Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Кемеровский государственный университет»
Новокузнецкий институт (филиал)

Факультет информатики, математики и экономики

Кафедра информатики и вычислительной техники им. В.К. Буторина

А. В. Маркидонов

ПРАКТИКУМ НА ЭЛЕКТРОННО-ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ

Методические указания к выполнению самостоятельной работы по теме «Введение в язык С++» для обучающихся по направлению подготовки

09.03.03 Прикладная информатика (направленность (профиль) «Прикладная информатика в экономике»)

Новокузнецк

2019

УДК 004.021

ББК 32.973.2
М25

Маркидонов А.В.

М25 Практикум на электронно-вычислительных машинах : метод. указ. к выполнению самостоятельной работы по теме «Введение в язык С++» для обучающихся по направлению подготовки 09.03.03 Прикладная информатика (направленность (профиль) «Прикладная информатика в экономике») / А.В. Маркидонов : Новокузнец. ин-т (фил.) Кемеров. гос. ун-та. – Новокузнецк : НФИ КемГУ, 2019 – 36 с.

В работе изложены методические рекомендации к самостоятельной работе для студентов по дисциплине «Практикум на электронно-вычислительных машинах»: необходимый теоретический минимум, примеры программ, задания для самостоятельного решения и тесты для самопроверки.

Методические указания предназначены для студентов, обучающихся по направлению 09.03.03 Прикладная информатика (профиль «Прикладная информатика в экономике»).

Рекомендовано
на заседании кафедры
информатики и вычислительной
техники им. В.К. Буторина
26 ноября 2019 г.
Заведующий кафедрой

А.В. Маркидонов

Утверждено
методической комиссией
факультета информатики,
математики и экономики
12 декабря 2019 г.
Председатель комиссии

Г.Н. Бойченко

УДК 004.021
ББК 32.973.2

Текст представлен в авторской редакции

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Представленные методические рекомендации (МР) предназначены для выполнения самостоятельной работы обучающимися при изучении раздела «Введение в язык С++» дисциплины Б1.Б11 «Практикум на электронно-вычислительных машинах». На самостоятельную работу при изучении данного раздела дисциплины отводится от 20 до 30 часов в зависимости от формы обучения. Дисциплина включена в базовый цикл учебного плана бакалавриата по направлению подготовки 09.03.03 Прикладная информатика (направленность (профиль) «Прикладная информатика в экономике»).

Целью дисциплины «Практикум на электронно-вычислительных машинах» является приобретение студентами теоретических знаний и практических навыков программирования. Данная дисциплина направлена на формирование компетенции ОПК-4 (способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности).

В основной учебной литературе, включенной в перечень рабочей программы дисциплины, и представленной в конце данных МР, рассматриваемый раздел представлен в излишне большом объеме. В предлагаемых МР осуществлена попытка структурировать излагаемый материал без потери информативности и глубины проработки затрагиваемых тем.

Основная цель МР – систематизация и закрепление полученных теоретических знаний и практических умений, а также формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации.

Задачи:

- углубить и расширить теоретическую подготовку и практические умения;
- развить активность, познавательные способности и исследовательские умения;
- сформировать умение использовать учебную и специализированную литературу;
- подготовить к промежуточной аттестации по дисциплине.

Структура МР выстроена в соответствии с последовательностью изучения разделов в рабочей программе дисциплины. Излагаемый материал структурирован так, чтобы обучающийся мог изучить теорию, рассмотреть ряд примеров, а затем закрепить полученные знания путем решения задач, предназначенных для самостоятельного выполнения, а также путем ответов на поставленные вопросы.

ВВЕДЕНИЕ

Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день C++ является одним из самых популярных и распространенных языков.

Своими корнями он уходит в язык Си, который был разработан в 1969 – 1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист Бьерн Страуструп (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал C++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге "Си с классами" был переименован в C++. И с тех по оба языка стали развиваться независимо друг от друга.

C++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко C++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на C++. Но только системным программированием применение данного языка не ограничивается. C++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где C++ тоже нашел свое применение. И даже в веб-разработке также можно использовать C++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем C++ – язык широкого пользования, на котором можно создавать практически любые виды программ.

C++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на C++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на C++ в приложения под эти платформы.

В отличие от Си язык C++ позволяет писать приложения в объектно-ориентированном стиле, представляя программу как совокупность взаимодействующих между собой классов и объектов. Что упрощает создание крупных приложений.

Современные системы программирования на C++ состоят из нескольких составных частей. Это такие части, как сама среда программирования, язык, стандартная библиотека C-функций и различные библиотеки C-классов.

1. СТРУКТУРА ПРОГРАММЫ

Инструкции

Программа на C++ состоит из набора инструкций. Каждая инструкция (statement) выполняет определенное действие. В конце инструкции в языке C++ ставится точка с запятой (;). Данный знак указывает компилятору на завершение инструкции. Например:

```
std::cout << "Hello World!";
```

Данная строка выводит на консоль строку "Hello world!", является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может представлять блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

```
{
    std::cout << "Hello World!";
    std::cout << "Bye World!";
}
```

В этом блоке кода две инструкции, которые выводят на консоль определенную строку.

Функция main

Каждая программа на языке C++ должна иметь как минимум одну функцию - функцию main(). Именно с этой функции начинается выполнение приложения. Ее имя main фиксировано и для всех программ на Си всегда одинаково.

Функция также является блоком кода, поэтому ее тело обрамляется фигурными скобками, между которыми определяется набор инструкций.

В частности, при создании первой программы использовалась следующая функция main:

```
#include <iostream>               // подключаем заголовочный файл iostream

int main()                          // определяем функцию main
{                                   // начало функции
    std::cout << "Hello World!";   // выводим строку на консоль
    return 0;                       // выходим из функции
}                                   // конец функции
```

Определение функии main начинается с возвращаемого типа. Функция main в любом случае должна возвращать число. Поэтому ее определение начинается с ключевого слова int.

Далее идет название функции, то есть main. После названия в скобках идет список параметров. В данном случае функция main не принимает никаких параметров, поэтому после названия указаны пустые скобки. Однако есть другие варианты определения функции main, которые подразумевают использование параметров. В частности, нередко может встречаться следующее определение функции main, использующей параметры:

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

}
```

И после списка параметров идет блок кода, который и содержит в виде инструкций собственно те действия, выполняемые функцией main.

Директивы препроцессора

В примере выше на консоль выводится строка, но чтобы использовать вывод на консоль, необходимо в начале файла с исходным кодом подключать библиотеку iostream с помощью директивы include.

Директива include является директивой препроцессора. Каждая директива препроцессора размещается на одной строке. И в отличие от обычных инструкциий языка C++, которые завершаются точкой с запятой ; , признаком завершения препроцессорной директивы является перевод на новую строку. Кроме того, директива должна начинаться со знака решетки #. Непосредственно директива "include" определяет, какие файлы и библиотеки надо подключить в данном месте в код программы.

Комментарии

Исходный код может содержать комментарии. Комментарии позволяют понять смыл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияние на работу приложения и на его размер.

В языке C++ есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //:

```
#include <iostream>               // подключаем библиотеку iostream

int main()                          // определяем функцию main
{                                   // начало функции
    std::cout << "Hello World!";   // выводим строку на консоль
    return 0;                       // выходим из функции
}                                   // конец функции
```

Многострочный комментарий заключается между символами /* текст комментария */. Он может размещаться на нескольких строках. Например:

```
#include <iostream>
/*
    Определение функции Main
    Выводит на консоль строку Hello World!
*/
int main()
{
    std::cout << "Hello World!"; // вывод строки на консоль
    return 0;
}
```

2. ПЕРЕМЕННЫЕ

Как и во многих языках программирования, в C++ для хранения данных используются переменные. Переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная.

Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

```
тип_переменной имя_переменной;
```

Простейшее определение переменной:

```
int age;
```

Здесь определена переменная age, которая имеет тип int. Поскольку определение переменной представляет собой инструкцию, то после него ставится точка с запятой.

Имя переменной может представлять последовательность символов латинского алфавита, чисел и знака подчеркивания. При этом имя должно начинаться либо с алфавитного символа, либо со знака подчеркивания.

```
int _age33;
```

Других символов в названии переменной не должно быть. Например, следующее определение будет неправильным:

```
int $age;
```

Так как символ $ не является ни буквой, ни цифрой, ни символом подчеркивания.

Также стоит учитывать, что C++ – регистрозависимый язык, а это значит, что регистр символов имеет большое значение. То есть в следующем коде будут определяться две разные переменные:

```
int age;
int Age;
```

Поэтому переменная Age не будет представлять то же самое, что и переменная age.

Кроме того, в качестве имени переменной нельзя использовать ключевые слова языке C++, например, for или if. Но таких слов не так много: alignas, alignof, asm, auto, bool, break, case, catch, char, char16_t, char32_t, class, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutubale, namespace, new, noexcept, nullptr, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while.

Также нельзя объявить больше одной переменной с одним и тем же именем, например:

```
int age;
int age;
```

Подобное определение вызовет ошибку на этапе компиляции.

И в довершение следует сказать, что переменным стоит давать осмысленные имена, которые будут говорить об их предназначении.

Инициализация

После определения переменной можно присвоить некоторое значение:

```
int age;
age = 20;
```

Например, определим в прогамме переменную и выведем ее значение на консоль:

```
#include <iostream>

int main()
{
    int age;
    age = 28;
    std::cout<<"Age = " << age;
    return 0;
}
```

С помощью последовательности операторов << можно вывести несколько значений на консоль.

После компиляции и запуска скомпилированной программы на консоль будет выведено число 28.

Однако также можно сразу при определении переменной дать ей некоторое начальное значение. Данный прием называется инициализацией, то есть присвоение переменной начального значения:

```
#include <iostream>

int main()
{
    int age = 28;
    std::cout<<"Age = " << age;
    return 0;
}
```

Инициализация по умолчанию

Если переменную не инициализировать, то происходит ее инициализация по умолчанию. И переменная получает некоторое значение по умолчанию, которое зависит от места, где эта переменная определена.

Если переменная, которая представляет встроенный тип (например, тип int), определена внутри функции, то она получает неопределенное значение. Если переменная встроенного типа определена вне функции, то она получает то значение по умолчанию, которое соответствует ее типу. Для числовых типов это число 0. Например:

```
#include <iostream>

int x;
int main()
{
    int y;
    std::cout <<"X = " << x << "\n";
    std::cout <<"Y = " << y;

    return 0;
}
```

Переменная x определена вне функции, и поэтому она получит значение по умолчанию - число 0. А переменная y определена внутри функции main, поэтому ее значение будет неопределенным. В частности, вывод программы на консоль может выглядеть следующим образом:

```
X = 0
Y = 4200475
```

Но в любом случае перед использованием переменной лучше явным образом назначать ей определенное значение, а не полагаться на значение по умолчанию.

Изменение значения

Ключевой особенностью переменных является то, что мы можем изменять их значения:

```
#include <iostream>

int main()
{
    int x = 6;
```

```
    x = 8;
    x = 10;
    std::cout <<"X = " << x; // X = 10

    return 0;
}
```

3. ТИПЫ ДАННЫХ

Каждая переменная имеет определенный тип. И этот тип определяет, какие значения может иметь переменная, какие операции с ней можно производить и сколько байт в памяти она будет занимать. В языке C++ определены следующие базовые типы данных:

bool: логический тип. Может принимать одну из двух значений true (истина) и false (ложь). Размер занимаемой памяти для этого типа точно не определен.

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

signed char: представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любой значение из диапазона от -128 до 127

unsigned char: представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любой значение из диапазона от 0 до 255

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

char16_t: представляет один символ в кодировке Unicode. Занимает в памяти 2 байта (16 бит). Может хранить любой значение из диапазона от 0 до 65 535

char32_t: представляет один символ в кодировке Unicode. Занимает в памяти 4 байта (32 бита). Может хранить любой значение из диапазона от 0 до 4 294 967 295

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

unsigned short: представляет целое число в диапазоне от 0 до 65535. Занимает в памяти 2 байта (16 бит).

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). Но в любом случае размер должен быть больше или равен размеру типа short и меньше или равен размеру типа long

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).

long: представляет целое число в диапазоне от −2 147 483 648 до 2 147 483 647. Занимает в памяти 4 байта (32 бита).

unsigned long: представляет целое число в диапазоне от 0 до 4 294 967 295. Занимает в памяти 4 байта (32 бита).

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).

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).

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)

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)

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.

void: тип без значения

Таким образом, все типы данных за исключением void могут быть разделены на три группы: символьные (char, wchar_t, char16_t, char32_t), целочисленные (short, int, long, long long) и типы чисел с плавающей точкой (float, double, long double).

Символьные типы

Для представления символов в приложении используются типы char, wchar_t, char16_t и char32_t.

Определим несколько переменных:

```
char c ='d';
wchar_t d ='c';
```

Переменная типа char в качестве значения принимает один символ в одинарных кавычках: char c ='d'. Также можно присвоить число из указанного выше в списке диапазона: char c = 120. В этом случае значением переменной с будет тот символ, который имеет код 120 в таблице символов ASCII.

Стоит учитывать, что для вывода на консоль символов wchar_t следует использовать не std::cout, а поток std::wcout:

```
#include <iostream>

int main()
{
    char a = 'H';
    wchar_t b = 'e';
    std::wcout << a << b << '\n';
    return 0;
}
```

При этом поток std::wcout может работать как с char, так и с wchar_t. А поток std::cout для переменной wchar_t вместо символа будет выводить его числовой код.

В стандарте C++11 были добавлены типы char16_t и char32_t, которые ориентированы на использование Unicode. Однако на уровне ОС пока не реализованы потоки для работы с этими типами. Поэтому если потребуется вывести на консоль значения переменных этих типов, то необходимо преобразовать переменные к типам char или wchar_t:

```
#include <iostream>

int main()
{
    char a = 'H';
    wchar_t b = 'e';
    char16_t c = 'l';
    char32_t d = 'o';
    std::cout << a << (char)b << (char)c << (char)d << "\n";
    return 0;
}
```

В данном случае при выводе перед переменными указывается операция приведения к типу char - (char), благодаря чему значения переменных b, c и d преобразуются в тип char и могут быть выведены на консоль с помощью потока std::cout.

Целочисленные типы

Целочисленные типы представлены следующими типами: short, unsigned short, int, unsigned int, long, unsigned long, long long и unsigned long long:

```
short a = -10;
unsigned short b= 10;
int c = -30;
unsigned int d = 60;
long e = -170;
unsigned long f = 45;
long long g = 89;
```

Типы чисел с плавающей точкой

Типы чисел с плавающей точкой иили дробные числа представлены такими типами как float, double и long double:

```
float a = -10.45;
double b = 0.00105;
long double c = 30.890045;
```

Размеры типов данных

В выше приведенном списке для каждого типа указан размер, который он занимает в памяти. Однако стоит отметить, что предельные размеры для типов разработчики компиляторов могут выбирать самостоятельно, исходя из аппаратных возможностей компьютера. Стандарт устанавливает лишь минимальные значения, которые должны быть. Например, для типов int и short минимальное значение – 16 бит, для типа long – 32 бита, для типа long double – 64 бита. При этом размер типа long должен быть не меньше размера типа int, а размер типа int – не меньше размера типа short, а размер типа long double должен быть больше double. К примеру, компилятор g++ под Windows для long double использует 12 байт, а компилятор, встроенный в Visual Studio и также работающий под Windows, для long double использует 8 байт. То есть даже в рамках одной платформы разные компиляторы могут по разному подходить к размерам некоторых типов данных. Но в целом используются те размеры, которые указаны выше при описании типов данных.

Однако бывают ситуации, когда необходимо точно знать размер определенного типа. И для этого в C++ есть оператор sizeof(), который возвращает размер памяти в байтах, которую занимает переменная:

```
#include <iostream>

int main()
{
    long double number = 2;
    std::cout << "sizeof(number) =" << sizeof(number);
    return 0;
}
```

Консольный вывод при компиляции в g++:

```
sizeof(number) = 12
```

При этом при определении переменных важно понимать, что значение переменной не должно выходить за те пределы, которые очерчены для ее типа. Например:

```
unsigned short number = -65535;
```

Компилятор g++ при компиляции программы с этой строкой выдаст ошибку о том, что значение -65535 не входит в диапазон допустимых значений для типа unsigned short и будет усечено.

В Visual Studio компиляция может пройти без ошибок, однако при этом переменная number получит значение 2 – результат преобразования числа -65535 к типу unsigned short. То есть опять же результат будет не совсем тот, который ожидается. Значение переменной – это всего лишь набор битов в памяти, которые интерпретируются в соответствии с определенным типом. И для разных типов один и тот же набор битов может интерпретироваться по разному. Поэтому важно учитывать диапазоны значений для того или иного типа при присвоении переменной значения.

Спецификатор auto

Иногда бывает трудно определить тип выражения. И согласно последним стандартам можно предоставить компилятору самому выводить тип объекта. И для этого применяется спецификатор auto. При этом если мы определяем переменную со спецификатором auto, эта переменная должна быть обязательно инициализирована каким-либо значением:

```
auto number = 5;
```

На основании присвоенного значения компилятор выведет тип переменной. Неинициализированные переменные со спецификатором auto не допускаются:

```
auto number;
```

4. КОНСТАНТЫ

Отличительной особенностью переменных является то, что мы можем многократно в течение работы программы изменять их значение:

```
int x = 7;
x = 9;
x = 5;
```

Но кроме переменных в языке программирования C++ можно определять константы. Их значение устанавливается один раз и впоследствии мы его не можем изменить. Константа определяется практически также, как и переменная за тем исключением, что в начале определения константы идет ключевое слово const. Например:

```
const int x = 22;
std::cout << x;
```

Если же мы захотим после определения константы присвоить ей некоторое значение, то компилятор не сможет скомпилировать программу и выведет ошибку:

```
const int x = 22;
x = 78;
```

То есть такой код не будет работать. И так как нельзя изменить значения константы, то ее всегда необходимо инициализировать, если мы хотим, чтобы она имела некоторое значение.

Если константа не будет инициализирована, то компилятор также выведет ошибку и не сможет скомпилировать программу, как в следующем случае:

```
const int x;
```

В качестве значения константам можно передавать как обычные литералы, так и динамически вычисляемые значения, например, значения переменных или других констант:

```
int a = 10;
const int b = 7;
const int d = b;
const int x = a;
```

Обычно в качестве констант определяются такие значения, которые должны оставаться постоянными в течение работы всей программы и не могут быть изменены. Например, если программы выполняет математические операции с использованием числа π, то было бы оптимально определить данные значение как константу, так как оно все равно в принципе неизменно:

```
const float pi = 3.14;
```

5. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Арифметические операции производятся над числами. Значения, которые участвуют в операции, называются операндами. В языке программирования C++ арифметические операции бинарными (производятся над двумя операндами) и унарными (выполняются над одним операндом). К бинарным операциям относят следующие:

1. Операция сложения + возвращает сумму двух чисел:

```
int a = 10;
int b = 7;
int c = a + b;  // 17
int d = 4 + b;  // 11
```

2. Операция вычитания - возвращает разность двух чисел:

```
int a = 10;
int b = 7;
int c = a - b;  // 3
int d = 41 - b; // 34
```

3. Операция умножения * возвращает произведение двух чисел:

```
int a = 10;
int b = 7;
int c = a * b;  // 70
int d = b * 5;  // 35
```

4. Операция деления / возвращает частное двух чисел:

```
int a = 20;
int b = 5;
int c = a / b;  // 4
double d = 22.5 / 4.5;  // 5
```

При делении стоит быть внимательным, так как если в операции участвуют два целых числа, то результат деления будет округляться до целого числа, даже если результат присваивается переменной float или double:

```
double k = 10 / 4;     // 2
```

```
std::cout << k;
```

Чтобы результат представлял число с плавающей точкой, один из операндов также должен представлять число с плавающей точкой:

```
double k = 10.0 / 4;     // 2.5
std::cout << k;
```

5. Операция получения остатка от целочисленного деления %:

```
int a = 33;
int b = 5;
int c = a % b;  // 3
int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

Также есть две унарные арифметические операции, которые производятся над одним числом: ++ (инкремент) и -- (декремент). Каждая из операций имеет две разновидности: префиксная и постфиксная:

6. Префиксный инкремент.

Увеличивает значение переменной на единицу и полученный результат используется как значение выражения ++x

```
int a = 8;
int b = ++a;
std::cout << a << "\n"; // 9
std::cout << b << "\n"; // 9
```

7. Постфиксный инкремент.

Увеличивает значение переменной на единицу, но значением выражения x++ будет то, которое было до увеличения на единицу

```
int a = 8;
int b = a++;
std::cout << a << "\n"; // 9
std::cout << b << "\n"; // 8
```

8. Префиксный декремент.

Уменьшает значение переменной на единицу, и полученное значение используется как значение выражения --x

```
int a = 8;
int b = --a;
std::cout << a << "\n"; // 7
std::cout << b << "\n"; // 7
```

9. Постфиксный декремент.

Уменьшает значение переменной на единицу, но значением выражения x-- будет то, которое было до уменьшения на единицу

```
int a = 8;
int b = a--;
std::cout << a << "\n"; // 7
std::cout << b << "\n"; // 8
```

Арифметические операции вычисляются слева направо. Одни операции имеют больший приоритет чем другие и поэтому выполняются вначале. Операции в порядке уменьшения приоритета:

1) + (инкремент), - (декремент)
2) (умножение), / (деление), % (остаток от деления)
3) + (сложение), - (вычитание)

Приоритет операций следует учитывать при выполнении набора арифметических выражений:

```
int a = 8;
int b = 7;
int c = a + 5 * ++b;      // 48
std::cout << c;
```

Хотя операции выполняются слева направо, но вначале будет выполняться операция инкремента ++b, которая увеличит значение переменной b и возвратит его в качестве результата, так как эта операция имеет больший приоритет. Затем выполняется умножение 5 * ++b, и только в последнюю очередь выполняется сложение a + 5 * ++b

Скобки позволяют переопределить порядок вычислений. Например:

```
int a = 8;
int b = 7;
int c = (a + 5) * ++b;      // 104
std::cout << c;
```

Несмотря на то, что операция сложения имеет меньший приоритет, но вначале будет выполняться именно сложение, а не умножение, так как операция сложения заключена в скобки.

6. УСЛОВНЫЕ ВЫРАЖЕНИЯ

Условные выражения представляют собой некоторое условие и возвращают значение типа bool, то есть значение true (если условие истинно), либо значение false (если условие ложно). К условным выражениям относятся операции отношения и логические операции.

Операции отношения

В языке программирования C++ есть следующие операции отношения:

1. Операция "равно" = =. Возвращает true, если оба операнда равны, и false, если они не равны:

```
int a = 10;
int b = 4;
bool c = a == b;    // false
bool d = a == 10;       // true
```

2. Операция "больше чем" >. Возвращает true, если первый операнд больше второго, и false, если первый операнд меньше второго:

```
int a = 10;
int b = 4;
bool c = a > b;  // true
```

3. Операция "меньше чем" <. Возвращает true, если первый операнд меньше второго, и false, если первый операнд больше второго:

```
bool c = 10 < 4; // false
```

4. Операция "меньше или равно" <=. Возвращает true, если первый операнд меньше или равен второму, и false, если первый операнд больше второго:

```
bool c = 10 <= 4;    // false
bool d = 10 <= 14;   // true
```

5. Операция "больше или равно" >=. Возвращает true, если первый операнд больше или равен второму, и false, если первый операнд меньше второго:

```
bool c = 10 >= 4;    // true
bool d = 10 >= 14;   // false
```

6. Операция "не равно" !=. Возвращает true, если первый операнд не равен второму, и false, если оба операнда равны:

```
bool c = 10 != 4;   // true
bool d = 4 != 4;    // false
```

Обычно операции отношения применяются в условных конструкциях типа if...else, которые будут рассмотрены далее.

Логические операции

Логические операции обычно объединяют несколько операций отношения. К логическим операциям относят следующие:

1. ! (операция отрицания)

Унарная операция, которая возвращает true, если операнд равен false. Если операнд равен true, операция возвращает false.

```
bool a = true;
bool b = !a;    // false
bool c =!10<5;   // true
```

2. && (конъюнкция, логическое умножение)

Возвращает true, если оба операнда не равны false. Возвращает false, если хотя бы один операнд равен false.

```
bool a = true;
bool b = false;
bool c = a && b;    // false
bool d = a && true; // true
```

3. || (дизъюнкция, логическое сложение)

Возвращает true, если хотя бы один операнд равен true. Возвращает false, если оба операнда равны false.

```
bool a = true;
bool b = false;
bool c = a || b;        // true
bool d = b || false;    // false
```

Используем одновременно несколько логических операций и операций сравнения:

```
#include <iostream>

int main()
{
    bool a = -2 > 5;            // false
```

```
    bool b = 0 < 7;              // true
    bool c = 10 > 5 && 7 < 11;    // true
    bool d = a && b || c;       // true
    std::cout << a << "\n";     // 0
    std::cout << b << "\n";     // 1
    std::cout << c << "\n";     // 1
    std::cout << d << "\n";     // 1
    return 0;
}
```

Следует обратить внимание, что на консоль выводится не непосредственно true или false, а их числовые эквиваленты - 1 для true и 0 для false.

И также стоит учитывать, что операции отношения имеют больший приоритет, чем логические операции. Поэтому в выражении 10 > 5 && 7 < 11 вначале будут выполняться подвыражения – операции отношения 10 > 5 и 7 < 11, а затем собственно операция логического умножения.

7. УСЛОВНЫЕ ВЫРАЖЕНИЯ

Побитовые операции выполняются над отдельными разрядами или битами чисел. Данные операции производятся только над целыми числами.

Операции сдвига

Каждое целое число в памяти представлено в виде определенного количества разрядов. И операции сдвига позволяют сдвинуть битовое представление числа на несколько разрядов вправо или влево. Операции сдвига применяются только к целочисленным операндам. Есть две операции:

1. << Сдвигает битовое представление числа, представленного первым операндом, влево на определенное количество разрядов, которое задается вторым операндом.
2. >> Сдвигает битовое представление числа вправо на определенное количество разрядов.

Применение операций:

```
int a = 2 << 2;           // 10  на два разрядов влево = 1000 - 8
int b = 16 >> 3;          // 10000 на три разряда вправо = 10 - 2
```

Число 2 в двоичном представлении 10. Если сдвинуть число 10 на два разряда влево, то получится 1000, что в десятичной системе равно число 8.

Число 16 в двоичном представлении 10000. Если сдвинуть число 10 на три разряда вправо (три последних разряда отбрасываются), то получится 10, что в десятичной системе представляет число 2.

Поразрядные операции

Поразрядные операции также проводятся только над соответствующими разрядами целочисленных операндов:

1. &: поразрядная конъюнкция (операция И или поразрядное умножение). Возвращает 1, если оба из соответствующих разрядов обоих чисел равны 1
2. |: поразрядная дизъюнкция (операция ИЛИ или поразрядное сложение). Возвращает 1, если хотя бы один из соответствующих разрядов обоих чисел равен 1
3. ^: поразрядное исключающее ИЛИ. Возвращает 1, если только один из соответствующих разрядов обоих чисел равен 1
4. ~: поразрядное отрицание или инверсия. Инвертирует все разряды операнда. Если разряд равен 1, то он становится равен 0, а если он равен 0, то он получает значение 1.

Применение операций:

```
int a = 5 | 2;          // 101 | 010 = 111  - 7
int b = 6 & 2;          // 110 & 010 = 10  - 2
int c = 5 ^ 2;          // 101 ^ 010 = 111 - 7
int d = ~9;             // -10
```

Например, выражение 5 | 2 равно 7. Число 5 в двоичной записи равно 101, а число 2 - 10 или 010. Сложим соответствующие разряды обоих чисел. При сложении если хотя бы один разряд равен 1, то сумма обоих разрядов равна 1. Поэтому получаем:

```
1 0 1
0 1 0
1 1 1
```

В итоге получаем число 111, что в десятичной записи представляет число 7.

Возьмем другое выражение 6 & 2. Число 6 в двоичной записи равно 110, а число 2 – 10 или 010. Умножим соответствующие разряды обоих чисел. Произведение обоих разрядов равно 1, если оба этих разряда равны 1. Иначе произведение равно 0. Поэтому получаем:

```
1 1 0
0 1 0
0 1 0
```

Получаем число 010, что в десятичной системе равно 2.

8. ОПЕРАЦИИ ПРИСВАИВАНИЯ

Операции присваивания позволяют присвоить некоторое значения. Эти операции выполняются над двумя операндами, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную.

Базовая операция присваивания = позволяет присвоить значение правого операнда левому операнду:

```
int x;
x = 2
```

То есть в данном случае переменная x (левый операнд) будет иметь значение 2 (правый операнд).

Стоит отметить, что тип значения правого операнда не всегда может совпадать с типом левого операнда. В этом случае компилятор пытается преобразовать значение правого операнда к типу левого операнда.

При этом операции присваивания имеют правосторонний порядок, то есть выполняются справа налево. И, таким образом, можно выполнять множественное присваивание:

```
int a, b, c;
a = b = c = 34;
```

Здесь сначала вычисляется значение выражения c = 34. Значение правого операнда - 34 присваивается левому операнду c. Далее вычисляется выражение b = c: значение правого операнда c (34) присваивается левому операнду b. И в конце вычисляется выражение a = b: значение правого операнда b (34) присваивается левому операнду a.

Кроме того, следует отметить, что операции присваивания имеют наименьший приоритет по сравнению с другими типами операций, поэтому выполняются в последнюю очередь:

```
int x;
x = 3 + 5;
```

В соответствии с приоритетом операций вначале выполняется выражение 3 + 5, и только потом его значение присваивается переменной x.

Все остальные операции присваивания являются сочетанием простой операции присваивания с другими операциями:

1. +=: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: A += B эквивалентно A = A + B
2. -=: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: A -= B эквивалентно A = A - B
3. *=: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: A *= B эквивалентно A = A * B
4. /=: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: A /= B эквивалентно A = A / B
5. %=: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: A %= B эквивалентно A = A % B
6. <<=: присваивание после сдвига разрядов влево. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: A <<= B эквивалентно A = A << B
7. >>=: присваивание после сдвига разрядов вправо. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: A >>= B эквивалентно A = A >> B
8. &=: присваивание после поразрядной конъюнкции. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: A &= B эквивалентно A = A & B
9. |=: присваивание после поразрядной дизъюнкции. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: A |= B эквивалентно A = A | B
10. ^=: присваивание после операции исключающего ИЛИ. Присваивает левому операнду результат операции исключающего ИЛИ его битового представления с битовым представлением правого операнда: A ^= B эквивалентно A = A ^ B

Примеры операций:

```
int a = 5;
a += 10;        // 15
a -= 3;         // 12
a *= 2;         // 24
a /= 6;         // 4
a <<= 4;      // 64
a >>= 2;      // 16
```

9. ВВОД И ВЫВОД В КОНСОЛИ

По умолчанию язык C++ не содержит встроенных средств для ввода с консоли и вывода на консоль, эти средства предоставляются библиотекой iostream. В ней определены два типа: istream и ostream. istream представляет поток ввода, а ostream – поток вывода.

Вообще сам термин "поток" в данном случае представляет последовательность символов, которая записывается на устройство ввода-вывода или считывается с него. И в данном случае под устройством ввода-вывода рассматривается консоль.

Для записи или вывода символов на консоль применяется объект cout, который представляет тип ostream. А для чтения с консоли используется объект cin

Для использования этих объектов в начало исходного файла необходимо подключить библиотеку iostream:

```
#include <iostream>
```

Вывод на консоль

Для вывода на консоль применяется оператор <<. Этот оператор получает два операнда. Левый операнд представляет объект типа ostream, в данном случае объект cout. А правый операнд – значение, которое надо вывести на консоль.

Так как оператор << возвращает левый операнд – cout, то с помощью цепочки операторов мы можем передать на консоль несколько значений. Например, определим простейшую программу вывода на консоль:

```
#include <iostream>

int main()
{
    int age = 33;
    double weight = 81.23;
    std::cout << "Name: " << "Tom" << "\n";
    std::cout << "Age: " << age << std::endl;
    std::cout << "Weight: " << weight << std::endl;
    return 0;
}
```

Консольный вывод программы:

```
Name: Tom
Age: 33
Weight: 81.23
```

Оператору << передаются различные значения – строки, значения переменных, которые выводятся на консоль.

Строки могут содержать управляющие последовательности, которые интерпретируются определенным образом. Например, последовательность "\n" интерпретируется как перевод на новую строку. Из других управляющих последовательностей также нередко употребляется "\t", которая интерпретируется как табуляция.

Также цепочку операторов << можно завершать значением std::endl, которое вызывает перевод на новую строку и сброс буфера. При выводе в поток данные вначале помещаются в буфер. И сброс буфера гарантирует, что все переданные для вывода на консоль данные немедленно будут выведены на консоль.

Ввод с консоли

Для считывания с консоли данных применяется оператор ввода >>, который принимает два операнда. Левый операнд представляет объект типа istream (в данном случае объект cin), с которого производится считывание, а правый операнд - объект, в который считываются данные.

Например, считаем данные с консоли:

```
#include <iostream>

int main()
{
    int age;
```

```
    double weight;
    std::cout << "Input age: ";
    std::cin >> age;
    std::cout << "Input weight: ";
    std::cin >> weight;
    std::cout << "Your age: " << age << "\t your weight: " << weight << std::endl;
    return 0;
}
```

Здесь после приглашений к вводу программа ожидает ввода значений для переменных age и weight.

Пример работы программы:

```
Input age: 32
Input weight: 67.45
Your age: 32  your weight: 67.45
```

Стоит отметить, что так оператор ввода в первом случае будет добавлять данные в целочисленную переменную age, то он ожидает ввода числа. В случае с переменной weight оператор ввода ожидает дробное число, причем разделителем целой и дробной части должна быть точка. Поэтому мы не можем ввести любые значения, например, строки. В этом случае программа может выдать некорректный результат.

Оператор ввода >> возвращает левый операнд - объект cin, поэтому мы можем по цепочке считывать данные в различные переменные:

```
#include <iostream>

int main()
{
    int age;
    double weight;
    std::cout << "Input age: ";
    std::cin >> age >> weight;
    std::cout << "Your age: " << age << "\t your weight: " << weight << std::endl;
    return 0;
}
```

Пример работы программы:

```
Input age: 32 67.45
Your age: 32  your weight: 67.45
```

После ввода одного из значений надо будет ввести пробел и затем вводить следующее значение.

10. УСЛОВНЫЕ КОНСТРУКЦИИ

Условные конструкции направляют ход программы по одному из возможных путей в зависимости от условия.

Конструкция if

Конструкция if проверяет истинность условия, и если оно истинно, выполняет блок инструкций. Этот оператор имеет следующую сокращенную форму:

```
if (условие)
{
    инструкции;
}
```

В качестве условия использоваться условное выражение, которое возвращает true или false. Если условие возвращает true, то выполняются последующие инструкции, которые входят в блок if. Если условие возвращает false, то последующие инструкции не выполняются. Блок инструкций заключается в фигурные скобки.

Например:

```
#include <iostream>

int main()
{
    int x = 60;

    if(x > 50)
    {
        std::cout << "x is greater than 50 \n";
    }

    if(x < 30)
    {
        std::cout << "x is less than 30 \n";
    }

    std::cout << "End of Program" << "\n";
    return 0;
}
```

Здесь определены две условных конструкции if. Они проверят больше или меньше значение переменной x, чем определенное значение. В качестве инструкции в обоих случаях выполняется вывод некоторой строки на консоль.

В первом случае x > 50 условие истинно, так как значение переменной x действительно больше 50, поэтому это условие возвратит true, и, следовательно, будут выполнятся инструкции, которые входят в блок if.

Во втором случае операция отношения x < 30 возвратит false, так как условие ложно, поэтому последующий блок инструкций выполняться не будет. В итоге при запуске программы вывод консоли будет выглядеть следующим образом:

```
x greater than 50
End of Program
```

Также мы можем использовать полную форму конструкции if, которая включает оператор else:

```
if(выражение_условия)
    инструкция_1
else
    инструкция_2
```

После оператора else мы можем определить набор инструкций, которые выполняются, если условие в операторе if возвращает false. То есть если условие истинно, выполняются инструкции после оператора if, а если это выражение ложно, то выполняются инструкции после оператора else.

```
int x = 50;
if(x > 60)
    std::cout << "x is greater than 60 \n";
else
    std::cout << "x is less or equal 60 \n";
```

В данном случае условие x > 60 ложно, то есть возвращает false, поэтому будет выполняться блок else. И в итоге на консоль будет выведена строка "x is less or equal 60 \n".

Однако нередко надо обработать не два возможных альтернативных варианта, а гораздо больше. Например, в случае выше можно насчитать три условия: переменная x может быть больше 60, меньше 60 и равна 60. Для проверки альтернативных условий мы можем вводить выражения else if:

```
int x = 60;

if(x > 60)
{
    std::cout << "x is greater than 60 \n";
}
else if (x < 60)
{
    std::cout << "x is less than 60 \n";
}
else
{
    std::cout << "x is equal 60 \n";
}
```

То есть в данном случае мы получаем три ветки развития событий в программе.

Если в блоке if или else или else-if необходимо выполнить только одну инструкцию, то фигурные скобки можно опустить:

```
int x = 60;

if(x > 60)
    std::cout << "x is greater than 60 \n";
else if (x < 60)
    std::cout << "x is less than 60 \n";
else
    std::cout << "x is equal 60 \n";
```

Конструкция switch

Другую форму организации ветвления программ представляет конструкция switch...case. Она имеет следующую форму:

```
switch(выражение)
{
    case константа_1: инструкции_1;
    case константа_2: инструкции_2;

    default: инструкции;
}
```

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case.

В конце конструкции switch может стоять блок default. Он необязателен и выполняется в том случае, если значение после switch не соответствует ни одному из операторов case. Например:

```
#include <iostream>
```

```
int main()
{
    int x = 2;

    switch(x)
    {
        case 1:
            std::cout << "x = 1" << "\n";
            break;
        case 2:
            std::cout << "x = 2" << "\n";
            break;
        case 3:
            std::cout << "x = 3" << "\n";
            break;
        default:
            std::cout << "x is undefined" << "\n";
            break;
    }
    return 0;
}
```

Чтобы избежать выполнения последующих блоков case/default, в конце каждого блока ставится оператор break. То есть в данном случае будет выполняться оператор

```
case 2:
    std::cout << "x = 2" << "\n";
    break;
```

После выполнения оператора break произойдет выход из конструкции switch..case, и остальные операторы case будут проигнорированы. Поэтому на консоль будет выведена следующая строка

```
x = 2
```

Стоит отметить важность использования оператора break. Если мы его не укажем в блоке case, то после этого блока выполнение перейдет к следующему блоку case. Например, уберем из предыдущего примера все операторы break:

```
#include <iostream>

int main()
{
    int x = 2;

    switch(x)
    {
        case 1:
            std::cout << "x = 1" << "\n";
        case 2:
            std::cout << "x = 2" << "\n";
        case 3:
            std::cout << "x = 3" << "\n";
        default:
            std::cout << "x is undefined" << "\n";
    }
    return 0;
}
```

В этом случае опять же будет выполняться оператор case 2:, так как переменная x=2. Однако так как этот блок case не завершается оператором break, то после его

завершения будет выполняться набор инструкций после case 3: даже несмотря на то, что переменная x по прежнему равна 2. В итоге мы получим следующий консольный вывод:

```
x = 2
x = 3
x is undefined
```

Тернарный оператор

Тернарный оператор ?: позволяет сократить определение простейших условных конструкций if и имеет следующую форму:

```
[первый операнд - условие] ? [второй операнд] : [третий операнд]
```

Оператор использует сразу три операнда. В зависимости от условия тернарный оператор возвращает второй или третий операнд: если условие равно true (то есть истинно), то возвращается второй операнд; если условие равно false (то есть ложно), то третий. Например:

```
#include <iostream>

int main()
{
    setlocale(LC_ALL, "");
    int x = 5;
    int y = 3;
    char sign;
    std::cout << "Введите знак операции: ";
    std::cin >> sign;
    int result = sign=='+'?x + y:x - y;
    std::cout << "Результат: " << result << "\n";
    return 0;
}
```

В данном случае производится ввод знака операции. Здесь результатом тернарной операции является переменная result. И если переменная sign содержит знак "+", то result будет равно второму операнду – (x+y). Иначе result будет равно третьему операнду.

11. ЦИКЛЫ

Для выполнения некоторых действий множество раз в зависимости от определенного условия используются циклы. В языке C++ имеются следующие виды циклов:

1) for
2) while
3) do...while

Цикл while

Цикл while выполняет некоторый код, пока его условие истинно, то есть возвращает true. Он имеет следующее формальное определение:

```
while(условие)
{
    // выполняемые действия
}
```

После ключевого слова while в скобках идет условное выражение, которое возвращает true или false. Затем в фигурных скобках идет набор инструкций, которые

составляют тело цикла. И пока условие возвращает true, будут выполняться инструкции в теле цикла.

Например, выведем квадраты чисел от 1 до 9:

```
#include <iostream>

int main()
{
    int i = 1;
    while(i < 10)
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
        i++;
    }

    return 0;
}
```

Здесь пока условие i < 10 истинно, будет выполняться цикл while, в котором выводится на консоль квадрат числа и инкрементируется переменная i. В какой-то момент переменная i увеличится до 10, условие i < 10 возвратит false, и цикл завершится.

Консольный вывод программы:

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
```

Каждый отдельный проход цикла называется итерацией. То есть в примере выше было 9 итераций.

Если цикл содержит одну инструкцию, то фигурные скобки можно опустить:

```
int i = 0;
while(++i < 10)
    std::cout << i << " * " << i << " = " << i * i << std::endl;
```

Цикл for

Цикл for имеет следующее формальное определение:

```
for (выражение_1; выражение_2; выражение_3)
{
    // тело цикла
}
```

выражение_1 выполняется один раз при начале выполнения цикла и представляет установку начальных условий, как правило, это инициализация счетчиков – специальных переменных, которые используются для контроля за циклом.

выражение_2 представляет условие, при соблюдении которого выполняется цикл. Как правило, в качестве условия используется операция сравнения, и если она возвращает ненулевое значение (то есть условие истинно), то выполняется тело цикла, а затем вычисляется выражение_3.

выражение_3 задает изменение параметров цикла, нередко здесь происходит увеличение счетчиков цикла на единицу.

Например, перепишем программу по выводу квадратов чисел с помощью цикла for:

```
#include <iostream>

int main()
{
    for(int i =1; i < 10; i++)
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
    }

    return 0;
}
```

Первая часть объявления цикла – int i = 1 – создает и инициализирует счетчик i. Фактически это то же самое, что и объявление и инициализация переменной. Счетчик необязательно должен представлять тип int. Это может быть и другой числовой тип, например, float. И перед выполнением цикла его значение будет равно 1.

Вторая часть – условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока переменная i не станет равна 10.

И третья часть – приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--. Можно изменять на другое значение: i+=2.

В итоге блок цикла сработает 9 раз, пока переменная i не станет равна 10. И каждый раз это значение будет увеличиваться на 1. И по сути мы получим тот же самый результат, что и в случае с циклом while:

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
```

Необязательно указывать все три выражения в определении цикла, мы можем одно или даже все из них опустить:

```
int i = 1;
for(; i < 10;)
{
    std::cout << i << " * " << i << " = " << i * i << std::endl;
    i++;
}
```

Формально определение цикла осталось тем же, только теперь первое и третье выражения в определении цикла отсутствуют: for (; i < 10;). Переменная-счетчик определена и инициализирована вне цикла, а ее приращение происходит в самом цикле.

Можно определять вложенные циклы. Например, выведем таблицу умножения:

```
#include <iostream>

int main()
{
    for (int i=1; i < 10; i++)
    {
```

```
        for(int j = 1; j < 10; j++)
        {
            std::cout << i * j << "\t";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Цикл do

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, то есть не равно 0, то цикл повторяется. Формальное определение цикла:

```
do
{
    инструкции
}
while(условие);
```

Например:

```
#include <iostream>

int main()
{
    int i = 6;
    do
    {
        std::cout << i << std::endl;
        i--;
    }
    while(i>0);

    return 0;
}
```

Здесь код цикла сработает 6 раз, пока i не станет равным нулю.

Но важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно. То есть мы можем написать:

```
int i = -1;
do
{
    std::cout << i << std::endl;
    i--;
}
while(i>0);
}
```

Хотя у нас переменная i меньше 0, цикл все равно один раз выполнится.

Операторы continue и break

Иногда возникает необходимость выйти из цикла до его завершения. В этом случае можно воспользоваться оператором break. Например:

```
#include <iostream>

int main()
```

```
{
    int i = 1;
    for ( ; ; )
    {
        std::cout << i << " * " << i << " = " << i * i << std::endl;
        i++;
        if (i > 9) break;
    }
    return 0;
}
```

Здесь когда значение переменной i достигнет 10, осуществляется выход из цикла с помощью оператора break.

В отличие от оператора break, оператор continue производит переход к следующей итерации. Например, нам надо посчитать сумму только нечетных чисел из некоторого диапазона:

```
#include <iostream>
int main()
{
    int result = 0;
    for (int i=0; i<10; i++)
    {
        if (i % 2 == 0) continue;
        result +=i;
    }
    std::cout << "result = " << result << std::endl; // 25
    return 0;
}
```

Чтобы узнать, четное ли число, мы получаем остаток от целочисленного деления на 2, и если он равен 0, то с помощью оператора continue переходим к следующей итерации цикла. А если число нечетное, то складываем его с остальными нечетными числами.

12. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Линейные программы

1. Найти массу *x* литров молока, если известно, что плотность молока p = 1030 кг/м^3.
2. Объем цилиндра равен *V*, а площадь основания – *S*. Какова высота цилиндра *H*?
3. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей. Извне вводятся цена одной тетради *Ct* и количество тетрадей *Kt*, а также цена карандаша *Ck* и количество карандашей *Kk*.
4. Найти площадь равнобочной трапеции с основаниями *a* и *b* и углом при большем основании равным *x*.
5. Составить программу, которая поменяет местами значения введенных переменных *x* и *y*:
   а) используя дополнительную переменную;
   б) не используя дополнительной переменной.
6. Составить программу, которая поменяет местами значения введенных переменных *x, y, z* так, чтобы в переменной *x* оказалось значение переменной *y*, в *y* – значение переменной *z*, а в *z* – прежнее значение переменной *x*:
   а) используя дополнительную переменную;
   б) не используя дополнительной переменной.
7. Составить программу перевода единиц измерения (см. табл.1.1):
   а) килограммов в пуды, фунты, лоты, золотники;

б) метров в версты, сажени, футы, аршины;
в) литров в бочки, ведра, штофы, бутылки, чарки.

Таблица 1.1

Русская система мер

Меры массы (веса)	
1 пуд = 40 фунтов	0,016 т = 0,164 ц = 16,3805 кг
1 фунт = 32 лота = 96 золотников	0,410 кг = 409,512 г
1 лот = 3 золотника	12,7973 г
1 золотник	4,26575 г
Меры длины	
1 миля = 7 верст	7,4676 км
1 верста = 500 саженей	1,0668 км
1 сажень = 3 аршина = 7 футов	2,1336 м
1 аршин = 16 вершков	0,711 м = 71,120 см
1 вершок	4,445 см = 44,45 мм
1 фут = 12 дюймов	0,305 м = 30,48 см
1 дюйм	2,540 см = 25,4 мм
Меры емкости (для жидкости)	
1 бочка = 40 ведер	4, 920 гл (гекталитров)
1 ведро = 10 штофов = 20 бутылок	1,2299 дкл = 12,2994 л
1 штоф = 10 чарок	1,230 л
1 чарка	0,123 л
1 бутылка	0,615 л

Программы с ветвлением

1. Даны числа a, b, c. Проверить, выполняется ли неравенство $a < b < c$. Вывести об этом сообщение.
2. Даны три действительных числа a, b, c. Выбрать из них те, которые принадлежат интервалу (1, 3).
3. Даны числа x, y ($x \neq y$). Меньшее из них заменить полусуммой, а большее – их удвоенным произведением.
4. Выяснить, существует ли треугольник с длинами сторон x, y, z (в треугольнике большая сторона меньше суммы двух других сторон).
5. На окружности с центром в точке (x_0,y_0) задана дуга с координатами начальной (x_n, y_n) и конечной (x_k, y_k) точек. Определить номера четвертей окружности, в которых находятся начальная и конечная точки.
6. Даны действительные числа x_1, x_2, x_3, y_1, y_2, y_3. Выяснить, является ли треугольник с вершинами (x_1, y_1), (x_2, y_2), (x_3, y_3) прямоугольным?
7. Для пятиугольника, заданного координатами своих вершин, найти наибольшую и наименьшую стороны**.**
8. Написать программу вычисления площади кольца. Извне вводятся радиус кольца и радиус отверстия. В программе предусмотреть проверку правильности вводимых данных (радиусы положительны, причем радиус кольца больше радиуса отверстия).
9. Написать программу, которая запрашивает у пользователя номер дня недели и выводит название дня недели.
10. Написать программу, которая запрашивает у пользователя номер месяца и затем выводит соответствующее название времени года. Если вводится недопустимое число (<1 или >12), должно появиться сообщение «ошибка ввода».

Циклические программы

1. Для заданного натурального числа n рассчитать сумму (x вводится извне):

$$\cos x + \cos x^2 + \cos x^3 + \ldots + \cos x^n.$$

2. Для заданного натурального числа n рассчитать сумму и сравнить со значением y:

$$1 + 2 + 3 + \ldots + n; \qquad y = n\,(n + 1)/2.$$

3. Для заданного натурального числа n рассчитать величину $n!!$

$$n!! = \begin{cases} 1 \cdot 3 \cdot 5 \cdot \ldots \cdot n, & \text{если } n - \text{нечетное} \\ 2 \cdot 4 \cdot 6 \cdot \ldots \cdot n, & \text{если } n - \text{четное} \end{cases}$$

4. Написать программу, которая выводит таблицу (2×6) степеней двойки от нулевой до одиннадцатой степени.

5. Вычислить K – количество точек с целочисленными координатами, попадающих в круг радиуса R с центром в начале координат.

6. Пусть дано натуральное число n. Найдите первое число Фибоначчи, больше заданного n.

7. Вычислить для заданного x сумму вида: $x - x^2/2 + x^3/3 - x^4/4 + \ldots$ с заданной точностью E (когда очередное слагаемое по модулю станет меньше E, то суммирование прекратить). Результат сравнить с величиной $y = \ln(x + 1)$, $|x| < 1$.

8. Для заданного X в последовательности вида: $\sin X$, $\sin(\sin X)$), $\sin(\sin(\sin X))$,... найти первое число, меньшее по модулю 0,01.

9. Спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10 % от нормы предыдущего дня. Определить через сколько дней спортсмен:

а) будет пробегать более 20 км;

б) пробежит суммарный путь более 100 км.

Сколько он пробежит за 7 дней?

10. Найти наименьший номер n, для которого выполняется условие $|a_n - a_{n-1}| < 0.1$, если последовательность a_n имеет вид: $a_{n+1} = a_n + 2/a_n$, $a_1 = 1$.

13. ТЕСТОВЫЕ ЗАДАНИЯ ДЛЯ ПРОВЕРКИ ЗНАНИЙ

1. Файлы с текстами программ на языке С++ имеют расширение

1) **.h, *.hpp, *.c* или **.cpp*;
2) **.txt* или **.doc*;
3) **.obj* или **.lib*.

2. Заголовочные файлы (с расширением **.h* или **.hpp*) в языке С++ используются для

1) объявления в них переменных программы;
2) раздельной компиляции модулей программы;
3) хранения массивов данных программы.

3. Заголовочные файлы (с расширением **.h* или **.hpp*) в языке С++ подключаются к компилируемому файлу

1) с помощью директивы *#include*;
2) с помощью директивы *#inpute*;
3) с помощью директивы *#insert*.

4. Точкой входа в программу на языке С++ (из перечисленных) является функция

1) *begin( )*;
2) *start( )*;
3) *main( )*.

5. Один и тот же заголовочный файл (с расширением **.h* или **.hpp*) можно подключать

1) только к одному модулю программы;
2) только к двум модулям программы;
3) к любому количеству модулей программы.

6. Программа на языке С++ начинает свою работу
1) с первой строки первого модуля программы;
2) с функции *main( ) или WinMain( )*;
3) с произвольного места, помеченного программистом директивой *#begin*.

7. В языке С++ встроенный тип данных «*char*» предназначен для хранения
1) целых чисел или символов;
2) вещественных чисел;
3) символов.

8. В языке С++ встроенный тип данных «*int*» предназначен для хранения
1) символов;
2) положительных и отрицательных целых чисел;
3) вещественных чисел.

9. В языке С++ встроенный тип данных «*double*» предназначен для хранения
1) символов;
2) вещественных чисел;
3) целых чисел.

10. Именно такое объявление переменной в языке С++ является НЕ правильным
1) *int x*;
2) *int x = 0*;
3) *x: int*.

11. В языке С++ результатом выполнения операции 4.0 * 5 будет число
1) 20;
2) 20.0;
3) 0.

12. В языке С++ результатом выполнения операции 5 % 2 будет число
1) 1;
2) 2;
3) 3.

13. В результате выполнения программы
```
int x, y;
x = 0;
y = 0;
x = y++;
```
переменная x получит значение
1) 0;
2) 1;
3) 2.

14. В результате выполнения программы
```
int x, y;
x = 10;
```

```
y = 10;
x = --y;
```
переменная y получит значение

1) 0;
2) 9;
3) 10.

15. В результате выполнения программы
```
int x, y;
x = 0;
y = 0;
if (!x)
{
y = 1;
}
```
переменная y получит значение

1) 0;
2) 1;
3) -1.

16. В результате выполнения программы
```
int x, y;
x = 1;
y = 1;
while (x < 1)
{
x = x + 1;
y = y + 1;
}
```
переменная y получит значение

1) 1;
2) 2;
3) 0.

17. В результате выполнения программы
```
int x, y;
x = 1;
y = 1;
do
{
x = x + 1;
y = y + 1;
}
while (x < 2);
```
переменная y получит значение

1) 1;
2) 2;
3) 3.

18. В результате выполнения программы
```
int x, y;
y = 1;
for(x=0; x<3; x++)
{
y=y * 2;
}
```
переменная y получит значение

1) 2;
2) 4;

3) 6.

19. В языке C++ существует специальный оператор прерывания циклов *continue*. Он служит для того, чтобы

1) досрочно прекратить выполнение содержащего его ближайшего цикла *while, do ... while* или *for или* условного оператора *switch*;

2) досрочно прекратить выполнение текущей итерации содержащего его ближайшего цикла *while, do ... while* или *for*;

3) досрочно завершить программу.

20. Основное отличие операторов прерывания циклов *break* и *continue* состоит в том, что

1) оператор *break* прерывает выполнение содержащего его цикла, оператор *continue* только текущей итерации содержащего его цикла;

2) оператор *continue* прерывает выполнение содержащего его цикла, оператор *break* только текущей итерации содержащего его цикла;

3) оператор *break* может использоваться в циклах *for*, оператор *continue* не может.

ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. Девис, С. Р. C++ для «чайников» / С. Р. Девис. – 4-е изд.: пер. с англ. – Москва : ИД «Вильямс», 2003. – 336 с. – ISBN 5-8459-0160-X. – Текст : непосредственный.

2. Корчагин, Ю. Э. Программирование на языках С и С++: лабораторный практикум : учебное пособие / Ю. Э. Корчагин. – Воронеж : ГОУВПО «Воронежский государственный технический университет», 2009. – 251 с. – Текст : непосредственный.

3. Культин, Н. Б. С/С++ в задачах и примерах / Н. Б. Культин. – 2-е изд., перераб. и доп. – Санкт-Петербург : БХВ-Петербург, 2009. – 368 с. – ISBN 978-5-94157-406-3. – Текст : непосредственный.

4. Павловская, Т. А. С/С++. Программирование на языке высокого уровня : учебник / Т. А. Павловская. – Санкт-Петербург : Питер, 2003. – 461 с. – ISBN 5-94723-568-4. – Текст : непосредственный.

5. Столяров, А. В. Введение в язык Си++ : учебное пособие / А. В. Столяров. – 4-е изд., испр. и доп. – Москва : МАКС Пресс, 2018. – 136 с. – ISBN 978-5-317-05781-7. – Текст : непосредственный.

6. Рейзлин, В. И. Язык С++ и программирование на нём : учебное пособие / В. И. Рейзлин. – 2-е изд., перераб. – Томск : Изд-во Томского политехнического университета, 2015. – 212 с. – Текст : непосредственный.

7. Руководство по языку программирования C++ : сайт. – URL: https://metanit.com/cpp/tutorial/ – (дата обращения 01.11.2019). – Текст : электронный.

Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины

а) основная учебная литература:

1. Калобухова, Г. В. Компьютерный практикум по информатике. Офисные технологии: учеб. пособие / Г.В. Калобухова, В.М. Титов. – Москва : ИД «ФОРУМ»: ИНФРА-М, 2013. – 336 с. – ISBN 978-5-8199-0321-6. – URL: http://znanium.com/bookread.php?book=392417. – (дата обращения 01.10.2019). – Текст: электронный.

2. Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев; Под ред. Л.Г. Гагариной. – Москва : ИД ФОРУМ: ИНФРА-М, 2012. – 512 с. – ISBN 978-5-8199-0492-3. – URL: http://znanium.com/bookread2.php?book=244875. – (дата обращения 01.10.2019). – Текст: электронный.

б) дополнительная учебная литература:

1. Истомин, Е. П. Высокоуровневые методы информатики и программирования : учебник для вузов / Е. П. Истомин, В. В. Новиков, М. В. Новикова. – Санкт-Петербург : Андреевский издательский дом, 2008. – 228 с. – ISBN 978-5-902894-21-6 : 355-00. – Текст: непосредственный.