(double x, double y, double z){ //формула Герона
6     double p = (x + y + z)/2;
7     double s = sqrt(p * (p - x) * (p - y) * (p - z));
8     return s;
9 }
10
11 int main(){
12     double sq = square(3, 4, 5); //вызов функции
13     cout << sq << endl;
14     return 0;
15 }
```

□

Заголовок функции `square()` выглядит так:

```
double square(double x, double y, double z).
```

Вызов функции выглядит так:

```
double sq = square(3, 4, 5).
```

Когда эта функция вызывается, она создает три новых переменных типа `double` по имени `x`, `y` и `z` и присваивает им значения 3, 4 и 5 соответственно (на рисунке 4.1

по значениям локальных переменных `x`, `y` и `z` вычисляется значение локальной переменной `p` (пунктирная линия).

Затем при выполнении команды

```
double s = sqrt (p * (p - x)* (p - y)* (p - z));
```

по значениям локальных переменных `x`, `y`, `z` и `p` вычисляется значение локальной переменной `s` (линия пунктирная с точкой).

Наконец при выполнении команды `return s`; значение локальной переменной `s` будет передано в функцию `main()` глобальной переменной, указанной в вызове функции. В нашем случае это `sq` (линия пунктирная с двумя точками).

Теперь скажем несколько слов о передаче по ссылке. Для использования ссылочных аргументов характерен ряд особенностей, о которых следует знать. Программа, представленная в листинге 4.5, использует две функции для возведения значения аргумента в куб. Одна из них — `cube()` — принимает аргумент типа `double`, в то время как другая — `refcube()` — получает ссылку на значение типа `double`. Программный код процедуры возведения в куб выглядит несколько необычно, дабы нагляднее проиллюстрировать работу со ссылками.

Пример 4.4. Найти площадь треугольника по трем его сторонам.

Листинг 4.5.

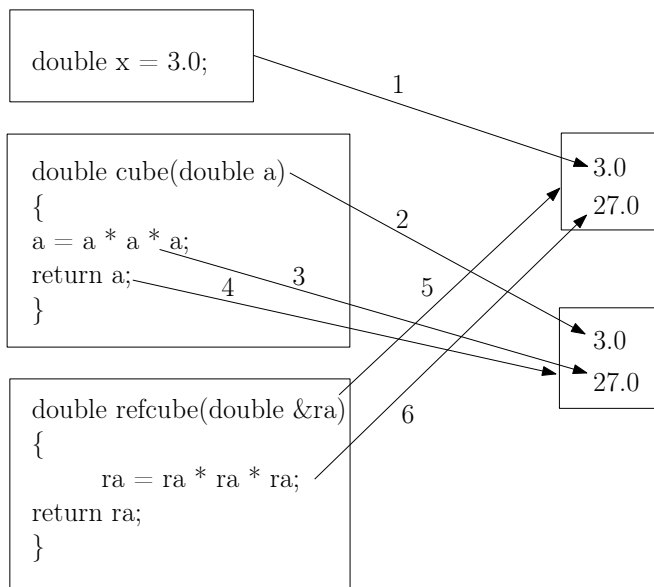
```
1 #include <iostream>
2 using namespace std;
3
4 double cube (double a);
5 double refcube (double &a);
6 int main(){
7     double x = 3.0;
8     cout << cube(x);
9     cout << " = куб числа " << x << endl;
10    cout << refcube(x);
11    cout << " = куб числа " << x << endl;
12    return 0;
13 }
14
15 double cube (double a){
16     a = a * a * a;
17     return a;
```

```

18 }
19
20 double refcube (double &ra){
21     ra = ra * ra * ra;
22     return ra;
23 }

```

□



- 1 - Создается глобальная переменная *a* и ей присваивается значение 3
- 2 - Создается локальная переменная *a* и ей присваивается значение 3
- 3 - Локальной переменной *a* присваивается значение 27
- 4 - Локальная переменная *a* уничтожается
- 5 - В функции используется глобальная переменная *a*
- 6 - Глобальной переменной *a* присваивается значение 27

Рис. 4.2. Выполнение функций из листинга 4.5

Ниже показан результат выполнения программы из листинга 4.5:

```

27 = куб числа 3
27 = куб числа 27

```

Таким образом функция `refcube()` изменяет значение `x` в функции `main()`, в то время как функция `cube()` этого не делает (рисунок 4.2). Т. о. передача пара-

метров по ссылке позволяет вызываемой функции получить доступ к переменным в вызывающей функции.

Заметим, что если в `main()` будет объявлена глобальная переменная `x` и другая локальная переменная `x` в некоторой другой функции, то это будут две совершенно различных, никак не связанных переменных.

Листинг 4.6.

```
1 #include <iostream>
2 using namespace std;
3 int x;                //Объявление глобальной переменной
4 void funk();
5 int main(){
6     cout << x << ' '; //1-й cout
7     funk();
8     cout << x << endl; //3-й cout
9     return 0;
10 }
11
12 void funk(){
13     int x;            //Объявление локальной переменной
14     x = 2;
15     cout << x << ' '; //2-й cout
16 }
```

В приведенном примере будет создана глобальная переменная `x`, которой будет присвоено значение 1. Поэтому при вызове первого `cout` на экран будет выведено значение 1. Затем при вызове функции `funk()` будет создана локальная переменная `x` со значением 2, которое и будет выведено на экран вторым вызовом `cout`. При этом глобальная переменная `x` со значением 1 по прежнему существует, хотя и пока не доступна. По окончании работы функции `funk()` локальная переменная будет уничтожена. Поэтому третий `cout` выведет на экран значение глобальной переменной `x`. Таким образом на экран будет выведена строка:

1 2 1.

Однако если убрать определение локальной переменной в теле функции `funk()`, то вся работа будет производиться только с одной глобальной переменной `x`. Тогда в результате работы программы на экран будет выведена строка:

1 2 2.

Рассмотрим еще несколько примеров, иллюстрирующих передачу по ссылке и по значению.

Листинг 4.7.

```
1 #include <iostream>
2 using namespace std;
3 void change(int a, int b){ //Строка с формальными параметрами
4     a = 3; b = 4;          //Внутренние переменные
5 }
6
7 int main(){
8     int a = 1, b = 2;      //Внешние переменные
9     change(a, b);          //Строка с фактическими параметрами
10    cout << a << ' ' << b << endl;
11    return 0;
12 }
```

В этом примере при начале работы функции `main()` будут созданы две глобальные переменные `a` и `b`, которым будут присвоены значения 1 и 2 соответственно. При обращении к функции `change()` переменная `a` используется в качестве фактического параметра для формального параметра `a`, а переменная `b` — в качестве фактического параметра для формального параметра `b`. Однако, поскольку оба параметра передаются по значению, будут созданы временные переменные `a` и `b`, которые получив от глобальных переменных значения 1 и 2 соответственно, больше никак не будут с ними связаны. При выполнении тела функции `change()` эти временные переменные получают значения 3 и 4, однако сразу после окончания ее работы они будут уничтожены. Поэтому `cout` на экран выведет значения глобальных переменных `a` и `b`, т. е. 1 и 2.

Теперь заменим строку 3 листинга 4.7 на строку

```
void change (int &a, int &b){ .
```

Оба параметра передаются по ссылке и, следовательно, по окончании работы функции `change()` формальный параметр `a` вернет измененное значение 3 глобальной переменной `a`, а формальный параметр `b` — значение 4 глобальной переменной `b`. Поэтому на экран будут выведены новые значения 3 и 4.

Внесем еще одно изменение в программу, сохраняя передачу параметров по

ссылке. Вместо строки 9 листинга 4.7 напечатаем

```
change (b, a).
```

В этом случае при начале работы функции `main()` будут созданы две глобальные переменные `a` и `b`, которым будут присвоены значения 1 и 2 соответственно. При обращении к функции `change()` переменная `b` (как стоящая в списке фактических параметров первой) используется в качестве фактического параметра для формального параметра `a` (стоящего первым в списке формальных параметров) и передает ему значение 2, а переменная `a` — в качестве фактического параметра для формального параметра `b` со значением 1. В теле функции `change()` эти значения будут изменены на 3 для внутренней переменной `a` и 4 для внутренней переменной `b`. По окончании работы этой функции внутренняя переменная `b` передаст свое значение 4 глобальной переменной `a` (от которой она получила свое начальное значение), а внутренняя переменная `a` — значение 3 глобальной переменной `b`. Поэтому на экране будет выведена строка: 4 3.

Теперь рассмотрим этот же код но со следующими изменениями:

```
void change(int &a, int b){ //Строка 3
change(b, a)//Строка 10.
```

От предыдущего примера этот отличается тем, что для формального параметра `b` используется передача по значению. Следовательно при работе функции `change()` будет создана отдельная локальная переменная `b`, которая уже не будет связана с глобальной переменной `a`, которая служит для нее фактическим параметром. Поэтому по окончании работы этой функции значение глобальной переменной `a` не будет меняться и останется равным 1. На экране будет выведено: 1 3.

4.1.5. Параметры по умолчанию

При работе со библиотечными функциями C++ можно встретиться с ситуацией, когда в функции используется три аргумента, но разрешается вводить только два. В этом случае для «отсутствующего» параметра будет использовано *значение по умолчанию*. В листинге 4.8 приведен простой пример, демонстрирующий использование значений по умолчанию для параметров функций.

Листинг 4.8.

```
1 #include <iostream>
```



```

2 using namespace std;
3 void f(int a, int b = 10, int c = 20){
4     return a + b + c;
5 }

```

Если обратиться к функции `cout << f(1, 2, 3);`, то на экране появится ожидаемое значение 6. Если вызвать функцию так: `cout << f(1, 2);` параметр `a` получит значение 1, параметр `b` — 2, а параметр `c`, поскольку отсутствует третий аргумент, использует значение по умолчанию 20. На экране появится число 23. Соответственно, при обращении `cout << f(1);` будет выведено 31, т. к. не только параметр `c`, но и параметр `b` получит значение по умолчанию. Однако не получится использовать следующее обращение `cout << f(1, , 2);`. Если опускается значение одного параметра, то следует опустить значения для всех последующих параметров; не допускается опускать средний параметр.

4.2. Рекурсивные функции

4.2.1. Рекурсивные алгоритмы

В математике нередки задачи, для решения которых требуется решить некоторое количество подзадач, представляющих собой исходную задачу меньшей размерности. Возьмем, к примеру, задачу подсчета факториала натурального числа n . $n! = 1 \times 2 \times \dots \times (n - 1) \times n = (1 \times 2 \times \dots \times (n - 1)) \times n = (n - 1)! * n$. Т. о. чтобы подсчитать факториал числа n нужно сначала подсчитать факториал числа $n - 1$, а затем домножить его на n .

Такие алгоритмы называются рекурсивными. Однако если алгоритм каждый раз будет вызывать сам себя, получится бесконечная последовательность вызовов. Поэтому задача должна иметь при некоторых наборах исходных данных простое рекурсивное решение. Рекурсивный процесс должен шаг за шагом так упрощать задачу, чтобы в итоге получилось одно из этих нерекурсивных решений. Для рассматриваемого примера таким решением можно считать факториал числа 0, который, по определению, равен 1. Так достигается прерывание последовательности рекурсивных вызовов.

Учитывая вышесказанное можно так описать формулу для рекурсивного алгоритма подсчета факториала.

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ (n-1)! \times n, & \text{в противном случае.} \end{cases}$$

В листинге 4.9 приведена реализация соответствующей рекурсивной функции.

Листинг 4.9.

```
1 long factorial (int n){
2     if (n == 0) return 1;
3     return factorial(n - 1) * n;
4 }
```

В результате получим следующий порядок вызовов и возвратов для обращения `factorial(3)`.

<code>factorial(3)</code>	уровень 1; добавление уровней рекурсии
<code>factorial(2)</code>	уровень 2;
<code>factorial(1)</code>	уровень 3;
<code>factorial(0)</code>	уровень 4; финальный рекурсивный вызов
<code>return 1</code>	уровень 4; начало обратного прохода
<code>return 1*1=1</code>	уровень 3;
<code>return 1*2=2</code>	уровень 2;
<code>return 2*3=6</code>	уровень 1;

При этом возникает закономерный вопрос. Как программа не запутывается в многочисленных `n`. Заметим, что каждый рекурсивный вызов создает свой собственный набор переменных, поэтому на момент четвертого вызова она и имеет четыре отдельных переменных по имени `n` — каждая со своим собственным значением. В этом можно убедиться модифицируя листинг 4.9 таким образом, чтобы отображать адрес переменной `n`. Для этого достаточно в начало тела функции добавить строки

```
cout << "Вызов factorial("<< n << ")";
cout << "(n по адресу"<< &n << ")<<endl;
```

```
Вызов factorial(3) (n по адресу 001DF9E4)
Вызов factorial(2) (n по адресу 001DF90C)
Вызов factorial(1) (n по адресу 001DF834)
Вызов factorial(0) (n по адресу 001DF75C)
```

Обратите внимание, что переменная `n`, имеющая значение 2, размещается в одном месте (в данном примере по адресу 001DF90C), а переменная `n`, имеющая значение 3, находится в другом месте (адрес памяти 001DF9E4) и так далее.

4.2.2. Когда не следует использовать рекурсию

Рассмотрим следующую рекурсивную функцию вычисления n -го члена последовательности Фибоначчи.

Листинг 4.10.

```
1 long fib(int n){
2     cout << "Вычисляется f(" << n << ")\n";
3     if (n == 1 || n == 2) return 1;
4     return fib(n - 1) + fib(n - 2);
5 }
```

Данная функция, кроме непосредственного вычисления числа Фибоначчи отображает на экране каждое обращение к функции.

Если обратиться к функции `long f=fib(1);`, то переменная `f` получит значение 1, а на экране появится:

Вычисляется f(1)

Если обратиться к функции `long f=fib(7);`, то переменная `f` получит значение 13, а на экране появится страшная последовательность из 25 строк:

Вычисляется f(7)
Вычисляется f(6)
Вычисляется f(5)
Вычисляется f(4)
Вычисляется f(3)
Вычисляется f(2)
Вычисляется f(1)
Вычисляется f(2)
Вычисляется f(3)
Вычисляется f(2)
Вычисляется f(1)
Вычисляется f(4)
Вычисляется f(3)
Вычисляется f(2)
Вычисляется f(1)
Вычисляется f(2)
Вычисляется f(5)
Вычисляется f(4)

Вычисляется $f(3)$
Вычисляется $f(2)$
Вычисляется $f(1)$
Вычисляется $f(2)$
Вычисляется $f(3)$
Вычисляется $f(2)$
Вычисляется $f(1)$

В таблице приведено для каждого значения аргумента количество вызовов функции с этим аргументом. Как видим, кроме последнего значения последовательность вызовов представляет собой уже знакомую последовательность Фибоначчи!!!

Аргумент	Количество вызовов
7	1
6	1
5	2
4	3
3	5
2	8
1	5

Рекурсия является очень мощным оружием программиста, способным украсить любую программу. Но применять ее все-таки следует с осторожностью.

4.3. Перегрузка функций

4.3.1. Описание

Выше было описано, как аргументы, определенные по умолчанию, позволяют вызывать одну и ту же функцию с указанием различного количества аргументов. Теперь познакомимся с так называемой перегрузкой функций, которая предоставляет возможность использовать несколько функций с одним именем.

Главную роль в перегрузке функций играет список аргументов, который еще называют сигнатурой функции. Если две функции используют одно и то же количество аргументов и одинаковые их типы, причем они следуют в одном и том же порядке, то функции имеют одну и ту же сигнатуру. Имена переменных не имеют значения. Язык C++ предоставляет возможность определить две функции с одним именем при условии, что эти функции обладают разными сигнатурами. Сигнатуры мо-

гут различаться по количеству аргументов или по их типам, либо по тому и другому. Например, можно определить набор функций `print()` со следующими прототипами:

```
void print(const char *str, int width); //Сигнатура 1
void print(double d, int width); //Сигнатура 2
void print(long i , int width); //Сигнатура 3
void print(int i, int width); //Сигнатура 4
void print(const char *str); //Сигнатура 5
```

При последующем вызове функции `print()` компилятор сопоставит используемый вариант с прототипом, имеющим ту же сигнатуру:

```
print("Перегрузка", 10); используется сигнатура 1
print("функция"); используется сигнатура 5
print(1111.0, 10); используется сигнатура 2
print(1999, 12); используется сигнатура 4
print(1999L ,15); используется сигнатура 3
```

При вызове перегруженных функций важно правильно указывать типы аргументов.

Для примера рассмотрим следующие операторы:

```
unsigned int year = 3210;
print(year, 6);
```

В этом примере вызов функции `print()` не соответствует ни одному прототипу. Однако отсутствие подходящего прототипа не исключает автоматическое использование одной из функций, поскольку среда C++ предпримет попытку выполнить стандартное приведение типов, чтобы достичь соответствия. Если бы, скажем, единственным прототипом функции `print()` был прототип 2, вызов функции `print(year, 6)` повлек бы за собой преобразование значения `year` к типу `double`. Однако в программном коде, приведенном выше, имеется три прототипа, которые используют число в качестве первого аргумента, при этом возникают три варианта преобразования значения `year`. И не всегда можно догадаться какой из них выберет конкретная среда разработчика, или вообще объявит об ошибке.

Нельзя определить принадлежность вызова `f(a)` одной из следующих сигнатур:

```
void f(int a);  
void f(int &a);
```

Также возникнут проблемы определения прототипа `f(1,2)` при использовании следующих сигнатур:

```
void f(int a, int b, int c=10);  
void f(int a, int b);
```

Возможность перегрузки функций является хорошим помощником при написании программ, но не следует ей злоупотреблять. Перегрузку целесообразно использовать для функций, которые выполняют в основном одни и те же действия, но с различными типами данных. Кроме того, имеет смысл оценить возможность достижения той же цели посредством аргументов, принимаемых по умолчанию. Ведь использование единственной функции с аргументами, определенными по умолчанию, проще. Прежде всего, достаточно создавать только одну функцию, а не две, к тому же программе потребуется меньше памяти. Если потребуется внести изменения, достаточно будет отредактировать только одну функцию. Тем не менее, если необходимо применять аргументы разного типа, то принимаемые по умолчанию аргументы здесь не помогут. Придется использовать перегрузку функций.

Листинг 4.11 демонстрирует использования перегрузки на примере двух функций.

Листинг 4.11.

```
1 #include <iostream>  
2 float f(float x){           //Сигнатура 1  
3     if (x < 0) return 0;  
4     if (x > 1) return 1;  
5     return x;  
6 }  
7  
8 void f(float x, float &y){ //Сигнатура 2  
9     if( x < 0 ) y = 0;
```

```

10  else if (x > 1) y = 1;
11  else y = x;
12  }
13
14  int main(){
15      float x = 0.3, y;
16      cout << "1-вариант: " << f(x) << endl; //Выполняется функция 1
17      f(x,y);                                //Выполняется функция 2
18      cout << "2-вариант: " << y << endl;
19      return 0;
20  }

```

Почему же компилятор не путается с функциями, имеющими одно и то же имя. Каждой из перегруженных функций присваивается скрытый идентификатор. Компилятор C++ скрыто от пользователя осуществляет так называемое декорирование имен. При этом имя каждой функции кодируется с учетом типов формальных параметров, указанных в прототипе.

Например имеется следующий прототип в недекорированном виде:

```
long MyFunctionFoo(int, float);
```

Этот формат удобен для человека. Из него видно, что функция принимает два аргумента типов `int` и `float`, а возвращает значение типа `long`. Для собственных нужд компилятор использует внутреннее представление прототипа, которое имеет примерно следующий вид: `?MyFunctionFoo@@YAXH@Z`. В этом невразумительном обозначении закодировано количество аргументов и их типы. Получаемый набор символов зависит от сигнатуры функции, а также от применяемого компилятора. Поэтому получив правильный вызов функции компилятор может без труда определить, какую из перегруженных функций следует выполнить.

4.4. Шаблоны функций

4.4.1. Описание

Современные компиляторы языка C++ реализуют одно из его новейших средств — *шаблоны функций*. Они представляют собой обобщенное описание функций. Шаблон определяет функцию на основе обобщенного типа, вместо которого может быть подставлен определенный тип данных, такой как `int` или `double`. Передавая шаблону

тип в качестве параметра, можно заставить компилятор сгенерировать функцию для этого типа данных.

Допустим, создана функция, осуществляющая обмен значениями двух переменных типа `int`. Предположим, что вместо этого необходимо произвести обмен значениями двух переменных типа `double`. Один из подходов к решению этой задачи состоит в дублировании исходного программного кода с последующей заменой каждого слова `int` словом `double`. Чтобы произвести обмен значениями двух переменных типа `char`, придется повторить эту процедуру. Тем не менее, чтобы выполнить такие незначительные изменения, придется затратить драгоценное время, да при этом еще и не исключена вероятность ошибки.

Средство использования шаблонов функций в языке C++ автоматизирует этот процесс, обеспечивая высокую надежность и экономию времени. Шаблоны функций предоставляют возможность давать определения функций на основе некоторого произвольного типа данных. Например, можно создать шаблон для осуществления обмена значениями, подобный приведенному ниже:

Листинг 4.12.

```
1 template<class Any>
2 void Swap(Any &a, Any &b){
3     Any temp;
4     temp = a;
5     a = b;
6     b = temp;
7 }
```

Первая строка указывает, что устанавливается шаблон, и произвольному типу данных присваивается имя `Any`. В описание обязательно входят ключевые слова `template` и `class`, а также угловые скобки. Имя типа выбирается любое (в примере — `Any`), при этом необходимо соблюдать обычные правила именования языка C++. Остальная часть программного кода описывает алгоритм обмена значениями типа `Any`. Шаблон не создает никаких функций. Вместо этого он предоставляет компилятору указания по определению функции. Если необходимо, чтобы функция осуществляла обмен значениями типа `int`, то компилятор создаст функцию, соответствующую образцу шаблона, подставляя `int` вместо `Any`. Подобно этому, для создания функции, которая производит обмен значениями типа `double`, компилятор будет руководство-

ваться требованиями шаблона, подставляя тип `double` вместо `Any`. Чтобы сообщить компилятору о том, что необходима определенная форма функции обмена значениями, достаточно в программе вызвать функцию `Swap()`. Компилятор проанализирует типы передаваемых аргументов, а затем создаст соответствующую функцию. В листинге 4.13 показано, как это делается.

Листинг 4.13.

```
1 #include <iostream>
2 template<class Any>
3 void Swap(Any &a, Any &b){
4     Any temp;
5     temp = a;
6     a = b;
7     b = temp;
8 }
9
10 void Swap(Any &a, Any &b);
11
12 int main(){
13     using namespace std;
14     int i = 10;
15     int j = 20;
16     cout << "i, j = " << i << ", " << j << ".\n";
17     cout << "Использование сгенерированной компилятором функции int swappe r
        :.\n";
18     Swap(i, j);                // генерирует функцию void Swap(int &, int &)
19     cout << "Теперь i, j = " << i << ", " << j << ".\n";
20     double x = 24.5;
21     double y = 81.7;
22     cout << "x, y = " << x << ", " << y << ".\n";
23     cout << "Использование сгенерированной компилятором функции double
        swappe r:\n";
24     Swap(x, y); // генерирует функцию void Swap(double &, double &)
25     cout << "Теперь x, y = " << x << ", " << y << ".\n";\
26     return 0;
27 }
```

В листинге 4.13 первая функция `Swap()` содержит два аргумента типа `int`, поэтому компилятор генерирует версию функции, предназначенную для обработки

данных типа `int`. Другими словами, он замещает каждое вхождение слова `Any` словом `int`, создавая определение следующего вида:

Листинг 4.14.

```
1 void Swap(int &a, int &b){
2   int temp;
3   temp = a;
4   a = b;
5   b = temp;
6 }
```

Этот код не отображается, но компилятор генерирует его, а затем использует в программе.

Вторая функция `Swap()` содержит два аргумента типа `double`, поэтому компилятор генерирует версию функции, предназначенную для обработки данных типа `double`. Таким образом, он замещает все вхождения слова `Any` словом `double`, генерируя следующий код:

Листинг 4.15.

```
1 void Swap(double &a, double &b){
2   double temp;
3   temp = a;
4   a = b;
5   b = temp;
6 }
```

Выходные данные программы из листинга 4.13 показывают, что она работоспособна:

`i, j = 10, 20.`

Использование сгенерированной компилятором функции `int swapper`:

Теперь `i, j = 20, 10.`

`x, y = 24.5, 81.7.`

Использование сгенерированной компилятором функции `double swapper`:

Теперь `x, y = 81.7, 24.5.`

Следует обратить внимание, что применение шаблонов функций не сокращает размер исполняемого кода. Выполнение программы из листинга 4.13 приводит к созданию двух отдельных определений функции точно так же, как если бы программист

реализовал это вручную. Получаемый в итоге код не содержит шаблонов. Он содержит реальные функции, сгенерированные для программы. Преимущества шаблонов состоит в упрощении процесса создания нескольких определений функции, а также в повышении его надежности.

4.5. Упражнения

Номера и количество заданий, а также способ отчетности определяются преподавателем.

I. Выполнить следующие задания, используя функции, возвращающие значения.

Во всех заданиях вещественные числа выводятся с точностью 2 знака после запятой.

- 1) Создайте функцию, которая для заданных x и n вычисляет значение T_n полинома Чебышева 1-го рода, используя рекуррентное соотношение:

$$T_0(x) = 1;$$

$$T_1(x) = 2 * x - 1;$$

$$T_{n+1}(x) = 2 * (2 * x - 1)T_n(x) - T_{n-1}(x).$$

- 2) Создайте функцию, которая для заданного $n \in N$ вычисляет $T = n^2$, используя лишь аддитивные операции.

- 3) Создайте функцию, которая для заданных x и n вычисляет значение H_n полинома Эрмита, используя рекуррентное соотношение:

$$H_0(x) = 1;$$

$$H_1(x) = 2 * x;$$

$$H_{n+1}(x) = 2 * x * H_n(x) - 2 * n * H_{n-1}(x).$$

- 4) Создайте функцию, которая для заданных x и n вычисляет $\sin nx$. При вычислении используйте следующие рекуррентные соотношения:

$$\sin(n+1)x = 2 * \sin nx \cos x - \sin(n-1)x,$$

$$\cos(n+1)x = 2 * \cos nx \cos x - \cos(n-1)x.$$

- 5) Создайте функцию, которая для заданного $n \in N$ вычисляет $n!!$ ($n!!$ перемножает только числа, имеющие ту же четность, что и n , например, $4!! = 2 \times 4$, $5!! = 1 \times 3 \times 5$).

- 6) Создайте функцию, которая для заданных n и m ($1 \leq n \leq m$) вычисляет количество всех различных размещений из m элементов по n , используя рекуррентное соотношение:

$$A_m^1 = m;$$

$$A_m^n = (m - n + 1)A_m^{n-1}.$$

- 7) Создайте функцию, которая для заданных k и n ($1 \leq k \leq n$) вычисляет биномиальные коэффициенты, используя рекуррентное соотношение:

$$C_n^0 = 1;$$

$$C_n^k = \frac{n+1-k}{k} C_n^{k-1}.$$

- 8) Создайте функцию, которая для заданных x , k и n вычисляет значение H_n полинома Лаггера, используя рекуррентное соотношение:

$$L_0^k(x) = 1;$$

$$L_1^k(x) = x - (k+1);$$

$$L_n^k(x) = (x - k - 2n - 3)L_{n-1}^k(x) - (n-1)(k+n-1)L_{n-2}^k(x).$$

- 9) Создайте функцию, которая для заданных k и n ($1 \leq k \leq n$) вычисляет биномиальные коэффициенты, используя рекуррентное соотношение:

$$C_m^0 = C_m^m = 1;$$

$$C_m^n = 0, \quad (n > m \geq 0);$$

$$C_m^n = C_{m-1}^{n-1} + C_{m-1}^n, \quad (m \geq n > 0).$$

- 10) Создайте функцию, которая для заданных x и k ($x > 0, k \geq 0$) вычисляет значение H_k функции Неймана полуцелого порядка, используя рекуррентное соотношение:

$$N_{-1/2}(x) = \sqrt{2/(\pi x)} \sin x;$$

$$N_{1/2}(x) = -\sqrt{2/(\pi x)} \cos x;$$

$$N_{k-3/2}(x) + N_{k+1/2}(x) = (2/x) \times (k-1/2)N_{k-1/2}(x), \quad (k > 1/2).$$

- 11) Создайте функцию, которая для заданных x и n ($x > 0, n \geq 0$) вычисляет значение H_n функции Бесселя полуцелого порядка, используя рекуррентное соотношение:

$$J_{-1/2}(x) = \sqrt{2/(\pi x)} \cos x;$$

$$J_{1/2}(x) = \sqrt{2/(\pi x)} \sin x;$$

$$J_{n+1/2}(x) = (2(n-1/2)/x) \times J_{n-1/2}(x) - J_{n-3/2}(x), \quad (n > 1/2).$$

- 12) Создайте функцию, которая для заданного $m \in \mathbb{N}$ вычисляет значение полного эллиптического интеграла 1-го рода ($k_0 = 0.5$) (ответ выведите с точностью 16 знаков после запятой):

$$K = \frac{\pi}{2} \prod_{n=1}^m (1 + k_n);$$

$$k_n = \frac{1 - \sqrt{1 - k_{n-1}^2}}{1 + \sqrt{1 + k_{n-1}^2}}.$$

- 13) Создайте функцию, которая методом деления отрезка пополам находит с точностью $\text{EPS}=0.0001$ корень уравнения

$$\cos\left(\frac{1}{x}\right) - 2 * \sin\left(\frac{1}{x}\right) + \frac{1}{x} = 0.$$

(ответ выведите с точностью 4 знаков после запятой).

- 14) Создайте функцию, которая для заданного $n \in \mathbb{N}$ вычисляет:

$$S = \sqrt{1 + 2\sqrt{1 + 3\sqrt{1 + 4\sqrt{1 + \dots + n}}}}.$$

- 15) Создайте функцию, которая вычисляет выражение

$$S = \sqrt{8 - \sqrt{8 + \sqrt{8 - \sqrt{8 - \dots}}}},$$

где число 8 встречается n раз, а знаки перед корнями периодически повторяются группами по три: «-», «+», «-».

- 16) Создайте функцию, которая вычисляет выражение

$$S = \sqrt{23 - 2\sqrt{23 + 2\sqrt{23 + 2\sqrt{23 - \dots}}}},$$

где число 23 встречается n раз, а знаки перед корнями периодически повторяются группами по три: «-», «+», «+».

- 17) Создайте функцию, которая вычисляет для заданного n :

$$S = \sqrt{6 + 2\sqrt{7 + 3\sqrt{8 + 4\sqrt{9 + \dots + \sqrt{n}}}}}$$

- 18) Создайте функцию, которая вычисляет для заданных k и n :

$$S = \sqrt{1 + (n+1)\sqrt{1 + (n+2)\sqrt{1 + (n+3)\sqrt{1 + \dots + (n+k)}}}}.$$

19) Создайте функцию, которая вычисляет :

$$S = \sqrt{11 - 2\sqrt{11 + 2\sqrt{11 - 2\sqrt{11 - 2\sqrt{11 + \dots}}}}}$$

где число 11 встречается n раз, а знаки перед корнями периодически повторяются группами по три: «-», «+», «-».

20) Описать функцию $RootK(X, K, N)$ вещественного типа, находящую приближенное значение корня K -й степени из числа X по формуле:

$$Y_0 = 1;$$

$$Y_{N+1} = Y_N - \frac{(Y_N - X/(Y_N)^{K-1})}{K}.$$

где Y_N обозначает $RootK(X, K, N)$ при фиксированных X и K .

Параметры функции: $X > 0$ — вещественное число, $K > 1$ и $N > 0$ — целые. Вычислить функцию для заданных X , K , N .

II. Выполнить следующие задания, используя функции, не возвращающие значения. Во всех заданиях вещественные числа выводятся с точностью 2 знака после запятой.

- 1) Создайте функцию, печатающую все перестановки чисел от 1 до n , в которой первая цифра m , $1 \leq m \leq n$.
- 2) Создайте функцию, печатающую разложение целого числа n на простые множители.
- 3) Создайте функцию, печатающую всевозможные последовательности, состоящие из трех натуральных чисел, сумма которых равна m .
- 4) Создайте функцию, печатающую в обратном порядке цифры некоторого числа n .
- 5) Создайте функцию, печатающую последовательность ненулевых целых чисел, за которой следует 0, в следующем порядке. Сначала выводятся все отрицательные элементы последовательности, а затем — все положительные (в любом порядке).
- 6) Создайте функцию, печатающую всевозможные последовательности, состоящие из трех натуральных чисел, каждое из которых не превосходит m .
- 7) Напишите функцию, печатающую все натуральные решения неравенства $x^2 + y^2 < n$ для введенного натурального числа n .
- 8) Напишите функцию, вводящую натуральное число R , и печатающую точки с целочисленными координатами внутри замкнутого шара радиуса R с центром в начале координат.

- 9) Создайте функцию, печатающую номера всех «счастливых» билетов длиной 4 символа в диапазоне от m до n . (Билет считается «счастливым», если в нем сумма цифр первой половины номера билета совпадает с суммой цифр второй его половины). Если таких чисел нет, выдать соответствующее сообщение.
- 10) Создайте функцию, печатающую Выведите все «счастливые» билеты длиной 6 символов в диапазоне от m до n . (Билет считается «счастливым» если в нем сумма цифр первой половины номера билета совпадает с суммой цифр второй его половины). Если таких чисел нет, выдать соответствующее сообщение.
- 11) Создайте функцию, печатающую заданное число n в виде $n = 1 + 2 + \dots + k + l$, где $l \leq k$.
- 12) Создайте функцию, печатающую заданное число n в виде $n = 1 * 2 * \dots * k + l$, где k — максимальное из тех, для которых такое соотношение возможно.
- 13) Создайте функцию, печатающую заданное число n в виде $n = 1 - 2 + 3 - 4 + \dots + k$.
- 14) Создайте функцию, печатающую число A в виде $A = a_0 + \dots + a_k E(k) + \dots + a_{m-1} E(m-1)$.
- 15) Создайте функцию, которая для заданного нечетного числа n печатает последовательность из $(n+1)/2$ строк вида:
- $$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & \dots & n-1 & n \\ & 2 & 3 & 4 & \dots & n-1 & \\ & & & & \dots & & \\ & & & & & & (n+1)/2 \end{array}$$
- 16) Создайте функцию, которая для заданного числа n печатает последовательность из $(n+1)/2$ строк вида:
- $$\begin{array}{ccccccc} n & n-1 & & \dots & 2 & 1 & \\ & n-1 & & \dots & 2 & & \\ & & & \dots & & & \\ & & & & & & (n+1)/2 \end{array}$$
- 17) Создайте функцию, которая для заданного нечетного числа n печатает последовательность из $(n+1)/2$ строк вида:
- $$\begin{array}{ccccccc} & & & & \dots & & \\ & 2 & 3 & 4 & \dots & n-1 & \\ 1 & 2 & 3 & 4 & \dots & n-1 & n \end{array}$$

- 18) Создайте функцию, которая для заданного нечетного числа n печатает последовательность из $(n+1)/2$ строк вида:

```

      (n + 1)/2
      ...
      n - 1      ...      2
n      n - 1      ...      2      1

```

- 19) Создайте функцию, которая для заданного нечетного числа n печатает последовательность из $(n+1)/2$ строк вида:

```

**...**      (n звездочек)
*...*        (n-2 звездочка)
...
*

```

- 20) Создайте функцию, которая для заданного нечетного числа n печатает последовательность из $(n+1)/2$ строк вида:

```

*
...
*...*        (n-2 звездочка)
**...**      (n звездочек)

```

III. Создать рекурсивные функции для заданий из пункта I данного раздела.

IV. Для задач из пункта I данного раздела создать 2 перегруженные функции с сигнатурами, аналогичными приведенным в листинге 4.11.

4.6. Контрольные вопросы

1. Математическое понятие функции?
2. Какая функция называется рекурсивной?
3. Всегда ли следует применять рекурсию?
4. Как представить вычисление x^n с помощью рекурсивной функции?
5. Что такое сигнатура функции?
6. В каких случаях компилятор может некорректно определить, какую из перегруженных функций следует вызывать?
7. Каким образом компилятор хранит функции с одинаковыми именами?
8. Вызывает ли использование шаблона сокращение кода программы?

Работа с массивами

Очень часто в процессе программирования возникают ситуации, когда необходимо накапливать и обрабатывать большие объемы информации и использование стандартных типов данных нерационально. Для таких целей используются составные типы данных, т. е., типы, состоящие из стандартных типов данных.

В данной главе будет рассмотрен один из простейших составных типов — массив. Массив — это именованная структура, которая содержит элементы одного типа. Также будет рассмотрен еще один составной тип данных — указатели, которые хранят адрес переменной определенного типа.

5.1. Указатели

До этого в программе всегда объявлялись простые переменные. Оператор объявления определяет тип и символическое имя переменной, а также требует от программы выделить память под эту переменную и определяет скрытым образом ее местоположение. Для того, чтобы определить адрес простой переменной необходимо применить операцию *взятия адреса*: `&<имя переменной>`.

Следующая программа демонстрирует операцию взятия адреса для переменной типа `int`:

Листинг 5.1. Операция взятия адреса переменной

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int a = 5;
6     cout << "value=" << a << " address=" << &a << endl;
7     return 0;
8 }
```

Результат работы программы:

```
value=5 address=0026FE40
```

Результат работы программы — это адрес ячейки памяти, в которой расположен первый байт, занимаемый этой переменной. Обычно операция взятия адреса дает результат в шестнадцатиричной нотации.

Таким образом, значение переменной воспринимается как именованная величина, а ее адрес — как производная. Однако, во многих языках программирования существуют специальные типы переменных, которые в качестве именованной величины хранят не значение переменной, а ее адрес. Такие переменные называются *указателями*. Для получения значения, находящегося в ячейке памяти, адрес которой хранит указатель, используется операция *разыменования*: **⟨имя переменной⟩*. Следующая программа демонстрирует операцию разыменования для указателя *p*:

Листинг 5.2. Пример операции разыменования типов

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int a = 5;
6     int *p = &a;
7     cout << "address: " << p << " value: " << *p << endl;
8     return 0;
9 }
```

Результат работы программы:

address=002EF8D4 value=5

Как видно из листинга 5.2 объявление указателя имеет следующий вид:

$$\langle \text{тип} \rangle * \langle \text{имя переменной} \rangle$$

Например, запись `int *p` означает, что значение переменной `*p` имеет тип `int`, а `p` — это указатель на тип `int`.

Переменная-указатель никогда не бывает просто указателем, она всегда указывает на какой-нибудь тип. Указатели могут указывать на различные типы данных, имеющие разный размер, но сами указатели в памяти обычно занимают 2 или 4 байта.

Необходимо помнить, что при объявлении указателей автоматически выделяется память только для записи адреса, но выделения памяти под хранение значения, расположенного по этому адресу, не происходит.

Если указатель не инициализировать, то в поле для записи адреса будет записано случайное шестнадцатиричное число, которое может означать любую ячейку памяти, в том числе, занятую системными данными. Попытка изменить данные по этому адресу может привести к непредсказуемым последствиям.

Способов инициализации несколько:

- Как показано в листинге 5.2, сначала инициализируется переменная определенного типа, затем объявляется указатель на адрес этой переменной:

```
int a = 5; //переменная типа int
int *p = &a //указатель на адрес переменной типа int
```

- Указателю присваивается адрес в явном виде, но необходимо точно знать, что нужная ячейка памяти свободна:

```
int *p;
p = (int *)0x002EF8D4; //указан адрес ячейки памяти
```

- Используется нулевой указатель (null-указатель). Стандарт языка C++ гарантирует, что нулевой указатель никогда не указывает на корректные данные, поэтому он часто используется в качестве признака окончания каких-либо действий:

```
int *p = NULL; //null-указатель или
int *p = 0;
```

- Используется операция выделения памяти **new**. С помощью операции выделения памяти, память выделяется не на стадии компиляции, а на стадии выполнения программы, что позволяет экономить память при условии, если размер и количество данных заранее неизвестны. Общий вид:

$$\langle \text{тип} \rangle * \langle \text{имя указателя} \rangle = \text{new } \langle \text{тип} \rangle$$

Поскольку память выделяется программистом на стадии выполнения программы, то и освобождается память тоже программистом с помощью операции **delete**. Например, объявляется указатель на тип **int** и выделяется память под переменную типа **int**, выполняются какие-либо действия и освобождается память:

Листинг 5.3.

```
1 int main(){
2     ...
3     int *p = new int;
4     //операторы
5     delete p;
6     ...
7 }
```

5.2. Одномерные массивы

Массив — это структура данных, которая содержит множество значений, относящихся к одному и тому же типу. Тип может быть любым, как базовым, так и составным. Каждое значение сохраняется в отдельном элементе массива, и компьютер сохраняет все элементы в памяти последовательно, друг за другом.

Существует два способа представления массивов: *статические*, когда память под элементы массива выделяется на стадии компиляции и необходимо заранее знать размер массива, и *динамические*, когда память выделяется программистом с помощью команды `new` на стадии выполнения программы.

Рассмотрим сначала статические массивы.

5.2.1. Статические одномерные массивы

Статический одномерный массив — это последовательность элементов одного типа, размер которого должен быть известен заранее. Каждое значение хранится в отдельном элементе, и в памяти компьютера элементы массива располагаются последовательно, один за другим.

Для создания массива должно быть известно три составляющих: тип элементов, имя массива и количество элементов в массиве.

Формат объявления массива:

$$\langle \text{тип} \rangle \langle \text{имя массива} \rangle [\langle \text{размерность массива} \rangle] .$$

В отличие от остальных случаев, в этом представлении квадратные скобки являются обязательными, а не обозначают необязательные элементы.

Например, `int a[5]` — массив целых чисел, состоящих из пяти элементов. К каждому элементу массива можно обратиться с помощью индекса (порядковый номер элемента), т. е., первый элемент массива — `a[0]`, второй — `a[1]`, последний — `a[4]`.

Нумерация элементов массива начинается с нуля. Следовательно, последний элемент массива, состоящего из `n` элементов, имеет индекс `a[n-1]`. Язык C++ не проверяет правильность вводимого индекса элемента массива, поэтому возможны ошибки при попытке работы с несуществующим элементом, например, вызывается элемент `a[5]` массива, который содержит всего 5 элементов (от `[0]` до `a[4]`). Такая проверка остается за программистом. Об этом необходимо помнить при работе с массивами.

С массивами можно работать только ПОЭЛЕМЕНТНО. Их нельзя присвоить друг другу, с массивами нельзя выполнять никаких операций, можно работать только с элементами массивов.

Возможно несколько вариантов объявления и инициализации массива:

1. С помощью одновременно объявления и инициализации массива:

```
int a[5] = {1, 3, 8, 9, 7};
```

Массив содержит 5 элементов:

{1, 3, 8, 9, 7}.

2. Можно инициализировать только часть элементов, тогда остальным элементам будет присвоено значение 0:

```
int a[5] = {1, 2, 3};
```

Массив содержит 5 элементов:

{1, 2, 3, 0, 0}.

Тогда можно легко обнулить элементы массива:

```
int a[5] = {0};
```

Первому элементу будет присвоено значение 0, следуя операции присваивания, остальным — по умолчанию.

3. Если оставить квадратные скобки пустыми и инициализировать массив, то компилятор подсчитает количество элементов самостоятельно:

```
int a[] = {1, 2, 3, 5};
```

Результатом будет массив, состоящий из четырех элементов. Не стоит использовать этот способ слишком часто, так как компилятор может определить размер массива непредсказуемо.

4. Заполнить массив поэлементно, используя оператор цикла и вводя элементы самостоятельно:

```
for (int i = 0; i < n; i++){
    cout << "a[" << i << "]=";
    cin >> a[i];
}
```

5. Заполнить массив поэлементно, используя оператор псевдослучайных чисел:

Листинг 5.4. Использование генератора псевдослучайных чисел

```
1 #include<iostream>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <time.h>
5 using namespace std;
6
7 int main(){
8     int a[10], n;
9     cout << "n="; cin >> n;           // число элементов массива
10    srand ((unsigned int)time(NULL)); // начальная точка генерации
11    for (int i = 0; i < n; i++){
12        a[i] = rand() % 15;           //псевдослучайное число
13        cout <<"a[" << i << "]=" << a[i] << endl;
14    }
15    return 0;
16 }
```

Результат работы программы:

```
n = 8
a[0] = 3
a[1] = 6
a[2] = 9
a[3] = 11
```

```
a[4] = 6
a[5] = 12
a[6] = 9
a[7] = 8
```

Функция `rand()` определяет псевдослучайное число из диапазона `[0, MAX RAND]` (`MAX RAND=32767`). Элементом массива в листинге 5.4 является остаток от деления соответствующего числа на 15. Для вызова функции `rand()` необходима стартовая точка. Она генерируется с помощью функции `srand()`. Если не использовать эту функцию, то при каждом следующем запуске программы будет генерироваться та же самая последовательность чисел, что и при первом запуске. Функция `time(NULL)` определяет количество секунд, прошедших с 01.01.1970 до текущего времени процессора. Естественно, что это время при новом запуске будет другим, следовательно, будет сгенерирована другая последовательность.

5.2.2. Динамические одномерные массивы

При работе со статическими массивами размер массива должен быть определен на стадии компиляции, т. е., если для решения какой-то задачи заранее определили, что массив состоит из 200 элементов, а реально используется только 10, следовательно, будет выделена излишняя память, и наоборот, если изначально определили, что массив состоит из 10 элементов, а используется 200, то произойдет ошибка выполнения. Поэтому проще работать с динамическими массивами, память под которые выделяется на этапе выполнения программы.

Создать динамический массив достаточно просто с помощью операции `new`.

Для этого необходимо создать указатель определенного типа, и, используя `new`, указать тип элементов и количество таких элементов в квадратных скобках:

```
int *mas = new int [10];
```

В данном примере указатель `mas` возвращает адрес нулевого элемента массива, а всего память выделяется под 10 элементов типа `int`.

Если массив был создан с помощью операции `new`, соответственно необходимо очистить память с помощью операции `delete`:

```
delete [] mas;
```

Квадратные скобки перед указателем означают, что необходимо очистить всю память, выделенную под массив, а не только удалить нулевой элемент.

Общая форма выделения и присваивания памяти для динамического массива выглядит следующим образом:

$$\langle \text{тип} \rangle * \langle \text{имя указателя} \rangle = \text{new } \langle \text{тип} \rangle [\langle \text{число элементов} \rangle] .$$

Вызов операции **new** выделяет достаточно большой блок памяти, необходимый для того, чтобы в нем уместилось **число элементов** элементов типа **тип** и устанавливает в **имя указателя** указатель на нулевой элемент массива.

Как обращаться к элементам динамического массива? Элементы массива в памяти компьютера расположены подряд, поэтому возможны два варианта: через *арифметику указателей* и через *индексирование массива*.

Рассмотрим арифметику указателей. Указатель определяет адрес нулевого элемента массива, предположим типа **int**. Если сдвинуть указатель на 4 байта, то он будет указывать уже на первый элемент массива. Арифметика указателей в данном случае заключается в том, что увеличение указателя на единицу, означает сдвиг на столько байт, сколько занимает элемент соответствующего типа (см. рисунок 5.1).

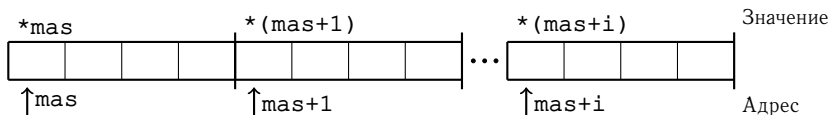


Рис. 5.1. Схематическое представление одномерного массива в памяти компьютера

Пример 5.1. Дан динамический массив типа **int**, состоящий из пяти элементов:

Листинг 5.5. Работа с динамическим массивом

```
1 #include<iostream>
2 using namespace std;
3
4 int main (){
5     int n = 5, i = 1;
6     int *mas = new int [n];
7     int *p;
8     p = (mas + n);
```

// указатель на адрес памяти,


```

9  cout << "address p=" << p << endl;           // следующий за
10 while (mas != p){                               // последним элементом
11     *mas = i;                                    // значение элемента массива
12     cout << "mas[" << i-1 << "]" << *mas << endl;
13     cout<< "address " << mas <<endl;
14     mas++;                                       // указатель на следующий элемент
15     i++;                                       // увеличиваем значение i
16 }
17 mas = mas - n;                                //возврат указателя на 0-ой элемент
18 delete [] mas;
19 return 0;
20 }

```

Результат работы программы:

```

address p = 00141DA4
mas[0] = 1
address p = 00141D90
mas[1] = 2
address p = 00141D94
mas[2] = 3
address p = 00141D98
mas[3] = 4
address p = 00141D9C
mas[4] = 5

```

В строке 6 листинга 5.5 объявлен массив типа `int`, состоящий из пяти элементов. Указатель `p`, объявленный в строке 7 и равный `mas + n` (строка 8), определяет адрес памяти, следующий за последним элементом массива. До тех пор, пока `mas` не равен `p`, выполняется цикл (строки 10 — 16):

1. Элементу, на который указывает `mas`, присваивается значение `i` (строка 11).
2. Выводится значение элемента и его адрес на экран (строки 12–13).
3. Увеличивается указатель на единицу, т. е., адрес увеличивается на 4 байта (строка 14).
4. Увеличивается значение `i` на единицу (строка 15).

После окончания цикла указатель `mas` возвращается на нулевой элемент, т. е., адрес уменьшается на `n * sizeof(int)` байт (строка 17), и очищается память, выделенная под массив (строка 18).

Как можно видеть в результате, элементы массива действительно располагаются последовательно, каждый элемент занимает 4 байта. □

Таким образом, к `i`-тому элементу массива можно обратиться следующим образом: `*(mas+i)`;

Скобки обязательны, поскольку приоритет операции разыменования (*) выше, чем операции сложения. Поэтому запись `*mas+i` означает, что к значению текущего элемента `mas` добавляется значение `i`, а запись `*(mas+i)` означает, что сначала указатель сдвигается на `i * sizeof(тип)` байт, а потом используется значение элемента, на который указывает указатель после сдвига.

Например, если массив имеет вид: `int a[5] = {5 4 3 2 1}`, то результатом выражения `*mas+3` будет 8, так как `*mas` содержит адрес нулевого элемента массива (его значение равно 5). Результатом выражения `*(mas+3)` будет 2, так как сначала увеличивается адрес на 12 байт (3*4 байта, занимаемыми типом `int`), а только потом определяется значение переменной, расположенной по данному адресу.

Рассмотрим теперь работу с массивами через индексацию массивов. В данном случае, работа с динамическим массивом ничем не отличается от работы со статическим, только имя указателя используется как имя массива:

Листинг 5.6. Индексация одномерного динамического массива

```
1 #include<iostream>
2 using namespace std;
3
4 int main (){
5     int n = 5, i = 1;
6     int *mas = new int [n];
7     for (int i = 0; i < n; i++) {
8         mas[i] = i + 1;           //заполняем i элемент
9         cout << "mas[" << i << "]=" << mas[i] << endl;
10        cout<< "adress " << &(mas[i]) <<endl;
11    }
12    delete [] mas;
13    return 0;
```

В строке 9 листинга 5.6 на экран выводится значение i -ого элемента массива, в строке 10 — выводится адрес элемента с помощью операции взятия адреса (&).

Как можно легко проверить, результат работы программы листинга 5.6 будет таким же, как и в случае листинга 5.5, за исключением адресов памяти (при повторном запуске может быть выделен другой участок памяти).

Т. о., массивы и указатели взаимозаменяемы. На самом деле, в случае работы с массивами с помощью индексов, при запуске программы сначала происходит неявный переход к указателям, а потом доступ к элементам массива происходит с использованием библиотеки указателей.

Т. е., работа с массивами проще для понимания и чтения кода, работа с указателями быстрее по времени исполнения.

5.2.3. Примеры работы с одномерными массивами

Работа с одномерными массивами в итоге сводится к нескольким общим задачам:

1. Поиск суммы (произведения, разности и т. д.) элементов, попадающих (не попадающих) в заданный интервал.
2. Поиск минимума (максимума) элементов массива.
3. Замена значений элементов, удовлетворяющих некоторому условию.

Если надо заменить значение элемента, удовлетворяющего условию, например, нулем, то оператор выглядит следующим образом: `a[i] = 0`.

Если нужно поменять местами значения двух элементов, то можно воспользоваться встроенной функцией `swap`: `swap(a[i], a[j])`.

Во всех случаях рассматриваются именно значения элементов. Если необходимо работать с индексами элементов, в условии задачи будет особо выделено, что речь идет именно об индексах (Например, найти сумму элементов с четными порядковыми номерами.)

Поиск элементов, попадающих (не попадающих) в заданный интервал

Условие поиска элементов, попадающих (не попадающих) в заданный интервал $[a, b]$: необходимо определить удовлетворяет ли ЗНАЧЕНИЕ ЭЛЕМЕНТА массива следующим неравенствам:

- Если элемент массива попадает в заданный интервал:

$$a \leq mas[i] \leq b.$$

Условие, записанное на языке C++:

```
if (mas[i] >= a && mas[i] <= b)...
```

- Если элемент массива не попадает в заданный интервал:

$$mas[i] \leq a \vee mas[i] \geq b.$$

Условие, записанное на языке C++:

```
if (mas[i] < a || mas[i] > b)...
```

Пример 5.2. Дан массив, содержащий целые числа. Найти среднее арифметическое элементов, кратных трем, не попадающих в заданный интервал $[a, b]$. Если таких чисел нет, вывести сообщение об этом.

Например,

n	Массив	Диапазон	Результат
4	9, 4, 6, 3	[4, 7]	6
4	8, 4, 6, 2	[4, 7]	таких элементов нет

Листинг 5.7. Код программы

```
1 #include<iostream>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <time.h>
5 using namespace std;
6
7 int main(){
8     setlocale (LC_ALL,"rus");
9     int n;
```

//вывод данных русском языке

```

10  cout << "n="; cin >> n;                                //размер массива
11  int *a = new int [n];                                   //выделение памяти под массив
12  srand ((unsigned)time(NULL));                           // начальная точка генерации
13  for (int i = 0; i < n; i++){
14      a[i] = rand() % 15;                                //псевдослучайное число
15      cout << a[i] << " ";
16  }
17  cout << endl;
18  int b,c;
19  cout << "b="; cin >> b;                                //ввод диапазона
20  cout << "c="; cin >> c;
21  if (b > c) swap(b,c);                                   //замена значений, если b > c
22  int k = 0;
23  int sum = 0;
24  for (int i = 0; i < n; i++)
25      if (a[i]%3 == 0 && ( a[i] < b || a[i] > c)){ //поиск элементов
26          k++;                                           //количество элементов
27          sum += a[i];                                   //сумма значений элементов
28      }
29  if (!k) cout << "Таких элементов нет\n"; //k = 0
30  else
31      cout << "Среднее арифметическое: " << 1.0*sum/k << endl;
32  return 0;
33 }

```

Рассмотрим работу программы 5.7 для первого примера:

i	$a[i]$	строка 25 листинга 5.7	k	S
н.у.			0	0
0	9	true	1	9
1	4	false (попадает в интервал)	1	9
2	6	false (кратно трем, но попадает в интервал)	1	9
3	3	true	2	12

В итоге среднее арифметическое равно 6.

□

Поиск минимальных (максимальных) элементов массива

Для того, чтобы определить экстремумы массива, необходимо сравнивать каждый элемент с текущим экстремумом, и, если значение элемента меньше (больше) текущего экстремума, присвоить текущему экстремуму значение этого элемента. Т. е.:

```
if (mas[i] < min) min = mas[i];
```

Для того, чтобы начать сравнение, необходимо присвоить начальное значение переменной `min`. Возможны два варианта:

- Если надо найти минимум или максимум в массиве, то переменной `min` присваивается значение первого элемента массива, т. е., `min = a[0]`.

Поскольку в массиве обязательно существует и минимальное и максимальное значение элементов (в случае, если массив состоит из одинаковых элементов, то его минимум и максимум совпадают), такое присваивание позволяет гарантировано определить искомые экстремумы. Например, фрагмент кода программы 5.8 позволяет найти минимальное и максимальное значение в массиве:

Листинг 5.8. Поиск экстремумов массива

```
1 ...
2 int min = a[0];
3 int max = a[0];
4 for (int i = 1; i < n; i++){
5     if (a[i] < min)
6         min = a[i];
7     if (a[i] > max)
8         max = a[i];
9 }
```

Рассмотрим массив `a[4] = {1, 5, 0, 9}`:

i	$a[i]$	строка 6 листинга 5.8	min	строка 8 листинга 5.8	max
0	1		1		1
1	5	false	1	true	5
2	0	true	0	false	5
3	9	false	0	true	9

- В случае, если необходимо найти экстремум из элементов, удовлетворяющих некоторому условию (например, максимум из четных элементов), нельзя присваивать начальному значению переменной `min` (или `max`) значение первого элемента.

Например, надо найти максимум из четных элементов массива.

Пусть есть следующий массив: $a[4] = \{9, 2, 8, 1\}$. Пусть начальное значение переменной $\text{max} = a[0]$.

Рассмотрим пошаговую реализацию следующего фрагмента кода программы:

Листинг 5.9. Поиск максимума четных элементов массива

```
1 ...
2 int max = a[0];
3 for (int i = 1; i < n; i++){
4     if (!(a[i] % 2) && a[i] > max)    //если число четное и > max
5         max = a[i];
6 }
```

i	$a[i]$	строка 4 листинга 5.9	max
0	9		9
1	2	false (четное, но меньше max)	9
2	8	false (четное, но меньше max)	9
3	1	false (нечетное)	9

В итоге получили неправильный ответ.

В таком случае, самый простой вариант выбирать в качестве начального значения переменных предельные значения для соответствующего типа, например, если необходимо найти максимум, то в качестве начального значения переменной max надо выбрать минимально возможное значение для этого типа. Тогда любое значение элементов массива было гарантировано больше начального значения переменной max .

Для этого необходимо подключить библиотеку `<limits.h>`, в которой хранятся предельные значения для порядковых типов данных на данном компиляторе или `<float.h>`, в которой хранятся предельные значения для порядковых типов данных на данном компиляторе.

Значения записаны во встроенных переменных, их имена можно посмотреть в справочнике. Например, для типа `int` предельные значения хранятся в переменных `INT_MIN` и `INT_MAX`. Тогда листинг 5.9 будет иметь вид:

Листинг 5.10. Поиск максимума четных элементов массива

```
1 #include<limits.h>
```

```

2 ...
3 int max = INT_MIN;
4 for (int i = 0; i < n; i++){
5     if (!(a[i] % 2) && a[i] > max)
6         max = a[i];
7 }

```

i	$a[i]$	строка 6 листинга 5.9	max
нач. усл.			-2147483648
0	9	false (нечетное)	-2147483648
1	2	true	2
2	8	true	8
3	1	false (нечетное)	8

Ответ: `max = 8`.

Пример 5.3. Дан массив, содержащий целые числа. Заменить все минимальные элементы нулем.

Листинг 5.11. Код программы

```

1 #include<iostream>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <time.h>
5 using namespace std;
6
7 int main(){
8     setlocale (LC_ALL,"rus");           //вывод данных на русском языке
9     int n;
10    cout << "n="; cin >> n;              //размер массива
11    int *a = new int [n];                //выделение памяти под массив
12    srand ((unsigned)time(NULL));         // начальная точка генерации
13    for (int i = 0; i < n; i++){
14        a[i] = rand() % 15;              //псевдослучайное число
15        cout << a[i] << " ";
16    }
17    cout << endl;
18    int min = a[0];                       //минимум = нулевой элемент
19    for (int i = 1; i < n; i++)
20        if (a[i] < min) min = a[i];       //поиск минимума

```



```

21  for (int i = 0; i < n; i++)
22      if (a[i] == min) a[i] = 0;           //заменяем нулем минимумы
23  for (int i = 0; i < n; i++)             //вывод на экран нового массива
24      cout << a[i] << " ";
25  cout << endl;
26  return 0;
27  }

```

Результаты работы программы:

<i>n</i>	массив	результат
10	8, 14, 8, 7, 8, 4, 2, 2, 8, 2	1, 3, 8, 7, 14, 4, 7, 2, 10, 1
10	8, 14, 8, 7, 8, 4, 0, 0, 8, 0	0, 3, 8, 7, 14, 4, 7, 2, 10, 0

□

Пример 5.4. Вывести номера максимальных четных элементов.

Листинг 5.12. Код программы

```

1  #include<iostream>
2  #include<math.h>
3  #include<limits.h>
4  using namespace std;
5
6  int main(){
7      int n;
8      cout << "n ="; cin >> n;           //ввод размерности
9      int *a = new int [n];
10     for (int i = 0; i < n; i++){         //заполняем массив элементами
11         cout << "a[" << i << "]=";
12         cin >> a[i];
13     }
14     int max = INT_MIN;
15     for (int i = 0; i < n; i++)           //поиск максимального четного элемента
16         if (!(a[i] % 2) && a[i] > max)
17             max = a[i];
18     if (max == INT_MIN)                   //четных элементов в массиве нет
19         cout << "четных элементов нет\n";
20     else
21         for (int i = 0; i < n; i++)       //выводим на экран номера max
22             if (a[i] == max)
23                 cout << i << " ";

```

```
24     cout << endl;
25     return 0;
26 }
```

Результат работы программы:

<i>n</i>	массив	результат
10	8, 9, 8, 7, 8, 4, 2, 2, 8, 2	0, 2, 4, 8
10	7, 9, 3, 7, 5, 1, 9, 7, 7, 3	четных элементов нет



5.3. Двумерные массивы

Рассмотрим теперь работу с двумерными массивами. Это массив, который удобно представлять в виде таблицы, состоящей из строк и столбцов. Тогда элемент массива с индексами *i* и *j* находится на пересечении *i*-ой строки и *j*-ого столбца массива. Двумерные массива также бывают статические и динамические.

5.3.1. Статические двумерные массивы

Описание статического двумерного массива в общем случае имеет вид:

⟨тип⟩ ⟨имя массива⟩ [⟨число строк⟩] [⟨число столбцов⟩].

Например, запись вида `int mas[10][10];` означает, что создан массив, состоящий из 10 строк и 10 столбцов, элементы которого целые числа.

Работа с массивами происходит поэлементно. Так, для того, чтобы создать массив, заполнить его и вывести на экран, необходимо использовать вложенные циклы:

Листинг 5.13. Пример работы с двумерным статическим массивом

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int mas[100][100];    //массив из 100 строк и 100 столбцов
6     int n;
7     cout << "n="; cin >> n; //ввод размерности массива
8     /*-----ввод массива-----*/
9     for (int i = 0; i < n; i++)
10        for (int j = 0; j < n; j++){
```

```

11     cout << "mas[" << i << "]"[" << j << "]=";
12     cin >> mas[i][j];
13 }
14 /*-----ВЫВОД массива-----*/
15 cout << "Array:\n";
16 for (int i = 0; i < n; i++, cout << endl)
17     for (int j = 0; j < n; j++)
18         cout << mas[i][j] << " ";
19 return 0;
20 }

```

Результат работы программы:

```

n = 3
mas[0][0]=1
mas[0][1]=2
mas[0][2]=3
mas[1][0]=4
mas[1][1]=5
mas[1][2]=6
mas[2][0]=7
mas[2][1]=8
mas[2][2]=9
matrix:
1 2 3
4 5 6
7 8 9

```

Ввод вручную элементов двумерного массива занимает достаточно времени и этим способом лучше пользоваться только в случае, когда необходимо ввести массив специфическим образом, например, все строки состоят из одинаковых чисел и т. д. Лучше пользоваться генератором псевдослучайных чисел, по аналогии с одномерным массивом (см. раздел 5.2.1):

Листинг 5.14. Ввод двумерного массива с помощью генератора псевдослучайных чисел

```

1 #include<iostream>

```

```

2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <time.h>
5 using namespace std;
6
7 int main (){
8     int a[10][10],n;
9     cout << "n="; cin >> n;                // число элементов массива
10    srand (((unsigned)time(NULL)));          // начальная точка генерации
11    for (int i = 0; i < n; i++, cout << endl)
12        for (int j = 0; j < n; j++){
13            a[i][j] = rand() % 15;           //псевдослучайное число
14            cout << a[i][j] << " ";
15        }
16    return 0;
17 }

```

Результат работы программы:

```

n = 5
0 12 14 7 12
14 1 7 0 6
1 0 1 2 8
5 1 11 9 7
3 4 13 12 12

```

Как видно из листингов 5.13–5.14 для объявления двумерного статического массива необходимо сразу определить количество строк и столбцов массива. Следовательно, память выделяется под массив существенно большего размера, чем реально необходимо.

На самом деле двумерный статический массив в памяти представляется как массив одномерный: сначала записываются элементы нулевой строки, потом первой и т. д. Получается, что память расходуется нерационально, поэтому лучше использовать динамические двумерные массивы, память под который выделяется на этапе работы программы самим программистом.

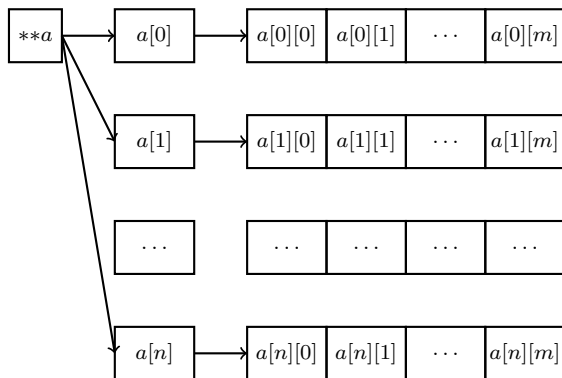


Рис. 5.2. Представление двумерного динамического массива в памяти компьютера

5.3.2. Двумерные динамические массивы

Динамический двумерный массив расположен в памяти более рационально. Как видно на рисунке 5.2, двумерный динамический массив можно трактовать как набор отдельных одномерных динамических массивов, которые могут быть расположены в любом свободном месте памяти, необязательно подряд. При таком подходе каждая строка двумерного массива может содержать разное количество элементов, что абсолютно неприемлемо для статического массива.

Другими словами, динамический двумерный массив — это указатель на массив указателей, каждый из которых в свою очередь указывает на одномерный динамический массив.

Работать с двумерными динамическими массивами также можно через арифметику указателей и через индексирование. Арифметика указателей для двумерного массива достаточно сложна, поэтому в будущем будем пользоваться индексами. Арифметику указателей можно рассмотреть самостоятельно.

Объявление двумерного динамического массива имеет свои особенности. Так как речь идет об указателях, то сначала с помощью оператора `new` выделяется память под массив указателей:

$$\langle \text{тип} \rangle **\langle \text{имя массива} \rangle = \text{new } \langle \text{тип} \rangle * [\langle \text{число строк} \rangle].$$

Затем для каждого элемента одномерного массива указателей выделяется память под соответствующий массив:

```

for(i = 0; i < <число строк>; i++)
    <имя массива>[i] = new <тип> [<число столбцов>].

```

И только после этого можно заполнять массив. Основное преимущество использования двумерного динамического массива — это возможность задавать размерность массива в ходе выполнения программы, а не определять ее заранее.

Ниже приведен пример объявления и инициализации двумерного динамического массива:

Листинг 5.15. Пример работы с двумерным динамическим массивом

```

1 #include<iostream>
2 #include<time.h>
3 #include<stdlib.h>
4 using namespace std;
5
6 int main(){
7     int n, m;
8     cout << "n = "; cin >> n;      //размерность массива
9     cout << "m = "; cin >> m;
10    int **a = new int *[n];        //выделение памяти под массив указателей
11    for (int i = 0; i < n; i++)
12        a[i] = new int [m];        //выделение памяти для каждого указателя
13    srand((unsigned)time(NULL));
14    for (int i = 0; i < n; i++, cout << endl)
15        for (int j = 0; j < m; j++){
16            a[i][j] = rand() % 15;   //заполнение элемента массива
17            cout << a[i][j] << " "; //вывод на экран
18        }
19    return 0;
20 }

```

Если запустить листинг 5.15, то можно увидеть, что результат работы программы будет похожим на результат работы со статическим двумерным массивом (листинг 5.14).

Примером работы с динамическим двумерным массивом, каждая строка которого содержит различное количество элементов, может служить треугольник Паскаля.

Под треугольником Паскаля подразумевается бесконечный треугольник, состоящий из целых чисел. В каждой строке первый последний элемент равен единице,

остальные — сумме элементов, находящихся над ним. На рисунке 5.3 а) представлены первые 4 строки треугольника Паскаля. Треугольник Паскаля применяется во многих областях алгебры чисел и комбинаторики (например, с его помощью можно определить число сочетаний, числа Фибонначчи, степень числа 2 и многое другое.)



Рис. 5.3. Треугольник Паскаля, представленный в а) обычной и б) прямоугольной форме

Треугольник Паскаля можно переписать в другом виде (см. рисунок 5.3 б). В таком случае треугольник Паскаля можно представить в виде динамического двумерного массива.

Каждая i -ая строка содержит $i+1$ элементов. Элементы массива с номерами $[i][0]$ и $[i][i]$ равны единице, элемент с номером $[i][j]$ равен сумме элементов с номерами $[i-1][j-1]$ и $[i-1][j]$. Нумерация строк и столбцов начинается с нуля.

В листинге 5.16 приведен код программы, реализующий ввод и вывод треугольника Паскаля длины n . Можно запустить эту программу и убедиться, что в результате получается треугольник, изображенный на рисунке 5.3

Листинг 5.16. Создание треугольника Паскаля

```

1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int n;
6     cout << "n = "; cin >> n;           // кол-во строк треугольника
7     int **a = new int *[n + 1];
8     for (int i = 0; i <= n; i++)
9         a[i] = new int [i + 1];         //выделяем память под i-ую строку
10    /*****заполнение массива*****/

```

```

11  a[0][0] = 1;
12  a[1][0] = a[1][1] = 1;
13  for (int i = 2; i <= n; i++){
14      a[i][0] = 1;
15      for (int j = 1; j < i; j++)
16          a[i][j] = a[i-1][j-1] + a[i-1][j];
17      a[i][i] = 1;
18  }
19  /*****Вывод массива*****/
20  for (int i = 0; i <= n; i++, cout << endl)
21      for (int j = 0; j <= i; j++)
22          cout << a[i][j] << " ";
23  return 0;
24  }

```

5.4. Функции и массивы

5.4.1. Одномерные массивы

Функции могут служить инструментами и для обработки более сложных типов, таких как массивы и структуры.

Допустим у нас есть некий талантливый программист, работающий сразу в четырех организациях. У него есть данные по каждой организации, сколько он там в какой месяц заработал. Теперь он хочет узнать суммарную, максимальную и минимальную сумму, полученную им в каждом месяце. В листинге 5.17 приведен соответствующий код.

Листинг 5.17.

```

1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      double work1[12], work2[12], work3[12], work4[12], max[12], min[12], sum[12];
6      cout << "Введите данные по первой работе\n";
7      for (int i = 0; i < 12; i++){
8          cout << i + 1 << "-й месяц - ";
9          cin>>work1[i];
10     }

```



```

11  cout << "Введите данные по второй работе\n";
12  for (int i = 0; i < 12; i++){
13      cout << i + 1 << "-й месяц - ";
14      cin >> work2[i];
15  }
16  cout << "Введите данные по третьей работе\n";
17  for (int i = 0; i < 12; i++){
18      cout << i + 1 << "-й месяц - ";
19      cin >> work3[i];
20  }
21  cout << "Введите данные по четвертой работе\n";
22  for (int i = 0; i < 12; i++){
23      cout << i + 1 << "-й месяц - ";
24      cin >> work4[i];
25  }
26  for (int i = 0; i < 12; i++){
27      max[i] = work1[i] > work2[i] ? work1[i] : work2[i];
28      if (max[i] < work3[i]) max[i] = work3[i];
29      if (max[i] < work4[i]) max[i] = work4[i];
30      min[i] = work1[i] < work2[i] ? work1[i] : work2[i];
31      if (min[i] > work3[i]) min[i] = work3[i];
32      if (min[i] > work4[i]) min[i] = work4[i];
33      sum[i] = work1[i] + work2[i] + work3[i] + work4[i];
34  }
35  cout << "Суммарная зарплата\n";
36  cout << "_____ \n";
37  cout << "| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10| 11| 12|\n";
38  cout << "_____ \n";
39  cout << ' | ' ;
40  for (int i = 0; i < 12; i++){
41      cout.width(5);
42      cout << sum[i] << ' | ' ;
43  }
44  cout << endl;
45  cout << "_____ \n";
46  cout << "максимальная зарплата\n";
47  cout << "_____ \n";
48  cout << "| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10| 11| 12|\n";
49  cout << "_____ \n";

```

```
50  cout << ' | ';\n";
51  for (int i = 0; i < 12; i++){
52      cout.width(5);
53      cout << max[i] << ' | ';\n";
54  }
55  cout << endl;
56  cout << "_____\\n";
57  cout << "минимальная зарплата\\n";
58  cout << "_____\\n";
59  cout << "| 1| 2| 3| 4| 5| 6| 7| 8| 9| 10| 11| 12|\\n";
60  cout << "_____\\n";
61  cout << ' | ';\n";
62  for (int i = 0; i < 12; i++){
63      cout.width(5);
64      cout << min[i] << ' | ';\n";
65  }
66  cout << endl;
67  cout << "_____\\n";
68  return 0;
69 }
```

В результате работы для некоторого набора входных данных программа, например, выведет:

Суммарная зарплата

	1	2	3	4	5	6	7	8	9	10	11	12
	3	4	5	6	7	8	9	10	11	12	13	14

максимальная зарплата

	1	2	3	4	5	6	7	8	9	10	11	12
	1	1	2	3	4	5	6	7	8	9	10	11

минимальная зарплата

	1	2	3	4	5	6	7	8	9	10	11	12
--	---	---	---	---	---	---	---	---	---	----	----	----

Очевидно, что использование функций для ввода и вывода элементов массива сильно уменьшило бы размеры программ и облегчило работу программиста.

Итак, требуется создать функцию, в которую можно было бы передать массив в качестве параметра. Возьмем следующее объявление функции:

```
void input_arr (char* title, int arr[ ], int n).
```

Выглядит вполне прилично. Квадратные скобки указывают на то, что `arr` — массив, а тот факт, что они пусты, говорит о том, что эту функцию можно применять с массивами любого размера. Но бывает, что некоторые вещи не являются тем, чем они кажутся: `arr` — на самом деле не массив, а указатель! Однако этот факт никак не повлияет на код программы, его можно писать так, как если бы `arr` все-таки был массивом. Во-первых, убедимся, что такой подход работает, а потом разберемся, почему он работает.

В листинге 5.18 приведен код решаемой задачи, но с использованием функций ввода и вывода. Запустив его легко убедиться, что при том же наборе входных данных результат будет идентичный уже приведенному. Только код стал в полтора раза меньше.

Листинг 5.18.

```
1 #include <iostream>
2 using namespace std;
3
4 void printline (int n){
5     for (int i = 0; i < n; i++)
6         cout << "_____";
7     cout << ' ' << endl;
8 }
9
10 void input_arr (char* title, int arr[ ], int n){
11     cout << title << endl;
12     for (int i = 0; i < n; i++){
13         cout << i << "-й месяц - ";
14         cin >> arr[i];
15     }
16 }
```

```

17
18 void output_arr (char* title, int arr[ ], int n){
19     cout << title << endl;
20     printline (n);
21     cout << ' | ' ;
22     for (int i = 0; i < n; i++){
23         cout.width(5);
24         cout << i + 1 << ' | ' ;
25     }
26     cout << endl;
27     printline (n);
28     cout << ' | ' ;
29     for (int i = 0; i < n; i++){
30         cout.width(5);
31         cout << arr[i] << ' | ' ;
32     }
33     cout << endl;
34     printline (n);
35 }
36
37 int main(){
38     int work1[12], work2[12], work3[12], work4[12], max[12], min[12], sum[12];
39     input_arr("Введите данные по первой работе", work1, 12);
40     input_arr("Введите данные по второй работе", work2, 12);
41     input_arr("Введите данные по третьей работе", work3, 12);
42     input_arr("Введите данные по четвертой работе", work4, 12);
43     for (int i = 0; i < 12; i++){
44         max[i] = work1[i] > work2[i] ? work1[i] : work2[i];
45         if (max[i] < work3[i]) max[i] = work3[i];
46         if (max[i] < work4[i]) max[i] = work4[i];
47         min[i] = work1[i] < work2[i] ? work1[i] : work2[i];
48         if (min[i] > work3[i]) min[i] = work3[i];
49         if (min[i] > work4[i]) min[i] = work4[i];
50         sum[i] = work1[i] + work2[i] + work3[i] + work4[i];
51     }
52     output_arr("Суммарная зарплата", sum, 12);
53     output_arr("Максимальная зарплата", sum, 12);
54     output_arr("Минимальная зарплата", sum, 12);
55     return 0;

```

Заметим, что C++, в большинстве контекстов трактует имя массива как указатель, т.е. имя массива `arr == &arr[0]` рассматривается как адрес его первого элемента.

В листинге 5.18 присутствует следующий вызов функции:

```
input_arr ("Введите данные по первой работе", work1, 12);
```

Здесь `work1` — имя массива, поэтому, согласно правилам C++, `work1` представляет собой адрес первого элемента этого массива. То есть функции передается адрес. Поскольку массив имеет тип элементов `int`, `work1` должно иметь тип указателя на `int`, или `int *`. Это предполагает, что корректный заголовок функции должен быть таким:

```
void input_arr (char *title, int *arr, int n).
```

Здесь `int *arr` заменяет `int arr[]`. На самом деле оба варианта заголовка корректны, потому что в C++ нотации `int *arr` и `int arr[]` имеют идентичный смысл, когда применяются в заголовке или прототипе функции (и только в этом случае).

Теперь понять, как работает эта программа не представляет труда для тех, кто внимательно ознакомился с разделом «массивы».

Рассмотрим, что следует из листинга 5.18. Вызов функции

```
output_arr("Суммарная зарплата", sum, 12);
```

передает адрес первого элемента массива `sum` и количество его элементов в функцию `output_arr`. Функция `output_arr` присваивает адрес `sum` переменной-указателю `arr`, а значение `12` — переменной `n` типа `int`. Это значит, что в листинге 5.18 на самом деле в функцию не передается содержимое массива. Вместо этого программа сообщает функции, где находится массив (то есть сообщает его адрес), каков тип его элементов (тип массива) и сколько в нем содержится элементов (переменная `n`). Вооруженная этой информацией, функция затем использует исходный массив. Если передается обычная переменная, то функция работает с ее копией. Но если передается массив, то функция работает с его **оригиналом**.

Такой подход к обработке массива экономит время и память, необходимые для копирования всего массива. Накладные расходы, связанные с использованием таких копий, могли быть весьма ощутимыми при работе с большими массивами. С

копиями программам понадобилось бы не только больше компьютерной памяти, но и больше времени, чтобы копировать крупные блоки данных. Кроме того появляется возможность использовать некоторые нестандартные обращения. Допустим массив `sum` заполнен так, что каждый элемент равен порядковому номеру месяца, т. е. `sum[0] = 1` и т. д. Тогда можно вывести не весь массив, а только его часть. Например обращение `output_arr("", sum, 5);` выведет на экран следующую таблицу:

	1	2	3	4	5
	1	2	3	4	5

А если использовать следующее обращение `output_arr("", sum + 4, 5)`, то результат будет выглядеть так:

	1	2	3	4	5
	5	6	7	8	9

Это произойдет потому, что `sum+4` в качестве указателя начала массива в функцию передаст указатель на его 5-й элемент.

Работа с двумерными массивами мало отличается от работы с одномерными массивами. Ведь и в этом случае имя массива трактуется как его адрес, поэтому соответствующий формальный параметр является указателем — так же, как и в случае одномерного массива. Только двумерный массив — это как-бы массив массивов, и поэтому при обращении к его конкретному элементу придется встретиться с двумя указателями. Рассмотрим это на простом примере.

Следующие два обращения к элементу массива `ar[5][4]` идентичны:

```
ar[r][c] == *( *( ar + r)+ c)
```

Чтобы понять это, нужно разобрать выражение по частям, начиная изнутри:

```
ar //указатель на первую строку - массив из 4 int
ar + r //указатель на строку r (также массив из 4 int)
*( ar + r) //сама строка r (сам массив из 4 int,
           //т.~е. указатель на первое число в ней - ar[r])
```

```

* (ar + r)+ c //указатель на элемент int под номером c
    //в строке r, т.е. ar[r] + c
* ( *(ar + r)+ c //значение int под номером c
    //в строке r, т.е. ar[r][c]

```

Таким образом следующие описания функций будут идентичны:

```

void input_array (int arr[][], int col, int row);
void input_array (int **arr, int col, int row);
void input_array (int ( *arr)[], int col, int row);

```

Для работы с двумерным динамическим массивом для облегчения чтения кода необходимы как минимум две функции: объявление и инициализация массива и вывод массива на экран.

Функция объявления и инициализации массива представлена в листинге 5.19:

Листинг 5.19. Функция ввода двумерного массива с клавиатуры

```

1 int **create (int n, int m){ //возвращает указатель на первый элемент
2   int **a = new int *[n];
3   /*****выделение памяти*****/
4   for (int i = 0; i < n; i++)
5     a[i] = new int [m];
6   /****заполнение массива с экрана****/
7   for (int i = 0; i < n; i++)
8     for (int j = 0; j < m; j++){
9       cout << "a[" << i << "][" << j << "]=";
10      cin >> a[i][j];
11    }
12   return a;
13 }

```

Функция вывода массива на экран представлена в листинге 5.20:

Листинг 5.20. Функция вывода массива на экран

```

1 void print (int **a, int n, int m){ //функция выводит массив на экран
2   for (int i = 0; i < n; i++, cout << endl)
3     for (int j = 0; j < m; j++)
4       cout << a[i][j] << " ";
5 }

```

5.4.2. Примеры работы с двумерными массивами

Работа с двумерными массивами также сводится к нескольким общим задачам. Некоторые задачи, такие как поиск экстремумов в массиве, поиск значений, попадающих в заданный интервал, замена значений элементов на новые, уже были рассмотрены для случая одномерных массивов и отличаются только наличием вложенного цикла (изменение как параметра i , так параметра j).

Рассмотрим более подробно задачи, связанные с поиском элементов в строке или столбце.

Поиск экстремумов в строках или столбцах

Пусть есть задача найти сумму максимальных элементов каждой строки.

То есть, для каждой строки надо определить свой локальный максимум. Поскольку строку можно рассматривать как одномерный массив, то поиск максимума в строке был показан в разделе 5.2.3. Необходимо обязательно при переходе к рассмотрению новой строки переопределять значение переменной `max`.

Например, фрагмент кода программы 5.21 позволяет найти сумму максимальных элементов каждой строки:

Листинг 5.21. Поиск экстремумов массива

```
1 ...
2 int sum = 0;
3 for (int i = 0; i < n; i++){
4     int max = a[i][0];           //начальное значение для каждой строки
5     for (int j = 1; j < m; j++){
6         if (a[i][j] > max)
7             max = a[i][j];       //max для i-ой строки
8     sum += max;                 //находим сумму
9 }
```

Для i -ой строки массива сначала присваивается переменной `max` значение первого элемента i -ой строки массива (строка 4 листинга 5.21). После этого производится поиск максимального элемента i -ой строки массива (строки 5–7 листинга 5.21). Далее полученное значение добавляется к искомой сумме (строка 8 листинга 5.21).

Пусть есть массив $\begin{pmatrix} 4 & 5 & \mathbf{9} \\ 2 & \mathbf{4} & 1 \\ \mathbf{7} & 3 & 4 \end{pmatrix}$.

Результат работы программы для этого массива:

i	j	$a[i][j]$	max	sum
0	0	4	4	0
0	1	5	5	0
0	2	9	9	9
1	0	2	2	9
1	1	4	4	9
1	2	1	4	13
2	0	7	7	13
2	1	3	7	13
2	2	4	7	20

Ответ sum = 20.

Обмен строк и столбцов

Строки и столбцы можно менять местами только поэлементно. Фиксируем номера строк (столбцов) и меняем местами все элементы этих строк (столбцов).

Листинг 5.22. Поиск экстремумов массива

```

1  ...
2  for (int i = 0; i < n/2; i++){           //меняем первую строку с последней и т.д.
3      for (int j = 0; j < m; j++){
4          swap(a[i][j], a[n-1-i][j]);
5      }
6  ...
7  for (int j = 0; j < m; j += 2){         //меняем первый столбец со вторым и т.д.
8      for (int i = 0; i < n; i++){
9          swap(a[i][j], a[i][j+1]);
10 }

```

В строках 2–5 листинга 5.22 меняются местами первая и последняя строки, потом вторая и предпоследняя и т. д. Условие завершения цикла (строка 2):

$i < n/2$.

Пусть счетчик в операторе цикла будет изменяться до n .

Результаты такой замены представлены ниже для массива размерностью 4×3 :

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}.$$

В таблице полужирным шрифтом выделена i -ая строка, а курсивом — $(n - 1 - i)$ -ая строка:

i	$n - 1 - i$	матрица	результат
0	3	$\begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$
1	2	$\begin{pmatrix} 4 & 4 & 4 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} \\ 3 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 3 & 3 & 3 \\ 2 & 2 & 2 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$
2	1	$\begin{pmatrix} 4 & 4 & 4 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$
3	0	$\begin{pmatrix} \mathbf{4} & \mathbf{4} & \mathbf{4} \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$

Как видно, до середины цикла строки будут меняться правильно. Далее будут меняться уже новые строки, и в итоге получится первоначальная матрица.

Поэтому в строке 2 листинга 5.22 цикл должен ОБЯЗАТЕЛЬНО продолжаться только до $n/2$.

В строках 7–10 производится замена первого столбца со вторым, третьего с четвертым и т. д. Так как меняются столбцы с четными порядковыми номерами с соседними им столбцами, счетчик цикла (строка 7 листинга 5.22) должен увеличиваться на два.

В случае, если необходимо поменять местами строки, содержащие какие-либо элементы, то можно воспользоваться следующим приемом:

1. Создать переменную типа `bool` и присвоить ей значение `true` (предполагаем, что все элементы удовлетворяют нужному условию.)

2. Проверить удовлетворяют ли нужному условию все элементы строки. Если нет, присвоить переменной типа `bool` значение `false` и завершить цикл с помощью операции `break`.
3. Если переменная типа `bool` имеет значение `true`, выполнять необходимые действия.

Фрагмент кода программы (листинг 5.23) демонстрирует поиск в двумерном массиве столбца, содержащего только двузначные элементы и водит на экран номера столбцов:

Листинг 5.23. Поиск экстремумов массива

```
1 ...
2 for (int j = 0; j < m; j++){           //для каждого столбца
3     bool flag = true;
4     for (int i = 0; i < n; i++)
5         if (a[i][j] <= 9 || a[i][j] >= 100){ //число не двузначное
6             flag = false;
7             break;                       //прерывание цикла
8         }
9     if (flag) cout << j << " ";        //если flag = true
10 }
11 ...
```

Диагонали матрицы

В случае квадратной матрицы (размерности $n \times n$) часто встречаются задачи, связанные с работой с диагоналями матрицы (например, один из способов нахождения детерминанта матрицы: преобразовать матрицу в треугольную, потом найти произведение элементов, расположенных на главной диагонали). Если матрица является прямоугольной, о диагоналях говорить нельзя.

У матрицы существует две диагонали: *главная* и *побочная*. На рисунке 5.4 представлены все случаи расположения элементов двумерного массива относительно диагоналей матрицы. Главная диагональ представлена на рисунке 5.4 а.–b., побочная — на рисунке 5.4 с.–d.

Рассмотрим массив размерностью $n \times n$. Слева полужирным шрифтом выделена главная диагональ этого массива, справа — побочная:

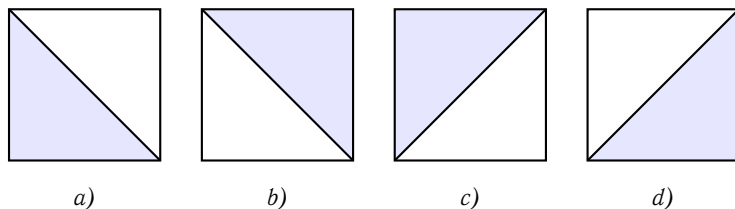


Рис. 5.4. Диагонали матрицы

$$\begin{pmatrix} \mathbf{1} & 2 & 3 & 4 & 5 \\ 1 & \mathbf{2} & 3 & 4 & 5 \\ 1 & 2 & \mathbf{3} & 4 & 5 \\ 1 & 2 & 3 & \mathbf{4} & 5 \\ 1 & 2 & 3 & 4 & \mathbf{5} \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 & 4 & \mathbf{5} \\ 1 & 2 & 3 & \mathbf{4} & 5 \\ 1 & 2 & \mathbf{3} & 4 & 5 \\ 1 & \mathbf{2} & 3 & 4 & 5 \\ \mathbf{1} & 2 & 3 & 4 & 5 \end{pmatrix}$$

Чтобы определить условие расположения элемента относительно диагоналей, выпишем индексы двумерного массива:

$$\begin{pmatrix} \mathbf{00} & 01 & 02 & 03 & 04 \\ 10 & \mathbf{11} & 12 & 13 & 14 \\ 20 & 21 & \mathbf{22} & 23 & 24 \\ 30 & 31 & 32 & \mathbf{33} & 34 \\ 40 & 41 & 42 & 43 & \mathbf{44} \end{pmatrix} \quad \begin{pmatrix} 00 & 01 & 02 & 03 & \mathbf{04} \\ 10 & 11 & 12 & \mathbf{13} & 14 \\ 20 & 21 & \mathbf{22} & 23 & 24 \\ 30 & \mathbf{31} & 32 & 33 & 34 \\ \mathbf{40} & 41 & 42 & 43 & 44 \end{pmatrix}$$

Как видно из представленных примеров, на главной диагонали расположены следующие элементы массива:

$$a[0][0], a[1][1], a[2][2], \dots, a[n][n].$$

Следовательно, условие нахождения элемента на главной диагонали:

$$i = j.$$

На побочной диагонали расположены следующие элементы массива:

$$a[0][n - 1], a[1][n - 2], a[2][n - 3], \dots, a[n - 1][0].$$

Следовательно, условие нахождения элемента на главной диагонали:

$$j = n - 1 - i.$$

В таблице 5.1 представлены все случаи расположения элементов двумерного массива относительно диагоналей.

Таблица 5.1. Расположение элементов относительно диагоналей матрицы

Расположение	Условие	Рисунок	Код программы
Выше главной	$j > i$	Рис. 5.4b.	<pre> for(int i = 0; i < n; i++) for (int j = i + 1; j < n; j++) ... </pre>
Ниже главной	$j < i$	Рис. 5.4a.	<pre> for(int i = 0; i < n; i++) for (int j = 0; j < i; j++) ... </pre>
Выше побочной	$j < n - 1 - i$	Рис. 5.4c.	<pre> for(int i = 0; i < n; i++) for (int j = 0; j < n - 1 - i; j++) ... </pre>
Ниже побочной	$j > n - 1 - i$	Рис. 5.4d.	<pre> for(int i = 0; i < n; i++) for (int j = n - 1 - i; j < n; j++) ... </pre>

Примеры задач

Пример 5.5. Поменять местами строки, содержащие первый минимальный и последний максимальный элементы массива.

Помимо поиска минимального и максимального элементов в массиве, необходимо также хранить информацию об индексе строки, где находятся соответствующие элементы.

Когда находим элемент, меньший (больший) текущего экстремума, не только меняем значение экстремума, но и сохраняем номер строки, содержащий этот элемент:

Листинг 5.24.

```

1 ...
2 for (int i = 0; i < n; i++){
3     min = a[i][0];           //начальный min в i-ой строке
4     for (int j = 1; j < m; j++)
5         if ( a[i][j] > min){
6             min = a[i][j];    //определяем min
7             n_min = i;        //сохраняем номер строки с min
8         }
9     }

```

В случае, если минимальных или максимальных элементов массива больше одного, то в данной задаче необходимо запомнить номер строк, содержащих первый и последний экстремум. Если в качестве поиска экстремума использовать строгое неравенство: $a < \min$, то будет найден именно первый минимальный элемент, так как элемент, равный \min , не удовлетворяет этому неравенству.

Если же рассматривать нестрогое неравенство вида: $a \leq \min$, то будет найден номер последнего элемента, так как элемент, равный \min , удовлетворяет соответствующему неравенству.

Листинг 5.25. Реализация задания (пример 5.5)

```

1  #include<iostream>
2  using namespace std;
3
4  int **create (int n, int m){           //создание массива
5      int **a = new int *[n];          //выделение памяти под массив
6      for (int i = 0; i < n; i++)
7          a[i] = new int [m];
8      for (int i = 0; i < n; i++)        //заполняем массив
9          for (int j = 0; j < m; j++){
10             cout << "a[" << i << "][" << j << " ]";
11             cout << j << " ]=";
12             cin >> a[i][j];
13         }
14     return a;
15 }
16
17 void print (int **a, int n, int m){     //вывод массива на экран
18     for (int i = 0; i < n; i++, cout << endl)
19         for (int j = 0; j < m; j++)
20             cout << a[i][j] << " ";
21     cout << endl;
22 }
23
24 void change (int **a, int n, int m){    //меняем строки местами
25     int min = a[0][0], max = a[0][0];   //нач. знач. для min, max
26     int n_min = 0, n_max = 0;           //номера строк
27     for (int i = 0; i < n; i++)
28         for (int j = 0; j < m; j++){

```

```

29     if (a[i][j] < min){                //первый min
30         min = a[i][j];
31         n_min = i;                      //номер строки с min
32     }
33     if (a[i][j] >= max){                //последний max
34         max = a[i][j];
35         n_max = i;                      //номер строки с max
36     }
37 }
38 for (int j = 0; j < m; j++)            //обмен строк
39     swap( a[n_min][j], a[n_max][j]);
40 }
41
42 int main()
43 {
44
45     int n, m;                           //размерность массива
46     cout << "n = "; cin >> n;
47     cout << "m = "; cin >> m;
48     int **a = create (n, m);            //создание массива
49     print(a, n, m);                     //вывод на экран
50     change(a, n, m);                    //обмен строк
51     print(a, n, m);                     //вывод на экран
52     return 0;
53 }

```

Рассмотрим работу листига 5.25 на примере массива размерностью 3×3 (Полужирным шрифтом выделены искомые элементы):

Массив:	Результат:
$\begin{pmatrix} 3 & \mathbf{1} & 9 \\ 2 & 1 & 5 \\ 1 & \mathbf{9} & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 9 & 3 \\ 2 & 1 & 5 \\ 3 & 1 & 9 \end{pmatrix}$

Пошаговая реализация представлена в таблице:

i	j	$a[i][j]$	min	n_min	max	n_max
0	0	3	3	0	3	0
0	1	1	1	0	3	0
0	2	9	1	0	9	0
1	0	2	1	0	9	0
1	1	1	1	0	9	0
1	2	5	1	0	9	0
2	0	1	1	0	9	0
2	1	9	1	0	9	2
2	2	3	1	0	9	2

В результате нулевая и вторая строки массива меняются местами. □

Пример 5.6. Поменять местами столбец, содержащий только четные элементы, со столбцом, содержащим только нечетные элементы. Если таких столбцов несколько, поменять первые столбцы, удовлетворяющие условиям.

Для того, чтобы найти первый столбец, содержащий только нечетные элементы, понадобится две переменных логического типа: первая переменная, в листинге 5.26 обозначена как `flag_on`, отвечает за поиск нужного столбца (равна `true`, если такой столбец уже найден, и `false` в противоположном случае); вторая переменная, обозначенная в листинге 5.26 как `flag_o`, отвечает за поиск четных элементов в текущем столбце (равна `true`, если в столбце нет четных элементов, и `false` в противоположном случае).

Для столбца, содержащего только четные элементы, все аналогично. Соответствующие переменные называются `flag_en` и `flag_e`.

Рассмотрим алгоритм для поиска первого столбца, содержащего только нечетные элементы (строки 30–40).

Изначально переменной `flag_on` присваивается значение `false` (строка 25 листинга 5.26).

Для каждого столбца (строка 27) выполняются следующие действия:

1. Переменной `flag_o` присваивается значение `true` (предполагаем, что в столбце все элементы нечетные.)
2. Если `flag_on` равно `false` (еще не нашли нужного столбца), то ищем четный элемент в текущем столбце. Если такой элемент найден (строка 31), присваиваем переменной `flag_o` значение `false` и прекращаем цикл (строки 33–34).

3. Если после выполнения строк 31–35 листинга 5.26 значение переменной `flag_o` осталось `true`, значит в текущем столбце нет четных элементов. Соответственно, переменной `n_odd` присваиваем номер текущего столбца и меняем значение переменной `flag_on` на `true` (так как найден нужный столбец) (строки 36–39).

В строках 42–51 рассмотрен поиск столбца, содержащего только четные элементы по аналогичному алгоритму.

Замена столбцов происходит, если переменные `flag_on` и `flag_en` равны `true` (строки 55–59). Если одно из значений равно `false`, выводим сообщение о том, какого столбца не найдено (строки 60–68).

Листинг 5.26. Реализация задания (пример 5.6)

```
1 #include<iostream>
2 using namespace std;
3
4 int **create (int n, int m){           //создание массива
5     int **a = new int *[n];           //выделение памяти под массив
6     for (int i = 0; i < n; i++)
7         a[i] = new int [m];
8     for (int i = 0; i < n; i++)         //заполняем массив
9         for (int j = 0; j < m; j++){
10             cout << "a[" << i << "][" << j << " ] = ";
11             cout << j << " ] = ";
12             cin >> a[i][j];
13         }
14     return a;
15 }
16
17 void print (int **a, int n, int m){    //вывод массива на экран
18     for (int i = 0; i < n; i++, cout << endl)
19         for (int j = 0; j < m; j++)
20             cout << a[i][j] << " ";
21     cout << endl;
22 }
23
24 bool change (int **a, int n, int m){   //меняем столбцы местами
25     bool flag_on = false, flag_en = false; //true - если нашли соотв. столбцы
```

```

26  int n_odd = 0, n_even = 0;                // odd - нечетный, even - четный
27  for (int j = 0; j < m; j++){              //для каждого столбца
28      bool flag_o = true, flag_e = true;    //предпол., столбцы удовл. условию
29      /*-----проверка для нечетного столбца-----*/
30      if (!flag_on){                        //если не нашли столбца с нечет. эл-ми
31          for (int i = 0; i < n; i++)
32              if (!(a[i][j] % 2)){          //встретили четный элемент
33                  flag_o = false;          //меняем флаг
34                  break;
35              }
36          if (flag_o) {                     //если true - только нечетные элементы
37              flag_on = true;              //нашли нужный столбец
38              n_odd = j;                   //запоминаем номер столбца
39          }
40      }
41      /*-----проверка для четного столбца -----*/
42      if (!flag_en){                       //все аналогично поиску нечетного столбца
43          for (int i = 0; i < n; i++)
44              if (a[i][j] % 2){
45                  flag_e = false;
46                  break;
47              }
48          if (flag_e){
49              flag_en = true;
50              n_even = j;
51          }
52      }
53  }
54  /*-----замена-----*/
55  if (flag_on && flag_en){                  //если нашли нужные столбцы
56      for(int i = 0; i < n; i++)
57          swap( a[i][n_even], a[i][n_odd]); //меняем их местами
58      return true;
59  }
60  else{                                    //иначе выводим какой столбец не нашли
61      if (!flag_on && !flag_en)
62          cout << "Нет столбцов, удовл. условиям\n";
63      else if (!flag_on)
64          cout << "Нет столбцов, сод. только нечетные элементы\n";

```

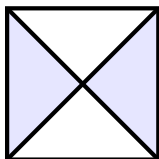
```
65     else
66         cout << "Нет столбцов, сод. только четные элементы\n";
67     return false;
68 }
69 }
70
71 int main()
72 {
73     setlocale(LC_ALL, "RUS");
74     int n, m; //размерность массива
75     cout << "n = "; cin >> n;
76     cout << "m = "; cin >> m;
77     int **a = create (n, m); //создание массива
78     print(a, n, m); //вывод на экран
79     bool flag = change(a, n, m); //обмен строк
80     if (flag) //если обмен произошел
81         print(a, n, m); //вывод на экран
82     return 0;
83 }
```

Пример работы программы:

Массив:	Результат:
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 4 & 5 \\ 1 & 3 & 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 4 & 2 & 1 & 5 \\ 4 & 2 & 1 & 5 \\ 4 & 3 & 1 & 5 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 4 & 4 \\ 2 & 3 & 4 & 4 \end{pmatrix}$	Нет столбца, сод. только нечетные элементы
$\begin{pmatrix} 1 & 2 & 4 & 5 \\ 1 & 2 & 1 & 5 \\ 1 & 3 & 4 & 5 \end{pmatrix}$	Нет столбца, сод. только четные элементы
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix}$	Нет столбцов, удовл. условиям

□

Пример 5.7. Найти сумму четных элементов, расположенных в заштрихованной области, включая границы. Если четных элементов нет, вывести сообщение об этом.



Левый треугольник означает, что элемент массива расположен ниже главной диагонали и выше побочной. Согласно таблице 5.1 условие расположения элемента ниже главной диагонали $j \leq i$ (сама диагональ также включена в заштрихованную область). Условие расположения элемента выше побочной диагонали: $j \leq n - 1 - i$. Тогда условие нахождения элемента внутри заштрихованной области (левый треугольник):

$$j \leq i \wedge j \leq n - 1 - i.$$

Правый треугольник означает, что элемент находится выше главной диагонали и ниже побочной диагонали. Соответственно, условие нахождения элемента внутри заштрихованной области (правый треугольник):

$$j \geq i \wedge j \geq n - 1 - i.$$

Если n нечетное, то элемент $a[n/2][n/2]$ принадлежит как правому, так и левому треугольнику (деление нацело, т. е., если $n = 5$, то $n/2 = 2$).

Например, если $n = 5$, то условие нахождения элемента в заштрихованной области:

$$\begin{cases} j \leq i \wedge j \leq 4 - i, \\ j \geq i \wedge j \geq 4 - i. \end{cases}$$

Листинг 5.27. Реализация задания (пример 5.7)

```
1 #include<iostream>
2 using namespace std;
3
4 int **create (int n){           //создание массива
5     int **a = new int *[n];    //выделение памяти под массив
6     for (int i = 0; i < n; i++)
7         a[i] = new int [n];
8     for (int i = 0; i < n; i++)  //заполняем массив
```

```

9     for (int j = 0; j < n; j++){
10         cout << "a[" << i << "][" << j << "]";
11         cout << j << "]=";
12         cin >> a[i][j];
13     }
14     return a;
15 }
16
17 void print (int **a, int n){                //вывод массива на экран
18     for (int i = 0; i < n; i++, cout << endl)
19         for (int j = 0; j < n; j++)
20             cout << a[i][j] << " ";
21     cout << endl;
22 }
23
24 int summa (int **a, int n){                //сумма элементов
25     int k = 0, sum = 0;
26     for (int i = 0; i < n; i++)
27         for (int j = 0; j < n; j++)
28             if ((j <= i && j <= n - 1 - i) //если элемент находится
29                 || (j >= i && j >= n - 1 - i)){ // в заштрихованной области
30                 if (!(a[i][j] % 2)){        //является четным
31                     sum += a[i][j];          //ищем сумму
32                     k++;                      //количество
33                 }
34             }
35     if (!k) sum = -100000;                  // если четных элементов нет
36     return sum;
37 }
38
39 int main()
40 {
41     setlocale (LC_ALL, "RUS");
42     int n;                                //размерность массива
43     cout << "n = "; cin >> n;
44     int **a = create (n);                 //создание массива
45     print(a, n);                          //вывод на экран
46     int sum = summa (a, n);
47     if (sum != -100000)

```

```
48     cout << "sum = " << sum << endl;
49     else
50         cout << "четных элементов нет\n";
51     return 0;
52 }
```

Пример работы программы:

Массив:	Результат:
$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & \mathbf{2} & 3 & \mathbf{4} & 5 \\ 1 & 3 & \mathbf{2} & \mathbf{4} & 5 \\ 1 & \mathbf{2} & 3 & \mathbf{4} & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$	18



5.5. Упражнения

- I. Дан одномерный массив, содержащий целые числа. Выполнить следующие действия:
- 1) Найти сумму четных элементов, попадающих в заданный интервал, включая границы интервала. Ноль не учитывать. Если четных элементов нет, вывести сообщение об этом.
 - 2) Найти среднее арифметическое четных элементов. Ноль не учитывать. Если четных элементов нет, вывести сообщение об этом.
 - 3) Найти сумму нечетных элементов, меньших заданного числа X . Если таких элементов нет, вывести сообщение об этом.
 - 4) Найти сумму четных элементов, больших заданного числа X . Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
 - 5) Найти сумму четных элементов, кратных 3. Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
 - 6) Найти сумму нечетных элементов, не кратных 3. Если таких элементов нет, вывести сообщение об этом.
 - 7) Найти среднее арифметическое нечетных элементов, не попадающих в заданный интервал. Если таких элементов нет, вывести сообщение об этом.
 - 8) Найти произведение четных элементов, не попадающих в заданный интервал. Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.

- 9) Вывести номера (нумерация элементов массива начинается с нуля) элементов, кратных 4 и попадающих в заданный интервал, включая границы интервала. Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 10) Вывести номера (нумерация элементов массива начинается с нуля) элементов, кратных 3 и не попадающих в заданный интервал. Если таких элементов нет, вывести сообщение об этом.
- 11) Найти сумму нечетных элементов с нечетными порядковыми номерами (нумерация элементов массива начинается с нуля). Если таких элементов нет, вывести сообщение об этом.
- 12) Найти произведение четных элементов, кратных 5. Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 13) Найти сумму четных элементов с нечетными порядковыми номерами (нумерация элементов массива начинается с нуля). Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 14) Найти среднее арифметическое четных элементов, кратных 3. Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 15) Найти сумму четных элементов с четными порядковыми номерами (нумерация элементов массива начинается с нуля). Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 16) Вывести номера четных элементов, не кратных 3 (нумерация элементов массива начинается с нуля). Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 17) Найти сумму нечетных элементов с четными порядковыми номерами (нумерация элементов массива начинается с нуля). Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 18) Найти среднее арифметическое четных элементов, кратных 3 и не попадающих в заданный интервал (нумерация элементов массива начинается с нуля). Ноль не учитывать. Если таких элементов нет, вывести сообщение об этом.
- 19) Вывести номера нечетных элементов, не кратных 3 (нумерация элементов массива начинается с нуля). Если таких элементов нет, вывести сообщение об этом.
- 20) Найти сумму нечетных элементов, не попадающих в заданный интервал. Если нечетных элементов нет, вывести сообщение об этом.

II. Дан одномерный массив, содержащий целые числа. Выполнить следующие действия:

- 1) Вывести номера всех минимальных элементов. Нумерация начинается с нуля.
- 2) Поменять местами первый и средний элемент, если количество элементов массива нечетное, и два средних элемента, если размерность массива четная.
- 3) Поменять местами первый элемент и последний минимальный элемент массива.
- 4) Поменять местами последний элемент и первый максимальный элемент массива.
- 5) Вывести номер первого минимального элемента. Нумерация начинается с нуля.
- 6) Поменять местами первый минимальный и последний максимальный элементы массива.
- 7) Вывести номер последнего минимального элемента. Нумерация начинается с нуля.
- 8) Найти среднее арифметическое элементов, расположенных между первым минимальным и последним максимальным элементами. Если последний максимальный элемент расположен раньше первого минимального вывести сообщение об этом.
- 9) Все минимальные элементы увеличить в два раза.
- 10) Поменять местами первый элемент и первый максимальный элемент массива.
- 11) Заменить все максимальные элементы их противоположными.
- 12) Все максимальные элементы уменьшить в три раза.
- 13) Вывести номер последнего максимального четного элемента. Нумерация начинается с нуля. Если таких элементов нет, то вывести сообщение об этом.
- 14) Поменять местами первый максимальный четный элемент и первый минимальный нечетный элемент. Если таких элементов нет, то вывести сообщение об этом.
- 15) Вывести номера всех минимальных нечетных элементов. Нумерация начинается с нуля. Если таких элементов нет, то вывести сообщение об этом.

- 16) Поменять местами первый элемент и первый четный максимальный элемент. Если таких элементов нет, то вывести сообщение об этом.
- 17) Вывести номера всех максимальных четных элементов. Нумерация начинается с нуля. Если таких элементов нет, то вывести сообщение об этом.
- 18) Поменять местами последний элемент и последний нечетный минимальный элемент. Если таких элементов нет, то вывести сообщение об этом.
- 19) Поменять местами первый минимальный нечетный элемент и последний максимальный четный элемент. Если таких элементов нет, то вывести сообщение об этом.
- 20) Все максимальные четные элементы увеличить в два раза. Если таких элементов нет, то вывести сообщение об этом.

III. Дан двумерный массив, содержащий целые числа, размерностью $n \times n$.

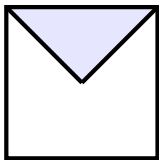
Выполнить следующие действия:

- 1) Поменять местами строки по следующему правилу: первую с последней, вторую с предпоследней и т. д.
- 2) Поменять местами столбцы по следующему правилу: первый с последним, второй с предпоследним и т. д.
- 3) Поменять местами строки по следующему правилу: первую со второй, третью с четвертой и т. д.
- 4) Поменять местами столбцы по следующему правилу: первый со вторым, третий с четвертым и т. д.
- 5) Найти первый минимальный и первый максимальный элементы массива. Поменять местами строки, содержащие эти элементы.
- 6) Найти первый минимальный и первый максимальный элементы массива. Поменять местами столбцы, содержащие эти элементы.
- 7) Найти первый минимальный и последний максимальный элементы массива. Поменять местами строки, содержащие эти элементы.
- 8) Найти первый минимальный и последний максимальный элементы массива. Поменять местами столбцы, содержащие эти элементы.
- 9) Все строки, содержащие минимальный элемент, заменить строкой X .
- 10) Все столбцы, содержащие минимальный элемент, заменить столбцом X .
- 11) Все четные строки заменить строкой X . (Нумерация строк массива начинается с нуля.)

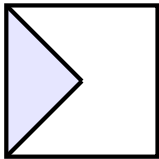
- 12) Все нечетные столбцы заменить столбцом X . (Нумерация столбцов массива начинается с нуля.)
- 13) Поменять местами первую строку и строку, сумма элементов которой максимальна.
- 14) Поменять местами две средних строки, если количество строк четное.
- 15) Поменять местами последний столбец и столбец, сумма элементов которого минимальна.
- 16) Поменять местами первый и средний столбцы, если количество столбцов нечетное.
- 17) Заменить строки, содержащие только нечетные элементы, строкой X .
- 18) Заменить столбцы, содержащие только четные элементы, столбцом X .
Если таких строк нет, выдать сообщение об этом.
- 19) Поменять местами строку, содержащую первый нечетный элемент массива, и строку, содержащую последний четный элемент массива.
- 20) Поменять местами столбец, содержащий первый четный элемент массива, и столбец, содержащий последний нечетный элемент массива.
Если таких элементов нет, выдать сообщение о том, какого именно элемента (четного или нечетного) в массиве нет.

IV. Дан массив из целых чисел размерностью $n \times n$. Выполнить следующие действия:

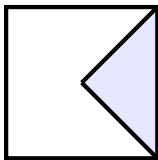
- 1) Определить максимальный элемент в заштрихованной области, включая границы.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	8

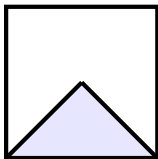
- 2) Определить минимальный элемент в заштрихованной области, включая границы.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	2

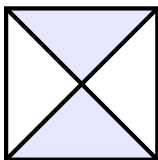
- 3) Определить среднее арифметическое четных элементов, расположенных в заштрихованной области, включая границы. Ноль не учитывать. Если четных элементов в заштрихованной области нет, выдать сообщение об этом.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	2.3333

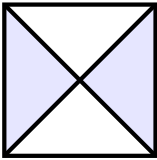
- 4) Определить максимальный нечетный элемент в заштрихованной области, включая границы. Если нечетных элементов в заштрихованной области нет, выдать сообщение об этом.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	7

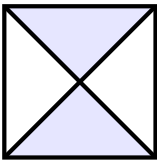
- 5) Определить сумму максимального элемента в заштрихованной области, включая границы, и минимального элемента в незаштрихованной области, не включая границы.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	8

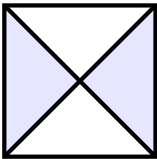
- 6) Определить максимальный четный элемент в заштрихованной области, включая границы. Ноль не учитывать. Если четных элементов в заштрихованной области нет, выдать сообщение об этом.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	4

- 7) Определить минимальный четный элемент в заштрихованной области, включая границы. Ноль не учитывать. Если четных элементов в заштрихованной области нет, выдать сообщение об этом.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	2

- 8) Определить среднее арифметическое нечетных элементов, расположенных в заштрихованной области, включая границы. Если нечетных элементов в заштрихованной области нет, выдать сообщение об этом.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	4.7

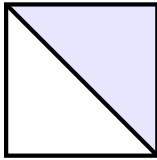
- 9) Заменить первый максимальный элемент в заштрихованной области, включая границы, на первый минимальный элемент из незаштрихованной области, не включая границы.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 9 \\ 0 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$

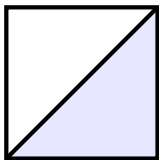
- 10) Заменить первый минимальный четный элемент в заштрихованной области, включая границы, на первый максимальный нечетный элемент из незаштрихованной области, не включая границы. Ноль не учитывать. Если таких элементов нет, выдать сообщение о том, какого именно элемента (четного или нечетного) нет в массиве.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 2 & 8 & 2 & 1 \\ 5 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$

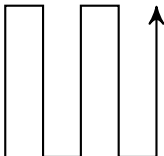
- 11) Заменить первый максимальный нечетный элемент в заштрихованной области, включая границы, на первый минимальный четный элемент из незаштрихованной области, не включая границы. Ноль не учитывать. Если таких элементов нет, выдать сообщение о том, какого именно элемента (четного или нечетного) нет в массиве.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 7 & 4 & 1 & 3 & 2 \\ 4 & 3 & 2 & 2 & 9 \\ 0 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$

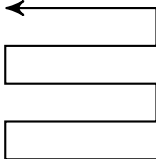
- 12) Заменить последний максимальный четный элемент в заштрихованной области, включая границы, на последний минимальный четный элемент из незаштрихованной области, не включая границы. Ноль не учитывать. Если таких элементов нет, выдать сообщение о том, какого именно элемента (четного или нечетного) нет в массиве.

Задание	Массив	Результат
	$\begin{pmatrix} 4 & 5 & 8 & 2 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 8 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 8 & 8 & 1 \\ 2 & 4 & 1 & 3 & 2 \\ 4 & 3 & 7 & 2 & 0 \\ 9 & 5 & 2 & 7 & 4 \\ 4 & 6 & 2 & 1 & 3 \end{pmatrix}$

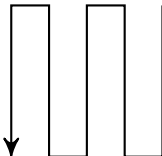
- 13) Заполнить массив $n \times n$ числами от 0 до $n^2 - 1$ в требуемом порядке:

Задание	Массив	Результат
	$n = 5$: числа от 0 до 24	$\begin{pmatrix} 4 & 5 & 14 & 15 & 24 \\ 3 & 6 & 13 & 16 & 23 \\ 2 & 7 & 12 & 17 & 22 \\ 1 & 8 & 11 & 18 & 21 \\ 0 & 9 & 10 & 19 & 20 \end{pmatrix}$

- 14) Заполнить массив $n \times n$ числами от 0 до $n^2 - 1$ в требуемом порядке:

Задание	Массив	Результат
	$n = 5$: числа от 0 до 24	$\begin{pmatrix} 24 & 23 & 22 & 21 & 20 \\ 15 & 16 & 17 & 18 & 19 \\ 14 & 13 & 12 & 11 & 10 \\ 5 & 6 & 7 & 8 & 9 \\ 4 & 3 & 2 & 1 & 0 \end{pmatrix}$

- 15) Заполнить массив $n \times n$ числами от 0 до $n^2 - 1$ в требуемом порядке:

Задание	Массив	Результат
	$n = 5$: числа от 0 до 24	$\begin{pmatrix} 20 & 19 & 10 & 9 & 0 \\ 21 & 18 & 11 & 8 & 1 \\ 22 & 17 & 12 & 7 & 2 \\ 23 & 16 & 13 & 6 & 3 \\ 24 & 15 & 14 & 5 & 4 \end{pmatrix}$

- V. Разработать программу для работы с многочленами. Многочлен представлен как одномерный массив размерности $n + 1$, где $a[i]$ — это коэффициенты многочлена вида

$$\sum_{i=0}^n a_i x^i$$

- 1) Дан многочлен $P(x)$ степени n и действительное число a . Получить многочлен $(x^2 - a)P(x)$.
- 2) Дан многочлен $P(x)$ степени n и действительное число a . Получить многочлен $(x^2 + 2ax + a^2)P(x)$.
- 3) Дан многочлен $P(x)$ степени n . Получить $\int P(x)dx$. Считать равной нулю.
- 4) Дан многочлен $P(x)$ степени n . Получить многочлен $P(x) + P'(x)$, где $P'(x)$ — это производная многочлена $P(x)$.
- 5) Дан многочлен $P(x)$ степени n . Получить многочлен $P(x) \cdot P'(x)$, где $P'(x)$ — это производная многочлена $P(x)$.
- 6) Дан многочлен $P(x)$ степени n . Получить его вторую производную $P''(x)$.
- 7) Даны многочлены $P(x)$ степени n и $Q(x)$ степени m . Получить многочлен $\int P(x) \cdot Q(x)dx$. Считать нулем.
- 8) Даны многочлены $P(x)$ степени n и $Q(x)$ степени m . Получить многочлен $(P(x) \cdot Q(x))'$.

5.6. Контрольные вопросы

1. Что такое указатель?
2. Опишите операции взятия адреса и разыменования.
3. Что такое массив?
4. Что такое статический массив?
5. Приведите пример работы со статическим одномерным массивом.
6. Что такое динамический массив?

7. Опишите операции поиска экстремумов в массиве.
8. Опишите двумерный статический массив. Как он представлен в памяти.
9. Опишите двумерный динамический массив.
10. Опишите алгоритм поиска минимального элемента, расположенного ниже главной диагонали.

Литература

1. *Страуструп, .* ПРОГРАММИРОВАНИЕ: принципы и практика использования C++ (для C++11 и C++14) / . Страуструп. — М.: «И.Д. Вильямс», 2015.
2. Создание приложения Windows Forms с помощью .NET Framework (C++) [Электронный ресурс]. — URL: [http://msdn.microsoft.com/ru-ru/library/vstudio/ms235634\(v=vs.100\).aspx](http://msdn.microsoft.com/ru-ru/library/vstudio/ms235634(v=vs.100).aspx) (Дата обращения 12.07.2013). Загл. с экр. Яз. рус.
3. *Абачиев, . .* Треугольник Паскаля и спектр арифметик для цифровых информационных технологий / . . Абачиев, . . Стахов // *Науковедение*. — 2012. — Т. 4.
4. *Прата, .* Язык программирования C++. Лекции и упражнения / . Прата. — М.: «И.Д. Вильямс», 2012.