Глава 15

Циклы в языке C

В этой главе...

- Понятие цикла
- Повторение части программы с помощью оператора `for`
- Счет с помощью цикла
- Отображение таблицы ASCII с помощью цикла
- Как избежать зацикливания
- Прерывание цикла с помощью оператора `break`

Больше всего компьютеры любят повторять одни и те же действия. А люди? О, нет! Нет большего наказания, чем заставить ребенка написать 100 раз на классной доске одну и ту же фразу, например "Не следует хихикать, когда учитель с умным видом делает глупые ошибки". А компьютеры? Они нисколько не возражают. На самом деле они наслаждаются выполнением подобных заданий.

Наиболее ценным свойством программ является возможность принятия решений, после которого следует возможность повторять действия. Единственная проблема заключается в том, чтобы вовремя их остановить. А для этого, прежде чем перейти к тонкостям выполнения инструкции цикла, нужно знать, как работает инструкция `if`. Эта глава начинается со знакомства с выполнением цикла `for`, наиболее древней из команд, связанных с циклами.

> ✓ Более подробно условный оператор `if` описан в главах 12 "Могущественная команда if" и 14 "Логические выражения и ключевое слово if".
>
> ✓ Полезно просмотреть табл. 12.1 из главы 12 "Могущественная команда if", где сравниваются функции операторов `if` и `for`.

Понятие цикла

Циклом называется многократное выполнение одних и тех же действий. Например, чтобы программа сосчитала от 1, скажем, до миллиона, в ней нужно записать цикл. Цикл — это та часть кода программы, которая выполняется заданное количество раз.

В качестве примера цикла рассмотрим следующие действия, обычно выполняемые маленьким ребенком и мамой в процессе знакомства со столь удивительным предметом, как ложка:

```
Ребенок берет ложку.
Ребенок бросает ложку на пол; ребенок смеется.
Мама поднимает ложку с пола.
Мама кладет ложку перед ребенком.
Повторение.
```

Это подобно выполнению некоей примитивной программы. Более того, это можно даже рассматривать как программу, ведь фактически это последовательность инструкций! И в ней есть цикл. Это видно по слову “Повторение”, которое показывает, что определенная последовательность шагов повторяется.

К сожалению, в примитивной предыдущей программе отсутствует условие остановки. Это и есть так называемый бесконечный цикл. Его выполнение приводит к зацикливанию. В языках программирования большинство операторов цикла содержат условия, подобные тем, которые используются в условном операторе. Эти условия указывают, когда нужно прекратить повторения. Отсутствие таких условий нежелательно.

Небольшие изменения в программе могут помочь избежать зацикливания:

```
Повторять, пока мама не перестанет класть ложку перед ребенком.
```

Теперь в программе есть условие остановки цикла.

Цикл имеет три части:

- ✓ начало;
- ✓ средняя часть, которая повторяется;
- ✓ конец.

Итак, всякий цикл состоит из этих трех частей. Начало — это то место, где устанавливаются начальные значения, используемые для запуска цикла, обычно это некоторая команда языка программирования, которая говорит что-нибудь вроде следующего: “здесь начинается цикл — надо будет что-то повторять”. Средняя часть состоит из команд, которые повторяются много раз. Конец отмечает завершение повторяющейся части или условие окончания цикла, программисты его часто называют условием выхода из цикла. (В нашем примере: `пока мама не перестанет класть ложку перед ребенком`).

- ✓ В языке программирования C предусмотрено несколько различных типов циклов. Так, в языке программирования C предусмотрен не только цикл `for`, который рассматривается в данной главе, но и цикл с условием продолжения `while`, а также цикл с условием продолжения `do-while`. Кроме того, цикл можно организовать также с помощью уродливого ключевого слова `goto`, использовать которое не рекомендуется. Конечно, не исключено, что оно может встретиться где-нибудь в этой книге.
- ✓ Команды внутри цикла выполняются определенное количество раз или же до тех пор, пока не будет выполнено условие выхода из цикла. Например, можно сказать компьютеру: “выполняй это много раз” или же “выполняй это, пока бегунок не дойдет до конца”. В любом случае последовательность операций будет выполняться много раз.
- ✓ После выполнения (повторения) цикла выполнение программы будет продолжено. Но пока цикл не закончится, много раз повторяется выполнение одной и той же части программы.

Циклическое повторение без проблем

Ниже представлен исходный текст программы OUCH.C. Эта программа показывает, что компьютеру не составляет труда повторять выполнение одних и тех же команд много раз. Делает он это с удовольствием, без каких бы то ни было жалоб, причем так быстро, что вы даже не успеете не то что поджарить кукурузу, но даже открыть коробку с попкорном. (Я не говорю уже о том, чтобы устроить себе банкет с газированными напитками.)

В данной программе используется ключевое слово `for`. На языке C с помощью этого ключевого слова записывается одна из самых основных команд, которая заставляет компьютер выполнять цикл. В данной программе ключевое слово `for` позволяет создать небольшой цикл, в котором пять раз повторяется единственная команда `printf()`.

```
#include <stdio.h>

int main()                                  // главная функция
{
    int i;

    for(i=0 ; i<5 ; i=i+1)
    {
 // Ой! Пожалуйста, остановитесь!
        printf("Ouch! Please, stop!\n");
    }
    return(0);
}
```

Введите этот исходный текст с помощью вашего редактора.

В этой программе встречается новое ключевое слово `for`. Тщательно перепроверьте текст. Обратите внимание, что строка, которая начинается с `for`, не заканчивается точкой с запятой. Вместо этого ставятся фигурные скобки, точно так же, как и в условном операторе. (Фактически цикл выглядит точно так, как условный оператор, если игнорировать отличие в ключевых словах.) Сохраните файл под именем OUCH.C.

Скомпилируйте файл OUCH.C, используя ваш компилятор. Исправьте все ошибки. Не пропустите точки с запятой в скобках ключевого слова `for`.

Выполните полученную программу. Вы увидите на экране следующее:

```
Ouch! Please, stop!
Ouch! Please, stop!
Ouch! Please, stop!
Ouch! Please, stop!
Ouch! Please, stop!
```

(Ой! Пожалуйста, остановитесь!
Ой! Пожалуйста, остановитесь!
Ой! Пожалуйста, остановитесь!
Ой! Пожалуйста, остановитесь!)

Теперь убедились, что повторение не вредит компьютеру? Ни капельки!

- ✓ Цикл `for` имеет начало, середину и конец. Средняя часть — инструкция `printf()` — это та часть, которая повторяется. Остальная часть цикла, начало и конец, находится в круглых скобках ключевого слова `for`. (В следующем разделе выполнение цикла рассматривается более подробно.)
- ✓ Как и в случае с конструкцией `if`, когда только одна инструкция принадлежит команде `for`, все можно записать так:

  ```
  for(i=0 ; i<5 ; i=i+1)
   // Ой! Пожалуйста, остановитесь!
      printf("Ouch! Please, stop!\n");
  ```

  Фигурные скобки не обязательны, если нужно повторять только одну инструкцию. Но они необходимы, если требуется повторять несколько инструкций.
- ✓ В цикле `for` инструкция `printf()` повторяется пять раз.

- ✓ В цикле `for` скрыта следующая инструкция:

  `i=i+1`

  В языке программирования C так обозначается добавление 1 к переменной. Это действие называется *приращением*, и вы должны прочитать главу 11 “Больше математики и Священный порядок (старшинство операций)”, если вы не знаете, что это такое.
- ✓ Не стоит волноваться, если вы все еще не можете понять суть ключевого слова `for`. Надо много раз прочитать всю эту главу, постоянно бормоча “я пойму этот материал”. Затем при необходимости перечитайте все снова, причем начните с самого начала. Но сперва прочитайте всю главу до конца.

Для того чтобы сделать что-нибудь много раз, используйте ключевое слово for

Слово `for` является одним из тех слов, которое становится все более и более сверхъестественным и чудодейственным, чем больше вы произносите его; во всяком случае, в английском языке оно может означать для (for), поскольку (for), передний (fore), четыре (four), холл (foyer)... Как показывают следующие примеры, оно даже может быть ключевым в грамматической структуре предложения:

For he’s a jolly good fellow. (Поскольку он — веселый хороший товарищ.)
These are **for** your brother. (Эти штуковины для твоего брата.)
For why did you bring me here? (Зачем вы привели меня сюда?)
An eye **for** an eye. (Око за око.)
For the better to eat you with, my dear. (Чтобы быстрее тебя съесть, моя дорогая.)
Three **for** a dollar. (Три за доллар.)

И так далее. Но, пожалуй, лучше я прекращу перечислять примеры с этим словом. Потому что оно может раздражать. (**For** it can be maddening.)

В языке программирования C ключевое слово `for` используется для организации цикла, который, естественно, называется циклом `for`. Ключевое слово `for` определяет начальное состояние, условие окончания и команды, которые выполняются много раз во время повторения цикла. Формат может показаться немного сложным, поэтому сначала рассмотрим несколько упрощенный синтаксис цикла `for`.

```
for(начало; условие; приращение)     // для(начало; условие; приращение)
    инструкция;
```

После ключевого слова `for` открываются круглые скобки. В круглых скобках содержится три элемента цикла, отделенные двумя точками с запятой. Точка с запятой в конце строки, содержащей слово `for`, не ставится.

Первый элемент *начало* устанавливает начальное состояние цикла. Элемент *начало* обычно представляет собой инициализацию некоторой переменной ее начальным значением. В OUCH.C *начало* обычно представляет собой инструкцию `i=0`.

Второй элемент — *условие*. Цикл продолжает выполняться до тех пор, пока истинно условие, указанное вторым элементом цикла. (Иными словами, повторения продолжаются до тех пор, пока выполняется условие, указанное вторым элементом цикла.) Как правило, *условие* содержит операцию сравнения; оно записывается точно так же, как и условие в команде `if`. В OUCH.C это `i<5`.

Последний элемент, содержащийся в скобках ключевого слова `for` — *приращение*. В этом элементе указывается, что нужно сделать после очередного выполнения цикла. В OUCH.C на этот элемент возложена задача увеличения переменной `i` на 1: `i=i+1`.

Элемент *инструкция* представляет собой оператор. Он следует после ключевого слова `for`. (Иногда даже говорят, что *инструкция* принадлежит ключевому слову `for`.) Выполнение *инструкции* повторяется заданное число раз — до тех пор, пока продолжается выполнение цикла, заданного ключевым словом `for`. Эта *инструкция* должна заканчиваться точкой с запятой.

Если конструкции `for` принадлежит больше одной инструкции, вы должны загнать эти инструкции в фигурные скобки. Вот так:

```
for(начало; условие; приращение)
{
    инструкция;
    инструкция;
 /* и т.д. */
}
```

Обратите внимание, что инструкции являются дополнительными. Вы можете написать цикл `for` (для) и так:

```
for(начало; условие; приращение)
    ;
```

В этом случае единственная точка с запятой как раз и будет той "*инструкцией*", которая предназначена для повторения.

- ✓ Одна из самых больших трудностей, связанных с ключевым словом `for`, состоит в том, чтобы понять, что происходит с тремя элементами, заключенными в круглые скобки ключевого слова `for`. Честно говоря, этого не понимают не только новички, но и знатоки языка C, хотя они и не признаются в этом.
- ✓ Какая самая большая ошибка связана с циклом `for`? Использование запятых, а не точек с запятой. Внутри круглых скобок элементы разделяются точками с запятой!
- ✓ Некоторые программисты используют гибкость языка C и пропускают части *начало* и *приращение* в команде `for`. Хотя это вполне законно, я не рекомендую такой прием для новичков. Просто не падайте в обморок, если вы когда-либо увидите такое. Помните, что внутри круглых скобок ключевого слова `for` всегда должны быть обе точки с запятой.

Пошаговое выполнение программы OUCH.C

Я признаю, что цикл `for` — не самая простая команда языка C, предназначенная для выполнения циклов. Основная трудность состоит в понимании его частей. Конечно, было бы проще написать следующее

```
for(6)
{
 /* инструкции */
}
```

и затем повторить шесть раз инструкции, заключенные в фигурные скобки. Но такой вариант оператора `for` значительно уменьшает его возможности управления инструкциями.

Чтобы лучше понять, какой смысл имеют три части в круглых скобках цикла `for`, в качестве примера рассмотрим следующую инструкцию цикла `for` из программы OUCH.C:

```
for(i=0 ; i<5 ; i=i+1)
```

В цикле `for` переменная `i` используется для того, чтобы сосчитать количество повторений инструкции в этом цикле.

Первый элемент цикла `for` определяет начальное состояние. В данной строке программы начальное значение целочисленной переменной `i` устанавливается равным 0. Эта простая старая инструкция языка C присваивает переменной значение:

```
i=0
```

Внутри круглых скобок значение 0 проскальзывает в целую переменную `i`. Ничего больше не происходит.

Второй элемент — условие. Оно подобно тем условиям, которые используются в условных операторах. Условие представляет собой *условие продолжения выполнения* цикла `for`; цикл может продолжать повторять выполнение своих инструкций много раз, но только до тех пор, пока указанное в нем условие истинно. В предыдущем коде цикл продолжает повторяться до тех пор, пока истинно условие цикла, т.е. пока в результате вычисления инструкции `i<5` (значение переменной `i` меньше 5) получается логическое значение `true`. Условие здесь то же самое, что и в следующем условном операторе:

```
if (i<5)
```

Если значение переменной `i` меньше 5, продолжается выполнение цикла.

Последний элемент цикла `for` указывает, что сделать перед очередным повторением цикла. Без этого элемента цикл повторялся бы вечно: `i` равно 0, а цикл повторяется, пока `i<5` (значение `i` меньше 5). Это условие истинно, так что цикл `for` продолжался бы бесконечно, подобно федеральной программе субсидирования фермеров. Однако последний элемент цикла `for` увеличивает значение переменной `i` в конце очередного повторения цикла:

```
i=i+1
```

Компилятор берет значение переменной `i` и добавляет к нему 1 каждый раз, когда завершается очередное повторение цикла `for`. (Подробнее операция приращения рассмотрена в главе 11 “Больше математики и Священный порядок (старшинство операций)”.) Таким образом, циклу `for` удается повторить себя — и все инструкции, которые принадлежат ему — всего пять раз. В табл. 15.1 подробно описано выполнение цикла.

Таблица 15.1. Изменение значения переменной i в цикле `for`

Значение `i`	*Выполнено ли условие `i<5`?*	*Инструкция*	*Что сделать*
`i=0`	Да, продолжать выполнение цикла ←	`printf()...` (1)	`i=0+1`
`i=1`	Да, продолжать выполнение цикла ←	`printf()...` (2)	`i=1+1`
`i=2`	Да, продолжать выполнение цикла ←	`printf()...` (3)	`i=2+1`
`i=3`	Да, продолжать выполнение цикла ←	`printf()...` (4)	`i=3+1`
`i=4`	Да, продолжать выполнение цикла ←	`printf()...` (5)	`i=4+1`
`i=5`	Нет — остановиться немедленно!		

В начале табл. 15.1 значение переменной `i` равно 0, т.е. начальному значению, заданному в инструкции `for`. Затем вычисляется второй элемент — выполняется операция сравнения, т.е. проверяется условие. Верно ли неравенство `i<5`? Если да, цикл продолжается.

При выполнении цикла вычисляется третья часть инструкции `for`, и значение `i` увеличивается. Наряду с этим, выполняются любые инструкции, принадлежащие команде `for`. Когда выполнение всех этих инструкций закончено, снова выполняется операция сравнения `i<5` и цикл либо повторяется, либо останавливается — в зависимости от результата этой операции сравнения.

- ✓ Цикл `for` может быть громоздким, ведь он имеет так много частей! Поэтому в нем иногда трудно разобраться. Но, тем не менее, в языке программирования C это самый удобный способ записать повторное выполнение группы инструкций заданное количество раз.
- ✓ Третий элемент в круглых скобках инструкции `for` — *приращение* — выполняется только один раз при каждом повторении цикла. И это утверждение истинно независимо от количества инструкций, принадлежащих циклу, — их может быть одна или несколько или даже они могут отсутствовать вообще (в этом случае их количество равно нулю).

- ✓ Большинство новичков, знакомясь с циклом `for`, испытывают трудности, стремясь понять назначение второго элемента. Они помнят, что первый элемент означает "здесь начало цикла", но иногда они думают, что второй элемент означает "здесь конец цикла". Это не так! Второй элемент означает "продолжать выполнение цикла, пока заданное условие истинно". Этим второй элемент подобен операции сравнения в `if`. Компилятор, конечно, не подозревает о таких ляпах, но программа работает не так, как ожидает новичок. В результате, когда новичок делает эту ошибку, он думает, что программа не работает должным образом, хотя на самом деле ошибочно его представление о том, как она должна работать.
- ✓ Не забывайте объявить переменную, используемую в круглых скобках цикла `for`. Эту частую ошибку допускают почти все. Подробно объявление переменных рассмотрено в главе 8 "Переменные в языке C".

- ✓ Ниже в виде цикла `for` приведена заготовка, которую удобно использовать в программах. В следующей строке нужно только заменить прописную (большую) букву *X* числом — количеством выполнений цикла:

  ```
  for(i=1 ; i<=X ; i=i+1)
  ```

  Вы должны объявить, что `i` будет переменной целочисленного типа. Начальное ее значение равно 1, а конечное — значению *X*. Например, чтобы повторить цикл 100 раз, напишите следующую команду:

  ```
  for(i=1 ; i<=100 ; i=i+1)
  ```

Забава: считаем до 100

В этом разделе представлен исходный текст для программы 100.C. Эта программа использует цикл `for`, чтобы посчитать до 100 и отобразить каждое число на экране. И действительно, это — большое достижение: первые компьютеры могли считать только до 50, а затем они начинали делать дикие и часто ошибочные предположения о том, какое число должно быть следующим. (Посмотрите на ваш телефонный счет, и вы сразу поймете, что я имею в виду.) Основная часть программы 100.C подобна OUCH.C. На самом деле здесь этот пример приведен только потому, что цикл `for` многим новичкам кажется настолько странным, что одной программы мало для разъяснения всех тонкостей. Поэтому-то мы переходим к рассмотрению следующего примера программы:

```
#include <stdio.h>

int main()                              // главная функция
{
    int i;

    for(i=1 ; i<=100 ; i=i+1)
```

```
        printf("%d\t",i);
    return(0);
}
```

Введите этот исходный текст с помощью вашего редактора. Соблюдайте отступы; в лучших традициях языка C даже когда фигурные скобки опущены, нужно делать отступ, если одна инструкция принадлежит другой (как показано в примере).

В инструкции `for` переменной `i` сначала присваивается 1. Условие (второй элемент инструкции `for`) — `i<=100`. Благодаря этому условию цикл будет повторяться, пока значение `i` меньше или равно 100. Заключительная часть инструкции увеличивает значение `i` на 1 в конце каждого повторения цикла.

Инструкция `printf` отображает значение целой переменной `i`, используя метку-заполнитель `%d`. Escape-последовательность `\t` вставляет символ табуляции в вывод, выстраивая все выводимые числа в красивые, ровные, аккуратные столбцы.

Сохраните файл на диске под именем 100.C. (Это имя должно вам дважды напоминать о назначении программы, ведь римская цифра C — это 100. Вот так.)

Скомпилируйте программу и выполните ее. Вы увидите на экране числа от 1 до 100, расположенные в десяти аккуратных строках и десяти столбцах. Просто удивительно, как быстро компьютер может сделать это.

- ✓ При каждом из 100 повторений инструкции `printf()` в цикле `for` отображается значение переменной `i` и увеличивается значение этой переменной. (Таким образом, увеличение выполняется 100 раз.)
- ✓ Измените исходный текст 100.C. В инструкции `for` в заголовке цикла измените количество повторений так, чтобы цикл выполнялся 10 000 раз. Для этого с помощью редактора отредактируйте заголовок цикла так:

  ```
  for(i=1 ; i<=10000 ; i=i+1)
  ```

  Достаточно лишь вставить два дополнительных нуля после числа 100, которое уже имеется в заголовке цикла. Сохраните измененный файл на диске и перетранслируйте его. Компьютеру потребуется не на много больше времени, чтобы сосчитать до 10 000; однако ему понадобится гораздо больше времени, чтобы отобразить все числа от 1 до 10 000.

Поговорим о циклах!

Цикл — это одно из самых полезных средств программирования. Циклы в программах — это как роскошные, фантастические, достойные самых высоких призов сливочные украшения на вершинах холодных унылых гор салата из хрустящей капусты. Исключительно благодаря циклам компьютеры могут выполнять полезные действия. Поэтому не удивительно, что без цикла не обходится ни одна из следующих операций: сортировка, поиск, распечатка, организация задержки. И, конечно же, ни в одной полезной программе нельзя обойтись без цикла. В циклах — соль программирования.

Я откладываю полный обзор применения циклов до главы 17 “Познакомьтесь с циклом `while` (циклом с условием продолжения)”. Моя задача в этом разделе состоит в том, чтобы завершить это краткое (но все же несколько преждевременное) описание замечательного ключевого слова `for` и показать вам некоторые интересные средства — я называю их антициклами, — предназначенные для прерывания даже самой (сверх) прилежной компьютерной программы.

Полезная программа ASCII

Наконец, в этом разделе я приведу исходный текст программы ASCII.C. Несомненно, это первая полезная программа в книге. Программа ASCII.C отображает символы и их соответствующие коды из таблицы ASCII, начиная от кода 32 и до кода 127. Эта программа полезна потому, что из подсказки DOS вы можете легко напечатать таблицу ASCII, и вам не нужно будет постоянно искать коды в приложениях самых разных книг. (Программисту приходится искать коды ASCII почти ежечасно.)

```
#include <stdio.h>

int main()                              // главная функция
{
    unsigned char a;                    // символ без знака a;

    for(a=32;a<128;a=a+1)
        printf("%3d = '%c'\t",a,a);
    return(0);
}
```

Введите этот исходный текст с помощью вашего редактора. Этот исходный текст содержит основной цикл `for`, который повторяет инструкцию `printf()` несколько десятков раз.

После перепроверки исходного текста сохраните его на диске под именем ASCII.C.

С помощью компилятора скомпилируйте программу.

После запуска программы на экране вы увидите 5 столбцов, в которых будут отображены символы и их ASCII-коды, начиная с кода 32 (символ пробела) и заканчивая кодом 127, который выглядит как маленький домик, хотя на самом деле является греческой буквой дельта Δ.

- ✓ Эту программу я использую для быстрого поиска ASCII-кодов.
- ✓ В программе ASCII.C в цикле `for` начальное значение переменной `a` равно 32. Затем значение переменной `a` увеличивается (на 1) до значения 127 — последнего значения, при котором выполняется оператор печати, поскольку `a<128`. (После этого условие `a<128` выполнено не будет, и цикл остановится.) Приращение удобно выполнить с помощью инструкции `a=a+1`, расположив ее в круглых скобках ключевого слова `for`.
- ✓ Стоит отметить, что в функции `printf()` используется символьная переменная `a` дважды (сначала как целое число, а потом — как символ). Однако чтобы не писать инструкцию вывода дважды, в строке формата для `printf()` используются метки-заполнители `%d` и `%c`.
- ✓ Число 3 в метке-заполнителе `%3d` форматной строки функции `printf()` указывает, что `printf()` должна всегда отображать три символа, когда печатает целочисленное значение. Если ширину поля, используемого для отображения значения, установить равной трем символам, все значения (коды) будут выстроены в столбик, выровненный по правому краю. Благодаря этому все знаки разряда единиц окажутся друг под другом (поскольку перед двухцифровыми числами будет вставлен пробел).

- ✓ Вот теперь пришло время открыть вам страшную тайну: переменная `a` в ASCII.C может рассматриваться и как целое число, и как символьная переменная. В программе ее можно объявить как имеющую любой из этих типов. Дело в том, что в цикле `for` значение переменной `a` используется как целое число, а в функции

printf() это же значение используется и как символ, и как число. Эта двойственность справедлива и для типа int (для целых чисел), и для типа char (для символов), однако лишь для тех переменных, значения которых не превосходят 255 (это наибольшее "значение", которое можно хранить в символьной (типа char) переменной).

- ✓ Вот еще одна страшная тайна: переменная должна быть объявлена как символ без знака (unsigned char). Если в программе переменная объявлена как unsigned (без знака), ее значения не могут быть отрицательными числами. Если бы переменная была объявлена только как символьная переменная (типа char), цикл повторялся бы бесконечно; сложение 1 с a, когда значение a равняется 127, дает значение –127, и цикл просто повторялся бы вечно. (Чтобы доказать это, вырежьте слово unsigned (без знака) в исходном тексте ASCII.C, и перетранслируйте программу, после этого она будет выполняться вечно (без останова), а это совсем не то, что вам надо.)
- ✓ Разговор о выполнении цикла можно вести в цикле (бесконечном)...

Остерегайтесь бесконечных циклов!

Некоторые вещи вечны. Говорят, Любовь вечна. Бриллианты, конечно. Смерть и налоги — да, они тоже, к сожалению. Некоторые циклы также могут быть вечны, даже если вы не хотите, чтобы они были вечными. "Вечный" — слишком философский термин. Капризные программисты предпочитают термин "бесконечный". "Бесконечный" значит "продолжающийся вечно и никогда не останавливающийся". Да, это что-нибудь вроде кролика из рекламы "Energizer".

Бесконечный цикл — это повторяющийся раздел программы, притом повторяющийся бесконечно. Я не могу придумать никакого практического применения для такого цикла. Фактически бесконечный цикл — обычно несчастный случай в программировании. Это ошибка, которая может произойти при проектировании даже весьма необходимой программы. Часто о ее существовании программист даже не подозревает до выполнения программы. Только когда программа зациклится и ничего другого не делает или когда она снова и снова выплескивает на экран одни и те же символы, не подсказывая, как остановиться, вы понимаете, что вы создали бесконечный цикл. Увы, когда-нибудь такое случается со всеми.

Следующая программа — FOREVER.C — нарочно содержит бесконечный цикл. Вы можете ввести эту программу и испытать. Используя обман в команде for, программа повторяет инструкцию printf()до бесконечности:

```
#include <stdio.h>

int main()                                  // главная функция
{
    int i;

    for(i=1;i=5;i=i+1)
 // компьютер взбесился!
        printf("The computer has run amok!\n");
    return(0);
}
```

Введите этот исходный текст с помощью текстового редактора. Эта программа похожа на первую программу в этой главе, содержащую цикл for, — на программу OUCH.C. Различие находится в цикле for, точнее в той его части, которая проверяет необходимость повторения цикла, т.е. в условии. Кроме того, отличается и текст сообщения, отображение которого повторяется. Сохраните исходный текст на диске под именем FOREVER.C.

Скомпилируйте программу. Даже при том, что в инструкции `for` преднамеренно содержится бесконечный цикл, никакое сообщение об ошибках не отображается (если вы не попали впросак и напечатали что-нибудь неправильно). В конце концов, компилятор может думать, что вы пытаетесь делать что-то вечно, и эти вечно повторяемые действия являются частью вашего главного плана. Действительно, что бы это еще могло значить?

Когда вы будете выполнять программу, вы увидите, что на экране безумно, без останова будет пробегать следующее сообщение:

```
The computer has run amok!
```

(Компьютер взбесился!)

Действительно, это так и есть! Нажмите комбинацию клавиш <Ctrl+C>, чтобы остановить безумие.

- ✓ Большинство циклов проектируются так, чтобы сработало условие окончания цикла. Цикл, который зацикливается, либо не имеет условия, либо условие записано так, чтобы цикл не заканчивался. Это плохо.
- ✓ Бесконечные циклы очень коварны! Часто вы не можете обнаружить их до тех пор, пока программа не начнет выполняться. Именно поэтому нужно проверить каждую создаваемую программу.
- ✓ И в Windows, и в Unix для отмены команды печати на стандартное устройство вывода используется комбинация клавиш <Ctrl+C>. Как вы помните, команда печати на стандартный вывод выполняется в FOREVER.C много раз (бесконечно, если ее не остановить). И, как вы помните, именно комбинация клавиш <Ctrl+C> позволила отменить печать на стандартное устройство вывода, притом независимо от того, запустили вы программу в Windows или в Unix. Другие типы программ с бесконечными циклами, особенно те, которые ничего не печатают на стандартное устройство вывода, остановить намного тяжелее. Если <Ctrl+C> не помогает, часто приходится прибегать к специальным средствам конкретной операционной системы (например, к команде *kill*), чтобы *убить* взбесившуюся программу.
- ✓ В былые дни часто приходилось полностью перезапускать компьютер, чтобы восстановить контроль над ним, утерянный в результате выполнения взбесившейся программы, зацикливание которой произошло из-за выполнения бесконечного цикла.

- ✓ Программа зацикливается (выполняет бесконечный цикл) из-за ошибки в условии цикла `for`. Условие, как вы помните, — второй элемент в круглых скобках:

  ```
  for(i=1;i=5;i=i+1)
  ```

  Компилятор языка C видит `i=5` и рассуждает: “Хорошо, я помещу 5 в переменную `i`.” Но это — не операция сравнения, в результате которой получается истина или ложь. Ведь это совсем не похоже на обычное условие в условном операторе, — а именно его там и ожидал компилятор! Поэтому компилятор предполагает, что нужно вычислить записанное на месте условия выражение. В данном случае в результате вычисления условие оказывается истинным и потому выполнение цикла продолжается — фактически независимо ни от чего. Обратите внимание, что переменная `i` всегда равна 5 именно по причине вычисления “условия” `i=5`; ведь даже после того, как она увеличивается с помощью инструкции `i=i+1`, срабатывает инструкция `i=5`, и значение переменной `i` снова становится равным 5.

- ✓ Вот как, вероятно, инструкцию `for` хотел написать программист:

  `for(i=1;i<=5;i=i+1)`

  Эта строка повторяет цикл пять раз.
- ✓ Некоторые компиляторы иногда могут обнаружить условие бесконечного цикла в инструкции `for` и пометить такую инструкцию как бесконечный цикл. Если ваш компилятор это может, вам сильно повезло. Например, старый компилятор Borland C++ отметил в FOREVER.C возможное наличие бесконечного цикла так: `Possibly incorrect assignment error` (Возможная ошибка: неправильное присваивание). Однако компилятор все равно генерирует объектную (неправильную, конечно) программу.

Выход из цикла

Цикл не бесконечен, если есть способ прервать (`break`) его. Для большинства циклов условие выхода определяется в самой инструкции цикла. В инструкции `for` условие выхода определяется средним элементом цикла. Вот пример:

```
for(a=32;a<128;a=a+1)
```

Условие выхода в этом цикле определяется условием `a<128`; условие выхода из цикла — это условие, при выполнении которого цикл заканчивается.

Некоторые циклы, однако, разработаны так, что не имеют конца. Дело в том, что условие, при котором заканчивается цикл, вычисляется где-нибудь в другом месте. В этом случае цикл разрабатывается так, чтобы он выполнялся вечно, — и выглядит это прекрасно, если только некоторое условие в другом месте в конечном счете не прерывает цикл.

Приведу пример. Большинство текстовых процессоров представляют собой программы со скрытыми бесконечными циклами. Цикл много раз проверяет ввод с клавиатуры — он ждет, пока вы напечатаете какую-нибудь команду. Только когда вы напечатаете команду выхода из текстового процессора, эта команда остановит текстовый процессор, и вы возвратитесь в DOS. Это похоже на управляемый бесконечный цикл — на самом деле, конечно, не бесконечный, потому что из него есть выход.

Следующая программа — TYPER1.C, примитивный текстовый редактор. (Настолько примитивный, что он хоть и напоминает текстовый процессор, но не более чем кукла может напоминать живого человека.) В этом текстовом редакторе возможен лишь набор текста и его отображение на экране, и ничего более. В самой программе нарочно используется бесконечный цикл `for`. Чтобы выйти из цикла, когда пользователь нажимает клавишу <~> (тильда), используется ключевое слово `break` в команде `if`:

```
#include <stdio.h>

int main()                              // главная функция
{
    char ch;

    puts("Start typing");               // Запустить печать
 // Нажать <~>, затем <Enter>, чтобы остановиться
    puts("Press ~ then Enter to stop");
    for(;;)
    {
        ch=getchar();
        if(ch=='~')                     // если (ch == '~')
        {
            break;
        }
```

```
    }
    printf("Thanks!\n");                    // Спасибо!
    return(0);
}
```

Среди программ этой книги данная программа является чемпионом по количеству фигурных скобок и отступов. Будьте внимательны при вводе данной программы с помощью редактора. Сохраните файл на диске под именем TYPER1.C.

Скомпилируйте TYPER1.C.

Выполните полученную программу. Вы увидите, что данная программа работает подобно пишущей машинке. Вы можете напечатать довольно много символов и заполнить весь экран текстом.

Когда вы закончите печать, нажмите клавишу с тильдой (~), а затем — <Enter>. (Это единственная команда вашего "печатающего устройства" — программы TYPER.) Теперь вы действительно закончили.

- ✓ Инструкция `for(;;)` не является ошибочной. Дело в том, что точки с запятой обязательны, но все, что стоит между ними, можно опустить (это особенность ключевого слова `for`).
- ✓ Вслух я читаю команду `for(;;)` "for ever" ("вечно").
- ✓ Цикл `for` в TYPER1.C бесконечен: его условие отсутствует, а потому рассматривается как (вечно) истинное, поскольку проверять нечего. Отсутствие условия в цикле `for` ведь не является синтаксической ошибкой! Соответственно, компилятор предполагает, что условие истинно все время (т.е. всегда принимает значение `true` (истина)), и поэтому такой цикл в программе повторяется бесконечно.
- ✓ Первая и последняя части (элементы инструкции `for`) также отсутствуют, хотя точки с запятой в круглых скобках должны быть обязательно — таковы требования этикета в языке программирования C.
- ✓ Поскольку циклу `for` принадлежат несколько инструкций, они заключены в фигурные скобки.
- ✓ Функция `getchar` ждет нажатия клавиши на клавиатуре и отображает на экране символ, соответствующий нажатой клавише. Затем символ сохраняется в переменной `ch`.
- ✓ Условный оператор проверяет, равно ли значение переменной `ch` символу ~ (тильда). Обратите внимание, что используются два знака "=". Если в результате операции сравнения будет получено значение `true` (истина), то это укажет на то, что пользователь нажал клавишу ~, и будет выполнена команда `break`. В противном случае `break` пропускается, и цикл повторяется — с клавиатуры считывается еще один символ.

Ключевое слово break

Ключевое слово `break` языка C позволяет выйти из любого цикла, а не только из цикла `for`. Независимо от условия окончания цикла, `break` немедленно запускает команду "выйти из цикла". Программа продолжает выполняться со следующей инструкции после цикла:

```
break;                                  // прервать
```

Ключевое слово `break` само по себе является инструкцией языка C и потому после него обязательно должна стоять точка с запятой.

Ключевым словом `break` можно прервать не только бесконечный цикл, но и любой другой цикл. Иными словами, с помощью этого ключевого слова можно выйти из любого цикла, а не только избежать зацикливания. Инструкция `break` может остановить любой цикл в любое время, поэтому она часто используется в условном операторе, который проверяет некоторое условие. В зависимости от результата проверки цикл может быть остановлен с помощью `break`. (Именно так было сделано в программе TYPER1.C в предыдущем разделе.)

- ✓ Инструкция `break` останавливает только тот цикл, в котором она находится. Она не позволяет выйти из вложенного цикла, или цикла внутри другого цикла. Более подробно вложенные циклы рассматриваются в главе 18 “Циклы с условием продолжения. Организация задержки”.
- ✓ Если ключевое слово `break` встретится вне цикла, т.е. там, где нет ничего такого, из чего можно выйти, компилятор обидится и сгенерирует сообщение об ошибке.
- ✓ Забавно, что используется английское слово `break`, а не *brake* (тормоз, задерживать). Такая же несколько неожиданная логика, вероятно, применялась и тогда, когда давали название клавише прерывания на клавиатуре (клавише <Break>).