цикл for

Допустим нам необходимо написать программу, которая выводит 100 раз строку "**Привет**!". Для решения данной задачи, мы напишем примерно следующий код:

Console.WriteLine("Привет!");

Console.WriteLine("Привет!");

Console.WriteLine("Привет!");

……

Console.WriteLine("Привет!");

Console.WriteLine("Привет!");

Console.WriteLine("Привет!");

То есть, одну и ту же команду нам придется написать **100** раз. Согласитесь это долго, скучно и практически невозможно. Для выполнения одной и той же команды, мы можем воспользоваться **циклом**. **Циклы** служат для **многократного повторения** некоторого фрагмента кода. Существуют разные типы циклов, сперва мы изучим цикл **for,** который используется тогда, когда наперед **известно**, **сколько повторений** нужно сделать.

Цикл for имеет следующую структуру:

for (инициализация счетчика; условие продолжения; итерация)

{

блок кода, который будет повторяться

}

Данная структура состоит из ключевого слова **for**, за которым следуют круглые скобки, содержащие внутри три выражения, разделенных точками с запятой(**;**). Он имеет следующий порядок выполнения:

1. В начале цикла выполняется **инициализирующее выражение**, оно всегда вычисляется (выполняется) только **один раз в самом начале.**

Чаще всего создается целочисленная переменная **i** и присваивается начальное значение 0. Также эту переменную называют **переменной цикла**.

1. Далее следует **условное выражение**. Перед началом каждого захода в цикл, вычисляется значение условного выражения. Если оно принимает значение **true**, то тело цикла **выполняется**, если оно принимает значение **false**, выполнение цикла **завершается**. Если при первой проверке условие оказывается ложным, тело цикла **не** выполнится ни разу.

В примере, мы будем заходить в цикл, когда значение переменной меньше ста.

1. В самом конце, **после** выполнения действий в теле цикла обрабатывается последнее выражение (следующее выражение после условия).

В данном случае это **инкрементирующее** выражение - оно увеличивает с помощью инкремента значение переменной-цикла.

Для нашей задачи точно подходит, ведь нам нужно делать одно и тоже (выводить на консоль строку) 100 раз. Давайте посмотрим на программу, которая выводит нужную нам строку 100 раз:

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

{

Console.WriteLine("Привет!");  
}

Заход в цикл называется **итерацией.**

Для того, чтобы посчитать количество итераций цикла, то есть сколько раз выполняться команды внутри фигурных скобок, нужно  посчитать два значения:

* **max** - максимальное значение переменной цикла, при котором мы зашли в цикл
* **min** - минимальное значение переменной цикла, при котором мы зашли в цикл

**Количество итераций** вычисляется следующим образом: **max - min + 1**.

Заметим, что количество итераций цикла зависит от количества целых чисел на отрезке [min, мах]. Так как нам нужно лишь количество, то мы можем данный отрезок сдвинуть по числовой прямой в любую сторону, ведь тогда количество целых чисел на отрезке не измениться, например:

for (int i = 24; i <= 123; i++)

{

  Console.WriteLine("Привет!");  
}

**max** = 123, **min** = 24, следовательно количество итераций равно 123 - 24 + 1  = **100**.

Таким образом, выбирать границы отрезка можно выбирать любые, главное чтобы количество нужных нам итераций цикла сохранялось.

Переменная цикла

Разберем задачу, в котором надо вывести **четные числа от 1 до 100**. Воспользуемся циклом **for**, в котором переменная цикла будет принимать все значения от 1 до 100. Каждое такое значение будем проверять на четность и выводить на консоль. Следующий фрагмент кода показывает, как это сделать:

for (int i = 1; i <= 100; i++)  
{

   if(i % 2 == 0)  
   {  
      Console.WriteLine(i);  
   }  
}

Можно решить эту задачу при помощи манипулирования переменной цикла, а именно ее изменением. В прошлом решении, мы увеличивали переменную цикла на 1 и тем самым перебирали все числа. Не сложно догадаться, что мы можем перебирать не все числа, а **каждое второе**. То есть начнем с двойки и будем прибавлять к переменной цикла по два:

for (int i = 2; i <= 100; i = i + 2)  
{  
  Console.WriteLine(i);  
}

Накопление суммы (сумматор)

Данный алгоритмический прием используется, когда надо просуммировать большое количество чисел. Для этого переменной, в которую будет записываться сумма, в начале присваивается **нулевое** значение, затем делается цикл, где **на каждом** шаге к этой переменной добавляется очередное число.

**Пример:** Просуммировать все целые числа от 1 до 100.

int sum = 0;

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

{

   sum = sum + i;

}

Console.WriteLine(sum);

Очень важная, фундаментальная идея, использованная в данном приеме, состоит в том, что результат выполнения каждого шага цикла, зависит от значения переменной, вычисленной **на предыдущем шаге**. Таким образом, мы на каждом шаге получаем **новый результат**. Так в приведенном примере очередное число добавляется к значению переменной **sum**, полученному **на предыдущем шаге**. Чтобы было к чему добавлять, перед циклом обязательно должна присутствовать **инициализация** (присваивание начального значения) переменной, в которой накапливается **сумма**. Чаще всего требуется присвоить ей начальное значение **0**.

Сумма 10 чисел

Задача следующая: пользователь вводит 10 чисел. Нужно найти сумму введенных чисел. Будем решать следующим образом: В цикле которая выполняется 10 раз, будем **вводить** число и **суммировать в переменную-**сумматор. Каждый раз в нашей переменной будет хранится очередное число, следовательно в сумматор мы будем складывать все числа. После цикла в переменной будет находится сумма 10 чисел. Рассмотрим на примере:

int sum = 0;

int number;

for (int i = 1; i <= 10; i++)

{

  number = Convert.ToInt32(Console.ReadLine());  
  sum = sum + number;

}

Console.WriteLine(sum);

Накопление произведения (мультипликатор)

Аналогично накоплению суммы можно в отдельной переменной накапливать произведение. Переменной, в которой производится накопление, присваивается **начальное значение 1**. Это нужно, чтобы не испортить вычисления при перемножении, ведь при умножении на ноль всегда получается ноль.

Рассмотрим пример с вычислением **факториала**.  **Факториалом** целого числа *n* называется произведение всех целых чисел от 1 до *n*. Обозначается *n*!.

То есть: *n*!=1⋅2⋅3⋅…⋅*n*.

int n = Convert.ToInt32(Console.ReadLine());  
int fact = 1;

for (int i = 1; i <= n; i++)  
{  
   fact = fact \* i;   
}

Console.WriteLine(fact);

Цикл While

Цикл **while** позволяет выполнить одну и ту же последовательность действий пока проверяемое условие **истинно**. При этом условие записывается до тела цикла и проверяется до выполнения тела цикла.

while (условие продолжения)

{

  //блок кода, который будет повторяться

}

При выполнении цикла **while** сначала проверяется условие. Если оно **ложно**, то цикл **не** выполняется и управление передается на следующую инструкцию после тела цикла **while**. Если условие **истинно**, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, **пока** условие будет истинно. Как только условие станет ложно, работа цикла **завершится** и управление передастся следующей инструкции после цикла.

**Если результат условие окажется ложным при первой проверке, то тело цикла не выполнится ни разу.**

Слово **while** переводится, как «**пока**», что хорошо его характеризует. Он продолжает выполнятся до тех пор, **пока** истинно некоторое условие.

Пример программы, которая выводит на экран числа 0, 1, 2, 3, 4:

int i = 0;

while (i < 5)  
{  
  Console.WriteLine(i);

  i++;  
}

**While** очень похож на **if**, но если **if** выполняет действия **один раз**, то **while** будет делать их до тех пор, **пока выполнено условие**.

Используйте цикл **while**, когда заранее **неизвестно** количество итераций цикла.

Используйте цикл **for**, когда заранее **известно** точное количество итераций цикла.

Так же есть

Давайте разберем следующую задачу: по данному натуральному числу n, найти .

Итак, =2⋅2⋅2⋅...⋅2, то есть у нас производится накопление умножения на 2 , значит нужно использовать мультипликатор:

int n = Convert.ToInt32(Console.ReadLine());  
int pokazatel = 0;   
int stepen = 1;  
  
while (pokazatel < n)  
{  
  stepen = stepen \* 2;  
  pokazatel++;  
}  
  
Console.WriteLine(stepen);

У нас есть 2 переменные, в которых заключается основная идея:

int pokazatel = 0;

int stepen = 1;

Заметим, что = stepen. Данное условие всегда должно сохраняться после  каждой итерации цикла. Также заметим, что количество итераций цикла равно n, то есть умножение на 2 происходит n раз.

Анализ цифр числа

Рассмотрим следующую задачу. Нужно посчитать сумму цифр числа, при этом мы **не** знаем, сколько знаков содержит число.

Давайте рассуждать. Мы знаем, как находить **последнюю** цифру числа — для этого достаточно посчитать **остаток от деления** на 10. Также мы знаем, как отбросить **последнюю** цифру числа — для этого нужно поделить число на 10 **нацело**.

Если теперь выполнить для полученного числа те же шаги, то можно заметить, что мы можем обойти все цифры числа **справа налево**. Остановиться нам нужно будет, когда **цифры в числе закончатся** (оно станет равным **нулю**).

Сумму цифр мы умеем считать, вы ведь помните про **сумматор**?

int n = Convert.ToInt32(Console.ReadLine());

int sum = 0;

while (n > 0)   
{

   int digit = n % 10

   sum = sum + digit;

   n = n / 10;   
}

Console.WriteLine(sum);

Перевод в двоичную систему счисления

int n = Convert.ToInt32(Console.ReadLine());  
  
while (n > 0)  
{  
   Console.Write(n % 2);  
   n = n / 2;  
}

Последовательности

Разберем следующую задачу: Последовательность состоит из **положительных** чисел и оканчивается **нулём**, нужно вывести **сумму** всех элементов в последовательности.

Сначала давайте разберемся с вводом:

int n = Convert.ToInt32(Console.ReadLine());

while (n != 0)

{

   n = Convert.ToInt32(Console.ReadLine());

}

То есть, в первый раз мы вводим число, проверяем на равенство с **нулем**, и **пока** наше число **не равно нулю**, мы **постоянно** в него записываем **новое** число, которое ввели с клавиатуры.

Теперь нам нужно **суммировать** все такие числа. Естественно, к нам на помощь приходит знание о **сумматоре**. Заводим переменную, в котором будем хранить сумму **уже** введенных чисел. В конце программы в ней окажется сумма **всех** введенных чисел **до** числа ноль, так как на **нуле** последовательность **заканчивается**.

Для лучшего понимания, рассмотрим решение:

int n = Convert.ToInt32(Console.ReadLine());

int sum = 0;

while (n != 0)

{  
   sum = sum + n;

   n = Convert.ToInt32(Console.ReadLine());

}

Console.WriteLine(sum);

Вложенные циклы

Циклы позволяют повторять выполнение любого набора операторов. В частности можно повторять много раз выполнение другого цикла. Такие циклы называются **вложенными**. Разберем следующую задачу: напечатать числа в виде следующей таблицы:

    5 5 5 5 5 5

    5 5 5 5 5 5

    5 5 5 5 5 5

Данная таблица состоит из трех строчек, в каждой из которых число 5 напечатано 6 раз. Строчку из шести чисел можно напечатать с помощью одного цикла **for**:

for (int i = 1; i <= 6; i++)

{

   Console.Write(5 + " ");

}

Чтобы повторить вывод строчки 3 раза, вставляем этот цикл внутрь другого:

for (int k = 1; k <= 3; k++)

{  
   for (int i = 1; i <= 6; i++)

   {

      Console.Write(5 + " ");

   }

   Console.WriteLine();

}

Типичная ошибка, когда в качестве счетчиков вложенных циклов (**i** и **k** в приведенном примере) используется **одна и та же** переменная. То есть **нельзя** в каждом из циклов использовать одну переменную **i**. Помнить об этом особенно **важно**, поскольку данная ошибка **не** обнаруживается на этапе компиляции. Ваша программа запустится, но делать будет вовсе не то, что вы от нее ждете.

Break и Continue

Ходом выполнения цикла можно управлять с помощью двух операторов **break** и **continue**.

* **break** – прерывает выполнение цикла, управление передается операторам, следующим за оператором цикла.
* **continue** – прерывает выполнение очередного шага цикла и возвращает управление в начало цикла, начиная следующий шаг.

Например:

for (int n = 1; n <= 10; n++)  
{  
   if (n % 2 == 0)  
   {  
      continue;  
   }  
   if (n == 7)  
   {  
      break;  
   }  
  
   Console.WriteLine(n);  
}

Данная программа будет печатать только нечетные числа (из-за срабатывания **continue**). Цикл прекратит выполняться, когда *n* станет равно 7. В итоге будут напечатаны числа: 1,3,5.

Перебор

Имеется целый класс задач, решение которых сводится к перебору различных вариантов, среди которых выбирается такой, который удовлетворяет условию задачи.

Рассмотрим задачу:

Найти и вывести все целые решения квадратного уравнения на отрезке [1; 10000]

*a*+*bx*+*c*=0

Эту задачу с первого взгляда хочется решать через дискриминант. Такое решение не очень подходит, так как при взятии дискриминанта происходит потеря точности и некоторые решения могут быть пропущены.   
Вспомним, что такое решение уравнения: это такое число, при подстановке которой в уравнение вместо x получается верное равенство. Давайте так и будем делать: возьмем каждое целое значение из данного отрезка и будем подставлять вместо x. Если при этом у нас получится верное равенство, значит взятое значение является решением уравнения. Так сделаем для каждого целого числа из отрезка [1,10000]. Для лучшего понимания, рассмотрим код:

int a = Convert.ToInt32(Console.ReadLine());

int b = Convert.ToInt32(Console.ReadLine());

int c = Convert.ToInt32(Console.ReadLine());

for (int x = 1; x <= 10000; x++)

{

   if (a \* x \* x + b \* x + c == 0)

   {

      Console.WriteLine(x);

   }

}

Минимаксные задачи

Часто в задачах приходится искать **максимальное** и/или **минимальное** значения. Разберем основную идею нахождения максимума, а для минимума алгоритм аналогичен за мелким исключением. Итак, дается последовательных натуральных чисел. Каждое число не превосходит 106. Задача найти **максимальное** число в последовательности.

Идея алгоритма нахождения максимального значения: есть переменная **max**, в котором хранится **максимум из ранее просмотренных чисел**. Проходим по последовательности чисел и **сравниваем** каждое число **с** **max**. Если текущее число больше **max**, то в **max** записываем текущее число. **После** просмотра **всех** чисел последовательности, в **max** будет хранится **максимальное** число последовательности.

Единственное осталось определиться с **начальным значением** переменной **max**. Заметим, что в **max** должно хранится заведомо **меньшее** значение, чем **минимальное возможное** значение чисел в последовательности. Давайте поймем на примере, почему это важно. Пусть начальное значение переменной **max** = 5, а последовательность выглядит следующим образом: **1 2 3 4 2 1**. Теперь если пройдемся по последовательности и будем сравнивать с **max**, то **после** просмотра всех чисел последовательности, значение переменной **max не изменится**. Следовательно, окажется, что **максимальное** значение последовательности равно **5**, однако в последовательности **все числа меньше 5**. Следовательно, следует выбирать **начальное значение max заведомо меньшее всех возможных чисел.**

Рассмотрим код программы для лучшего понимания алгоритма:

int n = Convert.ToInt32(Console.ReadLine());  
int number;  
int max = 0;

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

{  
   number = Convert.ToInt32(Console.ReadLine());

   if (number > max)   
   {  
      max = number;   
   }  
}

Console.WriteLine(max);