

IMPLEMENTAREA STRUCTURILOR DE CONTROL

3.1. Implementarea structurii secvențiale

3.2. Implementarea structurii de decizie

3.3. Implementarea structurilor repetitive

3.4. Facilități de întrerupere a unei secvențe

Algoritmul proiectat pentru rezolvarea unei anumite probleme trebuie implementat într-un limbaj de programare; prelucrarea datelor se realizează cu ajutorul **instrucțiunilor**. Instrucțiunea descrie un proces de prelucrare pe care un calculator îl poate executa. O instrucțiune este o construcție validă (care respectă sintaxa limbajului) urmată de ; . Ordinea în care se execută instrucțiunile unui program definește așa-numita **structură de control** a programului.

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

- structura secvențială;
- structura alternativă (de decizie, de selecție);
- structura repetitivă (ciclică).

3.1. IMPLEMENTAREA STRUCTURII SECVENȚIALE

Structura secvențială este o înșiruire de secvențe de prelucrare (instrucțiuni), plasate una după alta, în ordinea în care se dorește execuția acestora.

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

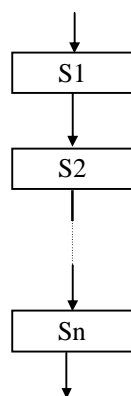


Figura 3.1. Schema logică pentru structura secvențială

Reprezentarea structurii secvențiale cu ajutorul *pseudocodului*:

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

Implementarea structurii secvențiale se realizează cu ajutorul instrucțiunilor:

- Instrucțiunea *vidă*

Sintaxa: ;

Instrucțiunea vidă nu are nici un efect. Se utilizează în construcții în care se cere prezența unei instrucțiuni, dar nu se execută nimic (de obicei, în instrucțiunile repetitive).

Exemple:

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

```

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

```

- Instrucțiunea *expresie*

Sintaxa: **expresie;**
sau: **apel_funcție;**

Exemple:

```

int b, a=9;
double c;
b=a+9;
cout<<a;
c=sqrt(a);
clrscr();//apelul funcției predefinite care șterge ecranul; prototipul în headerul conio.h

```

- Instrucțiunea compusă (instrucțiunea **bloc**)

Sintaxa: {
 declaratii;
 instr1;
 instr2;

 }

Într-un bloc se pot declara și variabile care pot fi accesate doar în corpul blocului. Instrucțiunea bloc este utilizată în locurile în care este necesară prezența unei singure instrucțiuni, însă procesul de calcul este mai complex, deci trebuie descris în mai multe secvențe.

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

Reprezentarea prin schemă logică și prin pseudocod a structurilor de decizie și repetitive sunt descrise în capitolul 1. Se vor prezenta în continuare doar instrucțiunile care le implementează.

- Instrucțiunea **if**:

Sintaxa:

```

if (expresie)
    instrucțiune1;
[ else
    instrucțiune2; ]

```

Ramura `else` este opțională.

La întâlnirea instrucțiunii `if`, se evaluează *expresie* (care reprezintă o condiție) din paranteze. Dacă valoarea expresiei este 1 (condiția este îndeplinită) se execută *instrucțiune1*; dacă valoarea expresiei este 0 (condiția nu este îndeplinită), se execută *instrucțiune2*. Deci, la un moment dat, se execută doar una dintre cele două instrucțiuni: *fie* *instrucțiune1*, *fie* *instrucțiune2*. După execuția instrucțiunii `if` se trece la execuția instrucțiunii care urmează acesteia.

Observatii:

1. *instrucțiune1* și *instrucțiune2* pot fi instrucțiuni compuse (blocuri), sau chiar alte instrucțiuni `if` (if-uri imbricate).
2. Deoarece instrucțiunea `if` testează valoarea numerică a expresiei (condiției), este posibilă prescurtarea: `if (expresie)`, în loc de `if (expresie != 0)`.
3. Deoarece ramura `else` a instrucțiunii `if` este opțională, în cazul în care aceasta este omisă din secvențele `if-else` imbricate, se produce o ambiguitate. De obicei, ramura `else` se asociază ultimei instrucțiuni `if`.

Exemplu:

```

if (n>0)
    if (a>b)
        z=a;
    else z=b;

```

4. Pentru claritatea programelor sursă se recomandă alinierea instrucțiunilor prin utilizarea tabulatorului orizontal.

5. Deseori, apare construcția:

```

if (expresie1)
    instrucțiune1;
else
    if (expresie2)
        instrucțiune2;
    else
        if (expresie3)
            instrucțiune3;
        . . . . .
        else
            instrucțiune_n;

```

Aceeași construcție poate fi scrisă și astfel:

```

if (expresie1)
    instrucțiune1;
else if (expresie2)
    instrucțiune2;
else if (expresie3)
    instrucțiune3;
. . . . .
else
    instrucțiune_n;

```

Expresiile sunt evaluate în ordine; dacă una dintre expresii are valoarea 1, se execută instrucțiunea corespunzătoare și se termină întreaga înlănțuire. 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 instrucțiunile din cadrul if-else sunt simple, se poate folosi operatorul condițional.

Exercitii:

1. Să se citească de la tastatură un număr real. Dacă acesta se află în intervalul [-1000, 1000], să se afișeze 1, dacă nu, să se afișeze -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 funcției f(x), știind că x este un număr real introdus de la tastatură:

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

```

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

3. Se citește un caracter reprezentând un operator aritmetic binar simplu. În funcție de caracterul citit, se afișează numele operației pe care acesta o poate realiza.

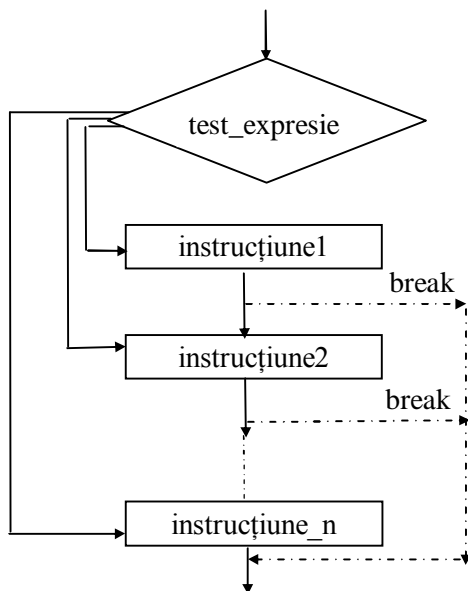
```
#include <iostream.h>
void main()
{
    char oper;
    cout<<"Introdu operator aritmetic, simplu, binar:"; cin>>oper;
    if (oper == '+')
        cout<<"Operatorul de adunare!\n";
    else if (oper=='-')
        cout<<"Operatorul de scadere!\n";
    else if (oper=='*')
        cout<<"Operatorul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

□ Instrucțiunea **switch**

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

Reprezentare prin schema logică (figura 3.2):

Reprezentare prin pseudocod:



```
Dacă expresie=expr_const_1
    instrucțiune1;
    [ieșire;]
Altfel dacă expresie=expr_const_2
    instrucțiune2;
    [ieșire;]
    .
    .
    .
Altfel dacă expresie=expr_const_n-1
    instrucțiune_n-1;
    [ieșire;]
Altfel instrucțiune_n;
```

Figura 3.2. Decizia multiplă

Se testează dacă valoarea pentru *expresie* este una dintre constantele specificate (*expr_const_1*, *expr_const_2*, etc.) și se execută instrucțiunea de pe ramura corespunzătoare. În schema logică *test_expresie* este una din condițiile: *expresie=expr_const_1*, *expresie=expr_const_2*, etc.

Sintaxa:

```
switch (expresie)
{
    case expresie_const_1:    instrucțiune_1;
                            [break;]
    case expresie_const_2:    instrucțiune_2;
                            [break;]
    .
    .
    .
    case expresie_const_n-1:  instrucțiune_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 conțin variabile). În situația în care valoarea expresie este egală cu valoarea `expr_const_k`, se execută instrucțiunea corespunzătoare acelei ramuri (`instructiune_k`). Dacă se întâlnește instrucțiunea **break**, parcurgerea este întreruptă, deci se va trece la execuția primei instrucțiuni de după `switch`. Dacă nu este întâlnită instrucțiunea `break`, parcurgerea continuă. Break-ul cauzează deci, ieșirea 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`.

Exercițiu: Să rescriem programul pentru problema 3, utilizând instrucțiunea `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 instrucțiuni ciclice: cu test inițial și cu test final.

3.3.1. Implementarea structurilor ciclice cu test inițial

Structura ciclică cu test inițial este implementată prin instrucțiunile **while** și **for**.

□ Instrucțiunea **while**

Sintaxa:

```

while (expresie)
    instructiune;

```

La întâlnirea acestei instrucțiuni, se evaluează `expresie`. Dacă aceasta are valoarea 1 - sau diferită de 0 - (condiție îndeplinită), se execută `instructiune`. Se revine apoi în punctul în care se evaluează din nou

valoarea expresiei. Dacă ea este tot 1, se repetă instrucțiune, ș.a.m.d. Astfel, instrucțiunea (corpul ciclului) *se repetă atât timp cât expresie* are valoarea 1. În momentul în care expresie ia valoarea 0 (condiție neîndeplinită), se iese din ciclu și se trece la următoarea instrucțiune de după *while*.

Observatii:

1. În cazul în care la prima evaluare a expresiei, aceasta are valoarea zero, corpul instrucțiunii *while* nu va fi executat niciodată.
2. Instrucțiune din corpul ciclului *while* poate fi compusă (un bloc), sau o altă instrucțiune ciclică.
3. Este de dorit ca instrucțiunea din corpul ciclului *while* să modifice valoarea expresiei. Dacă nu se realizează acest lucru, corpul instrucțiunii *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 afișarea mesajului
```

□ Instrucțiunea **for**

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

Reprezentare prin schema logică (figura 3.3.):

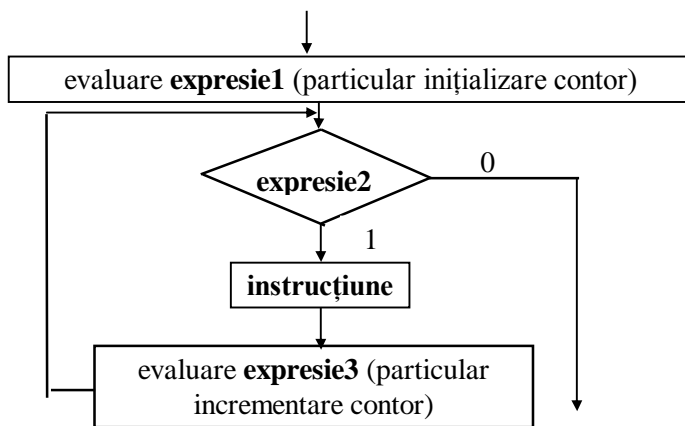


Figura 3.3. Structura ciclică cu test inițial

Reprezentare în pseudocod:

```
evaluare expresie1
CÂT TIMP expresie2
REPETĂ
    ÎNCEPUT
    instrucțiune
    evaluare expresie3
SFÂRȘIT
```

Sintaxa:

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

Nu este obligatorie prezența expresiilor, ci doar a instrucțiunilor vide.

Exemplu:

```
for ( ; expresie2; )           sau:           for ( ; ; )
    instructiune;                instructiune;
```

3.3.2. Implementarea structurilor ciclice cu test final

□ Instrucțiunea **do-while**

Sintaxa:

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

Se execută instrucțiune. Se evaluează apoi expresie. Dacă aceasta are valoarea 1, se execută instrucțiune. Se testează din nou valoarea expresiei. Se repetă instrucțiune *cât timp* valoarea expresiei este 1 (condiția este îndeplinită). În cazul instrucțiunii do-while, corpul ciclului se execută cel puțin o dată.

Exerciții:

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

```
#include <iostream.h>
#include <conio.h>
void main()
{ char c; clrscr();
int lmic=0, lmare=0, lcif=0;
int altcar=0;
cout<<"Aștept 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";
        lmic++; }
    else if (c>='0' && c<='9') {
        cout<<"Cifră!\n";
        lcif++; }
    else {
        cout<<"Alt car.!\n";
        altcar++; }
    cout<<"Aștept car.:";cin>>c;
}
cout<<"Ați introdus \n";
cout<<lmare<<" litere mari, ";
cout<<lmic<<" litere mici\n";
cout<<lcif<<" cifre și \n";
cout<<altcar<<" alte carctere\n";
```

Observații legate de implementare

Variabila c (tip char) memorează caracterul introdus la un moment dat, de la tastatură. Variabilele întregi lmic, lmare, lcif și altcar sunt utilizate pe post de contor pentru litere mari, mici, cifre, respectiv alte caractere. Acțiunea care se repetă cât timp caracterul citit este diferit de constanta caracter '@' constă din mai multe acțiuni simple: citirea unui caracter (cu afișarea în prealabil a mesajului "Aștept car.:"; testarea caracterului citit (operatorii relaționali pot fi aplicați datelor de tip char).

Ca urmare, acțiunea din corpul instrucțiunii while a fost implementată printr-o instrucțiune bloc.

Tot instrucțiuni bloc au fost utilizate pe fiecare ramură a instrucțiunii if (afișare mesaj referitor la caracter și incrementare contor).

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

Pentru implementarea aceleiași algoritm se poate utiliza instrucțiunea for. În cadrul acesteia, expresie1 și expresie3 lipsesc, însă prezența instrucțiunilor vide este obligatorie.

O altă variantă de implementare poate fi următoarea, în care și inițializarea 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 instrucțiunea do-while:

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

2. Să se calculeze suma și produsul primelor n numere naturale, n fiind introdus de la tastatură. Se vor exemplifica modalitățile de implementare cu ajutorul instrucțiunilor do-while, while, și 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 posibilități oferite de instrucțiunea 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 șir de numere reale, până la întâlnirea numărului 900. Să se afișeze 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 șir este:"<<max<<"\n";
}
```

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

4. Să se afișeze literele mari ale alfabetului și codurile aferente acestora în ordine crescătoare, iar literele mici și codurile aferente în ordine descrescătoare. Afișarea 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 numerație, $b < 10$ (N și b citite de la tastatură). Conversia unui număr întreg din baza 10 în baza b se realizează prin împărțiri succesive la b și 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):

$$1 + \sum_{k=1}^{\infty} \frac{x^k}{k}, \quad x \in [0,1], \quad x \text{ citit de la tastatură.}$$

Vom aduna la sumă încă un termen cât timp diferența dintre suma calculată la pasul curent și 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; //funcția 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. FACILITĂȚI DE ÎNTRERUPERE A UNEI SECVENȚE

Pentru o mai mare flexibilitate (tratarea excepțiilor care pot apare în procesul de prelucrare), în limbajul C se utilizează instrucțiunile **break** și **continue**. Ambele instrucțiuni sunt utilizate în instrucțiunile ciclice. În plus, instrucțiunea **break** poate fi folosită în instrucțiunea **switch**.

□ Instrucțiunea **break**

Așa cum se observă din figura 3.4., utilizată în cadrul instrucțiunilor ciclice, instrucțiunea **break** "forțează" ieșirea din acestea. Fără a se mai testa valoarea expresiei (condiția) care determină repetarea corpului instrucțiunii ciclice, se continuă execuția cu instrucțiunea care urmează instrucțiunii ciclice. Astfel, se întrerupe repetarea corpului instrucțiunii ciclice, indiferent de valoarea condiției de test.

Utilizarea în cadrul instrucțiunii **switch**: În situația în care s-a ajuns la o valoare a unei expresii constante egală cu cea a expresiei aritmetice, se execută instrucțiunea corespunzătoare acelei ramuri. Dacă se întâlnește instrucțiunea **break**, parcurgerea este întreruptă (nu se mai compară valoarea expresiei aritmetice cu următoarele constante), deci se va trece la execuția primei instrucțiuni de după **switch**. Dacă nu este întâlnit **break**, parcurgerea continuă. Instrucțiunea **break** cauzează deci, ieșirea imediată din **switch**.

□ Instrucțiunea **continue**

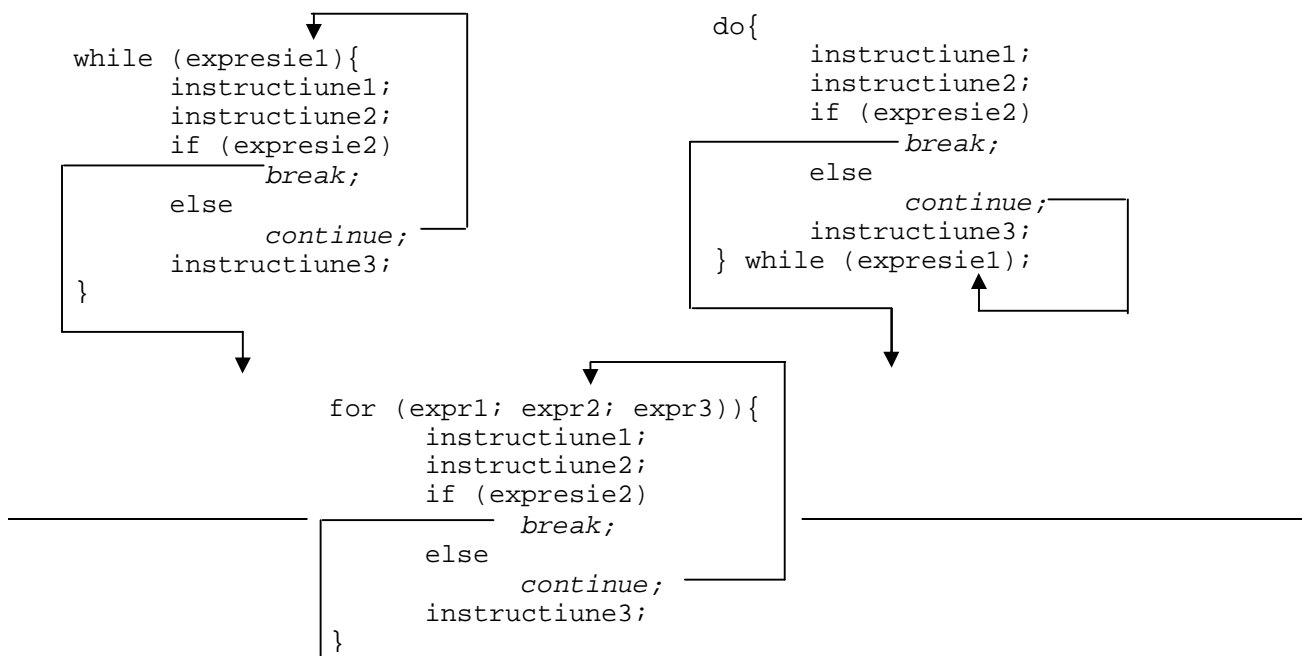
Întâlnirea instrucțiunii **continue** (figura 3.4.) determină ignorarea instrucțiunilor care o urmează în corpul instrucțiunii ciclice și reluarea execuției cu testarea valorii expresiei care determină repetarea sau nu a corpului ciclului.

Exemplu: Să revenim la programul realizat pentru problema 1, care folosește instrucțiunea **dowhile**. Dacă primul caracter citit este chiar caracterul **@**, se realizează testarea acestuia; ca urmare, se afișează mesajul "Alt car.!" și se incrementează valoarea contorului **altcar**. Dacă nu se dorește ca acest caracter să fie testat și numărat, în corpul instrucțiunii **do while** putem face un test suplimentar.

```

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

```



ÎNTREBĂRI ȘI EXERCII

Chestiuni teoretice

1. Care sunt instrucțiunile care implementează în limbajul C structura condițională?
2. Care sunt instrucțiunile care implementează în limbajul C structura secvențială?
3. Care sunt instrucțiunile care implementează în limbajul C structura repetitivă cu test inițial?
4. Care sunt instrucțiunile care implementează în limbajul C structura repetitivă cu test final?
5. Ce deosebiri sunt între instrucțiunea while și instrucțiunea do-while?
6. Pornind de la sintaxa instrucțiunii for, stabiliți echivalența între aceasta și instrucțiunile while și do-while.

Chestiuni practice

1. Să se implementeze programele cu exemplele prezentate.
2. Să se scrie programele pentru exercițiile rezolvate care au fost prezentate.
3. Să se implementeze algoritmi proiectați 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 ($a \leq b+c$, $b \leq c+d$, $c \leq a+b$).
5. Să se rescrie următoarea secvență, folosind o singură instrucțiune if.

```
if (n<0)
    if (n>=90)
        if (x!=0)
            int b= n/x;
```
6. Să se citească un număr natural n. Să se scrie un program care afișează dacă numărul n citit reprezintă sau nu, un an bisect (anii biseși sunt multipli de 4, exceptând multiplii de 100, dar incluzând multiplii de 400).
7. Să se găsească toate numerele de două cifre care satisfac relația:

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

8. Să se citească un șir de numere reale, până la întâlnirea numărului 800 și să se afișeze valoarea minimă introdusă, suma și produsul elementelor șirului.
9. Scrieți 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 funcția

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

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

11. Să se scrie un program care calculează și afișează maximul a 3 numere reale (a, b și c) citite de la tastatură.
12. Să se scrie un program care calculează și afișează minimul a 3 numere reale (a, b și c) citite de la tastatură.
13. Să se citească 2 caractere care reprezintă 2 litere mari. Să se afișeze caracterele citite în ordine alfabetică.
14. Să se citească 3 caractere care reprezintă 3 litere mici. Să se afișeze caracterele citite în ordine alfabetică.
15. Să se scrie un program care citește o cifră. În funcție de valoarea ei, să se facă următorul calcul: dacă cifra este 3, 5 sau 7 să se afișeze pătratul valorii numerice a cifrei; dacă cifra este 2, 4 sau 6 să se afișeze

cubul valorii numerice a cifrei; dacă cifra este 0 sau 1 să se afișeze mesajul "Valori mici"; altfel., să se afișeze mesajul "Caz ignorat!".

16. Fie șirul 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 șirului Fibonacci.

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

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

18. Să se citească câte 2 numere întregi, până la întâlnirea perechii $(0, 0)$. Pentru fiecare pereche de numere, să se calculeze și să se afișeze 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 afișeze maximum.
20. Se citește câte un caracter până la întâlnirea caracterului @. Să se afișeze numărul literelor mari, numărul literelor mici și numărul cifrelor citite; care este cea mai mare (lexicografic) literă mare, literă mică și 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 afișeze cel mai mare divizor comun al acestora.
22. Să se calculeze suma seriei

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

cu o eroare mai mică decât epsilon (epsilon citit de la tastatură). Să se afișeze și numărul de termeni ai sumei.

23. Să se citească un număr întreg format din 4 cifre (abcd). Să se calculeze și să se afișeze valoarea expresiei reale: $4*a + b/20 - c + 1/d$.
24. Să se scrie un program care afișează 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 următoare, cu o eroare EPS mai mică de 0.0001:
- $$S=1+(x+1)/2! + (x+2)/3! + (x+3)/4! + \dots$$
- , unde $0 \leq x \leq 1$, 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 și unităților.
28. Să se citească câte un număr întreg, până la întâlnirea numărului 90. Pentru fiecare număr să se afișeze un mesaj care indică dacă numărul este pozitiv sau negativ. Să se afișeze cel mai mic număr din șir.
29. Să se genereze toate numerele naturale de 3 cifre pentru care cifra zecilor este egală cu diferența cifrelor sutelor și unităților.
30. Să se calculeze suma:

$$(1 + 2!) / (2 + 3!) - (2+3!) / (3+4!) + (3+4!) / (4+5!) - \dots$$