2. Типы, операторы и выражения

Переменные и константы являются основными объектами данных, с которыми имеет дело программа. Переменные перечисляются в объявлениях, где устанавливаются их типы и, возможно, начальные значения. Операции определяют действия, которые совершаются с этими переменными. Выражения комбинируют переменные и константы для получения новых значений, Тип объекта определяет множество значений, которые этот объект может принимать, и операций, которые над ними могут выполняться. Названные "кирпичики" и будут предметом обсуждения в этой главе.

Стандартом ANSI было утверждено значительное число небольших изменений и добавлений к основным типам и выражениям. Любой целочисленный тип теперь может быть со знаком, `signed`, и без знака, `unsigned`. Предусмотрен способ записи беззнаковых констант и шестнадцатеричных символьных констант. Операции с плавающей точкой допускаются теперь и с одинарной точностью. Введен тип `long double`, обеспечивающий повышенную точность. Строковые константы конкатенируются ("склеиваются") теперь во время компиляции. Частью языка стали перечисления (`enum`), формализующие для типа установку диапазона значений. Объекты для защиты их от каких-либо изменений разрешено помечать как `const`. В связи с введением новых типов расширены правила автоматического преобразования из одного арифметического типа в другой.

2.1. Имена переменных

Хотя мы ничего не говорили об этом в главе 1, но существуют некоторые ограничения на задание имен переменных и именованных констант. Имена составляются из букв и цифр; первым символом должна быть буква. Символ подчеркивания "_" считается буквой; его иногда удобно использовать, чтобы улучшить восприятие длинных имен переменных. Не начинайте имена переменных с подчеркивания, так как многие переменные библиотечных программ начинаются именно с этого знака. Большие (прописные) и малые (строчные) буквы различаются, так что x и X — это два разных имени. Обычно в программах на Си малыми буквами набирают переменные, а большими — именованные константы.

Для внутренних имен значимыми являются первые 31 символ[4]. Для имен функций и внешних переменных число значимых символов может быть меньше 31, так как эти имена обрабатываются ассемблерами и загрузчиками и языком не контролируются. Уникальность внешних имен гарантируется только в пределах 6 символов, набранных безразлично в каком регистре. Ключевые слова `if`, `else`, `int`, `float` и т. д. зарезервированы, и их нельзя использовать в качестве имен переменных. Все они набираются на нижнем регистре (т. е. малыми буквами).

Разумно давать переменным осмысленные имена в соответствии с их назначением, причем такие, чтобы их было трудно спутать друг с другом. Мы предпочитаем короткие имена для локальных переменных, особенно для счетчиков циклов, и более длинные для внешних переменных.

2.2. Типы и размеры данных

В Си существует всего лишь несколько базовых типов:

`char` — единичный байт, который может содержать один символ из допустимого символьного набора;

`int` — целое, обычно отображающее естественное представление целых в машине;

`float` — число с плавающей точкой одинарной точности;

`double` — число с плавающей точкой двойной точности.

---

[4] Это, как и другие ограничения на длину имен, является минимальными требованиями стандарта. Компиляторы могут поддерживать и имена большей длины. — *Примеч. корр.*

Имеется также несколько квалификаторов, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы `short` (короткий) и `long` (длинный) применяются к целым:

```
short int sh;
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. Чаще всего для представления целого, описанного с квалификатором `short`, отводится 16 битов, с квалификатором `long` — 32 бита, а значению типа `int` — или 16, или 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`.

Квалификаторы `signed` (со знаком) или `unsigned` (без знака) можно применять к типу `char` и любому целочисленному типу. Значения `unsigned` всегда положительны или равны нулю и подчиняются законам арифметики по модулю 2^n, где n — количество битов в представлении типа. Так, если значению `char` отводится 8 битов, то `unsigned char` имеет значения в диапазоне от 0 до 255, а `signed char` — от -128 до 127 (в машине с двоичным дополнительным кодом). Являются ли значения типа просто `char` знаковыми или беззнаковыми, зависит от реализации, но в любом случае коды печатаемых символов положительны.

Тип `long double` предназначен для арифметики с плавающей точкой повышенной точности. Как и в случае целых, размеры объектов с плавающей точкой зависят от реализации; `float`, `double` и `long double` могут представляться одним размером, а могут двумя или тремя разными размерами.

Именованные константы для всех размеров вместе с другими характеристиками машины и компилятора содержатся в стандартных заголовочных файлах `<limits.h>` и `<float.h>` (см. приложение B).

Упражнение 2.1. Напишите программу, которая будет выдавать диапазоны значений типов `char`, `short`, `int` и `long`, описанных как `signed` и как `unsigned`, с помощью печати соответствующих значений из стандартных заголовочных файлов и путем прямого вычисления. Определите диапазоны чисел с плавающей точкой различных типов. Вычислить эти диапазоны сложнее.

2.3. Константы

Целая константа, например 1234, имеет тип `int`. Константа типа `long` завершается буквой `l` или `L`, например `123456789L`; слишком большое целое, которое невозможно представить как `int`, будет представлено как `long`. Беззнаковые константы заканчиваются буквой `u` или `U`, а окончание `ul` или `UL` говорит о том, что тип константы `unsigned long`.

Константы с плавающей точкой имеют десятичную точку (`123.4`), или экспоненциальную часть (`1e-2`), или же и то и другое. Если у них нет окончания, считается, что они принадлежат к типу `double`. Окончание `f` или `F` указывает на тип `float`, а `l` или `L` — на тип `long double`.

Целое значение помимо десятичного может иметь восьмеричное или шестнадцатеричное представление. Если константа начинается с нуля, то она представлена в восьмеричном виде, если с `0x` или с `0X`, то — в шестнадцатеричном. Например, десятичное целое 31 можно записать как `037` или как `0X1F`. Записи восьмеричной и шестнадцатеричной констант могут завершаться буквой `L` (для указания на тип `long`) и `U` (если нужно показать, что константа беззнаковая). Например, константа `0XFUL` имеет значение `15` и тип `unsigned long`.

Символьная константа есть целое, записанное в виде символа, обрамленного одиночными кавычками, например `'x'`. Значением символьной константы является числовой код символа из набора символов на данной машине. Например, символьная константа `'0'` в кодировке ASCII имеет значение 48, которое никакого отношения к числовому значению 0 не имеет. Когда мы пишем `'0'` , а не какое-то значение (например, 48), зависящее от способа кодировки, мы делаем программу независимой от частного значения кода, к тому же она и легче читается. Символьные константы могут участвовать в операциях над числами точно так же, как и любые другие целые, хотя чаще они используются для сравнения с другими символами.

Некоторые символы в символьных и строковых константах записываются с помощью эскейп-последовательностей, например `\n` (символ новой строки); такие последовательности изображаются двумя символами, но обозначают один. Кроме того, произвольный восьмеричный код можно задать в виде

`'\ooo'`

где *ooo* — одна, две или три восьмеричные цифры (0…7) или

`'\xhh'`

где *hh* — одна, две или более шестнадцатеричные цифры (0…9, a…f, A…F). Таким образом, мы могли бы написать

```
#define VTAB '\013' /* вертикальная табуляция в ASCII */
#define BELL '\007' /* звонок В ASCII */
```

или в шестнадцатеричном виде:

```
#define VTAB '\xb' /* вертикальная табуляция в ASCII
*/ #define BELL '\x7' /* звонок в ASCII */
```

Полный набор эскейп-последовательностей таков:

`\a`	сигнал-звонок	`\\`	обратная наклонная черта
`\b`	возврат-на-шаг (забой)	`\?`	знак вопроса
`\f`	перевод-страницы	`\'`	одиночная кавычка
`\n`	новая-строка	`\"`	двойная кавычка
`\r`	возврат-каретки	`\ooo`	восьмеричный код
`\t`	горизонтальная-табуляция	`\xhh`	шестнадцатеричный код
`\v`	вертикальная-табуляция		

Символьная константа `'\0'` — это символ с нулевым значением, так называемый символ `null`. Вместо просто 0 часто используют запись `'\0'`, чтобы подчеркнуть символьную природу выражения, хотя и в том и другом случае запись обозначает нуль.

Константные выражения — это выражения, оперирующие только с константами. Такие выражения вычисляются во время компиляции, а не во время выполнения, и поэтому их можно использовать в любом месте, где допустимы константы, как, например, в

```
#define MAXLINE 1000
char line[MAXLINE+1];
```

или в

```
#define LEAP 1 /* in leap years - в високосные годы */
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

Строковая константа, или *строковый литерал*, — это нуль или более символов, заключенных в двойные кавычки, как, например,

```
"Я строковая константа"
```

или

```
"" /* пустая строка */
```

Кавычки не входят в строку, а служат только ее ограничителями. Так же, как и в символьные константы, в строки можно включать эскейп-последовательности; `\"`, например, представляет собой двойную кавычку. Строковые константы можно конкатенировать ("склеивать") во время компиляции; например, запись двух строк

```
"Здравствуй," " мир!"
```

эквивалентна записи одной следующей строки:

```
"Здравствуй, мир!"
```

Указанное свойство позволяет разбивать длинные строки на части и располагать эти части на отдельных строчках.

Фактически строковая константа — это массив символов. Во внутреннем представлении строки в конце обязательно присутствует нулевой символ `'\0'`, поэтому памяти для строки требуется на один байт больше, чем число символов, расположенных между двойными кавычками. Это означает, что на длину задаваемой строки нет ограничения, но чтобы определить ее длину, требуется просмотреть всю строку. Функция `strlen(s)` вычисляет длину строки `s` без учета завершающего ее символа `'\0'`. Ниже приводится наша версия этой функции:

```
/* strlen: возвращает длину строки s
*/ int strlen(char s[])
{
    int i;
    i = 0;
    while (s[i] !=
        '\0') ++i;
    return i;
}
```

Функция `strlen` и некоторые другие, применяемые к строкам, описаны в стандартном заголовочном файле `<string.h>`.

Будьте внимательны и помните, что символьная константа и строка, содержащая один символ, не одно и то же: `'x'` не то же самое, что `"x"`. Запись `'x'` обозначает целое значение, равное коду буквы x из стандартного символьного набора, а запись `"x"` - массив символов, который содержит один символ (букву x) и `'\0'`.

В Си имеется еще один вид константы — *константа перечисления*. Перечисление — это список целых констант, как, например, в

```
enum boolean { NO, YES };
```

Первое имя в enum[5] имеет значение 0, следующее — 1, и т. д. (если для значений констант не было явных спецификаций). Если не все значения специфицированы, то они продолжают прогрессию, начиная от последнего специфицированного значения, как в следующих двух примерах:

```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t',
    NEWLINE = '\n', VTAB = '\v', RETURN = '\r' };
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
    JUL, AUG, SEP, OCT, NOV, DEC };
    /* FEB есть 2, MAR есть 3 и т.д. */
```

Имена в различных перечислениях должны отличаться друг от друга. Значения внутри одного перечисления могут совпадать.

Средство enum обеспечивает удобный способ присвоить константам имена, причем в отличие от #define значения констант при этом способе могут генерироваться автоматически. Хотя разрешается объявлять переменные типа enum, однако компилятор не обязан контролировать, входят ли присваиваемые этим переменным значения в их тип. Но сама возможность такой проверки часто делает enum лучше, чем #define. Кроме того, отладчик получает возможность печатать значения переменных типа enum в символьном виде.

2.4. Объявления

Все переменные должны быть объявлены раньше, чем будут использоваться, при этом некоторые объявления могут быть получены неявно — из контекста. Объявление специфицирует тип и содержит список из одной или нескольких переменных этого типа, как, например, в

```
int lower, upper, step;
char c, line [1000];
```

Переменные можно распределять по объявлениям произвольным образом, так что указанные выше списки можно записать и в следующем виде:

```
int  lower;
int  upper;
int   step;
char c;
char line[1000];
```

Последняя форма записи занимает больше места, тем не менее, она лучше, поскольку позволяет добавлять к каждому объявлению комментарий. Кроме того, она более удобна для последующих модификаций.

В своем объявлении переменная может быть инициализирована, как, например:

```
char esc = '\\';
int i = 0;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

Инициализация неавтоматической переменной осуществляется только один раз — перед тем, как программа начнет выполняться, при этом начальное значение должно быть константным выражением. Явно инициализируемая автоматическая переменная получает начальное значение каждый раз при входе в функцию или блок, ее начальным значением может быть любое выражение. Внешние и статические переменные по умолчанию получают нулевые значения. Автоматические переменные, явным образом не инициализированные, содержат неопределенные значения ("мусор").

---

[5] От английского слова *enumeration* — перечисление. — *Примеч. ред.*

К любой переменной в объявлении может быть применен квалификатор `const` для указания того, что ее значение далее не будет изменяться.

```
const double e = 2.71828182845905;
const char msg[] = "предупреждение: ";
```

Применительно к массиву квалификатор `const` указывает на то, что ни один из его элементов не будет меняться. Указание `const` можно также применять к аргументу-массиву, чтобы сообщить, что функция не изменяет этот массив:

```
int strlen(const char[] );
```

Реакция на попытку изменить переменную, помеченную квалификатором `const`, зависит от реализации компилятора.