

ОСНОВЫ ЯЗЫКА C++

В наше время, кажется, нет такой отрасли знаний, которая бы так стремительно развивалась, как программирование и вычислительная техника. Никакая еще наука не развивалась такими семимильными шагами и такими темпами. Возникает новая техника: компьютеры, процессоры, дисководы. Появляются новые возможности и новые информационные технологии.

Программирование сейчас везде и всюду. Оно обслуживает предприятия, конторы, учебные заведения – все, где есть управленческий труд и потоки информации. Нелегко труд программиста. Трудны языки программирования. Особенно поражает их многообразие. И сам процесс программирования становится таким объемным и сложным, что старые методы уже никого не удовлетворяют, и на смену им приходят новые методы и новые языки программирования, подобные языку C++, способные ускорить во много раз разработку и сопровождение программ. Сегодня мы смотрим назад из XXI-ого века в XX-й век и восхищаемся новейшими Windows-технологиями, визуальным подходом и объектно-ориентированным программированием. За короткий срок они покорили и завоевали весь мир.

Немаловажную роль здесь играет язык программирования C++. Но зачем он был нужен, как и почему возник и был востребован?

C++ – расширение языка C – был разработан сотрудником научно-исследовательского центра AT&T Bell Laboratories (Нью-Джерси, США) Бьерном Струстромом в 1979 году. C++ содержит в себе все, что есть в C. Но, кроме того, он поддерживает объектно-ориентированное программирование (ООП, Object Oriented Programming, OOP). Изначально C++ был создан для того, чтобы облегчить разработку больших программ. ООП – это новый подход к созданию программ. [11]

В 60-е годы XX века особо остро возникла потребность создавать большие и сложные программы. Однако, она натолкнулась на ряд трудностей. Люди, связанные с разработкой программ, начали понимать, что создание сложных программ – это гораздо более сложная задача, чем они себе представляли. Проведенные в этот период исследования привели к появлению и интенсивному развитию структурного программирования. Этот подход отличался большей дисциплинированностью, ясностью и простотой тестирования и отладки программ, легкостью их модификации.

Создание в 1971 году Никлаусом Виртом (швейцарским математиком) языка Паскаль было одним из замечательных результатов проводившихся исследований в ученой университетской среде. Созданный первоначально исключительно для изучения структурного программирования в академической среде, он стал наиболее предпочитаемым языком во многих университетах мира. Однако, отсутствие в нем необходимых свойств для решения коммерческих задач сдерживало его применение в коммерции, в промышленности и управлении.

В течение 70-х и в начале 80-х годов при огромной заинтересованности и поддержке Министерства Обороны США был создан язык программирования Ада. Министерством Обороны США использовались сотни отдельных языков. Но все время хотелось иметь один язык, который бы удовлетворял всем интересам этого ведомства. Таким языком был выбран Паскаль. Но в итоге разработки язык Ада оказался совсем не похожим на Паскаль. Наиболее важное свойство Ады – многозадачность. Оно позволяет программистам разрабатывать алгоритмы параллельного выполнения действий.

Язык C++ является универсальным языком программирования, основные характеристики которого следующие:

- является развитием (надмножеством) языка C;
- поддерживает абстракцию данных;
- поддерживает объектно-ориентированное программирование;
- поддерживает обобщенное программирование.

4.1. Типичная среда C++ программирования

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

Как правило, чтобы выполнить программу на C++, необходимо пройти через 6 этапов: редактирование, препроцессорную (т.е. предварительную) обработку, компиляцию, компоновку, загрузку и выполнение.

Первый этап представляет создание и редактирование файла с исходным текстом программы. Он может выполняться с помощью простейшего редактора текстов программ. Программист набирает в этом редакторе свою C++ программу. При необходимости он снова обращается к ней и вносит с помощью этого редактора изменения в исходный текст программы. Далее программа запоминается на диске. Имена файлов C/C++ программ оканчиваются на «с» или «сpp».

На втором этапе компилятор начинает препроцессорную обработку текста программы прежде, чем ее компилировать. Компилятор. Что он делает? Он переводит программу в машинный код, т.е. это объектный код программы.

Следует знать, что в системе C++ программирования перед началом этапа самой трансляции всегда выполняется программа предварительной обработки. Что она делает? Она отыскивает так называемые «директивы трансляции» или «директивы препроцессора», которые указывают, какие нужно выполнить преобразования перед трансляцией исходного текста программы. Обычно это включение других текстовых файлов в файл, который подлежит компиляции. Препроцессорная обработка инициируется компилятором перед тем, как программа будет преобразована в машинный код. Это позволяет забирать нужные программы-функции в текст компилируемой программы до начала процесса компоновки.

Третий этап - это компиляция. Как правило, программы на языке C++ содержат ссылки на различные функции, которые определены вне самой программы. Например, в стандартных библиотеках или в личных библиотеках программистов. Объектный код, созданный компилятором, содержит «дыры» на месте этих отсутствующих частей.

Четвертый этап – компоновка. Компоновщик связывает объектный код с кодами отсутствующих функций и создает, таким образом, исполняемый загрузочный модуль (без пропущенных «дыр»).

Пятый этап – загрузка. Перед выполнением программа должна быть размещена в памяти. Это делается с помощью загрузчика, который забирает загрузочный модуль программы с диска и перемещает его в память.

Шестой этап – это выполнение. Программа редко заработает с первой попытки. Каждый из названных этапов может заканчиваться ошибкой или неудачей из-за ошибки. Тогда программист должен вернуться к редактированию исходного текста программы. Он должен внести необходимые изменения в текст программы, предварительно его хорошо проанализировав. Затем снова пройти через все этапы работы с исходным текстом программы до получения работающего без ошибок загрузочного модуля.

4.2. Структура программы на C++

Программа на языке C имеет следующую структуру [11]:

```
#директивы препроцессора
.....
#директивы препроцессора
функция a()
{ тело функции a }

...
функция b()
{тело функции b }

void main ( )      //функция, с которой начинается выполнение программы
{последовательность определений, описаний и исполняемых операторов}
```

Директивы препроцессора – управляют преобразованием текста программы до ее компиляции. Исходная программа, подготовленная на C++ в виде текстового файла, проходит 3 этапа обработки (рис. 1):

- 1) препроцессорное преобразование текста;
- 2) компиляция;
- 3) компоновка (редактирование связей или сборка).

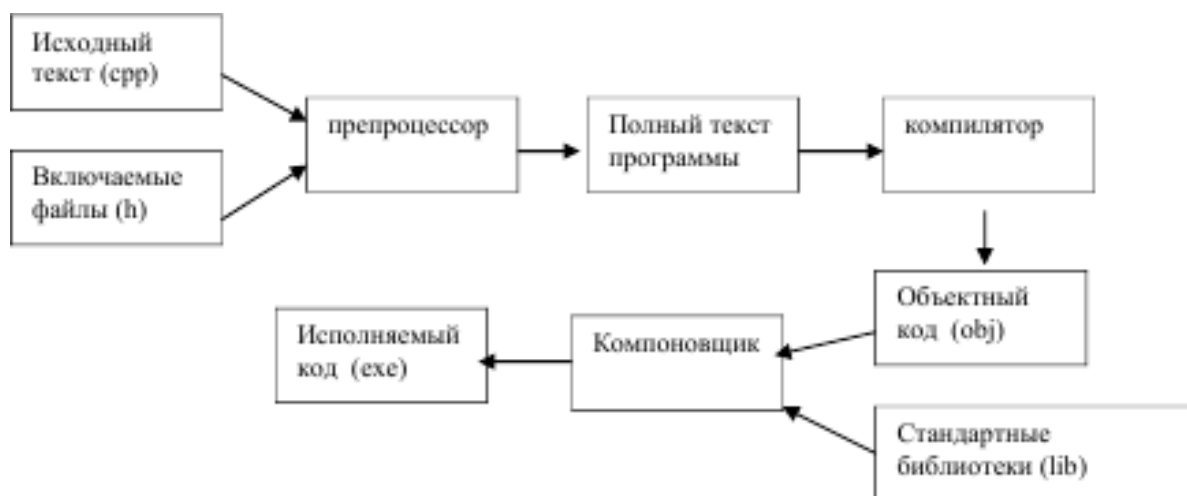


Рис. 1. Этапы создания исполняемого кода

После этих трех этапов формируется исполняемый код программы. Задача препроцессора – преобразование текста программы до ее компиляции. Правила препроцессорной обработки определяет программист с помощью директив препроцессора.

Директива начинается с #.

#define – указывает правила замены в тексте. Например, `#define ZERO 0.0` означает, что каждое использование в программе имени `ZERO` будет заменяться на `0.0`.

Директива **#include < имя заголовочного файла >** – предназначена для включения в текст программы текста из каталога «Заголовочных файлов», поставляемых вместе со стандартными библиотеками. Каждая библиотечная функция C++ имеет соответствующее описание в одном из заголовочных файлов. Список заголовочных файлов определен стандартом языка. Употребление директивы *include* не подключает соответствующую стандартную библиотеку, а только позволяют вставить в текст программы описания из указанного заголовочного файла. Подключение кодов библиотеки осуществляется на этапе компоновки, т.е. после компиляции. Хотя в заголовочных файлах содержатся все описания стандартных функций, в код программы включаются только те функции, которые используются в программе.

После выполнения препроцессорной обработки в тексте программы не остается ни одной препроцессорной директивы.

Программа представляет собой набор описаний и определений, и состоит из набора функций. Среди этих функций всегда должна быть функция с именем **main**. Без нее программа не может быть выполнена. Это точка входа в программу.

Перед именем функции помещаются сведения о типе возвращаемого функцией значения (тип результата): **double sqrt (double arg)**

Если функция ничего не возвращает, то указывается тип **void**: **void main ();**

Каждая функция, в том числе и **main** должна иметь набор параметров, он может быть пустым, тогда в скобках указывается (**void**).

Если функция не объявлена как **void**, она должна возвращать значение. Возвращаемое значение задается инструкцией **return e;** где **e** – выражение, значение которого возвращается в качестве результата работы функции.

Пример 1. Определение функции извлечения квадратного корня.

```
double sqrt (double arg)    // вещественный аргумент, и возвращается
// вещественное число.
{
    //код, вычисляющий квадратный корень
}
Использование (вызов) этой функции:
int main()
{
    double root2 = sqrt(2);
    ...
    return 0;    // возврат значения из функции
}
```

За заголовком функции размещается *тело функции*. Тело функции – это последовательность определений, описаний и исполняемых операторов, заключенных в фигурные скобки. Каждое определение, описание или оператор заканчивается точкой с запятой.

Определения – вводят объекты (**объект** – это именованная область памяти, частный случай объекта - переменная), необходимые для представления в программе обрабатываемых данных.

Пример 2.

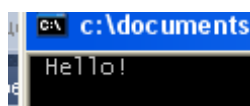
```
float x; //переменная
int y=10; //инициализированная переменная
```

Описания – уведомляют компилятор о свойствах и именах объектов и функций, описанных в программе.

Операторы – определяют действия программы на каждом шаге ее исполнения.

Пример 3. Простейшая программа на языке C++

```
#include <iostream>    //препроцессорная директива
using namespace std;
void main()            //функция
{                      //начало
cout << " Hello!\n";   //печать
}
```



В программах на C++ используется два способа комментариев: первый способ для многострочных комментариев (начинается с комбинации символов косой черты и звёздочки (/*), и заканчивается обратной комбинацией этих же символов (*/). Не может быть вложенным.), а второй – для коротких замечаний (начинается с двух символов косой черты (//) и заканчивается концом строки).

Пример 4. Использование комментариев

```
#include <iostream>
#include <windows.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    int num; /* это многострочный комментарий C++
              ввод числа */
    cout << "Введите проверяемое число: ";
    cin >> num;
    // проверка на чётность. Это однострочный комментарий C++
    if((num%2)==0) cout << "Число чётное.\n";
    else cout << "Число нечётное.\n";
    return 0;
} //конец
```

4.3. Базовые средства языка C++

4.3.1. Состав языка C++

В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания и предложения.

Алгоритмический язык также содержит такие элементы, только **слова называют лексемами (элементарными конструкциями), словосочетания – выражениями, предложения – операторами**. Лексемы образуются из символов, выражения из лексем и символов, операторы из символов выражений и лексем (рис. 2).



Рис. 2. Состав алгоритмического языка

Таким образом, **элементами алгоритмического языка являются:**

1. **Алфавит языка C++, который включает**

- прописные и строчные латинские буквы и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки “ { } , [] () + - / % * . \ ' : ; & ? < > = ! # ^
- пробельные символы (пробел, символ табуляции, символы перехода на новую строку).

2. Из символов формируются лексемы языка:

Идентификаторы – имена объектов С-программ. В идентификаторе могут быть использованы латинские буквы, цифры и знак подчеркивания. Первым символом не может быть цифра. Прописные и строчные буквы различаются, (PROG1, prog1 и Prog1 – три различных идентификатора). Пробелы не допускаются.

_PROG1, prog_1, P1ro2g1 - правильные идентификаторы.

3PROG1, prog_*, №5_Prog1 – ошибка!

Ключевые (зарезервированные) слова – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов.

Знаки операций – это один или несколько символов, определяющих действие над операндами. Операции делятся на унарные (применяется к одному операнду, например: ++a), бинарные (применяется к двум операндам, например: a+b) и тернарные (применяется к трем операндам, например: условие ? действие A : действие B).

Константы – это неизменяемые величины. Существуют целые, вещественные, символьные и строковые константы. Компилятор выделяет константу в качестве лексемы (элементарной конструкции) и относит ее к одному из типов по ее внешнему виду.

Разделители – скобки, точка, запятая, пробельные символы.

4.3.1.1. Константы в C++

Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения.

Константы делятся на пять групп:

- 1) целые;
- 2) вещественные (с плавающей точкой);
- 3) перечислимые;
- 4) символьные;
- 5) строковые.

Компилятор выделяет лексему и относит ее к той или другой группе, а затем внутри группы к определенному типу по ее форме записи в тексте программы и по числовому значению.

1) Целые константы могут быть десятичными, восьмеричными и шестнадцатеричными.

Десятичная константа определяется как последовательность десятичных цифр, начинающаяся не с 0, если это число не 0 (примеры: 8, 0, 192345).

Восьмеричная константа – это константа, которая всегда начинается с 0. За 0 следуют восьмеричные цифры (примеры: 016 – десятичное значение 14).

Шестнадцатеричные константы – последовательность шестнадцатеричных цифр, которым предшествуют символы 0x или 0X (примеры: 0xA, 0X00F).

В зависимости от значения целой константы компилятор по-разному представит ее в памяти компьютера (т.е. компилятор припишет константе соответствующий тип данных).

2) **Вещественные константы** имеют другую форму внутреннего представления в памяти компьютера. Компилятор распознает такие константы по их виду.

Вещественные константы могут иметь две формы представления:

- с фиксированной точкой:
[цифры].[цифры] (примеры: 5.7, .0001, 41.).
- с плавающей точкой:
[цифры][.][цифры]E|e[+|-][цифры] (примеры: 0.5e5, .11e-5, 5E3).

В записи вещественных констант может опускаться либо целая, либо дробная части, либо десятичная точка, либо признак экспоненты с показателем степени.

3) **Перечислимые константы** вводятся с помощью ключевого слова *enum*.

Это обычные целые константы, которым приписаны уникальные и удобные для использования обозначения.

Пример 5. Перечислимые константы.

```
enum { one=1, two=2, three=3, four=4 };  
enum { zero, one, two, three }
```

Если в определении перечислимых констант опустить знаки = и числовые значения, то значения будут приписываться по умолчанию. При этом самый левый идентификатор получит значение 0, а каждый последующий будет увеличиваться на 1.

```
enum { ten=10, three=3, four, five, six };  
enum { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
```

4) **Символьные константы** – это один или два символа, заключенные в апострофы.

Символьные константы, состоящие из одного символа, имеют тип **char** и занимают в памяти *один* байт, символьные константы, состоящие из двух символов, имеют тип **int** и занимают *два* байта.

Последовательности, начинающиеся со знака \, называются управляющими, они используются:

- для представления символов, не имеющих графического отображения, например:

- \a – звуковой сигнал,
 - \b – возврат на один шаг,
 - \n – перевод строки,
 - \t – горизонтальная табуляция.

- для представления символов: \, ', ? , " (\\, \', \?, \").
- для представления символов с помощью шестнадцатеричных или восьмеричных кодов (\073, \0xF5).

Строковая константа – это последовательность символов, заключенная в кавычки. Внутри строк также могут использоваться управляющие символы.

Пример 6. Строковые константы.

```
cout<< "\nНОВАЯ СТРОКА";  
cout<< "\n\"АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ \";
```

```
НОВАЯ СТРОКА  
"АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ " _
```

4.3.2. Типы данных в C++

Данные отображают в программе окружающий мир. Цель программы состоит в обработке данных. Данные различных типов хранятся и обрабатываются по-разному. *Тип данных определяет:*

- 1) внутреннее представление данных в памяти компьютера;
- 2) множество значений, которые могут принимать величины этого типа;
- 3) операции и функции, которые можно применять к данным этого типа.

В зависимости от требований задания программист выбирает тип для объектов программы. *Типы C++ можно разделить на простые и составные.* К простым типам относят типы, которые характеризуются одним значением.

В C++ определены следующие простые типы данных:

int (целый)
char (символьный)
bool (логический)
float (вещественный)
double (вещественный с двойной точностью)

Логические, символьные и целые типы все вместе обобщённо называются *целыми (строго говоря, интегральными) типами*. Целые типы совместно с типами с плавающей точкой называются *арифметическими типами*.

Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон стандартных типов:

- 1) **short** (короткий)
- 2) **long** (длинный)
- 3) **signed** (знаковый)
- 4) **unsigned** (беззнаковый)

Тип int

Значениями этого типа являются целые числа.

Размер типа **int** не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора под него отводится 2 байта, для 32-разрядного – 4 байта.

Если перед **int** стоит спецификатор **short**, то под число отводится 2 байта, а если спецификатор **long**, то 4 байта. От количества отводимой под объект памяти зависит множество допустимых значений, которые может принимать объект:

short int – занимает 2 байта, следовательно, имеет диапазон:

–32768 ... +32767;

long int – занимает 4 байта, следовательно, имеет диапазон:

–2 147 483 648 ... +2 147 483 647

Тип **int** совпадает с типом **short int** на 16-разрядных ПК и с типом **long int** на 32-разрядных ПК.

Модификаторы *signed* и *unsigned* также влияют на множество допустимых значений, которые может принимать объект:

unsigned short int – занимает 2 байта, следовательно, имеет диапазон:

0 ... 65536;

unsigned long int – занимает 4 байта, следовательно, имеет диапазон:

0 ... +4 294 967 295.

Тип char

Значениями этого типа являются элементы конечного упорядоченного множества символов. Каждому символу ставится в соответствие число, которое называется кодом

символа. Под величину символьного типа отводится 1 байт. Тип **char** может использоваться со спецификаторами **signed** и **unsigned**. В данных типа **signed char** можно хранить значения в диапазоне от -128 до 127. При использовании типа **unsigned char** значения могут находиться в диапазоне от 0 до 255. Для кодировки используется код ASCII(American Standard Code for International Interchange). Символы с кодами от 0 до 31 относятся к служебным и имеют самостоятельное значение только в операторах ввода-вывода.

Величины типа **char** также применяются для хранения чисел из указанных диапазонов.

Тип bool

Тип **bool** называется логическим. Его величины могут принимать значения *true* и *false*. Внутренняя форма представления *false* – 0, любое другое значение интерпретируется как *true*.

Типы с плавающей точкой

Внутреннее представление вещественного числа состоит из 2 частей: мантиссы и порядка. В IBM-совместимых ПК величины типа **float** занимают 4 байта.

Величины типа **double** занимают 8 байтов.

Если перед именем типа *double* стоит спецификатор *long*, то под величину отводится 10 байтов.

Тип void

К основным типам также относится тип **void**. Множество значений этого типа – пусто.

Переменные

Переменная в C++ – именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение.

Имя служит для обращения к области памяти, в которой хранится значение. Перед использованием любая переменная должна быть описана.

Пример 7. Описание переменных
int a; float x;

Общий вид оператора описания:

[класс памяти][const]тип имя [инициализатор];

Класс памяти может принимать значения: *auto*, *extern*, *static*, *register*. Класс памяти определяет время жизни и область видимости переменной. Если класс памяти не указан явно, то компилятор определяет его исходя из контекста объявления. *Время жизни* может быть постоянным – в течение выполнения программы или временным – в течение блока. *Область видимости* – часть текста программы, из которой допустим обычный доступ к переменной. Обычно область видимости совпадает с областью действия. Кроме того, когда во внутреннем блоке существует переменная с таким же именем.

Const – показывает, что эту переменную нельзя изменять (именованная константа). При описании можно присвоить переменной начальное значение (инициализация).

Классы памяти:

auto – автоматическая локальная переменная. Спецификатор *auto* может быть задан только при определении объектов блока, например, в теле функции. Этим

переменным память выделяется при входе в блок и освобождается при выходе из него. Вне блока такие переменные не существуют.

extern – глобальная переменная, она находится в другом месте программы (в другом файле или далее по тексту). Используется для создания переменных, которые доступны во всех файлах программы.

static – статическая переменная, она существует только в пределах того файла, где определена переменная.

register – аналогичны auto, но память под них выделяется в регистрах процессора. Если такой возможности нет, то переменные обрабатываются как *auto*.

Пример 8.

```
int a;           //глобальная переменная
void main()
{
    int b;       //локальная переменная
    extern int x; //переменная x определена в другом месте
    static int c; //локальная статическая переменная
    a=1;         //присваивание глобальной переменной
    int a;       //локальная переменная a
    a=2;         //присваивание локальной переменной
}
a=3;           //присваивание глобальной переменной
int x=4;       //определение и инициализация x
```

В примере переменная **a** определена вне всех блоков. Областью действия переменной **a** является вся программа, кроме тех строк, где используется локальная переменная **a**. Переменные **b** и **c** – локальные, область их видимости – блок. Время жизни различно: память под **b** выделяется при входе в блок (т.к. по умолчанию класс памяти **auto**), освобождается при выходе из него. Переменная **c** (**static**) существует, пока работает программа.

Если при определении начальное значение переменным не задается явным образом, то компилятор обнуляет глобальные и статические переменные. Автоматические переменные не инициализируются.

Имя переменной должно быть уникальным в своей области действия.

Область видимости переменной (если не указан класс памяти) ограничивается скобками, в которых она объявлена.

Пример 9.

```
float f;         // глобальная переменная – к ней можно обращаться из любого
// места программы
int main()
{
    f = 3.17;
    double d = 2.2;
    int i = 7;
    if (f == 5.0)
    {
        float f2 = f + 10;
    }
    f2 = 14.87;   // ошибка – вышли из области видимости переменной
    d = d + i;
    return 0;
}
```

Обратите внимание: **=** означает оператор присваивания,
 == проверку равенства.

Если имя описано в функции (обычно его называют "**локальным именем**"), то область видимости имени простирается от точки описания до конца блока, в котором появилось это описание. Если имя не находится в описании функции или класса (его обычно называют "**глобальным именем**"), то область видимости простирается от точки описания до конца файла, в котором появилось это описание.

Если только переменная (объект) не объявлена как глобальная или статическая (static), выделение памяти под переменную выполняется каждый раз, когда поток управления достигает этого объявления. Объявления разрешается помещать в любом месте программы, где допустима инструкция.

Пример 10.

```
int x = 0;    // объявление и инициализация глобальной переменной x
void f()
{
    int x = -1; // локальная переменная x скрывает глобальную x
    {
        int x;    // объявление без инициализации новой
                 // переменной скрывает
                 // предыдущую локальную переменную x
        x = 2;    // присваивание второй локальной переменной x
    }
    x = 3;       // присваивание первой локальной переменной x
}
int *p = &x;    // взятие адреса глобальной переменной x
```

Наиболее частой причиной объявления переменной без инициализации является случай, когда для инициализации в программе имеется специальная инструкция, что показано в примере выше, или в эту переменную будет осуществляться ввод.

4.3.3. Знаки операций в C++

Знаки операций обеспечивают формирование *выражений*. Выражения состоят:

- 1) из операндов,
- 2) знаков операций,
- 3) скобок.

Каждый операнд является, в свою очередь, выражением или частным случаем выражения – константой или переменной.

Унарные операции приведены в табл. 1.

Таблица 1

Унарные операции

Операция	Описание
&	Получение адреса операнда
*	Обращение по адресу (разыменование)
-	унарный минус, меняет знак арифметического операнда
~	побитовое инвертирование внутреннего двоичного кода целочисленного операнда (побитовое отрицание)
!	! логическое отрицание (НЕ). В качестве логических значений используется 0 – ложь и не 0 – истина, отрицанием 0 будет 1, отрицанием любого ненулевого числа будет 0.
++	++ Увеличение на единицу: префиксная операция – увеличивает операнд до его использования, постфиксная операция увеличивает операнд после его использования
--	-- уменьшение на единицу: префиксная операция – уменьшает операнд до его использования, постфиксная операция уменьшает операнд после его использования
sizeof	вычисление размера (в байтах) для объекта того типа, который имеет операнд имеет две формы: sizeof выражение; sizeof (тип)

Операторы инкремента (++) и декремента (--) являются унарными операторами присваивания. Они, соответственно, увеличивают или уменьшают значение операнда на единицу. Операнд может быть целого типа или типа с плавающей точкой, или типа указатель и должен быть модифицируемым. В языке имеется **префиксная** (знак оператора стоит перед операндом, изменение операнда происходит до его использования в выражении и результатом операции является увеличенное или уменьшенное значение операнда) и **постфиксная формы** (операнд вначале используется для вычисления выражения, а затем происходит изменение операнда) операторов инкремента и декремента.

Пример 11.

```
int t = 1, s = 2, z, f;
z = (t++) * 5;
```

Вначале происходит умножение $t * 5$, а затем увеличение t . В результате получится: $z = 5, t = 2$.

```
f = (++s) / 3;
```

Вначале значение s увеличивается, а затем используется в операции деления. В результате получим: $s = 3, f = 1$.

В случае, если операторы инкремента и декремента используются как самостоятельные инструкции, префиксная и постфиксная формы записи становятся эквивалентными:

```
z++; // эта инструкция
      // эквивалентна
++z; // этой инструкции
```

Бинарные операции представлены в табл. 2.

Бинарные операции

Операция	Описание
Аддитивные	
+	бинарный плюс (сложение арифметических операндов)
-	бинарный минус (вычитание арифметических операндов)
Мультипликативные	
*	умножение операндов арифметического типа
/	деление операндов арифметического типа (если операнды целочисленные, то выполняется целочисленное деление)
%	получение остатка от деления целочисленных операндов
Операции сдвига (определены только для целочисленных операндов). Формат выражения с операцией сдвига: <i>операнд_левый операция сдвига операнд_правый</i>	
<<	сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого операнда, освободившиеся разряды обнуляются
>>	сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого операнда, освободившиеся разряды обнуляются, если операнд беззнакового типа и заполняются знаковым разрядом, если – знакового
Поразрядные операции	
&	поразрядная конъюнкция (И) битовых представлений значений целочисленных операндов (бит =1, если соответствующие биты обоих операндов=1)
	поразрядная дизъюнкция (ИЛИ) битовых представлений значений целочисленных операндов (бит =1, если соответствующий бит одного из операндов=1)
^	поразрядное исключающее ИЛИ битовых представлений значений целочисленных операндов (бит =1, если соответствующий бит только одного из операндов=1)
Операции сравнения: <i>результатом являются true (не 0) или false (0)</i>	
<	меньше, чем
>	больше, чем
<=	меньше или равно
>=	больше или равно
==	равно
!=	не равно
Логические бинарные операции	
&&	конъюнкция (И) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина (не 0)
	дизъюнкция (ИЛИ) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина (не 0)

Пример 12.

Оператор умножения (*) выполняет умножение операндов:

```
int i = 5;
float f = 0.2;
double g;
g = f * i;
```

Тип результата умножения *f* на *i* преобразуется к типу *double*, затем результат присваивается переменной *g*.

Оператор деления (/) выполняет деление первого операнда на второй. Если две целые величины не делятся нацело, то результатом будет целая часть от деления (дробная часть отбрасывается):

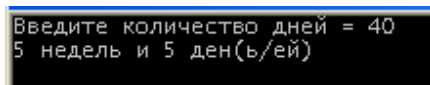
```
int i = 49, j = 10, n, m;  
n = i / j;           // результат  4  
m = i / (-j);        // результат -4
```

Оператор остаток от деления (%) дает остаток от деления первого операнда на второй (только для целых операндов). **Знак результата совпадает со знаком делимого:**

```
int n = 49, m = 10, i, j, k, l;  
i = n % m;           // результат  9  
j = n % (-m);         // результат  9  
k = (-n) % m;         // результат -9  
l = (-n) % (-m);      // результат -9
```

Пример13. Написать программу, которая преобразует введенное пользователем количество дней в количество полных недель и оставшихся дней.

```
#include <iostream> // подключить файл, где объявлены cin и cout  
#include <windows.h> // файл, где определена функция SetConsoleOutputCP(1251);  
int main()  
{  
    using namespace std;           // использовать стандартное пространство имен  
    SetConsoleOutputCP(1251); // функция для вывода текста кириллицей  
    int day, days, week;  
    cout<<"Введите количество дней = ";  
    cin>>days;  
    day=days%7;  
    week=(days-day)/7;  
    cout<<week<<" недель и "<<day<<" ден(ь/ей)";  
    return 0;           // функция возвращает 0  
}
```



Результат деления на 0 в операциях деления и получения остатка не определён и вызывает ошибку (исключение) во время выполнения. Поэтому следующие две инструкции дадут ошибочные, неопределённые результаты:

```
i = n % 0;  
g = f / 0.0;
```

Операции присваивания имеют следующие виды:

=, +=, -=, *= и т.д.

Сокращенные формы арифметических действий, иногда нужно сделать какое либо одно арифметическое действие над одной переменной, прибавить, умножить.

Например:

$S = S + 32$; в сокращенной форме это будет $S += 32$;

$F = F - k$; в сокращенной форме это будет $F -= k$;

$N=N/2$; в сокращенной форме это будет $N/=2$;

Формат операции простого присваивания:

операнд1=операнд2

Леводопустимое значение (L-значение) – выражение, которое адресует некоторый участок памяти, т.е. в него можно занести значение. Это название произошло от операции присваивания, т.к. именно левая часть операции присваивания определяет, в какую область памяти будет занесен результат операции. Переменная – это частный случай леводопустимого выражения.

Условная операция.

В отличие от унарных и бинарных операций в условной операции используется три операнда. Форма представления:

Выражение1 ? Выражение2 : Выражение3;

Первым вычисляется значение *выражения1*. Если оно истинно, то вычисляется значение *выражения2*, которое становится результатом.

Если при вычислении *выражения1* получится 0, то в качестве результата берется значение *выражения3*.

Пример 14. Условная операция

$x < 0 ? -x : x$; //вычисляется абсолютное значение x.

Приведение типов

Нужно для того, чтобы производить операции одновременно над разными типами данных. Существует

- **неявное приведение типов**, т.е. компилятор сам решает к какому типу ему нужно привести данные

Например, сложить переменные типа `int` и `double`. Тип **меняется только на время операции**.

При сложении **`double=int+double`**, `int` приводится к `double` на время операции – это расширяющие приведения.

Если **`int=int+double`**, `double` приводится к `int` и происходит потеря данных, т.к. дробная часть просто отбрасывается. Это сужающие приведения, при этом некоторые данные могут теряться.

Расширяющие `short, bool, char -> int -> long -> float -> double`

Сужающие `double -> float - long -> int -> char, bool, short`

- **явное приведение типов:**

Существует две формы **явного приведения (преобразования) типа**:

- 1) каноническая, общий вид: **(имя_типа) операнд**;
- 2) функциональная, общий вид: **имя_типа (операнд)**.

Пример 15. Операции явного преобразования типа

`(int)a` //каноническая форма

`int(a)` //функциональная форма

4.3.4. Выражения

Из констант, переменных, разделителей и знаков операций можно конструировать выражения. Каждое выражение представляет собой правило вычисления нового значения. Если выражение формирует целое или вещественное число, то оно называется арифметическим. Пара арифметических выражений, объединенная операцией сравнения, называется *отношением*. Если отношение имеет ненулевое значение, то оно – истинно, иначе – ложно. Приоритеты операций в выражениях представлены в табл. 3.

Таблица 3

Приоритеты операций в выражениях	
Ранг	Операции
1	() [] -> .
2	! ~ - ++ -- & * (тип) sizeof тип()
3	* / % (мультипликативные бинарные)
4	+ - (аддитивные бинарные)
5	<< >> (поразрядного сдвига)
6	< > <= >= (отношения)
7	== != (отношения)
8	& (поразрядная конъюнкция «И»)
9	^ (поразрядное исключающее «ИЛИ»)
10	(поразрядная дизъюнкция «ИЛИ»)
11	&& (конъюнкция «И»)
12	(дизъюнкция «ИЛИ»)
13	?: (условная операция)
14	= *= /= %= -= &= ^= = <<= >>= (операция присваивания)
15	, (операция запятой)

4.3.5. Инструкции

Инструкции задают последовательность вычислений в программе.

Все инструкции языка C++, кроме блоков инструкций, заканчиваются точкой с запятой (;).

Одна инструкция может занимать одну или более строк. Две или большее количество инструкций могут быть расположены на одной строке.

В табл. 4 приведена сводка инструкций языка C++:

Таблица 4

Список инструкций языка C++

Тип инструкции	Синтаксис инструкции, примечания
Пустая инструкция	;
Блок	<i>{последовательность инструкций}</i>
Объявление	Объявления переменных, структур, классов, функций
Инструкция-выражение	<i>выражение;</i>
Ветвление по условию if	<i>if (условие)</i> <i>инструкция</i>
Ветвление по условию if / else	<i>if (условие)</i> <i>инструкция 1</i> <i>else</i> <i>инструкция 2</i>
Инструкция выбора switch	<i>switch (выбирающее выражение) {</i> <i>объявления</i> <i>case константа 1:</i> <i>последовательность инструкций1</i> <i>case константа 2:</i> <i>последовательность инструкций2</i> <i>...</i> <i>case константа N: последовательность</i> <i>инструкцийN</i>

	<i>default:</i> <i>последовательность инструкций</i> <i>}</i>
Цикл while	<i>while (условие)</i> <i>инструкция</i>
Цикл do / while	<i>do</i> <i>инструкция</i> <i>while (условие)</i>
Цикл for	<i>for (инициализирующее выражение;</i> <i>условие; модифицирующее выражение)</i> <i>инструкция</i>
Инструкция прекращения break	<i>break;</i>
Инструкция прекращения с продолжением continue	<i>continue;</i>
Инструкция возврата return	<i>return выражение;</i>
Безусловный переход goto	<i>goto идентификатор;</i> <i>...</i> <i>идентификатор: инструкция</i>
Контроль и обработка исключений try / catch	<i>try</i> <i>{</i> <i> последовательность инструкций,</i> <i> генерирующих исключения</i> <i>}</i> <i>catch (mun1 arg)</i> <i>{</i> <i> последовательность инструкций,</i> <i> обрабатывающих исключения</i> <i>mun1</i> <i>}</i> <i>catch (mun2 arg)</i> <i>{</i> <i> последовательность инструкций,</i> <i> обрабатывающих исключения</i> <i>mun2</i> <i>}</i> <i>...</i> <i>catch (munN arg)</i> <i>{</i> <i> последовательность инструкций,</i> <i> обрабатывающих исключения</i> <i>munaN</i> <i>}</i>
Генерация исключения throw	<i>throw expr; //тип выражения</i> <i>//идентифицирует генерируемое исключение</i>

Инструкции, включая инструкции, изменяющие порядок выполнения программы (**if, if / else, switch, while, do / while, for, исключения**), могут вкладываться друг в друга, что даёт возможность строить весьма сложные алгоритмы со многими путями выполнения программы.