6. Операторы цикла

**Операторы цикла** используются для организации серии повторяющихся действий. Каждый отдельно взятый проход повторяющегося действия называется **шагом** или **итерацией**. Операторы, которые должны повторяться, называются **телом цикла**. Любой цикл должен когда-нибудь закончиться, иначе произойдёт **зацикливание программы** (бесконечный цикл). Отслеживание момента окончания цикла делается обычно с помощью **параметров цикла** (чаще всего одного, но бывает, что и нескольких), изменяющих своё значение (модифицирующихся) в процессе работы цикла. Соответственно, в цикле кроме основных операторов, предназначенных к повтору, могут участвовать параметры, их начальные установки и правила модификации, кроме того, всегда указывается условие продолжения работы цикла (при невыполнении которого цикл прекращается). На языке C и C++ имеются три варианта записи оператора цикла:

**1. Цикл с предусловием** имеет вид:

while (*условие*) *оператор*;

Здесь *условие* — это выражение логического или целого типа (оно определяет условие повторения тела цикла), *оператор* — это собственно тело цикла (либо одиночный оператор, либо серия операторов, объединённых в блок). Выполнение цикла начинается с вычисления *условия*. Если оно истинно (не равно 0), то выполняется *оператор*. И так делается перед каждым шагом. Если оно ложно (равно 0), то выполнение цикла завершается (происходит **выход из цикла**). Если оно ложно при первой же проверке, то тело цикла (*оператор*) не выполняется ни разу. Модификация параметров здесь входит в тело цикла и как бы сокрыто в *операторе*, который в данном случае чаще всего представляет собой блок. Начальные установки параметров делаются перед входом в цикл.

Блок-схема цикла while:

*условие*

*оператор*

true

false

выход из цикла

…

**2. Цикл с постусловием** имеет вид:

do *оператор* while (*условие*);

В этом цикле *условие* и *оператор* имеют тот же смысл, что и в цикле while, только проверка *условия* производится не перед шагом, а после него. Если *условие* истинно, то происходит переход к следующему шагу, если нет — то цикл завершается. Первый шаг в этом цикле производится всегда.

Блок-схема цикла do … while:

*оператор*

*условие*

false

true

…

выход из цикла

**3. Цикл с параметром** по сути представляет собой вариант записи цикла с предусловием:

for(*начальные\_установки*;

*условие*;

*модификация\_параметров*) *оператор*;

Здесь *начальные\_установки* используются для объявления и присвоения начальных значений величинам, используемым в цикле. Этот элемент может представлять собой не одно, а несколько действий, разделённых запятой (операцией «последовательное выполнение»). Но может и отсутствовать (когда сами начальные установки делаются перед началом цикла), однако точка с запятой в этом случай присутствовать обязана. *Условие* — это обязательный элемент, условие продолжения цикла. Элемент *модификация\_параметров* — это то, что исполняется после каждой итерации цикла и служит обычно для изменения параметров цикла. Может представлять собой несколько действий, разделённых запятой, и может отсутствовать — в случае, когда эта модификация записывается непосредственно в *операторе* (теле цикла).

*Замечание.* В случае отсутствия в скобках при for одного или двух элементов, наличие двух точек с запятой (обрамляющих *условие*) — обязательно.

Блок-схема цикла for:

*условие*

*оператор*

true

false

выход из цикла

…

*начальные\_установки*

*модификация\_параметров*

**Операторы досрочного завершения цикла.** Оператор break осуществляет немедленный выход из цикла, до возникновения «штатного» условия его завершения. Оператор continue прерывает текущий шаг цикла и переходит к следующему шагу (используется, когда тело цикла представляет собой составной оператор). Из соображений здравого смысла вытекает, что и break, и continue, должны производиться при выполнении некоторого условия.

*Примечание.* Среди программистов часто практикуется следующий способ организации циклов. Особенно тогда, когда проверку *условия* надо производить не до, не после, а в середине шага. Цикл делается как бы бесконечным, но внутри тела стоит оператор break, выполняемый при условии завершения цикла. Например, так:

while(1) {

*оператор*;

...

*оператор*;

if(*условие\_завершения*) break;

*оператор*;

...

*оператор*; }

**Пример 1.** Напечатать таблицу квадратных корней целых чисел в промежутке от 1 до 20 с шагом 1. Таблица должна выглядеть следующим образом:

x sqrt(x)

1 1.00000

2 1.41421

3 1.73205

..........

*x* выводится как целое число, колонка выравнивается по правому краю.  выводится с 1 знаком в целой части и 5 знаками в дробной части.

В этой программе отсутствует ввод исходных данных, а поскольку вывод требуется форматированный, удобно производить его в стиле C, с подключением файла stdio.h.

#include <stdio.h>

#include <math.h>  
int main() {

int x;

float y;

printf(” x sqrt(x)\n”);

printf(’\n’);

for(x=1; x<=20; x++) {

y=sqrt(x);

printf(”%2d %7.2f\n”,x,y);}

return 0; }

**Пример 2.** Даны целые *n* и *x*. Вычислить сумму

.

Для подсчёта суммы вводится переменная s, которой изначально присваивается значение 0. Затем организуется цикл с параметром k, на каждом шаге которого к s прибавляется очередное слагаемое.

#include <iostream>

using namespace std;

int main() {

int n,x; double s;

cout<<”задайте n,x\n”;

cin>>n>>x;

for(int k=2,s=0; k<=n; k++)

s+=(x-k)/(1.0+k\*k\*k);

cout<<”s=”<<s<<endl;

return 0; }

*Замечание 1.* В этой программе переменная k объявлена внутри цикла, соответственно, область её действия ограничена только этим циклом, и вне цикла она не видна. Обращаем также внимание на необходимость того, чтобы в дроби, стоящей в правой части оператора суммирования, присутствовал хотя бы один вещественный операнд, во избежание округления (в данном случае — обнуления) операции деления. Здесь это достигается написанием единицы в вещественном формате.

*Замечание 2.* По такому же принципу можно вычислять не только «длинные» суммы, но и «длинные» произведения. ***Упражнение:*** какое следует взять начальное значение для переменной, в которой будем подсчитывать произведение?

**Пример 3 (суммирование с точностью).** Требуется вычислить сумму тех же слагаемых, что и в Примере 2, только вместо *n* задаётся малое положительное вещественное число ε (степень точности). Суммировать надо до тех пор, пока очередное слагаемое по модулю не станет меньше ε (и это слагаемое в сумму войти уже не должно).

Для решения этой задачи применим цикл do … while. Для обозначения очередного слагаемого удобно будет ввести переменную *а*.

#include <iostream>

#include <math.h>

using namespace std;

int main() {

int x,k=2; double s=0,eps,a;

cout<<”задайте x,eps\n”;

cin>>x>>eps;

a=(x-2)/9.0;

do {

s+=a;

k++;

a=(x-k)/(1.0+k\*k\*k);}

while (fabs(a)>=eps);

cout<<”s=”<<s<<endl;

return 0; }

***Упражнение.*** Как можно узнать число шагов, которое произвёл этот цикл?

**Пример 4 — двойной цикл.** Даны *n, m.* Вычислить сумму  .

Суммирование просходит о двум независимым параметрам, следовательно, придется использовать конструкцию «**цикл в цикле**». Внутренний цикл считает сумму по *j* при фиксированном *i*, внешний — сумму по *i*.

#include <iostream>

using namespace std;

int main() {

int m,n; double s=0;

cout<<”введите m,n\n”;

cin>>m>>n;

for(int i=2;i<=m;i++)

for(int j=1;j<=n;j++) s+=1.0/(i\*i+j\*j\*j);

cout<<”s=”<<s<<endl;

return 0; }

**Пример 5. Использование рекуррентных формул.** Требуется при заданных *x* и *n* вычислить сумму

,

представляющую собой разложение в ряд Тейлора функции *ex*. Здесь каждое из слагаемых имеет сложную структуру — содержит факториал (*k*!=1∙2∙3∙…∙*k*, 0!=1), специальное вычисление которого представляет собой отдельный цикл. Чтобы избежать конструкции «цикл в цикле», для подобных задач рекомендуется использовать более оптимальную технологию, с участием рекуррентных формул (соотношений).

***Рекуррентным соотношением*** *называется формула, позволяющая выражать каждый член заданной последовательности через предыдущий (или — в общем случае — через несколько предыдущих).*

Для выведения рекуррентного соотношения требуется проделать некоторую предварительную работу перед написанием текста программы. В нашем примере это будет выглядеть следующим образом.

Обозначим *k*-й член суммы через :  .

Требуется вывести соотношение, позволяющее выразить  через . Для этого выпишем  (подставим в  *k*+1 вместо *k*):



Воспользуемся теперь свойством, что степени и факториалы сокращаются при делении. Разделим  на :



Следовательно, . Это и есть искомое рекуррентное соотношение.

Отдельно выпишем первое , входящее в сумму — в данном случае *a*0: *a*0=1.

Теперь, подставив *a*0 в правую часть рекуррентного соотношения, получим *a*1; подставив *a*1 в правую часть, получим *a*2 — и т. д.

В программе это реализуется следующим образом: в цикле параллельно с подсчетом *s* вычисляется и следующее слагаемое *ak.*

#include <iostream>

using namespace std;

int main() {

int n; double x,s,a;

cout<<”введите x,n\n”;

cin>>x>>n;

s=0; a=1;

for(int k=0;k<=n;k++) {

s+=a;

a\*=x/(k+1); }

cout<<”s=”<<s<<endl;

return 0; }

**Задачи**

1. Ввести целое число *n*. Выписать все его делители.
2. Натуральное число называется простым, если у него имеется только два делителя: оно само и единица (например, 19, 23). Выписать все простые числа в заданном диапазоне.
3. Дано натуральное число *n*. Выписать его разложение на простые множители.
4. В банке под 6% годовых открыт вклад на *S* рублей. Начисление процентов происходит ежемесячно (с капитализацией). Какой станет сумма вклада через  лет?
5. Вводятся координаты  точек. Определить, сколько из них попадает в круг радиусом  с центром в точке (*a*, *b*).
6. Дано *x*. Вычислить



1. Даны *x*, ε. Вычислить ту же сумму, что и в предыдущей задаче, но конечное слагаемое заранее не известно. Суммировать до тех пор, пока очередное слагаемое по модулю не станет меньше ε.
2. Даны *x*, *n*. Вычислить .
3. Даны *x*, ε. Вычислить



Суммировать до тех пор, пока очередное слагаемое по модулю не станет меньше ε. Результат сравнить (визуально) с  (должны совпасть первые несколько знаков).

1. Даны *x*, ε. Вычислить



Суммировать до тех пор, пока очередное слагаемое по модулю не станет меньше ε. Результат сравнить (визуально) с  (должны совпасть первые несколько знаков).

1. Вычислить величину .
2. Последовательность Фибоначчи  определяется следующим образом: *x*1=*x*2=1, *xn*=*xn*–1+*xn*–2 (*n*≥3) (числа 1, 1, 2, 3, 5, 8, 13, 21, 34, …). Найти *x*40 (массивов не использовать).