Циклы

Инструкция цикла в структурном программировании предназначена для повторения некоторого блока программы до достижения определенного состояния. Требуемое состояние записывается в форме логического выражения.

Часто выделяется один тип циклических конструкций, который называют циклом с фиксированным числом повторений или циклом со счетчиком. Главная особенность таких циклов заключается в том, что требуемое состояние вычислительной среды наступает после того, как блок инструкций вложенных в цикл выполнится определенное и известное до начала цикла число раз. Для записи циклических конструкций такого вида в C++ служит инструкция `for`. Потенциально она позволяет записывать гораздо более сложные конструкции, но, как правило, используется в следующем виде

```
for (int i=0;i<n;i++) {
    ...
}
```

Данная инструкция создает новый блок, в пределах которого создается новая переменная `i` с начальным значением 0. Объявление переменной может быть выполнено и до цикла, однако более правильным стилем в данном случае будет сделать это ближе к месту ее использования. Второе выражение в инструкции является логическим, цикл повторяется пока оно является истинным. Третье выражение выполняется в конце каждой итерации и, обычно, указывает способ изменения переменной. Операция «`++`» приводит к увеличению переменной на единицу. Таким образом, цикл будет выполняться n раз, и переменная i будет последовательно принимать значения 0, 1, 2, ..., $n-1$. При достижении переменной значения n цикл прекратит работу.

Изменяя каждое из выражений, можно добиться более сложного поведения для счетчика `i`. В следующем примере цикл повторяется 11 раз, при этом `i` последовательно принимает значения 100, 90, 80, ..., 10, 0.

```
for (int i=100;i>=0;i-=10) {
    ...
}
```

Как легко догадаться операция «`-=`» приводит к уменьшению значения `i` на заданную величину.

Язык C++ не содержит запретов на изменение счетчика в теле цикла `for`, однако этого лучше избегать для того, чтобы не ухудшать читаемость программы.

Цикл с предусловием

Другим вариантом организации циклических вычислений является цикл «`while`», выполняющийся пока некоторое логическое выражение P является истинным. С одной стороны это создает угрозу зацикливания программы — ошибки в разработке, из-за которой P будет истинным все время, и цикл будет выполняться до внешнего прерывания. С другой стороны такая конструкция гарантирует, что после завершения цикла выражение P наверняка будет ложным. Этот факт требуется использовать для осмысленного программирования.

Приведем пример. Для заданного натурального числа x найти наименьшее число y, которое превосходит x, и является степенью двойки. Очевидно, что задачу можно решить при помощи цикла с предусловием, но какое логическое выражение следует использовать в нем? Ответ должен получиться не в результате подбора или угадывания, а логически обоснованно. Если в результате нам требуется получить состояние при котором `y>x`, то логическое выражение в цикле должно быть его отрицанием `y<=x`, и начало инструкции пример вид.

```
while (y<=x) {
  ...
}
```

Еще один пример. Программа должна запросить у пользователя целое число от 1 до 100. Если на вход поступит число, не принадлежащее заданному диапазону, то попытку надо повторить. Программа будет иметь вид

```
int x;
cin>>x;
while (...) {
  cin>>x;
}
```

Какое логическое выражение должно быть записано в скобках? После завершения цикла требуется, чтобы выполнялось условие `x<=100␣&&␣x>=1`. Следовательно выражением в инструкции `while` будет его отрицание `x>100␣||␣x<1`.

Понятие инварианта и разработка циклических программ

Инвариантом в программировании называют логическое выражение, которое выполняется при каждом попадании программы в некоторое состояние. В частности, инвариантом цикла называется выражение, истинное в момент входа в цикл и в момент завершения каждой итерации цикла. Обычно инвариант описывает закономерности между значениями переменных или входными данными. Рассмотрим пример программы.

```
int p,x,v;
cin>>x;
for (p=0,v=1;v<x;p++)
  v*=2;
cout<<p;
```

Здесь инвариантом цикла является утверждение $v = 2^p$. Сразу отметим, что инвариант не встречается в тексте программы, и может быть записан только в комментариях. Несмотря на это, именно он оказывает основное влияние на сам текст программы, причем его обоснование является весьма схожим с доказательством по индукции. Заметим, что в момент входа в цикл $p = 0$ и $v = 1$, то есть $2^p = v$, что соответствует базису индукции. Далее, пусть на некотором шаге цикла утверждение $2^p = v$ выполнялось, тогда за одну итерацию v увеличится в два раза, а p на единицу и утверждение $2^{p+1} = 2 * v$ также будет верным.

Но фактическая разработка циклических программ представляет обратный процесс, в котором по инварианту мы строим программу. Можно порекомендовать следующую последовательность действий при разработке циклических алгоритмов.

1. Подумать и определить, какая информация должна храниться в программе для достижения требуемого результата. При этом, лучше начинать рассмотрение не с первой итерации цикла, а с некоторой промежуточной. Например: «Цикл будет повторяться 1000 раз. После того, как цикл повторится 500 раз будут найдены следующие величины ...»

2. Определить имена переменных для хранения требуемых величин, максимально четко сформулировать определение для каждой переменной и определить взаимосвязь между ними.

3. Записать тело цикла, исходя из требуемой взаимосвязи.

4. Присвоить начальные значения переменным так, чтобы требуемая взаимосвязь между ними выполнялась в момент входа в цикл.

5. Записать условие продолжения цикла, исходя из требуемого состояния после его завершения.

Рассмотрим следующий пример. На участке железной дороги расположено n станций. Известно на каком расстоянии от начала дороги расположенная каждая станция. Перегоном называется участок дороги между двумя соседними станциями. Найти длину самого длинного перегона. На вход подается количество станций и расстояние до них в порядке движения.

Последовательность решения этой задачи в соответствии с представленным выше планом может быть следующей.

1. Для решения задачи будем считывать последовательно станции, находя расстояние между соседними, и вычисляя ответ. Для этого потребуется запоминать количество станций, порядковый номер текущей станции (чтобы определить конец ввода), значение ответа. В ходе вычислений нам потребуется знать координату последней считанной и предудущей станции.

2. Количество станций будем хранить в переменной `n`, номер текущей станции в переменной `i` (нумерацию будем начинать с нуля), ответ будет вычисляться в переменной `ans`, координата станции с номером `i` будет храниться в переменной `cur`, предыдущей станции — в переменной `prev`. На i-том шаге цикла `ans` хранит длину самого длинного перегона для всех станций с номерами от 0 до $i-1$.

3. Пусть указанные свойства имеют место в начале некоторого шага. За одну итерацию цикла переменная `i` будет увеличена на 1. Это должно сопровождаться рядом других изменений. Во-первых, местоположение очередной станции должно быть прочитано в переменную `cur` инструкцией `cin>>cur`. Во-вторых, координата предшествующей станции должна попасть в переменную `cur`. В момент начала итерации она хранилась в переменной `cur`, поэтому потребуется выполнить присваивание `prev=cur`, причем его необходимо сделать до чтения нового значения в `cur`. Наконец, возможна ситуация что последний перегон оказался длиннее всех предыдущих, поэтому требуется выполнить проверку и измение ответа. Таким образом, программа примет вид

```
for (...;...;i++) {
  prev=cur;
  cin >> cur;
  if (cur-prev>ans)
    ans=cur-prev;
}
```

4. Начать работу цикла с $i = 0$ невозможно, ведь в этом случае в переменной `prev` должна оказаться координата станции с номером -1. Если начать работу с $i = 1$, то потребуется решить вопрос зо значением переменной `ans` в случае, когда ни одного перегона еще не было. Мы начнем работу цикла с $i = 2$. Тогда, в переменных `cur` и `prev` должны быть координаты станций с номерами 1 и 0 соответственно, а в переменной `ans` длина перегона между ними, пока единственного и потому самого длинного.

5. Наконец, условие продолжения цикла следует из утверждения о том, что `ans` хранит длину самого длинного перегона для всех станций с номерами от 0 до $i-1$. Последняя станция имеет номер $n-1$ поэтому в конце цикла значения переменных `n` и `i` должны совпасть, тогда условие продолжения цикла можно записать как `i<n`.

Программа, в итоге, примет вид.

```
int prev,cur,n;
cin >> n;
cin >> prev >> cur;
int ans=cur-prev;
for (i=2;i<n;i++) {
  prev=cur;
  cin >> cur;
  if (cur-prev>ans)
    ans=cur-prev;
}
cout<<ans<<endl;
```

Досрочное достижение постусловия цикла и инструкция «break»

В том случае, когда требуемое состояние после завершения цикла может быть достигнуто разными способами можно использовать инструкцию «`break`». В результате выполнения этой инструции в языке C++ работа текущего цикла прекращается и выполнение программы продолжается с точки завершения цикла. Обычно «`break`» используется вместе с условием, проверяющим, что постусловие цикла уже достигнуто.

Рассмотрим следующий пример. Во входном потоке записано `n` целых чисел. Требуется прочитать числа входного потока до первого отрицательного числа. Требуется вывести номер этого числа или `0`, если отрицательные чисела отсутствуют.

Будем хранить в переменной `x` последнее прочитанное число, а в переменной `i` его порядковый номер. В цикле будет выполняться операция чтения числа из потока. Желаемым постусловием цикла будет `(i==n)||(x<0)`. Другими словами, в момент завершения цикла требуется, чтобы либо текущее прочитанное число было отрицательным, либо были прочитаны все `n` чисел. Можно записать требуемую программу и без инструкции «`break`». Для этого надо правильно найти отрицание постусловия `(i!=n)&&(x>=0)` (отрицание «или» является «и») и записать его в качестве логического выражания инструкции `while`. Но второе условие выхода из цикла можно записать и с помощью инструкции `break` в следующей программе.

```
int n,x,i=0;
cin >> n;
while (i!=n) {
  ++i;
  cin >> x;
  if (x<0)
    break;
}
if (x<0)
  cout << i;
else
  cout << 0;
```