**Типы данных языка С++. Часть 1**

**Имена, переменные и константы**

### Имена

Для символического обозначения величин, имен функций и т.п. используются ***имена*** или *идентификаторы* .

***Идентификаторы*** в языке Си++ – это последовательность знаков, начинающаяся с буквы или знака подчеркивания. В *идентификаторах* можно использовать заглавные и строчные латинские буквы, цифры и знак подчеркивания. *Длина* *идентификаторов* произвольная. Примеры правильных *идентификаторов*:

abc A12 NameOfPerson BYTES\_PER\_WORD

Отметим, что abc и Abc – два разных *идентификатора*, т.е. заглавные и строчные буквы различаются. Примеры неправильных *идентификаторов*:

12X a-b

Ряд слов в языке Си++ имеет особое *значение* и не может использоваться в качестве *идентификаторов*. Такие зарезервированные слова называются ключевыми.

*Список* ключевых слов:

№ п/п Ключевое слово Описание

1 and альтернатива оператору &&

2 and\_eq альтернатива оператору &=

3 asm вставить код на ассемблере

4 auto объявить локальную переменную

5 bitand альтернатива оператору bitwise &

6 bitor альтернатива оператору |

7 bool объявить булеву переменную

8 break выйти с цикла

9 case определенная ветка в операторе ветвления switch

10 catch обработать исключение от throw

11 char объявить символьную переменную

12 class объявить класс

13 compl альтернатива оператору ~

14 const объявить неизменяемые данные или функцию, которая не изменяет данных

15 const\_cast приведение типа от константной переменной

16 continue пропустить код до конца цикла и начать новую итерацию

17 default вариант по умолчанию в switch

18 delete освобождение памяти, выделенной new

19 do оператор цикла

20 double объявить вещественное число двойной точности

21 dynamic\_cast выполнить приведение типов во время выполнения

22 else ветка оператора if, которая выполняется при ложном условии

23 enum создание перечисляемого типа

24 explicit использовать данный конструктор только при полном соответствии типов

25 export позволить разделить объявление и реализацию шаблона

26 extern указание компилятору, что переменная объявлена в другом файле

27 false константа для ложного значения булевой переменной

28 float объявить вещественную переменную

29 for оператор цикла

30 friend разрешить другим функциям доступ к приватным данным класса

31 goto безусловный переход

32 if оператор условия

33 inline оптимизация вызовов для функций (встраиваемые функции)

34 int объявить переменную целого типа

35 long объявить следующий тип длинным

36 mutable перекрыть константность

37 namespace определить новое пространство имен

38 new выделить динамическую память под новую переменную

39 not альтернатива оператору !

40 not\_eq альтернатива оператору !=

41 operator создание перегруженных операторов

42 or альтернатива оператору ||

43 or\_eq альтернатива оператору |=

44 private объявить приватное поле класса

45 protected объявить защищенное поле класса

46 public объявить общее поле класса

47 register запрос компилятору на оптимизацию переменной по скорости

48 reinterpret\_cast изменить тип переменной

49 return возврат из функции

50 short объявить короткое целое

51 signed сделать данный целый тип знаковым

52 sizeof возвратить размер переменной или типа

53 static создать статическую переменную

54 static\_cast сделать не полиморфное приведение типов

55 struct определить новую структуру

56 switch оператор ветвления

57 template создать шаблонную функцию

58 this указатель на текущий объект

59 throw выбросить исключение

60 true константа для истинного значения булевой переменной

61 try выполнить код, который может выкинуть исключение

62 typedef создание нового типа с существующего

63 typeid задать описание объекта

64 typename объявить класс или иной новый тип данных

65 union структура, содержащая несколько переменных в одной области памяти

66 unsigned объявить беззнаковое целое

67 using импортировать полностью или частично указанное пространство имен внутрь текущего блока

68 virtual создать виртуальную функцию

69 void объявить функцию или переменную без типа

70 volatile предупреждение компилятору, что переменная может измениться сама

71 wchar\_t объявить переменную типа wide-character

72 while оператор цикла

73 xor альтернатива оператору ^

74 xor\_eq альтернатива оператору ^=

В следующем примере

int max(int x, int y)

{

 if (x > y)

 return x;

 else

 return y;

}

max, x и y – *имена* или *идентификаторы*. Слова int, if, return и else – ключевые слова, они не могут быть *именами переменных* или функций и используются для других целей.

### Переменные

*Программа* оперирует информацией, представленной в виде различных объектов и величин. ***Переменная*** – это символическое обозначение величины в программе. Как ясно из названия, **значение**   *переменной* (или величина, которую она обозначает) во *время выполнения* программы может изменяться.

С точки зрения архитектуры компьютера, ***переменная*** – это символическое обозначение ячейки оперативной памяти программы, в которой хранятся данные. Содержимое этой ячейки – это текущее *значение* *переменной*.

В языке Си++ прежде чем использовать *переменную*, ее необходимо **объявить**. Объявить *переменную* с *именем*   x можно так:

int x;

В объявлении первым стоит название типа *переменной*   int (*целое число*), а затем *идентификатор*   x – *имя переменной*. У *переменной*   x есть тип – в данном случае *целое число*. Тип *переменной* определяет, какие возможные значения эта *переменная* может принимать и какие *операции* можно выполнять над данной *переменной*. Тип *переменной* изменить нельзя, т.е. пока *переменная*   x существует, она всегда будет целого типа.

Язык Си++ – это строго *типизированный язык*. Любая величина, используемая в программе, принадлежит к какому-либо типу. При любом использовании *переменных* в программе проверяется, применимо ли *выражение* или операция к типу *переменной*. Довольно часто смысл выражения зависит от типа участвующих в нем *переменных*.

Например, если мы запишем x+y, где x – объявленная выше *переменная*, то *переменная* y должна быть одного из числовых типов.

Соответствие типов проверяется во *время компиляции* программы. Если *компилятор* обнаруживает *несоответствие типа* *переменной* и ее использования, он выдаст ошибку (или предупреждение). Однако во *время выполнения* программы типы не проверяются. Такой подход, с одной стороны, позволяет обнаружить и исправить большое количество ошибок на стадии компиляции, а, с другой стороны, не замедляет выполнения программы.

*Переменной* можно присвоить какое-либо *значение* с помощью *операции* **присваивания**. Присвоить – это значит установить текущее *значение* *переменной*. По-другому можно объяснить, что *операция присваивания* запоминает новое *значение* в ячейке памяти, которая обозначена *переменной*.

int x; // объявить целую переменную x

int y; // объявить целую переменную y

x = 0; // присвоить x значение 0

y = x + 1; // присвоить y значение x + 1,

 // т.е. 1

x = 1; // присвоить x значение 1

y = x + 1; // присвоить y значение x + 1,

 // теперь уже 2

### Константы

В программе можно явно записать величину – число, символ и т.п. Например, мы можем записать *выражение* x + 4 – сложить текущее *значение* *переменной*   x и число 4. В зависимости от того, при каких условиях мы будем выполнять программу, *значение* *переменной*   x может быть различным. Однако *целое число* четыре всегда останется прежним. Это неизменяемая величина или константа.

Таким образом, явная *запись* значения в программе – это константа.

Далеко не всегда удобно записывать *константы* в тексте программы явно. Гораздо чаще используются *символические константы*. Например, если мы запишем

const int BITS\_IN\_WORD = 32;

то затем *имя*   BITS\_IN\_WORD можно будет использовать вместо целого числа 32.

Преимущества такого подхода очевидны. Во-первых, *имя*   BITS\_IN\_WORD (битов в машинном слове) дает хорошую подсказку, для чего используется данное число. Без комментариев понятно, что *выражение*

b / BITS\_IN\_WORD

(*значение* b разделить на число 32 ) вычисляет количество машинных слов, необходимых для хранения b битов информации. Во-вторых, если по каким-либо причинам нам надо изменить эту константу, потребуется изменить только одно *место* в программе – *определение* *константы*, оставив все случаи ее использования как есть. (Например, мы переносим программу на *компьютер* с другой длиной машинного слова.)