# Л Е К Ц И Я 2



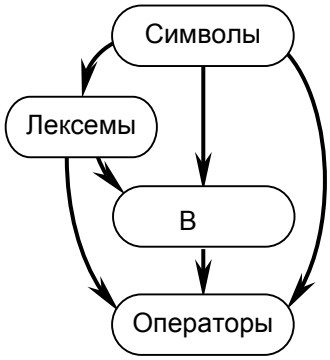
## БАЗОВЫЕ СРЕДСТВА ЯЗЫКА C++

"Язык формирует наш способ мышления и определяет, о чем мы можем мыслить."

*Б.Л.Ворф*

### *Состав языка*

В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания и предложения. Подобные элементы содержит и алгоритмический язык, только слова называют лексемами (элементарными конструкциями), словосочетания – выражениями, а предложения – операторами.



Лексемы образуются из символов, выражения – из лексем и символов, а операторы – из символов, выражений и лексем:

| | |
|---|---|
| Символы | основные неделимые знаки, с помощью которых пишутся все тексты на языке. |
| Лексема | минимальная единица языка, имеющая самостоятельный смысл. |
| Выражение | задает правило вычисления некоторого значения. |
| Оператор | задает законченное описание некоторого действия. |

Для описания сложного действия требуется последовательность операторов. Операторы могут быть объединены в составной оператор, или блок. Блоком в языке C++ считается последовательность операторов, заключенная в фигурные скобки { }. В этом случае они рассматриваются как один оператор.

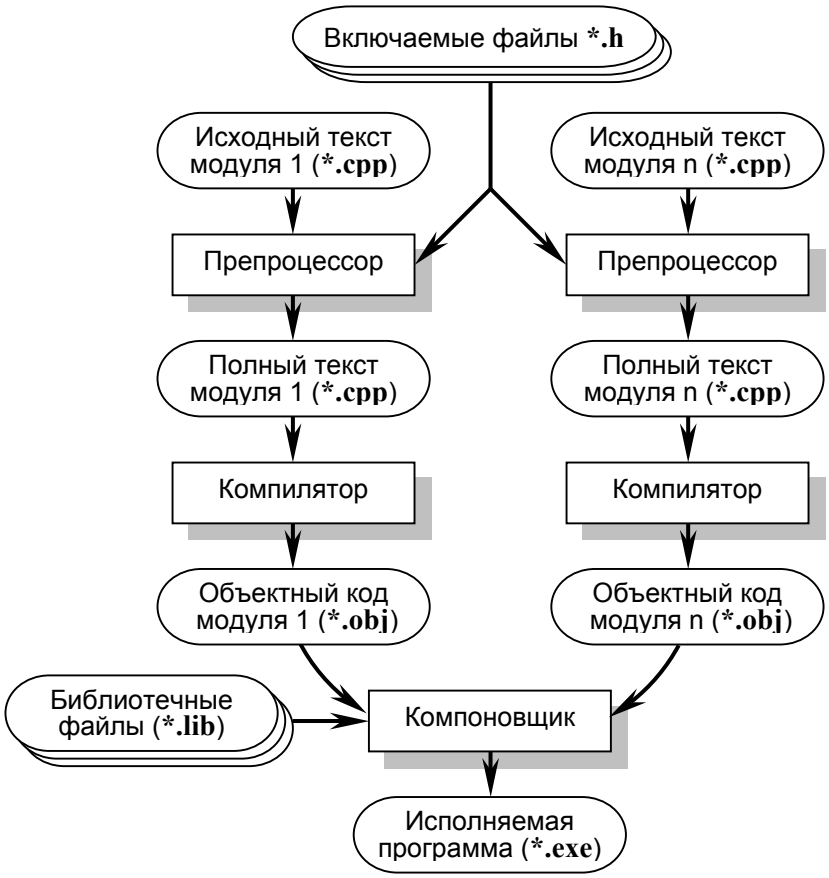
Операторы бывают исполняемые и неисполняемые. Исполняемые операторы задают действия над данными. Неисполняемые операторы служат для описания данных, поэтому их часто называют операторами описания или просто описаниями.

Каждый элемент языка определяется синтаксисом и семантикой. Синтаксические определения устанавливают правила построения элементов языка, а семантика определяет их смысл и правила использования.

Программное обеспечение, разработанное с использованием C++, включает: *идентификаторы*, *ключевые слова*, *функции*, *переменные*, *константы*, *операторы*, *выражения*, *директивы препроцессора*, *структуры*, *массивы* и ряд других элементов.

### *Этапы создания исполняемой программы*

Объединенная единым алгоритмом совокупность описаний и операторов образует программу на алгоритмическом языке. Для того чтобы выполнить программу, требуется перевести ее на язык, понятный процессору – в машинные коды. Этот процесс состоит из нескольких этапов.




Сначала программа передается препроцессору, который выполняет директивы, содержащиеся в ее тексте. Например, включение в текст так называемых заголовочных файлов – текстовых файлов, в которых содержатся описания используемых в программе элементов.

Получившийся полный текст программы поступает на вход компилятора, который выделяет лексемы, а затем на основе грамматики языка распознает выражения и операторы, построенные из этих лексем. При этом компилятор выявляет синтаксические ошибки и в случае их отсутствия строит объектный модуль.

Компоновщик, или редактор связей, формирует исполняемый модуль программы, подключая к объектному модулю другие объектные модули, в том числе содержащие функции библиотек, обращение к которым содержится в любой программе (например, для осуществления вывода на экран). Если программа состоит из нескольких исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение ***.exe** и запускается на выполнение обычным образом.

### *Неформальный способ описания*

Для описания языка в документации часто используется некоторый формальный метаязык, например, формулы Бэкуса-Наура или синтаксические диаграммы. Для наглядности и простоты изложения в этой книге используется широко распространенный неформальный способ описания, при котором необязательные части синтаксических конструкций заключаются в квадратные скобки, текст, который необходимо заменить конкретным значением, пишется по-русски, а выбор одного из нескольких элементов обозначается вертикальной чертой. Например, запись

**[ void | int ] имя ( ) ;**

означает, что вместо конструкции **имя** необходимо указать конкретное имя в соответствии с правилами языка, а перед ним может находиться либо **void**, либо **int**, либо ничего. Фигурные скобки используются для группировки элементов, из которых требуется выбрать только один. В тех случаях, когда квадратные скобки являются элементом синтаксиса, это оговаривается особо.

### *Алфавит языка*

Алфавит C++ включает:

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

Из символов алфавита формируются лексемы языка:

- идентификаторы;
- ключевые (зарезервированные) слова;
- знаки операций;
- константы;
- разделители: скобки, точка, запятая, пробельные символы.

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

### *Идентификаторы*

*Идентификатор – это имя программного объекта*. Под *идентификатором* понимается имя переменной, функции или метки. Для идентификаторов в C++ определены следующие правила:

- в идентификаторе могут использоваться цифры, знак подчеркивания и только латинские буквы;
- прописные и строчные буквы различаются;
- первым символом идентификатора может быть буква или знак подчеркивания, но не цифра;
- пробелы внутри имен не допускаются;
- идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка.

Для улучшения читаемости программы следует давать объектам осмысленные имена. Существует соглашение о правилах создания имен, называемое венгерской нотацией (поскольку предложил ее сотрудник компании Microsoft венгр по национальности), по которому каждое слово, составляющее идентификатор, начинается с прописной буквы, а вначале ставится префикс, соответствующий типу величины, например, **iMaxLength**, **lpfnSetFirstDialog**. Другая традиция – разделять слова, составляющие имя, знаками подчеркивания: **max_length**, **number_of_galosh**.

Длина идентификатора по стандарту не ограничена, но некоторые компиляторы и компоновщики налагают на нее ограничения. Идентификатор создается на этапе объявления переменной, функции, типа и т. п., после этого его можно использовать в последующих операторах программы. Не рекомендуется начинать идентификаторы с символа подчеркивания, поскольку они могут совпасть с именами системных функций или переменных, и, кроме того, это снижает мобильность программы.

### *Ключевые слова*

*Ключевые слова – это зарезервированные идентификаторы, которые наделены определенным смыслом*. Их можно использовать только в соответствии со значением известным компилятору языка C++. Приведем список ключевых слов:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **class** | **const** | **continue** | **default** | **delete** | **do** | **double** | **else** | **enum** |
| **extern** | **float** | **for** | **friend** | **goto** | **if** | **inline** | **int** | **long** |
| **new** | **operator** | **overload** | **public** | **register** | **return** | **short** | **sizeof** | **static** |
| **struct** | **switch** | **this** | **typedef** | **union** | **unsigned** | **virtual** | **void** | **while** |

При стандартных настройках редактор Visual C++ выделяет ключевые слова синим цветом.

### *Знаки операций*

*Знак операции – это один или более символов, определяющих действие над операндами*. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста. Все знаки операций за исключением **[ ]**, **( )** и **? :** представляют собой отдельные лексемы.

Большинство стандартных операций может быть переопределено (перегружено).

## КОНСТАНТЫ

*Константами называют неизменяемые величины.* В языке C++ определены *целые*, *вещественные*, *символьные* и *строковые* константы.

### *Целые константы*

*Целые константы* бывают следующих форматов: *десятичный*, *восьмеричный* и *шестнадцатеричный*:

| | |
|---|---|
| Десятичный | последовательность десятичных цифр, начинающаяся не с нуля.<br>Например: **123** **2384** |
| Восьмеричный | последовательность восьмеричных цифр (от **0** до **7**), начинающаяся с нуля.<br>Например: **034** **047** |
| Шестнадцатеричный | начинается с символов **0x** или **0X**, за которыми следуют шестнадцатеричные цифры (**0...9**, **A...F**).<br>Например : **0xF4** **0X5D** |

Константам, встречающимся в программе, приписывается тот или иной тип в соответствии с их видом. Если этот тип по каким-либо причинам не устраивает программиста, он может явно указать требуемый тип с помощью суффиксов **L**, **l** (**long**) и **U**, **u** (**unsigned**). Например, константа **32L** будет иметь тип **long** и занимать 4 байта. Можно использовать суффиксы **L** и **U** одновременно, например, **Ox22UL** или **05Lu**.

### *Вещественные константы*

*Вещественные константы – числа с плавающей запятой* могут быть записаны в *десятичном* или в *экспоненциальном* формате:

| | |
|---|---|
| Десятичный | **[ + \| – ] [ цифры ] . [ цифры ] [ F \| f \| L \| l ]**<br>Могут быть опущены либо целая часть, либо дробная, но не обе сразу.<br>Например: **35.** **.001f** **5.7** |
| Экспоненциальный | **[ + \| – ] [ цифры ] [ . ] [ цифры ] { E \| e } [ + \| – ] цифры [ F \| f \| L \| l ]**<br>Могут быть опущены либо целая часть, либо дробная, но не обе сразу. Если указаны обе части, символ точки обязателен.<br>Например: **5.2E6** **.011e-3** **5E+10F**<br>Эти константы соответствуют следующим значениям, записанным в общепринятом виде: **$5.2*10^6$** **$0.011*10^{-3}$** **$5*10^{10}$** |

Константы с плавающей точкой имеют по умолчанию (без суффикса) тип **double**. Если вещественная константа записана в программе с суффиксом **F** или **f**, то её тип – **float**, если с суффиксом **L** или **l**, то её тип – **long double**.

Вещественная константа в экспоненциальном формате представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты (**E** или **e**), порядок – справа от знака. Значение кон-

станты определяется как произведение мантиссы и возведенного в указанную в порядке степень числа 10. Обратите внимание, что пробелы внутри числа не допускаются, а для отделения целой части от дробной используется не запятая, а точка.

Если требуется сформировать отрицательную целую или вещественную константу, то перед константой ставится знак унарной операции изменения знака ( – ), например: **–218** , **–022** , **–0x3C** , **–4.8** , **–0.1e4** .

Другое общепринятое название этих способов записи значений вещественных чисел – формат с фиксированной и плавающей запятой.

### *Символьные константы*

*Символьная константа – символ, заключенных в одинарные кавычки* (апострофы). Например: **'e'**, **'@'**, **'<'**, **'A'**, **'ю'**, **'*'**, **'\0'**, **'\n'**.

Символьные константы в С++ занимают в памяти 1 байт и, следовательно, могут принимать значения от 0 до 255. При этом существует ряд символов, которые не отображаются при печати, – они выполняют специальные действия: возврат каретки, табуляция, и называются *символами escape-последовательности* или *управляющими последовательностями*. Термин "escape-последовательность" ввела компания Epson, ставшая первой фирмой, которая для управления выводом информации на своих принтерах стала использовать неотображаемые символы. Исторически сложилось так, что управляющие последовательности начинались с кода с десятичным значением 27 (0x1B), что соответствовало символу "Escape" кодировки ASCII.

Escape-символы в программе изображаются в виде обратного слеша, за которым следует буква или символ:

| | |
|---|---|
| \\ | вывод на печать обратной черты; |
| **\'** | вывод апострофа; |
| **\"** | вывод при печати кавычки; |
| **\?** | символ вопросительного знака; |
| **\a** | подача звукового сигнала; |
| **\b** | возврат курсора на 1 символ назад; |
| **\n** | перевод строки; |
| **\r** | возврат курсора на начало текущей строки; |
| **\t** | перевод курсора к следующей позиции табуляции; |

### *Строковые константы*

*Строковая константа – последовательность символов, заключённая в кавычки*. Управляющие последовательности могут использоваться и в строковых константах, называемых иначе строковыми литералами. Например, если внутри строки требуется записать кавычку, ее предваряют косой чертой, по которой компилятор отличает ее от кавычки, ограничивающей строку:

**"\tИздательский дом \"Питер\"\n"**

Строковый литерал также содержит знак табуляции и символ перехода на другую строку.

Все строковые литералы рассматриваются компилятором как различные объекты.

Строковые константы, отделенные в программе только пробельными символами, при компиляции объединяются в одну. Длинную строковую константу можно разместить на нескольких строках, при этом следующая строка воспринимается как продолжение предыдущей. Например, строки

**"Никто не доволен своей "**
**"внешностью, но все довольны "**
**"своим умом."**

полностью эквивалентны строке

**"Никто не доволен своей внешностью, но все довольны своим умом."**

В конец каждого строкового литерала компилятором добавляется нулевой символ, представляемый управляющей последовательностью **\0**. Поэтому длина строки всегда на единицу больше количества символов в ее записи. Таким образом, пустая строка **""** имеет длину 1 байт. Обратите внимание на разницу между строкой из одного символа, например, **"A"**, и символьной константой **'A'**.

Пустая символьная константа недопустима.

## КОММЕНТАРИИ

При создании программного продукта разработчик всегда ясно представляет, что будет выполнять та или иная часть его программы. Однако, если программа довольно сложная (большой размер исходного текста, нетривиальность алгоритма), через какое-то время бывает трудно вспомнить всю цепь логических рассуждений, предшествующих написанию программы. Особенно сложно бывает разобраться в текстах программ других разработчиков.

Чтобы избежать подобных неприятностей, в процессе составления программного кода используются так называемые комментарии. Текст комментариев *компилятор всегда игнорирует*, но позволяет программисту описывать назначение какой-либо части программы. Размер комментариев ограничен только размером свободной памяти, хотя, конечно, не стоит превращать текст программы в литературное произведение.

В С++ используется две разновидности комментариев. Первый, традиционный многострочный комментарий, представляет собой блок, начинающийся с последовательности символов ”косая черта со звездочкой” ( /* ) и заканчивающийся символами ”звездочка с косой чертой” ( */ ). Как следует из названия, данный вид комментария может располагаться на нескольких строках. Комментарии этого типа не могут быть вложенными друг в друга.

Второй вид – однострочный комментарий – следует за “двойной косой чертой” ( // ) до конца текущей строки. Этот тип комментария может быть вложенным в многострочный комментарий.

Кроме пояснений текста программы комментарии можно использовать для временного исключения из программы некоторой ее части. Этот прием обычно используется при отладке. Например:

```
int main ( )              // главная функция
{
   int a = 0 ;            // назначение переменной
//  int d ;
   int c = 7 ;
/*                        // начало комментария
   int b =15 ;            // объявляет и инициализирует переменную b
   a = 2*c ;
*/                        // конец комментария
   return 0 ;             // возвращает 0
}
```

В приведенном примере компилятор проигнорирует объявление переменной **b** и **d**, а также присвоение переменной **a** значения переменной **c**.

При стандартных настройках редактора Visual C++ комментарии выделяются красным цветом.

## ТИПЫ ДАННЫХ С++

### *Концепция типа данных*

Основная цель любой программы состоит в обработке данных. Данные различного типа хранятся и обрабатываются по-разному. В любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип. Тип данных определяет:

- *внутреннее представление* данных в памяти компьютера;
- *множество значений*, которые могут принимать величины этого типа;
- *размер памяти*, выделяемой компилятором для данных этого типа;
- *операции и функции*, которые можно применять к величинам этого типа.

Исходя из этих характеристик, программист выбирает тип каждой величины, используемой в программе для представления реальных объектов. Обязательное описание типа позволяет компилятору производить проверку допустимости различных конструкций программы. От типа величины зависят машинные команды, которые будут использоваться для обработки данных.

Все типы языка С++ можно разделить на *основные* и *составные*. В языке С++ определено шесть основных типов данных для представления целых, вещественных, символьных и логических величин. На основе этих типов программист может вводить описание составных типов. К ним относятся массивы, перечисления, функции, структуры, ссылки, указатели, объединения и классы.

### *Основные типы данных*

Основные (стандартные) типы данных часто называют арифметическими, поскольку их можно использовать в арифметических операциях. Для описания основных типов определены следующие ключевые слова:

| | |
|---|---|
| **int** | целый; |
| **char** | символьный; |
| **bool** | логический; |
| **float** | вещественный; |
| **double** | вещественный с двойной точностью. |

Первые три типа называют целочисленными (целыми), последние два – типами с плавающей точкой. Код, который формирует компилятор для обработки целых величин, отличается от кода для величин с плавающей точкой.

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

| | |
|---|---|
| **short** | короткий; |
| **long** | длинный; |
| **signed** | знаковый; |
| **unsigned** | беззнаковый. |

### *Целый тип ( int )*

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

Спецификатор **short** перед именем типа указывает компилятору, что под число требуется отвести 2 байта независимо от разрядности процессора. Спецификатор **long** означает, что целая величина будет занимать 4 байта. Таким образом, на 16-разрядном компьютере эквиваленты **int** и **short int**, а на 32-разрядном – **int** и **long int**.

*Внутреннее представление* величины целого типа – целое число в двоичном коде. При использовании спецификатора **signed** старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Спецификатор **unsigned** позволяет представлять только положительные числа, поскольку старший разряд рассматривается как часть кода числа. Таким образом, диапазон значений типа **int** зависит от спецификаторов. По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор **signed** можно опускать.

### *Символьный тип ( char )*

Под величину символьного типа отводится количество байт, достаточное для размещения любого символа из набора символов для данного компьютера, что и обусловило название типа. Как правило, это 1 байт. Тип **char**, как и другие целые типы, может быть со знаком или без знака. В величинах со знаком можно хранить значения в диапазоне от –128 до 127. При использовании спецификатора **unsigned** значения могут находиться в пределах от 0 до 255. Этого достаточно для хранения любого символа из 256-символьного набора ASCII. Величины типа **char** применяются также для хранения целых чисел, не превышающих границы указанных диапазонов.

### *Логический тип ( bool )*

Величины логического типа могут принимать только значения **true** и **false**, являющиеся зарезервированными словами. Внутренняя форма представления значения **false** – 0 (нуль). Любое другое значение интерпретируется как **true**. При преобразовании к целому типу **true** имеет значение 1.

### *Типы с плавающей точкой ( float, double и long double )*

Стандарт C++ определяет три типа данных для хранения вещественных значений: **float**, **double** и **long double**.

Типы данных с плавающей точкой хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка. В IBM PC-совместимых компьютерах величины типа **float** занимают 4 байта, из которых один двоичный разряд отводится под знак мантиссы, 8 разрядов под порядок и 23 под мантиссу. Мантисса – это число, большее 1.0, но меньшее 2.0. Поскольку старшая цифра мантиссы всегда равна 1, она не хранится.

Для величин типа **double**, занимающих 8 байт, под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка – его диапазон. При одинаковом количестве байт, отводимом под величины типа **float** и **long int**, диапазоны их допустимых значений сильно различаются из-за внутренней формы представления.

Спецификатор **long** перед именем типа **double** указывает, что под величину отводится 10 байт.

| Тип | Диапазон значений | Размер (байт) | Значащих цифр |
|---|---|---|---|
| **bool** | **true \| false** | **1** | |
| **signed char** | **–128 ... 127** | **1** | |
| **unsigned char** | **0 ... 255** | **1** | |
| **signed short int** | **–32 768 ... 32 767** | **2** | |
| **unsigned short int** | **0 ... 65 535** | **2** | |
| **signed long int** | **–2 147 483 648 ... 2 147 483 647** | **4** | |
| **unsigned long int** | **0 ... 4 294 967 295** | **4** | |
| **float** | **3.4e-38 ... 3.4e+38** | **4** | **7** |
| **double** | **1.7e-308 ... 1.7e+308** | **8** | **15** |
| **long double** | **3.4e-4932 ... 3.4e+4932** | **10** | **20** |
| **void** | **–** | **4** | |

Для вещественных типов в таблице приведены абсолютные величины минимальных и максимальных значений.

Для написания переносимых на различные платформы программ нельзя делать предположений о размере типа **int**. Для его получения необходимо пользоваться операцией **sizeof**, результатом которой является размер типа в байтах. Например, для операционной системы MS-DOS **sizeof ( int )** даст в результате 2, а для Windows 9X или OS/2 результатом будет 4.

Различные виды целых и вещественных типов, различающиеся, диапазоном и точностью представления данных, введены для того, чтобы дать программисту возможность наиболее эффективно использовать возможности конкретной аппаратуры, поскольку от выбора типа зависит скорость вычислений и объем памяти. Но оптимизированная для компьютеров какого-либо одного типа программа может стать не переносимой на другие платформы, поэтому в общем случае следует избегать зависимостей от конкретных характеристик типов данных.

### *Отсутствие типа. Тип void*

Переменная типа **void** не имеет значения и служит для согласования синтаксиса. Например, синтаксис требует, чтобы функция возвращала значение. Если не требуется использовать возвращенное значение, перед именем функции ставится тип **void**.

Множество значений этого типа пусто. Кроме того, он используется для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

## СТРУКТУРА ПРОГРАММЫ

Программа на языке C++ состоит из *функций*, *описаний* и *директив препроцессора*. Одна из функций должна иметь имя **main ( )**. Программа обычно состоит из следующих разделов:

```
[ директивы препроцессора ]
[ объявление типов ]
[ определение глобальных констант и переменных ]
[ прототипы функций ]
int main ( )
{
   операторы главной функции
}
[ реализация объявленных функций ]
```

Выполнение программы начинается с первого оператора функции **main ( )**. Программа может состоять из нескольких модулей (исходных файлов).

### *Директивы препроцессора*

Одно из назначений директив препроцессора – подключение стандартных библиотек и других модулей программы. Синтаксис директивы, которая подключает заголовочный файл, расположенный в каталоге **\INCLUDE** имеет вид:

**#include <имя_файла.h>**

Все директивы препроцессора начинаются со знака #. Имя заготовочного файла вместе с расширением заключено в косые скобки. Если имя файла указано в кавычках (**" "**), то препроцессор ищет файл в текущем каталоге (в папке проекта).

### *Пример программы*

Приведём программу, использующую функции ввода/вывода *библиотеки классов С++*:

```
#include <iostream.h>                          // подключает библиотеку ввода/вывода
int main ( )                                   // заголовок главной функции
{
   int A ;                                     // определяет переменную целого типа
   cout << "Input integer number\t" ;           // выводит на экран приглашение
   cin >> A ;                                  // вводит число с клавиатуры
   cout << "You input number\t" << A << ".\t Thank you!\n";     // вывод на экран
   return 0;                                   // функция возвращает ноль
}
```

Первая строка этой программы – директива препроцессора, по которой в текст программы вставляется заголовочный файл **iostream.h**, содержащий описание использованных в программе функций ввода/вывода. Заголовочный файл **iostream.h** содержит описание набора классов для управления вводом/выводом. В нем определены стандартные объекты-потоки **cin** для ввода с клавиатуры и **cout** для вывода на экран, а также операции помещения в поток << и чтения из потока >>.

Главная функция содержит определение переменной целого типа с именем **A**. Далее в поток вывода на экран **cout** помещается строковая константа, которая содержит в конце знак табуляции **'\t'**.

Следующая строка содержит оператор чтения целого значения из входного потока (клавиатуры) **cin** в переменную **A**. При этом программа приостанавливает свою работу и ждёт ввода цифр и клавиши **Enter**.

Далее в выходной поток с помощью одного оператора помещаются две строковые константы и значение переменной **A**. Строковые константы содержат знаки табуляции **'\t'** и символ перехода на следующую строку **'\n'**.

Последний оператор функции **main ( )** содержит команду передачи значения выражения (в данном случае ноль) в место вызова. Поскольку функцию **main ( )** вызывает операционная система, то она получит ноль. Это означает, что программа успешно (без ошибок) завершена.

## ПЕРЕМЕННЫЕ

### *Определение переменной*

*Переменная – это именованная область оперативной памяти, в которой хранятся данные*. У переменной есть *имя*, *тип, значение, время жизни, область действия и область видимости*. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Тип переменной определяет размер памяти, выделяемой для её хранения. Кроме того, от типа переменной зависит набор операций, которые определены для этого типа данных. Перед использованием любая переменная должна быть описана. Имя переменной должно быть уникальным в своей области действия (например, в одном блоке не может быть двух переменных с одинаковыми именами).

Синтаксис оператора определения переменных имеет вид:

**[ Класс памяти ] Тип Имя1 [ Инициализатор ] [ , Имя2 [ Инициализатор ] ] ;**

Рассмотрим правила задания составных частей этого оператора.

| | |
|---|---|
| **Класс памяти** | необязательный параметр, может принимать одно из значений: **auto**, **extern**, **static** и **register**; |
| **Тип** | идентификатор типа: стандартный тип или тип, определённый пользователем; |
| **Имя** | идентификатор переменной, уникальное имя в блоке; |
| **Инициализатор** | оператор, который присваивает переменной начальное значение (*инициализация переменной*). Инициализатор можно записывать в двух формах: |

**= Выражение** со знаком равенства;
**( Выражение )** в круглых скобках.

С помощью одного оператора можно определить несколько переменных одного типа.

### *Определение типизованной константы*

*Типизованная константа – это переменная, значение которой не может быть изменено после инициализации*. Такую переменную также называют именованной константой или просто константой. Синтаксис определения типизованной константы имеет вид:

**[ Класс памяти ] const [ Тип ] Имя1 Инициализатор [ , Имя2 Инициализатор ] ;**

Типизованная константа объявляется с помощью ключевого слова **const**, за которым следует указание типа константы. Если тип константы не указан, то по умолчанию её тип **int**. В отличие от переменных, константы всегда должны быть инициализированы. **Инициализатор** *должен* использовать только *константное выражение*. С помощью одного оператора можно объявить несколько констант одного типа.

Переменная и типизованная константа может быть объявлена в любом месте программы. Если тип инициализирующего выражения не совпадает с типом переменной, выполняются *преобразования типа* по определенным правилам.

### *Пример определения типизованных констант*

### *Область действия переменной*

*Описание переменной*, кроме типа и класса памяти, явно или по умолчанию *задает* ее *область действия*. Класс памяти и область действия зависят не только от собственно объявления, но и от места его размещения в тексте программы.

*Область действия переменной – это часть программы, в которой её можно использовать для доступа к связанной с ним области памяти*. В зависимости от области действия переменная может быть *локальной* или *глобальной*.

Если переменная определена внутри блока (блок ограничен фигурными скобками), она называется *локальной*, область ее действия – *от точки описания до конца блока*, включая все вложенные блоки.

Если переменная определена вне любого блока, она называется *глобальной* и *областью* ее *действия* считается *файл*, в котором она определена, от точки описания до его конца.

Класс памяти определяет время жизни и область видимости программного объекта (в частности, переменной). Если класс памяти не указан явным образом, он определяется компилятором исходя из контекста объявления.

### *Время жизни переменной*

*Время жизни переменной* может быть *постоянным* (в течение выполнения программы) и *временным* (в течение выполнения блока).

### *Область видимости переменной*

*Областью видимости переменной называется часть текста программы*, *из которой допустим обычный доступ* к *связанной с переменной областью памяти*. Чаще всего область видимости совпадает с областью действия. Исключением является ситуация, когда во вложенном блоке описана переменная с таким же именем. В этом случае внешняя переменная во вложенном блоке невидима, хотя он и входит в ее область действия. Тем не менее, к этой переменной, если она глобальная, можно обратиться, используя операцию доступа к области видимости **::** (два двоеточия).

### *Классы памяти*

Для задания *класса памяти* используются следующие спецификаторы:

**auto** автоматическая переменная. Память под нее выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти происходит при выходе из блока, в котором описана переменная. Время ее жизни – с момента описания до конца блока. Для глобальных переменных этот спецификатор не используется, а для локальных он принимается по умолчанию, поэтому задавать его явным образом большого смысла не имеет.

**extern** означает, что переменная определяется в другом месте программы (в другом файле или дальше по тексту). Используется для создания переменных, доступных во всех модулях программы, в которых они объявлены.

**static** статическая переменная. Время жизни – постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть глобальными и локальными. Глобальные статические переменные видны только в том модуле, в котором они описаны.

**register** аналогичен **auto**, но память выделяется по возможности в регистрах процессора. Если такой возможности у компилятора нет, переменные обрабатываются как **auto**.

Если при определении начальное значение переменных явным образом не задается, компилятор присваивает глобальным и статическим переменным нулевое значение соответствующего типа. Автоматические переменные по умолчанию не инициализируются. Поэтому в переменной, которая объявлена без инициализации, хранится случайное значение, которое было в оперативной памяти в момент объявления переменной.

### *Пример описания переменных*

Рассмотрим пример программы, которая использует локальные, глобальные, статические и внешние переменные:

```
int A ;                      // 1   определяет глобальную переменную A, инициализирует её нулём
int main ( )
{
   {
      int B ;                // 2   определяет локальную переменную B без инициализации
   }
   cout << "B = " << B ;      // 3   Ошибка: B – необъявленная переменная
   extern int X ;            // 4   объявляет переменную X, которая определена в другом месте
   cout << "X = " << X ;      // 5   выводит на экран значение внешней переменной: X = 4
   static int C ;            // 6   определяет локальную статическую переменную C
                             //       и инициализирует её нулём
   cout << "C = " << C ;      // 7   выводит на экран значение статической переменной: C = 0
   int A ;                   // 8   определяет локальную переменную A без инициализации
   cout << "A = " << A ;      // 9   выводит на экран случайное значение локальной переменной:
                             //       A = -858993460
   cout << "A = " << ::A ;    // 10  выводит на экран значение глобальной переменной: A = 0
   return 0 ;
}
int X = 4 ;                  // 11  определяет и инициализирует внешнюю переменную
```

В этом примере глобальная переменная **A** определена вне всех блоков. Память под нее выделяется в сегменте данных в начале работы программы, областью действия является вся программа. Область видимости – вся программа, кроме строк 6-8, так как в первой из них определяется локальная переменная с тем же именем, область действия которой начинается с точки ее описания и заканчивается при выходе из блока. Переменные b и c – локальные, область их видимости – блок, но время жизни различно: память под b выделяется в стеке при входе в блок и освобождается при выходе из него, а переменная c располагается в сегменте данных и существует все время, пока работает программа.

### *Объявление и определение переменной*

*Описание* переменной может выполняться в форме *объявления* или *определения*. *Объявление* информирует компилятор о *типе переменной и классе памяти*. *Определение* содержит, кроме этого, указание компилятору *выделить память* в соответствии с типом переменной. В C++ большинство объявлений являются одновременно и определениями. В приведенном выше примере только описание 3 является объявлением, но не определением.

Переменная может быть *объявлена многократно, но определена только в одном месте программы*, поскольку объявление просто описывает свойства переменной, а определение связывает ее с конкретной областью памяти.

### *Совет*

Не жалейте времени на придумывание подходящих имен для переменных. Имя должно отражать смысл хранимой величины, быть легко распознаваемым и, желательно, не содержать символов, которые можно перепутать друг с другом, например, **1**, **l** (строчная L) или **I** (прописная i). Для разделения частей имени можно использовать знак подчеркивания. Не давайте переменным имена, демонстрирующие знание иностранного сленга – ведь “как вы лодку назовёте, так она и поплывет”. Как правило, переменным с большой областью видимости даются более длинные имена (желательно с префиксом типа), а для переменных, вся жизнь которых проходит на протяжении нескольких строк исходного текста, хватит и одной буквы с комментарием при объявлении.

## ПРЕОБРАЗОВАНИЕ ТИПОВ

В C++ существует *явное* и *неявное* преобразование типов. Синтаксис явного вызова функции преобразование типа имеет вид:

**( Тип ) Имя**

Функция получает значение, которое хранится в переменной **Имя**, преобразует его в указанный **Тип** и возвращает в место вызова. Функция преобразования может быть также применена к выражению.

В C++ определены функции преобразования *только для стандартных типов*. Для преобразования значений в тип, определённый пользователем, необходимо определить соответствующую функцию.

Необходимо отметить, что функция преобразования преобразует только тип значения, которое хранится в переменной. *Функция преобразования не изменяет тип переменной*.

Рассмотрим пример явного вызова функций преобразования типа:

```
float F = 15.8F ;       // определяет и инициализирует вещественную переменную одинарной точности
int I = 5 ;             // определяет и инициализирует переменную целого типа
double D = 6.6 ;        // определяет и инициализирует вещественную переменную двойной точности
bool B = true ;         // определяет и инициализирует логическую переменную
char C = 'A' ;          // определяет и инициализирует переменную символьного типа
D = (double) F ;        // явный вызов функции преобразования данных типа float в double
cout <<"D="<< D ;       // выводит на экран: D=15.8
I = (int) D ;           // явный вызов функции преобразования данных типа double в int
cout <<"I="<< I ;       // выводит на экран: I=15
I = (int) C ;           // явный вызов функции преобразования данных типа char в int
cout <<"I="<< I ;       // выводит на экран: I=65 (код символа 'A')
C = (char) 66 ;         // явный вызов функции преобразования данных типа int в char
cout <<"C="<< C ;       // выводит на экран: C=B (символ, код которого равен 66)
I = (int) B ;           // явный вызов функции преобразования данных типа bool в int
cout <<"I="<< I ;       // выводит на экран: I=1
B = (bool) C ;          // явный вызов функции преобразования данных типа char в bool
cout <<"B="<< B ;       // выводит на экран: B=1
```

В общем случае неявное преобразование типов сводится к участию в выражении переменных разного типа (так называемая арифметика смешанных типов). Если подобная операция осуществляется над переменными базовых типов, она может повлечь за собой ошибки: в случае, например, если результат занимает в памяти больше места, чем отведено под принимающую переменную, неизбежна потеря значащих разрядов.

Рассмотрим пример неявного вызова функций преобразования типа:

```
double D = 6 ;          // неявный вызов функции преобразования данных типа const int в double
float F = 15.8 ;        // неявный вызов функции преобразования данных типа const double в float
                        // компилятор выводит на экран предупреждение о возможной потере точности
const int I = 5.8 ;     // не вызывает функцию преобразования типа
cout <<"I="<< I ;       // выводит на экран: I=2036603204, передача данных выполнена без преобразования
I = F ;                 // неявный вызов функции преобразования данных типа float в int
                        // компилятор выводит на экран предупреждение о возможной потере точности
```

Преобразование вещественного числа в целое происходит за счёт отсечение дробной части вещественного значения.