
ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ– СОФИЯ

# ИНФОРМАТИКА

## част първа

## Основи на програмирането на C++

**лектор: гл. ас. д-р Стефан М. Панов**

**Катедра “Информатика”**

# Лекция 5

# Инструкции за цикъл в C++

## Инструкция за цикъл while

Инструкция while служи за повторение на изпълнението на една или повече инструкции. Тя има следния синтаксис:

```
while (управляващ-израз)
{
    инструкции
}
```

Където управляващ-израз

може да бъде всеки израз, който връща стойност различна от void.

инструкции

могат да бъдат произволни инструкции, включително и друг **while**.

**Как се изпълнява:** Ако стойността на управляващ-израз е различна от нула, програмата преминава към изпълнение на инструкциите в тялото на while.

След това отново се проверява стойността на управляващ-израз. Ако стойността е различна от нула, инструкциите в тялото на while се изпълняват отново. Процесът се повтаря, докато стойността на управляващ-израз е различна от нула. Ако стойността на управляващ-израз е нула при първата проверка, инструкцията while се прескача и програмата преминава към изпълнение на първата инструкция след нея.

```
#include <iostream> // nashata 16-a programa
using namespace std; //
int main(){
	short int x = 58; // ako vmesto int e char ?
	while (x != 46) {
		cout << "x = " << x<< "\n";
		x--;
	}
	return 0;
}
```

## Елементи на цикъла

Показаната програма в предишния раздел ни дава удобна възможност да се запознаем с елементите на цикъла:

- **Инициализация** – **задаване на началната стойност на параметъра (управляващата променлива) на цикъла**. В нашия пример това е **x = 58;**
- **Тяло на цикъла** – **кодът, който трябва да се изпълни определен брой пъти.** За примера това са инструкциите поставени между къдравите скоби.
- **Актуализация** – **обновяване на стойността на параметъра на цикъла**. В програмата това е инструкцията **x--;**
- **Управляващ-израз (прекъсващо условие)** – **изразът, стойността на който спира или продължава действието на цикъла**. При нас това е **x != 46**

**Разгледаните накратко елементи се нуждаят от допълнителни разяснения:**

Пропускането на инициализацията създава неопределеност и не може да се каже нито дали цикълът изобщо ще се изпълни или пък ще се отиде във вариант когато броят на изпълненията се различава от замисъла на програмата. Същите проблеми се получават и при погрешна инициализация. Например в показаната програма всяка инициализация със стойност по-малка от 46 (примерно x = 38;) би довела до отпечатване на всички числа от зададеното до минус MAX INT, а после а после от MAX INT до 47. (Изпробвайте го в програмата, но с тип short int вместо int.)

Освен това трябва да се подчертае, че **инициализацията винаги се прави преди тялото на цикъла**. Лесно се вижда, че ако това изискване се наруши и се инициализира в тялото на цикъла има възможност за „вечен“ цикъл.

Циклите се делят на **индуктивни** (броят на повторенията БП може да се пресметне, както е в пример_16) и **итеративни** (БП не може да се пресметне).

От примера може да се убедим и във важността на добре подбраното прекъсващо условие. За примера по-добрият вариант би бил x > 46 вместо
x != 46. Тогава евентуална погрешна инициализация като вече предложената x = 38; би довела до прескачане на цикъла.

Също така е добре да се запомни, че актуализацията, поне за циклите **while** и **do-while** е част от тялото на цикъла. В някои учебници се говори за **Същинско тяло на цикъла** като се имат предвид всички инструкции от тялото без самата актуализация. **Пропускането на актуализацията също е груба грешка, която води до „вечен“ цикъл**.

## Инструкция за цикъл do-while

Инструкцията do-while служи за повторение на изпълнението на една или повече инструкции. Тя има следния синтаксис:

```
do
{
    инструкции
} while (управляващ-израз);
```

където

управляващ-израз

може да бъде всеки израз, който връща стойност различна от void.

инструкции

може да бъде всяка  инструкция, включително и друга do-while инструкция.

**Как се изпълнява:** При достигане до инструкция do-while, програмата първо изпълнява инструкциите в тялото на цикъла, след което се изчислява стойността на управляващ-израз.

Ако стойността на управляващ-израз е различна от нула, програмата отново преминава към изпълнение на инструкциите в тялото на do-while. Процесът се повтаря, докато стойността на управляващ-израз е различна от нула.

Ако стойността на управляващ-израз е нула (или false), операторът do-while се прескача и програмата преминава към изпълнение на първия оператор след do-while.

**Единствената разлика между while и do-while е, че инструкциите в тялото на do-while се изпълняват задължително поне един път.**

Получаването на следващата програма от програма 16 е много лесно:

```
#include <iostream> // nashata 17-a programa
using namespace std; //
int main(){
	short int x = 58; // ako vmesto int e char ?
	do {
		cout << "x = " << x<< "\n";
		x--;
	} while (x != 46); // prosto slojihme “do” i premestihme while ()
	return 0;
}
```

Използвайки do-while ние можем още повече да усъвършенстваме програмата “Отгатни магическото число”. **Тоя път програмата “няма да ни пусне” от цикъла за отгатване, докато не отгатнем числото.** Което не е трудно, ако отчитаме подадената ни информация.

```
// Progama "Otgatni magicheskoto chilso ".  Porednoto podobrenie
#include <iostream> // nashata 18-a programa
#include <cstdlib>
#include <time.h> // zaradi time()
using namespace std;
int main (){
    int magic; // magicheskoto chilso
    int guess; // variant na potrbitelia
    srand (time(NULL));// inicializira rand() s argumenta ot srand(). Poneje vremeto teche wseki pyt rand() shte dava razlichni stoinosti
    magic = rand(); // Poluchavame magicheskoto chilso. // rand() generira psevdo-sluchaino cialo polojitelno chislo
//  cout << "magic = "<<magic<<"\n\n";
    cout << "time = "<<time(NULL)<<"\n\n";
    do{
        cout << "Vavedete vashia variant na magicheskoto chislo: ";
        cin >> guess;
```

```
        if (guess == magic) {
            cout << "\n** Pravilno **\n";
            cout << "Imenno " <<magic << " e magicheskoto chislo.\n";
        }
        else {
            cout << "\nSyjaliavame, no sgreshihte.\n"; ;
            if(guess > magic) {
                cout << "Vashiqt izbor previshava magicheskoto chislo.\n";
            }
            else {
                cout << "Magicheskoto chislo previshava vashiqt izbor.\n";
            }
            cout << "izberete novo otchitaiki nalichnata informacia.\n\n";
        }
    } while (magic != guess);
    return 0;
}
```

Тъй като rand() е генератор на псевдослучайни числа всяко пускане на програмата би генерирало едно и също число. За да избегнем това се използва друга функция srand() имаща един аргумент. Когато srand() получава различни аргументи rand() също ще поражда различни числа. От своя страна функцията time(NULL) (именно извикана с параметъра NULL) връща броя на секундите, изминали от 1.1.1970 г. Следователно, при всяко пускане на нашата програма time(NULL) ще връща различно число, оттук и rand() ще генерира различни числа.

## Инструкция за цикъл for

Инструкцията for служи за повторение на изпълнението на една или повече инструкции. От всички инструкции за цикли тя е най-гъвкава и има следния синтаксис:

```
for(израз1; управляващ-израз ; израз3)
{
    инструкции
}
```

където

управляващ-израз

може да бъде всеки израз, който връща стойност, различна от void.

израз1/израз3 (т.е. инициализация/актуализация)

може да бъде произволен израз.

инструкции

може да бъде всяка инструкция (или последователност от инструкции), включително и друга for инструкция. **Как се изпълнява:**

При достигане до инструкция for програмата първо изчислява израз1, след което преминава към изчисляване на управляващ-израз.

Ако стойността на управляващ-израз е различна от нула, програмата преминава към изпълнение на инструкциите в тялото на for. След като изпълни инструкциите, програмата изчислява стойността на израз3, след което отново изчислява управляващ-израз. Ако стойността е различна от нула, инструкциите в тялото на for се изпълняват отново. Процесът се повтаря, докато стойността на управляващ-израз е различна от нула.

Ако стойността на управляващ-израз е нула, инструкцията for се прескача и програмата преминава към изпълнение на първата инструкция след for.

Да разгледаме следващия пример - програма_19 - получен от програма_16 като инструкция while е заменена от инструкция for. Макар че for е по-гъвкавата от двете инструкции от **практическа гледна точка много често for е инструкция while записана по-компактно – т.е. три от елементите на цикъла са записани на един ред – там където е ключовата дума for**.

След това важно уточнение да разгледаме особеностите на инструкция **for**.

```
#include <iostream> // nashata 19-a programa
using namespace std; //
int main(){
	short int x; // ako vmesto int e char ?
	for(x = 58; x != 46; x--) { // Ako wmesto 58 e 38 kakwo shte stane ?
		cout << "x = " << x<< "\n";
	}
	return 0;
}
```

**Местата, в които се разполагат израз1, управляващ-израз и израз3 могат да се оставят празни, но точките и запетаите задължително трябва да присъстват. Именно точките и запетаите показват какво е пропуснато, като могат да се пропуснат един, два или всичките 3 елемента.**

Три примера: **(израз1;;израз3) (;;) (;управляващ-израз;)**

Следва пример за програма, в която липсва израз3, т.е. актуализацията.

```
#include <iostream> // nashata 20-a programa
using namespace std; // primer za cikyl bez aktualizacia
int main()
{
	int x;
	cout << "Vavejdaite celi chisla. Za krai vavedete  chisloto 100 \xA\n";
	for(x=0; x != 100; ) {
		cout << "Vavedete chislo: ";
		cin>> x;
	}
	return 0;
}
```

Цикълът ще се изпълнява, докато не въведем от клавиатура 100.

Като правило разполагането на израз1 преди инструкция for се прави рядко, най-вече когато началната стойност се генерира от сложен процес, който е неудобно да се помести в определението на цикъла.

Цикълът for има следната особеност: ако управляващ-израз е празен **това се интерпретира от компилатора като ИСТИНА и цикълът става безкраен**

(но може да се излезе по друг начин, с break).

Ако тялото на цикъла е празно, примерно for (x=0; x<1000000000; x++) ; това е опростен таймер (цикъл за времева задръжка). Единствената задача на такъв цикъл е да изтече времето. В показания пример ; (точка и запетая) е задължителна, защото синтаксисът на инструкцията for очаква блок от инструкции или инструкция, а **; (точка и запетая) е празна инструкция**!

Също така израз1 и израз3 могат да се състоят от няколко израза разделени със запетая. Такъв случай имаме когато цикълът съдържа две управляващи

променливи - удачно е те да се инициализират и променят в израз1 и израз3 съответно. Ето такъв пример:

```
#include <iostream> // nashata 21-va programa

using namespace std; // primer za 2 upravliavashti promenlivi v cikyl

int main()
{
	int x, y;
	cout << "x\ty\n";// \t e tabulacia
	cout << "----------\n"; // kakvo shte stane, ako e x=10 ?
	for(y = 1, x = 11 ; x != y ; x--, y++) {
		cout << x << "\t" << y << "\n";
	}
	return 0;
}
```

**Използвайте израз1 и израз3 на цикъла for само за да инициализирате и променяте управляващи променливи на цикъла, в противен случай рискувате да направите кода си по-объркан и неясен.**

Следва пример за неудачно съставени израз1 и израз3.

```
#include <iostream> // nashata 22-ra programa
using namespace std ;
int main()// losh primer za 2  promenlivi pri inicializacia i aktualizaciq na cikyl
{
      int x, sum; // namirane na sumata na celite chisla ot 10 do 1
      for(sum = 0, x = 10 ; x != 0 ; sum += x , x--)
      {
// prazno tialo
      }
      cout << "sum = " <<  sum << "\n";
      return 0;
}
```

Следващият вариант на същата задача е много по-нагледен и позволява добавянето на коментари:

```
#include <iostream> // nashata 23-ta programa
using namespace std;// podobren variant na programa 22
int main()
{
	int x, sum; // namirane na sumata na celite chisla ot 10 do 1
	sum = 0;// nulirame promenlivata, koiato shte suhraniava rezultata
	for(x = 10 ; x != 0 ; x--)
	{
		sum += x ;// pribaviame sledvashtoto chislo
	}
	cout << "sum = " <<  sum << "\n";
	return 0;
}
```

Редът sum = 0; заслужава специално внимание: **Променливата, в която ще намираме каквато и да е сума се нулира в началото.** (инициализира с нула)

## Инструкция continue

Инструкцията continue може да се появява само в тялото на инструкциите за цикли while, do-while и for. Използването на continue извън тоя контекст ще доведе до грешка при компилиране. **Изпълнението на инструкция continue води до прескачане на всички инструкции, намиращи се след нея и започване на нова итерация (завъртане) на цикъла.** За циклите while и do-while това означава преход към изчисление на управляващ-израз, а за цикъл for преход към изчисление на израз3.

В примера, показан на следващата страница за всяко нечетно число се изпълнява инструкция continue, а това води до пропускане на инструкция cout.

**Ако инструкцията за цикъл, към която принадлежи continue е вложена в друга инструкция за цикъл, то continue води до стартиране на нова итерация само на вложения (вътрешния) цикъл!**

```
#include <iostream> // nashata 24-ta programa
using namespace std;
int main()
{
	int x; // otpechatvane na chetnite chisla ot 0 do 50
	for(x = 0 ; x <= 50 ; x++)
	{
		if (x%2) {
			continue;
		}
		cout<< x << " ";
	}
	return 0;
}
```

## Инструкция break

Инструкцията break може да се появява само в тялото на инструкция switch или в тялото на инструкциите за цикли while, do-while и for. Използването на break извън този контекст ще доведе до грешка при компилиране. **Резултатът от изпълнението на оператора break е прекратяване на изпълнението на инструкцията, към която той принадлежи и преход към следващата инструкция, намираща се след нея**.

**Обърнете внимание**, че ако инструкцията, към която принадлежи break, е вложена в друга инструкция за цикъл или switch, то break води до излизане само от вложената (вътрешната) инструкция!

Пример за вложени цикли е следващата програма. Когато сработи break се отива към реда след края на вътрешния for, т.е. на cout<< '\n'; Програмата ще отпечата 7 пъти цифрите от 0 до 9.

```
#include <iostream> // nashata 25-a programa
using namespace std; // primer za izpolzvane na break pri vlojeni cikli
int main() {
	int t, count;
	for(t=0; t<7; t++) {
		count = 0;
		for(;;) { // primer za vechen cikyl (ako go niamashe break)
			cout << count << ' ';
			count++;
			if (count == 10) {
				break;
			}
		}
		cout<< '\n';
	}
	return 0;
}
```

## Вложени цикли

Както беше показано в предишната програма, един цикъл може да се вложи в друг. **В C++ е разрешено да се използват до 256 нива на влагане.** Вложените цикли се използват за решаване на най-разнообразни задачи.
Например, в следващата програма с два вложени цикъла for се намират простите числа в обхвата от 2 до 1000. Числото, съдържащо се в променливата i, се дели последователно на стойностите, разположени между числото 2 и резултата от изчислението на израза i/j. (Можем да спрем проверката за множители на стойността на израза i/j, защото за число, което превишава i/j, ще бъде намерен другия множител по-рано, ако го има.) Ако остатъкът от делението i/j е равен на 0, то числото i не е просто. Но ако вътрешният цикъл завърши напълно, т.е. без предсрочно завършване чрез инструкцията break, това означава, че текущата стойност на променливата i действително е просто число. (**Да се разгледат числата 143 = 11*13, 49 и 53**.)

```
#include <iostream> // nashata 26-a programa
using namespace std; // namirane na prostite chisla mejdu 2 i 1000
int main() {// ako chisloto i ne e prosto ediniat mu mnojitel e <= sqrt(i)
	int i, j, count = 0; // count - za broia na prostite chisla v obhvata
	cout << "Prostite chisla v obhvata 2-1000 sa:\n\n";
	for(i=2; i<1000; i++) {
		for(j=2; j <= (i/j) ; j++){
			if (!(i%j)) { break; }// Ako chisloto ima delitel, znachi ne e prosto
		}
		if(j > (i/j)) {
			cout << " "<< i; // i e prosto chislo
			count++;
		}
	}
	cout << "\n\nBroiat na prostite chisla v obhvata 2-1000 e "<< count << "\n";
	return 0;
}
```

## Инструкция за безусловен преход goto

**Използването на инструкция goto изисква наличието на етикет.**

**Етикет – идентификатор, след когото е поставено двоеточие.** Правила:

- При избора на етикет се спазват същите правила, както и при име на променлива.
- Етикет може да се постави пред всяка инструкция. Етикираната инструкция може да се намира преди или след goto инструкцията, която прави преход към нея.
- Областта на действие на goto е само тялото на функцията, в която goto се използва. Преход между функции с помощта на goto е невъзможен.

Следващата програма е получена от програма_17 с използването на goto вместо do-while. (Следва да се подчертае, че реализацията в програма_17 е по-добра от тая в новата програма_27.)

```
#include <iostream> // nashata 27-a programa
using namespace std; // poluchena e ot programa 17 s izpolzvane na instrukcia goto vmesto do-while
int main(){
	short int x = 58;
	pak:
		cout << "x = " << x << "\n";
		x--;
	if  (x != 46) {
		goto pak;
	}
	return 0;
}
```

Етикетът pak: може да бъде поставен на същия ред където е инструкцията cout, но е поставен на отделен ред за по-добра прегледност на кода. Както се вижда от примера и с goto може да се организира цикъл.

Използването на инструкция goto може да направи програмата доста заплетена. Някои специалисти препоръчват тя да не си използва изобщо. Още повече, че е **доказано**, че всяка програма, където се среща goto може да се пренапише така, че goto да не се среща.

От друга страна има ситуации - примерно: **много вложени един в друг цикли и е необходимо от най-вътрешния, когато е изпълнено определено условие, да се излезе от всички** - при които употребата на goto дава естествено и удобно решение, което ни спестява много труд.

**Затова, нека гледаме на goto като на силно лекарство. Да се използва предпазливо и само там, където наистина е нужно и разумно.**

**В заключение отново относно употребата на къдрави скоби при цикли и if:**

**! Препоръчително е винаги да се използват къдрави скоби, даже когато тялото на if, for, while или do-while се състои от единична инструкция.**