LECȚIA 4. INSTRUCȚIUNI DE CICLARE (BUCLE)

4.1. Clasificarea buclelor

Cu elementele de programare învățate pînă acum (variabile, funcții de intrare / ieşire şi operatori) putem scrie programe de calcul în care instrucțiunile sunt executate secvențial o singură dată. La rezolvarea unor probleme concrete apare necesitatea ca o instrucțiune sau un grup de instrucțiuni să fie executate în mod repetat, adică în mod ciclic sau iterațiv, pînă cînd este îndeplinită o anumită condiție. Mecanismul care permite acest lucru se numeşte **ciclu** sau **buclă**.

În funcție de modul şi locul în care se efectuează testul de părăsire a unui ciclu există următoarele trei tipuri de cicluri:

- **ciclul cu contor** care este utilizabil ori de cîte ori se cunoaşte numărul de iterații ce trebuie efectuat. În acest caz, pentru a controla bucla, se foloseşte o variabilă de tipul întreg, numită **contor**, a cărei valoare este comparată în permanență cu numărul de iterații ce trebuie efectuate pentru a decide dacă este necesară repetarea secvenței de instrucțiuni ce defineşte corpul buclei sau părăsirea acesteia. Pentru funcționarea corectă a ciclului contorul trebuie inițializat în mod corespunzător, iar după fiecare iterație valoarea sa trebuie modificată.
- **ciclul cu test inițial** care este utilizabil atunci cînd nu se cunoaşte numărul de iterații ce trebuie efectuate, iar verificarea condiției de parcurgere a buclei este necesar a se realiza inaintea efectuării setului de instrucțiuni ce definesc corpul său. În cazul în care condiția de părăsire a buclei este satisfacută de la început, nu se va efectua nici o iterație.
- **ciclul cu test final** care este utilizabil, ca şi ciclul cu test inițial, atunci cînd nu se cunoaşte numărul de iterații ce trebuie efectuate. Deosebirea față de ciclul cu test inițial constă în faptul că verificarea condiției de continuare a procesului ciclic se realizează după efectuarea setului de instrucțiuni ce alcătuiesc corpul buclei. În acest caz, dacă condiția de părăsire a buclei este satisfacută de la început, se va executa totuşi o iterație.

4.2. Bucla for

Bucla **for** este un mecanism prin care se pot programa ciclurile cu test inițial cu sau fără contor. Pentru exemplificare, considerăm programul din exemplul 4.1, în care se calculează factorialul unui număr.

Exemplul 4.1.

```
/* Programul ex_4_1 */
#include <stdio.h>
#include <conio.h>
void main (void)
{
    int n, con;
    long fact=l;
    printf("\nIntroduceti un numar natural >");  scanf("%d",&n);
    for (con=i; con<=n; con++)
            fact *= con;        // similar cu fact = fact*con;
    printf("\n%d!=%ld",n,fact);
    getch ();
}
```

Programul solicită introducerea unui număr natural şi apoi calculează factorialul: *n!=1*2*3*...*(n-1)*n*, care indică faptul că trebuie efectuat un numar de n operații de înmulțire. Deoarece numărul de iterații necesar evaluării valorii *n!* este cunoscut, în cadrul programului s-a folosit un ciclu cu contor. În acest sens s-a folosit variabila de tipul întreg *con* ca variabilă de control a buclei şi variabila *fact* în care se va memora valoarea obținută. Deoarece valoarea factorialului depăşeşte în mod curent capacitatea de memorare a unei variabile de tipul întreg, variabila *fact* a fost declarată de tipul întreg **long**. În vederea efectuării calculului, variabila *fact* este inițializată cu *1* şi apoi instrucțiunea: *fact *= con;* este repetată de *n* ori pentru valori ale variabilei *con* egale cu *1, 2, ..., n*. Acest lucru este realizat de instrucțiunea:

```
for (con=1; con<=n; con++)
```

care mai întîi atribuie variabilei de control a buclei valoarea *1* şi apoi evaluează expresia logică *con<=n*. Dacă această expresie este adevărată, se execută instrucțiunea *fact*=con;* după care valoarea variabilei de control este incrementată şi se revine la evaluarea expresiei logice. Ciclul de operații este repetat pînă cînd valoarea expresiei logice devine falsă, adică *con* este mai mare decît *n*. În acest moment se părăseşte bucla şi execuția programului va continua cu apelul funcției **printf** care afişează rezultatul. Se constată că, în cazul în care n=0, expresia logică *con<=n* este falsă de la început, instrucțiunea *fact*=con* nu se va executa nici o dată, iar programul afişează rezultatul corect *n! = 1*.

Din cele prezentate concluzionăm că structura generală a buclei **for** cuprinde:

- o instrucțiune de forma:

```
for (expresie1; expresie2; expresie3)
```

care are rolul de a deschide bucla şi de a controla mecanismul de execuție a acesteia. Această instrucțiune începe cu cuvîntul cheie **for** şi nu se termină cu caracterul "**;**", aşa cum se termină în mod obişnuit instrucțiunile limbajului C.

- o instrucțiune sau un set de instrucțiuni care vor fi executate în mod repetat şi care formează **corpul buclei**.

Prima expresie din instrucțiunea de deschidere şi control a buclei **for** poartă numele de **expresie de inițializare** şi are rolul de a atribui o vaioare inițială variabilei de control a buclei. Variabila de control trebuie să fie de tipul întreg şi poate fi inițializată cu orice valoare întreagă. Inițializarea se poate realiza fie prin atribuirea unei valori, ca în exemplul 4.1 în care expresia de inițializare este *con=1*, fie prin utilizarea unei expresii aritmetice, ca de exemplu *con=i+2*k*. Cu alte cuvinte, practic orice expresie aritmetică poate fi utilizată ca expresie de inițializare. Deasemenea expresia de inițializare poate lipsi sau poate conține mai multe expresii, separate între ele prin virgula. Astfel, în programul ex_4_1 se poate elimina expresia *con=1* (nu şi caracterul ";" folosit ca terminator de instrucțiune) din instrucțiunea **for**. În acest caz ciclul fiind un ciclu cu contor este necesară inițializarea separată a contorului. Avînd în

vedere acest aspect, programul pentru calculul factorialului poate fi rescris ca în exemplul 4.1.a în care inițializarea contorului s-a făcut în instrucțiunea de declarare a acestuia.

Exemplul 4.1.a

```
/* progamul ex_4.1.a */
#include <stdio.h>
#include <conio.h>
void main(void)
 {
    int a, con=l;
    long fact=l;
    printf("\nIntroduceti un numar natural >"); scanf("%d",&n);
    for(; con<=n; con++)
            fact*=con;          // similar cu fact=fact*con;
    printf("\n%d!=%ld",n,fact);          getch ();
 }
```

Pentru a exemplifica modul de utilizare a mai multor expresii în cadrul expresiei de inițializare, considerăm programul din exemplul 4.2 în care se calculează suma primelor *n* numere naturale.

Exemplul 4.2.

```
/* Programul ex_4_2 */
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int n, i, sum;
    printf("\nIntroduceti un numar natural > "); scanf("%d",&n);
    for(i=l, sum=0; i<=n; i++)
            sum+=i;             /* similar cu sum=sum+i */
    printf("\Suma primelor %d numere naturale este %d",n,sum);getch ();
}
```

Programul este similar cu cel folosit la calculul factorialului. Pentru evaluarea sumei este necesară o variabilă de tipul întreg pe care am numit-o *sum*. Inițial, aceasta trebuie să aibă valoarea 0, urmînd ca la această valoare să însumăm valorile variabilei de control *i*. Noutatea programului constă în faptul că în cadrul expresiei de inițializare, pe lîngă atribuirea valorii inițiale 1 variabilei *i* de control a buclei, se realizează şi inițializarea variabilei *sum* cu zero.

A doua expresie din instrucțiunea de deschidere şi control a buclei **for** poartă numele de **expresie de test** şi are rolul de a detecta momentul în care se termină bucla. Acesta coincide cu momentul în care expresia de test devine falsă. Deoarece evaluarea acestei expresii se face la începutul fiecărui ciclu, este posibil ca instrucțiunile din corpul buclei să nu se execute niciodată. Astfel, dacă reluăm programul ex_4_2 şi tastăm 0, atunci expresia de control *i<=n* este falsă de la început şi programul va afişa răspunsul corect: suma primelor 0 numere naturale este 0.

Expresia de test poate lipsi din instrucțiunea de deschidere şi control a buclei **for**. În acest caz, prin convenție, se consideră că ea este adevărată tot timpul. Astfel, dacă rescriem programul ex_4_2 ca în exemplul ex_4_2_a în care s-au eliminat toate cele trei expresii ale instrucțiunii **for** dar nu şi caracterele ";", se obține o buclă infinită care va încerca să calculeze suma tuturor numerelor naturale. Desigur, acest lucru nu este posibil, iar modalitatea prin care se poate părăsi o buclă infinită constă în folosirea instrucțiunii de decizie **if** şi a instrucțiunii **break**, care vor fi prezentate in lecțiile următoare.

Exemplul 4.2.a

```
/* Programul ex_4_2_a */
#include <stdio.h>
#include <conio.h>
void main(void)
 {
    int i=l, sum=0;
    clrscr( );
    for( ; ; )
            sum+=i++;
 }
```

A treia expresie folosită în cadrul instrucțiunii de deschidere şi control a buclei **for** se numeşte **expresie de modificare** şi are rolul de a modifica variabila de control a buclei. În exemplele prezentate, modificarea s-a realizat cu ajutorul operatorului de incrementare ++ în cadrul instrucțiunilor *con++* şi respectiv *i++*. Deşi în mod uzual în cadrul expresiei de modificare se foloseşte incrementarea, se precizează faptul că se poate utiliza şi decrementarea sau, mai general, orice expresie aritmetică. De exemplu, în programul pentru calculul factorialului, în locul instrucțiunii *for (con=1;con<=n;con++)* se poate folosi instrucțiunea *for (con=n; con>=1;con --);* .

Expresia de modificare, ca şi expresiile de inițializare şi control, pot lipsi din instrucțiunea **for**. În acest caz, modificarea valorii contorului se realizeaza printr-o instrucțiune plasată în interiorul buclei. Această tehnică este ilustrată în exemplul 4.2.a în care inițializarea variabilei *i* s-a făcut în afara buclei, iar modificarea ei s-a efectuat în corpul buclei prin operația de postincrementare din instrucțiunea *sum+=i++;* echivalentă cu următoru1 set de instrucțiuni: *sum=sum+i; i=i+1;*.

În cazul în care corpul buclei este format din mai multe instrucțiuni, pentru delimitarea acestuia se folosesc acoladele. Pentru exemplificare, considerăm programul din exemplul 4.3 în care se calculează simultan factorialul şi suma primelor *n* numere naturale.

Exemplul 4.3.

```
/* Programul ex_4_3 */
```

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
    int i, n, sum;
    long fact;
    printf ("\nIntroduceti un numar natural >"); scanf ("%d",&n);
    for (i=l, fact=l, sum=0; i<=n; i++)
    {       /* inceput corp bucla */
            fact *= i;          sum += i;
    }       /* sfirsit corp bucla */
    printf("\nSuma primelor %d numere naturale este %d\nFactorialul lui %d este %1d",n,sum,n,fact);       getch( );
}
```

În acest program corpul buclei constă din 2 instrucțiuni grupate cu ajutorul unei perechi de acolade.

În limbajul C, un ansamblu de două sau mai multe instrucțiuni grupate într-o pereche de acolade se numeşte **instrucțiune compusă** sau **bloc de instrucțiuni**.

Deoarece compilatorul C tratează ca fiind o instrucțiune ansamblul format din cuvîntul cheie **for**, cele trei expresii cuprinse între parantezele rotunde ce-l urmează şi corpul buclei, este posibilă înlanțuirea buclelor **for**, adică utilizarea în corpul unei bucle **for** a altei bucle **for**. Pentru exemplificare, considerăm programul din exemplul 4.4 care generează tabla înmulțirii.

Exemplul 4.4.

```
/* Programul ex_4_4 */
#include <stdio.h>
#include <conio.h>
void main (void)
 {
    int i, j;   clrscr( );
    for (i=l; i<=9; i++)
     {
            for (j=l; j<=9; j++)        printf("%4d",i*j);
            printf ("\n");
     }      getch( );
 }
```

În cadrul programului se utilizează două bucle: **bucla exterioară** avînd ca variabilă de control pe *i*, care conține în corpul ei o altă buclă, numită **buclă interioară**, ce are ca variabila de control pe *j*.

Pentru fiecare valoare a lui *i* de 1 la 9 se execută bucla interioară care, prin apelul repetat al funcției **printf** afişează produsul lui *i* cu 1,2,3,...,9, adică tabla înmulțirii cu *i*. După terminarea buclei interioare, care conține în corpul ei o singură instrucțiune, se pregăteşte afişarea unui nou rînd prin instrucțiunea *printf("\n");* care face parte din corpul primei bucle.

Atunci cînd avem mai multe bucle **for** înlănțuite, instrucțiunile ce alcătuiesc corpul buclelor sunt scrise mai la dreapta față de acoladele ce le grupează, iar acestea la rîndul lor se scriu în aceeaşi coloană, dar pe linii diferite. Această tehnică, este numită **indentare multiplă**. Atunci cînd corpul unei bucle este format dintr-o singură instrucțiune, utilizarea acoladelor pentru delimitarea lui este opțională.

Exemplul 4.5.

```
/* Programul ex_4_5 */
#include <stdio.h>
#include <conio.h>
void main (void)
 {  int i, j;   clrscr( );
    for (i=l; i<=12; i++)
     {
            for (j=1; j<=40; j++)       printf ("\xB0");
            printf ("\n");
     }      getch ();
 }
```

Acest program este similar cu cel din exemplul anterior cu deosebirea că în loc de afişarea produsului *i*j* se afişează un caracter grafic, efectul final fiind trasarea unui dreptunghi pe ecran. Pentru afişarea caracterului grafic, care va genera dreptunghiul, s-a folosit secvența escape *\xB0* în cadrul funcției **printf**. Prin schimbarea secvenței escape, dreptunghiul poate fi umplut cu diversele caractere grafice existente în codul ASCII extins.

Aşa după cum s-a precizat instrucțiunea **for** poate fi utilizată şi pentru programarea ciclurilor cu un număr necunoscut de iterații. Pentru a exemplifica acest mod de utilizare a buclei **for** considerăm programul din exemplul 4.6.

Exemplul 4.6.

```
/* Programul ex_4_6 */
#include <stdio.h>
#include <conio.h>
void main (void)
 {  int ncar = 0;
    clrscr( );          printf("Tastati un sir de cazactere >");
    for( ; getche( )!=13 ; )    ncar++;
    printf("\nNumarul caracterelor tastate este %d", ncar);       getch ();
```

}

Programul îi solicită utilizatorului să tasteze un şir de caractere. Caracterele tastate, în număr necunoscut, sunt citite cu ajutorul funcției **getche**, în cadrul buclei **for**, pînă cînd s-a apăsat tasta <Enter>, şi sunt contorizate cu ajutorul variabilei de tipul întreg *ncar*. Pentru a sesiza momentul în care trebuie părăsită bucla, în cadrul expresiei de test valoarea returnată de funcția **getche** este comparată cu codul tastei <Enter>, care este 13. După cum se poate constata, instrucțiunea de deschidere şi control a buclei nu conține expresiile de inițializare şi modificare, operațiile de inițializare şi modificare a contorului numărului de caractere tastate fiind efectuate separat. Programul poate fi rescris ca în exemplul 4.6.a, astfel încît inițializarea şi modificarea variabilei *ncar* să se realizeze în cadrul expresiilor buclei **for**. Se precizează faptul că în acest caz variabila *ncar* nu este contorul ciclului, deoarece valoarea sa nu este folosită pentru a sesiza momentul părăsirii buclei. Remarcăm că în corpul buclei nu se execută nici o instrucțiune (corpul buclei este vid), in locul unei instrucțiuni sau al unui set de instrucțiuni se foloseşte caracterul ";".Dacă ar fi lipsit caracterul ";", atunci instrucțiunea de apel a funcției **printf** ar fi fost considerată ca făcînd parte din corpul buclei, ce nu-i corect.

Exemplul 4.6.a.

```
#include <stdio.h>
#include (conio.h>
void main (void)
 {
    int ncar;            clrscr( );
    printf ("Tastati un sir de caractere >");
    for(ncar=0; getche( )!=13; ncar++)
            ;
    printf("\nNumarul caracterelor tastate este %d", ncar);          getch( );
 }
```

4.3. Bucla while

A doua modalitate de programare a unei bucle, oferită de limbajul C, o constituie bucla **while**, care este similară ca structură şi mecanism de funcționare cu bucla **for**. Pentru exemplificare să rescriem programul de calcul al factorialului,utilizînd bucla **while**.

Exemplul 4.7.

```
/* Programul ex_4_7 */
#include <stdio.h>
#include <conio.h>
void main (void)
{
    int con=l, n;
    long fact=l;
    printf("\n Introduceti un numar natural > "); scanf("%d",&n);
    while(con<=n)
    {
            fact *= con; con++;
    }
    printf ("\n %d factorial este %ld",n,fact);    getch( );
}
```

Comparînd acest program cu programul ex_4_1 constatăm cădeosebirile constau în inițializarea variabilei de control *con* în momentul declarării şi înlocuirea buclei **for** cu bucla **while**.

Structura generală a buclei **while** se compune din:

- o instrucțiune de forma while (<expresie logică>) care are rolul de a deschide bucla şi de a controla mecanismul de execuție. Această instrucțiune începe cu cuvîntul cheie **while** urmat de o expresie logică cuprinsă între paranteze rotunde şi nu se termină cu caracterul ";".
- corpul buclei - format din una sau mai multe instrucțiuni.

Pentru a fi posibilă funcționarea buclei, este necesară inițializarea variabilei de control, fie în momentul declarării, fie printr-o instrucțiune separată, şi modificarea acesteia în cadrul corpului buclei. Ca şi în cazul buclei **for**, evaluarea expresiei logice de control (testul) se face la început, şi, deci, este posibil ca instrucțiunile din corpul buclei să nu se execute. De asemenea, dacă nu se modifică variabila de control astfel încît expresia logică să devină falsă, bucla nu se va mai termina niciodată, adică bucla este infinită. Pentru exemplificare, considerăm următorul program din exemplul 4.8 în care se calculează, pentru diverse valori ale lui *n* introduse de la tastatură, valoarea sumei: $S_ñ = 1^2 + 2^2 + 3^2 + ... + n^2$.

Exemplul 4.8

```
/* Program ex_4_8 */
#include <stdio.h>
#include <conio.h>
void main (void)
 {  int i, n, sum;
    while (i)
     {
            printf ("\n Introduceti un numar natural > ");          scanf("%d",&n);
            for(i=l, sum=0; i<=n; i++)sum+=i*i;
            printf("\nSuma patratelor primelor %d numere naturale este %d",n,sum);
     }
 }
```

Să remarcăm că în corpul buclei **while** se află o bucla **for**, lucru permis de compilatorul C. Este deci posibilă utilizarea în corpul unei bucle a altor bucle de acelaşi tip sau de tipuri diferite, adică o înlanţuire generală a acestora. Reamintim deasemnea faptul, că în limbajul C orice valoare întreagă diferită de zero reprezintă valoarea logică *adevărat*, iar zero valoarea logică *fals*. În exemplul 4.8 expresia logică a buclei **while** este 1, adică tot timpul adevărată. Deci bucla este infinită, şi prin urmare, după tastarea unui număr, şi calculul sumei şi afişarea rezultatului, se revine de fiecare dată la solicitarea introducerii unui nou număr. Acest mecanism de ciclare infinită este frecvent utilizat în cazul în care se doreşte selectarea unei variante dintr-un meniu şi revenirea în acesta după execuţia programelor asociate cu varianta selectată. Pentru a termina totuşi execuţia programului, se foloseşte facilitatea oferită de sistemul de operare MS-DOS care opreşte execuţia oricărui program ce execută operaţii de scriere sau citire, la apăsarea tastelor <Ctrl>+ <C>.

În exemplele prezentate, bucla **while** a fost utilizată pentru programarea unui ciclu cu contor (exemplul 4.7), respectiv a unei bucle infinite (exemplul 4.8).Ca şi bucla **for**,bucla **while** poate fi utilizată în programarea ciclurilor cu test iniţial şi cu un număr necunoscut de iteraţii. Pentru exemplificare, în exemplul 4.9 este rescris programul ex _4_ 6, folosind bucla **while** în locul buclei **for**.

Exemplul 4.9.

```
/* Programul ex_4_9 */
#include <stdio.h>
#include <conio.h>
main ( )
{
    int nrcar=0;
    clrscr ( );          printf ("\nTastaţi o propozitie: \n");
    while (getche( ) != '\r')    nrcar++;
    printf ("\nPropozitia contine %3d caractere",nrcar);  getch ( );
}
```

Caracterele ce alcătuiesc propoziţia, în număr necunoscut, sunt citite în corpul buclei **while** cu ajutorul funcţiei **getche** şi sunt contorizate de variabila *nrcar*. Bucla se termină în momentul în care se apasă tasta <Enter>, acţiune care marchează sfîrşitul propoziţiei. Execuţia programului se continuă cu afişarea numărului de caractere tastate. Se remarcă faptul că bucla **while** nu conţine o variabilă de control, iar expresia ei logică conţine apelul unei funcţii. Acest lucru este posibil deoarece funcţia **getche** este interpretată de compilatorul C ca o variabilă avînd valoarea codului asociat caracterului tastat. Această valoare este comparată cu codul tastei <Enter>, simbolizat în acest program prin secvenţa escape "\r", pentru a evalua expresia logică a buclei în care s-a folosit operatorul relational !=.

Deoarece funcţia **getche** este interpretată ca o variabilă avînd o valoare, aceasta poate fi utilizată fie în cadrul unor instrucţiuni de atribuire, fie în cadrul unor expresii relaţionale sau logice. Să considerăm pentru exemplificare programul din exemplul 4.10. Acesta are memorat codul ASCII al unei cifre şi solicită utilizatorului să descopere care este aceasta (prin tastarea unei cifre de la 0 la 9), în final afişîndu-se numărul de încercări efectuate.

Exemplul 4.10.

```
/* Programul ex_ 4_ 10 */
#include <stdio.h>
#include <conio.h>
void main (void)
 {
    int incerc=l;
    char cifra;
    clrscr( );printf("Introduceti o cifra > ");
    while( (cif ra=getche( )) != '5')
     {
            printf ("\nGreseala; cifra %c este incorecta. Incercati inca o data ",cifra);        incerc++;
     }
    printf ("\ncifra este %c. Ati reusit dupa %sd incercari",cifra,incerc);    getch ( );
 }
```

În acest caz, expresia logică a buclei **while** este mult mai complexă şi cuprinde o expresie de atribuire şi o expresie relaţională. Pentru a înţelege modul cum este evaluată această expresie, să presupunem că s-a tastat la un moment dat caracterul 3. La început, caracterul (codul acestuia) returnat de functia **getche** este atribuit variabilei *cifra*. După aceasta, întreaga expresie de atribuire *cifra=getche( )* capătă valoarea variabilei *cifra*, care este caracterul 3. În final această valoare este comparată cu caracterul 5 aflat în dreapta operatorului relaţional, pentru a evalua expresia logică, care în acest caz este adevărată (3 este diferit de 5) şi, deci, se va executa corpul buclei. Este important de notat faptul că punerea expresiei de atribuire între paranteze este strict necesară deoarece operatorii relaţionali au o precedentă mai mare decît operatorul de atribuire "=". Dacă nu am fi utilizat parantezele, adică am fi folosit expresia *cifra=getche( ) != "5"*, atunci mai întîi s-ar fi comparat caracterul returnat de funcţia **getche** cu caracterul "5", iar rezultatul (0 sau 1) ar fi fost atribuit variabilei *cifra*.

4.4. Bucla do – while

A treia şi ultima modalitate de programare a unei bucle, oferită de limbajul C, o constituie bucla **do while**. Ca structură, aceasta este similară cu buclele **for** şi **while**, deosebirea fundamentală constînd în mecanismul său de funcţionare.

Pentru a discuta structura şi modul de funcţionare al buclei **do while**, considerăm programul din exemplul 4.10 în care se calculează şi se afişează primii zece termeni ai şirului cu termenul general $a_ñ = 2n/(2n+1)$.

Exemplul 4.11.

```
/* Programul ex_4_11 */
#include <stdio.h>
```

```
#include <conio.h>
void main(void)
 {
    float an; int i=l;
    clrscr ();
    do {
            an = 2*i/(2*i+1.);
            printf("\na%2d=%7.4f",i,an);        i++;
     }
    while (i <=10);
    getch( );
 }
```

În cadrul programului se utilizează variabila reala *an*, în care se memorează valoarea termenului curent al şirului, şi variabila întreagă *i*, iniţializată cu 1, ca variabilă de control a buclei.

Iniţial, este executat setul de instrucţiuni cuprins între cuvintele cheie **do** şi **while** (corpul buclei) pentru calculul şi afişarea valorii termenului curent al şirului şi pentru modificarea variabilei de control. În final, este evaluată expresia relaţională i <=10. Dacă aceasta este adevărată se reia ciclul de instrucţiuni, în caz contrar, bucla se termină.

Din cele prezentate se constată că structura buclei **do while**, destinată programării ciclurilor cu test final, cuprinde:

- cuvîntul cheie **do** care marchează începutul buclei;
- un set de instrucţiuni cuprinse între acolade care formează corpul buclei;
- o instrucţiune de forma *while(expresie_logică);* care constituie testul buclei. Se remarcă faptul că această instrucţiune se termină cu caracterul ";".

Deosebirea fundamentală faţă de buclele **for** şi **while** constă în faptul că testul se efectuează la sfîrşit. În acest caz, chiar dacă expresia logică a buclei este falsă de la început, corpul buclei se execută o dată.

Ca şi în cazul buclelor **for** şi **while**,corpul buclei **do while** poate conţine alte bucle. Pentru exemplificare, considerăm programul:

Exemplul 4.12.

```
/* Programul ex_4_12 */
#include <stdio.h>
#include <conio.h>
void main(void)
 {
    int: incerc;          char litera;
    do {
            clrscr( );            incerc=l;
            printf("\nTastati o litera mica > ");
            while ((litera=getche( )) != 'b')
             {
            printf ("\nGreseala! Litera %c este incorecta. Incercati inca o data ", litera);      incerc++;
             }
            printf ("\nLitera este b. Ati reusit dupa %3d incercari ",incerc);          printf ("\nReluati jocul 7 ");
     }
    while (getch( ) == 'd');
    printf("\nMultumesc pentru joc");
 }
```

Acest program, similar cu cel din exemplul 4.9, are memorată litera *b* şi solicită partenerului de joc să o descopere prin încercări succesive. În momentul în care s-a reuşit acest lucru, programul părăseşte bucla internă **while**, afişează numărul de încercări şi întreabă dacă se doreşte reluarea jocului. Dacă se răspunde prin apăsarea tastei <d>, se reia corpul buclei externe **do while**, adică se începe un nou joc. În caz contrar, se părăseşte bucla şi programul afişează mesajul: "Mulţmesc pentru joc".

Fiind cunoscute cele 3 modalităţi de programare a unei bucle oferite de limbajul C (for, while şi do while), referitor la alegerea uneia dintre acestea în literatura de specialitate se fac următoarele recomandări:

- cînd numărul de iteraţii este cunoscut prin valoarea unei variabile sau a unei constante, se va folosi bucla **for** în forma ei completă, adică cea care conţine expresia de iniţializare a contorului, expresia de test şi expresia de modificare a contorului;
- cînd numărul de iteraţii este necunoscut, iar testul de părăsire a buclei trebuie realizat înaintea efectuării setului de instrucţiuni ce alcătuiesc corpul buclei, se va folosi bucla **while**. În schimb, dacă testul de părăsire a buclei trebuie realizat după parcurgerea corpului acesteia, se va folosi bucla **do while.**

4.5. Instrucţiunile break şi continue

Menţionăm faptul că limbajul C are 2 instrucţiuni, numite **break** şi respectiv **continue**, care pot fi folosite în cadrul oricăreia dintre cele trei tipuri de bucle discutate anterior.

Instrucţiunea **break** determină, în momentul în care este executată, părăsirea buclei chiar dacă expresia de test nu a devenit falsă.

Instrucţiunea **continue** determină, în momentul în care este executată, revenirea la începutul buclei, sărindu-se peste instrucţiunile care o urmează.