## *2. Типы данных C++*

***Тип данных*** определяет:

- внутреннее представление данных в памяти компьютера;
- множество допустимых значений величин этого типа;
- совокупность операций и функций, которые можно применять к величинам этого типа.

### *Основные типы данных*

C++ предоставляет набор встроенных (стандартных) типов данных: символьный, целый, вещественный, логический и набор составных и расширенных типов: строки, массивы, комплексные числа и т.д.

*Встроенные* типы данных предопределены в языке. Это самые простые величины, из которых составляют все производные типы.

Для описания основных (базовых) типов используются ключевые слова:

- **int** (целый)
- **char** (символьный)
- **wchar_t** (расширенный символьный)
- **bool** (логический)
- **float** (вещественный тип, или число с плавающей точкой одинарной точности)
- **double** (.вещественный с двойной точностью, или число с плавающей точкой двойной точности)

Первые четыре типа называют целочисленными (целыми), последние два – типами с плавающей точкой. Внутренне представление целых типов и типов с плавающей точкой различно.

Для уточнения внутреннего представления данных и диапазона значений используются ключевые слова (в различной литературе их называют поразному: модификаторы, спецификаторы):

- **short** (короткий)
- **long** (длинный)
- **signed** (знаковый)
- **unsigned** (беззнаковый)

### *Целый тип (`int`)*

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

Для представления целого, описанного с модификатором **`short`**, чаще всего отводится 2 байта (16 бит), с модификатором **`long`** – 4 байта (32 бита). Разработчики компилятора вправе сами выбирать подходящие размеры, сообразуясь с характеристиками своего компьютера и соблюдая следующие ограничения: значения типов **`short`** и **`int`** представляются по крайней мере 16 битами; типа **`long`** – по крайней мере 32 битами; размер

**short** не больше размера **int**, который в свою очередь не больше размера **long.**

Для 16-итового процессора эквивалентны **int** и **short int**, для 32-битового – **int** и **long int.**

Например:

```
short int summa;
long int counter;
```

В таких объявлениях слово **int** можно опускать, что обычно и делается. Если только не возникает противоречий со здравым смыслом, **short int** и **long int** должны быть разной длины, а **int** соответствовать естественному размеру целых на данной машине. Поэтому следующие объявления эквивалентны предыдущим:

```
short  summa;
long  counter;
```

По умолчанию все целочисленные типы являются знаковыми, т.е. квалификатор **signed** можно опускать.

Величины, описанные с **unsigned** всегда положительны или равны нулю.

В объявлениях слово **signed int, unsigned int** слово **int** можно опускать. Объявления

```
signed int summa1;
unsigned int counter1;
```

и

```
signed summa1;
unsigned counter1;
```

эквивалентны.

В таблице 6.1 приведены диапазоны значений величин целого типа с различными спецификаторами.

Чтобы не писать длинный список для определения типа можно воспользоваться ключевым словом **typedef** для определения синонима: так вместо объявлений

```
unsigned short int a,b;
unsigned short int x1, x2, x3;
```

написать

```
typedef unsigned short USHORT;
USHORT a,b;
USHORT x1, x2, x3;
```

### *Символьный тип (**char**)*

Символьный тип занимает 1 байт (8 битов). Так как этот тип относится к целым, то он может быть со знаком и без знака. Поэтому **unsigned char**

имеет диапазон значений от 0 до 255, а **`signed char`** – от -128 до 127. Являются ли значения типа просто **`char`** знаковыми или беззнаковыми, зависит от реализации.

Таблица 6.1.

| Тип | Длина в байтах | Диапазон значений |
|---|---|---|
| `bool` | 1 | значения true(истина) или false (ложь) |
| `char` | 1 | 0 ... 255 или –128…127 |
| `signed char` | 1 | –128…127 |
| `unsigned char` | 1 | 0 ... 255 |
| `short int`<br>`signed short int`<br>`short`<br>`signed short` | 2 | -32768 ... +32767 |
| `unsigned short int`<br>`unsigned short` | 2 | 0 ... 65535 |
| `int`<br>`signed int`<br>`signed` | Не менее 2<br>2(для 16-разрядных)<br>4(для 32-разрядных) | Зависит от длины<br>-32768 ... +32767<br><br>– 2 147 483 648 ... + 2 147 483 647 |
| `unsigned int`<br>`unsigned` | Не менее 2<br>2(для 16-разрядных)<br>4(для 32-разрядных) | Зависит от длины<br>0 ... 65535<br><br>0 ... 4 294 967 295 |
| `long int`<br>`signed long int`<br>`long`<br>`signed long` | 4 | – 2 147 483 648 ... + 2 147 483 647 |
| `unsigned long`<br>`unsigned long int` | 4 | 0 ... 4 294 967 295 |
| `float` | 4 | ±3.4e-38 ... ±3.4e+38 (7 значащих цифр) |
| `double` | 8 | ±1.7e-308 ... ±1.7e+308 [1] (15 значащих цифр) |

### *Расширенный символьный тип (`wchar_t`)*

Символьный тип **`wchar_t`** предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например Unicode. Размер

[1] Цветом выделены значения, которые в различных источниках отличаются.

этого типа зависит от реализации, как правило, он соответствует типу **short:** 2 байта (16 битов).

Строковые константы этого типа записыаются с префиксом L, например:

```
L"Hello"
L"кодировка"
```

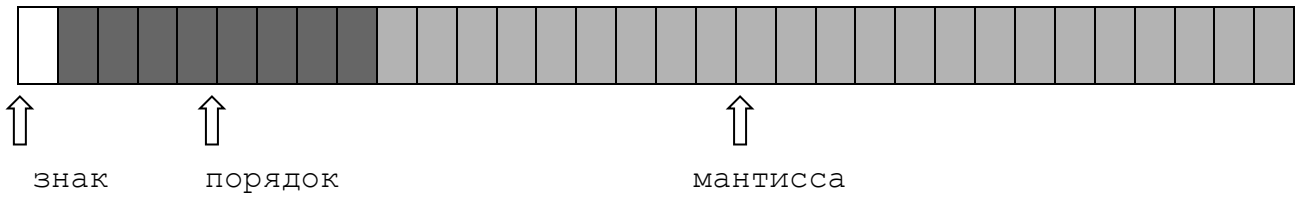
### *Логический тип (**bool**)*

Величины *логического (булевксого)* типа (**bool**) принимают значения **true** (истина) или **false** (ложь) и занимают один байт памяти. Значение **false** представляется в компьютере как 0, а любое другое значение трактуется как **true** и при преобразовании к целому типу имеет значение 1.

### *Типы с плавающей точкой*

Стандарт языка C++ определяет три типа данных для работы с вещественными числами: **float**, **double** и **long double.** Типы данных с плавающей точкой в компьютере хранятся в виде мантиссы и порядка.

В IBM-совместимых компьютерах тип **float** занимает 4 байта:




один разряд под знак числа, 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.

С типом **float** можно использовать **long,** тогда переменная будет занимать 8 байтов, но на этапе компиляции будет выдано предупреждение.

Тип **double** предназначен для арифметики с плавающей точкой повышенной точности, занимает 8 байт, под порядок и мантиссу в нем отводится 11 и 52 разряда соответственно.

С типом **double** можно использовать модификатор **long.** Однако в C++ Visual Studio 2005 **long double** аналогичен **double,long float** так же аналогичен **double,** только в этом случае при компиляции выдается предупреждение о неправильном использовании модификатора.

В других реализациях размеры вещественных типов могут быть иными.

Именованные константы для всех размеров вместе с другими характеристиками машины и компилятора содержатся в стандартных заголовочных файлах ***<limits.h>*** *и* ***<float.h>***

Константы с плавающей точкой по умолчанию имеют тип **double.** Можно явно задавать тип константы:

```
2e+2  – тип double
2e+2L – тип long double
2e+2f – тип float
```

***Тип* `void`**

В языке С существует тип данных, который обозначает пустое значение: **`void`**. Он используется для 2-ух целей:

- для определения функций, которые не возвращают значения, для указания пустого списка аргументов функции;
- как базовый тип для указателей;
- в операциях приведения типов.

Эти вопросы будут рассмотрены позже.

## *2.2. Объявление переменных*

Допустимо объявлять несколько переменных в одном операторе.

```
signed int summa1, summa2, x, y; // четыре переменные
unsigned int counter1, a, b;  // три переменные
```

Но нельзя объявить в одном операторе несколько переменных разных типов.

## *2.3. Присвоение значений переменным*

Для присвоения значений переменным в программе используется оператор присваивания =.

```
signed int summa1, summa2;
summa1=0;
summa2=-100;
```

Для присвоения начальных значений переменным можно объединить эти операторы присваивания с объявлением переменных, т. е. Выполнить инициализацию.

```
signed int summa1=0, summa2=-100;
```

В следующем примере не все переменные инициализируются при объявлении.

```
unsigned int counter1=1, a, b=100;
```

## *2.4. Константы*

**Константой** называется фиксированная величина, которая не изменяется в процессе выполнения алгоритма или программы. Примером является всем известное знаменитое число "ПИ", значение которого в компьютере принимается приближенно равным 3,1415926.

Создаваемые константы надо сразу инициализировать.

### *Литеральные константы (самоопределенные)*

Это значение, непосредственно вводимое в программе. В следующем примере числа 1 и 100 являются литералами.

```
unsigned int counter1=1, a, b=100;
```

В языке определены следующие типы констант:

| Константа | Формат | Примеры (разделены запятыми) |
|---|---|---|
| Целая | Десятичный: последовательность десятичных цифр, начинающихся не с нуля, если это не число нуль | 8, 0, -5, 4U, 3000L |
| | Восьмеричный: нуль, за которым следуют восьмеричные цифры | 077, 0111 |
| | Шестнадцатеричный: 0x или 0X, за которым следуют шестнадцатеричные цифры | 0xFFA2, 0X00FF, 0x12A09 |
| Вещественная | Десятичный: [цифры].[цифры][2] | 3.5, .001, 123. |
| | Экспоненциальный: [цифры][.][цифры]{E\|e}[+\|-][цифры][3] | 0.2e6, .1E10, 23.7e-4, 5E+2 |
| Символьная | Один или два символа, заключенные в апострофы | 's', 'П', '\n', '\0xFF', '\077' |
| Строковая | Последовательность символов, заключенная в кавычки | "Здесь был я", "ЗАО \"МММ\"", "C:\\autoexec.bat", "\tSumma=\0xF5\n" |

**Десятичная целая** константа, если она не ноль, записывается как обычная последовательность десятичных цифр. При этом старшая цифра десятичной константы не должна быть нулем.

Ноль, как старшая цифра числа, как префикс числа, принимается за признак **восьмеричной целой** константы. Естественно, что все последующие ее цифры не должны превышать числа 7.

Например:

`1864` и `3765` – десятичные целые константы, поскольку у них нет префикса "0" и старшая цифра есть не равное нулю число "1" или "3".

`03567` и `01762` – целые восьмеричные. Каждая из них имеет префикс "0" - первую цифру числа и все цифры каждого числа не превышают числа 7.

Признаком **шестнадцатеричной целой** константы является префиксы "0x" или "0X" перед последовательностью шестнадцатеричных цифр (0, 1, 2, ..., 9, A, B, C, D, E, F).

Например:

---

[2] Могут быть опущены либо целая, либо дробная часть, но не обе сразу

[3] Могут быть опущены либо целая, либо дробная часть, но не обе сразу.

0xA6 – шестнадцатеричное представление десятичного числа 10*16+6=160+6=166.

0XDF – шестнадцатеричное представление десятичного числа 13*16+15=208+15=223.

**Вещественные константы** представляются в памяти компьютера как числа с плавающей запятой. Число называется действительным или вещественным, если оно имеет целую и дробную части, между которыми проставлена десятичная точка. Вещественная константа состоит из двух целых констант. Одна для целой части числа, другая - для изображения дробной части числа. Между этими двумя целыми константами ставится десятичная точка. Иногда такая константа может не иметь или целой или дробной части, но точка между ними обязательна.

Например:

```
546.
3.1415926
.009865
```

и так далее. Иногда вещественная константа записывается в экспоненциальной форме. Это значит, что вначале пишется мантисса числа в стандартной форме, то есть не превышающая по абсолютной величине 10, вслед за нею записывается буква e или E, которая обозначает выражение "умножить на число 10 в степени". Далее записывается порядок числа. Порядок числа показывает количество нулей до первой значащей цифры числа, если само число по абсолютной величине не превышает 1, или количество нулей в конце очень большого числа.

Например:

```
.31415926E+01 = 0.31415626*10 = 3.1415926
7.897534E-03 = 7.897534*0.001 = 0.007897534
4.987535E+10 = 4.987535* 10000000000 = 49875350000.
```

Компилятор определяет тип константы по ее внешнему виду. Всем вещественным константам по умолчанию компилятор присваивает тип **double**, для которого в памяти компьютера будет автоматически отводиться 8 байт (64 двоичных разряда). Даже такая действительная константа, как школьное значение числа "ПИ", равная 3.14, будет иметь тип **double**. Программист имеет право пересмотреть это распределение памяти и задать тип констант. Для этого нужно указать после числового значения константы один из суффиксов:

F(f) – тип **float** (для вещественной константы);

L(l) – тип **long** (для целых и вещественных констант);

U(u) – тип **unsigned** (для целых констант).

Так, например, чтобы вещественная константа 6.85432 занимала 4 байта, а не 8 байт, то нужно написать так: 6.85432F - константа типа **`float`** (выделяется 4 байта).

**Символьные константы** служат для изображения отдельных символов из таблицы ASCII символов. Каждая такая константа состоит из изображения самого символа и ограничивающих его с двух сторон апострофов, например, 'A' или 'b'.

Символ обратной косой черты ("backslash" - обратный слэш) используется для представления:

- кодов, которые не имеют графического изображения на экране дисплея или на принтере, например,`'\r'`-возврат каретки (см. табл.);
- символов апострофа, кавычки, знака вопроса, обратной косой черты;
- любого символа с помощью его шестнадцатеричного или восьмеричного кода.

Последовательности символов, начинающиеся с обратной косой черты, называют *управляющими*, или *escape-последовательностями (ESC-sequence*).

В таблице представлены эти управляющие символы и дана расшифровка их назначения.

| | |
|---|---|
| **\a** | **звуковой сигнал, звонок (bell)** |
| **\b** | **возврат на одну позицию (на один символ) (забой)** |
| **\f** | **новая станица, перевод страницы (form feed)** |
| **\n** | **новая строка, перевод строки (new line)** |
| **\r** | **возврат каретки (курсора) к началу строки (carriage return)** |
| **\t** | **горизонтальная табуляция (tab)** |
| **\v** | **вертикальная табуляция (vertical tab)** |
| **** | **обратная косая черта - обратный слэш (backslash)** |
| **\"** | **кавычка (символ двойной кавычки)** |
| **\'** | **апостроф (одиночная кавычка)** |
| **\0** | **нулевой символ, нулевой байт** |
| **\0ddd** | **восьмеричная константа, здесь d - восьмеричная цифра** |
| **\0xhhh** | **шестнадцатеричная константа, здесь h - шестнадцатеричная цифра** |
| **\?** | **знак вопроса** |

Управляющая последовательность интерпретируется как один символ. Если непосредственно после обратного слэша следует символ, отсутствующий в таблице, то результат интерпретации не определен. Если в последовательности цифр встречается недопустимая, она считается концом цифрового кода.

Управляющие последовательности могут использоваться и в строковых константах (строковых литералах).

**Строковая константа** – это последовательность символов, заключенная в двойные кавычки. В языке Си нет стандартного типа "строка". Поэтому строки представляются как массив символов, заканчивающийся нулевым байтом. Таким образом, строка занимает на один байт памяти больше, чем количество символов в этой строке, ибо в конец строки заносится символьная константа - нулевой символ '\0'. В памяти компьютера все символы строки размешаются подряд друг за другом. Формально строковые константы не относятся к константам языка Си в соответствии с его стандартом.

Символьные и строковые константы – разные конструкции языка. Символьная константа 'a' отличается от строковой константы "a". Внешнее отличие состоит в способе записи констант. Первая из них записывается обрамленной в апострофы с двух сторон, вторая - ограничивается с двух сторон двойными кавычками. Кроме того, 'a' занимает 1 байт памяти, а "a" – два байта, т.е. на 1 байт больше.

Пустая строка "" имеет длину 1, а пустая символьная константа недопустима.

Строковые константы, отделенные в программе только пробельными символами, при компиляции объединяются в одну.

Длинную строковую константу можно разместить на нескольких строках. Для этого достаточно поместить обратную косую черту в самом конце строки:

```
"Эта строковая константа \
размещена на нескольких \
строках программы"
```

### *Определение констант с помощью директивы #define*

Для определения константы можно воспользоваться директивой:.

```
#define kurs_dollar 2115
```

Эта директива выполняет текстовую подстановку, везде в тексте программы препроцессором вместо константы будет подставлено ее значение. Такая константа не имеет никакого конкретного типа. В стандарте C++ этот способ объявлен устаревшим.

## *2.2. Подробнее о переменных*

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. Оператор описания переменной в простейшем случае выглядит так:

*[ класспамяти ] [* **const** *] тип*
*имя [ инициализатор ] {, имя [ инициализатор ] }…*;

Класс памяти может быть задан с помощью слов **auto**, **extern**, **static** либо **register**. Далее это понятие будет пояснено.

Модификатор **const** позволяет задать именованные константы.

Инициализатор позволяет присвоить начальное значение переменной (и обязателен при описании константы). Инициализатор можно записать в двух формах:

= *значение*

или

( *значение* )

Рассмотрим примеры описания переменных:

```
short int a = 1;
const char CR = '\n';
char s, sf('a'), st = '1';
static unsigned int P;
```

Кроме упомянутых характеристик, переменная обладает *областью действия*, *временем жизни* и *областью видимости*.

Область действия переменной (а в общем случае – любого идентификатора) – это часть программы, в которой его можно использовать. Переменная, описанная внутри блока, т.е. части программы, заключенной в фигурные скобки, считается локальной. Область ее действия – от точки описания до конца блока, включая вложенные блоки. Переменные, описанные вне блоков, являются глобальными, их область действия – до конца файла, в котором эта переменная описана (включая внешние файлы, в которые переменная экспортируется по директиве `#include`).

Область видимости чаще всего совпадает с областью действия, однако если во вложенном блоке описана переменная с уже существующим именем, то это делает невидимой внешнюю переменную. тем не менее, к глобальной невидимой переменной можно обратиться с использованием операции :: (расширение области видимости).

Время жизни переменной определяет, как долго сохраняется ее значение. Значение локальных переменных теряются при выходе из блока, в котором они описаны, если только эти переменные не были описаны с классом памяти **static**. Напротив, время жизни глобальных переменных и переменных с классом памяти **static** – постоянное.

Рассмотрим теперь остальные классы памяти. Класс памяти **auto** означает, что память для переменной выделяется в стеке. Этот класс задается по умолчанию. Для переменных с классом памяти **register** память по возможности выделяется в регистрах процессора. Класс памяти **extern** означает, что переменная будет определена далее по тексту (в этом же или других файлах). Использование класса памяти **extern** – это один из примеров объявления, но не определения переменной.

Продолжим рассмотрение примеров описания переменных.

```
int a;
int main() {
 int b;
 extern int x; //объявление, но не определение!
 a = 1;   // обращение к глобальной переменной
 float a; // а теперь она становится невидимой
 a = 2.5; // обращение к локальной переменной
 ::a = 4; // обращение к глобальной переменной за счет
          // расширения области видимости
 return 0;
}
int x = 4; // а вот теперь x определена!
```