CAPITOLUL 7

7. Instrucțiuni de ciclare. Introducere

S-a discutat până acum despre structuri de control care permit alegerea condiționată între execuția mai multor seturi de instrucțiuni. Sistemele de calcul au avantajul că pot repeta, o dată "instruite ce şi cât" (programate), un număr de operațiuni de câte ori este nevoie. Orice limbaj de programare este are implementată această posibilitate. Există două clase de instrucțiuni de ciclare (repetare).

O primă categorie se referă la instrucțiunile care repetă un bloc de instrucțiuni până când este îndeplinită o condiție (prin condiție putem înțelege o expresie multiplă ce va fi evaluată de către compilator). De obicei este folosită pentru interacțiuni repetitive cu utilizatorul sau în calculul numeric prin aproximații şi de aceea instrucțiunile respective se numesc instrucțiuni cu număr necunoscut de paşi.

O a doua categorie de instrucțiuni repetitive este aceea a instrucțiunilor cu număr cunoscut de paşi. Din punct de vedere al limbajului nu ar fi strict nevoie de acest tip de instrucțiuni, pentru că se poate realiza acest lucru cu instrucțiunile din prima clasă. Totuşi, după cum se va observa mai jos implementarea (cu instrucțiuni cu număr necunoscut de paşi) pentru această a doua clasă presupune un cod care va fi identic la absolut toate interpretările. Pentru a uşura scrierea codului şi a-l face mai clar s-a implementat o instrucțiune nouă for**... .**

7.1. Instrucțiuni cu număr necunoscut de paşi

7.1.1. Instrucțiunea while

Este o instrucțiune care permite repetarea sau nu a execuției unei bucăți de cod funcție de valoarea de adevăr a unei expresii.

Un exemplu în care ar fi necesar acest tip de instrucțiuni ar fi restricția domeniului de valori în cazul datelor care le poate introduce un utilizator. Acest lucru este necesar pentru că de obicei utilizatorul nu ține cont de toate restricțiile relativ la intervalul în care poate da o valoare de intrare,sau poate da din greşeală o valoare incorectă. Totuşi modelul matematic specifică clar că algoritmul lucrează corect numai în intervalul [a,b].

Pentru utilizator faptul că programul dă un rezultat eronat datorită unei valori de intrare introdusă greşit înseamnă că programul nu funcționează corect. Este necesar ca programatorul să nu permită introducerea altor date decât cele specificate de modelul matematic.

Observație: limbajul C spre deosebire de limbajul Pascal nu generează un mesaj de eroare dacă tipul de dată introdus nu coincide cu specificatorul dat funcției *scanf*. Din acest motiv în momentul în care se pregăteşte o versiune beta (prima versiune a aplicației care va fi folosită de către utilizator) trebuiesc introduse toate restricțiile de tipul menționat mai sus. La nivel de pseudocod metoda poate fi descrisă ca mai jos:

citeşte valoarea de intrare
atâta timp cât valoarea de intrare nu se află în intervalul [a,b]
emite un mesaj de atenționare către utilizator

reciteşte valoarea

Pentru aceasta se poate folosi instrucțiunea while… a cărei sintaxă este prezentată mai jos:

```
while(conditie)// atâta timp cât condiția este adevărată execută
{
bloc de instrucțiuni
}
```

În figura 7.1. este prezentată schema logică corespunzătoare instrucțiunii while.

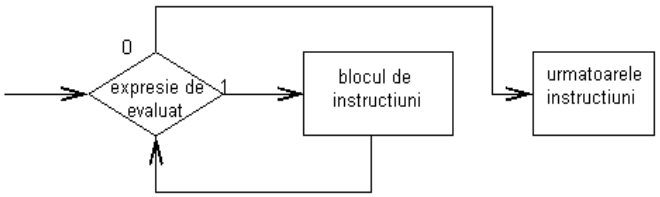

Fig. 7.1. Schema logică a instrucțiunii while

În exemplul de mai jos este vorba de citirea unei valori întregi care trebuie să aparțină intervalului [0,255].

```
#include <stdio.h>
#include <conio.h>

void main(void)
        {
         unsigned char value;
         clrscr();
        printf("Nr1:");
        scanf("%d",&op1);
        while((value>255)&&(value<0))
        {
        printf("Eroare ! Numarul trebuie sa fie intre 0 si 255 ");
         printf("Nr1:");
        scanf("%d",&op1);
        }
        // urmează restul programului
        }
```

7.1.2. Instrucțiunea do… while

După cum s-a menționat anterior instrucțiunea while… testează mai întâi valoarea de adevăr a expresiei de evaluat şi apoi, funcție de aceasta, execută sau nu blocul de instrucțiuni. De aceea i se mai spune instrucțiune de ciclare cu test anterior.

Dacă se analizează exemplul de mai sus din punct de vedere al simplității codului se observă că blocul de instrucțiuni pentru citirea variabilei a fost scris de două ori. Pentru a evita scriere de cod inutil în această situație pseudocodul trebuie să fie ca mai jos:

citeşte valoarea
verifică dacă se află în intervalul dorit
dacă da continuă programul
dacă nu reia de la citirea variabilei

Pentru a putea rezolva elegant situațiile de acest gen s-a mai introdus o instrucțiune şi anume do…while a cărei sintaxă este prezentată mai jos:

```
do //execută
 {
bloc de instrucțiuni
 }while(condiție)// atâta timp cât condiția este adevărată
```

În figura 7.2. avem prezentată schema logică pentru instrucțiunea do…while.

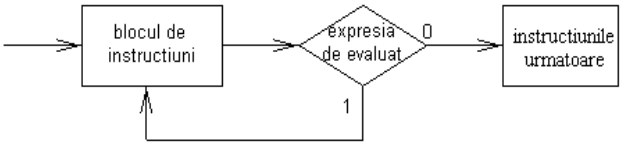

Fig. 7.2. Schema logică a instrucțiunii do…while

Folosind această instrucțiune codul prezentat mai sus devine:

```
#include <stdio.h>
#include <conio.h>

void main(void)
    {
        unsigned char value;
        clrscr();

    do
    {
        printf("Nr1:");
    scanf("%d",&op1);
    if( value>255)&&(value<0))
                printf("Eroare ! Numarul trebuie sa fie intre 0 si 255 ");
        } while((value>255)&&(value<0));

    // urmează restul programului
    }
```

Deci se citeşte numărul şi, dacă acesta nu este corect, se reia citirea. Pentru validarea afişării mesajului de eroare s-a prevăzut verificarea condiției pentru că dacă utilizatorul introduce corect valoarea de prima oară, nu este permisă afişarea mesajului de eroare.

Această instrucțiune execută măcar o dată blocul de instrucțiuni şi de-abia apoi face evaluarea expresiei pentru a putea vedea dacă acesta se mai repetă sau nu. De aceea i se mai spune şi instrucțiune de ciclare cu test posterior.

7.2. Instrucțiuni cu număr cunoscut de paşi. Instrucțiunea for.

Din cele prezentate până acum rezultă că se poate repeta un bloc de instrucțiuni până când este îndeplinită o condiție. Există cazuri când se cunoaşte de la început câte valori se doresc a fi prelucrate, fie prin specificarea directă a utilizatorului, fie ca rezultat a unor calcule anterioare. De obicei acest lucru se aplică la calculul vectorial sau matricial dar nu numai.

Întrucât nu s-a discutat încă despre acest tip de date se va lua în discuție un exemplu mai simplu, cum ar fi realizarea unui program ce va ridica un număr la o putere. Pseudocodul şi apoi programul sunt prezentate mai jos:

citeşte numărul (nr)
citeşte puterea la care va fi ridicat (pw)
inițializează variabila rezultat, ce va păstra nr la putere, cu 1 (rez=1)
atâta timp cât variabila ce a primit puterea >0 fă
*rez=rez*nr;*
pw=pw-1;
repetă
afişează rezultatul

Codul sursă corespunzător .

```
#include <stdio.h>
#include <conio.h>

void main(void)
        {
         int nr, pw;
         float rez;

         clrscr();
        printf("Numarul");
        scanf("%d",&nr);
        printf("\nPuterea");
        scanf("%d",&op2);
        rez=1; //****
        while(pw>0)//*****
        {
        rez=rez*nr;
```

```
pw=pw-1;//*****
}
printf("Numarul %d la puterea %d este %f",nr,pw,rez);
getch();
}
```

Instrucțiunile marcate cu asterix se repetă în foarte multe situații când numărul de paşi este cunoscut. De aceea s-a implementat instrucțiunea for… având următoarea sintaxă

```
for(contor=valoareinițială;contor<sau>valoarefinală;
                                        contor=contor+sau -pas)
{
bloc de instrucțiuni
}
```

La nivel de schema logică execuția instrucțiunii for este prezentată în figura 7.3.

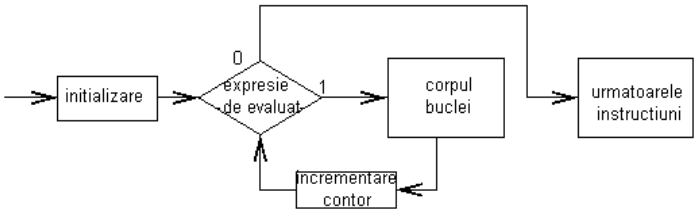

Fig. 7.3. Schema logică instrucțiunii ***for***

Acum se observă că această instrucțiune scuteşte programatorul de a scrie cod suplimentar. Programul anterior rescris folosind această instrucțiune este dat mai jos.

```
#include <stdio.h>
#include <conio.h>

void main(void)
        {
    int nr, pw,i;
    float rez;

    clrscr();
        printf("Numarul");
        scanf("%d",&nr);
        printf("\nPuterea");
        scanf("%d",&op2);
         for(i=0;i<pw;i=i+1)
                        rez=rez*nr;
        //codul mai poate fi scris şi ca in zona comentata ce urmează
        // for(i=pw;i>0;i=i-1)
        //                rez=rez*nr;
        printf("Numarul %d la puterea %d este %f",nr,pw,rez);
```

```
getch();
}
```

Există un control total asupra desfăşurării buclei exact ca şi în cazul while dar codul scris este mult mai succint.
Observația 1: Dacă blocul de instrucțiuni constă într-o singură instrucțiune parantezele nu mai sunt necesare
Observația 2: Dacă se pune ";" după for acesta va considera că are o buclă vidă, compilatorul nu va raporta eroare sintactică, dar ceea ce este scris în blocul de instrucțiuni va fi doar o singură dată executat.

7.3. Alte instrucțiuni

7.3.1. Instrucțiunea break

Efectul instrucțiunii break constă în părăsirea imediată a celei mai apropiate (textual) instrucțiuni switch, while, do, for înconjurătoare.

7.3.2. Instrucțiunea continue

Instrucțiunea continue nu se poate utiliza decât într-un ciclu while, do sau for. Efectul instrucțiunii continue constă :

- într-un salt la evaluarea condiției de ciclare, dacă cea mai apropiată (textual) instrucțiune de ciclare înconjurătoare este while sau do.
- într-un salt la evaluarea expresiei din a treia poziție , dacă cea mai apropiată (textual) instrucțiune de ciclare înconjurătoare este for.

Prezentăm mai jos un exemplu simplu de folosire a instrucțiunii break şi continue:

```
void main(void)
    {
    int xx;

      for(xx = 5;xx < 15;xx = xx + 1){
        if (xx == 8)
          break;
        printf("In bucla oprita fortat , xx are valoarea %d\n",xx);
      }

      for(xx = 5;xx < 15;xx = xx + 1){
        if (xx == 8)
          continue;
        printf("Prin aplicarea continue in bucla, xx devine %d\n",xx);
      }
    }
```

7.3.3. Instrucțiunea goto nume_etichetă

Efectul instrucțiunii goto constă într-un salt necondiționat la instrucțiunea cu eticheta citată în instrucțiunea de salt, care trebuie să se afle în aceeaşi funcție.

7.3.4. Instrucțiunea return

Efectul instrucțiunii return conduce la terminarea imediată a execuției funcției curente şi la revenirea la funcția apelantă. Dacă expresia este prezentă, ea trebuie să fie de un tip care poate fi convertit la tipul funcției în care apare instrucțiunea return. Valoarea expresiei este valoarea întoarsă de funcție.

7.3.5. Constante simbolice

Cu ajutorul directivei "#define" se pot defini la începutul programului nume sau constante simbolice care sunt un şir particular de caractere. După aceea, compilatorul va înlocui toate aparițiile, nepuse între ghilimele, ale numelui, prin şirul corespunzător. Înlocuirea efectivă a numelui poate fi orice text ea nu se limitează la numere.

```
#define MIN 0     //valoarea minimă admisă
#define MAX 300   //valoarea maximă admisă
# define AND &&

void main(void)
     {
      int x;
    printf("Dati un numar");
    scanf("%d",&x)
    if((x>=MIN)AND(x<=MAX))printf("\nNumarul este in interval");
        else printf("Numarul nu este in interval");
    getch();
    }
```

Cantitățile MIN şi MAX sunt constante, aşa încât ele nu apar în declarații. Numele simbolice se scriu în mod normal cu litere mari, aşa că pot fi uşor distinse de numele de variabile care se scriu cu litere mici. La sfârşitul unei definiții NU se pune punct şi virgulă.