Cursul 10

Instrucţiuni C\C++
(partea a II-a)

III. Instrucţiuni de ciclare

Instrucţiunile de ciclare (repetitive, iterative, etc) sunt utilizate în situaţia în care o anumită secvenţă de cod trebuie executată de mai multe ori în mod consecutiv, numărul de repetiţii fiind fix, stabilit iniţial, sau variabil, depinzând de rezultatul acţiunii repetate.

Limbajul C pune la dispoziţia programatorilor trei instrucţiuni de ciclare: ***while***, ***do-while*** şi ***for***, toate trei utilizând pentru continuarea ciclării o condiţie de tip "cât timp este adevărat că …", diferenţa dintre ele fiind dată, în principal, de momentul testării condiţiei de ciclare: în cazul instrucţiunii ***while*** testarea are loc înaintea fiecărei execuţii a acţiunii repetate (*testare anterioară*), în cazul ***do-while*** testarea se face la reluare, după fiecare execuţie (*testare posterioară*), iar în cazul instrucţiunii ***for*** testarea se face analog cazului ***while***, instrucţiunea ***for*** având însă prevăzută şi evaluarea unei anumite expresii înainte de fiecare reluare, precum şi o secvenţă iniţială, executată înaintea intrării în ciclare.

Instrucţiunile repetitive păstrează controlul fluxului de execuţie al programului până când are loc *ieşirea din ciclare*. O ieşire din ciclare poate fi *normală*, atunci când controlul este preluat de următoarea instrucţiune din textul sursă în urma evaluării condiţiei de repetare a ciclării, sau poate fi *forţată*, prin executarea unei instrucţiuni de salt aflate în corpul instrucţiunii repetitive.

În continuare, vom descrie cele trei instrucţiuni iterative ale limbajului C în situaţia în care ieşirile sunt normale, urmând ca ieşirile forţate din ciclare să fie tratate o dată cu instrucţiunile de salt.

7. Instrucţiunea ***while***

Sintaxa: ***while*** **(***expresie-test***)** *instrucţiune-corp*

Instrucţiunea ***while*** îşi inserează în mod repetat în fluxul de execuţie *instrucţiunea sa corp*, atât timp cât *expresia-test* este adevarată (are o valoare nenulă). Este asemănătoare cu instrucţiunea ***if***, cu diferenţa esenţială că instrucţiunea ***while*** păstrează controlul după executarea instrucţiunii corp, reluând evaluarea expresiei-test. Controlul execuţiei trece la următoarea instrucţiune din textul sursă al programului (dacă în corp nu sunt prevăzute salturi) numai atunci când expresia test devine falsă (egală cu zero). Dacă expresia test este falsă de la bun început, instrucţiunea corp nu este executată, se trece la următoarea instrucţiune din textul sursă.

Tipul *expresiei-test* trebuie sa fie boolean sau scalar (număr sau pointer, caz în care are loc conversia implicită la tipul ***bool***) şi la evaluare sunt completate toate efectele ei secundare înaintea luării deciziei de contiunare sau de terminare a ciclării. *Instrucţiunea-corp* poate fi de orice tip (inclusiv o instrucţiune ***while***), dar ca să aibe sens utilizarea ei trebuie sa fie o instrucţiune efectivă şi nu o declaraţie, de exemplu. De regulă, instrucţiunea corp este o instrucţiune compusă, conţinând o secvenţă de cod care descrie acţiunea care trebuie repetată.

De exemplu, pentru a tipări codurile ASCII ale caracterelor stringului ***text***, se poate utiliza următorul şir de instrucţiuni:

```
...
int i=0;
while(text[i]!='\0'){
      cout<<(int)text[i]<<" ";
      i++;
}
...
```

Dacă şirul de caractere este vid (adică `text[0]== '\0'`), nu este tiparit nici un cod ASCII.

Secventa de mai sus poate fi compactată astfel:

```
...
int i=0;
while(text[i]!='\0')
      cout<<(int)text[i++]<<" ";
cout<<endl;
...
```

sau

```
...
int ch,i=0;
while((ch=text[i++])!='\0')
      cout<<ch<<" ";
cout<<endl;
...
```

Primele două variante sunt complet echivalente, a treia, deşi execută acelaşi lucru, diferă de primele prin valoarea indexului ***i*** la ieşirea din ciclare: deoarece în ultima variantă incrementarea indexului are loc la evaluarea condiţiei de continuare, ***i*** este incrementat şi în cazul în care se decide ieşirea din ciclare, astfel că în final valoarea indexului depăşeşte cu o unitate locul terminatorului de string (caracterul nul).

8. Instrucţiunea *do-while*

Sintaxa: ***do*** *instrucţiune-corp* ***while*** **(***expresie-test***);**

Instrucţiunile ***while*** şi ***do-while*** sunt foarte asemănătoare, singura diferenţă apărând doar la iniţierea ciclării: la start, ***do-while*** execută o dată, în mod necondiţionat, instrucţiunea sa corp şi numai după aceasta intră în ciclul testare – repetare, spre deosebire de **while** care începe direct cu evaluarea expresiei test. In concluzie, instrucţiunea **do-while** este echivalentă cu următoarea secvenţă de instrucţiuni:

instrucţiune-corp
while **(***expresie-test***)** *instrucţiune-corp*

De remarcat că sintaxa instrucţiunii ***do-while*** prevede existenţa terminatorului "**;**"care nu poate fi înlocuit cu nimic altceva (aici nu are sensul de "instrucţiune nulă").

Instrucţiunea ***do-while*** este de preferat în locul instrucţiunii ***while*** în situaţia în care acţiunea iterată trebuie executată cel puţin o dată. De exemplu, în programul următor utilizatorul trebuie să introducă mai întâi un număr pentru ca acesta să poată fi testat dacă este sau nu strict pozitiv:

```
#include<iostream>
using namespace std;
int main(){

	int a;
	do{
		cout<<"Dati un numar strict pozitiv, a=";
		cin>>a;
	}while(a<=0);
	cout<<"Avem a="<<a<<endl;
	return 0;
}
/* EXEMPLU:
Dati un numar pozitiv, a=-2
Dati un numar pozitiv, a=-10
Dati un numar pozitiv, a=12
Avem a=12
Press any key to continue . . .*/
```

Un alt exemplu: pentru a copia tot stringul ***mesaj***, inclusiv terminatorul de string, este de preferat utilizarea instrucţiunii ***do-while***:

```
#include<iostream>
using namespace std;
const int dimMax=100;
int main(){
	char ch,text[dimMax],mesaj[dimMax]="Un exemplu banal!";
	int i=0;
	do {
		ch=text[i]=mesaj[i];
		i++;
	} while (ch!='\0');
	cout<<text<<endl;
	return 0;
}
```

Dacă şirul de caractere text este vid, este copiat numai codul caracterului nul (zero).

Atenţie, următoarea compactificare duce la o expresie ambiguă:

```
...
do
	ch=text[i]=mesaj[i++];
while (ch!='\0');
cout<<text<<endl;
...
```

deoarece nu se poate preciza momentul exact al incrementării indexului ***i***.

Instrucţiunile ***while*** şi ***do-while*** sunt preferate în locul instrucţiunii ***for*** atunci când expresia-test nu depinde direct de numărul de ordine al iteraţiei curente, aşa cum a fost cazul în exemplele date.

9. Instrucţiunea *for*

Sintaxa: ***for*** **(** *expresie-iniţială*; *expresie-test*; *expresie-reluare*) *instrucţiune-corp*

Instrucţiunea ***for***, cu o sintaxă mult mai complexă decât celelalte instrucţiuni de ciclare, furnizează o scriere compactată a etapelor standard de execuţie a unui proces iterativ: iniţializarea

variabilelor înaintea startului, evaluarea testului de ciclare, execuţia acţiunilor din corpul ciclului, actualizarea variabilelor implicate în expresia-test, reluarea testării. In general (cu câteva excepţii care vor fi precizate ulterior) instrucţiunea ***for*** este echivalentă cu urmatoarea secvenţă de instrucţiuni:

expresie-inţială;
while (*expresie-test*) {
instrucţiune-corp
expresie-reluare;
}

De exemplu, iniţializarea cu zero a componentelor tabloului ***tab*** executată de secvenţa de instrucţiuni:

```
…
      i=0;
while (i<100){
      tab[i]=0;
      i++;
}
…
```

poate fi efectuată de o singură instrucţiune ***for***:

```
…
for ( i=0; i<100; i++) tab[i]=0;
…
```

Execuţia instrucţiunii ***for*** începe cu evaluarea *expresie-iniţiale* şi completarea tuturor efectelor sale secundare. Expresia iniţială poate fi de orice tip, scopul ei este de iniţializare a valorilor variabilelor implicate în ciclare (dacă trebuie facute mai multe atribuiri se poate folosi operatorul virgulă, desigur). Ea este evaluată o singură dată, imediat înainte de intrarea în ciclarea propriu-zisă. După evaluarea expresiei de iniţializare se trece la evaluarea *expresiei-test* (şi completarea efectelor ei secundare). Expresia test trebuie să fie de tip boolean sau scalar. Dacă rezultatul evaluarii e fals execuţia instrucţiunii ***for*** se încheie aici, iar controlul este preluat de următoarea instrucţiune din textul sursă. Dacă expresia-test este adevarată se continuă cu execuţia *instrucţiunii-corp*. După finalizarea acţiunilor descrise de corpul ciclui ***for*** (instrucţiunea-corp) şi imediat înainte de reluarea testării este evaluată *expresia-reluare*. Expresia-reluare poate avea orice tip, ea este utilizată mai ales pentru actualizarea valorii variabilei index. După completarea efectelor secundare a expresiei-reluare se reia procesul iterativ prin re-evaluarea expresiei test.

Oricare dintre cele trei expresii din *antetul* instrucţiunii ***for*** poate lipsi, doar prezenţa celor două simboluri punct şi virgulă este obligatorie. In cazul în care lipseşte *expresia-test* (fapt interzis pentru ***if*** sau pentru ***while***) se consideră ca testul de reluare a iteraţiei este în mod implicit satisfăcut. In acest caz, dacă nu este prevăzută ieşirea forţată (prin salt) din ciclare, execuţia instrucţiunii ***for*** continuă la nesfârşit şi, în consecinţă, blocheaza execuţia programului.

De reţinut: următoarea formă a instrucţiunei ***for***:

for (; *expresie-test*;) *instrucţiune-corp*

este echivalentă cu

while (*expresie-test*) *instrucţiune-corp*

în timp ce forma:

for (; ;) *instrucţiune-corp*

este echivalentă cu

while (***true***) *instrucţiune-corp*.

Următoarele exemple ilustrează modul uzual de utilizare al instrucţiunii ***for***.

Exemplul 1. Sumarea din trei în trei a elementelor unui tablou uni-dimensional, ***tab***, începand cu ***tab***[2]:

```
...
for( suma=0,i=2; i<n; i+=3) suma+=tab[i];
...
```

Exemplul 2. Inversarea ordinei elementelor tabloului ***tab***:

```
...
for( i=0,j=n-1; i<j; i++,j--){
      aux=tab[i];
      tab[i]=tab[j];
      tab[j]=aux;
}
...
```

Exemplul 3. Determinarea valorii maxime a elementelor matricei ***mat***:

```
...
for(valMax=mat[0][0],i=0; i<n; i++)
      for (j=0; j<m; j++)
            if (valMax < mat[i][j]) valMax=mat[i][j];
cout<<"valoarea maxima="<<valMax<<endl;
...
```

In final menţionăm că în C++ sintaxa instrucţiunii ***for*** permite ca în locul expresiei iniţiale să apară o instrucţiune declaraţie, în acest caz domeniul de vizibilitate al variabilelor declarate aici este restrâns la blocul format de instrucţiunea ***for*** respectivă. Mai precis, următoarea secvenţă de cod

```
int n=10;
for(int i=1,s=0;i<=n; i++){
      s+=i;
      cout<<"i="<<i<<" s="<<s<<endl;
}
```

este echivalentă cu

```
int n=10;
{
      int i=1,s=0;
      while(i<=n){
            s+=i;
            cout<<"i="<<i<<" s="<<s<<endl;
            i++;
      }
}
```

Exerciţiu: ce eroare de compilare produce secvenţa următoare?

```
int suma(int n)
{
      for(int i=0,s=0; i<n; i++)
            s+=i;
      return s;
}
```

IV. Instrucţiuni de salt

10. Instrucţiunea ***break***

Instrucţiunea ***break*** are sintaxa

***break*;**

şi este este folosită pentru a ieşi imediat dintr-o instrucţiune de ciclare sau dintr-un ***switch***. Ea transferă controlul instrucţiunii imediat următoare celei părăsite. Instrucţiunea ***break*** poate fi folosită numai în aceste două situaţii, dacă ea apare fără să fie în corpul unei ciclări sau al unui ***switch*** avem o eroare la compilare. In cazul instrucţiunilor imbricate (***for*** în ***for***, etc) **break** încheie numai execuţia instrucţiunii în care este direct inclusă.

Exemplu:

```
#include<iostream>
using namespace std;
int main(void){
	int i, j;
	for (i = 0; i<5; i++){
		cout << "i=" << i;
		for (j = 0; j<5; j++){
			if (i == j) {
				cout << " break ";
				break;
			}
			cout << " j=" << j;
		}
		cout << endl;
	}
	return 0;
}


/* REZULTAT

i=0 break
i=1 j=0 break
i=2 j=0 j=1 break
i=3 j=0 j=1 j=2 break
i=4 j=0 j=1 j=2 j=3 break
Press any key to continue . . .*/
```

11. Instrucţiunea ***continue***

Instrucţiunea ***continue*** are sintaxa

***continue*;**

şi este este folosită în instrucţiuni de ciclare (şi numai acolo) pentru a încheia în mod forţat execuţia iteraţiei curente şi a trece la următoarea iteraţie. In interiorul unei bucle ***while*** sau ***do-while***, la întâlnirea instrucţiunii ***continue*** are loc un salt direct la re-evaluarea expresiei test, iar în cazul instrucţiunii ***for*** se trece mai întâi la evaluarea expresiei de reluare şi apoi la evaluarea expresiei test.

Exemplu:

```
#include<iostream>
using namespace std;
int main(void){
	int i, j;
	for (i = 0; i<5; i++){
		cout << "i=" << i;
		for (j = 0; j<5; j++){
			if (i == j) {
				cout << " continue ";
				continue;
			}
			cout << " j=" << j;
		}
		cout << endl;
	}
	return 0;
}
/* REZULTAT
i=0 continue  j=1 j=2 j=3 j=4
i=1 j=0 continue  j=2 j=3 j=4
i=2 j=0 j=1 continue  j=3 j=4
i=3 j=0 j=1 j=2 continue  j=4
i=4 j=0 j=1 j=2 j=3 continue
Press any key to continue . . .*/
```

Iată o ilustrare a utilizării instrucţiunii ***continue*** în rezolvarea următoarei probleme: să se scrie o funcţie care afişează numai valorile comune a doi vectori, fiecare valoare fiind afişată o singură dată.

```
#include<iostream>
using namespace std;

//afiseaza valorile  comune o singura data
void afisare(int a[], int  b[], int dim){
    for (int i = 0; i<dim; i++){

        int val = a[i];
        //aflam daca val este valoare comuna
        //cautam un b[j]==val
        bool amGasit = false;
        for (int j = 0; j < dim && !amGasit; j++){
            if (b[j] == val) amGasit = true;
        }
        if (!amGasit) continue;     //->i++
        //val este valoare comuna
        //aflam daca a mai fost afisata
        //cautam un h<i pentru care a[h]==val
        amGasit = false;
        for (int h = 0; h<i && !amGasit; h++){
            if (a[h] == val) amGasit = true;
        }
        if (amGasit) continue;            //->i++
        //val este valoare comuna
        //nu a mai fost afisata
        //o afisam
        cout << val << ' ';
    }
    cout << endl;
}
```

```
int main(){
	const int dim = 10;
	int a[dim] = { 1, 7, 3, 4, 4, 3, 7, 3, 9, 0 };
	int b[dim] = { 0, 9, 3, 5, 1, 3, 7, 3, 0, 1 };
	afisare(a, b, dim);
	return 0;
}

/*REZULTAT:
1 7 3 9 0
Press any key to continue . . .*/
```

12. Instrucţiunea ***return***

Instrucţiunea ***return*** permite unei funcţii să transfere controlul înapoi funcţiei apelante, (sau, în cazul funcţiei ***main***, returnarea controlului către sistemul de operare).

Execuţia unei funcţii care trebuie să returneze o valoare se încheie atunci când controlul întâlneşte o instrucţiune ***return*** cu sintaxa:

return *expresie-rezultat*;

caz în care se evaluează expresia-rezultat iar rezultatul obţinut este transferat funcţiei apelante.

Funcţiile care nu trebuie să returneze nimic (cele a căror declaraţie începe cu ***void***) îşi încheie execuţia atunci când controlul întâlneşte o instrucţiune ***return*** fără expresie rezultat, sau la întâlnirea acoladei care închide corpul funcţiei.

Exemplu:

```
#include<iostream>
using namespace std;
int dist(int a, int b){
	if(a>b) return a-b;
	return b-a;
}

void panaLaPrimulZero( int tab[], int dim){
	int i=0;
	for(i=0;i<dim;i++){
		cout<<tab[i]<<" ";
		if(tab[i]==0) return;
	}
}
int main(void){
	const int dim=6;
	int v[]={1,2,3,0,4,5};
	int i;
	for(i=0;i<dim;i++){
		v[i]=dist(i,v[i]);
	}
	panaLaPrimulZero(v,6);
	cout<<endl;
	return 0;
}
/* REZULTAT
1 1 1 3 0
Press any key to continue
*/
```

13. Instrucţiunea *goto*

Instrucţiunea ***goto*** transferă controlul execuţiei dintr-un anumit punct al corpului unei funcţii la o instruciune etichetată aflată în interiorul aceleiaşi funcţii. Exemplu:

```
#include<iostream>
using namespace std;
int main(){
	int a;
	cout<<"a="; cin>>a;
	if(a>0) goto pozitiv;
	if(a<0) goto negativ;
	cout<<"a==0"<<endl;
GATA:
	cout<<"Gata, am terminat."<<endl;
	return 0;
pozitiv:
	cout<<"a>0"<<endl;
	goto GATA;
negativ:
	cout<<"a<0"<<endl;
	goto GATA;
	cout<<"Eu cand apar pe monitor?"<<endl;
}
```

Sintaxa instrucţiunii ***goto*** este următoarea:

goto *etichetă*;

O etichetă este formată dintr-un identificator urmat de simbolul două puncte. O instrucţiune este etichetată dacă exact în faţa sa se află una sau mai multe etichete. În corpul unei funcţii nu poate să apară de mai multe ori aceeaşi etichetă, dar aceasta poate să apară în funcţii diferite deoarece domeniul de vizibilitate al unei etichete este restrâns la corpul funcţiei în care apare. Aceste etichete sunt utilizate numai de instrucţiunea **goto**, în orice alt context o instrucţiune etichetată este executată fără să se ţină seama de prezenţa etichetei.

Deoarece urmărirea salturilor către etichete reduce mult claritatea programului, este recomandată utilizarea salturilor date de **break**, **continue** sau **return** în locul instrucţiunii **goto**.

Un exemplu de utilizare justificată: ieşirea forţată din bucle imbricate:

```
int main(){
	const int dim=10;
	int mat[dim][dim];
	int i,j;
	citeste(mat,dim);
	for(i=0;i<dim;i++){
		for(j=0;j<dim;j++){
			if(mat[i][j]==0) goto stop;
			cout<<1.0/mat[i][j]<<endl;
		}
	}
stop: cout<<"GATA"<<endl;
	return 0;
}
```

Dacă în matricea ***mat*** nu există elemente nule sunt afişate toate inversele elementelor şi apoi apare textul `"GATA"`, altfel afişarea se încheie la prima apariţie a lui zero. Se poate înlocui **goto** cu **return** dacă afişarea se efectuează într-o funcţie adecvată.