Тема 7
Управляващи конструкции за организиране на програмни цикли в език C/C++

Конструкциите за реализация на програмни цикли, наричани още **итерационни**, в C/C++ са част от управляващите. В езика се предвидени общо три конструкции, които реализират цикли в програмите: ***while, do-while, for***. И при трите конструкции тялото на цикъла се състои само от един оператор. Ако алгоритъмът изисква в тялото да са повече от един оператори, те се обединяват чрез съставния оператор **{…}**. Самите конструкции ***while***, ***do-while*** и ***for*** се третират като една конструкция.

1. Конструкция за цикъл с предусловие *while*

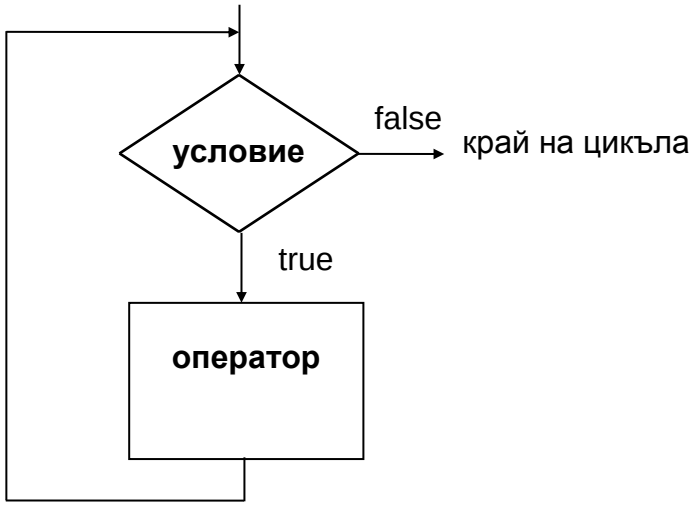
Управляваща конструкция ***while*** реализира цикъл с пред условие. Конструкцията има следния общ вид:

while (условие) оператор;

Операторът след условието може да е съставен. В този случай, конструкцията ***while*** има следния общ вид:

```
while (условие)
{
    // група оператори, съставляващи тялото на цикъла
}
```

Действието на конструкцията е следното: проверява се условието; ако стойността на израза е различна от 0 или ***true***, изпълнява се тялото на цикъла и отново се проверява условието за край, т.е. *операторът в тялото за цикъла се изпълнява, докато условието е вярно*. Когато стойността на условието стане равна на 0 или ***false***, прекратява се изпълнението на цикъла. Действието на конструкцията ***while*** може да бъде представено чрез блоковата диаграма на фиг. 8.1.



*Фиг. 8.1. Действие на итерационна конструкция **while***

2. Конструкция за цикъл със след условие *do-while*

Конструкцията ***do-while*** реализира цикъл със след условие (постусловие). Тя има следния общ вид:

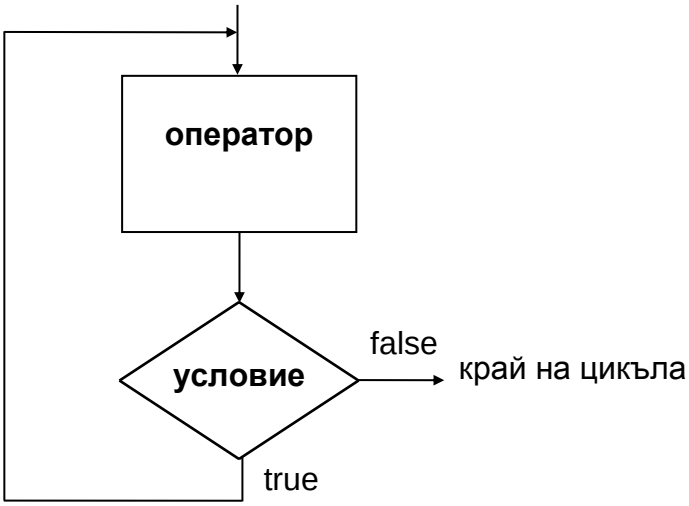
do оператор;

while (условие);

Операторът след ключова дума ***do*** може да е съставен. В случая, общият вид на конструкцията е следния:

```
do {
    // група оператори, съставляващи тялото на цикъла
}
while (условие);
```

Действието на ***do-while*** е следното: *операторът в тялото на цикъла се изпълнява до тогава, докато условието е вярно* т.е. докато изразът представящ условието има стойност различна от 0 или ***true***. Когато стойността на израза стане равна на 0 или ***false***, прекратява се изпълнението на цикъла. Действието на оператор ***do-while*** може да бъде представено чрез блоковата диаграма на фиг. 8.2.



*Фиг. 8.2. Действие на итерационна конструкция **do-while***

Действието на ***while*** и ***do-while*** е идентично. Основната разлика между двете конструкции произтича от това, че първата реализира цикъл с предусловие, а втората - цикъл със следусловие. Съответно, при ***while*** проверката за изход от цикъла е преди тялото, а при ***do-while*** – след тялото на цикъла. Следователно възможно е, тялото на оператор ***while*** да не бъде изпълнено нито веднъж, докато тялото на оператор ***do-while*** ще бъде изпълнено поне веднъж.

Използването на конструкции ***while*** и ***do-while*** може да бъде илюстрирано чрез задачата за намиране на N факториел.

```
// пресмятане на  N! чрез използване на оператор while
#include <stdio.h>
main()
{
      int n, nf;
      int i;

      printf("n=");
      scanf("%d",&n);

      nf=1;
      i=1;
      while (i<=n)
      {
            nf=nf*i;
            i++;
      }
      printf("\nN facturiel = %d\n\n",nf);
      return 0;
}
```

В примера променливата nf съдържа търсеният резултат, n е числото на което се търси N факториел, а променливата i е брояч, който изменя стойността си от 1 до n през 1. При всяка итерация, стойността на променливата nf се увеличава като се умножава по текущата стойност на промеливата i т.е. nf=1*2*3*4*..*n.

При използване на конструкция ***do-while***, програмата, реализираща пресмятането на N факториел, има следния вид:

```
#include <stdio.h>
main()
{
      int n, nf;
      int i;

      printf("n=");
      scanf("%d",&n);

      nf=1;
      i=1;
      do
      {
            nf=nf*i;
            i++;
      }
      while (i<=n);
      printf("\nN facturiel = %d\n\n",nf);
      return 0;
}
```

На практика, действието на двата програмни фрагмента е идентично. В този пример, няма разлика при използването на конструкции ***while*** и ***do-while***. Пример за друг програмен фрагмент, който реализаира намирането на N факториел чрез използване на конструкция ***do-while*** е следният:

```
…
int nf=1;

do
 {
```

```
        nf=nf*n;
        n--;
 }
 while (n>0);
…
```

И двата варианта на намиране на N факториел могат да бъдат реализирани както с конструкция ***while***, така и с конструкция ***do-while***. Има обаче случаи, когато използването на едната конструкция е за предпочитане. Например, ако искаме да организираме въвеждане на стойност в цикъл, който да се повтаря докато потребителят не въведе стойност в определен интервал, тогава е по-добре да се използва оператор ***do-while***. Пример за такъв програмен фрагмент е:

```
…
int k;
do
{
        printf("\nk=");
        scanf("%d",&k);
}
while ((k<1)||(k>10));
…
```

В случая се изисква да се въведе стойност за променливата k в интервала [1,10]. Ако се използва оператор ***while***, k трябва да бъде предварително инициализирана със стойност извън този интервал.

3. Итерационна конструкция *for*

Конструкция ***for*** се използва предимно за реализация на цикли с известен брой повторения. По своето действие тя реализира цикъл с предусловие. Синтактично общият вид на конструкцията е следния:

for (секция 1;секция 2;секция 3) оператор;

Операторът след скобите и операторите в *секция 3* съставляват тялото на цикъла. Операторът след скобите може да е съставен. В случая, общият вид на конструкцията е следния:

for (секция 1;секция 2;секция 3)

{ // група оператори, съставляващи тялото на цикъла

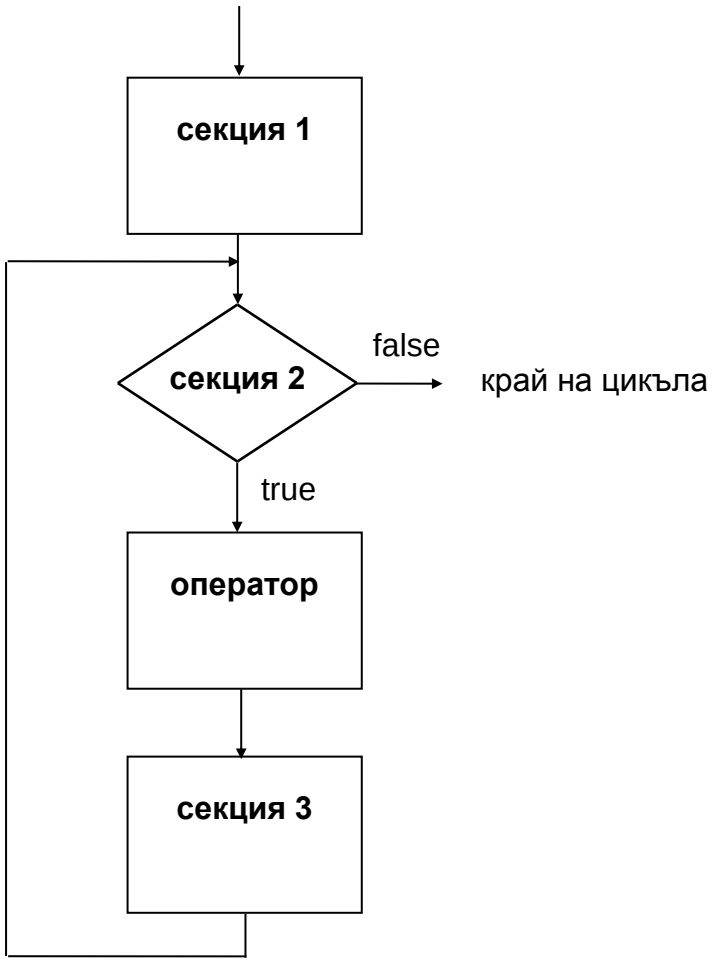
}

Секция 1 и секция 3 се състоят от един или няколко оператора, отделени със запетая. Секция 2 е условен израз.

Действието на конструкция ***for*** е следното:

- Изпълняват се действията в секция 1. Това е т.нар. инициализираща секция, която се изпълнява еднократно и с която се задават началните условия на цикъла.
- Изпълнението на ***for*** продължава в следната последователност: изчисляване на израза в секция 2; изпълнение на оператора след

скобите; изпълнение на секция 3. Тази последователност се изпълнява дотогава, докато стойността на израза в секция 2 има стойност различна от 0 или ***true***.



*Фиг. 8.3. Действие на итерационна конструкция **for***

Действието на итерационната конструкция ***for*** може да бъде представено чрез блоковата диаграма на фиг. 8.3. Операторите в секции 1 и 3 могат да са произволни, но най-често се използват следните: за инициализация в секция 1 и за промяната на стойностите след всяка итерация в секция 3.

Използването на конструкция ***for*** може да бъде илюстрирано чрез примера за пресмятане на N факториел:

```
#include <stdio.h>
main()
{
      int n, nf, i;

      printf("n=");
      scanf("%d",&n);

      nf=1;
      for(i=1;i<=n;i++) nf=nf*i;
```

```
      printf("\nN facturiel = %d\n\n",nf);
      return 0;
}
```

Тъй като операторите в *секция 1* могат да са повече от един, цикълът може да се организира и така:

```
…
for(i=1,nf=1;i<=n;i++)  nf*=i;
…
```

В посочената реализация отново променливата i се използва като брояч, чиято стойност се изменя от 1 до n. Обикновено, в секция 1 на конструкция ***for*** е инициализацията на брояча, а в секция 3 е нарастването (или намаляването) на неговата стойност.

Друг пример за използване на конструнция ***for*** е следващата програма, която извежда на екрана всички цели числа в даден интервал [m,n] в намаляваш ред. Ако въведените числа са такива, че m е по-голямо от n, в програмата е предвидено променливите да разменят своите стойности.

```
#include <stdio.h>
main()
{
      int n, m;

      printf("\nEnter first number:");
      scanf("%d",&m);
      printf("\nEnter second number:");
      scanf("%d",&n);

      if( m > n)
      {                  //размяна на двете стойности
            n=n+m;
            m=n-m;
            n=n-m;
      }
      for(int i=n; i>=m; i--) printf("%d ",i);
      return 0;
}
```

Секциите в конструкция ***for*** могат да са празни. Следващите програмни фрагменти са примери, в които се реализира извеждане на числата от 1 до 10 на екрана е посочено как може да се използва конструкция ***for***, като последователно са празни секции 1, 2 и 3, съответно.

Пример:

```
// извеждане на числата от 1 до 10 на екрана, като секция1 е празна
// операторът от секция1 е изнесен преди конструкция for
int i=1;
for(; i<=10;i++)
printf("%d\n",i);


// извеждане на числата от 1 до 10 на екрана, като секция2 е празна
// изходът от цикъла се реализира чрез оператор break
```

```
for(int i=1;  ; i++)
 {
    printf("%d\n",i);
    if (i==10) break;
}

// извеждане на числата от 1 до 10 на екрана, като секция3 е празна
// операторът от секция3 е пренесен в тялото на конструкция for
for(int i=1;i<=10;)
{
 printf("%d\n",i);
 i++;
}
```

Важно е да се отбележи, че когато в конструкция ***for*** липсва *секция 2*, необходимо е да бъдат взети мерки за изход от цикъла. В противен случай цикълът се превръща в безкраен.

4. Конструкция *continue*

Конструкция ***continue*** може да се реализира в тяло на конструкциите за организиране на програмен цикъл ***while***, ***do-while***, ***for***. Общият вид на конструкцията е:

continue;

Поставена в тяло на цикъл, конструкция ***continue*** предизвиква начало на нова ***итерация***, без да бъдат изпълнени операторите в тялото на цикъла, които са след ***continue***. Под понятието ***итерация*** се разбира едно повторение на цикъла.

Пример:

```
// Намиране броя на положителните числа, въведени от клавиатурата
// Въвеждане на стойност 0 служи за изход

#include <iostream.h>
void main()
{
      int count=0;
      int x;
      do
          {
            cout <<"\nInput x=  ";
            cin >>x;
            if(x<=0) continue;
            count ++;
      }
      while(x!=0);
      cout<<"The affirmatives are: "<<count;
}
```

В посочения пример, ако се използва конструкция ***while*** вместо ***do-while***, променливата **x** трябва да се инициализира със стойност, различна от 0. От примера може да се разбере разликата в приложението на двата оператора.

Тема 8
Дефиниране и използване на масиви в език C/C++

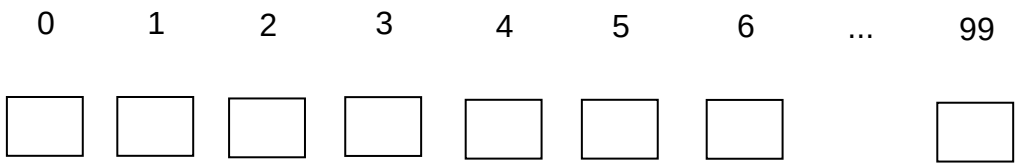
1. Обща характеристика на тип данни *масив*

Типовете данни в програмните езици се разделят най-общо на ***прости (скаларни)*** и ***структурирани***. Скаларните типове данни са тези, които се състоят от един елемент. В C/C++, пример за такива данни са аритметичните типове: ***int***, ***char***, ***float*** и ***double***. Структурираните типове се състоят от повече от един елемент.

Масивът е структура от данни, състояща се от множество последователно наредени елементи от един и същи тип, достъпът до който се осъществява чрез името на масива и поредния номер на елемента т.нар. **индекс**.Типът **масив** се характеризира със следните особености:

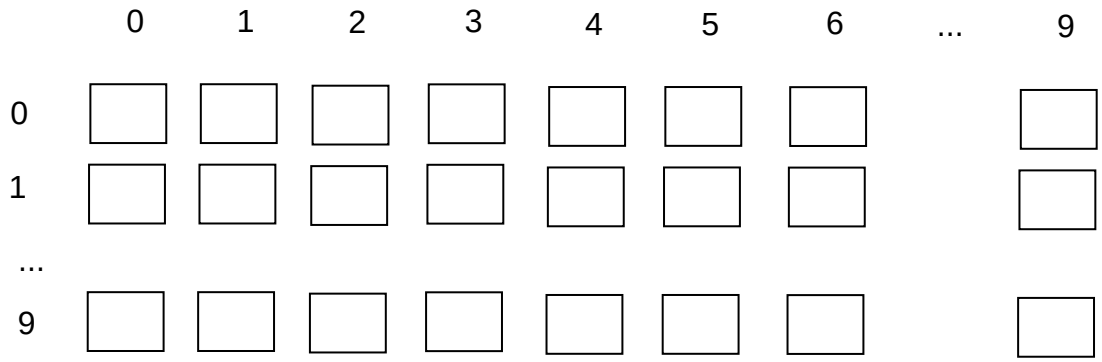
- *структуриран тип данни от еднотипни елементи* - за разлика от простите типове, масивите се състоят от повече от един елемент, като елементите задължително са от един и същи тип. Типът на елемента определя типа на масива.
- *структуриран тип данни от статичен вид* - броят на елементите на масива трябва да се определи по време на компилация; той не се изменя в процеса на изпълнение на програмата. Компилаторът заделя необходимата памет.
- *индексиран тип данни* - достъпът до даден елемент се осъществява чрез името на масива и неговият ***индекс*** (пореден номер).

Масивите, според размерността си се разделят на: ***едномерни*** и ***многомерни***. Многомерните масиви се разделят на двумерни, тримерни и т.н. Едномерните масиви могат да се представят като редица от последователно наредени елементи (фиг. 9.1). Двумерните се представят като матрици (фиг. 9.2). В тримерното пространство е трудно да се даде представа за масив с повече от три индекса и затова такъв тип масиви рядко се използва.



Фиг. 9.1. Структура на едномерен масив от 100 елемента

В паметта на компютъра многомерните масиви се представят като едномерни. Например, двумерен масив се нарежда последователно ред по ред. Двумерният масив се третира като *едномерен масив, чиито елементи са едномерни масиви*.



Фиг. 9.2. Структура на двумерен масив от 10x10 елемента

2. Дефиниране на масиви. Достъп до елементите

Общият вид на дефиницията на едномерен масив е следния:

имеТип имеМасив[размерност];

Съответно:

имеТип - определя типа на елементите на масива; може да е някой от стандартните типове или дефиниран от програмиста;

имеМасив е *идентификатор*, определящ името на променливата от тип *масив*;

размерност - константен израз, който определя броя на елементите в масива и се нарича още ***граница***.

Примери за дефиниции на масиви са:

```
int massiv[10];    /* дефиниране на едномерен масив massiv от 10 цели елемента */
char str[20];    // дефиниране на едномерен масив ch от 20 символа
float a[10], b[20];  /* дефиниране на два едномерни, реални масива a и b от 10 и 20, елемента съответно */
```

Достъпът до даден елемент на масива е чрез името на масива и индекса на елемента. Например:

```
massiv[0] = 10;
a[2]=0.5;
b[18]=1.9;
str[10]='y';
```

Особеност на език C/C++ относно масивите е, че индексите на елементите започват от **0** (нула), т.е. ако даден масив е с **n** елемента, първият елемент е с номер **0**, а последният е с номер **n-1**. Например, масив **a** от 10 елемента има следните индекси :

a[0],a[1],a[2], a[3], a[4], a[5], a[6], a[7], a[8],a[9];

Дефинирането на многомерни масиви е чрез следната декларация:

име_тип име_масив[граница1][граница2].....;

Примери:

```
float ax[10][20];  /* дефиниране на двумерен масив ax от 10x20 реални елемента  */
int bx [10][10][5];      /* дефиниране на тримерен масив bx от 10x10x5 целочислени елемента  */
```

Индексите при многомерните масиви се изменят по същия начин като едномерните: от **0** до **n-1**.

Двумерният масив е частен случай на многомерен. Достъпът до елементите е чрез два индекса. Подобно на математическото описание на матрица, първият индекс е за редове, вторият за стълбове.

Примери за достъп до елементи на многомерни масиви са:

```
ax[0][0]=0;
bx[0][0][0]=1;
```

Относно разположението на многомерните масиви в паметта на компютъра, елементите се подреждат последователно, по реда на нарастване на индексите, като най-бързо нарастват десните индекси.

Например, подредбата на матрица ***ax*** в паметта на компютъра е:

ax[0][0] ax[0][1] ax[0][2]……..ax[0][19] ax[1,0]……

3. Примерни задачи за работа с масиви

3.1. Едномерни масиви

За обработката на масиви се използват итерационните конструкции и в програмен език C/C++ това най-често е конструкция ***for***, чрез която се обхожда масива от първия до последния елемент или обратно.

Пример 1. Въвеждане на елементите на едномерен масив от целочислени елементи от клавиатурата и извеждане на стойностите в обратен ред.

Задачата е пример за обхождане елементите на масива от първия до последния и обратно – от последния до първия.

```
#include <stdio.h>
main()
{
      int mA[10],i;
      for(i=0; i<10; i++)     //въвеждане стойности на елементите
```

```
	{
		printf("Item[%d]=",i);
		scanf("%d",&mA[i]);
	}

	printf("\n");		// извеждане стойностите на елементите
	for(i=9; i>=0; i--) printf("%d ",mA[i]);

	return 0;
}
```

Пример 2. Търсене на средно аритметична стойност.

Задачата е да се намери средноаритметичната стойност от елементите на едномерен масив от 10 реални елемента с двойна точност. В реализацията, при намирането на общата сума на елементите, чрез конструкция ***for*** се обхожда масивът и при всяка итерация към общата сума се добавя стойността на поредния елемент.

```
#include <stdio.h>
main()
{

	double mB[10];
	int i;
	double suma, average;
	for(i=0;i<10;i++)	//въвеждане стойностите на елементите
	{
		printf("Item[%d]=",i);
		scanf("%lf",&mB[i]);
	}

	suma = 0;		// намиране сумата на елементите
	for(i=0; i<10; i++) suma += mB[i];
	average = suma/10;	// намиране на средноаритметична стойност
	printf("\nAverage = %lf",average);

	return 0;
}
```

3.2. Двумерни масиви

При работата с двумерни масиви е необходимо да се използват вложени цикли. Обикновено итерациите се реализират с две вложени една в друга конструкции ***for***.

Пример 1. Въвеждане и извеждане елементите на друмерен масив.

Задачата е да се въведат стойностите на двумерен масив от 5x4 целочислени елемента и да се изведат на екрана в подходящ фомат. Примерното решение е следното:

```
#include <stdio.h>
main()
{

	int matrix[5][4];
	int i,j;

	for(i=0;i<5;i++)
```

```
	{
		for(j=0;j<4;j++)
		{
			printf("Item[%d][%d]=",i,j);
			scanf("%d",&matrix[i][j]);
		}
	}

	for(i=0;i<5;i++)
	{
		printf("\n");
		for(j=0;j<4;j++)
		{
			printf("%d ",matrix[i][j]);
		}
	}
	return 0;
}
```

4. Инициализиране на масиви

При дефинирането на масиви могат да бъдат зададени начални стойности на елементите, т.е. *масивите да бъдат инициализирани*. Определянето на тип масив, заедно със задаване на начални стойности на елементите е чрез следната дефиниция:

име_тип име_масив [размерност]={списък стойности};

Стойностите на елементите се разделят със запетаи. Пример:

```
int array[3] = {1, 3 , 5};
```

Съответно стойностите на елементите са: array[0]=1; array[1]=3; array[2]=5.

Възможно е, когато се инициализира масив, да не се задава брой на елементите. Тази стойност се определя от броя на зададените стойности. Пример:

```
double d[] = {2.23, 0, 18.5, 4.8};
```

Масивът d е от четири реални елемента с двойна точност; стойностите на елементите са: d[0]=2.23; d[1]=0; d[2]=18.5; d[3]=4.8

Ако размерът на масива е по-голям от броя на зададените стойности, инициализират се първите елементи с посочените стойности, а останалите приемат стойност **0** (нула). Пример:

```
double d[10] = {2.23, 0, 18.5, 4.8};
```

В случая, задават се стойности само на първите четири елемента, а останалите шест приемат стойност 0.

При инициализацията на двумерни масиви се изисква да се зададе множество от списъци, съответстващи на редовете на двумерният масив.

Пример:

```
int array[2][3] = { {1,1,0}, {0,1,1}
                  };
```