Глава 1

Типы данных, переменные и константы

Языки C и C++ предлагают программисту богатый ассортимент встроенных типов данных. При этом для удовлетворения практически любых нужд могут быть созданы типы данных, определяемые самим программистом. Для каждого действительного типа данных можно создавать переменные. Кроме того, можно определять константы встроенных типов данных. В этой главе рассматриваются различные языковые средства, связанные с типами данных, переменными и константами.

Основные типы

В версии C89 определены следующие основные типы данных.

Тип	Ключевое слово
Символьный	`char`
Целочисленный	`int`
С плавающей запятой	`float`
С плавающей запятой двойной точности	`double`
Пустой	`void`

К перечисленным выше типам в версии C99 добавлены следующие.

Тип	Ключевое слово
Логический, или булев (ИСТИНА/ЛОЖЬ)	`_Bool`
Комплексный	`_Complex`
Мнимый	`_Imaginary`

В языке C++ определены следующие основные типы.

Тип	Ключевое слово
Логический, или булев (`true/false`)	`bool`
Символьный	`char`

Тип	Ключевое слово
Целочисленный	int
С плавающей запятой	float
С плавающей запятой двойной точности	double
Пустой	void
Символьный двубайтовый	wchar_t

Как видите, все версии языков C и C++ позволяют работать с пятью основными типами: char, int, float, double и void. Заметьте также, что ключевые слова, используемые для обозначения логического типа данных в версии C99 и языке C++, отличаются: _Bool (C99) и bool (C++). В версии C89 логический тип данных вообще отсутствует.

Некоторые основные типы могут быть модифицированы с помощью одного или нескольких модификаторов типов.

- signed
- unsigned
- short
- long

Модификаторы указываются перед наименованием типа, который они модифицируют. В следующей таблице приведены все разрешенные в языках C и C++ встроенные типы данных, включая модификаторы, а также их гарантированные минимальные диапазоны. Большинство компиляторов расширяет указанные минимумы для одного или нескольких типов. Кроме того, если в вашем компьютере используется двоичная арифметика (что имеет место в большинстве случаев), самое маленькое отрицательное значение, которое можно хранить с помощью целочисленного типа данных со знаком, будет на единицу больше указанных минимумов. Например, диапазон типа int для большинства компьютеров составляет –32 768–32 767. От конкретной реализации компилятора также зависит, каким является тип char: со знаком или без него.

Тип	Минимальный диапазон
char	–127–127 или 0–255
unsigned char	0–255
signed char	–127–127
int	–32 767–32 767

Тип	Минимальный диапазон
`unsigned int`	0–65 535
`signed int`	Аналогичен типу `int`
`short int`	Аналогичен типу `int`
`unsigned short int`	0–65 535
`signed short int`	Аналогичен типу `short int`
`long int`	–2 147 483 647–2 147 483 647
`signed long int`	Аналогичен типу `long int`
`unsigned long int`	0–4 294 967 295
`long long int`	$-(2^{63}-1)-2^{63}-1$ (только в C99)
`signed long long int`	Аналогичен типу `long long int` (только в C99)
`unsigned long long int`	$0-2^{64}-1$ (только в C99)
`float`	6 значащих цифр
`double`	10 значащих цифр
`long double`	10 значащих цифр
`wchar_t`	Аналогичен типу `unsigned int`

Если при объявлении переменных используется один модификатор (без наименования типа), то предполагается использование типа `int`. Например, целочисленную переменную без знака можно объявить, используя лишь ключевое слово `unsigned`. Таким образом, следующие объявления эквивалентны.

```
unsigned int i; // тип int указан явно
unsigned i;     // здесь тип int подразумевается
```

Объявление переменных

Все переменные должны быть объявлены до их использования. Общая форма объявления имеет такой вид:

тип имя_переменной;

Например, чтобы объявить `x` переменной типа `float`, `y` — целочисленной переменной и `ch` — символьной, необходимо записать следующее:

```
float x;
int y;
char ch;
```

Используя форму списка, можно в одной записи объявить сразу несколько переменных, разделив их запятыми. Например, следующая инструкция объявляет три целые переменные.

```
int a, b, c;
```

Инициализация переменных

Переменную можно инициализировать, записав после ее имени знак равенства и начальное значение. Например, следующее объявление позволяет присвоить переменной `count` начальное значение 100:

```
int count = 100;
```

Инициализатором может быть любое выражение, которое действительно при объявлении переменной. Оно может включать другие переменные и вызовы функций. Однако в языке C глобальные переменные и статические (`static`) локальные переменные должны быть инициализированы только с использованием константных выражений.

Идентификаторы

Имена переменных, функций и типов, определенные пользователем, — это примеры идентификаторов. В языках C и C++ идентификаторы представляют собой последовательности, состоящие из одной или нескольких букв, цифр и символов подчеркивания. (Однако идентификатор не может начинаться с цифры.)

Идентификаторы могут иметь любую длину, но не все символы являются значащими. Различают два типа идентификаторов: внешние и внутренние. Внешние идентификаторы участвуют во внешнем процессе компоновки. Они называются *внешними именами* и включают имена функций и глобальных переменных, которые используются в различных файлах. Если идентификатор не участвует во внешнем процессе компоновки, он является внутренним. Идентификаторы, относящиеся к этому типу, называются *внутренними именами* и включают, например, имена локальных переменных. В версии C89 значащими являются первые 6 символов внешнего и 31 символ внутреннего идентификатора. В версии C99 пределы значимости расширены: внешний идентификатор имеет 31 значащий символ, а внутренний — 63 значащих символа. В языке C++ 1 024 символа любого идентификатора являются значащими.

Символ подчеркивания часто используется для удобочитаемости (например, `first_time`) или в качестве первого символа

идентификатора (например, _count). Идентификаторы, записанные прописными и строчными буквами, распознаются как разные. Например, test и TEST — это две различные переменные. В языках C и C++ резервируются все идентификаторы, начинающиеся с двух символов подчеркивания или одного символа подчеркивания, за которым следует прописная буква.

Классы

Класс — это основной элемент инкапсуляции в языке C++. Класс определяется с помощью ключевого слова class. В языке C понятие класса отсутствует. По сути класс представляет собой коллекцию переменных и функций, которые манипулируют этими переменными. Переменные и функции, образующие класс, называются *членами*. Ниже показана общая форма записи класса.

```
class имя_класса : список_наследования {
  // Закрытые члены по умолчанию.
protected:
  // Закрытые члены, которые могут быть унаследованы.
public:
  // Открытые члены.
} список_объектов;
```

Здесь *имя_класса* — это имя типа класса. После компиляции объявления класса *имя_класса* становится именем нового типа данных, который можно использовать для объявления объектов класса. *Список_объектов* — это список объектов типа *имя_класса*, разделенных запятыми. Такой список необязателен. Объекты класса могут быть объявлены позже в программе путем использования имени класса. *Список_наследования* также необязателен. С его помощью указывается базовый класс или классы, наследуемые новым классом. (См. раздел "Наследование" далее в этой главе.)

Класс может включать *функцию конструктора* и *функцию деструктора*. (Обе функции необязательны.) Конструктор вызывается при первоначальном создании объекта класса, а деструктор — при разрушении этого объекта. Имя конструктора совпадает с именем класса. Функция деструктора имеет имя, совпадающее с именем класса, но ему предшествует символ "тильда" (~). Ни конструкторы, ни деструкторы не имеют возвращаемого типа, или типа результата. В иерархии классов конструкторы выполняются в порядке их "классового происхождения", а деструкторы — в обратном порядке.

По умолчанию все элементы класса закрыты, и к ним могут получить доступ только другие члены этого класса. Чтобы разрешить доступ к элементу класса со стороны функций, не являющихся членами данного класса, необходимо объявить этот элемент в качестве открытого члена после ключевого слова `public`, как в приведенном ниже примере.

```
class myclass {
  int a, b; // закрытые члены класса myclass
public:
  // Члены класса, доступные для не членов класса.
  void setab(int i, int j) { a = i; b = j; }
  void showab() { cout << a << ' ' << b << endl; }
} ;

myclass ob1, ob2;
```

Это объявление создает тип класса с именем `myclass`, который содержит две закрытые переменные: `a` и `b`. Он также содержит две открытые функции `setab()` и `showab()`. В приведенном выше фрагменте программы также объявляются два объекта типа `myclass` с именами `ob1` и `ob2`.

Чтобы сделать член класса наследуемым, но недоступным (закрытым) для внешних инструкций, объявите его *защищенным*, т.е. используйте в объявлении спецификатор доступа `protected`. Защищенный член доступен для производных классов, но недоступен за пределами его иерархии классов.

При выполнении операций над объектом класса для ссылки на отдельные его члены используйте оператор "точка" (`.`). Оператор "стрелка" (`->`) используется для доступа к объекту с помощью указателя. Например, в следующем фрагменте программы доступ к функции `putinfo()` объекта `ob` реализуется с использованием оператора "точка", а к функции `show()` — с помощью оператора "стрелка" .

```
struct cl_type {
  int x;
  float f;
public:
  void putinfo(int a, float t) { x = a; f = t; }
  void show() { cout << a << ' ' << f << endl; }
} ;

cl_type ob, *p;

// ...

ob.putinfo(10, 0.23);
```

```
p = &ob; // Помещаем адрес объекта ob в переменную p.

p->show(); // Отображаем данные объекта ob.
```

С помощью ключевого слова `template` можно создавать обобщенные, или шаблонные, классы. (См. раздел "`template`" в главе 5.)

Наследование

В языке C++ один класс может наследовать свойства другого. Унаследованный класс обычно называется *базовым классом*, а класс-наследник — *производным классом*. Когда один класс наследует другой, формируется *иерархия классов*. Общая форма наследования классов имеет такой вид:

```
class имя_класса : спецификатор_доступа
имя_базового_класса {
  // . . .
} ;
```

Здесь *спецификатор_доступа* определяет способ наследования базового класса и указывается с помощью одного из трех ключевых слов: `private`, `public` или `protected`. Его можно опустить, и в этом случае предполагается использование спецификатора `public`, если производный класс является структурой (`struct`), или `private`, если производный класс имеет тип `class`. При наследовании нескольких классов используется список имен классов, разделенных запятыми.

Если в качестве элемента *спецификатор_доступа* используется вариант `public`, все `public`- и `protected`-члены базового класса остаются `public`- и `protected`-членами производного класса. Если в качестве элемента *спецификатор_доступа* используется вариант `private`, все `public`- и `protected`-члены базового класса становятся `private`-членами производного класса. В случае защищенного доступа (`protected`) все `public`- и `protected`-члены базового класса становятся `protected`-членами производного класса.

В приведенной ниже иерархии классов класс `derived` наследует класс `base` закрытым способом, т.е. с использованием спецификатора `private`. Это означает, что переменная `i` становится закрытым членом класса `derived`.

```
class base {
public:
  int i;
```

```
};

class derived : private base {
  int j;
public:
  derived(int a) { j = i = a; }
  int getj() { return j; }
  int geti() { return i; } // Класс derived имеет
                           // доступ к переменной i.
};

derived ob(9); // Создаем объект класса derived.

cout << ob.geti() << " " << ob.getj(); // OK

// ob.i = 10; // ОШИБКА: переменная i недоступна
              // за пределами класса derived!
```

Структуры

Структура создается с помощью ключевого слова `struct`. В языке C++ структура также определяет класс. Единственное различие между `class`- и `struct`-объектами состоит в том, что по умолчанию все члены структуры являются открытыми. Чтобы сделать член закрытым, необходимо использовать ключевое слово `private`. Общая форма объявления структуры имеет такой вид:

```
struct имя_структуры: список_наследования {
  // Открытые члены по умолчанию.
protected:
  // Закрытые члены, которые могут быть
  // унаследованы.
private:
  // Закрытые члены.
} список_объектов;
```

В языке C к структурам применяются некоторые ограничения. Они могут содержать только данные-члены; функции-члены здесь не разрешены. C-структуры не поддерживают наследования. Кроме того, все члены являются открытыми, а ключевые слова `public`, `protected` и `private` использовать нельзя.

Объединения

Объединение — это тип класса, в котором все данные-члены разделяют одну область памяти. В языке C++ объединение может включать как функции-члены, так и члены данных. Все члены объединения открыты по умолчанию. Для создания закрытых элементов необходимо использовать ключевое слово `private`. Общая форма объявления объединения выглядит так:

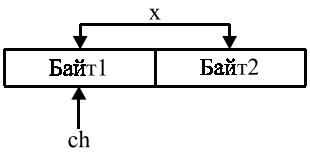
```
union имя_класса {
  // Открытые члены по умолчанию.
private:
  // Закрытые члены.
} список_объектов;
```

В языке C объединения могут содержать только члены данных, а спецификатор доступа `private` не поддерживается.

Элементы объединения перекрывают друг друга. Например, в записи

```
union tom {
  char ch;
  int x;
} t;
```

объявляется объединение `tom`, которое предполагает следующее распределение памяти (при использовании двубайтовых целых).



Как и в классе, на отдельные переменные, составляющие объединение, можно ссылаться с помощью оператора “точка”. Оператор “стрелка” используется для доступа к объединению с помощью указателя.

Применительно к объединениям существует несколько ограничений. Объединение не может наследовать классы любого типа. Объединение не может быть базовым классом. Объединение не может иметь виртуальные функции-члены. Члены объединения не могут быть объявлены как статические. Членом объединения не может быть ссылка. Объединение не может иметь в качестве члена объект, перегружающий оператор “равно” (=). Наконец, ни один объект не может быть членом объединения, если

класс этого объекта явно определяет функцию конструктора или деструктора. (Иначе говоря, приемлемы объекты, которые имеют только конструкторы и деструкторы по умолчанию.)

В языке C++ существует специальный тип объединения, называемый *анонимным*. В объявлении анонимного объединения не содержится имени класса и не объявляются объекты. Анонимное объединение просто уведомляет компилятор о том, что его переменные-члены должны разделять одну область памяти. Однако к самим переменным можно обращаться напрямую, не прибегая к обычному синтаксису операторов "точка" и "стрелка". Переменные, составляющие анонимное объединение, находятся на том же уровне области видимости, что и другие переменные, объявленные внутри того же блока. Это означает, что имена переменных объединения не должны конфликтовать с именами других переменных, которые действительны в пределах своей области видимости. Приведем пример анонимного объединения.

```
union { // Анонимное объединение.
  int a;   // Переменные a и f разделяют
  float f; // одну и ту же область памяти.
};

// ...

a = 10;    // Доступ к переменной a.
cout << f; // Доступ к переменной f.
```

Здесь обе переменные, `a` и `f`, разделяют одну область памяти. Как видите, на имена переменных объединения можно ссылаться напрямую, без оператора "точка" или "стрелка".

Совет программисту

В языке C++ при создании структур в стиле языка C (C-стиле), которые включают только данные-члены, обычной практикой считается использование типа `struct`. Тип `class`, как правило, резервируется для создания классов, содержащих функции-члены. Иногда для описания структуры, создаваемой в C-стиле, используется аббревиатура POD (Plain Old Data — простые данные в старом стиле).

Все ограничения, которые вообще применяются к объединениям, применимы и к анонимным объединениям. Кроме того, анонимные объединения должны содержать только данные — никакие функции-члены не разрешены. Анонимные объедине-

ния не могут содержать ключевые слова `private` и `protected`. Наконец, анонимное объединение, действующее в области видимости, которая определена с заданием пространства имен (`namespace`), должно быть объявлено с использованием модификатора типа данных `static`.

Перечисления

Перечисление представляет собой тип переменной, создаваемый программистом. *Перечисление* — это список именованных целочисленных констант. Таким образом, тип перечисления — это просто спецификация списка имен, принадлежащих конкретному перечислению.

Для создания перечисления используется ключевое слово `enum`. Общая форма типа перечисления имеет следующий вид:

```
enum имя_перечисления {список_имен} список_переменных;
```

Здесь *имя_перечисления* — имя типа данного перечисления. В списке имен, как и в списке переменных, элементы списка разделяются запятыми.

Например, в следующем фрагменте программы сначала определяется перечисление городов, именуемое `cities`, и переменная `c` типа `cities`, а затем переменной `c` присваивается значение `Houston`.

```
enum cities {Houston, Austin, Amarillo} c;
c = Houston;
```

В любом перечислении значение первого (крайнего слева) имени по умолчанию равно `0`, значение второго имени равно `1`, третье имеет значение `2` и т.д. Вообще, каждому имени присваивается значение, на единицу большее значения предыдущего имени. Добавив инициализатор, можно задать имени конкретное значение. Например, в следующем перечислении имя `Austin` будет иметь значение `10`.

```
enum cities {Houston, Austin=10, Amarillo };
```

В этом примере имя `Amarillo` будет иметь значение `11`, поскольку каждое имя должно иметь значение, на единицу большее значения предыдущего имени.

Теги языка C

В языке C, в отличие от C++, имя структуры, объединения или перечисления не определяет в полной мере имя типа. На-

пример, следующий фрагмент программы правомерен для языка C++, но не для C.

```
struct s_type {
  int i;
  double d;
};
// ...
s_type x; // OK для C++, но не для C
```

В языке C++ идентификатор `s_type` определяет полное имя типа, и его можно самостоятельно использовать для объявления объектов. В языке C `s_type` определяет лишь тег (признак). Поэтому в C при объявлении объектов имя тега необходимо предварять соответствующим ключевым словом (`struct`, `union` или `enum`), как в приведенном ниже примере.

```
struct s_type x; // теперь приемлемо для языка C
```

Этот синтаксис разрешен и в языке C++, но используется довольно редко.

Спецификаторы классов памяти

Спецификаторы классов памяти `extern`, `auto`, `register`, `static` и `mutable` используются для изменения способа выделения памяти для переменных в языках C и C++. Эти спецификаторы ставятся перед типом, который они модифицируют.

extern

Если спецификатор `extern` размещается перед именем переменной, компилятор "знает", что переменная имеет внешнее связывание. Внешнее связывание означает, что объект виден вне его собственного файла. По сути, спецификатор `extern` сообщает компилятору лишь тип переменной, не выделяя для нее области памяти. Спецификатор `extern` применяется в тех случаях, когда одни и те же глобальные переменные используются в двух или более файлах.

auto

Спецификатор `auto` уведомляет компилятор о том, что локальная переменная, перед именем которой он стоит, создается при входе в блок и разрушается при выходе из блока. Все переменные, определенные внутри функции, автоматически создаются по умолчанию, потому ключевое слово `auto` используется довольно редко.

register

Когда язык C был только изобретен, спецификатор `register` можно было использовать лишь для локальных целых или символьных переменных, поскольку он заставлял компилятор сохранить эту переменную в регистре центрального процессора, а не в памяти, как обычно. В таком случае все ссылки на эту переменную работали исключительно быстро. С тех пор определение спецификатора `register` расширилось. Теперь любую переменную можно определить как `register`, возложив заботу об оптимизации доступа к ней на компилятор. Для символов и целых это по-прежнему означает их хранение в регистре процессора, но для других типов данных это может означать, например, использование кэш-памяти.

Следует иметь в виду, что использование спецификатора `register` — всего лишь "заявка", которая может быть и не удовлетворена. Компилятор волен ее проигнорировать. Причина такого "неуважения" состоит в том, что только ограниченное число переменных можно оптимизировать ради ускорения обработки данных. При превышении этого предела компилятор будет просто игнорировать дальнейшие `register`-"заявки".

static

Модификатор `static` указывает компилятору на хранение локальной переменной во время всего жизненного цикла программы вместо ее создания и разрушения при каждом входе в область действия и выходе из нее. Следовательно, возведение локальных переменных в ранг статических позволяет поддерживать их значения между вызовами функций.

Модификатор `static` можно также применить к глобальным переменным. В этом случае область видимости такой переменной ограничивается файлом, в котором она объявлена. Это означает, что переменная будет иметь внутреннее связывание. Внутреннее связывание говорит о том, что идентификатор известен только внутри своего файла.

В языке C++ использование спецификатора `static` для членов данных класса приводит к созданию только одной копии этих членов, совместно используемой всеми объектами класса.

mutable

Спецификатор `mutable` применим только в языке C++. Он позволяет члену любого объекта переопределить "клеймо постоянства", т.е. любой член, определенный как `mutable` и принадлежащий объекту, который определен с помощью спецификато-

ра типа const, не несет на себе "печати" const и может быть модифицирован.

Спецификаторы типов

Спецификатор типов const и volatile предоставляют дополнительную информацию о переменных, перед именами которых они стоят.

const

Объекты типа const не могут быть изменены программой в процессе выполнения. Кроме того, объект, адресуемый с помощью указателя, который определен как const, также не может быть модифицирован. Компилятор волен поместить переменные этого типа в память, предназначенную только для чтения (read-only memory — ROM). Переменная, определенная как const, получит значение либо с помощью явной инициализации, либо посредством выполнения аппаратно-зависимых методов. Например, в результате выполнения строки

```
const int a = 10;
```

будет создана целочисленная переменная с именем a и со значением 10, которое не может быть изменено программой. Тем не менее эту переменную вполне можно использовать в выражениях других типов.

volatile

Модификатор volatile сообщает компилятору, что значение переменной может быть изменено средствами, заданными в программе неявным образом. Например, адрес глобальной переменной можно передавать системной процедуре отсчета времени и обновлять по окончании каждого такта системных часов. В этом случае содержимое переменной изменяется без выполнения явных операторов присваивания в программе. Это очень важный момент, поскольку компиляторы иногда автоматически оптимизируют выражения с учетом того, что содержимое переменной не изменяется внутри этого выражения. Оптимизация выполняется в целях достижения более высокой производительности. Однако модификатор volatile не допускает оптимизации программного кода в тех редких случаях, когда это предположение неоправдано.

Совет программисту

Если функция-член класса модифицирована спецификатором типа const, то она не может изменить объект, вызвавший эту функцию. Чтобы объявить функцию-член константной, укажите ключевое слово const после списка ее параметров, как показано ниже.

```
class MyClass {
  int i;
public:
  // const-функция.
  void f1(int a) const {
    i = a; // ОШИБКА! const-функция не может
           // модифицировать объект,
           // вызывающий ее.
  }
  void f2(int a) {
    i = a; // OK, это не const-функция.
  }
};
```

Как видно из комментариев, функция f1() определена с использованием квалификатора const и не может модифицировать объект, который ее вызывает.

restrict

В версию C99 добавлен новый спецификатор типа, именуемый restrict. Он применяется только к указателям. Указатель, квалифицированный с помощью ключевого слова restrict, изначально является единственным средством доступа к объекту, на который он указывает. Доступ к объекту с помощью другого указателя возможен только в том случае, если второй указатель основан на первом. Таким образом, доступ к объекту ограничивается выражениями, основанными на restrict-квалифицированном указателе. Указатели, определенные с помощью спецификатора restrict, используются главным образом как параметры функций или для указания на память, выделенную с помощью функции malloc(). Спецификатор restrict не изменяет семантику программы. Языком C++ спецификатор restrict не поддерживается.

Массивы

Массивы могут быть объявлены с использованием любого типа данных. Общая форма объявления одномерного массива имеет следующий вид:

```
тип имя_массива[размер];
```

Здесь *тип* определяет тип данных для каждого элемента в массиве, а *размер* указывает количество элементов в массиве. Например, чтобы объявить целочисленный массив `x` из 100 элементов, запишите следующее:

```
int x[100];
```

Эта запись создает массив, содержащий 100 элементов, причем номер первого элемента — `0`, а последнего — `99`. Например, при выполнении приведенного ниже цикла в массив `x` загрузятся числа от 0 до 99.

```
for(t=0; t<100; t++) x[t] = t;
```

Массивы можно объявлять, используя любой допустимый тип данных, в том числе созданные программистом классы.

Многомерные массивы объявляются посредством размещения дополнительных измерений внутри добавочных квадратных скобок. Например, чтобы объявить массив целых размерностью 10×20, необходимо записать следующее:

```
int x[10][20];
```

Массивы можно инициализировать с помощью списка инициализаторов, заключенного в фигурные скобки, как показано ниже.

```
int count[5] = { 1, 2, 3, 4, 5 };
```

В C89 и C++ размерность массива должна быть задана константными значениями. Другими словами, в C89 и C++ размерность массива фиксируется на этапе компиляции и не может быть изменена при выполнении программы. Однако в C99 размерность локального массива можно задать с помощью любых действительных целочисленных выражений, даже таких, значения которых становятся известными только при компиляции. Такой массив называется *массивом переменной длины*. Таким образом, размерность массива переменной длины может меняться каждый раз, когда встречается инструкция его объявления.

Определение новых имен типов с помощью ключевого слова typedef

С помощью ключевого слова `typedef` можно создать новое имя для уже существующего типа. Общая форма записи такова:

```
typedef тип новое_имя_типа;
```

Например, следующая программная инструкция сообщает компилятору, что `feet` — это еще одно имя для типа `int`.

```
typedef int feet;
```

После этого следующее объявление совершенно законно и создает целую переменную с именем `distance`.

```
feet distance;
```

Константы

Константы, называемые также *литералами*, относятся к фиксированным значениям, которые не могут быть изменены программой. Константы могут иметь любой базовый тип данных. Способ представления каждой константы зависит от ее типа. Символьные константы заключаются в одинарные кавычки. Например, `'a'` и `'+'` являются символьными константами. Целочисленные константы задаются как числа без дробной части. Например, `10` и `–100` — целочисленные константы. Вещественные константы содержат десятичную запятую, за которой следует дробная часть числа, например `11,123`. Для вещественных констант можно также использовать экспоненциальное представление чисел.

Существует два вещественных типа: `float` и `double`. Кроме того, существует несколько базовых типов, которые образуются с помощью модификаторов типов. По умолчанию компилятор присваивает числовой константе совместимый и одновременно наименьший по объему занимаемой памяти тип данных. Единственным исключением из правила “наименьшего типа” являются вещественные (с плавающей запятой) константы, которым по умолчанию присваивается тип `double`. Во многих случаях такие стандарты работы компилятора вполне приемлемы. Однако у программиста есть возможность точно определить нужный тип.

Чтобы задать точный тип числовой константы, используйте соответствующий суффикс. Для вещественных типов действуют

следующие суффиксы: если вещественное число завершить буквой F, оно будет обрабатываться с использованием типа float, а если буквой L, подразумевается тип long double. Для целых типов суффикс U означает использование типа unsigned, а суффикс L — тип long. Ниже приведены примеры.

Тип данных	Примеры констант
int	1, 123, 21000, –234
long int	35000L, –34L
unsigned int	10000U, 987U
float	123.23F, 4.34e–3F
double	123.23, 12312333, –0.9876324
long double	1001.2L

Шестнадцатеричные и восьмеричные константы

Иногда удобно вместо десятичной системы счисления использовать восьмеричную или шестнадцатеричную. В восьмеричной системе основанием служит число 8, а для выражения чисел используются цифры от 0 до 7. В восьмеричной системе число 10 имеет то же значение, что число 8 в десятичной. Система счисления по основанию 16 называется шестнадцатеричной и использует цифры от 0 до 9 плюс буквы от A до F, означающие шестнадцатеричные "цифры" 10, 11, 12, 13, 14 и 15. Например, шестнадцатеричное число 10 равно числу 16 в десятичной системе. Поскольку эти две системы счисления (шестнадцатеричная и восьмеричная) используются в программах довольно часто, в языках C и C++ разрешено задавать целые константы не в десятичной, а в шестнадцатеричной или восьмеричной системе. Шестнадцатеричная константа должна начинаться с префикса 0x (нуль и буква x) или 0X, а восьмеричная — с нуля. Приведем два примера.

```
int hex = 0x80; // 128 в десятичной системе
int oct = 012;  // 10 в десятичной системе
```

Строковые константы

Языки C и C++ поддерживают еще один встроенный тип данных, именуемый *строковым*. Строка — это набор символов, заключенных в двойные кавычки, например "это тест". Не следует путать строки с символами. Символьная константа заключается в одинарные кавычки, например 'a'. Однако "a" — это

уже строка, содержащая только одну букву. Строковые константы при компиляции автоматически завершаются нулевым символом. Кроме того, в языке C++ поддерживается класс `string`, который описан ниже.

Логические (булевы) константы

В языке C++ определены две булевы константы: `true` и `false`.

В версии C99, которая обогатила язык C типом `_Bool`, тем не менее, не определена ни одна встроенная логическая константа. Однако, если в программу включить заголовок `<stdbool.h>`, будут определены макросы `true` и `false`. Кроме того, после включения в программу заголовка `<stdbool.h>` определяется макрос `bool` как еще одно имя для типа `_Bool`. Это позволяет создать программу, совместимую как с версией C99, так и с языком C++. Однако не забывайте, что в версии C89 логический тип не определен вообще.

Комплексные константы

Если при использовании версии C99 в программу включить заголовок `<complex.h>`, будут определены следующие константы, позволяющие работать с комплексными числами.

`_Complex_I`	`(const float _Complex)`*i*
`_Imaginary_I`	`(const float _Imaginary)`*i*
`I`	`_Imaginary_I` (или `_Complex_I`, если мнимые типы не поддерживаются)

Здесь элемент *i* представляет мнимое значение, которое равно квадратному корню из −1.

Специальные (управляющие) символьные константы

С выводом большинства печатаемых символов прекрасно справляются символьные константы, заключенные в одинарные кавычки, но есть такие “экземпляры” (например, символ возврата каретки), которые невозможно ввести в исходный текст программы с клавиатуры. Поэтому в языках C и C++ разрешено использовать ряд специальных символьных констант (включающих символ “обратная косая черта”), которые также называются *управляющими последовательностями*. Приведем список этих констант.

Код	Значение
\b	Возврат на одну позицию
\f	Подача страницы (для перехода к началу следующей страницы)
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\"	Двойная кавычка
\'	Одинарная кавычка (апостроф)
\\	Обратная косая черта
\v	Вертикальная табуляция
\a	Звуковой сигнал (звонок)
N	Восьмеричная константа (где *N* — это сама восьмеричная константа)
\x*N*	Шестнадцатеричная константа (где *N* — это сама шестнадцатеричная константа)
\?	Вопросительный знак

Специальные константы можно использовать везде, где уместно использование символов. Например, следующая инструкция выполняет переход на новую строку, выводит символ табуляции, а затем строку "Это тест".

```
cout << "\n\tЭто тест";
```