Циклы

Для выполнения некоторых действий множество раз в зависимости от определенного условия используются циклы. В языке C++ имеются следующие виды циклов:

* for
* while
* do...while

**Цикл while**

Цикл while выполняет некоторый код, пока его условие истинно, то есть возвращает true. **Он имеет следующее формальное определение:**

|  |  |
| --- | --- |
| 1  2  3  4 | while(условие)  {      // выполняемые действия  } |

После ключевого слова while в скобках идет условное выражение, которое возвращает true или false. Затем в фигурных скобках идет набор инструкций, которые составляют тело цикла. И пока условие возвращает true, будут выполняться инструкции в теле цикла.

Например, выведем квадраты чисел от 1 до 9:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <iostream>    int main()  {      int i = 1;      while(i < 10)      {          std::cout << i << " \* " << i << " = " << i \* i << std::endl;          i++;      }        return 0;  } |

Здесь пока условие i < 10 истинно, будет выполняться цикл while, в котором выводится на консоль квадрат числа и инкрементируется переменная i. В какой-то момент переменная i увеличится до 10, условие i < 10 возвратит false, и цикл завершится.

Консольный вывод программы:

1 \* 1 = 1

2 \* 2 = 4

3 \* 3 = 9

4 \* 4 = 16

5 \* 5 = 25

6 \* 6 = 36

7 \* 7 = 49

8 \* 8 = 64

9 \* 9 = 81

Каждый отдельный проход цикла называется итерацией. То есть в примере выше было 9 итераций.

Если цикл содержит одну инструкцию, то фигурные скобки можно опустить:

|  |  |
| --- | --- |
| 1  2  3 | int i = 0;  while(++i < 10)      std::cout << i << " \* " << i << " = " << i \* i << std::endl; |

**Цикл for**

**Цикл for имеет следующее формальное определение**:

|  |  |
| --- | --- |
| 1  2  3  4 | for (выражение\_1; выражение\_2; выражение\_3)  {      // тело цикла  } |

выражение\_1 выполняется один раз при начале выполнения цикла и представляет установку начальных условий, как правило, это инициализация счетчиков - специальных переменных, которые используются для контроля за циклом.

выражение\_2 представляет условие, при соблюдении которого выполняется цикл. Как правило, в качестве условия используется операция сравнения, и если она возвращает ненулевое значение (то есть условие истинно), то выполняется тело цикла, а затем вычисляется выражение\_3.

выражение\_3 задает изменение параметров цикла, нередко здесь происходит увеличение счетчиков цикла на единицу.

Например, перепишем программу по выводу квадратов чисел с помощью цикла for:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <iostream>    int main()  {      for(int i =1; i < 10; i++)      {          std::cout << i << " \* " << i << " = " << i \* i << std::endl;      }        return 0;  } |

Первая часть объявления цикла - int i = 1 - создает и инициализирует счетчик i. Фактически это то же самое, что и объявление и инициализация переменной. Счетчик необязательно должен представлять тип int. Это может быть и другой числовой тип, например, float. И перед выполнением цикла его значение будет равно 1.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока переменная i не станет равна 10.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--. Можно изменять на другое значение: i+=2.

В итоге блок цикла сработает 9 раз, пока переменная i не станет равна 10. И каждый раз это значение будет увеличиваться на 1. И по сути мы получим тот же самый результат, что и в случае с циклом while:

1 \* 1 = 1

2 \* 2 = 4

3 \* 3 = 9

4 \* 4 = 16

5 \* 5 = 25

6 \* 6 = 36

7 \* 7 = 49

8 \* 8 = 64

9 \* 9 = 81

Необязательно указывать все три выражения в определении цикла, мы можем одно или даже все из них опустить:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int i = 1;  for(; i < 10;)  {      std::cout << i << " \* " << i << " = " << i \* i << std::endl;      i++;  } |

Формально определение цикла осталось тем же, только теперь первое и третье выражения в определении цикла отсутствуют: for (; i < 10;). Переменная-счетчик определена и инициализирована вне цикла, а ее приращение происходит в самом цикле.

Можно определять вложенные циклы. Например, выведем таблицу умножения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <iostream>    int main()  {      for (int i=1; i < 10; i++)      {          for(int j = 1; j < 10; j++)          {              std::cout << i \* j << "\t";          }          std::cout << std::endl;      }        return 0;  } |

**Цикл do**

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, то есть не равно 0, то цикл повторяется. **Формальное определение цикла:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | do  {      инструкции  }  while(условие); |

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | #include <iostream>    int main()  {      int i = 6;      do      {          std::cout << i << std::endl;          i--;      }      while(i>0);        return 0;  } |

Здесь код цикла сработает 6 раз, пока i не станет равным нулю.

Но важно отметить, что цикл do гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции while не будет истинно. То есть мы можем написать:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int i = -1;  do  {      std::cout << i << std::endl;      i--;  }  while(i>0);  } |

Хотя у нас переменная i меньше 0, цикл все равно один раз выполнится.

**Операторы continue и break**

Иногда возникает необходимость выйти из цикла до его завершения. В этом случае можно воспользоваться оператором **break**. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <iostream>    int main()  {      int i = 1;      for ( ; ; )      {          std::cout << i << " \* " << i << " = " << i \* i << std::endl;          i++;          if (i > 9) break;      }      return 0;  } |

Здесь когда значение переменной i достигнет 10, осуществляется выход из цикла с помощью оператора **break**.

В отличие от оператора **break**, оператор **continue** производит переход к следующей итерации. Например, нам надо посчитать сумму только нечетных чисел из некоторого диапазона:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include <iostream>  int main()  {      int result = 0;      for (int i=0; i<10; i++)      {          if (i % 2 == 0) continue;          result +=i;      }      std::cout << "result = " << result << std::endl; // 25      return 0;  } |

Чтобы узнать, четное ли число, мы получаем остаток от целочисленного деления на 2, и если он равен 0, то с помощью оператора continue переходим к следующей итерации цикла. А если число нечетное, то складываем его с остальными нечетными числами.