Часть I

Императивное программирование

Глава 1

Элементы программ

В этой главе...

1.1. Алфавит языка

Изучение любого иностранного языка начинается с алфавита. Именно из элементов алфавита складываются слова, конструкции и выражения, которые в итоге образуют осмысленный текст. Применим этот принцип и к языку C++.

Стандарт языка C++ разделяет алфавит на четыре части: *основной набор текстовых символов* (basic source character set), *множество универсальных символов* (universal character names), *основной набор управляющих символов* (basic execution character set) и *основной набор расширенных управляющих символов* (basic execution wide-character set). В свою очередь, на их основе создается *набор управляющих символов* (execution character set) и *набор расширенных управляющих символов* (basic execution wide-character set). Рассмотрим каждый из них подробнее.

Основной набор текстовых символов состоит из 96 символов: пробела, четырех управляющих символов (горизонтальная табуляция (`\t`), вертикальная табуляция (`\v`), перевод страницы (`\f`) и переход на новую строку (`\n`)) и символов ASCII (91 символ), а именно:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '
```

Как правило, этих символов вполне достаточно для создания практически любой программы. Однако иногда, учитывая национальные особенности, возникает необходимость расширить выбор символов. Вдруг вам захочется при наименовании идентификаторов воспользоваться русскими буквами! Для этого пригодится набор универсальных символов.

Универсальные символы кодируются последовательностями из четырех или восьми шестнадцатеричных цифр, которые называются *универсальными именами* (universal-name-symbol).

```
/Uhhhhhhhhh
/uhhhh
```

Здесь символ `h` обозначает шестнадцатеричную цифру.

Кроме символов, которые используются для кодирования элементов программ (идентификаторов, комментариев, литералов и т.п.), в языке C++ есть так называемые *управляющие символы*, с помощью которых можно выполнять небольшое количество операций, связанных, как правило, с вводом и выводом данных.

В *основной набор управляющих символов* входит основной набор текстовых символов, к которому добавлены символы сигнала (`\a`), возврата на одну позицию (`\b`), возврата в начало строки (`\r`) и нулевого символа (`\0`).

Основной набор расширенных управляющих символов состоит из основного набора текстовых символов, закодированных с помощью универсальных имен, к которому добавлены символы сигнала (`\a`), возврата на одну позицию (`\b`), возврата в начало строки (`\r`) и расширенного нулевого символа (`\u0000`).

На основе этих двух наборов формируется *набор управляющих символов* и *набор расширенных управляющих символов*, которые содержат буквы национальных алфавитов и управляющие символы. Их реализация зависит от конкретного компилятора.

В языке C++ для кодирования некоторых символов используется триграфы. Они применяются, когда выбор символов очень ограничен (например, 127 ASCII-символов). В этом случае кодируемый символ представляется в виде совокупности трех символов из основного набора текстовых символов, причем первые два символа — знаки вопроса. Например, триграф `??=` кодирует символ `#`. Заметим также, что обычно для кодирования программы используются ключевые слова и диграфы, т.е. двухзнаковые последовательности. Например, диграф `%:` кодирует символ `#`.

К сожалению, применение триграфов снижает наглядность программ. Одно дело, когда видишь выражение `a[i]`, и совсем другое, если оно закодировано на "тарабарском" языке: `a??(i??)`. К счастью, в языке C++ не так много триграфов. Все они перечислены в табл. 1.1.

Таблица 1.1. Триграфы

Триграф	*Подстановка*
??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

1.2. Ключевые слова

Дадим несколько необходимых определений. *Имя* — это строка символов, которая используется для обозначения какого-либо элемента программы (переменной, константы, функции, класса и т.п.). *Ключевым словом* называется строка символов, имеющая особое значение лишь в определенных ситуациях. В остальных случаях ключевые слова не отличаются от имен. Например, в ранних версиях языка FORTRAN слово REAL могло использоваться и как ключевое, определяя тип данных, и как имя переменной. *Зарезервированным словом* именуется строка символов, которую ни при каких условиях нельзя использовать как имя. Как видим, понятие ключевого слова является более широким. Однако в языке C++ под ключевым словом обычно понимают именно зарезервированное, хотя это и не соответствует их точному определению.

Стандарт языка C++ содержит 63 ключевых слова.

```
asm          do             if                 return       typedef
auto         double         inline             short        typeid
bool         dynamic_cast   int                signed       typename
break        else           long               sizeof       union
case         enum           mutable            static       unsigned
catch        explicit       namespace          static_cast  using
char         export         new                struct       virtual
class        extern         operator           switch       void
const        false          private            template     volatile
const_cast   float          protected          this         wchar_t
continue     for            public             throw        while
default      friend         register           true
delete       goto           reinterpret_cast   try
```

Ключевые слова можно разделить на следующие категории: операторы, спецификаторы, модификаторы и квалификаторы.

В языке C++ оператор — крайне многозначное слово. Как правило, им называют символы операций (+, * и т.д.). Кроме того, это слово традиционно используется для обозначения отдельных команд, выполняемых программой (например, оператор присваивания, оператор цикла и т.п.). В литературе по C++ можно встретить попытки устранить неоднозначность этого слова. Для этого слово “оператор” резервируют для обозначения операций, а команды программы называют “инструкциями”. Спор на эту тему неизбежно принимает схоластический характер — ведь перегрузка просто перемещается со слова “оператор” на слово “инструкция”, которое уже связано с понятием “машинная операция”. Впрочем, по контексту всегда можно определить, о чем идет речь — об инструкции программы или операции. В нашей книге мы продолжаем придерживаться исторически сложившейся традиции и называем оператором как обозначение операции, так и конкретную инструкцию программы. Читатели легко поймут, о чем идет речь в каждом конкретном высказывании. Существует и третье значение слова “оператор” — ключевое слово, являющееся основным элементом управляющей конструкции (оператор `if`, оператор `goto` и т.д.).

Ключевые слова `break`, `case`, `catch`, `continue`, `delete`, `do`, `else`, `export`, `for`, `goto`, `if`, `namespace`, `new`, `reinterpret_cast`, `return`, `static_cast`, `switch`, `throw`, `try`, `typedef`, `typeid`, `using` и `while` являются *операторами*.

Некоторые ключевые слова — это имена *типов*, т.е. диапазона значений, которые могут принимать переменные, и набора операций, которые можно выполнять над ними. Ключевые слова, определяющие тип и способ хранения переменной или функции, называются *спецификаторами*. К ним относятся *спецификаторы основных типов* `bool`, `char`, `double`, `float`, `int`, `long`, `short`, `signed`, `unsigned` и `void`, *спецификаторы сложных типов* `class`, `struct` и `union`, *спецификаторы хранения* `auto`, `extern`, `register` и `static`, *спецификаторы доступа* `public`, `private` и `protected`, *спецификаторы функций* `inline` и `virtual`, а также *спецификаторы дружественных классов и функций* `friend` и *спецификатор шаблонов* `template`.

В определенных ситуациях типы уточняются с помощью *модификаторов* `const` и `volatile`, именуемых также *cv-квалификаторами*. (Не ищите в этом названии зашифрованного смысла: *cv* — это простая аббревиатура `const` и `volatile`.)

Каждый компилятор расширяет этот набор ключевых слов собственными словами.

1.3. Идентификаторы

Идентификатор — это синоним слова "имя". Он связывается с конкретным элементом программы (именем, константой, функцией и т.п.) и в дальнейшем используется для их обозначения.

- Идентификаторы могут состоять только из букв, цифр и символа подчеркивания.
- Идентификаторы должны начинаться с буквы или символа подчеркивания.
- Ключевые слова не могут быть идентификаторами.
- В качестве идентификаторов нельзя использовать имена стандартных функций (`sin`, `cos` и т.п.).
- В стандарте языка C++ не установлена предельная длина идентификатора.
- Компилятор различает прописные и строчные буквы, входящие в состав идентификатора.

Прокомментируем каждое из этих ограничений.

То, что идентификатор должен состоять из букв и цифр, вполне естественно. Очень часто начинающие программисты называют переменные простейшими именами, вроде `a1`, `i2`, `f4`. В небольших программах с очевидным смыслом это вполне разумно. Однако стоит заметить, что довольно быстро они сами забывают, что именно имели в виду, когда писали идентификатор `k3`. Поэтому, когда авторы книг советуют своим читателям давать элементам программы осмысленные имена, имеются в виду целые куски фраз, например `sum_of_integer_numbers`. Однако здесь легко впасть в другую крайность, стараясь слишком подробно описать смысл идентификатора.

Еще одно затруднение связано с выбором внешнего вида идентификатора. Существуют три возможности: использовать для связки слов символ подчеркивания, "верблюжий горб" или так называемый венгерский стиль. Первую возможность мы

уже продемонстрировали. Это очень удобно и наглядно. Идентификатор "верблюжий горб" выглядит так: `sumOfIntegerNumbers`. Он начинается со строчной буквы, а каждое следующее слово начинается с прописной. Что выбрать — дело вкуса. Иногда люди плохо различают символы подчеркивания и пользуются второй возможностью, а иногда у них рябит в глазах от чередования прописных и строчных букв, и тогда они прибегают к символам подчеркивания.

Венгерский стиль состоит в том, что каждая переменная должна начинаться с буквы, указывающей на ее тип. Например, целочисленные переменные должны начинаться с буквы `i` (integer), а действительные переменные — с буквы `f` (float) или `d` (double). Это соглашение очень напоминает правила, принятые в языке FORTRAN, в котором все переменные, начинающиеся с букв I, J, K, L, M или N, по умолчанию считаются целочисленными. К венгерскому стилю можно отнести также привычку программистов начинать имена классов с буквы `T` (type).

Как правило, с подчеркивания начинаются ключевые слова, зарезервированные компиляторами. Это позволяет сразу узнать их среди остальных идентификаторов. В принципе, хороший идентификатор должен не только описывать назначение элемента программы, но и своим видом указывать на его природу. Проще говоря, по внешнему виду идентификатора программист должен легко распознавать переменные, константы, функции и классы. Например, Герб Саттер (Herb Sutter) предложил принять следующие соглашения.

- Классы, функции и перечислимые типы необходимо называть так, чтобы все слова, образующие их имена, начинались с прописной буквы.: `Stack`, `Queue`, `QueueWithPriority` и т.п.
- Имена переменных и перечислимых значений должны начинаться со строчной буквы, например: `stackA`, `queueB` и т.д.
- Имена переменных-членов должны заканчиваться символом подчеркивания: `x_`, `sum_` и т.п.

Кроме того, в программах на языках C и C++ уже давно негласно принято при наименовании переменных применять строчные буквы (`variable`), а при названии констант — прописные (`PI`). Это облегчает чтение программ, поскольку при этом не нужно возвращаться назад, уточняя смысл идентификатора.

Итак, если идентификатор начинается символом подчеркивания — перед вами ключевое слово, зарезервированное компилятором, а если имя переменной или функции завершается символом подчеркивания — перед вами член класса.

Следует подчеркнуть, что все сказанное о внешнем виде идентификатора — это не правила, а всего лишь общепринятые соглашения. Если хотите, можно применить собственный стиль, но, учитывая, что программирование в настоящее время приняло конвейерный и командный характер, это чревато недоразумениями. Например, Бьерн Страуструп (Bjarne Stroustrup) и Герб Саттер предпочитают использовать символ подчеркивания, Том Сван (Tom Svan) и многие другие авторы — стиль "верблюжий горб".

1.4. Литералы

Наряду с переменными неотъемлемой частью программ являются *литералы*, т.е. постоянные значения, например 3.141592 (число π) или 2.71828 (константа Эйлера — число *e*). Литерал может быть целочисленным, символьным, действительным, строковым и булевым.

Десятичный целочисленный литерал выглядит примерно так.

```
125   Целочисленный литерал
125u  Целочисленный литерал без знака
125U  Целочисленный литерал без знака
125l  Расширенный целочисленный литерал
125L  Расширенный целочисленный литерал
```

Кроме того, буквы, уточняющие тип литерала, можно комбинировать. Например, расширенный целочисленный литерал без знака можно задать следующим образом.

```
125ul
125UL
125uL
125Ul
```

Восьмеричный целочисленный литерал задается так: перед последовательностью восьмеричных цифр (`0-7`) ставится цифра `0`.

```
0125     Число 125
01       Число 1
031455   Число 31455
```

Шестнадцатеричный целочисленный литерал состоит из префикса `0x` и полубайтов, заданных шестнадцатеричными цифрами (`0-9`, `A-F`). Количество указываемых полубайтов может колебаться от одного до четырех.

```
0x125    Число 125
0x1      Число 1
0x3455   Число 3455
0x34567  Число 4567
0xf123   Число -3805
0xf123U  Число 3805
0xffff   Число -1
-0xffff  Число 1
0xffffU  Число -1
0xabcd   Число 21555
```

Обратите внимание на то, что при попытке указать больше четырех полубайтов произойдет усечение числа (отбрасываются старшие разряды), причем компилятор не выдает никаких предупреждений об этом. Например, литерал `0x34567` усекается до числа `4567`, и суффикс `L` не помогает снять это ограничение. Суффикс `U` на знак литерала не влияет. Знак восьмеричного или шестнадцатеричного литерала определяется только старшим битом и знаком, стоящим перед числом.

Действительный литерал состоит из десятичного числа, записанного в фиксированном или научном формате, за которым могут следовать буквы `f` или `F`, а также `l` или `L` (комбинировать буквы `f` и `l`, а также `F` и `L` запрещено). Суффикс f означает тип `float`, а `L` — `long double`. Они различаются лишь диапазонами изменения действительного числа и количеством знаков, которые можно считать достоверными.

```
5.0      Число 5.000000
6.123F   Число 6.123000
7.432f   Число 7.432000
```

```
5.0l     Число 5.000000
12.4569L Число 12.45690
15.05lf  Число 15.05000
```

Учтите, что больше шести знаков после десятичной точки для литерала типа `float` задавать не стоит — результат все равно будет содержать лишь шесть значащих цифр, причем последняя будет округлена, даже если указать суффикс `L`. Остальные цифры представляют собой просто мусор. Для литерала типа `double` после десятичной точки удается удержать девятнадцать знаков.

```
float x = 0.123456789             x = 0.123456791
float x = 0.123456789L            x = 0.123456791
double x = 0.123456789123456789L  x = 0.12345679123456781
double x = 1.234e+02              x = 123.4
double x = 0.12345678912E-02L     x = 0.0012345679120000008
```

Как видите, задавая литералы, нужно следить за типом переменной и помнить о количестве цифр после десятичной точки, которым можно доверять.

Символьные литералы очень широко распространены. Как правило, строковым литералом является символ ASCII, заключенный в одинарные кавычки, например `'a'`, а также некоторые управляющие символы, такие как `'\n'`. Расширенные литералы начинаются с буквы `L`.

```
'1'       Символ 1 (ASCII-код равен 49)
'A'       Символ A (ASCII-код равен 65)
'a'       Символ a (ASCII-код равен 97)
'\0'      Нулевой символ — признак конца строки
'\n'      Символ перехода на новую строку
L'b'      Расширенный литерал b
```

Управляющие символы, или *escape-последовательности*, играют особенно важную роль при выводе данных, поэтому перечислим их отдельно, указав соответствующие восьмеричный и шестнадцатеричный коды.

```
'\a'  '\07'  '\x07'    Звуковой сигнал
'\b'  '\08'  '\x08     Возврат на одну позицию назад
'\f'  '\14'  '\x0c     Прогон страницы
'\n'  '\12'  '\x0a'    Переход на новую строку
'\r'  '\15'  '\x0d'    Возврат каретки
'\t'  '\11'  '\x09'    Горизонтальная табуляция
'\v'  '\13'  '\x0b'    Вертикальная табуляция
'\\'  '\134' '\x5c'    Обратная косая черта
'\''  '\47'  '\x27'    Апостроф
'\"'  '\42'  '\x22'    Двойная кавычка
'\?'  '\77'  '\x77'    Знак вопроса
```

Каждая из escape-последовательностей представляет собой одиночный символ, хотя записывается с помощью обратной косой черты, а также букв и восьмеричных или шестнадцатеричных цифр. Их можно использовать и как элементы форматированного вывода, и как самостоятельные символы.

```
printf("Результат = %d \n",sum);
char a = '\x77'; // Знак вопроса
char b = '\42';  // Двойная кавычка
```

Строковый литерал представляет собой последовательность символов, заключенную в двойные кавычки (например, `"строковый литерал"`). Их можно комбинировать с символьными литералами, обеспечивая правильное форматирование текста.

```
printf("Строка \n Следующая строка");
```

Следует обратить внимание на одну полезную особенность строковых литералов — если два строковых литерала записать слитно, они автоматически объединяются в одно целое. Это свойство можно использовать для организации строк форматирования при вызове функции `printf`. Например, следующие два оператора эквивалентны.

```
printf("Первая часть строки ... "
       "вторая часть строки ... ");

printf("Первая часть строки ... вторая часть строки ...");
```

Булевыми литералами являются ключевые слова `true` и `false`.

1.5. Операторы и знаки пунктуации

Операторы в языке C++ обозначают различные операции: арифметические, логические, поразрядные и сравнения. Позднее мы рассмотрим их детально, а пока нас интересует лишь, как они обозначаются.

```
+       -       *       /       %       ^
!       =       <       >       +=      -=
^=      &=      |=      <<      >>      <<=
<=      >=      &&      ||      ++      --
( )     [ ]     new     delete  &       |
~       *=      /=      %=      >>=     ==
!=      ,       ->      ->*     .       .*
?:
```

Знаками пунктуации в языке C++ называются символы, которые имеют синтаксический и семантический смысл, но не обозначают самостоятельные операторы. К ним относятся круглые, квадратные и фигурные скобки, а также запятая. Следует подчеркнуть, что знаки пунктуации `()`, `[]` и `{}` должны использоваться исключительно парами. Нарушение баланса скобок является одной из наиболее грубых ошибок.