3 IMPLEMENTAREA STRUCTURILOR DE CONTROL

Algoritmul proiectat pentru rezolvarea unei anumite probleme trebuie implementat într-un limbaj de programare; prelucrarea datelor se realizează cu ajutorul ***instrucţiunilor***. Instrucţiunea descrie un proces de prelucrare pe care un calculator îl poate executa. O instrucţiune este o construcţie validă (care respectă sintaxa limbajului) urmată de **;** . Ordinea în care se execută instrucţiunile unui program defineşte aşa-numita ***structură de control*** a programului.

Limbajele moderne sunt alcătuite pe principiile *programării structurate*. Conform lui C. Bohm şi G. Jacobini, orice algoritm poate fi realizat prin combinarea a trei structuri fundamentale:

- structura secvenţială;
- structura alternativă (de decizie, de selecţie);
- structura repetitivă (ciclică).

3.1. IMPLEMENTAREA STRUCTURII SECVENŢIALE

Structura secvenţială este o înşiruire de secvenţe de prelucrare (instrucţiuni), plasate una după alta, în ordinea în care se doreşte execuţia acestora.

Reprezentarea structurii secvenţiale cu ajutorul ***schemei logice*** (figura 3.1.):

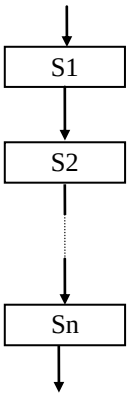


Figura 3.1. Schema logică pentru structura secvenţială

Reprezentarea structurii secvenţiale cu ajutorul ***pseudocodului***:

```
instr1;
instr2;
. . . . .
```

Implementarea structurii secvenţiale se realizează cu ajutorul instrucţiunilor:

- Instrucţiunea *vidă*

  Sintaxa: **;**

Instrucţiunea vidă nu are nici un efect. Se utilizează în construcţii în care se cere prezenţa unei instrucţiuni, dar nu se execută nimic (de obicei, în instrucţiunile repetitive).

Exemple:

```
int a;
. . . . . .
int j;
  ;
```

```
for (;;)
{
. . . .
}
```

- Instrucţiunea *expresie*

  Sintaxa: **`expresie;`**

  sau: **`apel_funcţie;`**

Exemple:

```
int b, a=9;
double c;
b=a+9;
cout<<a;
c=sqrt(a);
clrcsr();
```
//apelul funcţiei predefinite care şterge ecranul; prototipul în headerul `conio.h`

- Instrucţiunea compusă (instrucţiunea **bloc**)

  Sintaxa:
```
{
      declaratii;
      instr1;
      instr2;
      . . . .
}
```

Într-un bloc se pot declara şi variabile care pot fi accesate doar în corpul blocului. Instrucţiunea bloc este utilizată în locurile în care este necesară prezenţa unei singure instrucţiuni, însă procesul de calcul este mai complex, deci trebuie descris în mai multe secvenţe.

3.2. IMPLEMENTAREA STRUCTURII DE DECIZIE (ALTERNATIVE, DE SELECŢIE)

Reprezentarea prin schemă logică şi prin pseudocod a structurilor de decizie şi repetitive sunt descrise în capitolul 1. Se vor prezenta în continure doar instrucţiunile care le implementează.

- Instrucţiunea **`if`**:

Sintaxa:
```
if (expresie)
      instrucţiune1;
[ else
      instrucţiune2; ]
```

Ramura `else` este opţională.

La întâlnirea instrucţiunii `if`, se evaluează *`expresie`* (care reprezintă o condiţie) din paranteze. Dacă valoarea expresiei este `1` (condiţia este îndeplinită) se execută *`instrucţiune1`*; dacă valoarea expresiei este `0` (condiţia nu este îndeplinită), se execută *`instrucţiune2`*. Deci, la un moment dat, se execută doar una dintre cele două instrucţiuni: *fie* instrucţiune1, *fie* instrucţiune2. După execuţia instrucţiunii `if` se trece la execuţia instrucţiunii care urmează acesteia.

Observaţii:

1. `Instrucţiune1` şi `instrucţiune2` pot fi instrucţiuni compuse (blocuri), sau chiar alte instrucţiuni `if` (if-uri imbricate).
2. Deoarece instrucţiunea if testează valoarea numerică a expresiei (condiţiei), este posibilă prescurtarea: `if (expresie)`, în loc de `if (expresie != 0)`.
3. Deoarece ramura `else` a instrucţiunii if este opţională, în cazul în care aceasta este omisă din secvenţele if-else imbricate, se produce o ambiguitate. De obicei, ramura `else` se asociază ultimei instrucţiuni `if`.

   Exemplu:
```
if (n>0)
      if (a>b)
```

```
            z=a;
      else z=b;
```

4. Pentru claritatea programelor sursă se recomandă alinierea instrucţiunilor prin utilizarea tabulatorului orizontal.
5. Deseori, apare construcţia:

```
if (expresie1)
      instrucţiune1;
else
      if (expresie2)
            instrucţiune2;
      else
            if (expresie3)
                  instrucţiune3;
                  . . . . . .
                  else
                        instrucţiune_n;
```

Aceeaşi construcţie poate fi scrisă şi astfel:

```
if (expresie1)
      instrucţiune1;
else if (expresie2)
      instrucţiune2;
else if (expresie3)
      instrucţiune3;
. . . . . .
else
      instrucţiune_n;
```

Expresiile sunt evaluate în ordine; dacă una dintre expresii are valoarea 1, se execută instrucţiunea corespunzătoare şi se termină întreaga înlănţuire. Ultima parte a lui else furnizează cazul când nici una dintre expresiile 1,2,. . ., n-1 nu are valoarea 1.

6. În cazul în care instrucţiunile din cadrul `if-else` sunt simple, se poate folosi operatorul condiţional.

Exerciţii:

1. Să se citească de la tastatură un număr real. Daca acesta se află în intervalul [-1000, 1000], să se afiseze 1, dacă nu, să se afiseze -1.

```
#include <iostream.h>
void main()
  {
double nr; cout<<"Astept numar:"; cin>>nr;
int afis = (nr>= -1000 && nr <= 1000 ? 1 : -1); cout<<afis;
/* int afis;
   if (nr >= -1000 && nr <= 10000)
      afis = 1;
   else afis= -1;
cout<<afis; */
}
```

2. Să se calculeze valoarea funcţiei f(x), ştiind că x este un număr real introdus de la tastatură:

$$f(x) = \begin{cases} -6x + 20 & , \text{ dacă } x \in [-\infty, -7] \\ x + 30 & , \text{ dacă } x \in (-7, 0] \\ \sqrt{x} & , \text{ dacă } x>0 \end{cases}$$

```
#include <iostream.h>
#include <math.h>
void main()
{
double x,f;cout<<"x=";cin>>x;
if (x <= -7)
      f= -x* 6 +20;
else
      if  ( x<=0 )
            f= x+30;
      else f=sqrt(x);
cout<<"f="<<f<<'\n';
}
```

Sau:

```
#include <iostream.h>
#include <math.h>
void main()
{
double x,f;cout<<"x=";cin>>x;
if (x <= 7)
      f= -x* 6 +20;
if  (x>=-7 && x<=0 )
      f= x+30;
if (x>0) f=sqrt(x);
      cout<<"f="<<f<<'\n';
}
```

Uneori, construcţia if-else este utilizată pentru a compara valoarea unei variabile cu diferite valori constante, ca în programul următor:

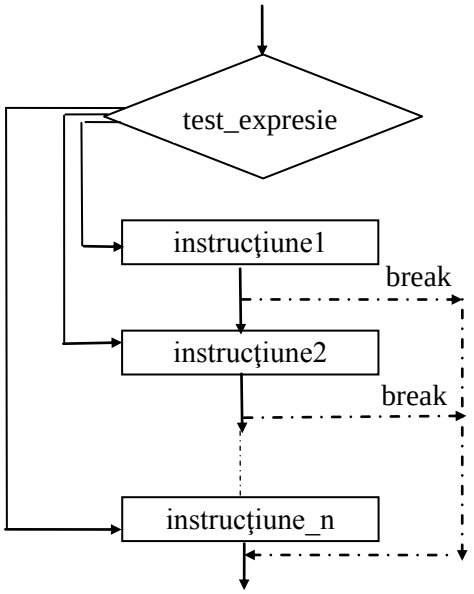
3. Se citeşte un caracter reprezentând un operator aritmetic binar simplu. În funcţie de caracterul citit, se afişează numele operaţiei pe care acesta o poate realiza.

```
#include <iostream.h>
void main()
{
char oper;
cout<<"Introdu operator aritmetic, simplu, binar:"; cin>>oper;
if (oper == '+')
      cout<<"Operatorul de adunare!\n";
else if (oper=='-' )
      cout<<"Operatorul de scadere!\n";
else if (oper=='*' )
      cout<<"Operatorul de inmultire!\n";
else if (oper=='/' )
      cout<<"Operatorul de impartire!\n";
else if (oper=='%' )
      cout<<"Operatorul rest!\n";
else  cout<<"Operator ilegal!!!\n";
}
```

- Instrucţiunea **`switch`**

În unele cazuri este necesară o decizie multiplă specială. Instrucţiunea ***`switch`*** permite acest lucru.

Reprezentare prin schema logică (figura 3.2):



Reprezentare prin pseudocod:

```
Dacă expresie=expr_const_1
      instrucţiune1;
      [ieşire;]
Altfel dacă expresie=expr_const_2
      instrucţiune2;
      [ieşire;]
      .
      .
Altfel dacă expresie=expr_const_n-1
      instrucţiune_n-1;
      [ieşire;]
Altfel instrucţiune_n;
```

Figura 3.2. Decizia multiplă

Se testează dacă valoarea pentru `expresie` este una dintre constantele specificate (`expr_const_1`, `expr_const_2`, etc.) şi se execută instrucţiunea de pe ramura corespunzătoare. În schema logică `test_expresie` este una din condiţiile: `expresie=expr_const_1`, `expresie=expr_const_2`, etc.
Sintaxa:

```
switch (expresie)
{
case expresie_const_1:      instructiune_1;
                      [break;]
case expresie_const_2:      instructiune_2;
                      [break;]
. . . . . .
case expresie_const_n-1:    instructiune_n-1;
                      [break;]
[ default: instructiune_n; ]
}
```

Este evaluată `expresie` (expresie aritmetică), iar valoarea ei este comparată cu valoarea expresiilor constante 1, 2, etc. (expresii constante=expresii care nu conţin variabile). În situaţia în care valoarea `expresie` este egală cu valoarea `expr_const_k`, se execută instrucţiunea corespunzătoare acelei ramuri (`instrucţiune_k`). Dacă se întâlneşte instrucţiunea ***break***, parcurgerea este întreruptă, deci se va trece la execuţia primei instrucţiuni de după switch. Dacă nu este întâlnită instrucţiunea *break,* parcurgerea continuă. Break-ul cauzează deci, ieşirea imediată din switch.
În cazul în care valoarea expresiei nu este găsită printre valorile expresiilor constante, se execută cazul marcat cu eticheta ***default*** (când acesta există). Expresiile `expresie, expresie_const_1, expresie_const_2,` etc., trebuie să fie întregi. În exemplul următor, ele sunt de tip `char`, dar o dată de tip `char` este convertită automat în tipul `int`.

Exerciţiu: Să rescriem programul pentru problema 3, utilizând instrucţiunea switch.

```
#include <iostream.h>
void main()
{
      char oper;
      cout<<"Introdu operator aritmetic, simplu, binar:";
      cin>>oper;
      switch (oper)
      {
            case ('+'):
                  cout<<"Operatorul de adunare!\n";
                  break;
            case ('-'):
                  cout<<"Operatorul de scadere!\n";
                  break;
            case ('*'):
                  cout<<" Operatorul de inmultire!\n";
                  break;
            case ('/'):
                  cout<<"Operatorul de impartire!\n";
                  break;
            case ('%'):
                  cout<<"Operatorul rest!\n";
                  break;
            default:
                  cout<<"Operator ilegal!\n";
      }
}
```

3.3. IMPLEMENTAREA STRUCTURILOR REPETITIVE (CICLICE)

Există două categorii de instrucţiuni ciclice: cu test iniţial şi cu test final.

3.3.1. Implementarea structurilor ciclice cu test iniţial

Structura ciclică cu test iniţial este implementată prin instrucţiunile ***while*** şi ***for***.

- Instrucţiunea **while**

Sintaxa:

```
while (expresie)
      instructiune;
```

La întâlnirea acestei instrucţiuni, se evaluează `expresie`. Dacă aceasta are valoarea `1` - sau diferită de 0 - (condiţie îndeplinită), se execută `instrucţiune`. Se revine apoi în punctul în care se evaluează din nou valoarea expresiei. Dacă ea este tot `1`, se repetă `instrucţiune`, ş.a.m.d. Astfel, instrucţiunea (corpul ciclului) *se repetă atât timp* cât `expresie` are valoarea `1`. În momentul în care `expresie` ia valoarea `0` (condiţie neîndeplinită), se iese din ciclu şi se trece la următoarea instrucţiune de după *while*.

Observaţii:

1. În cazul în care la prima evaluare a expresiei, aceasta are valoarea zero, corpul instrucţiunii while nu va fi executat niciodată.
2. `Instrucţiune` din corpul ciclului while poate fi compusă (un bloc), sau o altă instrucţiune ciclică.
3. Este de dorit ca instrucţiunea din corpul ciclului `while` să modifice valoarea expresiei. Dacă nu se realizează acest lucru, corpul instrucţiunii while se repetă de un număr infinit de ori.

Exemplu:

```
int a=7;
while (a==7)
      cout<<"Buna ziua!\n";     // ciclu infinit; se repetă la infinit afişarea mesajului
```

- Instrucţiunea **`for`**

În majoritatea limbajelor de programare de nivel înalt, instrucţiunea `for` implementează structura ciclică cu număr cunoscut de paşi (vezi reprezentarea prin schema logică şi pseudocod din capitolul 1). În limbajul C instrucţiunea `for` poate fi utilizată într-un mod mult mai flexibil.

Reprezentare prin schema logică (figura 3.3.):

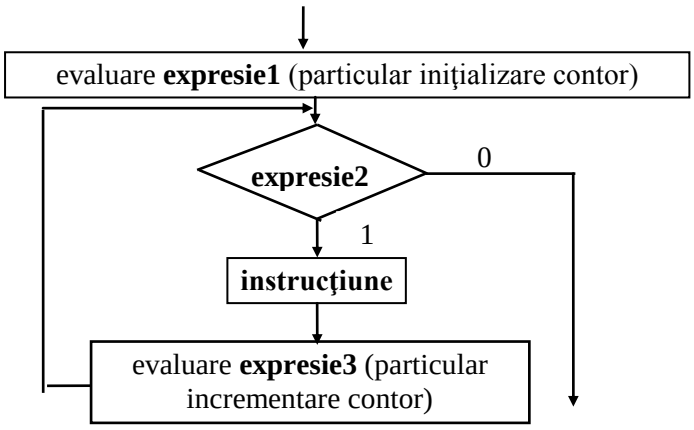


Figura 3.3. Structura ciclică cu test iniţial

Reprezentare în pseudocod:

```
evaluare expresie1
CÂT TIMP expresie2
REPETĂ
      ÎNCEPUT
      instrucţiune
      evaluare expresie3
      SFÂRŞIT
```

Sintaxa:

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

Nu este obligatorie prezenţa expresiilor, ci doar a instrucţiunilor vide.

Exemplu:

```
for ( ; expresie2; )
      instructiune;
```

sau:

```
for ( ; ; )
      instructiune;
```

3.3.2. Implementarea structurilor ciclice cu test final

- Instrucţiunea **`do-while`**

Sintaxa:

```
do   instructiune;
while(expresie);
```

Se execută `instrucţiune`. Se evaluează apoi `expresie`. Dacă aceasta are valoarea `1`, se execută `instrucţiune`. Se testează din nou valoarea expresiei. Se repetă `instrucţiune` *cât timp* valoarea expresiei este 1 (condiţia este îndeplinită). În cazul instrucţiunii do-while, corpul ciclului se execută cel puţin o dată.

Exerciţii:

1. Se citeşte câte un caracter, până la întâlnirea caracterului @. Pentru fiecare caracter citit, să se afişeze un mesaj care să indice dacă s-a citit o literă mare, o literă mică, o cifră sau un alt caracter. Să se afişeze câte litere mari au fost introduse, câte litere mici, câte cifre şi câte alte caractere. Se prezintă trei modalităţi de implementare (cu instrucţiunea `while`, cu instrucţiunea `for` şi cu instrucţiunea `do-while`).

```
#include <iostream.h>
#include <conio.h>
void main()
{ char c; clrscr();
int lmic=0, lmare=0, lcif=0;
int altcar=0;
cout<<"Aştept car.:"; cin>>c;
 while (c!='@'){
      if (c>='A' && c<='Z') {
            cout<<"Lit. mare!\n";
            lmare++; }
      else if (c>='a' && c<='z') {
            cout<<"Lit. mică!\n";
            lmica++; }
      else if (c>='0' && c<='9') {
            cout<<"Cifră!\n";
            lcif++; }
      else {
            cout<<"Alt car.!\n";
            altcar++; }
      cout<<"Aştept car.:";cin>>c;
   }
cout<<"Aţi introdus \n";
cout<<lmare<<" litere mari, ";
cout<<lmic<<" litere mici\n";
cout<<lcif<<" cifre şi \n";
cout<<altcar<<" alte carctere\n";
getch();    }
```

Observaţii legate de implementare
Variabila c (tip char) memorează caracterul introdus la un moment dat, de la tastatură.
Variabilele întregi lmic, lmare, lcif şi altcar sunt utilizate pe post de contor pentru litere mari, mici, cifre, respectiv alte caractere.
Acţiunea care se repetă cât timp caracterul citit este diferit de constanta caracter '@' constă din mai multe acţiuni simple: citirea unui caracter (cu afişarea în prealabil a mesajului "Aştept car.:"; testarea caracterului citit (operatorii relaţionali pot fi aplicaţi datelor de tip char).
Ca urmare, acţiunea din corpul instructiunii while a fost implementată printr-o instrucţiune bloc.
Tot instrucţiuni bloc au fost utilizate pe fiecare ramură a instrucţiunii if (afişare mesaj referitor la caracter şi incrementare contor).

```
#include <iostream.h>
#include <conio.h>
void main()
{ char c;clrscr();
intlmic=0,lmare=0,lcif=0;int altcar=0;
cout<<"Aştept caract.:"; cin>>c;
 for( ; c!='@'; ){
      // corp identic
   }
cout<<"Aţi introdus \n";
cout<<lmare<<" litere mari, ";
cout<<lmic<<" litere mici\n";
cout<<lcif<<" cifre şi \n";
cout<<altcar<<" alte carctere\n";
getch();    }
```

Pentru implementarea aceluiaşi algoritm se poate utiliza instrucţiunea for. În cadrul acesteia, expresie1 şi expresie3 lipsesc, însă prezenţa instrucţiunilor vide este obligatorie.

O altă variantă de implementare poate fi următoarea, în care şi iniţializarea variabilelor contor se realizează în cadrul expresiei `expresie1`.

```
int lmic, lmare, lcif, altcar;
for(lmare=0, lmic=0, lcif=0, altcar=0; c!='@'; ){
      // corp identic
}
```

Variantă de implementare care utilizează instrucţiunea `do-while`:

```
int lmic=0, lmare=0, lcif=0;
int altcar=0;
cout<<"Aştept caract.:";cin>>c;
do {
      //corp do-while
   } while (c!='@');
cout<<"Aţi introdus \n";
//...
```

2. Să se calculeze suma şi produsul primelor n numere naturale, n fiind introdus de la tastatură. Se vor exemplifica modalităţile de implementare cu ajutorul instrucţiunilor `do-while`, `while`, şi `for`. (Se observă că: $S = \sum_{k=1}^{n} k$, $P = \prod_{k=1}^{n} k$).

```
cout<<"n="; int n; cin>>n;
int S=0, P=1, k=1;
while (k <= n){
      S+=k; P*=k;
      k++;
}
cout<<"P="<<P<<"\tS="<<S<<'\n';
```

```
cout<<"n="; int n; cin>>n;
int S=0, P=1, k=1;
do{
      S+=k; P*=k;
      k++;
} while (k <= n);
cout<<"P="<<P<<"\tS="<<S<<'\n';
```

Pentru a ilustra multiplele posibilităţi oferite de instrucţiunea `for`, prezentăm variantele

```
// varianta1
int S=0, P=1, k;
for (k=1; k<=n; k++){
      S+=k; P*=k;
}
cout<<"P="<<P<<"\tS=";
cout<<S<<'\n';
```

```
// varianta2
int S=0, P=1;
for (int k=1; k<=n; k++){
      S+=k; P*=k;
}
cout<<"P="<<P<<"\tS=";
cout<<S<<'\n';
```

```
// varianta3
for (int S=0, P=1, k=1; k<=n; k++){
      S+=k; P*=k;
}
cout<<"P="<<P<<"\tS="<<cout<<S<<'\n';
```

```
// varianta4
for (int S=0, P=1, k=1; k<=n; S+=k, P*=k, k++)
            ;
cout<<"P="<<P<<"\tS="<<cout<<S<<'\n';
```

3. Să se citească un şir de numere reale, până la întâlnirea numărului 900. Să se afişeze maximul numerelor citite.

```
#include <iostream.h>
void main()
{double n;
cout<<"Introdu nr:"; cin>>n;
double max=n;
while (n!=900)
{     if (n>=max)
            max=n;
      cout<<"Introdu nr:";
      cin>>n;
}
cout<<"Max şir este:"<<max<<'\n';
}
```

Se presupune că primul element din şirul de numere are valoarea maximă. Se memorează valoarea sa în variabila max. Se parcurge apoi şirul, comparându-se valoarea fiecărui element cu valoarea variabilei max. În cazul în care se găseşte un element cu o valoare mai mare decât a variabilei max, se reţine noua valoare (max=n).

4. Să se afişeze literele mari ale alfabetului şi codurile aferente acestora în ordine crescătoare, iar literele mici şi codurile aferente în ordine descrescătoare. Afişarea se va face cu pauză după fiecare ecran.

```
#include <iostream.h>
#include <conio.h>
#define DIM_PAG 22          //dimensiunea paginii (numarul de randuri de pe o pagina)
void main()
{clrscr();
cout<<"LITERELE MARI:\n";int nr_lin=0; // nr_lin este contorul de linii de pe un ecran
for (char LitMare='A'; LitMare<='Z'; LitMare++){
   if (nr_lin==DIM_PAG){
         cout<<"Apasa o tasta...."; getch(); clrscr(); nr_lin=0;}
   cout<<"Litera "<<LitMare<<" cu codul ASCII "<<(int)LitMare<<'\n';
   // conversia explicita (int)LitMare permite afisarea codului ASCII al caracterului
   nr_lin++;
   }
cout<<"LITERELE MICI:\n";
for (char LitMica='z'; LitMica>='a'; LitMica--){
   if (nr_lin==DIM_PAG){
         cout<<"Apasa o tasta...."; getch(); clrscr(); nr_lin=0;}
   cout<<"Litera "<<LitMica<<" cu codul ASCII "<<(int)LitMica<<'\n';
   nr_lin++;
}
}
```

5. Să se scrie un program care realizează conversia numărului N întreg, din baza 10 într-o altă bază de numeraţie, b<10 (N şi b citite de la tastatură). Conversia unui număr întreg din baza 10 în baza b se realizează prin împărţiri succesive la b şi memorarea resturilor, în ordine inversă. De exemplu:
547:8=68 *rest* **3**; 68:8=8 *rest* **4**; 8:8=1 *rest* **0**; 1:8=0 *rest* **1** $547_{10} = 1043_8$

```
#include <iostream.h>
void main()
{ int nrcif=0,N,b,rest,Nv,p=1;
long Nnou=0;
cout<<"\nIntroduceti baza<10, b=";cin>>b;
cout<<"Introduceti numarul in baza 10, nr=";cin>>N;
Nv=N;
while(N!=0)
{
   rest=N%b;   N/=b;   cout<<"nr="<<N<<'\n'; cout<<"rest="<<rest<<'\n';
   nrcif++;  Nnou+=rest*p;   p*=10;   cout<<"Nr. nou="<<Nnou<<'\n';
}
 cout<<"Numarul de cifre este "<<nrcif<<'\n'; cout<<"Nr. in baza 10 "<<Nv;
 cout<<" convertit in baza "<<b<<" este "<<Nnou<<'\n';   }
```

6. Să se calculeze seria următoare cu o eroare mai mică decât EPS (EPS introdus de la tastatură): $1+\sum_{k=1}^{\infty}\frac{x^k}{k}$, $x \in [0,1]$, x citit de la tastatură. Vom aduna la sumă încă un termen cât timp diferenţa dintre suma calculată la pasul curent şi cea calculată la pasul anterior este mai mare sau egală cu EPS.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main()
{ double T,S,S1;long k;k=1;T=1;S=T;double x; cout<<"x="; cin>>x;
// T= termenul general de la pasul curent; S=suma la pasul curent; S1=suma la pasul anterior
  do  {
   S1=S;k=k+1;T=pow(x,k)/k;          //funcţia pow(x, k), aflată în <math.h> calculează x^k
   S=S+T;  // cout<<k<<" "<<T<<" "<<S<<'\n';getch();
  }  while ((S-S1)>=EPS);
 cout<<"Nr termeni="<<k<<"  T="<<T<<"   S="<<S<<'\n';    }
```

3.4. FACILITĂŢI DE ÎNTRERUPERE A UNEI SECVENŢE

Pentru o mai mare flexibilitate (tratarea excepţiilor care pot apare în procesul de prelucrare), în limbajul C se utilizează instrucţiunile **`break`** şi **`continue`**. Ambele instrucţiuni sunt utilizate în instrucţiunile ciclice. În plus, instrucţiunea break poate fi folosită în instrucţiunea `switch`.

- Instrucţiunea **`break`**

Aşa cum se observă din figura 3.4., utilizată în cadrul instrucţiunilor ciclice, instrucţiunea ***break*** "forţează" ieşirea din acestea. Fără a se mai testa valoarea expresiei (condiţia) care determină repetarea corpului instrucţiunii ciclice, se continuă execuţia cu instructiunea care urmează instructiunii ciclice. Astfel, se întrerupe repetarea corpului instrucţiunii ciclice, indiferent de valoarea condiţiei de test.
Utilizarea în cadrul instrucţiunii switch: În situaţia în care s-a ajuns la o valoare a unei expresiei constante egală cu cea a expresiei aritmetice, se execută instrucţiunea corespunzătoare acelei ramuri. Dacă se întâlneşte instrucţiunea `break`, parcurgerea este întreruptă (nu se mai compară valoarea expresiei aritmetice cu următoarele constante), deci se va trece la execuţia primei instrucţiuni de după switch. Dacă nu este întâlnit *break,* parcurgerea continuă. Instrucţiunea breakl cauzează deci, ieşirea imediată din `switch`.

- Instrucţiunea **`continue`**

Întâlnirea instrucţiunii `continue` (figura 3.4.) determină ignorarea instrucţiunilor care o urmează în corpul instrucţiunii ciclice şi reluarea execuţiei cu testarea valorii expresiei care determină repetarea sau nu a corpului ciclului.

Exemplu: Să revenim la programul realizat pentru problema 1, care foloseşte instrucţiunea `dowhile.` Dacă primul caracter citit este chiar caracterul @, se realizează testarea acestuia; ca urmare, se afişează mesajul `"Alt car.!"` şi se incrementează valoarea contorului `altcar`. Dacă nu se doreşte ca acest caracter să fie testat şi numărat, în corpul instrucţiunii do while putem face un test suplimentar.

```
int lmic=0,lmare=0,lcif=0,altcar=0;cout<<"Aştept caract.:";cin>>c;
do {
      if (c == '@')      break;        //ieşire din do while
      //corp do-while
   } while (c!='@');
cout<<"Aţi introdus \n";
//. . .
```

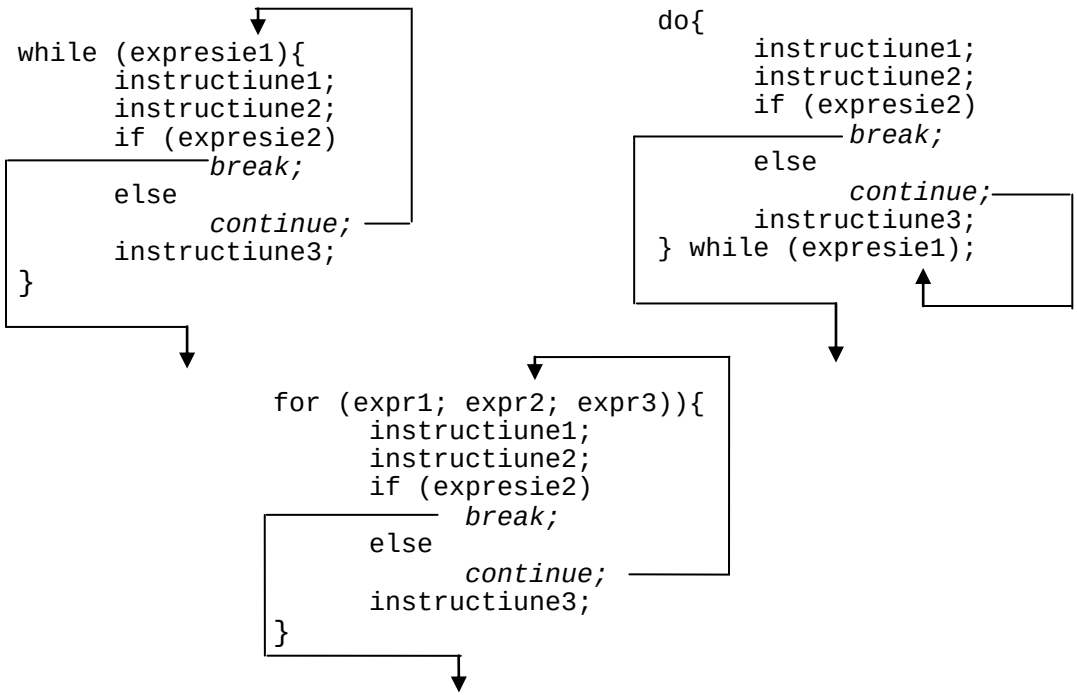


Figura 3.4. Modul de utilizare a instrucţiunilor `break` şi `continue`

Chestiuni teoretice

1. Care sunt instrucţiunile care implementează în limbajul C structura condiţională?
2. Care sunt instrucţiunile care implementează în limbajul C structura secvenţială?
3. Care sunt instrucţiunile care implementează în limbajul C structura repetitivă cu test iniţial?
4. Care sunt instrucţiunile care implementează în limbajul C structura repetitivă cu test final?
5. Ce deosebiri sunt între instrucţiunea while şi instrucţiunea do-while?
6. Pornind de la sintaxa instrucţiunii for, stabiliţi echivalenţa între aceasta şi instrucţiunile while şi do-while.

Chestiuni practice

1. Să se implementeze programele cu exemplele prezentate.
2. Să se scrie programele pentru exerciţiile rezolvate care au fost prezentate.
3. Să se implementeze algoritmii proiectaţi pentru problemele 1-7 din capitolul 1.
4. Să se calculeze aria unui triunghi, cunoscându-se mărimea laturilor sale. Numerele care reprezintă mărimile laturilor vor fi introduse de utilizator. Se va testa mai întâi dacă cele 3 numere reprezentând mărimea laturilor pot forma un triunghi (a <= b+c, b <= c+d, c <= a+b).
5. Să se rescrie următoarea secvenţă, folosind o singură instrucţiune if.

```
if  (n<0)
        if (n>=90)
              if (x!=0)
                    int b= n/x;
```

6. Să se citească un numar natural n. Să se scrie un program care afişează dacă numărul n citit reprezintă sau nu, un an bisect (anii bisecţi sunt multipli de 4, exceptând multiplii de 100, dar incluzând multiplii de 400).
7. Să se găsească toate numerele de două cifre care satisfac relaţia:

$$\overline{xy}=(x+y)^2$$

8. Să se citească un şir de numere reale, până la întâlnirea numarului 800 şi să se afişeze valoarea minimă introdusă, suma şi produsul elementelor şirului.
9. Scrieţi un program care să verifice inegalitatea 1/(n+1) < ln[(n+1)/n] < 1/n, unde n este un număr natural pozitiv, introdus de la tastatură.
10. Fie funcţia

$$f(x)=\begin{cases} e^{x-3} & , x\in[0, 1) \\ \sin x+\cos x & , x\in[1, 2) \\ 0{,}9\ln(x+3) & , x\in[2, 100] \end{cases}$$

Să se calculeze f(x), x citit de la tastatură.

11. Să se scrie un program care calculează şi afişează maximul a 3 numere reale (a, b şi c) citite de la tastatură.
12. Să se scrie un program care calculează şi afişează minimul a 3 numere reale (a, b şi c) citite de la tastatură.
13. Să se citească 2 caractere care reprezintă 2 litere mari. Să se afişeze caracterele citite în ordine alfabetică.
14. Să se citească 3 caractere care reprezintă 3 litere mici. Să se afişeze caracterele citite în ordine alfabetică.
15. Să se scrie un program care citeşte o cifră. În funcţie de valoarea ei, să se facă următorul calcul: dacă cifra este 3, 5 sau 7 să se afişeze pătratul valorii numerice a cifrei; dacă cifra este 2, 4 sau 6 să se afişeze cubul valorii numerice a cifrei; dacă cifra este 0 sau 1 să se afişeze mesajul "Valori mici"; altfel., să se afişeze mesajul "Caz ignorat!".
16. Fie şirul lui Fibonacci, definit astfel:

$$f(0)=0,\ f(1)=1,\ f(n)=f(n-1)+f(n-2) \text{ în cazul în care } n>1.$$

Să se scrie un program care implementează algoritmul de calcul al şirului Fibonacci.

17. Să se calculeze valoarea polinomului Cebîşev de ordin n într-un punct x dat, cunoscând relaţia:

$$T_0(x)=1,\ T_1(x)=x \text{ şi } T_{k+1}(x) - 2xT_k(x) + T_{k-1}(x) = 0$$

18. Să se citească câte 2 numere întregi, până la întâlnirea perechii (0, 0). Pentru fiecare pereche de numere, să se calculeze şi să se afişeze cel mai mare divizor comun.
19. Se citesc câte 3 numere reale, până la întâlnirea numerelor 9, 9, 9. Pentru fiecare triplet de numere citit, să se afişeze maximul.
20. Se citeşte câte un caracter până la întâlnirea caracterului @. Să se afişeze numărul literelor mari, numarul literelor mici şi numărul cifrelor citite; care este cea mai mare (lexicografic) literă mare, literă mică şi cifră introdusă.
21. Se citesc câte 2 numere întregi, până la întâlnirea perechii de numere 9, 9. Pentru fiecare pereche de numere citite, să se afişeze cel mai mare divizor comun al acestora.
22. Să se calculeze suma seriei

    $$1 + x^3/3 - x^5/5 + x^7/7 - \ldots$$

    cu o eroare mai mică decât epsilon (epsilon citit de la tastatură). Să se afişeze şi numărul de termeni ai sumei.
23. Să se citească un număr întreg format din 4 cifre (abcd). Să se calculeze şi să se afişeze valoarea expresiei reale: 4*a + b/20 -c + 1/d.
24. Să se scrie un program care afişează literele mari ale alfabetului în ordine crescătoare, iar literele mici - în ordine descrescătoare.
25. Să se scrie un program care generează toate numerele perfecte până la o limită dată, LIM. Un număr perfect este egal cu suma divizorilor lui, inclusiv 1 (exemplu: 6=1+2+3).
26. Să se calculeze valoarea sumei urmatoare, cu o eroare EPS mai mică de 0.0001:
    S=1+(x+1)/ 2! + (x+2)/ 3! + (x+3)/ 3! + ... , unde 0<=x<=1, x citit de la tastatură.
27. Să se genereze toate numerele naturale de 3 cifre pentru care cifra sutelor este egală cu suma cifrelor zecilor şi unităţilor.
28. Să se citească câte un număr întreg, până la întâlnirea numărului 90. Pentru fiecare numar să se afişeze un mesaj care indică dacă numărul este pozitiv sau negativ. Să se afişeze cel mai mic număr din şir.
29. Să se genereze toate numerele naturale de 3 cifre pentru care cifra zecilor este egală cu diferenţa cifrelor sutelor şi unităţilor.
30. Să se calculeze suma:

    (1 + 2!) / (2 + 3!) - (2+3!) / (3+4!) + (3+4!) / (4+5!) - .....