**Цикли от тип „аритметична прогресия“ с предварително известен брой на повторенията**

В тази статия ще научим как да използваме цикли в C #, какъв тип цикли съществуват и кога да ги използваме. Ще използваме няколко примера, за да обясним отблизо използването на всеки цикъл в C #.

**Какво е "цикъл" ?**

В програмирането често се налага многократно изпълнение на дадена последователност от операции. Цикъл е основна конструкция в програмирането, **която позволява многократно изпълнение на даден** **фрагмент сорс код**. В зависимост от вида на цикъла програмният код в него се повтаря **или фиксиран брой пъти, или докато е в сила дадено условие**.

**Видове цикли**

**Цикъл с предусловие - While Loop**

While цикълът е цикъл с **предварително условие**. Условието е произволен израз, който **определя докога ще се изпълнява тялото на цикъла** и връща булев резултат – истина (true) или лъжа (false).Тялото на цикъла е програмният код, изпълняван при всяко повторение (итерация) на цикъла, т.е. всеки път, когато входното условие е истина. Това означава, че **първо проверяваме условието и след това,** **ако условието е истина, изпълняваме израза в тялото**. Тялото на while цикълa може и да не се изпълни нито веднъж, ако в самото начало е нарушено условието на цикъла. Един while цикъл изглежда по следния начин :

Console.WriteLine("Enter your name,please !");

string name = Console.ReadLine();

Console.WriteLine("Confirm your name,please !");

string checkName = Console.ReadLine();

while (name != checkName) // условие за край на цикъла

{

Console.WriteLine("Maybe you don't know your name, hahahha"); //тяло на цикъла

checkName = Console.ReadLine();

}

Console.WriteLine("Hallo , " + checkName);

**Примерна задача 1:**

*Условие : Намерете сумата на числата от 1 до 10:*

static void Main(string[] args)

{

int sum,x; // декларация на променлива;

x = 1; // инициализация на променлива(присвояване на стойност);

sum = 0;

while (x <= 10) // условие за край;

{

sum = sum + x; // тяло на цикъла;

x++; // инкрементиране(обновяване) на променливата с 1;

}

Console.WriteLine("The sum of the numbers bewtween 1 and 10 is " + sum); //отпечатване на сумата

}

**Цикъл със следусловие - Do-While Loop**

Do-while цикълът е аналогичен на while цикъла, само че при него проверката на булевото условие се извършва след изпълнението на операциите в цикъла. Един do-while цикъл изглежда по следния начин:

int x;

do

{

Console.WriteLine("Enter a positive number, please!"); // код за изпълнение или тяло на цикъла

x = int.Parse(Console.ReadLine());

if (x <= 0) Console.WriteLine("Wrong number");

} while (x >= 0); // условие за край

Първоначално се изпълнява тялото на цикъла. След това се проверява неговото условие. Ако то е истина, тялото на цикъла се повтаря, а в противен случай цикълът завършва. **Тялото на цикъла се повтаря най-малко един път.** Ако условието на цикъла постоянно е истина, цикълът никога няма да завърши. **Do-while цикълът се използва, когато искаме да си гарантираме, че поредицата от** **операции в него ще бъде изпълнена многократно и задължително поне веднъж в началото на цикъла.**

**Конструкция за цикъл For**

For-циклите са малко по-сложни от while и do-while циклите, но за сметка на това могат да решават по-сложни задачи с по-малко код.

Те съдържат **инициализационен блок, условие на цикъла, тяло** **на цикъла и команди за обновяване на водещите променливи.** Ето как изглежда програмният код на един for-цикъл:

for (инициализация; условие; обновяване)

{

тяло на цикъла;

}

Като пример можем да разгледаме :

for (int i = 0; i < 10; i++)

{

Console.WriteLine("the number is : " + i); // тяло на цикъла

}

Използваме инициализацията в началото на цикъла за да дадем стойност на променливата. Условието се използва за определяне на това, кога цикълът е завършен. Обновяването е част, в която увеличаваме или декрементираме нашата променлива, инициализирана в инициализационната част. Тялото се състои от всички изрази, които трябва да изпълним, стига условието да е вярно.

Важно е да знаете, че редът на изпълнение е: **Инициализация, условие, тяло на цикъла, обновяване.**

**Инициализация на for цикъла**

for (int num = 0; ...; ...)

{

// Променливата num е видима тук и може да се използва

}

// Тук num не може да се използва

Той се изпълнява само веднъж, точно преди влизане в цикъла. Инициализационният блок се използва за деклариране на променливата-брояч и задаване на нейна начална стойност. Тази променлива е "видима" и може да се използва само в рамките на цикъла.

**Условие на for цикъла**

for (int num = 0; num < 10; ...)

{

тяло на цикъла

}

Условието за повторение (loop condition) се изпълнява веднъж, преди всяка итерация на цикъла, точно както при while циклите. При резултат true се изпълнява тялото на цикъла, а при false то се пропуска и завършва.

**Обновяване на водещата променлива**

for (int num = 0; num < 10; num++)

{

тяло на цикъла;

}

Този код се изпълнява след всяка итерация, след като е приключило изпълнението на тялото на цикъла. Най-често се използва за обновяване стойността на брояча.

**Примерна задача 2:**

*Като следващ пример ще напишем програма, която повдига числото n на степен m, като за целта ще използваме for-цикъл:*

static void Main(string[] args)

{

Console.Write("n = ");

int n = int.Parse(Console.ReadLine());

Console.Write("m = ");

int m = int.Parse(Console.ReadLine());

decimal result = 1;

for (int i = 0; i < m; i++) //инициализация; условие; обновяване на променливата

{

result \*= n; // тяло на цикъла

}

Console.WriteLine("n^m = " + result);

}

Първоначално инициализираме резултата (result = 1). Цикълът започваме със задаване на начална стойност за променливата-брояч (int i = 0). Определяме условието за изпълнение на цикъла i < m). Така цикълът ще се изпълнява от 0 до m-1 или точно m пъти. При всяко изпълнение на цикъла умножаваме резултата по n и така n ще се вдига на поредната степен (1, 2, … m) на всяка итерация. Накрая отпечатаме резултата, за да видим правилно ли работи програмата. Ето как изглежда изходът от програмата при n=2 и m=10:

n = 2, m = 10, n^m = 1024

**Примерна задача 3:**

*Условие: Създайте приложение, което отпечатва всички цели числа от n до 1:*

static void Main(string[] args)

{

{

Console.WriteLine("Enter number n that is greater than 1: ");

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

for (int i = n; i >= 1; i--) // започваме инициализацията от n и вървим в низходящ ред

{

Console.WriteLine(i);

}

Console.ReadKey();

}

}

**Трябва да използваме for цикъл, когато знаем колко повторения ще имаме. Това означава, ако повторим всички елементи вътре в колекция или имаме крайна точка за итерации.**

**Конструкция за цикъл Foreach**

Тази конструкция служи за обхождане на **всички** елементи на даден масив, списък или друга колекция от елементи. Подробно за масивите ще говорим по-нататък, така че за момента можем да си представяме един масив като наредена последователност от числа или други елементи. Ето как изглежда един foreach цикъл:

foreach (variable in collection)

{

statements;

}

Както виждате, той е значително по-прост от стандартния for-цикъл с разликата в това ,че for цикълът изпълнява израз или блок от израз, **докато даденото условие не стане false**. Foreach цикълът обаче изпълнява израз или блок от израз **за всеки елемент**, присъстващ в масива и не е необходимо да се определя минималната или максималната граница. Също така във for циклите, ние итерираме масива както във възходящ, така и в низходящ ред, например от индекс 0 до 9 и от индекс 9 до 0. Но в цикъла на foreach ние итерираме масив само възходящо.

**Примерна задача 4:**

*Ето един пример, който показва как можем да използваме foreach:*

static void Main(string[] args)

{

Console.WriteLine("Print array:");

//задаваме променливата string , която всъщност се явява масив от символи(char)

string name = "Veselina";

// foreach цикъла ще се върти

// до достигане на последния елемент в масива

foreach (char items in name)

{

Console.WriteLine(items);

}

}

В примера се задаваме променлива от тип string, която се състои от масив от елементи с тип char, след това те се обхождат с foreach цикъл и се отпечатват на конзолата.

**Вложени цикли**

Вложените цикли представляват **конструкция от няколко цикъла, разположени един в друг**. Най-вътрешният цикъл се изпълнява най-много пъти,а най-външният – най-малко. Нека да разгледаме как изглеждат два вложени цикъла:

for (инициализация; проверка; обновяване)

{

for (инициализация; проверка; обновяване)

{

код за изпълнение;

}

}

След инициализация на първия for цикъл ще започне да се изпълнява неговото тяло, което съдържа втория (вложения) цикъл. Ще се инициализира променливата му, ще се провери условието му и ще се изпълни кода в тялото му, след което ще се обнови променливата му и изпълнението му ще продължи, докато условието му не върне false. След това ще продължи втората итерация на първия for цикъл, ще се извърши обновяване на неговата променлива и отново ще бъде изпълнен целият втори цикъл. **Вътрешният цикъл ще започне своето изпълнение от началната си точка** **толкова пъти, колкото се изпълнява тялото на външния цикъл.**

**Примерна задача 5:**

*Условие: По дадено число n да отпечатаме на конзолата триъгълник с n на брой реда, изглеждащ по следния начин:*

1

1 2

1 2 3

. . .

1 2 3 . . . n

static void Main(string[] args)

{

int n = int.Parse(Console.ReadLine());

for (int row = 1; row <= n; row++) //външен цикъл, обхождащ редовете

{

for (int col = 1; col <= row; col++) // вътрешен(вложен) цикъл, обхождащ колоните

{

Console.Write(col + " ");

}

Console.WriteLine();

}

}

Решаваме задачата с два for-цикъла. Външният цикъл ще обхожда редовете, а вътрешният – колоните. Когато сме на първия ред, трябва да отпечатаме "1" (1 елемент, 1 итерация на вътрешния цикъл). На втория ред трябва да отпечатаме "1 2" (2 елемента, 2 итерации на вътрешния цикъл). **Иначе казано, минаваме през всеки ред на триъгълника чрез външния цикъл и** **отпечатваме с вложения цикъл толкова на брой числа , колкото трябва да са колоните във всеки ред.** Виждаме, че има зависимост между реда, на който сме и броя на елементите, който ще отпечатваме. Това ни подсказва как да организираме конструкцията на вътрешния цикъл:

* инициализираме водещата му променлива с 1: col = 1;
* условието за повторение зависи от реда, на който сме: col <= row;
* на всяка итерация на вътрешния цикъл увеличаваме с единица водещата променлива.

**На практика трябва да направим един for цикъл (външен) от 1 до n (за редовете) и в него още един** **for-цикъл (вътрешен) за числата в текущия ред, който да върти от 1 до номера на текущия ред.** **Външният цикъл трябва да ходи по редовете, а вътрешният – по колоните от текущия ред.**

**Оператор Break**

Операторът break **се използва за преждевременно излизане от цикъл, преди той да е завършил изпълнението си по естествения си начин.** При срещане на оператора break **цикълът се прекратява и изпълнението на програмата** **продължава от следващия ред веднага след тялото на цикъла**. Прекратяването на цикъл с оператора break може да стане **само от неговото тяло**, когато то се изпълнява в поредната итерация на цикъла. Когато break се изпълни, **кодът след него в тялото на цикъла се прескача и не се изпълнява**.

**Примерна задача 6:**

static void Main(string[] args)

{

for (int i = 0; i <= 10; i++)

{

if (i == 8)

{

break; // проверяваме ако променливата i = 8 , прекратяваме цикъла с break и излизаме от него

}

Console.WriteLine("value is: " + i);

}

Console.ReadLine();

}

*Изразът Break прекъсва цикъла в текущата точка или можем да кажем, че прекратява условието на цикъл.* На примера можете да видите, как се показват числата от 0 до 7 , докато не стигнем условието на if-конструкцията при i = 8, когато състоянието на декларацията за прекъсване принуждава да се прекрати цикъла, така че числата 8, 9 и 10 няма да се покажат.

**Оператор Continue**

Операторът continue спира текущата итерация на най-вътрешния цикъл, без да излиза от него , т.е прескача текущата итерация и минава на следващата.

**Примерна задача 7:**

static void Main(string[] args)

{

for (int i = 0; i <= 10; i++)

{

if (i == 5)

{

continue; // тук при проверката ако i = 5 прескачаме отпечатването и продължаваме на следващата итерация

}

Console.WriteLine("value is: " + i);

}

Console.ReadLine();

}

Сега можете да видите изхода. Условието продължава да протича в зависимост от функционирането на оператора. **Тук само пропускаме итерацията, удовлетворяваща частта с if-конструкцията, като продължаваме цикъла от следващата по ред итерация.**