Министерство науки и высшего образования Российской Федерации

КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Учебно-методическое пособие

Краснодар
2019

Министерство науки и высшего образования
Российской Федерации

КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Учебно-методическое пособие

Краснодар
2019

УДК 004.432.2
ББК 32.973 (018.2)
О 44

Рецензенты:
Кандидат физико-математических наук, доцент
М.Е. Бегларян
Кандидат экономических наук, доцент
В.В. Ткаченко

О 44 Основы программирования на языке С++: учеб.-метод. пособие / авт. В. В. Подколзин, Е. П. Лукащик, Н. Ю. Добровольская, О. В. Гаркуша, С. Г. Синица, А. А. Полупанов, А.Н. Полетайкин, Р.Х. Багдасарян, А. В. Харченко, А. А. Михайличенко, А. В. Уварова – Краснодар: Кубанский гос. ун-т, 2019. – 156 с. – 500 экз.

ISBN 978-5-8209-1622-9

Содержит материал об основах программирования на языке С++, а также о базовых аспектах алгоритмизации.

Адресуется студентам I курса направления бакалавриата 01.03.02 «Прикладная математика и информатика», направления бакалавриата 09.03.03 «Прикладная информатика», направления бакалавриата 02.03.03 «Математическое обеспечение и администрирование информационных систем», направления бакалавриата 02.03.02 «Фундаментальные информатика и информационные технологии» для изучения практических основ программирования на языках высокого уровня.

УДК 004.432.2
ББК 32.973 (018.2)

ISBN 978-5-8209-1622-9

ПРЕДИСЛОВИЕ

Предлагаемое издание адресуется студентам направления «01.03.02 Прикладная математика и информатика», «02.03.02 Фундаментальная информатика и информационные технологии», «02.03.03 Математическое обеспечение и администрирование информационных систем», «09.03.03 Прикладная информатика», а также всем желающим изучить язык программирования C++. Пособие предлагается использовать при выполнении лабораторных работ студентами I курса факультета компьютерных технологий и прикладной математики по дисциплине «Основы программирования». Примеры, рассмотренные в данном издании, позволят студентам более полно разобраться в изучаемом материале, а решение задач по темам – закрепить полученные знания на практике.

Усвоение информации по основам программирования на языке C++ создаст базу для изучения материала по дисциплинам «Методы программирования», «Объектно-ориентированное программирование», «Основы программирования в RAD системах».

Структура учебно-методического пособия соответствует учебному плану рабочей программы дисциплины «Основы программирования», а представленный материал ориентирован на использование в рамках самостоятельной работы студентов. При подготовке к лабораторным занятиям студентам рекомендуется изучить теоретический материал и ознакомиться с примерами решения задач по темам. Контрольные задания, приведенные в конце каждой темы, могут использоваться преподавателями для текущего контроля уровня усвоения обучающимися материала, а также в качестве самостоятельной работы студентами.

Понятия, объекты, имена переменных или другие элементы, представленные в примерах и описаниях программного кода, подлежащие замене на конкретные значения, обозначаются в угловых скобках. Например:

```
<тело программы>
```

ВВЕДЕНИЕ

Среди современных языков программирования язык С является одним из наиболее распространенных. Язык С универсален, однако наиболее эффективно его применение в задачах системного программирования: разработке трансляторов, операционных систем, инструментальных средств. Язык С хорошо зарекомендовал себя эффективностью, лаконичностью записи алгоритмов, логической стройностью программ. Во многих случаях программы, написанные на языке С, сравнимы по скорости с программами, написанными на Ассемблере, при этом они более наглядны и просты в сопровождении.

Язык С имеет ряд существенных особенностей, которые выделяют его среди других языков программирования. В значительной степени на формирование идеологии языка повлияла цель, которую ставили перед собой его создатели – обеспечение системного программиста удобным инструментальным языком, который мог бы заменить Ассемблер. В результате появился язык программирования высокого уровня, обеспечивающий необычайно легкий доступ к аппаратным средствам компьютера. С одной стороны, как и другие современные языки высокого уровня, язык С поддерживает полный набор конструкций структурного программирования, модульность, блочную структуру программы. С другой стороны, язык С имеет ряд низкоуровневых черт, перечислим некоторые из них.

В языке С реализован ряд операций низкого уровня. Некоторые из таких операций напрямую соответствуют машинным командам, например, поразрядные операции, или операции ++ и --.

Базовые типы данных языка С отражают те же объекты, с которыми приходится иметь дело в программе на Ассемблере – байты, машинные слова и т.д. Несмотря на наличие в языке С развитых средств построения составных объектов (массивов и структур), в нем практически отсутствуют средства для работы с ними как с единым целым.

Язык С поддерживает механизм указателей на переменные и функции. Указатель – это переменная, предназначенная для хранения машинного адреса некоторой переменной или функции. Поддерживается арифметика указателей, что позволяет осуществлять непосредственный доступ и работу с адресами памяти практически так же легко, как на Ассемблере. Использование указателей позволяет создавать высокоэффективные программы, однако требует от программиста особой осторожности.

Как никакой другой язык программирования высокого уровня, язык С «доверяет» программисту. Даже в таком существенном вопросе, как преобразование типов данных, налагаются лишь незначительные ограничения. Однако это также требует от программиста осторожности и самоконтроля.

Несмотря на эффективность и мощность конструкция языка С, он относительно мал по объему. В нем отсутствуют встроенные операторы ввода / вывода, динамического распределения памяти, управления процессами и т.п., однако в системное окружение языка С входит библиотека стандартных функций, в которой реализованы подобные действия.

Язык С++ – это язык программирования общего назначения, цель которого – сделать работу серьёзных программистов более приятным занятием. За исключением несущественных деталей язык С++ является надмножеством языка С. Помимо возможностей, предоставляемых языком С, язык С++ обеспечивает гибкие и эффективные средства определения новых типов.

Язык программирования служит двум взаимосвязанным целям: он предоставляет программисту инструмент для описания подлежащих выполнению действий и набор концепций, которыми оперирует программист, обдумывая, что можно сделать. Первая цель в идеале требует языка, близкого к компьютеру, чтобы все важные элементы компьютера управлялись просто и эффективно способом, достаточно очевидным для программиста. Язык С создавался на основе именно от этой идеи. Вторая цель в идеале требует языка, близкого к решаемой задаче, чтобы концепции решения могли

быть выражены понятно и непосредственно. Эта идея привела к пополнению языка С свойствами, превратившими его в язык С++.

Ключевое понятие в языке С++ – класс. Классы обеспечивают сокрытие информации, гарантированную инициализацию данных, неявное преобразование определяемых пользователем типов, динамическое определение типа, контроль пользователя над управлением памятью и механизм перегрузки операторов. Язык С++ предоставляет гораздо лучшие, чем язык С, средства для проверки типов и поддержки модульного программирования. Кроме того, язык содержит усовершенствования, непосредственно не связанные с классами, такие как символические константы, встраивание функций вместо вызова, параметры функций по умолчанию, перегруженные имена функций, операторы управления свободной памятью и ссылки. Язык С++ сохраняет способность языка С эффективно работать с аппаратной частью на уровне битов, байтов, слов, адресов и т. д. Это позволяет реализовывать пользовательские типы с достаточной степенью эффективности.

В предлагаемом издании рассматриваются следующие теоретические и практические вопросы разработки программ на языке программирования С++: создание консольного и пустого приложения в среде Microsoft Visual Studio, структура программы, стандартные типы, переменные, ввод данных, оператор присваивания, выражения, условный оператор, операторы цикла, массивы, функции.

1. СОЗДАНИЕ КОНСОЛЬНОГО И ПУСТОГО ПРИЛОЖЕНИЯ В СРЕДЕ MICROSOFT VISUAL STUDIO

Microsoft Visual Studio – одна из самых популярных сред разработки (IDE – Integrated development environment). Она включает различные компоненты по созданию приложений для мобильных устройств, настольных приложений, веб-приложений и облачных решений. Доступна возможность писать код для iOS, Android и Windows в одной интегрированной среде разработки.

Версию Microsoft Visual Studio Community Edition можно скачать бесплатно на официальном сайте.

Для создания проекта необходимо выбрать пункт «Новый проект» (New Project…) в левой части стартового окна или в меню Файл – Новый – Проект (File – New – Project). Затем в появившемся окне (рис. 1) выбрать тип проекта (Visual C++) и его подтип – «Консольное приложение Win32»

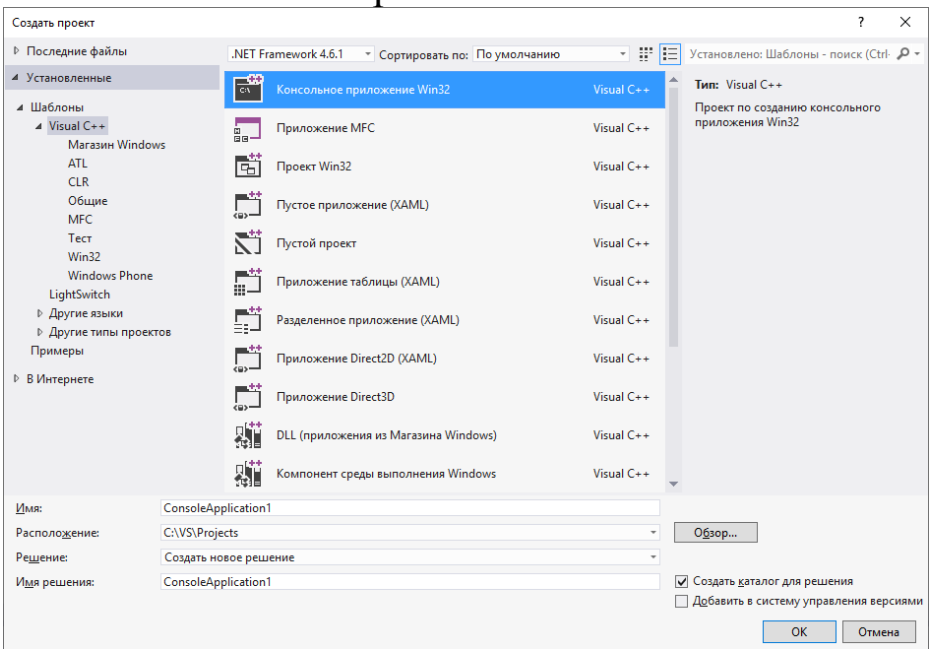


Рис. 1. Выбор типа проекта

Важно корректно заполнить поля Имя (Name), Расположение (Location) и Имя решения (Solution Name), которые находятся в нижней части окна. Поле Имя (Name) – Название проекта. При заполнении поля Имя (Name) поле Решение (Solution) заполняется тем же именем автоматически. Не

следует создавать отдельную директорию под свой проект, Visual Studio создаст ее автоматически.

После нажатия кнопки «Ок» на экране появится следующее окно (рис. 2):

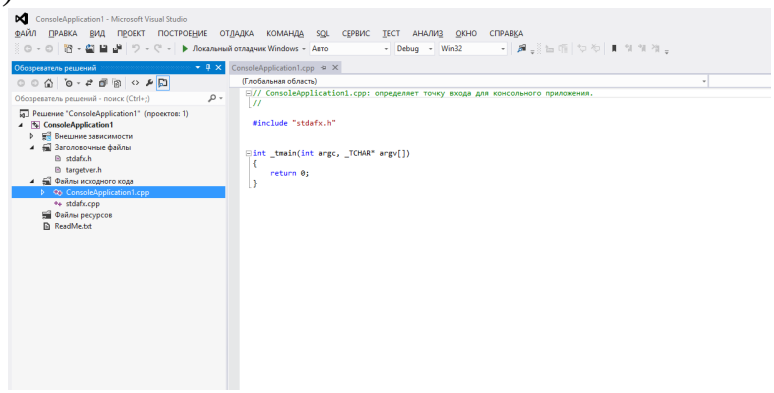


Рис. 2. Пример консольного приложения

В левой части окна находится Обозреватель решений (Solution Explorer). В нем отображаются файлы проекта – код, файлы ресурсов, а также ссылки на используемые сборки.

Для запуска проекта необходимо выбрать пункт меню Отладка – Начать отладку (Debug – Start debugging) или нажать клавишу F5.

Окно Вывод (Output) отобразит результаты запуска. Если при запуске возникли ошибки, то посмотреть их можно в окне Список ошибок (Error List). В нем выводятся все ошибки, возникшие при компиляции.

Для создания пустого проекта необходимо выбрать подтип – «Пустой проект» (рис. 3).

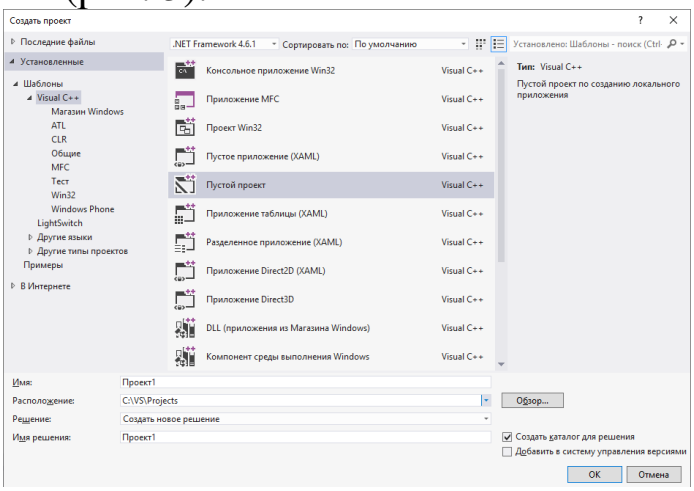


Рис. 3. Создание пустого проекта

При создании пустого проекта файл исходного кода не создается. Его необходимо создать самостоятельно. Для этого в Обозревателе решений (Solution Explorer) необходимо при нажатии правой кнопки мыши выбрать Добавить – Создать элемент (Add – Add new item) (рис. 4).

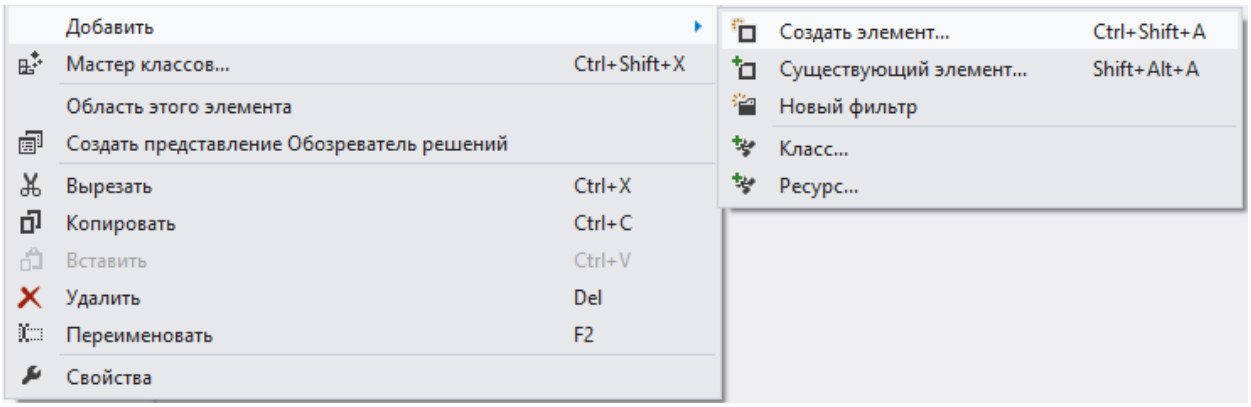


Рис. 4. Добавление файла исходного кода

А затем в появившемся окне выбрать Файл C++ (.cpp) (рис. 5).

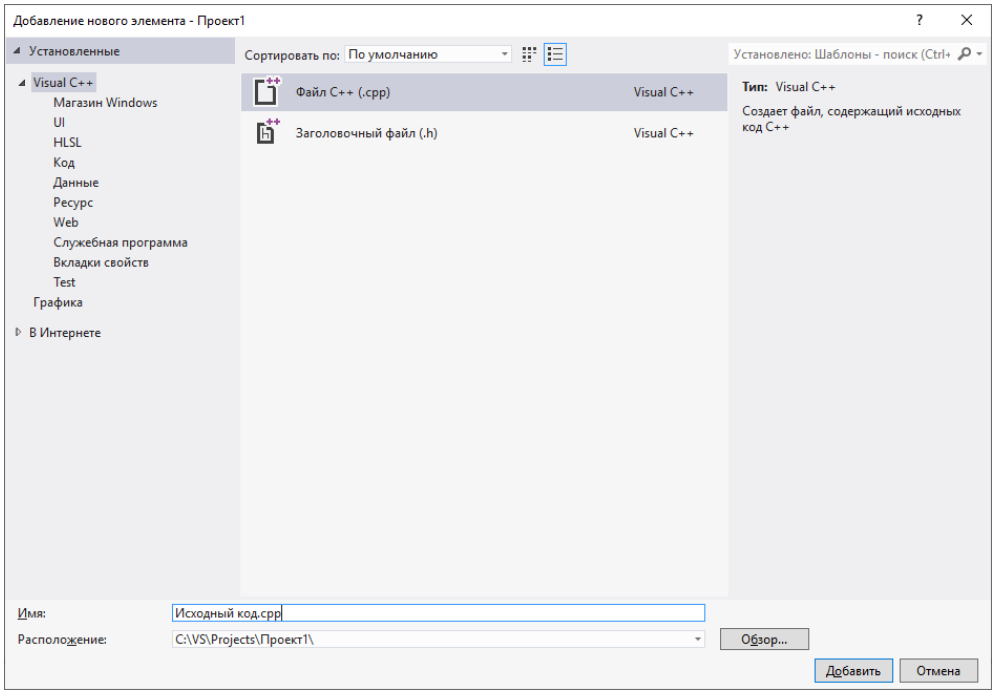


Рис.5. Создание файла исходного кода

Компиляция и запуск проекта осуществляются аналогично компиляции и запуску консольного приложения.

Контрольные вопросы и задания

1. Какие различия между консольным приложением и пустым проектом?
2. Какое имя имеет исполняемый файл созданного проекта?
3. Какие есть способы запуска проекта?
4. Каково назначение Обозревателя решений?
5. С помощью каких действий можно отрыть окно Список ошибок?
6. Чем отличается запуск проекта с отладкой и без отладки?
7. Какие цветовые темы среды разработки существуют?
8. Как поменять цветовую тему среды разработки?
9. Как изменить расположение проектов по умолчанию?
10. Как настроить параметры редактирования текста?
11. Как запустить приложение с определёнными аргументами командной строки?
12. Когда сделать так, чтобы скомпилированное консольное приложение сразу не закрывалось?
13. Для чего в консольном приложении служит строка #include “stdafx.h”?
14. Каким цветов выделяются ключевые слова языка C++?
15. Как запустить пошаговое выполнение программы?
16. Чем отличаются команды пошагового выполнения программы Step Info и Step Over?
17. Как установить контрольную точку (breakpoints)?
18. Как наблюдать значения переменных в момент останова выполнения программы?
19. Какие окна наблюдения значений переменных существуют?
20. Для чего предназначено окно Вывод (Output)?

2. СТАНДАРТНЫЕ ТИПЫ ДАННЫХ

Тип – это множество значений и операций над ними. Представление типа в языке программирования определяет, сколько байт в памяти будет занимать один элемент. В языке C++ определены следующие базовые типы данных:

Логический тип

```
bool
```

Может принимать одну из двух значений true (истина) и false (ложь). В языке C++ значение true представляет собой целое число 1, а false – 0. Занимает в памяти 1 байт (8 бит).

Для логического типа определены следующие операции:

`&&` – конъюнкция,

`||` – дизъюнкция,

`!` – отрицание.

Все операции сравнения принимают значения типа bool. Определены следующие операции сравнения:

> – больше;

< – меньше;

>= – больше или равно;

<= – меньше или равно;

== – равно;

`!=` – не равно.

Символьные типы представляют наборы символов и операций над ними. Каждому символу в C++ соответствует собственный код, который и хранится в памяти ЭВМ. При использовании код переводится компилятором в нужный символ.

Символьный тип

```
char
```

Представляет один символ в кодировке ASCII. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от – 128 до 127 либо от 0 до 255.

Символьный тип

```
signed char
```

Представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любой значение из диапазона от – 128 до 127.

Символьный тип

```
unsigned char
```

Представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от 0 до 255.

Символьный тип

```
wchar_t
```

Представляет расширенный символ. На Windows занимает в памяти 2 байта (16 бит), на Linux – 4 байта (32 бита). Может хранить любое значение из диапазона от 0 до 65 535 (при 2 байтах) либо от 0 до 4 294 967 295 (для 4 байт).

Символьный тип

```
char16_t
```

Представляет один символ в кодировке Unicode. Занимает в памяти 2 байта (16 бит). Может хранить любое значение из диапазона от 0 до 65 535.

Символьный тип

```
char32_t
```

Представляет один символ в кодировке Unicode. Занимает в памяти 4 байта (32 бита). Может хранить любое значение из диапазона от 0 до 4 294 967 295.

Для указания значения символьного типа необходимо указать нужный символ в апострофах, например:

```
'e'
```

Так как все символы в C++ хранятся в виде целого числа, то допустимы арифметические операции +, –, * над символами. Например, для `char` имеем:

```
'e'+1 равно 'f'
3 * 'k' равно 'A'
```

В языке C++ имеется возможность приведения одного типа к другому. Синтаксис приведения типов имеет следующий вид:

```
(<тип>) <значение>

<тип> (<значение>)
```

Например, для `char` любое из определений символа `x` допустимо:

```
'x'
(char)120
char(120)
```

В C++ введены специальные символы, обычно используемые при форматировании отображаемого текста. Список специальных символов приведен в табл. 1.

Таблица 1

Управляющие символы в C++

Название	**Символ**	**Значение**
`alert`	`\a`	Предупреждение (звуковой сигнал)
`backspace`	`\b`	Перемещение курсора на одну позицию назад
`formfeed`	`\f`	Перемещение курсора к следующей логической странице

Название	Символ	Значение
newline	\n	Перемещение курсора на следующую строку
carriage return	\r	Перемещение курсора в начало строки
horizontal tab	\t	Вставка горизонтального TAB-а
vertical tab	\v	Вставка вертикального TAB-а
single quote	\’	Вставка одинарной кавычки (или апострофа)
double quote	\”	Вставка двойной кавычки
backslash	\\	Вставка обратной косой черты (бэкслэша)
question mark	\?	Вставка знака вопроса
octal number	\(number)	Перевод числа из восьмеричной системы счисления в тип char
hex number	\x(number)	Перевод числа из шестнадцатеричной системы счисления в тип char

Целочисленные типы представляют наборы целых чисел и операций над ними.

Целочисленный тип

`short`

Представляет целое число в диапазоне от –32768 до 32767. Занимает в памяти 2 байта (16 бит).

Целочисленный тип

`unsigned short`

Представляет целое число в диапазоне от 0 до 65535. Занимает в памяти 2 байта (16 бит).

Целочисленный тип

`int`

Представляет целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита). Диапазон предельных значений соответственно также может варьироваться от –32768 до 32767 (при 2 байтах) или от –2 147 483 648 до 2 147 483 647 (при 4 байтах).

Целочисленный тип

`unsigned int`

Представляет положительное целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита), и из-за этого диапазон предельных значений может меняться: от 0 до 65535 (для 2 байт) либо от 0 до 4 294 967 295 (для 4 байт).

Целочисленный тип

`long`

Представляет целое число в диапазоне от –2 147 483 648 до 2 147 483 647. Занимает в памяти 4 байта (32 бита).

Целочисленный тип

`unsigned long`

Представляет целое число в диапазоне от 0 до 4 294 967 295. Занимает в памяти 4 байта (32 бита).

Целочисленный тип

`long long`

Представляет целое число в диапазоне от –9 223 372 036 854 775 808 до +9 223 372 036 854 775 807. Занимает в памяти, как правило, 8 байт (64 бита).

Целочисленный тип

`unsigned long long`

Представляет целое число в диапазоне от 0 до 18 446 744 073 709 551 615. Занимает в памяти, как правило, 8 байт (64 бита).

Для целочисленных типов определены следующие арифметические операции:

+ плюс;

– минус;

* умножить;

/ целая часть от деления (13/5 = 2);

% остаток от деления (13%5 = 3).

Приведение к целочисленному типу отбрасывает дробную часть (если таковая имеется).

Типы с плавающей точкой предназначены для работы с вещественными числами, их также называют вещественными типами.

Тип с плавающей точкой

`float`

Представляет вещественное число ординарной точности с плавающей точкой в диапазоне +/– 3,4E – 38 до 3,4E + 38. В памяти занимает 4 байта (32 бита).

Тип с плавающей точкой

`double`

Представляет вещественное число двойной точности с плавающей точкой в диапазоне +/– 1,7E – 308 до 1,7E + 308. В памяти занимает 8 байт (64 бита).

Тип с плавающей точкой

`long double`

Представляет вещественное число двойной точности с плавающей точкой не менее 8 байт (64 бит). В зависимости от размера занимаемой памяти может отличаться диапазон допустимых значений.

Для вещественных типов определены следующие арифметические операции:

+ плюс;

– минус;

* умножить;

/ деление.

Тип

`void`

Предназначен для использования указателей. Указатели void применяются, когда заранее неизвестно о типе данных, которые поступают на вход программы или не определён тип функции.

Контрольные вопросы и задания

1. Назовите все целочисленные типы.
2. Назовите все вещественные типы.
3. Когда обрабатываются строки, начинающиеся с символа #?
4. Назовите все строковые типы.
5. Назовите все символьные типы.

6. Назовите диапазон значений типа long.
7. Назовите диапазон значений типа short.
8. Назовите диапазон значений типа bool.
9. Назовите диапазон значений типа float.
10. Назовите диапазон значений типа double.
11. Назовите диапазон значений типа int.

3. СТРУКТУРА ПРОСТОЙ ПРОГРАММЫ

Программа на языке C++ состоит из функций, описаний и директив препроцессора. Одна из функций должна иметь имя main. Выполнение программы начинается с первого оператора этой функции.

Пример структуры программы, содержащей функцию main:

```
директивы препроцессора
описания
<тип> main{
операторы главной функции
}
```

Каждая программа в C++ обязательно имеет одну функцию, её называют главная, или main-функция, выполнение программы начинается именно с этой функции. Эта функция может иметь тип `void` или любой другой стандартный тип. Из главной функции можно вызывать любые другие функции. Main функция является точкой входа в программу. Функция main обладает следующими свойствами:

– она нигде не может быть использована в программе, не может быть вызвана рекурсивно, нельзя взять ее адрес;

– её нельзя объявлять и нельзя перегружать.

Первая строка программы – директива препроцессора, по которой в текст программы вставляются заголовочные файлы. Все директивы препроцессора начинаются со знака #. Строки, начинающиеся с символа #, обрабатываются препроцессором до компиляции программы. Директива `#include <имя файла>` вставляет содержимое указанного файла в ту точку исходного

файла, где она записана. Включаемый файл также может содержать директивы `#include`. Поиск файла, если не указан полный путь, ведется в стандартных каталогах включаемых файлов. Вместо угловых скобок могут использоваться кавычки (`" "`) – в этом случае поиск файла ведется в каталоге, содержащем исходный файл, а затем уже в стандартных каталогах. Директива `#include` является простейшим средством обеспечения согласованности объявлений в различных файлах, она включает в них информацию об интерфейсе из заголовочных файлов.

Если программа создана как проект консольного приложения, то visual Studio создает заголовочный файл `stdafx.h` или подобный и подключает его директивой `#include`. В случае, когда создается пустой проект с добавлением исходного файла `*.cpp`, дополнительные заголовочные файлы не создаются и приложение является стандартной программой C++. В конце главной функции для того, чтобы увидеть результат работы, можно написать команду `system("pause")`, которая служит задержкой консоли экрана.

После выполнения всех операторов тела главной функции программа завершается. Если необходимо досрочно завершить программу с типом главной функции `void`, то допустимо использовать оператор `return`. Для досрочного завершения программы с указанным типом (отличным от `void`) главной функции необходимо указать `return <значение>`, где значение принадлежит типу `main`. Обычно главную функцию в C++ объявляют с типом `int`, которая всегда завершается командой `return 0;`. В этом случае программа сообщает среде, что она закончила выполнение корректно (возвращает код окончания программы равным нулю).

В рамках данного пособия будут рассматриваться две основные структурные схемы программ.

Нетипизированная схема имеет вид

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

<тело программы>

}
```

Типизированная схема имеет вид

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

<тело программы>

return 0;
}
```

Для улучшения читаемости программного кода в текст программы добавляют комментарии. Комментарий – специальным образом оформленный текст, игнорируемый компилятором. Комментарии обычно используют для заметок программисту. В C++ определены две возможности введения комментария.

Однострочный комментарий имеет следующий вид:

```
// <текст>
```

Весь текст строки программного кода, находящийся справа от `"//"`, является комментарием.

Комментарий с указанием границ имеет следующий вид

```
/* <текст> */
```

В этом случае весь текст программы, находящийся между `/*` и `*/`, считается комментарием, независимо от того, сколько между ними строк.

Контрольные вопросы и задания

1. Напишите структуру главной функции.
2. Какое имя присваивается главной функции?
3. Могут ли быть другие функции в программе?
4. Какая функция является точкой входа в программу?
5. Что такое директивы препроцесора?
6. Назовите назначение директивы #include <имя файла>.
7. Назовите свойства главной функции.
8. Можно ли опустить return в главной функции?
9. Можно или нельзя перегружать главную функцию?

4. ПЕРЕМЕННЫЕ

Переменные в языках программирования используют для хранения различных данных.

C++ является типизированным языком программирования, поэтому все переменные перед использованием должны быть объявлены. Объявление переменной в C++ может располагаться в любой части программы. Структура объявления переменной следующая:

```
<тип> <имя переменной>;
```

В качестве имени переменной может выступать любой идентификатор. Идентификатор – любая последовательность букв, цифр и знака подчеркивания (_), которая начинается с буквы либо знака подчеркивания.

Идентификатор — весьма важное понятие. Этот термин происходит от слова «идентифицировать», т.е. «отождествлять». Поскольку алгоритм, определяющий процесс обработки данных, оперирует с различными программными объектами — переменными величинами, функциями и т.д., то при записи алгоритма приходится как-то ссылаться на используемые объекты. Для этой цели программным объектам даются индивидуальные имена, которые и представляют соответствующие объекты. Именами обозначаются и некоторые атрибуты используемых объектов, например, тип значений,

которые могут принимать программные объекты. Роль таких имен и выполняют идентификаторы. Примеры идентификаторов:

```
x   SUMMA   pi   stepl      Petrov a28cd5.
```

(поскольку пробелы внутри идентификаторов не допускаются, то наличие пробела означает конец идентификатора).

Следующие записи — не идентификаторы, поскольку каждая из них не подходит под синтаксическое определение идентификатора:

```
5f   sum(2)   step.7
```

Ряд слов в языке C++ имеет особое значение и не может использоваться в качестве идентификаторов. Такие зарезервированные слова называются ключевыми (табл. 2).

Таблица 2

Список ключевых слов

asm	auto	bad_cast
bad_typeid	bool	break
case	catch	char
class	const	const_cast
continue	default	delete
do	double	dynamic_cast
else	enum	extern
float	for	friend
goto	if	inline
int	long	mutable

namespace	new	operator
private	protected	public
register	reinterpret_cast	return
short	signed	sizeof
static	static_cast	struct
switch	template	then
this	throw	try
type_info	typedef	typeid
union	unsigned	using
virtual	void	volatile
while	xalloc	

Идентификаторы не имеют какого-либо постоянно присущего им смысла, а используются только в качестве имен программных объектов или их атрибутов, так что в дальнейшем вместо слова «идентификатор» будем использовать более короткий термин «имя».

Имена (идентификаторы) выбираются программистом по своему усмотрению. Чтобы сделать программу решения той или иной задачи более наглядной и понятной, следует избегать кратких, но маловыразительных имен типа `x, t, s` и т.д., а выбирать их так, чтобы имя отражало суть обозначаемого объекта (конечно, в рамках алфавита конкретной реализации языка), например: `summa`, `Time`, `STEP` и т.п.

Синтаксическое определение не накладывает ограничений на длину идентификаторов. Чтобы иметь более наглядные имена, целесообразно использовать строчные и прописные буквы, знак подчеркивания «_»: `Kol_Iter`.

Некоторым идентификаторам заранее предписан вполне определенный смысл. Например, идентификатор `sin` считается именем функции, значение которой равно синусу ее аргумента. Такие идентификаторы называются стандартными.

Имя переменной не должно совпадать с ключевыми словами языка C++. В качестве типа может выступать любой встроенный либо ранее объявленный тип.

Например:

```
int x; //объявлена целочисленная переменная x
double y;//объявлена переменная y типа double
```

Можно объявлять одновременно несколько переменных, например

```
int a,b;
```

Допустимо одновременно объявлять и инициализировать переменную

```
int x = 5;
```

При программировании на языке C++ считается хорошим стилем объявлять переменные непосредственно в той части программы, где она используется.

C++ регистрозависимый язык, поэтому переменные с именами `A` и `a` считаются различными переменными.

Константы, как и переменные, представляют собой ячейки памяти, предназначенные для хранения данных. Но в отличие от переменных, значение константы не может быть изменено в программе. Поэтому при объявлении константы необходимо сразу ее инициализировать.

Для объявления константы используется зарезервированное слово `const`, структура объявления следующая:

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

Например:

```
const int n = 10;
const double pi = 3.14;
```

5. ВВОД / ВЫВОД ДАННЫХ

В стандартной реализации C++ существует 2 основных способа ввода – вывода информации: с помощью потоков, реализованных в Standard Template Library (STL), или с помощью традиционной системы ввода – вывода, которая наследуется от языка C.

Для использования традиционной системы ввода – вывода, наследованной от языка C, в программу необходимо включить заголовочный файл `<cstdio>`. Основными функциями библиотеки `stdio` являются `printf()`, используемая для вывода информации, и `scanf()`, используемая для ввода информации.

Рассмотрим их более подробно.

Функция `scanf()` является функцией ввода данных, которая позволяет считывать данные всех базовых типов, автоматически конвертируя их в нужный внутренний формат.

Структура функции выглядит следующим образом:

```
int scanf(<строка формата>, <объект 1>,
<объект2>, …, <объект N>);
```

Функция возвращает количество считанных элементов или `EOF`, если произошла ошибка чтения.

Строка формата может содержать символы трех типов:

– спецификаторы формата;

– специальные символы;

– прочие символы.

Спецификаторы формата следуют за символом процент и сообщают `scanf()`, данные какого типа будут считаны следующими.

В табл 3. приведены распространенные коды спецификаторов формата.

Таблица 3

Коды спецификаторов в C++

%c	Считать один символ
%d	Считать десятичное число целого типа
%i	Считать десятичное число целого типа
%e	Считать число с плавающей точкой
%f	Считать число с плавающей точкой
%o	Считать восьмеричное число
%s	Считать строку
%x	Считать шестнадцатеричное число
%p	Считать указатель
%n	Принимает целое значение, равное количеству считанных до текущего момента символов
%u	Считывает беззнаковое целое
%[]	Просматривает набор символов

В качестве объекта выступает указатель на переменную, куда должно быть записано введенное значение.

Например, чтобы считать целое число и присвоить его значение переменной x, необходимо воспользоваться следующим вызовом:

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

Функция printf() используется для форматированного вывода информации. Она переводит данные в символьное представление и выводит их на экран. При этом параметры данной функции позволяют программисту настраивать представление на экране. Структуру функции printf() можно описать следующим образом:

```
printf( <строка формата>, <объект 1>,
<объект 2>, … ,<объект N>);
```

Строка формата может состоять из управляющих символов, текста, который непосредственно надо выводить, и встроенных форматов вывода для значений различных типов. Управляющие символы не выводятся на экран, они нужны для изменения

расположения выводимых символов. Основные управляющие символы следующие:

`'\n'` – перевод каретки на новую сроку;

`'\t'` – горизонтальная табуляция;

`'\r'` – возврат на начало строки.

Встроенные форматы позволяют определить вид, в котором информация будет представлена на экране. Формат всегда начинается с символа %. Перечислим основные форматы:

`%i` – целое число со знаком в десятичной системе типа `int`;

`%u` – целое число типа `unsigned int`;

`%x` – целое число со знаком типа `int` в шестнадцатеричной системе счисления;

`%o` – целое число со знаком типа `int` в восьмеричной системе счисления;

`%f` – вещественный формат (числа с плавающей точкой типа `float`);

`%lf` – вещественный формат удвоенной точности (числа с плавающей точкой типа `double`);

`%e` – вещественный формат в экспоненциальной форме (числа с плавающей точкой типа `float`);

`%c` – символьный формат;

`%s` – строковый формат.

Например, команда `printf("%10.3f", 12.234657)` позволяет вывести на экран вещественное значение с тремя знаками после запятой в поле длиной 10 символов. Результат работы такой команды будет число 12.235 на экране.

Команды

```
float x = 10.3568;
printf("Переменная x = %.3f",x);
```

позволяют вывести на экран строку

```
Переменная x = 10.357
```

Пример 1. Определить три целочисленных переменных, определить их значения равными 5, 6, 9 и вывести их значения.

```
#include <stdio.h>
int main()
{
  int a,b,c; // Объявление переменных a,b,c
  a=5;
  b=6;
  c=9;
  printf("a=%d, b=%d, c=%d",a,b,c);
}
```

Результат работы программы:

```
a=5, b=6, c=9
```

Пример 2. Определить три вещественных переменных, задать и вывести их значения.

```
#include <stdio.h>
int main()
{
  float x,y,z;
  x=10.5;
  y=130.67;
  z=54;
  printf("Координаты точки: x:%.2f, y:%.2f,
  z:%.2f", x, y, z);
}
```

Результат работы программы:

```
Координаты точки: x:10.50, y:130.67, z:54.00
```

Пример 3. Вывод текстовой строки.

```
#include <stdio.h>
int main()
{
  printf("\"Текст в кавычках\"");
  printf("\nСодержание кислорода: 100%%");
}
```

Результат работы программы:

```
"Текст в кавычках"
Содержание кислорода: 100%
```

Пример 4. Форматированный вывод

```
#include <stdio.h>
int main()
{
  int a;
  a=11; /* 11 в десятичной равно b в
шестнадцатеричной */
  printf("a-dec=%d, a-hex=%x",a,a);
}
```

Результат работы программы:

```
a-dec=11, a-hex=b
```

Для работы с потоковым вводом – выводом необходимо в программу включить заголовочный файл <iostream>, после чего программист получает доступ ко всей иерархии классов библиотеки iostream.

Класс istream используется для работы с входными потоками. Оператор **>>** используется для извлечения данных из потока. Класс ostream используется для работы с выходными потоками. Оператор **<<** используется для помещения данных в поток.

Класс iostream может обрабатывать как ввод, так и вывод данных, что позволяет ему осуществлять двунаправленный ввод / вывод.

Класс iostream определяет три стандартных потока: входной поток (cin), выходной поток (cout) и поток вывода сообщений об ошибках (cerr). Для их использования необходимо в программу, написанную в Visual Studio, добавить строку

```
using namespace std;
```

Для выполнения операций ввода – вывода используются операции: >> – получить из входного потока и << – поместить в выходной поток.

Для ввода информации используется конструкция вида

```
cin >> идентификатор1 >> идентификатор2 >> …
>> идентификаторN;
```

При этом из входного потока читается последовательность символов, затем она преобразуется к типу идентификатора и полученное значение помещается в идентификатор.

Например:

```
int i, j;
cin >> i >> j;
```

позволяет ввести два целочисленных значения, разделенных пробелом либо символом табуляции, либо символом перехода на новую строку, и записать эти значения в переменные `i` и `j`.

Для вывода информации можно использовать конструкцию вида

```
cout << значение1 << значение2 << … <<
значениеN;
```

Значения преобразуются в последовательность символов и выводятся в выходной поток.

Например:

```
int n;
char c;
cin >> n >> c;
cout << "n= " << n << "c= " << c;
```

В операции с потоками можно помещать функции-манипуляторы. Наиболее распространены следующие манипуляторы:

`endl` – помещение в выходной поток символа конца строки;

`dec` – установка десятичной системы счисления;

`oct` – установка восьмеричной системы счисления;

`hex` – установка шестнадцатеричной системы счисления;

`width` – устанавливает ширину поля вывода;

`precision` – устанавливает количество цифр в числе;

`fixed` – показывает, что установленная точность относится к количеству знаков после запятой.

```
#include <iostream>
using namespace std;
int main()
{
  double a = -112.234;
  double b = 4.3981;
  int c = 18;
  cout << endl << "Вещественные значения:"
<< endl;
  cout.width(10);
  cout << a << endl << b << endl;
  cout.precision(5);
  cout << "Точность 5 знаков" << endl
<< a << endl << b << endl;
  cout << "Точность после запятой" << endl
<< fixed << a << endl << b << endl;
  cout << endl << "Целые значения:" << endl;
  cout<< hex << c << " " << oct << c << " ";
  cout << dec << c << endl;
  return 0;
}
```

Результат работы программы будет следующий:

```
Вещественные значения:
-112.234
4.3981
Точность 5 знаков
-112.23
4.3981
Точность после запятой
-112.23400
4.39810
Целые значения:
0x12 022 +18
```

Контрольные вопросы и задания

1. С каких символов могут начинаться имена переменных?
2. Можно ли в имени переменной использовать какие-либо символы кроме букв?
3. Опишите и инициализируйте переменные вещественного, целого и символьного типов и выведите их значения на экран с указанием типа.
4. Опишите константы вещественного, целого и символьного типов и выведите их значения на экран с указанием типа.
5. Что такое форматная строка?
6. Что содержит форматная строка функции `printf`?
7. Что содержит форматная строка функции `scanf`?
8. Для чего зарезервировано слово `cout` и выражение << ? Приведите пример.
9. Для чего зарезервировано слово `cin` и выражение >> ? Приведите пример.
10. Приведите пример использования функции `printf()` для вывода значений двух целочисленных переменных на экран.
11. Приведите пример использования выходного потока `cout` для вывода значений двух целочисленных переменных на экран.
12. Как выполнить ввод переменных `x` и `y`, где `x` типа `long int`, а `y` типа `double` с помощью функции `scanf`?
13. Как выполнить ввод переменных `x` и `y`, где `x` типа `long int`, а `y` типа `double` с помощью операции >> ?
14. Приведите пример использования функции `printf()` для вывода значения переменной типа `double`, так чтобы после запятой было 3 цифры.
15. Приведите пример использования выходного потока `cout` для вывода значения переменной типа `double`, так чтобы после запятой было 3 цифры.
16. Приведите пример использования функции `scanf()` для ввода одного символа с клавиатуры.

17. Приведите пример использования входного потока `cin` для ввода трех значений символьного типа.

18. Напишите программу для ввода данных в переменные 4 различных типов и вывода этих значений на экран.

19. Напишите программу для ввода 2 целочисленных значений и вывода на экран их суммы, разности и произведения.

20. Определите, что будет выведено на экран в результате выполнения следующего фрагмента:

```
printf(“Введите стороны прямоугольника”);
    scanf (“%f %f”, &a, &b);
    printf(“Результат: a=%9.4f b=% 3d \n”,a,b);
```

21. Что будет выведено функцией:

```
printf(" Среднее арифметическое
последовательности чисел равно: %10.5f
Количество четных элементов последовательности
равно%10.5d ",S/n,k);
```

6. ОПЕРАТОР ПРИСВАИВАНИЯ

Переменная — программный объект, способный принимать значение. Значение переменная получает в процессе выполнения программы, обычно в результате выполнения оператора присваивания. Синтаксис оператора присваивания имеет следующий вид:

```
<переменная> = <выражение>;
```

Выражения состоят из операндов, знаков операций и скобок и используются для вычисления некоторого значения определенного типа. Каждый операнд является в свою очередь выражением или одним из его частных случаев — константой или переменной.

Знаки операций определяются типами своих операндов.

Операции выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки. Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операции присваивания выполняются справа налево, остальные — слева направо. Например, a + b + c означает (a + b) + c.

Результат вычисления выражения характеризуется значением и типом. Например, деление целых чисел дает в результате целое число. Дробная часть результата, если она есть, отбрасывается.

В выражение могут входить операнды различных типов. Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип. Если операнды разного типа, перед вычислениями выполняются преобразования типов по определенным правилам, обеспечивающим преобразование более коротких типов в более длинные для сохранения значимости и точности.

Преобразования бывают двух типов:

1) изменяющие внутреннее представление величин (с потерей точности или без потери точности);

2) изменяющие только интерпретацию внутреннего представления.

К первому типу относится, например, преобразование целого числа в вещественное (без потери точности) и, наоборот, (возможно, с потерей точности), ко второму — преобразование знакового целого в беззнаковое.

В любом случае величины типов `char`, `signed char`, `unsigned char`, `short int` и `unsigned short int` преобразуются в тип `int`, если он может представить все значения, или в `unsigned int` в противном случае.

После этого операнды преобразуются к типу наиболее длинного из них, и он используется как тип результата. Программист может задать преобразования типа и явным образом.

При организации вычислений желательно позаботиться о соответствии типов аргументов и результата вычислений.

В случаях, когда в выражении используются переменные разных типов, происходит неявное преобразование к типу «большего» операнда.

Все переменные типа `char` и `short int` преобразуются к типу `int`.

Все переменные типа `float` преобразуются к типу `double`.

В результате каждая пара операндов будет иметь одинаковый тип и результат каждой операции будет совпадать по типу с операндами.

Пример преобразования типов данных представлен в табл. 4.

Таблица 4

Преобразование типов

x	y	Результат операции + – * /	Пример
int	int	int	15/2=7
int	float	float	15/2=7.5
float	int	float	15/2=7.5

Переменная и выражение должны иметь совместимые типы. Пусть, например, имеются описания:

```
int a=1, b=2, c=3;
float x=4.2, y=5.3, z=6.4;
a=b+c;  // a=5
x=y+z;  // x=11.7
x:=y+a  // x=6.3
b=x+y; /*Ошибка: b типа int, выражение типа
                 float*/
```

В качестве операндов выражений могут указываться функции, описанные в стандартной математической библиотеке `<cmath>`.

Приведем некоторые из них:

`int abs (int к)` возвращает модуль целочисленного аргумента.

`double fabs(double x)` возвращает модуль вещественного аргумента.

Тригонометрические функции возвращают выраженную в радианах величину угла. Аргумент функции должен находиться в диапазоне от –1 до 1.

```
double cos (double x);
double sin (double x);
double tan (double x);
long double cosl(long double x);
long double sinl(long double x);
long double tanl(long double x);
```

Обратные тригонометрические функции:

```
double acos (double x);
double asin (double x);
double atan (double x);
long double acosl(long double x);
long double asinl(long double x);
long double atanl(long double x);
```

Вычисление e^x — экспонента аргумента (e — основание натурального логарифма):

```
double exp(double x);
long double exp(long double lx)
```

Возведение в степень x^y:

```
double pow (double x, double y);
long double powl(long double (x), long
double (y));
```

Квадратный корень аргумента:

```
double sqrt(double к)
```

Вычисление натурального логарифма и логарифма по основанию 10:

```
double log(double x);
double log10(double x);
long double logl(long double (x));
long double log10l(long double (x));
```

Для округления используются функции
`double round(double x)` — обычное округление;

`double ceil(double x)` — округление «вверх»;

`double floor(double x)` — округление «вниз»;

`double trunc(double x)` — «отбрасывание» дробной части.

В библиотеке `<cstdlib>` описаны генераторы случайных чисел.

`int rand(void);` – возвращает случайное целое число в диапазоне от 0 до `RAND_MAX`.

Перед первым обращением к функции `rand` необходимо инициализировать генератор случайных чисел вызовом функции `void srand(unsigned k)`. Обычно в качестве параметра функции используют переменную, значение которой предсказать заранее нельзя, например, это может быть текущее время.

Имеется возможность заставить выражение принять определенный тип с помощью оператора явных преобразований. Эта операция имеет следующий вид:

```
(тип) выражение,
```

где `тип` — это один из стандартных типов данных или определяемый пользователем тип. Например, `float(a+b)` – преобразовать значение суммы к вещественному типу.

Оператор принудительных преобразований — это унарный оператор, имеющий такой же приоритет, как и остальные унарные операторы.

Присвоенное значение переменная сохраняет до тех пор, пока ей не будет присвоено новое текущее значение — при этом предыдущее ее значение (если оно было определено) безвозвратно теряется. С каждой переменной связывается определенный тип значений, которые она может принимать. Попытка присвоить переменной значение несовместимого типа квалифицируется как ошибка в программе.

С точки зрения синтаксиса, переменная (в простейшем случае) — это идентификатор, который сопоставлен переменной в качестве ее имени. Это имя используется для ссылки на значение переменной. Другими словами, имя в тексте программы представляет значение этой переменной.

Что касается семантики понятия (переменная), то можно считать, что в вычислительной системе имеется несколько типов «запоминающих ячеек», каждая из которых способна хранить значения определенного типа. К началу выполнения программы каждой из используемых в ней переменных выделяется ячейка соответствующего типа и этой ячейке дается имя, совпадающее с именем самой переменной.

С точки зрения языка C++ операция присваивания является выражением, значение которой равно присвоенному значению. Например:

```
x=a=b=5
```

соответствует `x=(a=(b=5))` и последовательно присваивает значение `5` переменным `x, a, b`.

В практике программирования часты случаи, когда к объекту применяется некоторая операция, а результат этой операции присваивается тому же объекту. Например:

```
sum = sum + R;
```

Для более компактной записи C++ предоставляет составные операции присваивания. С использованием такого оператора данный пример можно переписать следующим образом:

```
sum += R;
```

Общий синтаксис составного оператора присваивания таков:

```
a op = b;
```

Эта запись в точности эквивалентна записи

```
a = a op b.
```

Для изменения значения переменной на единицу введены дополнительно префиксные и постфиксные операции ++ и --. Следующие операции увеличения значения переменной `i` на единицу эквивалентны:

```
i=i+1;
i+=1;
i++;
++i;
```

Следующие операции уменьшения значения переменной `i` на единицу эквивалентны:

```
i=i+1;
i+=1;
i++;
++i;
```

Как и операция присваивания префиксные и постфиксные операции ++ и – также возвращают значения. Префиксные операции вначале изменяют значение переменной, а потом возвращают новое значение. Постфиксные вначале возвращают значение переменной, а потом ее изменяют. Данное описание демонстрирует пример

```
int a,b;
a=5; //a равно 5, b – неопределенно
b=a++; //a равно 6, b равно 5
a=--b; //a равно 4, b равно 4
b=a--; //a равно 3, b равно 4
a=++b; //a равно 5, b равно 5
```

C++ поддерживает все существующие битовые операторы. Битовые операторы определены для целочисленных типов и не могут использоваться с float, double, long double, void и другими сложными типами. В табл. 5 приведены имеющиеся операторы.

Таблица 5

Битовые операторы

Оператор	**Действие**
&	Поразрядная конъюнкция
|	Поразрядная дизъюнкция
^	Исключающее ИЛИ

Оператор	Действие
~	Дополнение
>>	Сдвиг вправо
<<	Сдвиг влево

Побитовые операции организованы наиболее эффективно и поэтому вместо деления на 2 целесообразно использовать сдвиг на 1 бит (`x>>1` вместо `x/2`) вместо `x%2` – `x&1`.

Пример. Для заданного x вычислить выражения $f = 3x^2 + \sin(x)$:

```
#include <iostream>
using namespace std;
int main()
{
 double x, f;
 cout << "x=" <<;
 cin >> x;
 f = 3*x*x + sin(x);
 cout << "f=" << f;
return 0;
}
```

Для того чтобы поменять значения двух переменных `a` и `b`, обычно используется некоторая вспомогательная переменная для сохранения прежнего значения одной из переменных.

```
tmp = a;  // Сохраняем значение переменной a в tmp
a = b;    // Присваиваем переменной a новое
значение
b = tmp;  //Присваиваем переменной b значение tmp
```

Контрольные вопросы и задания

1. Произвести «циклический сдвиг» значений трех переменных *a, b, c* на одну позицию вправо, в результате чего переменная *a* должна получить значение *c*, переменная *b* — значение *a* и, наконец, переменная *c* — значение *b*.

2. С начала суток прошло *N* секунд (*N* — целое). Найти:

– количество полных минут, прошедших с начала суток;

– количество полных часов, прошедших с начала суток.

3. Даны длины ребер a, b, c прямоугольного параллелепипеда. Найти его объем $V = a \cdot b \cdot c$ и площадь поверхности $S = 2 \cdot (a \cdot b + b \cdot c + a \cdot c)$.

4. Даны два ненулевых числа A и B. Найти сумму разность, произведение и частное их квадратов.

5. С начала суток прошло *N* секунд (*N* — целое). Найти:

– количество секунд, прошедших с начала последнего часа;

– количество полных минут, прошедших с начала последнего часа.

6. Найти длину окружности L и площадь круга S заданного радиуса R: $L = 2 \cdot \pi \cdot R$, $S = \pi \cdot R^2$.

7. Даны два числа a и b. Найти их среднее арифметическое: (a + b)/2 и среднее геометрическое: $\sqrt{a \cdot b}$.

8. Даны два ненулевых числа A и B. Найти сумму разность, произведение и частное их модулей.

9. Дни недели пронумерованы следующим образом: 0 — воскресенье, 1 — понедельник, 2 — вторник, … , 6 — суббота. Дано целое число K, лежащее в диапазоне 1–365. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было:

– понедельником;

– четвергом.

10. Дан номер некоторого года (целое положительное число). Определить соответствующий ему номер столетия, учитывая, что, например, началом XX столетия был 1901 год.

11. Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P.

12. Даны катет прямоугольного треугольника a и гипотенуза c. Найти другой катет b и периметр P.

13. Даны два круга с общим центром и радиусами R_1 и R_2 ($R_1 > R_2$). Найти площади этих кругов S_1 и S_2, а также площадь S_3 кольца, внешний радиус которого равен R_1, а внутренний радиус равен R_2: $S_1 = \pi \cdot R_1^2$, $S_2 = \pi \cdot R_2^2$, $S_3 = S_1 - S_2$.

14. Даны три точки A, B, C на числовой оси. Найти длины отрезков |AC| и |BC| и их сумму.

15. Даны координаты двух противоположных вершин прямоугольника: (x_1, y_1), (x_2, y_2). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.

16. Даны три точки A, B, C на числовой оси. Точка C расположена между точками A и B. Найти произведение длин отрезков |AC| и |BC|.

17. Найти расстояние между двумя точками с заданными координатами (x_1, y_1), (x_2, y_2) на плоскости. Расстояние вычисляется по формуле $D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

18. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью, учитывая, что $L = 2 \cdot \pi \cdot R$, $S = \pi \cdot R^2$.

19. Дана площадь S круга. Найти его диаметр D и длину L окружности, ограничивающей этот круг, учитывая, что $L = 2 \cdot \pi \cdot R$, $S = \pi \cdot R^2$.

20. Даны координаты трех вершин треугольника: (x_1, y_1), (x_2, y_2), (x_3, y_3). Найти его периметр и площадь. Для нахождения площади треугольника со сторонами a, b, c использовать формулу Герона: $S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = (a + b + c)/2$ — полупериметр.

7. УСЛОВНЫЙ ОПЕРАТОР, УСЛОВНОЕ ВЫРАЖЕНИЕ

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие.

Синтаксис условного оператора языка C++:

```
if (<условие>) <оператор1>;
else <оператор2>;
```

В качестве `условия` может выступать любое выражение, принимающее логическое (булевское) или числовое значение. Вычисляется значение `условия`, если значение равно `true` или не равно нулю (в случае числового выражения), то выполняется `оператор1`, если вычисленное значение равно `false` или равно нулю – `оператор2`.

Например:

```
if (a < 10) //если a меньше 10
  a++;      // то увеличить a на 1
else a *= a; // иначе a умножить на себя
cout << a;
```

Если вместо `оператора1` или `оператора2` используется несколько операторов, то необходимы операторные скобки {…}. Все операторы, ограниченные операторными скобками, с точки зрения компилятора, рассматриваются как один большой оператор или блок команд. Особо отметим, если переменная введена (описана) в каком-то блоке, то она допустима только в этом блоке и внутренних, по отношению к данному, блоках, начиная с места описания и ниже по тексту программы:

```
int a=5;
{
  int b;
  b=a+3;
  {
    int c=a*b;
    cout << a << b << c; //без ошибок
  }
  b=a+c; // ошибка!!! c здесь не определено
}
a=a+b; // ошибка!!! b здесь не определено
```

Часть `else оператор;` условного оператора может быть опущена. Тогда при значении `true` условного выражения выполняется `оператор1`, в противном случае условный оператор ничего не выполняет.

```
if (<условие>) <оператор1>;
```

Например:

```
if (x < 0)
 x = -x;
 cout << x;
```

Пример 1. Вывести на экран, если «верно» – введённое пользователем число является положительным, чётным и не кратным 5, в противном случае – «не является».

```
#include <iostream>
using namespace std;
int main()
{
 setlocale (LC_ALL, "Russian"); /*включение
 поддержки русскоязычного отображения символов*/
 int a;
 cout << "Введите число:";
 cin >> a;
 if (a > 0 && a % 2 == 0 && a % 5 != 0)
     cout << "верно";
 else cout << " не является" << endl;
 system("pause"); /*предотвращает закрытие окна
 вывода пока пользователь не нажмет любую
клавишу*/
 return 0;
}
```

В условном операторе нет ограничений на операторы, следовательно, в качестве оператора может быть условный оператор. Такой оператор называется вложенным условным оператором.

Например:

```
if (<условие1>)
 if (<условие2>) <оператор21>;
 else <оператор22>;
else if (<условие3>) <оператор31>;
     else <оператор32>;
```

При использовании вложенной формы условного оператора раздел `else` связывается с ближайшим оператором `if`. Если нужно изменить порядок – используйте операторные скобки.

Пример 2. Найти и вывести на экран наименьшее из 3 введённых пользователем чисел.

```
#include <iostream>
using namespace std;
int main()
{
 setlocale (LC_ALL, "Russian");
 int a, b, c;
 cout << "Введите числа:";
 cin >> a >> b >> c;
 if (a < b && a < c) cout << a << endl;
 else if (b < c) cout << b << endl;
      else cout << c << endl;
 system("pause");
 return 0;
}
```

Для использования простого выбора при вычислении выражений в языке C++ введено условное выражение. Условное выражение принимает одно из двух значений в зависимости от выполнения условия. Условное выражение имеет следующий вид:

```
<условие> ? <выражение1> : <выражение2>
```

Если условие истинно, будет вычислено выражение1, и его значение будет взято в качестве значения условного выражения. Если условие ложно, то будет использовано выражение2.

Например, возможно следующее решение примера 1 с использованием условного выражения:

```
#include <iostream>
using namespace std;
int main()
{
 setlocale (LC_ALL, "Russian");
 int a;
 cout << "Введите число:";
 cin >> a;
 cout<< (a > 0 && a % 2 == 0 && a % 5 != 0 ?
                  "верно" : " не является") <<
endl;
```

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

Также возможно следующее решение примера 2 с использованием условного выражения:

```
#include <iostream>
using namespace std;
int main()
{
 setlocale (LC_ALL, "Russian");
 int a, b, c;
 cout << "Введите числа:";
 cin >> a >> b >> c;
 cout <<(a < b && a < c ? a : (b < c ? b : c)) <<
                      endl;
 system("pause");
 return 0;
}
```

В случае, когда в программе необходимо записать несколько условий, относящихся к одной переменной, то удобнее использовать оператор выбора `switch` вместо нескольких операторов `if-else`.

Синтаксис оператора switch:

```
switch (<выражение>)
{
 case <константа1>: <операторы1>;
 case <константа2>: <операторы2>;
 …
 case <константаN>: <операторыN>;
 [default: <операторы;>]
}
```

`Выражение`, `константа1`, `константа2` и последующие константы должны быть целочисленного типа.

Вычисляется значение `выражения`, затем его значение последовательно проверяется на равенство со значениями константных выражений (`константа1`, `константа2`, …,

константаN). Если значение выражения и некоторой константаi совпадают, то выполняются операторы операторыi, …, операторыN, операторы (использовать операторные скобки не обязательно).

Ветвь default в операторе выбора может отсутствовать. Если она присутствует, то ее операторы выполняются даже в том случае, если не нашлись совпадения ни с одной из констант.

Рассмотрим пример:

```
#include <iostream>
using namespace std;
int main()
{
 setlocale (LC_ALL, "Russian");
 int a;
 cout << "Введите число: ";
 cin >> a;
 switch (a)
 {
     case 1: cout << "Выполнено1" << endl;
     case 2: cout << "Выполнено2" << endl;
     case 3: cout << "Выполнено3" << endl;
     case 4: cout << "Выполнено4" << endl;
     default: cout << "Выполнено" << endl;
 }
 return 0;
}
```

При вводе числа 2 будет выведено:

```
Выполнено2
Выполнено3
Выполнено4
Выполнено
```

При вводе числа 8 будет выведено:

```
Выполнено
```

После прекращения выполнения оператора выбора используется оператор `break`. Если в процессе выполнения операторов, указанных в `switch`, встречается оператор `break`, то выполнение оператора выбора прекращается и управление передается следующему за ним оператору программы. В частности, если при выполнении оператора выбора допустима только одна альтернатива, то каждый набор операторов должен заканчиваться командой `break`.

Break необходимо использовать во всех случаях, если каждый случай нужно обрабатывать отдельно. Если `break` не закрывает какой-то случай, то выполнение кода продолжится до следующего.

Пример 3. Для введенной цифры вывести ее строчное представление.

```
#include <iostream>
using namespace std;
void main()
{
   int n;
   setlocale (LC_ALL, "Russian");
   cout <<"\n Введите цифру:";
   cin >> i;
   cout << '\n';
   switch (i)
   {
       case 0: cout << "ноль"; break;
       case 1: cout << "один"; break;
       case 2: cout << "два"; break;
       case 3: cout << "три"; break;
       case 4: cout << "четыре"; break;
       case 5: cout << "пять"; break;
       case 6: cout << "шесть"; break;
       case 7: cout << "семь"; break;
       case 8: cout << "восемь"; break;
       case 9: cout << "девять"; break;
       default: cout << "ОШИБКА!";
```

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

Если для нескольких константных значений оператора `switch` необходимо указать одни и те же операторы, то варианты `case` возможно объединять в виде:

```
case <константа1>: case <константа2>:...
:case <константаN>: <операторы>
```

Пример 4. Для введенной цифры указать ее четность.

```
#include <iostream>
using namespace std;
void main()
{
   int n;
   setlocale (LC_ALL, "Russian");
   cout <<"\n Введите цифру:";
   cin >> i;
   cout << '\n';
   switch (i)
   {
     case 0: case 2: case 4:
     case 6: case 8: cout <<
"четно\n";break;
     default: cout << "нечетно!\n";
   }
system("pause");
return;
}
```

Контрольные вопросы и задания

1. Пользователь вводит с клавиатуры 3 целых числа. Необходимо написать программу, которая выводит на экран только чётные числа.

2. Пользователь вводит с клавиатуры 3 целых числа. Необходимо написать программу, которая выводит на экран квадрат наибольшего из введённых чисел.

3. Пользователь вводит 2 целых числа. Необходимо написать программу, которая выводит на экран сумму этих чисел, если они оба положительные. В противном случае вывести на экран модуль суммы.

4. Пользователь вводит 2 целых числа. Необходимо написать программу, которая проверяет, делится ли первое число на второе без остатка и выводит сообщение об этом.

5. Пользователь вводит с клавиатуры координаты точки (x, y). Необходимо написать программу, которая проверяет, какой четверти принадлежит точка.

6. Пользователь вводит 2 целых числа. Необходимо написать программу, которая считает сумму квадратов введённых чисел и квадрат суммы и выводит наибольшую сумму на экран.

7. Пользователь вводит с клавиатуры два числа. Необходимо написать программу, которая проверяет, делится ли каждое число нацело на 2 и 3, и выводит сообщение об этом. Если же число не делится, выводит сообщение об этом и остаток от деления.

8. Пользователь вводит с клавиатуры число от 1 до 20. Если оно принадлежит интервалу [1, 10], вывести на экран квадрат этого числа, если принадлежит интервалу [11, 20], вывести на экран введённое число в кубе.

9. Пользователь вводит с клавиатуры 3 числа от – 0 до 10. Написать программу, которая считает сумму только нечётных положительных чисел.

10. Пользователь вводит с клавиатуры 2 числа. Написать программу, которая проверяет, является ли первое число квадратом второго, и выводит соответствующее сообщение.

11. Пользователь вводит с клавиатуры 2 числа от – 20 до 20. Если оба числа отрицательные, вывести их сумму. Если оба положительные, вывести модуль их разности. Если хотя бы одно отрицательное, посчитать сумму квадратов.

12. По длинам трех отрезков, введенных пользователем, определить возможность существования треугольника, составленного из этих отрезков. Если такой треугольник существует, то определить, является ли он разносторонним, равнобедренным или равносторонним.

13. Составить расписание на неделю. Пользователь вводит порядковый номер дня недели и у него на экране отображается то, что запланировано на этот день. Написать программу необходимо с помощью оператора switch.

14. Пользователь вводит порядковый номер пальца руки. Необходимо показать его название на экране. Написать программу необходимо с помощью оператора switch.

15. Пользователь вводит код города (Москва (905) – 4,15 р., Ростов (194) – 1,98 р., Краснодар (491) – 2,69 р.) Написать программу, вычисляющую стоимость 10 – минутного междугороднего разговора в зависимости от кода города и вывести на экран. Написать программу необходимо с помощью оператора switch.

16. Пользователь вводит с клавиатуры число от 1 до 3, которое является номером функции. По значению введённого числа вычислить значение соответствующей функции:

1) $2x + 8 = 4$; 2) $5x + 2 = 17$; 3) $15 – 3x = 18$.

Написать программу необходимо с помощью оператора switch.

17. Пользователь вводит с клавиатуры целое число от 1 до 5. Необходимо с помощью оператора switch написать программу, которая выводит название введённого числа.

18. Пользователь вводит символ. Необходимо с помощью оператора switch написать программу, которая выводит на экран сообщение о том, что было введено: строчная буква, заглавная буква или число.

19. Пользователь вводит номер месяца. Необходимо с помощью оператора switch написать программу, которая выводит название месяца.

20. Пользователь вводит с клавиатуры 2 числа, и число n от 1 до 4 (номер операции, которую нужно выполнить). Необходимо с помощью оператора switch написать программу, которая:

– складывает;
– вычитает;
– умножает;
– делит.

Результат запрошенной операции вывести на экран.

8. ОПЕРАТОР ЦИКЛА WHILE

Если в программе необходимо повторить несколько раз некоторый фрагмент кода, но число повторов заранее неизвестно, а зависит от условия, то следует использовать оператор цикла с предусловием.

Оператор цикла с предусловием имеет следующий синтаксис:

```
while (<условие>) <оператор>;
```

Цикл while объявляется с использованием ключевого слова `while`. В начале цикла вычисляется условие. Если его значение `true` (не ноль), тогда выполняется оператор, называемый телом цикла. Однако, в отличие от оператора `if`, после завершения выполнения тела цикла управление возвращается обратно к началу оператора `while`; процесс вычисления и проверки условия повторяется. Если условие снова `true`, тогда тело цикла выполняется еще раз. Важно, чтобы рано или поздно условие не выполнилось, иначе получится бесконечный цикл (зацикливание).

Одно выполнение тела цикла называется итерацией цикла.

Пример 1. Дано натуральное число. Найти сумму его цифр.

```
#include <iostream>
using namespace std;
void main()
{
 setlocale (LC_ALL, "Russian");
 int a;
 cout<<"Введите  число";
 cin>>a;
```

```
 int s=0;
 while (a!=0) /*действия повторяются, пока
число не обратится в ноль*/
 {
   s+= a % 10;  /*выделение последней цифры
                  числа*/
   a/= 10;  // отбрасывание последней цифры
}
cout<<s;
}
```

Пример 2. Дано целое число. Найти количество цифр числа.

```
#include <iostream>
using namespace std;
void main()
{
 setlocale (LC_ALL, "Russian");
 int x, y, count=0;
 cin >> x;
 /* после выделения цифр из числа, оно
обращается в ноль, поэтому сохраняем
исходное число*/
 y = x;
 if (x) //цикл выполняется если x!=0
  while (y) { // пока y!=0
   count++;
   y /= 10;
  }
 else
  count = 1; //если x равен 0, то одна цифра
 cout << "Кол-во " << x << " : " << count;
}
```

Пример 3. Дано натуральное число. Найти сумму его нечетных цифр.

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

```
void main()
{
 setlocale (LC_ALL, "Russian");
 int a;
 cout<<"Введите  число";
 cin>>a;
 int s=0;
 while (a){
  if (a%10%2!=0) s+= a % 10;
  a/= 10;
 }
 cout<<s;
}
```

Остановимся на условии `a%10%2!=0`. В данном случае выделяется последняя цифра числа (`a%10`), затем выделяется остаток от ее деления на 2 и сравнивается на неравенство с нулем. Сравнение с нулем не обязательно, т.е. в качестве условного выражения допустимо `a%10%2`.

Напомним, что четность числа определяется по последней цифре, следовательно, для проверки нечетности достаточно указать `a%2` (`a%2!= 0`).

В примере используется операция деления, которая по определению является долгой, в смысле реализации, операцией. Для оптимизации проверки на четность / нечетность целесообразно использовать поразрядную операцию `&`. Так как в двоичном преставлении четного числа последний бит равен нулю (нечетного – 1), то нулевое значение поразрядной конъюнкции числа и единицы указывает на его четность (единичное значение – нечетность). Таким образом, оптимальным решением проверки на нечетность числа `a` будет выражение `a&1`, а для четности – `!(a&1)`.

Пример 4. Дано натуральное число. Верно ли, что в его записи нет нулей?

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

```
{
 setlocale (LC_ALL, "Russian");
 int a;
 cout<<"Введите  число";
 cin>>a;
 int k=0;
 while (a) /*действия повторяются, пока
число  не обратится в ноль*/
 {
  if ( a % 10==0) k++; //подсчет нулевых
цифр
  a/= 10;
 }
If (k==0) cout<<"Верно";
 else cout<<”Неверно”;
}
```

В предложенном решении проводится подсчет количества цифр 0 в числе, а затем найденное количество сравнивается с нулем. Условие задачи позволяет прекратить выполнение цикла с помощью оператора break, если найден хоть один ноль. Тогда решение можно реализовать в виде:

```
#include <iostream>
using namespace std;
void main()
{
 setlocale(LC_ALL, "Russian");
 int a;
 cout << "Введите  число";
 cin >> a;
 int k = 0;
 while (a){
  if (a % 10 == 0) {
        k++;
        break;
    }
    a /= 10;
```

```
 }
 if(k == 0) cout << "Верно";
 else cout << "Неверно";
}
```

В последнем примере можно заметить, что если в числе найдена цифра 0, то цикл прекращает свою работу и значение переменной a отлично от нуля. Данный факт можно использовать как проверочный признак:

```
#include <iostream>
using namespace std;
void main()
{
 setlocale(LC_ALL, "Russian");
 int a;
 cout << "Введите  число";
 cin >> a;
 while (a) {
     if (a % 10 == 0) break;
     a /= 10;
 }
 if(a) cout << "Неверно"; //остались цифры
   else cout << "Верно";
}
```

Вариант решения без использования оператора break:

```
#include <iostream>
using namespace std;
void main()
{
 setlocale(LC_ALL, "Russian");
 int a;
 cout << "Введите  число";
 cin >> a;
 while (a % 10)//пока не найден 0 или a не 0
  a /= 10;
 if(a) cout << "Неверно";
```

```
   else cout << "Верно";
}
```

Рассмотрим следующий тип задач. На вход поступают элементы последовательности, их количество неизвестно, но указан маркер окончания последовательности. В этом случае удобно использовать цикл с предусловием. В условии цикла указывается маркер окончания. При этом необходимо до цикла считать первый элемент последовательности. Схема обработки последовательности имеет вид:

```
cin>>a; /*ввод первого элемента
                последовательности*/
while (a!= маркер) /* проверка ввода маркера
                      окончания*/
{
  <Обработка элемента a>
  cin>>a; /* ввод следующего элемента
           последовательности */

}
```

Пример 5. Дана последовательность целых чисел, оканчивающаяся нулем. Найти сумму не делящихся на 5 чисел последовательности.

```
#include <iostream>
using namespace std;
void main() {
 int s = 0, a;
 cin >> a;
 while (a != 0) {
  if (a % 5) s += a;
  cin >> a;
 }
 cout << s;
}
```

Пример 6. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти наибольшее число.

```
#include  <iostream>
using namespace std;
void main() {
 int a, max;
 cin>>a;
 max=a;
 while (a!= 0){
  if (a>max) max=a;
  cin>>a;
 }
 cout<<max;
}
```

В последовательности, ограниченной маркером, также можно одновременно обрабатывать два и более соседних элементов последовательности. До цикла вводится первый элемент последовательности, в цикле вводится очередной элемент и выполняется присваивание. В зависимости от решаемой задачи возможны следующие схемы обработки последовательности:

Схема 1. Значение маркера не влияет на результат

```
cin>>a; /*ввод первого элемента
                      последовательности*/
while (a!= маркер){
 cin>>b;   //  ввод очередного элемента
 <Обработка элементов a и b>
 a=b; /* присваивание второго элемента в
                  переменную первого элемента*/
}
```

Схема 2. Значение маркера влияет на результат (a – предыдущее значение, b – текущее значение последовательности)

```
cin>>a; /*ввод первого элемента
                      последовательности*/
while (a!= маркер){
 cin>>b;   //  ввод очередного элемента
 if (b!= маркер)
    <Обработка элементов a и b>
```

```
 a=b; /* присваивание второго элемента в
                  переменную первого элемента*/
}
```

Схема 3. Имеется возможность указать значение предыдущего элемента так, чтобы он не влиял на результат (a – предыдущее значение, b – текущее значение последовательности)

```
a=<значение, не влияющее на обработку>;
cin>>b;
while (b!= маркер){
 <Обработка элементов a и b>
 a=b;
 cin>>b;
}
```

Пример 7. Дана последовательность целых чисел, оканчивающаяся нулем. Найти сумму положительных чисел, после которых следует отрицательное число.

Решение согласно схеме 1 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 int  s=0, a, b;
 cin>>a;
 while (a!= 0) {
  cin>>b;
  if (a > 0 && b < 0)  s+=a;
  a=b;
 }
 cout<<s;
}
```

Решение согласно схеме 3 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 int  s=0, a, b;
 a=0;
```

```
 cin>>b;
 while (b!= 0) {
  if (a > 0 && b < 0)  s+=a;
  a=b;
  cin>>b;
 }
 cout<<s;
}
```

Пример 8. Дана последовательность целых чисел, оканчивающаяся -1. Верно ли, что последовательность является упорядоченной по возрастанию.

Решение согласно схеме 1 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 int  a, b;
 bool f=true;
 cin>>a;
 while (a!= -1){
  cin>>b;
  if (a > b && b!=-1)  f=false;
  a=b;
}
cout<<f;
}
```

В силу того, что очередное введенное значение влияет на результат, то в решение пришлось добавлять дополнительное условие `b! = -1`.

Более естественно использовать схему 2 или 3.

Решение согласно схеме 2 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 int  a, b;
 bool f=true;
```

```
 cin>>a;
 while (a!= -1){
  cin>>b;
  if (b!=-1){
   if (a > b)  f=false;

  }
  a=b;
}
cout<<f;
}
```

Решение согласно схеме 3 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 int  a, b;
 bool f=true;
 cin>>b;
 a=b-1; /*значение a изначально не нарушает
           упорядоченность*/
 while (b != -1){
  if (a > b)  f=false;
  a=b;
  cin>>b;
 }
cout<<f;
}
```

Пример 9. Дана последовательность целых чисел, оканчивающаяся – 100. Найти среднее арифметическое чисел, оканчивающихся на 5, перед которыми идет четное число. Если таких чисел нет, сообщить об этом.

Решение согласно схеме 3 имеет вид:

```
#include <iostream>
using namespace std;
void main() {
 setlocale (LC_ALL, "Russian");
```

```
 int  s=0, k=0, a, b;
 a=1; /*a-нечетное, поэтому первое значение
           пропустится*/
 cin>>b;
 while (b!= -100){
  if (a%2==0 && b %10==5){s+=b; k++;}
  a=b;
  cin>>b;
 }
 if (k==0) cout<<”Таких чисел нет”;
 else {
  float sr;
  sr=float(s)/k;//обязательно приведение
типа
  cout<<sr;

 }
}
```

Выполнение повторяющихся действий с заранее известным количеством итераций можно выполнить по схеме:

```
int i=<начальное значение>; /* инициализация
                               счетчика i*/
while (i<=<конечное значение>){
 <тело цикла>;
 i+=<шаг изменения>;
}
```

Если шаг изменения счетчика равен 1, то имеем:

```
int i=<начальное значение>;
while (i <= <конечное значение>){
 < тело цикла >;
 ++i;
}
```

Используя свойство операции ++, имеем:

```
int i=<начальное значение>-1;// на 1 меньше
while (++i <= <конечное значение>)
 < тело цикла >;
```

Пример 10. Дана последовательность из N целых чисел. Найти наибольшее число.

```
#include  <iostream>
using namespace std;
void main() {
 setlocale (LC_ALL, "Russian");
 int N, max, i=0;
 cin>>N;
 while (++i<=N){
  int x;
  cin>>x;
  if (i==1 || x>max) max=x;
 }
 if (N) cout<<max;
 else cout<< "неопределено"
}
```

Пример 11. Для заданного значения N и y вычислить $\sum_{k=1}^{N} \frac{(-1)^k(2+k!)}{y^{2k}}$.

Рассмотрим следующий пример решения данной задачи:

```
#include  <iostream>
using namespace std;
void main() {
 int n, k=1, p=1, t=1, f, y;
 float s=0;
 cin>>n>>y;
 while (k<= n){
  if (k%2==0)  f=1; else f=-1;
  p*=k;
  t*=y*y;
  s+=f*(2+p)/t;
  k++;
 }
 cout<<s;
}
```

Здесь k – определяет текущее значение счетчика итераций, p – определяет значение k!=1*2*…*k, t – y^{2k}, f – $(-1)^{2k}$. Отметим специфику языка C++ приводящую к ошибкам нахождения результата:

1. Переменная p объявлена типа int. Следовательно при k>15 возникнет переполнение типа и значение p будет неверно. Аналогичная ситуация возникает при вычислении t.

2. Операнды выражения f*(2+p)/t все имеют тип int, следовательно, и результат тоже имеет тип int, что приводит к ошибке округления.

Первая ошибка возникает в силу ограничения значения типа int. Можно заметить, что при вычислении ищется результат деления большого (с какого-то момента) значения переменной p на большое значение t. В данной ситуации целесообразно хранить значение отношения в вещественном типе, а не отдельно числитель и знаменатель.

Исправление второй ошибки потребует приведения к вещественному типу одного из операндов (например, float(f)*(2+p)/t) или определение одного из операндов как переменной вещественного типа.

В силу определенности значения k на каждой итерации целесообразно заменить if (k%2==0) f=1; else f=-1; на f=-f. С учетом того, что (2+p)/t равно 2/t+p/t, получим:

```
#include <iostream>
using namespace std;
void main() {
 int n, k=0, f=1;
 float s=0, p, t;
 p=2; // p=2/y^2k, здесь k=0
 t=1; // t=k! / y^2k, здесь k=0
 cin>>n>>y;
 while (++k <= n){
  f=-f;
  p/=y*y;
  t/=k/y*y;
```

```
  s+=f*(p+t);
 }
 cout<<s;
}
```

Пример 12. Для заданного y вычислить $\sum_{k=1}^{\infty} \frac{(-1)^k(2+k!)}{y^{2k}}$ с заданной точностью eps. При вычислении суммы ряда считается, что нужная точность достигнута, если модуль очередного слагаемого меньше значения точности eps.

Вариант решения задачи:

```
#include  <iostream>
using namespace std;
void main() {
 int n, k=1, p=1, t=1, f, y;
 float s=0;
 float eps = 1.0E-5;// точность 0.00001
 cin>>n>>y;
 while ((p+t)>eps ){
  f=-f;
  p/=y*y;
  t/=k/y*y;
  s+=f*(p+t);
 }
 cout<<s;
}
```

Контрольные вопросы и задания

1. Дано целое число. Найти количество его нечетных цифр.
2. Дано целое число. Найти сумму его четных цифр.
3. Дано целое число. Верно ли, что в числе нет двоек и троек?
4. Дано целое число. Верно ли, что в числе все цифры одинаковые?

5. Дано целое число. Найти количество заданной цифры в числе.

6. Дано целое число. Найти наименьшую, делящуюся на 3 цифру. Если такой нет, вывести «Нет».

7. Десятичный N-значный (N — четное) номер является «счастливым», если сумма первых (N % 2) цифр этого номера равна сумме последующих его (N% 2) цифр. Является ли заданный номер N «счастливым»?

8. Даны натуральное число N и цифры X, Y. Верно ли, что N содержит X и заканчивается цифрой Y?

9. Определить количество и сумму цифр, больших K заданного натурального числа N.

10. Дано натуральное число N. Получить число M, записанное цифрами числа N в обратном порядке. Например, при N = 125, M = 521.

11. Перевести заданное натуральное число N из десятичной системы счисления в пятеричную систему счисления.

12. Перевести заданное натуральное число N из двоичной системы счисления в десятичную систему счисления.

13. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти произведение последних цифр положительных чисел.

14. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму квадратов двузначных отрицательных чисел. Если таких чисел нет, сообщить об этом.

15. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти наименьшее четное число.

16. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, у которых последняя цифра больше предпоследней. Если таких чисел нет, сообщить об этом.

17. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму квадратов чисел, оканчивающихся на 2 или на 13.

18. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, принадлежащих диапазону [–20; 35]. Если таких чисел нет, сообщить об этом.

19. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, у которых последняя и предпоследняя цифры равны. Если таких чисел нет, сообщить об этом.

20. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти среднее арифметическое положительных чисел, квадрат которых оканчивается на 1, 6 или 9.

21. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму положительных двузначных чисел, за которыми идет оканчивающееся на 11 число.

22. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество троек соседних элементов, где каждое следующее число больше предыдущего.

23. Дана последовательность целых чисел, оканчивающаяся числом 0. Верно ли, что последовательность является знакочередующейся?

24. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, не делящихся на 12, за которыми идет отрицательное число.

25. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму чисел, после которых идет число, не оканчивающееся на 2 и 3.

26. Для заданного натурального числа N найти количество его нечетных делителей.

27. Для заданного натурального числа N найти сумму делителей, заканчивающихся заданной цифрой K.

28. Является ли заданное натуральное число N > 1 простым?

29. Натуральное число N называется совершенным, если оно равно сумме всех своих делителей, исключая само это число (например: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14). Определить, является ли заданное натуральное K совершенным?

30. Натуральное число N называется совершенным, если оно равно сумме всех своих делителей, исключая само это число (например: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14). Найти все совершенные числа отрезка [A, B].

31. Натуральные числа M и N называются дружественными, если сумма собственных делителей одного из них равна другому и наоборот (например, 220 и 284). Являются заданные натуральные числа M и N дружественными?

32. Найти все дружественные числа отрезка [A, B].

33. Разложить натуральное число N на простые множители, указав кратность каждого из них.

34. Найти наибольший общий делитель двух заданных натуральных чисел M и N по алгоритму Евклида.

35. Найти наименьшее общее кратное двух заданных натуральных чисел.

36. Дано вещественное число R — цена 1 кг конфет. Вывести стоимость 1, 2, … , 10 кг конфет.

37. Для заданного натурального значения переменной M вычислить y = M!. По определению, N! = 1·2·3...·(N-1)·N.

38. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

39. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи.

40. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

41. Для заданного x найти сумму первых N чисел ряда:

1) $\cos x = \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{(2k)!}$;

2) $e^x = \sum_{k=0}^{N} \frac{x^k}{k!}$;

3) $sh\ x = \sum_{k=0}^{N} \frac{x^{2k+1}}{(2k+1)!}$;

4) $ln\ (1+x) = \sum_{k=1}^{N} (-1)^{k-1} \frac{x^k}{k}$, $-1 < x \leq 1$;

5) $ln\ \frac{x+1}{x-1} = 2 \sum_{i=0}^{N} \frac{1}{(2i+1)x^{2i+1}}$, $|x| > 1$;

6) $e^{-x^2} = \sum_{k=0}^{N}(-1)^k \frac{x^{2k}}{k!}$;

7) $(1+x)^m = \sum_{k=0}^{N} \frac{m!\, x^k}{(m-k)!\,(k)!}$, $-1 < x < 1$;

8) $arcsin\, x = \sum_{k=0}^{\mathrm{N}} \frac{(2\mathrm{k})!\, x^{2k+1}}{2^{2k}\,(\mathrm{k}!)^2(2\mathrm{k}+1)}$, $|x| \le 1$;

9) $\mathrm{arctg}\, x = \sum_{k=0}^{\mathrm{N}} \frac{(-1)^{\mathrm{k}} x^{2\mathrm{k}+1}}{2\mathrm{k}+1}$, $|x| \le 1$;

10) $tg\, \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\mathrm{N}} \frac{1}{(2k-1)^2 - x}$.

42. Для заданного x найти сумму ряда с точностью eps:

1) $\sin x = \sum_{k=0}^{\infty}(-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!}$;

2) $\mathrm{ch\ x} = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$;

3) $ln\, x = \sum_{k=1}^{\infty}(-1)^{k+1} \frac{(x-1)^k}{k}$, $x \ge 1/2$;

4) $ln\,(1 - \mathrm{x}) = -\sum_{k=1}^{\infty} \frac{\mathrm{x}^k}{k}$, $-1 < \mathrm{x} \le 1$;

5) $ln\, \frac{x+1}{x-1} = 2\sum_{i=0}^{\infty} \frac{1}{(2i+1)x^{2i+1}}$, $|x| > 1$;

6) $\mathrm{e}^{-\mathrm{x}^2} = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{\mathrm{k}!}$;

7) $(1+\mathrm{x})^{\mathrm{m}} = \sum_{k=0}^{\infty} \frac{\mathrm{m}!\, x^k}{(\mathrm{m}-\mathrm{k})!\,(k)!}$, $-1 < \mathrm{x} < 1$;

8) $arcsin\, x = \sum_{k=0}^{\infty} \frac{(2\mathrm{k})!\, x^{2k+1}}{2^{2k}\,(\mathrm{k}!)^2(2\mathrm{k}+1)}$, $|x| \le 1$;

9) $\mathrm{arctg}\, x = \sum_{k=0}^{\infty} \frac{(-1)^{\mathrm{k}} x^{2\mathrm{k}+1}}{2\mathrm{k}+1}$, $|x| \le 1$;

10) $tg\, \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\infty} \frac{1}{(2k-1)^2 - x}$.

9. ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ DO WHILE

Часто встречаются алгоритмы, в которых число повторений заранее неизвестно, но известно, что их больше 0, а выход из цикла происходит при выполнении некоторого условия.

Рассмотрим, например, итерационный метод Ньютона извлечения квадратного корня, т.е. вычисление $y = \sqrt{x}$ с заданной точностью ε > 0. Запись этого алгоритма в виде формул имеет вид

$$y_0 = \frac{x}{2}; \quad y_{i+1} = \frac{1}{2}(y_i + \frac{x}{y_i}); \quad i = 0, 1, 2, \ldots$$

Вычисления прекращаются при выполнении условия $|y_{i+1} - y_i| \leq \varepsilon$.

Этот метод заключается в следующем. Выбирается некоторое начальное приближение y_0, а затем находятся последующие приближения по рекуррентной формуле, причем очередное значение y_{i+1} — поправка (уточнение) к предыдущему приближению y_i. Известно, что этот процесс сходится при любом начальном приближении.

В приведенном алгоритме индекс i показывает лишь переход от одного приближения к другому. Будет ошибкой попытка использования переменной с индексом, т.е. элемента массива, так как количество элементов этого массива необходимо задать, а оно нам заранее неизвестно. Процесс носит циклический характер: после задания начального приближения, в качестве которого можно взять, например, *x/2*, дальнейшие операции сводятся к многократному вычислению очередного приближения к искомому результату.

Этот процесс заканчивается, когда два очередных приближения достаточно близки.

В общем случае число повторений этого цикла заранее неизвестно, а его определение равносильно решению поставленной задачи. В подобных случаях формулируем лишь условие, при выполнении которого этот циклический процесс должен завершиться.

Реализовать данный циклический алгоритм можно с использованием оператора цикла с постусловием, синтаксис которого имеет вид

```
do
{
  <операторы>;
}
while (<условие>);
```

Семантика оператора цикла с постусловием определена следующим образом: выполняется последовательность операторов, затем проверяется условие. Если условие принимает значение `true` (отлично от нуля), то выполнение операторов повторяется. Процесс завершается, когда после очередного выполнения операторов логическое условие примет значение `False` (равно `0`). Важно, чтобы рано или поздно условие не выполнилось.

Пример 1. Вычислить $y = \sqrt{x}$ с точностью ε, заданной в виде константы. Для контроля произвести сравнение с результатом вычисления с использованием стандартной процедуры *Sqrt(x)*.

```
/* Вычисление корня из x с использованием
итерационного метода Ньютона с заданной
точностью Eps*/

#include  <iostream>
using namespace std;
int main(){
 float x, a, b, delta;
 float Eps = 1.0E-5; //точность вычислений
 cout << "x="; cin >> x;
 b = x / 2; // начальное приближение
 do {
     a = b;
     b = (a + x / a) / 2;
     cout << "a=" << a << " b=" << b <<endl;
 }
 while (abs(b - a) >= Eps);
```

```
 cout << "sqrt(" << x << ")=" << b << endl;
 delta = abs(b - sqrt(x)); /*сравнение
         полученного значения E со значением
               стандартной функции sqrt(x)*/
 cout << "delta=" << delta << endl;
 return 0;
}
```

Пример 2. Найти количество цифр и сумму цифр в заданном числе N.

```
#include <iostream>
using namespace std;
int main(){
 int N, k = 0, s = 0;
 cout << "N="; cin >> N;
 do {
  s += N % 10;  // Суммирование цифр
  ++k;           // Увеличение счетчика цифр
  N /= 10; //отбрасываем младшую цифру числа
 }
 while (N > 0);
 cout << "k=" << k << endl << "s=" << s;
 return 0;
}
```

Выполнение повторяющихся действий с заранее известным положительным количеством итераций можно выполнить по схеме

```
int i=<начальное значение>; /* инициализация
                               счетчика i*/
do {
  <операторы>;
  i+=<шаг изменения>;
}
while (i<= <конечное значение>);
```

Если шаг изменения счетчика равен 1, то имеем:

```
int i=<начальное значение>;
do {
  <операторы>;
}
while (++i <= <конечное значение>);
```

Пример 3. Дана последовательность из N (N > 0) целых чисел. Найти наибольшее число.

```
#include  <iostream>
using namespace std;
void main() {
 int N, max, i=1;
 cin>>N;
 do{
  int x;
  cin>>x;
  if (i==1 || x>max) max=x;
 }
 while ( ++i <= N);
 cout<<max;
}
```

Пример 4. Для заданного значения N и y вычислить $\sum_{k=1}^{N} \frac{(-1)^k(2+k!)}{y^{2k}}$.

```
#include  <iostream>
using namespace std;
void main() {
 float s=0, p, t;
 p=2; // p=2/y^2k, здесь k=0
 t=1; // t=k! / y^2k, здесь k=0
 int n, k=1, f=1;
 cin>>n>>y;
 do{
  f=-f;
  p/=y*y;
  t/=k/y*y;
  s+=f*(p+t);
```

```
 }
 while (++k <= n);
 cout<<s;
}
```

Пример 5. Для заданного y вычислить $\sum_{k=1}^{\infty} \frac{(-1)^k(2+k!)}{y^{2k}}$ с заданной точностью eps. При вычислении суммы ряда считается, что нужная точность достигнута, если модуль очередного слагаемого меньше значения точности eps.

```
#include <iostream>
using namespace std;
void main() {
 int n, k=1, p=1, t=1, f, y;
 float s=0, eps = 1.0E-5;;
 cin>>n>>y;
 do {
  f=-f;
  p/=y*y;
  t/=k/y*y;
  s+=f*(p+t);
  ++k;
 }
 while ((p+t)>eps);
 cout<<s;
}
```

Контрольные вопросы и задания

1. Дано целое число. Найти количество его нечетных цифр.
2. Дано целое число. Найти сумму его четных цифр.
3. Дано целое число. Верно ли, что в числе нет двоек и троек?
4. Дано целое число. Верно ли, что в числе все цифры одинаковые?
5. Дано целое число. Найти количество заданной цифры в числе.

6. Дано целое число. Найти наименьшую, делящуюся на 3 цифру. Если такой нет, вывести «Нет».

7. Десятичный N-значный (N — четное) номер является «счастливым», если сумма первых (N % 2) цифр этого номера равна сумме последующих его (N% 2) цифр. Является ли заданный номер N «счастливым»?

8. Даны натуральное число N и цифры X, Y. Верно ли, что N содержит X и заканчивается цифрой Y?

9. Определить количество и сумму цифр, больших K заданного натурального числа N.

10. Дано натуральное число N. Получить число M, записанное цифрами числа N в обратном порядке. Например, при N = 125, M = 521.

11. Перевести заданное натуральное число N из десятичной системы счисления в пятеричную систему счисления.

12. Перевести заданное натуральное число N из двоичной системы счисления в десятичную систему счисления.

13. Дано вещественное число R — цена 1 кг конфет. Вывести стоимость 1, 2, … , 10 кг конфет.

14. Для заданного натурального значения переменной M вычислить y = M!. По определению, N! = 1·2·3...·(N-1)·N.

15. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

16. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи.

17. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

18. Для заданного x найти сумму первых N чисел ряда:

1) $\cos x = \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{(2k)!}$;

2) $e^x = \sum_{k=0}^{N} \frac{x^k}{k!}$;

3) $sh\ x\ = \sum_{k=0}^{N} \frac{x^{2k+1}}{(2k+1)!}$;

4) $ln\ (1+x) = \sum_{k=1}^{\mathrm{N}} (-1)^{k-1} \frac{x^k}{k}\ , -1 < x \leq 1$;

5) $ln\ \frac{x+1}{x-1} = 2\sum_{i=0}^{\mathrm{N}} \frac{1}{(2i+1)x^{2i+1}}, |x| > 1$;

6) $e^{-x^2}\ = \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{k!}$;

7) $(1+x)^m\ = \sum_{k=0}^{N} \frac{m!\ x^k}{(m-k)!\ (k)!}\ , -1 < x < 1$;

8) $arcsin\ x\ = \sum_{k=0}^{\mathrm{N}} \frac{(2\mathrm{k})!\ x^{2k+1}}{2^{2k}\ (\mathrm{k}!)^2(2\mathrm{k}+1)}, |x| \leq 1$;

9) $\mathrm{arctg}\ x\ = \sum_{k=0}^{\mathrm{N}} \frac{(-1)^{\mathrm{k}} x^{2\mathrm{k}+1}}{2\mathrm{k}+1}, |x| \leq 1$;

10) $tg\ \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\mathrm{N}} \frac{1}{(2k-1)^2 - x}$.

19. Для заданного x найти сумму ряда с точностью eps:

1) $\sin x\ = \sum_{k=0}^{\infty} (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!}$;

2) $\mathrm{ch\ x}\ = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$;

3) $ln\ x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(x-1)^k}{k},\ x \geq 1/2$;

4) $ln\ (1\ -\ \mathrm{x}) = -\sum_{k=1}^{\infty} \frac{\mathrm{x}^k}{k}, -1 < \mathrm{x} \leq 1$;

5) $ln\ \frac{x+1}{x-1} = 2\sum_{i=0}^{\infty} \frac{1}{(2i+1)x^{2i+1}}, |x| > 1$;

6) $\mathrm{e}^{-\mathrm{x}^2}\ = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{\mathrm{k}!}$;

7) $(1+\mathrm{x})^{\mathrm{m}}\ = \sum_{k=0}^{\infty} \frac{\mathrm{m}!\ x^k}{(\mathrm{m}-\mathrm{k})!\ (k)!}\ , -1 < \mathrm{x} < 1$;

8) $arcsin\ x\ = \sum_{k=0}^{\infty} \frac{(2\mathrm{k})!\ x^{2k+1}}{2^{2k}\ (\mathrm{k}!)^2(2\mathrm{k}+1)}, |x| \leq 1$;

9) $\operatorname{arctg} x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{2k+1}, |x| \leq 1;$

10) $tg \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\infty} \frac{1}{(2k-1)^2 - x}.$

10. ЦИКЛ С ПРЕДУСЛОВИЕМ FOR

Цикл For в языке C++ является универсальным циклом с предусловием, хотя обычно используется для создания циклов, которые должны выполняться заданное число раз.

Оператор For цикла с предусловием имеет следующий синтаксис:

```
for(<Инициализация>;<Условие>;<Модификация>)
<Оператор>;
```

Выражение <Инициализация> выполняется только один раз в самом начале перед любой другой частью инструкции `for`. Затем управление передается <Условие>. Часто используется для инициализации индексов цикла. Может содержать выражения или объявления.

<Условие> проверяется перед выполнением каждой итерации <Оператор>, включая первую итерацию. <Условие> – выражение, значение которого относится к целочисленному типу или типу класса, для которого имеется однозначное преобразование к целочисленному типу. Обычно используется для проверки критериев завершения цикла for. Если <Условие> выполняется, т.е. <Условие> имеет значение *true* (целочисленное значение, не равное нулю), то управление передается <Оператор>, называемое телом цикла. Если тело цикла состоит из более чем одного оператора, то необходимо использовать операторные скобки ({…}).

<Модификация> выполняется после выполнения тела цикла <Оператор>. Обычно используется для приращения индексов цикла. После выполнения <Модификация> управление передается на этап проверки <Условие>.

Допускается неуказание любого из элементов цикла For. В частности, если не указать выражения <Инициализация> и <Модификация>, то получим цикл For, полностью соответствующий циклу while:

```
for ( ;<Условие>; ) <Оператор>;
```

Бесконечный цикл, не выполняющий никаких действий, может быть представлен в виде

```
for ( ; ; ) ;
```

Пример 1. Дано натуральное число. Найти сумму его цифр.

```
#include <iostream>
using namespace std;
int main() {
 setlocale (LC_ALL, "Russian");
 int a, s=0; //в s будет накапливаться сумма
 cout<<"Введите натуральное число";
 for (cin>>a; a ; a/=10)s+=a%10;
 cout<<s; //вывод результата
 return 0;
}
```

В разделе <Инициализация> выполняется ввод числа. <Условие> определяет выполнение цикла, пока число a не равно 0 (возможен вариант a!=0 или a>0). В теле цикла переменная s увеличивается на значение последней цифры числа a. <Модификация> удаляет последнюю цифру числа a (возможно указание a=a/10).

Следующий код решает ту же задачу, но с другим порядком действий, предшествующих итерации цикла:

```
#include <iostream>
using namespace std;
int main() {
 setlocale (LC_ALL, "Russian");
 int a, s;
 cout<<"Введите натуральное число";
```

```
 cin>>a;
 for (s=0; a ; a/=10)s+=a%10;
  cout<<s;
 return 0;
}
```

<Инициализация> и <Модификация> могут состоять из более чем одного оператора. В этом случае операторы перечисляются через запятую и выполняются в порядке следования слева неправо. Для решаемого примера допустим следующий программный код (хотя он менее читаемый):

```
#include <iostream>
using namespace std;
int main(){
 setlocale(LC_ALL, "Russian");
 int a, s;
 cout << "Введите натуральное число";
 for (s=0, cin>>a; a; s+=a%10, a/=10);
 cout << s;
return 0;
}
```

Пример 2. Дано целое число. Найти сумму его четных цифр.

```
#include <iostream>
using namespace std;
int main(){
 setlocale (LC_ALL, "Russian");
 int a,s;
 cout<<"Введите  число";
 cin>>a;
 for (s=0;a; a/=10)
  if (!(a&1))  s+= a % 10;
 cout<<s;
return 0;
}
```

Программа отличается от примера 1 только телом цикла, а именно условным оператором. Выражение a&1 выполняет

поразрядную конъюнкцию двоичных кодов своих операндов, а следовательно, будет равно 1 (true), если a нечетное и 0 (false) – иначе.

В цикле for допустимо использование команд break и continue.

Команда continue прекращает выполнение текущей итерации цикла и управление передается непосредственно <Модификация>.

Решение задачи примера 2 с использованием continue имеет вид

```
#include <iostream>
using namespace std;
int main() {
 setlocale (LC_ALL, "Russian");
 int a,s;
 cout<<"Введите  число";
 cin>>a;
 for (s=0;a; a/=10){
  if (a&1) continue;
  s+= a % 10;
 }
 cout<<s;
 return 0;
}
```

Команда break прекращает выполнение цикла и управление передается команде, следующей за оператором For. Использование команды break рассмотрим на примере следующей задачи:

Пример 3. Дано целое число. Определить, есть ли в его десятичной записи цифры 5 или 7.

```
#include <iostream>
using namespace std;
int main(){
 setlocale (LC_ALL, "Russian");
 int a;
```

```
 bool f; // f –флаг, 0–нет цифр, 1 – есть
 cout<<"Введите число";
 cin>>a;
 for (f=0; a; a/=10)
  f= f || (a%10==5) || (a%10==7);
 if (f) cout<<"есть";
 else cout<<"нет";
return 0;
}
```

В данном примере в цикле просматриваются все цифры числа `a` (пока `a` не ноль) и если после цикла переменная `f` равна true, то в числе `a` были искомые цифры, иначе – нет.

Нет необходимости рассматривать все цифры числа, а достаточно найти только одну из них, равную 3 или 5. Пример соответствующего программного кода:

```
#include <iostream>
using namespace std;
int main(){
 setlocale (LC_ALL, "Russian");
 int a;
 cout<<"Введите  число";
 for (cin>>a; a; a/=10)
  if (a%10==5 || a%10==7) break;
 if (a) cout<<"есть";
 else cout<<"нет";
 return 0;
}
```

От числа `a` отбрасываются последние цифры до тех пор, пока либо `a` не станет раной нулю, либо последней цифрой станет `3` или `5`. В последнем случае выполнение цикла прекратится, а число `a` будет отлично от нуля, что и используется как критерий ответа в условном операторе.

Рассмотрим следующий тип задач обработки элементов последовательности с маркером окончания. При этом

необходимо до цикла считать первый элемент последовательности.

Пример 4. Дана последовательность целых чисел, оканчивающаяся нулем. Найти сумму элементов последовательности, не делящихся на 5.

```
#include <iostream>
using namespace std;
int main() {
 int s=0, a;
 for (cin >> a; a; cin >> a)
  if (a%5) s+=a;
cout<<s;
return 0;
}
```

Обработка двух и более соседних элементов последовательности, ограниченной маркером, продемонстрирована на следующих примерах.

Пример 5. Дана последовательность целых чисел, оканчивающаяся нулем. Найти сумму положительных чисел, после которых следует отрицательное число.

```
#include  <iostream>
using namespace std;
int main() {
 int  s=0, a, pred;
 cin>>pred;
 if (pred){
  for (cin>>a; a; cin>>a){
   if (a < 0 && pred > 0)  s+=pred;
   pred=a;
  }
 }
 cout<<s;
return 0;
}
```

Пример 6. Дана последовательность целых чисел, оканчивающаяся -1. Верно ли, что последовательность является упорядоченной по возрастанию.

```
#include <iostream>
using namespace std;
int main() {
 int pred, a;
 cin>>pred;
 bool f = true;
 if (pred!=-1){
  for (cin>>a; a!= -1; pred=a, cin>>a)
   if (pred > a) f=false;
 }
 cout<<f;
 return 0;
}
```

Предложенное решение можно оптимизировать с учетом того, что потоки ввода и вывода в C++ разделяются. Другими словами, ввод можно прекратить, если найдено нарушение, выдать соответствующее сообщение и закончить выполнение программы:

```
#include  <iostream>
using namespace std;
int main() {
 setlocale (LC_ALL, "Russian");
 int  pred, a;
 cin>>pred;
 if (pred!=-1){
 for (cin>>a; a!= -1; pred=a, cin>>a)
  if (pred > a) {cout<<"неверно"; return 0;}
 }
 cout<<"верно";
 return 0;
}
```

Пример 7. Дана последовательность целых чисел, оканчивающаяся –100. Найти среднее арифметическое чисел, оканчивающихся на 5, перед которыми идет четное число. Если таких чисел нет, сообщить об этом.

```
#include <iostream>
using namespace std;
int main() {
 setlocale (LC_ALL, "Russian");
 int  s=0, k=0, a, pred;
 pred= 1; /*гарантируется, что для первого
числа последовательности условие задачи не
выполняется */
 for (cin>>a; a!= -100; pred=a, cin>>a)
  if (pred&1==0 && a%10==5) {s+=a; k++;}
 if (!k) cout<<"Таких чисел нет";
 else cout<< float(s)/k;
 return 0;
}
```

Цикл `for` в основном применяется в случаях использования различных счетчиков или перебора значений. Такие циклы еще называются циклами с параметром.

Пример 8. Найти сумму всех четных чисел от `a` до `b`.

```
#include <iostream>
using namespace std;
int main() {
 int a, b;
 cin>>a>>b;
 if (a&1)//если a нечетно, то увеличить на 1
  ++a;
 int s=0;
 for (int i=a; i<=b; i+=2) s+=i;
 cout<< s;
 return 0;
}
```

Обратите внимание, что переменная `i` существует только в цикле `For` и принимает значения всех последовательных четных чисел на отрезке от `a` до `b`.

Пример 9. Найти количество всех делителей числа.

```
#include <iostream>
using namespace std;
int main() {
int n, k=0;
cin>>n;
for (int i=1; i<=n; ++i) /* i перебирает все
                            числа от 1 до n*/
  k+=(n%i==0); /* к k прибавляется 1 если
                  условие выполняется */
cout<< k;
return 0;
}
```

Пример 10. Дано `n` чисел. Найти наибольшее из них.

```
#include <iostream>
using namespace std;
int main() {
int n, max;
cin>>n; //вводим количество
for (int i=0; i<n; ++i){ /* организуем цикл
                         выполняющийся n раз*/
 int x;
 cin>>x; // вводим очередное число
 if (i==0 || x>max) max=x;
}
cout<< max;
return 0;
}
```

Пример 11. Дано `n` целых чисел. Найти количество тех, у которых сумма цифр четна.

```
#include <iostream>
using namespace std;
int main() {
 int n, kol=0;
 cin>>n;
 for (int i=0; i<n; ++i){
  int x;
  cin>>x;
 /* для поиска суммы цифр используем цикл с
постусловием */
  int s=0;
  do{
   s+=x%10;
   x/=10;
  }
  while (x);
  kol+=((s&1)==0); /* kol увеличится на 1
                      если s четно*/
 }
 cout<<kol;
 return 0;
}
```

Пример 12. Дано n целых чисел. Найти сумму тех, у которых в десятичной записи присутствует цифра 7.

```
#include <iostream>
using namespace std;
int main() {
 int n, sum=0;
 cin>>n;
 for (int i=0; i<n; ++i){
  int x;
  cin>>x;
 /* для поиска суммы цифры 7 используем цикл
                           for */
  for (int a=abs(x); a; a/=10)
    if (a%10==7) { sum+=x; break;}
```

```
 }
 cout<<sum;
 return 0;
}
```

Для поиска цифры 7 в очередном числе используется цикл `for` с переменной `a`. Начальное значение a определяется как модуль x, если этого не сделать, то для отрицательных целых чисел выражение `a%10` будет отрицательно (например (–17)%10 равно (–7)). Как только цифра 7 в числе найдена, сумма изменяется и внутренний цикл прерывается.

Пример 13. Найти сумму первых n членов ряда

$$\frac{sin\ \ x}{x} = \sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i+1)!}, |x| < \infty;$$

```
#include <iostream>
using namespace std;
int main() {
 int n;
 double x,s=1; // сумма для i=0
 double elem=1; // первое слагаемое
 cin>>n>>x;
 for (int i=1; i<=n; ++i){ /* i перебирает
                          все числа от 1 до n*/
  elem=-elem*x*x/(2*i)/(2*i+1); /* вычисляем
                        очередное слагаемое */
  s+=elem;
  }
 cout>>s;
 return 0;
}
```

Пример 14. Найти с заданной точностью eps сумму ряда

$$\frac{sin\ \ x}{x} = \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i}}{(2i+1)!}, |x| < \infty;$$

```
#include <iostream>
using namespace std;
int main() {
 int n;
 double x,eps,s=1, elem=1;
 cin>>n>>x>>eps;
 for (int i=1; abs(elem)>=eps; ++i){ /* i
перебирает все числа пока не достигнута
точность*/
  elem=-elem*x*x/(2*i)/(2*i+1); /* вычисляем
очередное слагаемое */
  s+=elem;
 }
 cout>>s;
 return 0;
}
```

Контрольные вопросы и задания

1. Дано целое число. Найти количество его нечетных цифр.
2. Дано целое число. Найти сумму его четных цифр.
3. Дано целое число. Верно ли, что в числе нет двоек и троек?
4. Дано целое число. Верно ли, что в числе все цифры одинаковые?
5. Дано целое число. Найти количество заданной цифры в числе.
6. Дано целое число. Найти наименьшую, делящуюся на 3 цифру. Если такой нет, вывести «Нет».
7. Десятичный N-значный (N — четное) номер является «счастливым», если сумма первых (N % 2) цифр этого номера равна сумме последующих его (N% 2) цифр. Является ли заданный номер N «счастливым»?
8. Даны натуральное число N и цифры X, Y. Верно ли, что N содержит X и заканчивается цифрой Y?
9. Определить количество и сумму цифр, больших K заданного натурального числа N.

10. Дано натуральное число N. Получить число M, записанное цифрами числа N в обратном порядке. Например, при N = 125, M = 521.

11. Перевести заданное натуральное число N из десятичной системы счисления в пятеричную систему счисления.

12. Перевести заданное натуральное число N из двоичной системы счисления в десятичную систему счисления.

13. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти произведение последних цифр положительных чисел.

14. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму квадратов двузначных отрицательных чисел. Если таких чисел нет, сообщить об этом.

15. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти наименьшее четное число.

16. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, у которых последняя цифра больше предпоследней. Если таких чисел нет, сообщить об этом.

17. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму квадратов чисел, оканчивающихся на 2 или на 13.

18. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, принадлежащих диапазону [– 20; 35]. Если таких чисел нет, сообщить об этом.

19. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, у которых последняя и предпоследняя цифры равны. Если таких чисел нет, сообщить об этом.

20. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти среднее арифметическое положительных чисел, квадрат которых оканчивается на 1, 6 или 9.

21. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму положительных двузначных чисел, за которыми идет оканчивающееся на 11 число.

22. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество троек соседних элементов, где каждое следующее число больше предыдущего.

23. Дана последовательность целых чисел, оканчивающаяся числом 0. Верно ли, что последовательность является знакочередующейся.

24. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти количество чисел, не делящихся на 12, за которыми идет отрицательное число.

25. Дана последовательность целых чисел, оканчивающаяся числом 0. Найти сумму чисел, после которых идет число, не оканчивающееся на 2 и 3.

26. Для заданного натурального числа N найти количество его нечетных делителей.

27. Для заданного натурального числа N найти сумму делителей, заканчивающихся заданной цифрой K.

28. Является ли заданное натуральное число N>1 простым?

29. Натуральное число N называется совершенным, если оно равно сумме всех своих делителей, исключая само это число (например: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14). Определить, является ли заданное натуральное K совершенным?

30. Натуральное число N называется совершенным, если оно равно сумме всех своих делителей, исключая само это число (например: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14). Найти все совершенные числа отрезка [A, B].

31. Натуральные числа M и N называются дружественными, если сумма собственных делителей одного из них равна другому и наоборот (например, 220 и 284). Являются ли заданные натуральные числа M и N дружественными?

32. Найти все дружественные числа отрезка [A, B].

33. Разложить натуральное число N на простые множители, указав кратность каждого из них.

34. Найти наибольший общий делитель двух заданных натуральных чисел M и N по алгоритму Евклида.

35. Найти наименьшее общее кратное двух заданных натуральных чисел.

36. Дано вещественное число R — цена 1 кг конфет. Вывести стоимость 1, 2, … , 10 кг конфет.

37. Для заданного натурального значения переменной M вычислить y = M!. По определению, N! = 1·2·3...·(N-1)·N.

38. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

39. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи.

40. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекурентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для $n \geq 3$. Найти сумму первых N чисел Фибоначчи. Определить количество чисел Фибоначчи, кратных пяти из диапазона [A, B].

41. Для заданного x найти сумму первых N чисел ряда:

1) $\cos x = \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{(2k)!}$;

2) $e^x = \sum_{k=0}^{N} \frac{x^k}{k!}$;

3) $sh\ x = \sum_{k=0}^{N} \frac{x^{2k+1}}{(2k+1)!}$;

4) $ln\ (1+x) = \sum_{k=1}^{\mathrm{N}} (-1)^{k-1} \frac{x^k}{k}$, $-1 < x \leq 1$;

5) $ln\ \frac{x+1}{x-1} = 2 \sum_{i=0}^{\mathrm{N}} \frac{1}{(2i+1)x^{2i+1}}$, $|x| > 1$;

6) $e^{-x^2} = \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{k!}$;

7) $(1+x)^m = \sum_{k=0}^{N} \frac{m!\, x^k}{(m-k)!\,(k)!}$, $-1 < x < 1$;

8) $arcsin\ x = \sum_{k=0}^{\mathrm{N}} \frac{(2\mathrm{k})!\, x^{2k+1}}{2^{2k}\,(\mathrm{k}!)^2(2\mathrm{k}+1)}$, $|x| \leq 1$;

9) $\mathrm{arctg}\, x = \sum_{k=0}^{\mathrm{N}} \frac{(-1)^{\mathrm{k}} x^{2\mathrm{k}+1}}{2\mathrm{k}+1}$, $|x| \leq 1$;

10) $tg\ \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\mathrm{N}} \frac{1}{(2k-1)^2 - x}$.

42. Для заданного x найти сумму ряда с точностью eps:

1) $\sin x = \sum_{k=0}^{\infty} (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!}$;

2) $\text{ch x} = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$;

3) $ln\ x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(x-1)^k}{k},\ x \geq 1/2$;

4) $ln\ (1 - \text{x}) = -\sum_{k=1}^{\infty} \frac{\text{x}^k}{k}, -1 < \text{x} \leq 1$;

5) $ln\ \frac{x+1}{x-1} = 2\sum_{i=0}^{\infty} \frac{1}{(2i+1)x^{2i+1}}, |x| > 1$;

6) $\text{e}^{-\text{x}^2} = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{\text{k!}}$;

7) $(1 + \text{x})^{\text{m}} = \sum_{k=0}^{\infty} \frac{\text{m!}\, x^k}{(\text{m}-\text{k})!\ (k)!}, -1 < \text{x} < 1$;

8) $arcsin\ x = \sum_{k=0}^{\infty} \frac{(2\text{k})!\, x^{2k+1}}{2^{2k}\ (\text{k!})^2 (2\text{k}+1)}, |x| \leq 1$;

9) $\text{arctg}\ x = \sum_{k=0}^{\infty} \frac{(-1)^{\text{k}} x^{2\text{k}+1}}{2\text{k}+1}, |x| \leq 1$;

10) $tg\ \frac{\pi}{2} x = \frac{4x}{\pi} \sum_{k=1}^{\infty} \frac{1}{(2k-1)^2 - x}$.

11. СТАТИЧЕСКИЕ МАССИВЫ В С++

Для работы с множеством однотипных данных (строками, датами, целочисленными значениями и т.п.) удобно использовать массивы. Например, можно создать строковый массив для хранения списка работников организации или целочисленный массив для хранения порядковых номеров работников. Существует много определений массива, но одно из них может быть таким. Массив – это множество однотипных данных, имеющих одинаковое имя, но различные индексы.

Массивы могут иметь как одно, так и более одного измерений. В зависимости от количества измерений массивы делятся на одномерные массивы, двумерные массивы, трёхмерные массивы и т. д. до n-мерного массива, но чаще всего на практике используют одномерные и двумерные массивы. Стоит заметить, что в оперативной памяти компьютера все элементы хранятся в виде одномерной последовательности (не

обязательно подряд), поэтому массивы любой размерности можно представить как одномерные.

Статические массивы – это такие массивы, размер которых задаётся программистом в программе при помощи констант и не изменяется до конца работы программы. Размер статического массива известен компилятору до выполнения программы на основе исходного кода программы, поэтому пользователь не может в процессе работы программы изменить размер массива. Один из главных недостатков использования статических массивов – неэффективное использование оперативной памяти компьютера, одно из главных достоинств – простота понимания кода для его обработки.

Любой массив характеризуется следующими основными понятиями:

1) *элемент массива* (значение элемента массива) – значение, хранящееся в определенной ячейке памяти, расположенной в пределах массива. Каждый элемент массива характеризуется тремя величинами:

– адресом элемента – адресом начальной ячейки памяти, в которой расположен этот элемент;

– индексом элемента (порядковым номером элемента в массиве);

– значением элемента;

2) *адрес массива* – адрес начального элемента массива;

3) *имя массива* – идентификатор, используемый для обращения к элементам массива;

4) *размер массива* – количество элементов массива;

5) *размер элемента* – количество байт, занимаемых одним элементом массива;

6) *длина массива* – количество байт, отводимое в памяти для хранения всех элементов массива. Длина массива равна произведению размера элемента на количество элементов массива.

Объявление и инициализация одномерного массива. Для объявления массива в языке С++ используется следующий синтаксис:

```
<тип> <имя_массива> [размерность]
= {инициализация};
```

При описании массив можно инициализировать, т. е. присвоить его элементам начальные значения. *Инициализация* представляет собой набор начальных значений элементов массива, указанных в фигурных скобках и разделённых запятыми:

```
//массив a из 10 целых чисел
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Если инициализирующих значений меньше, чем элементов в массиве, остаток массива обнуляется, если больше – лишние значения не используются.

Поскольку все инструкции по выделению памяти статического массива формирует компилятор до выполнения программы, то размерность массива может быть задана только константой или константным выражением, например:

```
/*одномерный массив целочисленного типа из 5
элементов*/
int array[5];

/*одномерный массив вещественного типа из N
элементов (N – константа)*/
const int N = 20;
float x[N];

/одномерный массив беззнакового
целочисленного типа из 2*M элементов*/
const int M = 5;
unsigned int  b[2*M];
```

Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности. Автоматический контроль выхода индекса за границы массива не производится, поэтому программист должен следить за этим самостоятельно, например:

```
int a[3] = {1, 2, 3}; /* a[0]=1, a[1]=2,
                         a[2]= 3*/

/*недостающие значения инициализации
заполняются нулями*/
int b[5] = {1, 2, 3}; /*b[0] = 1, b[1] = 2,
              b[2] = 3, b[3] = 0, b[4] = 0*/

//лишнее значение (5) не используется

int c[4] = {1, 2, 3, 4, 5}; /*c[0] = 1,
              c[1] = 2, c[2] = 3, c[3] = 4*/

/*индекс может быть опущен и размер
вычисляется по количеству элементов
инициализации */
int d[] = {0, 2, 4}; /*c[0] = 0, c[1] = 2,
                       c[2] = 4 (размер 3)*/

/*строка так же является массивом символов,
оканчивающимся нуль-символом (управляющей
последовательностью \0) */
char s[] = "Hi!"; /*s[0]='H', s[1]='i',
            s[2]='!', s[3]='\0' (размер 4)*/
```

Для доступа к элементу массива после его имени указывается номер элемента (индекс) в квадратных скобках:

```
имя_массива [индекс];
```

Элемент массива считается переменной: он может получать значения (например, в операторе присваивания), а также участвовать в выражениях.

Например, для объявленных выше массивов обращение и использование будет выглядеть следующим образом:

```
cout << a[1];
int i = 2;
cout << b[i-1];
cout << c[2*i];
int z;
```

```
a[2] = -1;
z = a[0] + (b[i] + d[i+1]) / c[2*i+1];
d[0] = (z - d[i]) / c[2*i+1];
```

Для перебора элементов массива удобно использовать цикл `for` согласно следующей схеме:

```
for (int i=0; i< <размер массива>; ++i)
 <обработка i-го элементамассива>;
```

Например, следующий код позволяет ввести с клавиатуры элементы массива `a` размерности `N`:

```
for (int i=0; i<N; ++i)
 cin >> a[i];
```

Пример 1. Найти сумму элементов массива. Вывести найденные значения и сам массив на экран. Элементы массива ввести с клавиатуры.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 const int N = 5;   // размер массива  a[N]
 int mas[N]; /* одномерный статический
                массив из 5 элементов*/
 //введём элементы массива с клавиатуры
 for (int i = 0; i < N; i++) {
     cout << "Введите " << i << "-й элемент
массива: ";
     cin >> mas[i];
 }
 int sum = 0; // для суммы элементов
 for (int i = 0; i < N; i++)
     sum += mas[i]; //считаем сумму
 for (int i = 0; i < N; i++)
     cout << " " << mas[i];//вывод массива
 cout << endl << "Сумма элементов: " << sum;
```

Часто возникают ситуации, когда необходимо определить размера массива, для этого используется следующее выражение:

```
sizeof(array)/sizeof(array[0]);
```

Здесь `array` – имя массива, `N` – полученный размер массива, функция `sizeof(array)` – возвращает размер всего массива в байтах, а `sizeof(array[0])` – размер первого элемента в байтах (естественно совпадающий с размером остальных элементов). Таким образом, поделив размер всего массива на размер одного элемента, мы получим количество элементов, т.е. искомый размер массива.

Пример 2. Найти значение минимального элемента одномерного массива.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 //объявим массив с инициализацией
 int b[]={1, -5, 9, 7, 4, -7, 3, 0, -8, 6 };
 //определим размер массива
 int n = sizeof(b) / sizeof(b[0]);
 int min = b[0];/*переменная min для
хранения значения минимального элемента,
инициализуется значением первого элемента*/
 for (int i = 1; i < n; i++)
  if (b[i] < min)  min = b[i];
 cout << endl << "Минимальный элемент: " <<
min;
}
```

В примере 2 необходимо найти безусловный минимум, поэтому переменная `min` инициализировалась первым элементом массива. В случае, если необходимо найти значение элемента массива, удовлетворяющего некоторому условию, используют либо переменную – маркер инициализации, либо в качестве начального указывают некорректное значение, которое гарантированно заменится.

Пример 3. Найти значение минимального, кратного 5, элемента одномерного массива.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 const int n = 10;
 int a[n];
 for (int i = 0; i < n; i++) cin>> a[i];
 int min;
 bool flag=false; // min не инициализирован
 for (int i = 0; i < n; i++)
  if (a[i]%5==0 && (!flag || min>a[i])) {
   min=a[i];
   flag=true; // min имеет значение
  }
 if (flag) cout<<min;
 else cout<<"нет элементов кратных 5";
}
```

В данном решении переменная `flag` указывает на наличие у переменной `min` значения (инициализации). Соответственно условие `(a[i]%5==0 && (!flag || min>a[i]))` выполняется, когда либо найден первый элемент, кратный 5, либо очередной подходящий элемент меньше `min`.

Следующий вариант решения инициализирует min заведомо недопустимым значением, что одновременно является признаком инициализации.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 const int n = 10;
 int a[n];
 for (int i = 0; i < n; i++) cin>> a[i];
 int min=1; //не кратно 5
```

```
 for (int i = 0; i < n; i++)
  if (a[i]%5==0 && (min==1 || min>a[i]))
   min=a[i];
 if (min!=1) cout<<min;
 else cout<<"нет элементов кратных 5";
}
```

Пример 4. Дан массив целых чисел mas[3], элементы которого создаются при помощи случайной генерации N[-9; 9]. Проверить массив на упорядоченность по возрастанию.

```
#include <iostream>
#include <ctime>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 int sum = 0, up = 0;
 const int N = 10;
 int mas[N];
 srand(time(0)); /*инициализация случайных
                   значений*/
 cout << "Исходный массив: " << endl;
 for (int i = 0; i < N; i++) {
    mas[i] = rand() % 19 - 9; /*Заполняем
массив случайными числами из [-9; 9] */
    cout << mas[i] << ' ';//Вывод на экран
 }
 for (int i = 0; i < N - 1; i++)
  if (mas[i] < mas[i + 1]) up++;
 if (up == N - 1)
  cout << "\nМассив упорядочен\n";
 else cout << "\nМассив НЕ упорядочен!\n";
}
```

Обратите внимание, что в теле цикла в качестве индекса массива используется выражение `i + 1`, поэтому в условии выполнения цикла указано `i < N - 1`, чтобы не выйти за границы массива.

В данном решении определяется количество up элементов массива, больших чем предыдущие, и если таковых будет N−1 (без первого), то ответ положительный, иначе – отрицательный.

Задачи на свойства массивов целесообразно решать до первого нарушения и не проверять до конца, если массив уже не удовлетворяет проверяемому признаку.

Пример 5. Дан массив целых чисел A[10], элементы которого вводятся с клавиатуры. Проверить массив на симметричность. Симметричный массив – это массив, у которого первый элемент равен последнему, второй – предпоследнему и т.д.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Rus");
 bool simm = 1;//пока не нашли нарушения
 const int N = 10;
 int A[N];
 for (int i = 0; i < N; i++)
  cin >> A[i];
 for (int i = 0; i < N / 2; i++)
  if (A[i] != A[N - i - 1]){
   simm = 0; // нашли нарушение
   break;
  }
 if (simm) cout << "\nМассив симметричен !";
 else cout << "\nМассив НЕ симметричен !";
}
```

В решении используется переменная simm, значение которой указывает, симметричен массив (равен 1) или было найдено нарушение симметричности (равно 0). При первом найденном нарушении значение simm «сбрасывается» в ноль и выполнение цикла прекращается. Обычно такое решение применяется, когда результат проверки будет использоваться в программе где-то в дальнейшем.

Если результат нужно только вывести и прекратить работу программы, то допустим следующий код (здесь используется команда `return` для завершения программы):

```
#include <iostream>
using namespace std;
int main(){
 setlocale(LC_ALL, "Russian");
 const int N = 10;
 int A[N];
 for (int i = 0; i < N; i++)
  cin >> A[i];
 for (int i = 0; i < N / 2; i++)
  if (A[i] != A[N - i - 1]){
   cout << "\nМассив НЕ симметричен !";
   return 0;
  }
 cout << "\nМассив симметричен !";
 return 0;
 }
```

Если в процессе работы цикла нарушений симметричности обнаружено не было, то оператор вывода положительного ответа достижим.

Пример 6. Дан массив целых чисел mas[20], элементы которого принадлежат интервалу [0; 3] и создаются при помощи случайной генерации. Посчитать количество элементов, перед которыми и после которых стоят нули.

```
#include <iostream>
#include <ctime>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 int kol = 0;
 const int N = 20;
 int mas[N];
 srand(time(0));
 cout << "Исходный массив: " << endl;
```

```
 for (int i = 0; i < N; i++) {
  mas[i] = rand() % 4;
  cout << mas[i] << ' ';
 }
 for (int i = 1; i < N-1; i++)
  if (mas[i - 1] == 0 && mas[i + 1] == 0)
   kol++;
 cout << "\nКоличество = " << kol << endl;
}
```

Напомним результат булевского выражения – `true` в языке C++ равно целочисленной единице, а `false` – нулю. Используя данный факт, в указанной программе корректно заменить

```
if (mas[i - 1] == 0 && mas[i + 1] == 0)
   kol++;
```

на

```
kol+=(mas[i - 1] == 0 && mas[i + 1] == 0);
```

Пример 7. Дан массив целых чисел mas[20]. Найти сумму положительных двузначных чисел, в записи которых хотя бы одна цифра 3. Вывести сумму и количество таких чисел.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, " Russian ");
 int sum = 0, kol = 0;
 const int N = 20;
 int mas[N];
 for (int i = 0; i < N; ++i)
  cin>> mas[i];
 for (int i = 0; i < N; i++)
  if (mas[i] > 9 && mas[i] < 100 &&
    (mas[i] / 10 == 3 || mas[i] % 10 == 3)){
   sum += mas[i];
   kol++;
  }
```

```
 cout << "\nСумма = " << sum << endl;
 cout << "\nКоличество: " << kol << endl;
}
```

Пример 8. Дан массив целых чисел mas[20]. Найти количество различных элементов массива.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, " Russian ");
 int kol = 0;
 const int N = 20;
 int mas[N];
 for (int i = 0; i < N; ++i)
  cin>> mas[i];
 for (int i = 0; i < N; ++i){
  int flag=1;
  for (int j = 0; j < i; ++j)
   if (mas[i]==mas[j]) {flag=0; break;}
  kol+=flag;
  }
 cout << "\Различных элементов: " << kol;
}
```

В массиве просматриваются все элементы массива (цикл с параметром `i`). Для каждого элемента `mas[i]` определяется значение переменной-флага `flag`. Если `flag` равен `1`, то значение, равное `mas[i]`, еще не встречалось, в противном случае `flag` равен `0`. Для определения значения `flag` используется цикл с параметром `j`, который перебирает индексы элементов, предшествующих `i`-му, и сравнивает их значения со значением `mas[i]`. Если значения совпадают, то флаг обнуляется и выполнение внутреннего цикла прекращается.

Контрольные вопросы и задания

1. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Вывести массив на экран в обратном порядке.

2. Одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов заполнить случайными числами. Вычислить сумму нечетных элементов массива.

3. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Вычислить среднеарифметическое положительных элементов массива.

4. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Вывести отрицательные элементы на экран в обратном порядке.

5. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти максимальный отрицательный элемент.

6. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти минимальный положительный элемент.

7. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Вычислить сумму положительных элементов массива, кратных 3.

8. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Вывести на экран сначала его четные элементы, затем нечетные.

9. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Все элементы, кратные числу 17, заменить нулем.

10. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Все нечетные элементы удвоить, а четные уменьшить вдвое.

11. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить количество четных элементов, индексы которых не кратны 3.

12. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить сумму положительных элементов с четными индексами.

13. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Вывести значения, которые не превосходят среднего арифметического.

14. Дан массив целых чисел. Определить количество положительных элементов с четными порядковыми номерами.

15. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить сумму элементов, больших заданного числа x, порядковые номера которых не кратны 5.

16. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Заменить все кратные пяти элементы значением наибольшего элемента.

17. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Увеличить все нечетные элементы на значение среднего арифметического положительных элементов.

18. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных чисел. Проверить, упорядочен ли массив по возрастанию модулей своих элементов

19. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Заменить все нечетные элементы, до и после которых следуют положительные на значение среднего арифметического элементов массива.

20. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Поменять в этой последовательности местами наибольший и наименьший элементы.

21. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов, и число x. Заменить все элементы с индексами, кратными 3, квадратом числа x.

22. Дан одномерный массив, состоящий из N (0 < N < 100). Если он является симметричным, то увеличить его элементы вдвое, в противном случае все отрицательные элементы заменить модулем числа.

23. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Переставить первый элемент массива

на место последнего. При этом второй, третий, ..., последний элементы сдвинуть влево на 1 позицию.

24. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Все элементы, индексы которых являются простыми числами, заменить нулем.

25. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти количество чисел с не кратной 3 суммой цифр.

26. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти количество элементов, сумма цифр которых четна, а индексы нечетны.

27. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти количество простых элементов, до и после которых располагаются нечетные числа.

28. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти сумму простых чисел, индекс которых не кратен 3.

29. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Найти среднее арифметическое простых чисел, индексы которых кратны 3.

30. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Заменить положительные числа, индексы которых четны, значением наибольшего по модулю числа.

31. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить, есть ли в нем хотя бы одно простое число.

32. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить, есть ли в нем два совершенных элемента.

33. Дан одномерный массив, состоящий из N (0 < N < 100) целочисленных элементов. Определить, есть ли в нем хотя бы одно число Фибоначчи.

34. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Если в результате замены отрицательных элементов их квадратами элементы будут образовывать неубывающую последовательность, то получить

сумму элементов исходной последовательности, в противном случае получить их произведение.

35. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Определить количество максимальных элементов в массиве.

36. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Определить количество минимальных элементов в массиве.

37. Дан одномерный массив, состоящий из N (0 < N < 100) вещественных элементов. Поменять местами первый отрицательный и последний положительный элементы массива. Учесть возможность того, что отрицательных или положительных элементов в массиве может не быть.

12. ДВУМЕРНЫЕ МАССИВЫ

Двумерный массив в математике соответствует матрице, поэтому его принято рассматривать как совокупность строк и столбцов, на пересечении которых находятся ее элементы. Количество строк и столбцов задает размер матрицы.

Синтаксис описания матрицы задается структурным типом вида

```
<тип элементов> <имя> [m][n];
```

где m – количество строк матрицы; n – количество столбцов матрицы.

В соответствии с этим описанием во внутренней памяти ЭВМ выделяется область из *m*×*n групп* последовательных ячеек, в которые при работе программы записываются двоичные значения элементов матрицы. Например, описание

```
float A [3][5];
```

объявляет матрицу из трех строк и пяти столбцов. В памяти компьютера для элементов матрицы выделяется область, состоящая из 3 × 5 = 15 последовательных ячеек вещественного типа, или 15 × 4 = 60 байт, так как вещественное число типа `float` занимает в памяти ЭВМ 4 байта.

Обращение к элементам матрицы осуществляется так же, как и элементам одномерного массива, с помощью операторов индексирования [ ] и индексированной переменной целого типа (идентификатора матрицы). Так же, как и одномерный массив (вектор), элементы матрицы располагаются в памяти ЭВМ последовательно, поэтому ее можно рассматривать как вектор с числом элементов $m \times n$. Однако в отличие от вектора для адресации элементов матрицы используется пара индексов, причем индексы, как и при адресации элементов одномерного массива, начинаются с нуля. Примеры адресации элементов матрицы:

```
A[i][j] // элемент a_{i,j};
A[2][3] // элемент a_{2,3};
A[2*n][k+1] // элемент a_{2n,k+1}.
```

Для поэлементного ввода и вывода матрицы используется двойной параметрический цикл. Если задать индекс *i* как параметр внешнего цикла, а индекс *j* – как параметр внутреннего цикла, то ввод – вывод матрицы осуществляется построчно.

Рассмотрим пример организации ввода целочисленной матрицы M по строкам:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Описание матрицы вместе с текущими индексами имеет вид

```
int main()
int M[2][3] ;
int i, j;
```

Ввод в программе реализуется в форме диалога, т.е. сопровождается поясняющим сообщением:

```
cout<<"Введите матрицу M\n";
for(i = 0; i<2; i++)
 for(j = 0; j< 3; j++)
  cin>>M[i][j];
```

На клавиатуре желательно для наглядности восприятия набирать элементы матрицы по строкам, отделяя числа друг от друга одним или несколькими пробелами:

```
1 2 3 [Enter]
4 5 6 [Enter]
```

Хотя и такой вариант ввода приведет к аналогичному результату:

```
1 2 3 4 5 6 [Enter]
```

В этих примерах после каждого нажатия клавиши [Enter] из потока консольного ввода значения будут присваиваться соответствующим элементам матрицы.

Вывод матрицы, как правило, реализуется в удобном для чтения виде, т.е. в прямоугольном виде. С этой целью в тело внешнего цикла, помимо внутреннего цикла, включается оператор вывода литерала `"\n"` или константы `endl` для перевода курсора к началу следующей строки экрана:

```
for ( i = 0; i<2; i++) {
 for ( j = 0; j< 3; j++)
  cout<<M[i][j]<<" ";
 cout<<endl;
}
```

Вид матрицы на экране будет таким:

```
1 2 3
4 5 6
```

Для статических массивов для указания размерности целесообразно использовать константы. Для приведенной матрицы описание индексами имеет вид

```
const int n = 2;
const int m = 3;
int M[n][m];
```

Базовыми алгоритмами обработки матриц являются те же алгоритмы, которые используются при обработке векторов.

Однако реализацию этих алгоритмов можно условно рассматривать для двух типов задач.

Задача первого типа. Алгоритмы реализуются при просмотре всех элементов матрицы (просмотр может быть с условием). Начальная установка алгоритма выполняется перед двойным циклом. В этом случае запись операторов цикла для параметров i и j осуществляется последовательно друг за другом и имеет вид

```
<начальная установка искомых параметров>
 for (int i = 0; i<n; i++)
  for (int j = 0; j< m; j++)
   <тело цикла>;
```

Пример 1. Дана матрица вещественных чисел $A = \{a_{ij}\}_{2x4}$. Вычислить значение $Z = P1/|P2|$, где P1 и P2 – произведения положительных и отрицательных элементов матрицы соответственно.

```
#include<iostream>
using namespace std;
void main() {
 setlocale(LC_ALL, "Russian");
 const int n = 2;
 const int m = 4;
 float A[n][m] ;     // описание матрицы A
 cout<<"Введите матрицу A:\n";
 for (int i =0; i<n; i++)
  for (int j = 0;j<m; j++)
    cin>>A[i][j];
 float P1=1, P2=1; //инициализация
 for ( i = 0; i<n; i++) //перебор строк
  for ( j = 0; j<m ;j++){ //перебор столбцов
   if (A[i][j]>0) P1*= A[i][j];
   if (A[i][j]<0) P2*= A[i][j];
  }
 float Z=P1/fabs(P2);
```

```
 cout<<"\nZ="<<Z;
}
```

Задача второго типа. Алгоритмы реализуются при обработке отдельной строки или отдельного столбца матрицы. В этом случае начальная установка алгоритма выполняется между операторами цикла, записанными для индексов строк и столбца (`i` и `j`). Например, если алгоритм реализуется для каждой строки, то запись в программе имеет следующий вид:

```
for (i = 0; i< n ; i++) {
 <предустановка искомых параметров>
 for (j = 0; j< m; j++)
  <тело цикла>;
 <последействие>
}
```

Пример 2. В матрице $X = \{x_{ij}\}_{3x6}$ целых чисел первый элемент каждой строки поменять местами с минимальным элементом этой строки. Вывести матрицу `X` после обмена. Для заполнения матрицы использовать генератор случайных чисел.

```
#include<ctime>
#include<iostream>
using namespace std;
void  main() {
 setlocale(LC_ALL, "Russian");
 const int n = 3;
 const int m = 6;
 int X[n][m];
 int a=-20, b=20; /* Границы интервала
                     случайных чисел*/
 srand(time(0)); /* инициализация генератора
                     случайных чисел*/
 for(int i=0; i<n; i++)
  for (int j= 0; j<m; j++)
   X[i][j]= a+rand()%(b-a+1); // заполнение
 cout<<"Исходная матрица X\n";
 for(int i=0; i<n; i++) {
```

```
 for (int j=0; j<m; j++)
  cout<<X[i][j]<<" ";
 cout<<endl;
}
for(int i=0; i<n; i++) { //цикл по  строкам
 int jmin=0; /*индекс минимального элемента
           изначально равен индексу первого
           элемента строки*/
 for (j=0; j<m; j++) // цикл по  столбцам
  if (X[i][j]< X[i][jmin]) jmin=j;
 int a= X[i][jmin]; //обмен значений
 X[i][jmin]=X[i][0];
 X[i][0]=a;
}
cout<<"\n Преобразованная матрица\n";

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

Здесь во внешнем цикле выполняется предустановка вспомогательной переменной jmin, служащей для хранения индекса минимального элемента.

Пример 3. Дана матрица вещественных чисел $C = \{c_{ij}\}_{2x4}$. Вычислить среднее арифметическое каждого столбца. Результат записать в одномерный массив $S = \{s_j\}$; $j = \overline{1,4}$.

```
#include <iostream>
using namespace std;
void main() {
 setlocale(LC_ALL, "Russian");
 const int n = 2;
 const int m = 4;
 float  C[n][m], float S[m];
 cout<<"Введите матрицу C:\n";
```

```
 for(int i=0; i<n;  i++)
  for (int j= 0; j<m;  j++)
   cin>>C[i][j];
 for (int j= 0; j<m;  j++) {
  S[j]=0; /*начальная установка элемента
                 массива для сумм*/
  for(int i=0; i<n; i++)
   S[j]+= C[i][j]; /*накопление суммы j-го
                      столбца */
  S[j]=S[j]/n;  /*вычисление среднего
                  значения суммы j столбца */
 }
 cout<<endl<<"Результатный вектор:\n";
 for (int j= 0; j<m;  j++) cout<<S[j]<<" ";
}
```

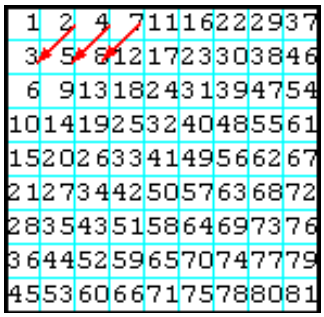
В приведенной программе для вычисления каждого элемента `S[j]` организован двойной цикл, в котором индекс `j` является параметром внешнего цикла, а индекс `i` – внутреннего. Во внешнем цикле выполняется предустановка значения `j`-го элемента одномерного массива `S` в нулевое значение для накопления суммы элементов столбца. В качестве последействия выполняется вычисление среднего арифметического на основе накопленной во внутреннем цикле суммы.

Далее представлен вариант программы без использования одномерного массива. Вычисленные средние выводятся непосредственно на экран:

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 const int n = 2;
 const int m = 4;
 float  C[n][m];
 cout<<"Введите матрицу C:\n";
 for(int i=0; i<n; i++)
```

```
  for (int j= 0; j<m;   j++)
   cin>>C[i][j];
 for (int j= 0; j<m;   j++) {
  float S=0;
  for(int i=0; i<n;   i++)
   S+= C[i][j];
  S /= n;
  cout<<"Среднее арифметическое "<<j<<"-го
столбца = "<<S<<endl;
 }
}
```

Пример 4. Записать в матрицу M = $\{m_{ij}\}_{9x9}$ линейную последовательность натуральных чисел (1, 2, 3, ...) от левого верхнего угла по диагонали: влево – вниз. Результатную матрицу вывести на экран.



При решении подобной задачи следует определить правило изменения значений индексов очередного элемента матрицы. Например, при анализе данной задачи можно определить, что индексы изменяются так, как показано в таблице:

i	1	**1**	**2**	1	2	3	**1**	**2**	**3**	**4**	…
j	1	**2**	**1**	3	2	1	**4**	**3**	**2**	**1**	…

Обобщая эти изменения, можно заметить, что в начале (на и выше побочной диагонали) каждая новая «линия» заполнения начинается в ячейке с индексами [1, j] и заканчивается в ячейке [j, 1], j = 1..9, а затем (ниже побочной диагонали) начинается в ячейке с индексами [i, 9] и заканчивается в ячейке [9, i], i = 2..9. С учетом того, что индексация в C++ начинается с нуля, имеем:

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

```
void main() {
 setlocale(LC_ALL, "Russian");
 const int n = 9;
 int mas[n][n];
 int num = 0;
// заполняем над побочной диагональю
 for (int j = 0; j < n; ++j)
     for (int i = 0; i <= j; ++i)
         mas[i][j - i] = ++num;
// заполняем под побочной диагональю
 for (int i = 1; i < n; i++)
     for (int j = n - 1; j >= i; --j)
         mas[i+(n-1-j)][j] = ++num;
// выводим ответ
 cout << "Сформированный массив: \n";
 for (int i = 0; i < n; i++) {
     for (int j = 0; j < n; j++)
         cout << mas[i][j] << " ";
     cout << endl;
 }
}
```

Пример 5. Дан двумерный массив целых чисел размерности `n x m` (0 < `n`, `m` < 100). Определить минимальный номер строки, состоящей только из положительных элементов.

В случае, когда размерность матрицы заранее не известна, но известны границы размерности, память резервируется для максимальной размерности, а используется в программе только часть.

Для искомого номера строки определим переменную `Ni`, начальное значение определим больше максимально возможного номера (что одновременно позволит определить, существует ли такая строка)

```
#include <iostream>
using namespace std;
void main() {
 setlocale(LC_ALL, "Russian");
```

```
 int mas[100][100];
 int n, m;
 cin >> n >> m; /*ввели количество строк и
                  столбцов*/
 for (int i= 0; i < n; ++i)
  for (int j = 0; j < m; ++j)
   cin>>mas[i][j ];
 int Ni = 101;
 for (int i = 0; i < n; ++i){
  bool flag = 1;
  for (int j = 0; j <m; ++j)
   if (mas[i][j] <= 0) { flag = 0; break; }
  if (flag) { //строка найдена
   Ni = i; break;
  }
 }
 if (Ni < n) cout << Ni;
 else cout << "строка не найдена";
}
```

Можно заметить, что если строка найдена, то цикл с параметром `i` досрочно прекращает свою работу (`i<n`). В текущей программе переменная `i` перестает существовать при выходе из цикла (так как описана внутри). Если описание переменной произвести до цикла, тогда после выполнения цикла ее значение позволит определить режим его завершения:

```
#include <iostream>
using namespace std;
void main() {
 setlocale(LC_ALL, "Russian");
 int mas[100][100];
 int n, m;
 cin >> n >> m; /*ввод количества строк и
                  столбцов*/
 for (int i= 0; i < n; ++i)
  for (int j = 0; j < m; ++j)
   cin>>mas[i][j ];
```

```
 int i;
 for (i = 0; i < n; ++i){
  bool flag = 1;
  for (int j = 0; j <m; ++j)
   if (mas[i][j] <= 0) { flag = 0; break; }
  if (flag) break;
 }
 if (i < n) cout << i;
 else cout << "строка не найдена";
}
```

Аналогично можно использовать параметр j внутреннего цикла:

```
#include <iostream>
using namespace std;
void main() {
 setlocale(LC_ALL, "Russian");
 int mas[100][100];
 int n, m;
 cin >> n >> m;
 for (int i= 0; i < n; ++i)
  for (int j = 0; j < m; ++j)
   cin>>mas[i][j ];
 int i;
 for (i = 0; i < n; ++i){
  int j;
  for (j = 0; j <m; ++j)
   if (mas[i][j] <= 0) break;
  if (j<m) break;
 }
 if (i < n) cout << i;
 else cout << "строка не найдена";
}
```

Контрольные вопросы и задания

1. Дана матрица B(11, 11) вещественных чисел. Найти количество отрицательных элементов матрицы.

2. Дана матрица M(10, 5) вещественных чисел. Найти минимальный по абсолютной величине элемент матрицы

3. Дана матрица A(10, 8) вещественных чисел. Найти сумму произведений элементов строки и столбца матрицы, на пересечении которых находится наибольший по абсолютной величине элемент матрицы.

4. Дана матрица X(5, 10) вещественных чисел. Найти количество элементов матрицы, превышающих ее среднее арифметическое значение.

5. Дана матрица Y(10, 12) вещественных чисел. Найти сумму элементов матрицы, для которых i + j = k, где k – введенное с клавиатуры значение. Проверить, что значение k позволяет найти решение для каждой из матриц.

6. Дана матрица Z(10, 7) вещественных чисел. Найти сумму положительных элементов столбца, содержащего максимальный элемент матрицы

7. Дана матрица B(6, 11) вещественных чисел. Найти разность между максимальным и минимальным значениями элементов массива.

8. Дана матрица C(8, 12) вещественных чисел. Найти сумму элементов, расположенных по периметру матрицы.

9. Дана матрица X(9, 12) вещественных чисел. Найти произведение ненулевых элементов строки матрицы, на которой расположен максимальный элемент.

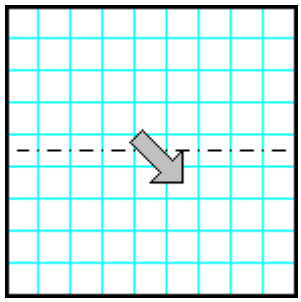
10. Дана матрица T(5, 14) вещественных чисел. Найти разность между суммой значений элементов четных и нечетных строк.

11. Дана матрица U(6, 15) вещественных чисел. Найти сумму элементов столбца, в котором расположен максимальный элемент матрицы.

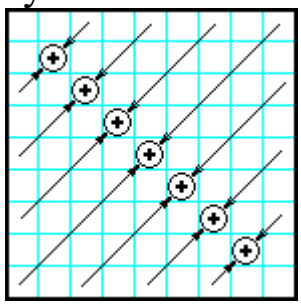
12. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Найти отношение сумм элементов, лежащих выше и ниже главной диагонали матрицы.

13. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Найти произведение ненулевых элементов, лежащих выше главной диагонали матрицы

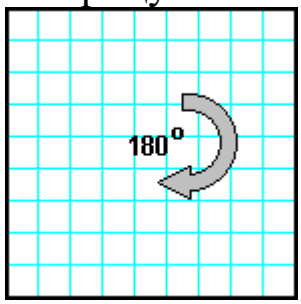
14. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Отобразить верхнюю половину матрицы на нижнюю зеркально симметрично относительно горизонтальной оси.



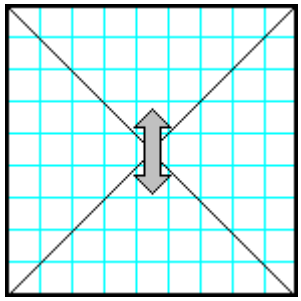
15. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Найти сумму элементов



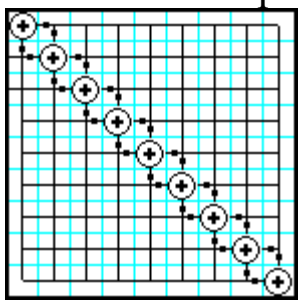
16. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Развернуть матрицу на 180º.



17. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Отобразить симметрично относительно горизонтальной оси секторы матрицы, которые лежат выше и ниже главной и побочной диагоналей.



18. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). На главной диагонали разместить суммы элементов, которые лежат на той же строке и том же столбце.



19. Дана матрица целых чисел размером N × M (3 < N, M < 100). Найти количество максимальных элементов в массиве

20. Дана квадратная матрица целых чисел размером N × N (3 < N < 100). Найти среднее арифметическое четных элементов.

21. Сформировать и вывести на экран матрицу целых чисел размером N × M (3 < N, M < 100), инициализировав его элементы случайными числами из диапазона 0…99. В каждой его строке найти максимальный элемент.

22. Сформировать и вывести на экран матрицу целых чисел размером N × M (3 < N, M < 100), инициализировав его элементы случайными числами из диапазона 0…99. Найти столбец с минимальной суммой элементов. Если таких столбцов несколько, должен быть найден номер самого правого из них.

23. Сформировать и вывести на экран матрицу целых чисел размером N × M (3 < N, M < 100), инициализировав его элементы случайными числами из диапазона 0…99. Для каждой строки выяснить, имеются ли в ней элементы, последняя цифра которых равна Z.

24. Сформировать и вывести на экран матрицу целых чисел размером N × M (3 < N, M < 100), инициализировав его элементы случайными числами из диапазона 0…99. Определить

минимальный номер столбца, состоящего только из нечетных элементов.

25. Сформировать и вывести на экран матрицу целых чисел размером N × N (3 < N < 100), инициализировав его элементы случайными числами из диапазона – 99…99. Если сумма элементов главной диагонали больше суммы элементов побочной, то обнулить элементы каждой четной строки, в противном случае – каждой нечетной.

13. УКАЗАТЕЛИ

Память компьютера можно представить в виде последовательности пронумерованных байтов, с которыми можно работать по отдельности или блоками. Когда компилятор обрабатывает оператор определения переменной, например

```
int i = 10;
```

он выделяет память в соответствии с типом (int) и инициализирует ее указанным значением (10). Все обращения в программе к переменной по ее имени (i) заменяются компилятором на адрес области памяти, в которой хранится значение переменной.

Каждый объект (переменная или константа) связан с адресом памяти – номером первого байта блока для размещения ее значения. Существуют правила выбора адреса начала блока для объекта определенного типа. Так, адрес блока для объектов целого типа, как правило, должен быть четным, для значения типа float – кратным четырем и т.д.

Указатель – это тоже объект, который размещается в памяти. В случае 32-битной версии компилятора указатель занимает 4 байта в оперативной памяти, а его значением является адрес некоторой области памяти. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом.

В C++ различают три вида указателей – указатели на объект, на void и на функцию, отличающуюся свойствами и набором допустимых операций.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного).

Объявление указателя на объект в простейшем случае (в дальнейшем называемого просто указателем) имеет вид

```
<тип> *<имя>;
```

где `тип` может быть любым, кроме ссылки и битового поля. Например,

```
int *z;   char *s;
```

Здесь `z` и `s` – указатели разного типа, так как они ссылаются на данные разного типа, в силу чего значением `s` может быть любой адрес байта (данные типа char занимают блок в 1 байт), а значением `z` – четный адрес байта (если данные типа `int` занимают 2 байта).

Размер указателя зависит от модели памяти. Можно определить указатель на указатель и т.д.

К моменту объявления указателя `тип` может быть только объявлен, но еще не определен (следовательно, в структуре, например, может присутствовать указатель на структуру того же типа). Например,

```
struct node
{ int info;
node *next;}
```

Для работы с указателями чаще всего используют следующие операции:

– операция косвенной адресации (разыменовывание) (*) определяет значение объекта, которое содержится в области памяти, адрес которого является значением указателя;

– операция взятия адреса (&). Ее результатом является адрес объекта.

Следующий код иллюстрирует применение указанных операций:

```
char c;   // переменная символьного типа
char *p; /* указатель на переменную
                символьного типа*/
cin<< c;  //ввод значения переменной c
p = &c; // p = адрес c
cout << c << " " << *p; /* выводится
значение переменной c и значение переменной,
адрес которой находится в указателе p, т.е.
два одинаковых числа */
```

Пример. Рассмотрим программу, которая позволяет определить значение целочисленной переменной, адрес блока памяти, выделенного под эту переменную, а также значение указателя, ссылающегося на эту переменную, и адрес памяти для указателя. Адреса байтов памяти будем, как принято, выводить в шестнадцатеричном формате:

```
#include <iostream>
using namespace std;
int main() {
  int a, *b;
  a = 134;
  b = &a;
  cout << "\n value"  << a <<  " = " << hex
<< a << "hex.";
  cout << "\n adress" << dec << a << " = "
<< hex << &a << "hex.";
  cout << "\n value for pointer   b =" <<
dec << *b << "= " << hex << *b << "hex.";
  cout << "\n value of pointer  b =" << hex
<< b << "hex.";
  cout << "\n address of pointer b =" << hex
<<  &b << "hex.";
  return 0;
}
```

Результат работы программы:

```
value 134=0x86hex.
address 134=0x22fecchex.
value for pointer b= 134=86hex.
value of pointer b=0x22fecchex.
address of pointer b=0x22fec8hex.
```

Указатель может быть константой или переменной, а также указывать на константу или переменную, например:

```
int n;   // целая переменная
const int cn = 7;     // целая константа
int *pt_n;  // указатель на целую переменную
const int *pt_c; /*указатель на целую
                      константу */
int * const cp = &n; /* указатель-константа
                      на целую переменную*/
const int * const cpc = &cn  /* указатель-
             константа на целую константу*/
```

Можно определить указатель на указатель и т.д. Так, при объявлении

```
<тип> **<имя>;
```

переменная `<имя>` создается для хранения адреса другой переменной, которая предназначена для хранения адреса третьей переменной типа `<тип>`.

В следующем примере `p2` содержит адрес `p1`, `p1` содержит адрес `p`, в `p` находится значение 7:

```
int p=7;
int *p1=&p;
int **p2=&p1;
```

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

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

```
void *ptr; // Указатель на void
int i; // Целая переменная
int *ptri; // указатель на int
ptr= &i; // Допустимо
ptr= ptri; // Допустимо
ptri=ptr; // Недопустимо!!!
```

Над указателем неопределенного типа нельзя выполнять и операцию разыменования без явного приведения типа

```
ptri = (int *)ptr ;
```

или с помощью определения нового типа

```
deftype int * pt;
ptri = pt(ptr);
```

Ссылка позволяет определять альтернативное имя переменной (псевдоним). Ссылка не создает копии объекта, а является лишь другим именем объекта. Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.

Синтаксис объявления ссылки:

```
<тип> &<Name2> = <Name1>;
```

Такое объявление назначает переменной с именем Name1 второе имя Name2.

Ссылка при объявлении всегда должна быть проинициализирована.

```
int a,b;
int &alt=a;  // ссылка на a
alt = b;  // равносильно a=b;
alt++;  // равносильно a++;
```

Если объявлен указатель

```
int *ptr = &a;
```

то истинны следующие выражения:

```
*ptr == alt;  // true, сравниваются значения
ptr == &alt;  // true, сравниваются адреса
```

Ссылку можно рассматривать как постоянный указатель, который всегда разыменован, для него не надо выполнять операцию косвенной адресации *.

Возможно инициализировать ссылку на константу:

```
const char &new_line='\n';
```

В этом случае компилятор создает некоторую временную переменную и ссылку на нее. Следующий код иллюстрирует данный процесс:

```
char temp = '\n';
const char &new_line = temp;
```

Указатели чаще всего используют при работе с динамической памятью или кучей (от англ. heap). Это свободная память, в которой можно во время выполнения программы выделять место в соответствии с потребностями. Доступ к выделенным участкам динамической памяти, называемым динамическими переменными, производится только через указатели. Время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти. В C++ используется два способа работы с динамической памятью. Первый использует семейство функций `malloc` и достался в наследство от языка C, второй использует операции `new` и `delete`.

При определении указателя надо стремиться выполнить его инициализацию, т. е. присвоение начального значения. Непреднамеренное использование неинициализированных указателей – распространенный источник ошибок в программах. Инициализатор записывается после имени указателя либо в круглых скобках, либо после знака равенства.

Существуют следующие способы инициализации указателя:

1. Присваивание указателю адреса существующего объекта:

– с помощью операции получения адреса:

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

– с помощью значения другого инициализированного указателя:

```
int* r = p;
```

– с помощью имени массива или функции, которые трактуются как адрес:

```
int b[10]; // массив
int *t = b; /* присваивание адреса начала
                  массива*/
void f(int a){<тело>} // определение функции
void (*pf) (int);   // указатель на функцию
pf = f;       // присваивание адреса функции
```

2. Присваивание указателю адреса области памяти в явном виде:

```
Char *cp = (char *) 0xB8000000;
```

Здесь `0xB8000000` — шестнадцатеричная константа, `(char *)` – операция приведения типа: константа преобразуется к типу «указатель на `char`».

3. Присваивание пустого значения:

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

В первой строке используется константа `NULL`, определенная в некоторых заголовочных файлах C как указатель, равный нулю. Рекомендуется использовать просто `0`, так как это значение типа `int` будет правильно преобразовано стандартными способами в соответствии с контекстом.

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

– с помощью операции new:

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

– с помощью функции malloc:

```
int* u = (int *)malloc(sizeof(int)); // 4
```

В операторе 1 операция new выполняет выделение достаточного для размещения величины типа int участка динамической памяти и записывает адрес начала этого участка в переменную n. Память под саму переменную n (размера, достаточного для размещения указателя) выделяется на этапе компиляции. В операторе 2, кроме описанных выше действий, производится инициализация выделенной динамической памяти значением 10.

В операторе 3 операция new выполняет выделение памяти под 10 величин типа int (массива из 10 элементов) и записывает адрес начала этого участка в переменную q, которая может трактоваться как имя массива. Через имя можно обращаться к любому элементу массива.

В операторе 4 делается то же самое, что и в операторе 1, но с помощью функции выделения памяти malloc, унаследованной из библиотеки C. В функцию передается один параметр – количество выделяемой памяти в байтах.

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

Освобождение памяти, выделенной с помощью операции new, должно выполняться с помощью delete, а для выделенной с помощь malloc – с помощью free.

Для описанных переменных освобождение памяти будет выглядеть так:

```
delete n;    // 1
delete m;    // 2
delete[] q; // 3
free(u);     // 4
```

При выделении памяти с помощью `new[]` необходимо применять `delete[]`. Размерность массива при этом не указывается.

С указателями можно выполнять следующие операции: разыменовывание (или косвенное обращение к объекту) (`*`), присваивание, сложение с константой, вычитание, инкремент (`++`), декремент (`--`), сравнение, приведение типов.

При работе с указателями часто используется операция получения адреса `&`.

Операция разыменования предназначена для доступа к величине, адрес которой хранится в указателе. Эту операцию можно использовать как для получения, так и для изменения значения величины (если она не объявлена как константа):

```
char a;   // переменная типа char
char *p = new char; /* описание динамической
                переменной и выделение памяти*/
*p = 'Ю'; // присваивание значения
a = *p; /* копирование содержимого ячейки
           памяти на которую указывает p
           в переменную a*/
```

Как видно из примера, конструкцию `*имя` указателя можно использовать в левой части оператора присваивания. С ней допустимы все действия, определенные для величин соответствующего типа (если указатель инициализирован). На одну и ту же область памяти может ссылаться несколько указателей различного типа.

Арифметические операции с указателями (сложение с константой, вычитание, инкремент и декремент) автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных,

последовательно размещенными в памяти, например, с массивами.

Инкремент перемещает указатель к следующему элементу массива, декремент – к предыдущему. Фактически значение указателя изменяется на величину `sizeof(тип)`. Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа, например:

```
short *p = new short [5];
p++; // значение p увеличивается на 2
long *q = new long [5];
q++; // значение q увеличивается на 4
```

Разность двух указателей – это разность их значений, деленная на размер типа в байтах (в применении к массивам разность указателей, например на третий и шестой элементы, равна 3). Суммирование двух указателей не допускается.

При записи выражений с указателями следует обращать внимание на приоритеты операций. В качестве примера рассмотрим последовательность действий, заданную в операторе.

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

```
int a = 5;     // переменная a
int *p = &a; /* в указатель записывается
                адрес a
```

Контрольные вопросы и задания

Используя вместо самой переменной указатель на нее, написать программу в соответствии с заданием:

1. Вычислить площадь треугольника по стороне и высоте.
2. Вычислить площадь окружности по заданному радиусу.

3. Даны значения a и b, найти их среднее арифметическое, среднегеометрическое.

4. Вычислить $y = tg(x)+5x^3 – 4x^2$.

5. Вычислить площадь квадрата.

6. Вычислить высоту треугольника, зная две стороны треугольника и угол между ними.

7. Вычислить $y = |x-\cos(x)|$.

8. Ввести сторону квадрата a. Вычислить площадь вписанной окружности.

9. Задается длина окружности. Найти площадь круга, ограниченного этой окружностью.

10. Вычислить углы треугольника, зная его стороны.

11. Вычислить площадь трапеции.

12. Вычислить $y = \cos|x^3-x^2|$.

13. Вычислить длину гипотенузы прямоугольного треугольника, зная длины двух катетов.

14. Вычислить корень квадратный от $(|x^5|-x^4+|x^3|)$.

15. Вычислить корень квадратный от $(\sin(x)+\cos(x))$.

16. Вычислить объем цилиндра, зная радиус основания и высоту.

17. Вычислить объем конуса.

18. Определить время, через которое встретятся два тела, равноускоренно движущиеся друг к другу. Известны: v1 и v2 – начальные скорости, a1 и a2 – ускорения, s – расстояние между ними.

19. Вычислить сторону треугольника, зная две другие стороны и угол между ними.

20. Вычислить площадь ромба, зная длину стороны и угол.

14. ДИНАМИЧЕСКИЕ МАССИВЫ

Доступ к произвольному элементу массива обеспечивается по имени массива и индексу – целочисленному смещению от начала:

```
имя_массива [ индекс ]
```

Имя массива является указателем-константой. К нему приемлемы все правила адресной арифметики, связанные с указателями. Для любого массива соблюдается равенство:

```
имя_массива == &имя_массива ==
                                &имя_массива[]
```

Доступ к элементам массива возможен по его имени как указателю. Если описан массив

```
int z[3];
```

справедливы равенства:

```
z[0] == *(z) == *(z + 0)
z[1] == *(z + 1)
z[2] == *(z + 2)
z[i] == *(z + i)
```

Так как имя массива есть не просто указатель, а указатель константа, то значение имени невозможно изменить. Получить доступ ко второму элементу массива `int z[4]` с помощью выражения `*(++z)` будет ошибкой, а выражение `*(++z)` допустимо.

Пример 1. Вывести на экран элементы массива, используя имя массива как указатель.

```
#include <iostream>
using namespace std;
void main(){
 int x[ ]= {1, 2, 3, 4, 5, 0};
 int i = 0;
 int size = sizeof(x)/sizeof(x[0]);
 for (int i=0; i < size; ++i)
  cout << endl << *(x + i);//аналогично x[i]

 }
```

Обращение к элементу массива относят к постфиксному выражению вида `PE[IE]`. `PE` – указатель на нужный тип, `IE` – целочисленный тип, `PE[IE]` – индексированный элемент этого

массива. Аналогично при использовании адресной арифметики `*(PE + IE)`. Поскольку сложение коммутативно, то запись `*(PE + IE)` эквивалентна `*(IE + PE)`, а следовательно, и `IE[PE]` именует тот же элемент массива, что и `PE[IE]`. То есть для приведенного примера будут справедливы равенства:

```
*(x + i) == *(i + x) == i[x] == x[i]
```

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе следует использовать динамические массивы. Память под них выделяется с помощью операции `new` или функции `malloc` в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

```
int n = 10:
int *a = new int[n];
```

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции `new`. Выделяется столько памяти, сколько необходимо для хранения n величин типа `int`. Величина n может быть переменной.

Обнуления памяти при ее выделении не происходит. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу статического – например `a[3]`. Можно обратиться к элементу массива и другим способом – `*(a + 3)`. В этом случае мы явно задаем те же действия, что выполняются при обращении к элементу массива обычным образом. Рассмотрим их подробнее. В переменной-указателе *a* хранится адрес начала массива. Для получения адреса третьего элемента к этому адресу прибавляется смещение 3. Операция сложения с константой для указателей учитывает размер адресуемых элементов, т. е. на самом деле индекс умножается на длину элемента массива: `a + 3 * sizeof(int)`. Затем с помощью операции *(разыменовывание) выполняется выборка значения из указанной области памяти.

Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, необходимо освободить ее с помощью операции `delete[]`. Квадратные скобки в операции `delete []` при освобождении памяти из-под массива обязательны. Их отсутствие может привести к неопределенному поведению программы.

Таким образом, время жизни динамического массива, как и любой динамической переменной, – с момента выделения памяти до момента ее освобождения. Область действия зависит от места описания указателя, через который производится работа с массивом. Область действия и время жизни указателей подчиняются общим правилам. Локальная переменная при выходе из блока, в котором она описана, «теряется». Если эта переменная является указателем и в ней хранится адрес выделенной динамической памяти, при выходе из блока эта память перестает быть доступной, однако не помечается как свободная, поэтому не может быть использована в дальнейшем. Это называется утечкой памяти и является распространенной ошибкой.

Пример 2. Дан массив из N элементов. Найти сумму элементов массива. Использовать различные варианты при обращении к массиву.

```
#include <iostream>
using namespace std;
void main(){
 setlocale(LC_ALL, "Russian");
 int N;
 cout << "Введите размерность массива: ";
 cin >> N;
 int *a = new int[N]; //создаем массив
 for (int i = 0; i < N; i++)
  cin >> a[i];
 int s = 0;
 for (int i = 0; i < N; i++)
  s += *(a + i);
```

```
 //выведем элементы массива и их сумму
 cout << "Массив: ";
 for (int i = 0; i < N; i++)
  cout << " " << i[a];
 cout << endl << "Сумма элементов: " << s;
 delete[] a;
}
```

Для динамического выделения памяти под двумерные массивы (матрицы) можно предложить несколько способов: через одномерный массив, через массив указателей, с использованием дополнительного массива. Рассмотрим каждый из этих способов.

Реализация двумерного массива посредством одномерного является естественной в силу того, что в памяти ЭВМ элементы обоих массивов хранятся последовательно. Единственный недостаток такого представления заключается в том, что недоступен автоматический перевод двумерного индекса в одномерный.

Двумерная матрица, содержащая `n` строк и `m` столбцов, будет располагаться в оперативной памяти в форме ленты, состоящей из элементов строк. При этом индекс любого элемента двумерной матрицы можно получить по формуле

```
index = i * m + j;
```

где `i` – номер текущей строки; `j` – номер текущего столбца. Объем памяти, требуемый для размещения двумерного массива, определится как `n*m*(размер элемента)`. Однако поскольку при таком объявлении компилятору явно не указывается количество элементов в строке и столбце двумерного массива, традиционное обращение к элементу путем указания индекса строки и индекса столбца является некорректным:

`a[i][j]` – некорректно.

Правильное обращение к элементу с использованием указателя будет выглядеть как `*(p+i*m+j)`, где `p` – указатель на

массив, m – количество столбцов, i – индекс строки, j – индекс столбца.

Пример 3. Реализовать определение динамического двумерного массива размерности N × M, ввести и вывести его элементы.

```
#include <iostream>
using namespace std;
int main() {
 setlocale(LC_ALL, "Russian");
 int n, m;
 cout << "введите количество строк: ";
 cout << "n= ";   cin >> n;
 cout << "введите количество столбцов: ";
 cout << "m= ";   cin >> m;
 int *a = new int[n*m];  // Выделение памяти
 for (int i = 0; i<n; i++)
  for (int j = 0; j<m; j++) {
   cout << "a[" << i << "," << j << "] = ";
  cin >> *(a + i*m + j);
 }
 // Вывод элементов массива
 int *p = a; //p-для адреса элемента массива
 for (int i = 0; i<n; i++) {
     for (int j = 0; j<m; j++, p++)
         cout << " " << *p;
     cout<<endl;
 }
 delete[] a;
 return 0;
}
```

В цикле ввода явно определяется местоположение адреса очередного элемента массива, а в цикле вывода используется дополнительная переменная p, которая последовательно указывает на каждый элемент.

Для реализации двумерного массива с помощью массива указателей необходимо:

– выделить блок оперативной памяти под массив указателей;

– выделить блоки оперативной памяти под одномерные массивы, представляющие собой строки искомой матрицы;

– записать адреса строк в массив указателей.

При таком способе выделения памяти компилятору явно указывается количество строк и количество столбцов в массиве.

Пример 4. Реализовать определение динамического двумерного массива размерности N × M с помощью массива указателей. Ввести и вывести его элементы.

```
#include <iostream>
using namespace std;
int main(){
 int n, m;  cout << "n = ";   cin >> n;
 cout << "m = ";   cin >> m;
 int **a = new int*[n]; /* выделение памяти
                    под указатели на строки*/
 for(int i = 0; i<n; i++){
  a[i] = new int[m]; /* выделение памяти под
                          i-ю строку*/
  for (int j = 0; j<m; j++)
   cin >> a[i][j];
  }
 // вывод элементов массива
 for (int i = 0; i < n; i++) {
     for (int j = 0; j < m; j++)
         cout << a[i][j] << " ";
     cout << endl;
 }
 for (int i = 0; i < n; i++)
  delete[] a[i]; /*освобождение памяти под
                       i-ю строку*/
 delete[] a; //удаление указателей на строки
 return 0;
}
```

Схема получившегося массива представлена на рис. 6.

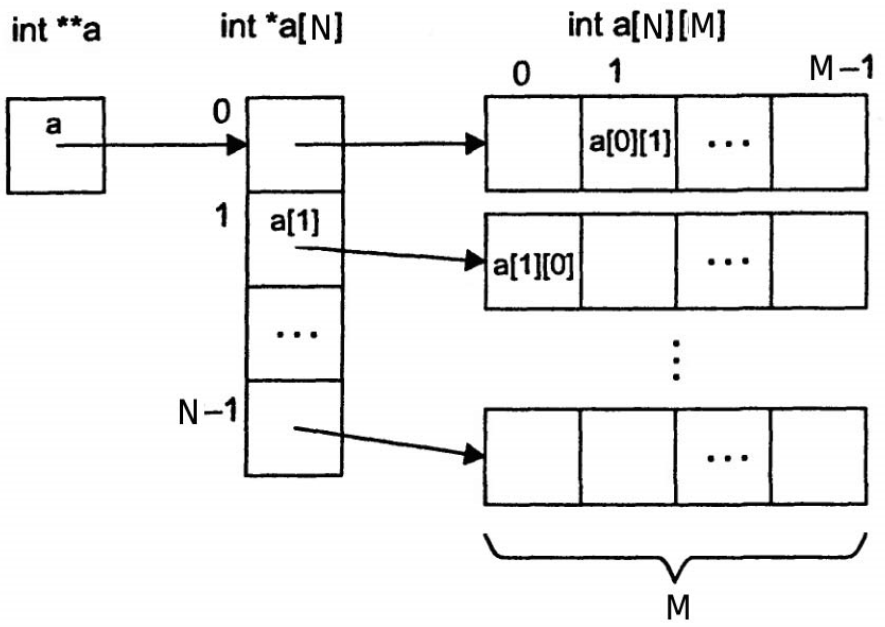


Рис. 6. Схема двумерного массива для передачи в функцию

Реализация двумерного массива с помощью вспомогательного массива, хранящего длину каждой строки, описывает так называемые свободные массивы. Свободным называется двухмерный массив, размер строк которого может быть различным. Преимущество использования свободного массива заключается в том, что не требуется отводить память компьютера с запасом для размещения строки максимально возможной длины. Фактически свободный массив представляет собой одномерный массив указателей на одномерные массивы данных.

Для размещения в оперативной памяти двухмерного массива со строками разной длины необходимо ввести дополнительный массив m, в котором будут храниться размеры строк.

Пример 5. Реализовать определение динамического свободного массива размерности. Ввести и вывести его элементы.

```
#include <iostream>
using namespace std;
int main(){
 setlocale(LC_ALL, "Russian");
```

```
int n;
cout << "n= ";
 cin >> n; // количество строк
int **a = new int*[n];  /* массив
                     указателей на строки*/
int *m = new int[n];  // массив длин строк
for (int i = 0; i<n; i++) {
    cout << " размер " << i << " строки: ";
    cin >> m[i];/*количество элементов i-ой
                    строки*/
    a[i] = new int[m[i]]; //адрес строки
    for (int j = 0; j < m[i]; j++)   {
     cout << "a["<< i <<"]["  << j << "]= ";
        cin >> a[i][j];
    }
}
// Вывод элементов массива
for (int i = 0; i < n; i++)   {
    for (int j = 0; j < m[i]; j++)
        cout << a[i][j]<<" ";
    cout<<endl;
}
for (int i = 0; i < n; i++)
 delete[] a[i]; /*освобождение памяти под
                    строки*/
delete[] a;  /* удаление массива указателей
                на строки*/
delete[] m;  // удаление массива длин строк
return 0;
}
```

Контрольные вопросы и задания

1. Дан одномерный массив, состоящий из N целочисленных элементов. Вывести массив на экран в обратном порядке.

2. Одномерный массив, состоящий из N целочисленных элементов, заполнить случайными числами. Вычислить сумму нечетных элементов массива.

3. Дан одномерный массив, состоящий из N вещественных элементов. Вычислить среднеарифметическое положительных элементов массива.

4. Дан одномерный массив, состоящий из N вещественных элементов. Вывести отрицательные элементы на экран в обратном порядке.

5. Дан одномерный массив, состоящий из N целочисленных элементов. Найти максимальный отрицательный элемент.

6. Дан одномерный массив, состоящий из N целочисленных элементов. Найти минимальный положительный элемент.

7. Дан одномерный массив, состоящий из N целочисленных элементов. Вычислить сумму положительных элементов массива, кратных 3.

8. Дан массив целых чисел. Вывести на экран сначала его четные элементы, затем нечетные.

9. Дан массив целых чисел. Все элементы, кратные числу 17, заменить нулем.

10. Дан массив целых чисел. Все нечетные элементы удвоить, а четные уменьшить вдвое.

11. Дан массив целых чисел. Определить количество четных элементов, индексы которых не кратны 3.

12. Дан массив целых чисел. Определить сумму положительных элементов с четными индексами.

13. Дан массив целых чисел. Вывести значения, которые не превосходят среднего арифметического.

14. Дан массив целых чисел. Определить количество положительных элементов с четными порядковыми номерами.

15. Дан массив целых чисел. Определить сумму элементов больших заданного числа x, порядковые номера которых не кратны 5.

16. Дан массив целых чисел. Заменить все кратные пяти элементы значением наибольшего элемента.

17. Дан двумерный массив вещественных чисел. Найти количество отрицательных элементов матрицы.

18. Дан двумерный массив вещественных чисел. Найти минимальный по абсолютной величине элемент матрицы.

19. Дан двумерный массив вещественных чисел. Найти сумму произведений элементов строки и столбца матрицы, на пересечении которых находится наибольший по абсолютной величине элемент матрицы.

20. Дан двумерный массив вещественных чисел. Найти количество элементов матрицы, превышающих ее среднее арифметическое значение.

21. Дан двумерный массив вещественных чисел. Найти сумму элементов массива, для которых сумма индексов равна k, где k – введенное с клавиатуры значение. Проверить, что значение k позволяет найти решение.

22. Дан двумерный массив вещественных чисел. Найти сумму положительных элементов столбца, содержащего максимальный элемент массива.

23. Дан двумерный массив вещественных чисел. Найти разность между максимальным и минимальным значениями элементов массива.

24. Дан двумерный массив вещественных чисел. Найти сумму элементов, расположенных по периметру массива.

25. Дан двумерный массив вещественных чисел. Найти произведение ненулевых элементов строки массива, на которой расположен максимальный элемент.

26. Дан двумерный массив вещественных чисел. Найти разность между суммой значений элементов четных и нечетных строк.

27. Дан двумерный массив вещественных чисел. Найти сумму элементов столбца, в котором расположен максимальный элемент массива.

28. Дана квадратная матрица целых чисел. Найти отношение сумм элементов, лежащих выше и ниже главной диагонали матрицы.

29. Дана квадратная матрица целых чисел размером. Найти произведение ненулевых элементов, лежащих выше главной диагонали матрицы.

15. ФУНКЦИИ В C++

Функции позволяют структурировать код и разбить его на индивидуальные задачи. Это упрощает написание и отладку кода, позволяет избежать избыточного дублирования кода в программе. В каждой программе на C или C++ определена минимум одна функция под названием main. Эта функция запускается автоматически. Все остальные функции вызываются из main тем или иным образом.

Функция – это подпрограмма в виде именованной последовательности операторов языка программирования, реализующая конкретную задачу и возвращающая некоторое значение. Вызов функции приводит к выполнению операторов в теле функций. К функции можно обратиться по имени, передать ей значения параметров и получить результат ее работы в виде возвращаемого значения.

Функция может принимать параметры и возвращать значение. Передача в функцию различных аргументов позволяет использовать один и тот же программный код многократно для разных данных.

Прототип функции (объявление, сигнатура, интерфейс) задает ее имя, арность, тип возвращаемого значения и список передаваемых параметров:

```
[ класс ] <тип> имя
            ([список_формальных_параметров]);
```

Определение функции содержит, кроме прототипа, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

```
[ класс ] тип имя ([список_форм_парам.]) {
  <тело функции>
}
```

С помощью необязательного модификатора `класс` можно явно задать область видимости функции, используя ключевые слова `extern` и `static`:

– `extern` – глобальная видимость во всех модулях программы (по умолчанию);

– `static` – видимость только в пределах модуля, в котором определена функция.

Тип возвращаемого функцией значения может быть любым, кроме массива и функции. Функция также может возвращать указатель на массив или функцию. Если функция не должна возвращать значение, указывается тип `void`.

Список формальных параметров определяет переменные, которые требуется передать в функцию при ее вызове. Элементы списка параметров разделяются запятыми. Для каждого параметра, передаваемого в функцию, указывается его тип и имя (в объявлении имена можно опускать).

Функция активируется с помощью оператора вызова функции, в котором содержатся имя функции и фактические параметры:

```
имя([список_фактических_параметров]);
```

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. Фактическими параметрами могут быть выражения или вызов других функций соответствующего формальным параметрам типа.

Для параметров можно указать значения по умолчанию. При этом все параметры правее данного должны также иметь значения по умолчанию. Если в определении функции присутствует параметр со значением по умолчанию, то данный параметр можно опустить при вызове функции. При этом все параметры левее данного должны присутствовать. Значение данного параметра внутри функции будет совпадать со значением параметра по умолчанию.

Функцию можно определить как встроенную с помощью модификатора `inline`, который рекомендует компилятору вместо обращения к функции копировать ее код непосредственно в каждую точку вызова. Модификатор `inline` ставится перед типом функции. Он применяется для коротких функций, чтобы

снизить накладные расходы на вызов (передача управления, передача параметров через стек). Директива `inline` носит рекомендательный характер и выполняется компилятором по мере возможности. Использование `inline`-функций может увеличить объем исполняемой программы. Определение функции должно предшествовать ее вызовам, иначе вместо `inline`-расширения компилятор сгенерирует обычный вызов.

Все переменные, описанные внутри функции, а также ее параметры, являются локальными переменными. При вызове функции в стеке потока автоматически выделяется память под локальные переменные. Кроме того, в стеке сохраняется содержимое регистров процессора, используемых вызываемой функцией, на момент, предшествующий её вызову, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции и восстановить значения регистров.

При вызове функции слева-направо вычисляются выражения, стоящие на месте аргументов, затем в стеке выделяется память под формальные параметры функции в соответствии с их типом, каждому из них присваивается значение соответствующего аргумента. При этом проверяется соответствие типов и при необходимости выполняются их преобразования.

При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются. Если требуется сохранить значение локальных переменных при последовательных вызовах функции, то при объявлении переменной используется модификатор `static`.

Выделение и освобождение памяти под локальные переменные функции происходит очень эффективно, по сравнению с динамической памятью, так как для этого достаточно передвинуть указатель на верхушку стека потока.

Для возврата из функции в теле функции должен присутствовать оператор `return`, после которого следует выражение, вычисляющее возвращаемое функцией значение соответствующего типа. Операторов `return` может быть

несколько в одной функции, однако для корректного завершения работы функции обязательно должен выполниться только один.

Для вызова функции достаточно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов или выражения. Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который возвращает функция. Если тип возвращаемого функцией значения не `void`, то она может входить в состав выражений или располагаться в правой части оператора присваивания.

Пример 1. Использование функции, возвращающей сумму двух целых величин.

```
#include <iostream>
using namespace std;
int sum(int a, int b); // объявление функции
int main() {
 int a = 2, b = 3, c, d;
 c = sum(a, b);              // вызов функции
 cin >> d;
 cout << sum(c, d) << endl; // вызов функции
 a = 3 * sum(c, d) - b;     // вызов функции
 cout << a << endl;
 return 0;
}
int sum(int a, int b){// определение функции
 return (a + b);
}
```

Объявление и определение функции может совпадать, но перед вызовом компилятор должен знать прототип функции, поэтому одновременное объявление и определение должно следовать перед вызовом функции.

Пример 2. Использование функции нахождения максимума двух чисел:

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

```
int max(int x, int y) { /* одновременно
          объявление и определение функции*/
 if (x > y)
  return x;
 else
  return y;
}
int main() {
 int a = 2, b = 3, c, d;
 c = max(a, b);       // вызов функции
 cout << c << endl;
 cin >> d;
 cout << max(c, d) << endl; // вызов функции
 return 0;
}
```

Помимо локальных переменных в теле функции можно также обращаться ко всем глобальным переменным:

Пример 3. Использование функцией глобальной переменной.

```
#include <iostream>
using namespace std;
int count = 200;
void func2() {
  cout << count;
}

void func1(int new_count = 100){ /* параметр
                  со значением по умолчанию*/
 count = new_count;
}
int main() {
 func2(); // выведет 200
 func1();
 func2(); // выведет 100
 return 0;
}
```

Существует два основных способа передачи параметров в функцию: по значению и по адресу.

При передаче по значению в стек заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а следовательно, нет и возможности их изменить.

При передаче по адресу в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов. Данный способ подразделяется на передачу параметров через указатель и по ссылке.

Пример 4. Передача параметров функции.

```
#include <iostream>
using namespace std;
void f(int a, int* b, int& c){
  a++;
  (*b)++;
  c++;
}
int main() {
 int a = 1, b = 1, c = 1;
 cout << "a b c" << endl;
 cout << a << ' ' << b << ' ' << c << endl;
 f(a, &b, c);
 cout << a << ' ' << b << ' ' << c << endl;
 return 0;
}
```

Результат работы программы:

```
a b c
1 1 1
1 2 2
```

Первый параметр (a) передается по значению. Его изменение в функции не влияет на исходное значение. Второй параметр (b) передается по адресу с помощью указателя, при этом для передачи в функцию адреса фактического параметра

используется операция взятия адреса, а для получения его значения в функции требуется операция разыменования. Третий параметр (c) передается по адресу с помощью ссылки.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются. Поэтому использование ссылок вместо указателей улучшает читаемость программы, избавляя от необходимости применять операции получения адреса и разыменования. Использование ссылок вместо передачи по значению более эффективно, поскольку не требует копирования параметров, что имеет значение при передаче структур данных большого объема.

Если требуется изменить значение параметра внутри функции, то он передается либо через ссылку, либо через указатель.

Если требуется запретить изменение параметра внутри функции, используется модификатор `const`:

```
int f(const char *);
```

Таким образом, исходные данные, которые не должны изменяться в функции, предпочтительнее передавать ей с помощью константных ссылок.

По умолчанию параметры любого типа, кроме массива и функции (например, вещественного, структурного, перечисление, объединение, указатель), передаются в функцию по значению.

Параметрами функции могут быть массивы, и функции могут возвращать указатель на массив в качестве результата. При использовании массивов в качестве параметров в функцию передается указатель на его первый элемент, т.е. массив всегда передается по адресу. При этом информация о количестве элементов теряется, поэтому следует передавать размерность массива как дополнительный параметр.

Пример 5. Использование функции, возвращающей значение максимального элемента массива неотрицательных целых чисел:

```
#include <iostream>
using namespace std;
int max_vect(int n, int *x) {
   int max = 0;
   for (int i = 0; i < n; i++)
     if (x[i] > max)
       max = x[i];
  return max;
}
int main() {
  const int N = 7;
  int a[N] = {1,4,3,1,5,6,1};
  cout << max_vect(N, a);
  return 0;
}
```

При передаче многомерных массивов все размерности, если они неизвестны на этапе компиляции, должны передаваться в качестве параметров. Внутри функции массив интерпретируется как одномерный, а его индекс пересчитывается в программе.

Пример 6. Использование функции нахождения суммы элементов двумерного массива.

```
#include <iostream>
using namespace std;
int sum(int *x, const int n, const int m) {
   int s = 0;
   for (int i=0; i < n; i++)
     for (int j=0; j < m; j++)
       s += x[i,j];
 return s;
}
int main() {
   int a[2][2]={{1,2},{3,4}};
   /* имя массива a напрямую передавать
нельзя из-за несоответствия типов поэтому
используется адрес первой ячейки &a[0][0] */
   cout << sum(&a[0][0], 2, 2);
```

```
    return 0;
}
```

Приведенное решение использует статический массив.

Решение с использованием динамического массива имеет следующий вид:

```
#include <iostream>
using namespace std;
int sum(int **x, const int n, const int m) {
    int s = 0;
    for (int i=0; i < n; i++)
        for (int j=0; j < m; j++)
            s += x[i][j];
    return s;
}
int main() {
    const int N=4, M=3;
    int **a = new int* [N];
    for(int i=0; i < N; i++)
        a[i] = new int[M];
    for(int i=0; i < N; i++){
     for(int j=0; j < M; j++){
            a[i][j] = rand()%10;
            cout << a[i][j] << "\t";
     }
     cout << endl;
    }
    cout << sum(a, N, M);
    for(int i=0; i < N; i++)
        delete [] (a[i]);
    delete [] a;
}
```

В C++ допустимо использовать указатели на функцию. Указатели на функции задаются следующим образом:

```
void f() {} // некоторая функция
void g() {} // еще функция
```

```
void (*pf)() = &f; // указатель на функцию с
присваиванием адреса f
f(); // вызов функции f

pf = &g; /* присваиваем указателю
            адрес функции g*/
pf = g; // работает аналогично
pf(); // вызов функции g
```

Когда функция вызывает саму себя, то говорят, что имеет место рекурсия.

Пример 7. Пример рекурсивной реализации функции подсчета факториала:

```
long factorial(long i) {
 if (i > 0)
  return i * factorial(i – 1); //вызов
 else
  return 1; // выход из рекурсии
}
```

Большая вложенность рекурсивных вызовов может приводить к исчерпанию памяти стека. Для оптимизации использования памяти и увеличения скорости работы рекурсивных алгоритмов современные компиляторы выполняют оптимизацию хвостового вызова (tail call optimization, TCO), т. е. преобразуют рекурсию в цикл в случае, если рекурсивный вызов расположен в конце тела функции.

Контрольные вопросы и задания

1. Написать функцию, возвращающую процент нулевых чисел последовательности из N целых чисел.

2. Написать функцию, которая определяет, относится ли вводимая последовательность из N натуральных чисел к ряду Фибоначчи. Если да, то возвратить 1, иначе – возвратить 0.

3. Написать функцию, которая определяет и возвращает количество совершенных чисел в последовательности натуральных чисел N.

4. Написать функцию, которая определяет, является ли последовательность натуральных чисел N строго возрастающей. Возвратить 1 – если является, 0 – в противном случае.

5. Написать функцию, которая определяет наименьшее число последовательности из N натуральных чисел, среди чисел, больших 5. Возвратить наименьшее число, 0 – в противном случае.

6. Написать функцию, которая определяет, содержит ли последовательность целых чисел N хотя бы три отрицательных числа. Возвратить 1 – если содержит, 0 – в противном случае.

7. Написать функцию, которая определяет, является ли последовательность целых чисел N знакочередующейся. Возвратить 1 – если является, 0 – в противном случае.

8. Написать функцию, возвращающую процент отрицательных чисел последовательности из N целых чисел.

9. Написать функцию, которая определяет и возвращает сумму натуральных чисел последовательности N, порядковые номера которых являются числами Фибоначчи.

10. Написать функцию, которая определяет и возвращает разность между произведением нечётных чисел и наибольшим среди отрицательных чисел последовательности целых чисел N.

11. Написать функцию, которая определяет и возвращает среднее арифметическое делителей натурального числа N.

12. Написать функцию, которая определяет и возвращает разницу между минимальным положительным и максимальным отрицательным элементами последовательности.

13. Написать функцию, которая определяет среднее арифметическое среди элементов последовательности из N натуральных чисел, кратных 5. Возвратить 0 – если таких чисел нет, иначе – среднее арифметическое.

14. Написать функцию, которая определяет, содержит ли последовательность из N натуральных чисел хотя бы два числа, кратных 3 и 5. Возвратить 1 – если содержит, 0 – в противном случае.

15. Написать функцию, которая определяет, является ли последовательность строго убывающей. Возвратить 1 – если

является, 0 – в противном случае. Использовать передачу массива по адресу.

16. Написать функцию, возвращающую i-й по порядку элемент последовательности Фибоначчи (0 1 1 2 3 5 8 ...). Использовать рекурсию.

17. Дан двумерный массив. С использованием функции найти строку с максимальной суммой элементов. Если таких строк несколько, должен быть найден номер самой нижней из них.

18. Дан двумерный массив. С использованием функции найти столбец с минимальной суммой элементов. Если таких столбцов несколько, должен быть найден номер самого левого из них.

19. Дан двумерный массив. С использованием функции проверить, является ли последовательность элементов некоторой строки массива упорядоченной по неубыванию. В случае отрицательного ответа в обеих задачах должны быть напечатаны координаты первого элемента, нарушающего указанную упорядоченность.

20. Дан двумерный массив целых чисел. С использованием функции каждом его столбце найти первый нечетный элемент (принять, что нечетные элементы есть в каждом столбце).

21. Дан двумерный массив целых чисел. С использованием функции каждой строки выяснить, имеются ли в ней положительные элементы.

22. Дан двумерный массив целых чисел. С использованием функции найти минимальный номер строки, состоящей только из положительных элементов.

23. Дан двумерный массив целых чисел. С использованием функции в каждой его строке заменить любой минимальный элемент на максимальный.

24. Дан двумерный массив целых чисел. С использованием функции в каждой строке двумерного массива поменять местами первый нулевой элемент и последний отрицательный. Если таких элементов нет, то должно быть выведено соответствующее сообщение.

ЗАКЛЮЧЕНИЕ

Освещение теоретических и практических вопросов разработки программ на языке программирования C++, представленных в данном пособии, помогает студентам создавать программные приложения в среде Microsoft Visual Studio, понимать структуру программы, граммотно использовать структурные элементы языка, такие как стандартные типы, переменные, ввод данных, оператор присваивания, выражения, условный оператор, операторы цикла, массивы, функции.

При дальнейшем изучении языка программирования C++ следует уделить внимание структурным типам, реализации алгоритмов обработки информации, классам, объектно-ориенированному подходу разработки приложений.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Седжвик Р. Алгоритмы на C++. 2-е изд., испр. М., 2016.
2. Страуструп Б. Язык программирования C++ для профессионалов. М., 2006.
3. Синицын С.В., Хлытчиев О.И. Основы разработки программного обеспечения на примере языка С. 2-е изд., испр. М., 2016.
4. Белоцерковская И.Е., Галина Н.В., Катаева И.Е. Алгоритмизация. Введение в язык программирования C++. 2-е изд., испр. М., 2016.
5. Павловская Т. А.С/C++. Процедурное и объектно-ориентированное программирование: учебник. СПб., 2019.
6. Ишкова Э. А. Изучаем C++ на задачах и примерах. СПб., 2016.
7. Лубашева Т.В., Железко Б.А. Основы алгоритмизации и программирования: учеб. пособие. Минск, 2016.
8. Информатика и программирование: учеб. пособие / Р.Ю. Царев, А.Н. Пупков, В.В. Самарин, Е.В. Мыльникова. Красноярск, 2014.
9. Информатика: учеб. пособие. Тамбов, 2015.

СОДЕРЖАНИЕ

Учебное издание

Авторы: Подколзин Вадим Владиславович
Лукащик Елена Павловна
Добровольская Наталья Юрьевна
Гаркуша Олег Васильевич
Синица Сергей Геннадьевич
Полупанов Алексей Александрович
Полетайкин Алексей Николаевич
Багдасарян Рафаэль Хачикович
Харченко Анна Владимировна
Уварова Анастасия Викторовна
Михайличенко Анна Александровна

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ С++

Учебно-методическое пособие

Подписано в печать 07.05.2018. Печать цифровая.
Формат 60×84 1/16. Уч.-изд. л. 9,8.
Тираж 500 экз. Заказ №

Кубанский государственный университет
350040, г. Краснодар, ул. Ставропольская, 149.

Издательско-полиграфический центр
Кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149