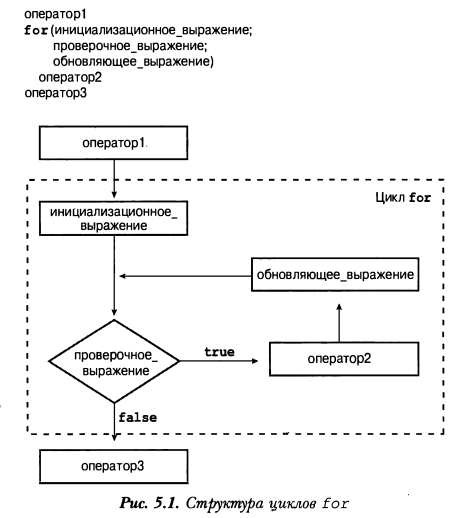
## Цикл For

**Листинг 5.1. forloop.cpp**

*// forloop.cpp -- представление цикла for*

*#include <iostream>*

*int main ()*

*{*

*using namespace std;*

*int i; // создание счетчика*

*// инициализация; проверка; обновление*

*for (i = 0; i < 5; i + + )*

*cout << "C + + knows loops.\n";*

*cout << "C++ knows when to stop.\n";*

*return 0;*

*}*

***Ниже показан вывод программы из листинга 5.1:***

*C++ knows loops.*

*C++ knows loops.*

*C++ knows loops.*

*C++ knows loops.*

*C++ knows loops.*

*C++ knows when to stop.*

Присваивание целочисленной переменной і значения 0:

*і = 0; // инициализации цикла*

*і < 5; // проверка цикла программы*

*і++ // операция инкремента (і = і + 1);*

*cout <"< "C++ knows loops.\n"; // тело программы*

**Обычно части цикла for выполняют следующие шаги.**

*1. Установка начального значения.*

*2. Выполнение, проверки условия для продолжения цикла.*

*3. Выполнение, действий цикла.*

*4. Обновление значения (значений), используемых в проверочном условии.*

*for (инициализация; проверочное выражение; обновляющее выражение)*

*тело*

**Листинг 5.2. num\_test.cpp**

*// num\_test.cpp -- использование числовой проверки в цикле*

*#include <iostream>*

*int main ()*

*{*

*using namespace*

*stakeout « "Enter the starting countdown value: "; // ввод начального значения счетчика*

*int limit;*

*cin » limit;*

*int i;*

*for (i = limit; i; i--) // завершается, когда і равно О-*

*cout « "i = " « i « "\n";*

*cout « "Done now that i = " « i « "\n"; // цикл завершен, вывод значения і*

*return 0;*

*}*

*Совет*

*В C++ принят стиль помещения пробелов между for и последующими скобками, а также*

*пропуск пробела между именем функции и следующими за ним скобками:*

*for (і = 6; і < 10; і++)*

*smart\_function(i);*

*Другие управляющие операторы, такие как if и while, трактуются аналогично for. Это служит ^ля визуального подчеркивания разницы между управляющим оператором и вызовом функции. К тому же общепринятая практика заключается в снабжении тела функции отступом, чтобы выделить его визуально.*

**Выражения и операторы**

Управляющий раздел for включает три выражения.

Выражения

*22 + 27*

*х = 20*

*maids = (cooks = 4) + 3;*

*х = у = z = 0;*

**Листинг 5.3. express. срр**

*// express.срр -- значения выражений*

*#include <iostream>*

*int main ()*

*{*

*using namespace std;*

*int x;*

*cout « "The expression x = 100 has the value ";*

*cout « (x = 100) « endl; // вывод значения*

*cout « "Now x = " « x « endl;*

*cout << "The expression x < 3 has the value ";*

*cout « (x < 3) << endl; // вывод значения*

*cout « "The expression x > 3 has the value ";*

*cout << (x > 3) << endl; // вывод значения*

*cout.setf(ios\_base::boolalpha);*

*cout << "The expression x < 3 has the value ";*

*cout « (x < 3) << endl; // вывод значения*

*cout << "The expression x > 3 has the value ";*

*cout « (x > 3) << endl;*

*return 0;*

*}*

*выражения х = 100*

*выражения х < 3*

*выражения х > 3*

*выражения х < 3*

*выражения х > 3*

Обычно cout преобразует значения bool в int перед тем, как отобразить их, но вызов функции cout. setf (ios: :boolalpha) устанавливает флаг, который инструктирует cout отображать true и false вместо 1 и 0.

От выражения до оператора программы один шаг; для этого достаточно добавить точку с запятой. То есть следующий код будет выражением:

*age = 100*

В то же время показанный ниже код является оператором:

*age = 100;*

Точнее, это оператор выражения. Любое выражение может стать оператором, если к нему добавить точку с запятой, но результат может не иметь смысла с точки зрения программы. Например, если rodents — переменная, то следующий код представляет

собой допустимый оператор C++:

rodents + 6; // допустимое, однако бессмысленное выражение

Язык C++ добавляет к циклам С возможность, которая требует некоторой поправки к синтаксису цикла for. Исходный синтаксис выглядел следующим образом:

*for {выражение; выражение; выражение)*

*оператор*

В частности, управляющий раздел конструкции for состоял из трех выражений, разделенных точками с запятой. Однако

циклы C++ позволяют поступать так, как показано ниже:

for (int i = 0; i < 5; i + + )

To есть в области инициализации цикла for можно объявить переменную. Часто поступать подобным образом очень удобно, но это не вписывается в исходный синтаксис, поскольку объявление не является выражением. Это единственное незаконное

поведение, которое подгонялось под правила за счет определения нового вида выражений — выражения оператора объявления, которое представляло собой объявление без точки с запятой и могло встречаться только в операторе for. Однако эта поправка была отброшена. Вместо нее решили модифицировать синтаксис оператора for:

*for (оператор-инициализации for условие; выражение)*

*оператор*

На первый взгляд это выглядит не совсем понятно, т.к. содержит только одну точку с запятой вместо двух. Но здесь все в порядке, поскольку оператор-инициализации-for идентифицируется как оператор, а оператор имеет собственную точку

с запятой. Что касается оператора оператор-инициализации-for, то он идентифицируется и как выражение-оператор, и как объявление. Это синтаксическое правило заменяет выражение с последующей точкой с запятой оператором, который имеет

собственную точку с запятой. Следствием этого является возможность для программистов C++ объявлять и инициализировать переменные внутри оператора цикла for, и они могут теперь выразить все, что нужно, с помощью синтаксиса C++. С объявлением переменной внутри оператор-инициализации-for связан один практический аспект, о котором вы должны знать. Такая переменная существует только внутри оператора for. To есть после того, как программа покидает цикл, переменная исчезает:

*for (int i = 0; i < 5; i + + )*

*cout << "C + + knows loop.\n";*

*cout « i << endl; // переменная і больше не определена*

Еще одна вещь, о которой следует знать — это то, что некоторые старые реализации C++ придерживаются старых правил и трактуют приведенный выше цикл, как если бы і была объявлена перед циклом, таким образом, делая ее доступной после

завершения цикла.

**Возврат к циклу for**

Давайте попробуем сделать что-то более сложное с помощью цикла. Код в листинге 5.4 использует цикл для вычисления и сохранения первых 16 факториалов.

Факториалы, которые являются удобным примером автоматизации обработки,

вычисляются следующим образом. Ноль факториал, записываемый как 0!, определен

как равный 1. Далее, 1! равен 1\*0!, т.е. 1. 2! равно 2\*1!, или 2. 3! равно 3\*2!, или 6,

и т.д. То есть факториал каждого целого числа равен произведению этого числа на

факториал предыдущего числа. В программе один цикл используется для вычисления

значений последовательных факториалов, с сохранением их в массиве. Второй цикл

служит для отображения результатов. Также программа демонстрирует применение

внешних объявлений для значений.

**Листинг 5.4. formore. срр**

*// formore.срр — дополнительные сведения о циклах for*

*#include <iostream>*

*const int ArSize =16; // пример внешнего объявления*

*int main()*

*{*

*long long factorials[ArSize];*

*factorials [1] = factorials[0] = ILL;*

*for (int i = 2; i < ArSize; i + + )*

*factorials[i] = i \* factorials[i-l];*

*for (i = 0; i < ArSize; i + + )*

*std::cout « i « " ! = " << factorials [i] « std::endl;*

*return 0;*

*}*

**Вывод программы из листинга 5.4 выглядит следующим образом:**

*0!*

*1!*

*2!*

*3!*

*4!*

*5!*

*6!*

*7!*

*8!*

*9!*

*10!*

*11!*

*12!*

*13*

*14*

*15*

*= 1*

*= 1*

*= 2*

*= 6*

*= 24*

*= 120*

*= 720*

*= 5040*

*= 40320*

*= 362880*

*! = 3628800*

*! = 39916800*

*! = 479001600*

*! = 6227020800*

*! = 87178291200*

*! = 1307674368000*

Как видите, факториалы растут очень быстро.

на заметку!

В этом листинге используется тип long long. Если он не доступен в вашей системе, можете воспользоваться типом double. Однако целочисленный формат дает более наглядное Й визуальное представление о том, насколько быстро растут значения.

Замечания по программе Программа из листинга 5.4 создает массив для значений факториалов. Элемент О хранит 0!, элемент 1 — 1! и т.д. Поскольку первые два факториала равны 1, программа присваивает первым двум элементам массива factorials значение ILL. (Вспомните, что первый элемент массива имеет индекс 0.) После этого в программе используется цикл для вычисления каждого факториала как произведения индекса на значение предыдущего факториала. Цикл иллюстрирует возможность применения счетчика цикла в его теле как переменной. Программа из листинга 5.4 демонстрирует, как цикл for работает рука об руку с массивами, предоставляя удобное средство доступа к каждому члену массива по очереди. К тому же в formore. cpp используется const для создания символического представления (ArSize) для размера массива. После этого ArSize применяется везде, где вступает в игру размер массива — в определении массива, а также в выражении, ограничивающем количество шагов циклов, обрабатывающих массив. Если теперь вы решите расширить программу для вычисления, скажем, 20 факториалов, для этого понадобится только установить ArSize в 20 и перекомпилировать программу. Благодаря использованию символической константы, вы избегаете необходимости

изменять индивидуально каждое вхождение 16 на 20.

На заметку!

Определение значения const для представления размера массива обычно всегда является щ хорошей идеей. Это значение const можно использовать в объявлении массива и во всех в других случаях ссылок на его размер, как, например, в циклах for. Ограничивающее выражение і < ArSize отражает тот факт, что индексы элементов массива лежат в пределах от 0 до ArSize - 1, т.е. значение индекса должно останавливаться за один шаг до достижения ArSize. Вместо него можно было бы

использовать проверочное условие і <= ArSize - 1, но оно выглядит менее изящно и не меняет сути проверки.

Обратите внимание, что в программе объявляется переменная ArSize типа const int вне тела функции main (). Как упоминалось в конце главы 4, это делает ArSize внешними данными. Объявление ArSize в подобной манере имеет два

последствия: ArSize существует на протяжении всего времени жизни программы, и все функции в файле программы могут использовать ArSize. В данном конкретном случае в программе присутствует только одна функция, поэтому внешнее объявление ArSize не имеет особого практического смысла. Однако программы с множеством функций часто выигрывают от совместного доступа к внешним константам, поэтому дальше мы еще попрактикуемся в их применении. Кроме того, в этом примере напоминается о том, что для доступа к выбранным стандартным именам можно использовать std: : вместо директивы using.

**Изменение шага цикла**

До сих пор в примерах циклов счетчик цикла увеличивался или уменьшался на единицу на каждом шаге. Это можно изменить, модифицировав обновляющее выражение. Программа в листинге 5.5, например, увеличивает счетчик цикла на величину введенного пользователем шага. Вместо применения і + + в качестве обновляющего выражения она использует выражение і = і + by, где by — выбранный пользователем шаг цикла.

**Листинг 5.5. bigstep. срр**

*// bigstep.срр — цикл указанным пользователем шагом*

*#include <iostream>*

*int main ()*

*{*

*using std::cout; // объявление using*

*using std::cin;*

*using std::endl;*

*cout « "Enter an integer: "; // ввод целого числа*

*int by;*

*cin » by;*

*cout « "Counting by " << by << "s:\n";*

*for (int i = 0; i < 100; i = i + by)*

*cout « i << endl;*

*return 0;*

*}*

**Ниже показан пример запуска программы из листинга 5.5:**

*Enter an integer: 17*

*Counting by 17s:*

*0*

*17*

*34*

*51*

*68*

*85*

Когда і достигает значения 102, цикл завершается. Главное, на что здесь нужно обратить внимание: в качестве обновляющего выражения можно использовать любое допустимое выражение. Например, если вы захотите на каждом шаге цикла возводить і в квадрат и прибавлять 10, можете воспользоваться выражением і = і \* і + 10. Другой момент, который следует отметить: часто лучше проверять на предмет неравенства, чем равенства. Например, проверка і == 100 в этом примере не подойдет, поскольку і перепрыгивает через значение 100. Наконец, в этом примере иллюстрируется применение объявлений using вместо директивы using.

**Доступ внутрь строк с помощью цикла for**

Цикл for предоставляет прямой способ доступа к каждому символу в строке, позволяет ввести строку и отобразить ее символ

за символом в обратном порядке. Метод size () класса string возвращает количество символов в строке; цикл использует это значение в выражении инициализации для установки і в индекс последнего символа строки, исключая нулевой символ. Для выполнения обратного отсчета в программе применяется операция декремента (—), уменьшающая

значение индекса массива на каждом шаге цикла. операцию сравнения "больше или равно" (>=), чтобы проверить, достигли цикл первого элемента.

**Листинг 5.6. f orstrl. срр**

*// forstrl.cpp -- использование цикла for для строки*

*#include <iostream>*

*#include <string>*

*int main ()*

*{*

*using namespace*

*stakeout << "Enter a word: ";*

*string word;*

*cin » word;*

*// Отображение символов в обратном порядке*

*for (int i = word, size () - 1; i >= 0; i--)*

*cout << word[i];*

*cout « "\nBye.\n";*

*return 0;*

*}*

**Ниже показан пример запуска программы из листинга 5.6**:

*Enter а word: animal*

*lamina*

*Bye.*

**Операции инкремента и декремента**

Язык C++ снабжен несколькими операциями, которые часто используются в циклах; операция инкремента (+ + ), которая получила отражение в самом названии C++, а также операция декремента (--). Эти операции выполняют два чрезвычайно часто встречающихся действия в циклах: увеличивают и уменьшают на единицу значение счетчика цикла.. Каждая

из них имеет два варианта.

**Листинг 5.7. plus\_one. срр**

*// plus\_one.cpp -- операция инкремента*

*#include <iostream>*

*int main()*

*{*

*using std::cout;*

*int a = 2 0;*

*int b = 20;*

*Циклы и выражения отношений 217*

*cout « "а = " « а « ": b = " « b « "\n";*

*cout « "a++ = " « a++ « " : ++b = " « ++b « "\n";*

*cout « "a = " « a « ": b = " « b « "\n";*

*return 0;*

*}*

Р**езультат выполнения этой программы показан ниже:**

*а =20: b = 20*

*а++ = 20: ++Ь = 21*

*а =21: b = 21*

Грубо говоря, нотация а++ означает "использовать текущее значение а при вычислении выражения, затем увеличить а на единицу". Аналогично, нотация ++а означает "сначала увеличить значение а на единицу, затем использовать новое значение при вычислении выражения".

int х = 5;

int у = ++х; // изменить х, затем присвоить его у // у равно 6, х равно 6

int z = 5;

int у = z++; // присвоить у, затем изменить z //у равно 5, z равно 6

Операции инкремента и декремента — симпатичные и компактные, но не стоит поддаваться соблазну и применять их к одному и тому же значению более одного раза в одном и том же операторе. Проблема в том, что при этом правила "использовать и изменить" и "изменить и использовать" становятся неоднозначными. То есть, следующий оператор в различных системах может дать совершенно разные результаты:

х = 2 \* х++ \* (3 - ++х) ; //не поступайте так

**Побочные эффекты и точки следования**

Давайте посмотрим внимательнее на то, что в C++ говорится, и что не говорится о том, когда операции инкремента вступают в силу. Для начала вспомните, что побочный эффект — это эффект, который проявляется, когда вычисление выражения приводит к модификации чего-либо, например, значения переменной. Точка следования (sequence point) — это точка при выполнении программы, где все побочные эффекты гарантированно будут завершены, прежде чем программа перейдет к следующему шагу. В C++ точка с запятой в операторе отмечает точку следования. Это значит, что все изменения, выполненные операциями присваивания, инкремента и декремента в операторе, должны произойти, прежде чем программа перейдет к следующему оператору. Некоторые операторы, рассматриваемые в последующих разделах, имеют

точки следования. К тому же конец любого полного выражения представляет точку следования. Что такое полное выражение? Это выражение, которое не является частью более крупного выражения. Примеры полных выражений включают часть выражения в операторе выражения (без точки с запятой), а также выражение, служащее проверочным условием в цикле while. Точки следования помогают прояснить, когда выполняется постфиксный инкремент. Рассмотрим, например, следующий код:

*while (guests++ < 10)*

*cout << guests << endl;*

**Сравнение префиксной и постфиксной форм**

Понятно, что разница между префиксной и постфиксной формами операций проявляется, если значение их операнда используется для каких-то целей — в качестве аргумента функции или для присваивания переменной. Но что если инкрементируе- мое или декрементируемое значение не используется? Например, отличаются ли

*х+ + ;*

*И*

*++х;*

друг от друга? Или же отличаются друг от друга

*for (n = lim; n > 0; —п)*

*и*

*for (n = lim; n > 0; п--)*

**Циклы и выражения отношений**

**Листинг 5.8. block. срр**

*// block.срр — использование блока*

*#include <iostream>*

*int main()*

*{*

*cout-« "The Amazing Accounto will sum and average ";*

*cout « "five numbers for you.\n";*

*cout « "Please enter five values:\n";*

*double number;*

*double sum = 0.0;*

*for (int i = 1; i <= 5; i + + )*

*{ // начало блока*

*cout « "Value " « i « " : "; . // ввод числа*

*cin >> number;*

*sum += number;*

*// конец блока*

*cout << "Five exquisite choices indeed! ";*

*cout << "They sum to " << sum << endl; // вывод суммы*

*cout « "and average to " « sum / 5 « " .\n"; // вывод среднего значения*

*cout « "The Amazing Accounto bids you adieu! \n";*

*return 0;*

}

**Ниже показан пример запуска программы из листинга 5.8:**

*The Amazing Accounto will sum and average five numbers for you.*

*Please enter five values:*

*Value 1: 1942*

*Value 2: 1948*

*Value 3: 1957*

*Value 4: 1974*

*Value 5: 1980*

*Five exquisite choices indeed! They sum to 9801*

*and average to 1960.2.*

*The Amazing Accounto bids you adieu!*