

**Introducere**

Așa cum s-a arătat, calculatorul prelucrează datele de intrare printr-o succesiune de transformări pentru a obține datele de ieșire. Succesiunea de prelucrări este descrisă prin program. Fiecare prelucrare este rezultatul uneia sau mai multor instrucțiuni.

Ordinea de efectuare a prelucrărilor este importantă, ea respectând un anumit algoritm.

Având în vedere faptul că un program este construit pe baza unui algoritm, în acest laborator vor fi prezentate generalități despre algoritmi și modul de reprezentare a acestora.

Prezentarea algoritmilor nu este exhaustivă, având în vedere faptul că prezentul curs are ca subiect de interes limbaje de programare (C/C++), prezentarea se limitează doar la câteva aspecte necesare în înțelegerea instrucțiunilor specifice limbajelor de programare și a structurării programelor.

În continuare sunt prezentate instrucțiunile specifice limbajelor C/C++.

În prezentul laborator sunt prezentate instrucțiunile expresie, instrucțiuni compuse și instrucțiunile de decizie, celelalte instrucțiuni urmând a fi discutate în laboratorul 6.

**5.1. Algoritmi****5.1.1. Generalități**

Pot fi scriși algoritmi care să descrie activități din cele mai diverse domenii de viață, dar algoritmi destinați scrierii programelor de calculator sunt cei care rezolvă probleme ce admit o formulare matematică.

Algoritmul poate fi gândit ca o succesiune de acțiuni ce se pot aplica pentru obținerea unui rezultat. În toate domeniile există activități care pot fi descrise prin algoritmi.

**5.1.2. Reprezentarea algoritmilor**

În general, descrierea algoritmului este o etapă intermediară între formularea matematică și programul scris într-un limbaj oarecare.

Descrierea algoritmului se poate face folosind diferite metode. Frecvent folosite sunt următoarele:

- **reprezentarea în limbaj natural** – acest mod de reprezentare nu presupune o pregătire specială, dar pentru probleme complexe poate introduce neclarități;

- **reprezentarea prin scheme logice** – este o reprezentare grafică în care fiecare etapă este reprezentată printr-un bloc, blocurile fiind înălțuite prin segmente orientate, ordinea lor respectând logica problemei ;

- **reprezentarea în limbaj pseudocod** – limbajul pseudocod este apropiat de codificarea din Pascal sau C, permite o reprezentare clară și concisă;

- **reprezentarea prin tabele de decizie** – metoda e specifică problemelor cu număr mare de alternative, ele apelând la logica booleană inventariază toate acțiunile posibile prin combinarea condițiilor precizate.

Pentru etapa de familiarizare cu un limbaj de programare și cu logica algoritmică și

utilizarea structurilor de control, metoda de reprezentare prin scheme logice, care este o metodă ilustrativă, este sugestivă, etapele și fluxul de prelucrare fiind ușor de urmărit.

Există trei tipuri de scheme logice:

- **scheme de program** – care descriu toate procesele de prelucrare pornind de la datele de intrare până la obținerea rezultatelor;

- **scheme de detaliere** – care descriu procese complexe la nivel de detaliu, secvențele apărând o singură dată în program;

- **scheme de subprograme (rutine)** – care descriu procese repetabile în program.

Blocurile folosite pentru reprezentarea etapelor de evoluție a algoritmului sunt următoarele:

Pentru realizarea schemelor logice, în anii '70, s-au impus **principiile programării structurate**. Acestea au dus la o puternică dezvoltare a limbajelor de programare.

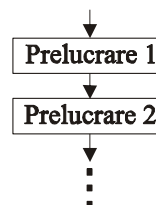
Prin aplicarea principiilor programării structurate, se obține:

- claritate în citirea programelor;
- depanarea ușoară a programelor;
- generalitatea programelor și ușurință în refolosirea codului scris.

Structurile de control care stau la baza programării structurate sunt:

- **structura secvențială;**
- **structura alternativă;**
- **structura repetitivă.**

- **Structura secvențială** – reprezintă o înlanțuire de prelucrări în care o prelucrare începe numai după finalizarea precedentei.



- **Structuri alternative (de decizie)** – selectează variante de prelucrate în funcție de condițiile evaluate prin intermediul unei expresii logice.

Acestea pot fi:

- **Structura pseudo-alternativă** (fig. 5.1.)
- **Structura alternativă simplă** (fig. 5.2.)
- **Structura alternativă multiplă** (fig. 5.3.)

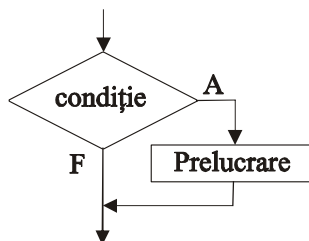


Fig. 5.1. Structură pseudo-alternativă

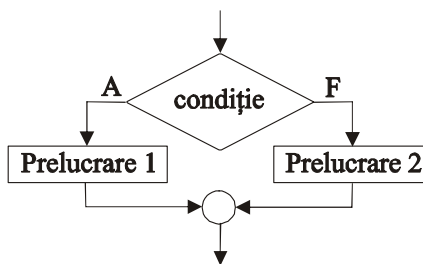


Fig. 5.2. Structură alternativă simplă

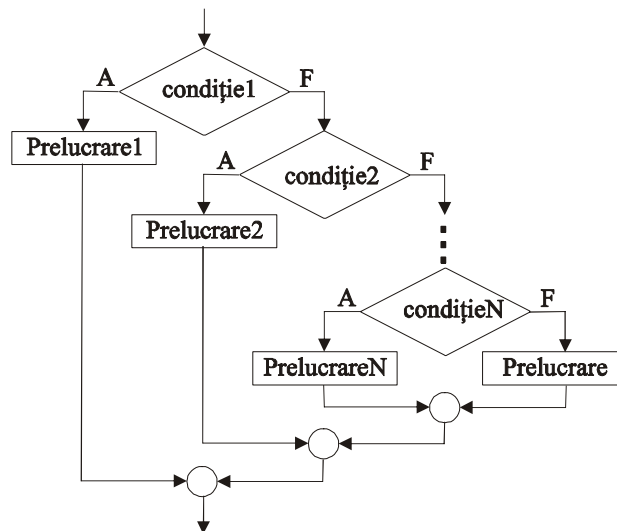


Fig. 5.3. Structură alternativă multiplă

- **Structuri repetitive** – descriu un proces ce se repetă în funcție de îndeplinirea anumitor condiții:
  - Structura repetitivă cu test anterior (test inițial) (fig. 5.4.)
  - Structura repetitivă cu test posterior (test final) (fig. 5.5.)
  - Structura repetitivă cu contor (fig. 5.6.)

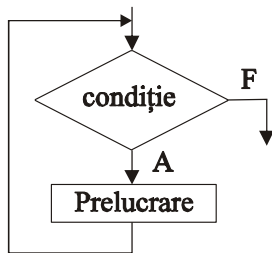


Fig.5.4. Structură repetitivă cu test inițial

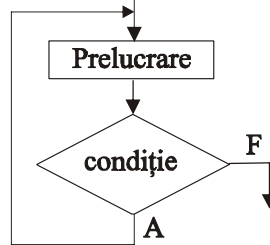


Fig.5.5. Structură repetitivă cu test final

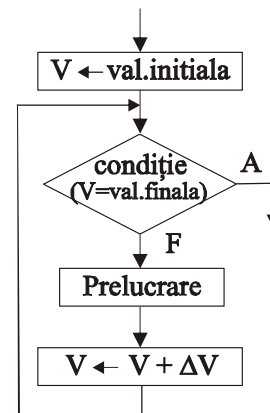


Fig.5.6. Structură repetitivă cu contor

## 5.2. Instrucțiunea expresie

Prelucrările efectuate de program sunt descrise prin instrucțiuni.

Modificările făcute asupra unor variabile se realizează prin operația de atribuire, valoarea atribuită fiind rezultatul evaluării unei expresii. Instrucțiunea care realizează acest lucru se numește **instrucțiune expresie**.

Toate celelalte instrucțiuni stabilesc succesiunea prelucrărilor folosind cele trei structuri de control: secvențiale (succesiuni de operații, instrucțiuni compuse), de decizie și repetitive.

O instrucțiune expresie este o expresie care se încheie cu punct și virgulă (;) sintaxa fiind:

**< expresie > ;**

Exemple de instrucțiuni expresie:

```
int i, j;
float r;
...
i = 10;
j = i * 5;
r = (float) i / j + 1.5 ;
```

Instrucțiunile se pot scrie pe mai multe linii de program, spațiile nesemnificative fiind ignorate.

```
int i;
i=
5;
```

Pe o linie de program se pot scrie mai multe instrucțiuni.

```
int i, j;
i = 2; j += i;
```

Pot fi incluse instrucțiuni expresie care nu produc efecte. Acestea sunt semnalate de compilator printr-un mesaj de atenționare (warning - cod as no efect). Acestea nu introduc erori în evoluția programului, dar încarcă în mod inutil codul.

- **Instrucțiunea vidă** – se încadrează tot în categoria instrucțiunilor expresie.

Instrucțiunea vidă are forma:

;

Instrucțiunea vidă semnifică lipsa oricărei acțiuni. Ea este utilă în anumite situații când sintaxa impune o instrucțiune, însă programul nu cere nici o acțiune.

Exemplele de folosire a instrucțiunii vide vor fi incluse în paragrafele următoare.

### 5.3. Instrucțiunea compusă (blocul de instrucțiuni)

Instrucțiunea compusă grupează declarații și instrucțiuni, începutul și sfârșitul blocului fiind marcat de acolade ( { } ). Blocul de instrucțiuni este sintactic echivalent cu o instrucțiune.

Sintaxa generală este:

```
{
    lista_declaratii;
    lista_instructiuni;
}
```

Conținutul unui bloc de instrucțiuni poate fi descris într-un algoritm printr-o structură secvențială.

Instrucțiunile compuse se utilizează deseori în combinație cu instrucțiunile de ciclare și cele de decizie.



#### Observații

- Toate variabilele declarate în interiorul unei instrucțiuni compuse **nu pot fi accesate** în afara acesteia, domeniul lor de existență fiind blocul de instrucțiuni.

Exemplu de utilizare a blocului de instrucțiuni:

```
#include <stdio.h>

void main()
{
    int i;                // se declară i, variabilă locală funcției main()
    int k;                // se declară k, variabilă locală funcției main()
    i=7;
    k=12;
    {                    // începutul blocului
        int i;            // se declară variabila i locală blocului
        int j;            // se declară variabila j locală blocului
        i=9;              // se folosește variabila i locală blocului
        j=2*i;
        k++;              // în interiorul blocului se folosește
                          // variabila k, declarată în main(), ea având
                          // ca domeniu întreaga funcție, inclusiv
                          // blocul de instrucțiuni

        printf("\ni=%d", i); // se afișează i=9, valoarea variabilei
                              // locale blocului de instrucțiuni

        printf("\nj=%d", j); // se afișează j=18
        printf("\nk=%d", k); // se afișează k=12
    }                       // sfârșitul blocului

    printf("\ni=%d", i);     // se afișează i=7, valoarea variabilei locale
                              // funcției main()
    printf("\nj=%d", j);     // eroare, instrucțiunea se află în afara
                              // domeniului de existență a lui j
    printf("\nk=%d", k);     // se afișează k=13
}
```

## 5.4. Instrucțiuni de selecție (decizie)

### 5.4.1. Instrucțiunea de decizie if, if-else

Sintaxa instrucțiunii poate fi:

```
if(condiție)
    instrucțiune_1;                                (1)
```

sau

```
if(condiție)
    instrucțiune_1;
else
    instrucțiune_2                                (2)
```

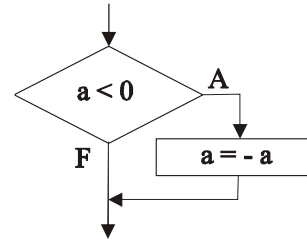
Instrucțiunea if evaluează logic expresia “condiție”. Se are în vedere faptul că se interpretează valoare logic adevărată orice valoare numerică diferită de 0 (zero), și logic falsă valoarea 0. Deci, în cazul în care valoarea expresiei “condiție” este nenulă, se execută instrucțiune\_1, iar dacă este nulă se execută instrucțiunea imediat următoare instrucțiunii if, pentru prima formă a sintaxei, sau instrucțiune\_2 în cazul celei de a doua forme.

Instrucțiunea if / if-else reprezintă o structură alternativă, prima formă, (1), fiind o structură pseudo-alternativă, cea de a doua, (2) reprezentând structura alternativă simplă.

Expresia “condiție” trebuie să fie o expresie scalară, care returnează o valoare.

De exemplu, dacă se dorește ca o variabilă să păstreze valoarea sa absolută, se poate folosi instrucțiunea if (secvență pseudo-alternativă):

```
int a;           //
...             //
if (a < 0)       // instrucțiunea if pseudo-alternativă
    a = -a;     // poate fi descrisă în schemă logică
...             // prin figura alăturată
```



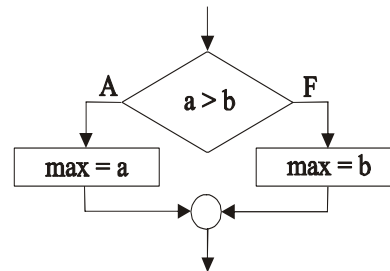
### Observații

- Secvența scrisă anterior poate fi scrisă și folosind operatorul ternar, folosind instrucțiunea:

$a = (a < 0) ? -a : a ;$

Dacă se dorește determinarea maximului dintre două valori, se poate folosi instrucțiunea if – else (secvență alternativă simplă):

```
int a, b, max;  //
...            //
if (a > b)       // instrucțiunea if – else poate fi
    max = a;    // descrisă în schemă logică prin
else            // figura alăturată
    max = b;    //
...            //
```



### Observații

- Altă variantă de scriere a secvenței anterioare poate folosi o secvență pseudo-alternativă (if):

```
max = a;
if (max < b)
    max = b;
```

- Secvența scrisă anterior poate fi scrisă și folosind operatorul ternar, folosind instrucțiunea:

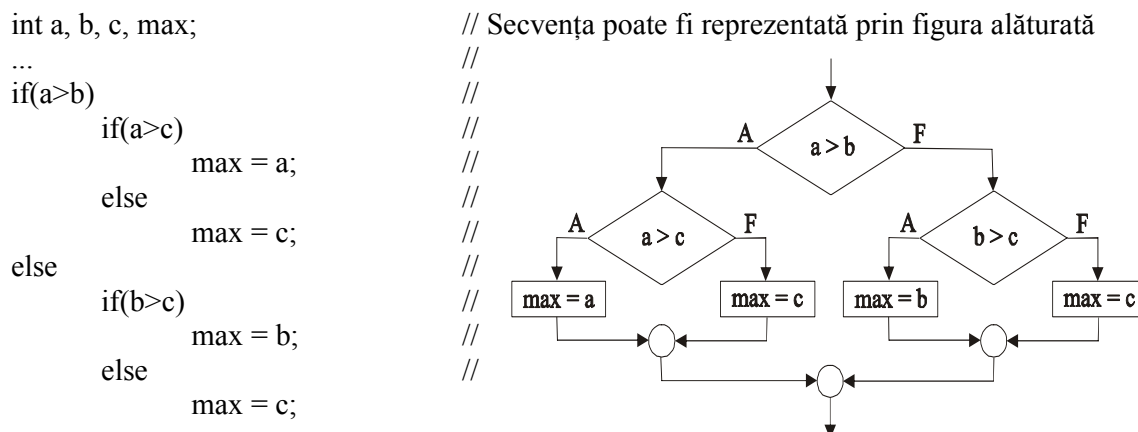
$max = (a > b) ? a : b ;$

- Nu se poate înlocui instrucțiunea if / if-else cu operatorul ternar în orice situație. Acest lucru este posibil doar dacă instrucțiune\_1 și instrucțiune\_2 returnează valoare compatibilă cu tipul variabilei aflată în stânga operatorului de atribuire.
- Prin folosirea operatorului ternar rezultă un cod mai compact, dar folosirea instrucțiunii if este mai sugestivă și face mai ușoară citirea programului.

Instrucțiunea if/if-else poate fi folosită și pentru descrierea secvențelor alternative multiple. În acest caz se recurge la folosirea instrucțiunilor if imbricate, adică, una, sau ambele dintre

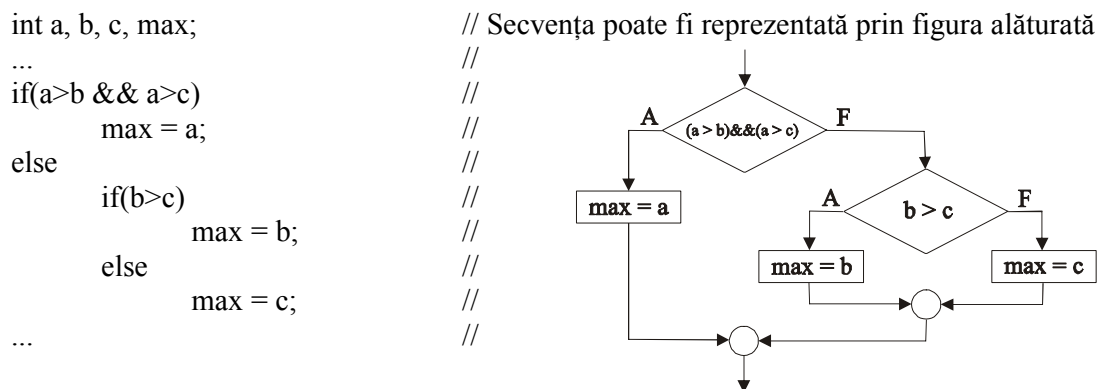
instrucțiune\_1 și instrucțiune\_2 pot fi la rândul lor instrucțiuni if/if-else.

Pentru ilustrarea scrierii acestor secvențe, se ia ca exemplu secvența următoare, în care se determină maximul dintre două valori:



Aceeași secvență se poate rescrie în diferite variante. În general, orice problemă poate fi implementată în diferite variante. Aceasta depinde de stilul adoptat de programator și, de multe ori, o variantă este adoptată în urma analizei algoritmului adoptat, astfel încât să se obțină eficiență cât mai mare.

În continuare este prezentată o altă variantă de secvență care determină maximul dintre trei valori.



### 5.4.2. Instrucțiunea de selecție multiplă switch

Instrucțiunea switch este o instrucțiune decizională, ea permițând selecția, în funcție de valoarea unei expresii, între mai multe opțiuni posibile. Sintaxa instrucțiunii este:

```

switch (expr)
{
    case const1: lista_instructiuni;
                <break;>
    case const2: lista_instructiuni;
                <break;>
    ....
    <default> lista_instructiuni;
}

```

unde:

- *expr* este o expresie întreagă (orice tip întreg sau enumerare);
- *const1, const2,...* sunt constante de selecție, de tip întreg, cu valori distincte;
- *lista\_instructiuni* este o secvență de instrucțiuni;

La execuția instrucțiunii, se evaluează *expr*. Valoarea obținută se compară succesiv cu valorile constantelor de selecție. Dacă se găsește identitate de valoare, se execută succesiunea de instrucțiuni asociată.

Instrucțiunea *break* întrerupe execuția instrucțiunii *switch*. Dacă *lista\_instructiuni* nu este urmată de *break*, se continuă execuția cu lista de instrucțiuni atașată următoarei etichete.

Dacă valoarea expresiei nu se regăsește printre etichete, se execută secvența de instrucțiuni asociată cu eticheta *default*. În lipsa acesteia, se finalizează execuția instrucțiunii *switch*.

Pentru exemplificarea folosirii instrucțiunii *switch* este prezentată în continuare următoarea aplicație: se citesc de la tastatură două valori numerice întregi; se citește de la tastatură operația care se dorește a se efectua între acești doi întregi (+, -, \*, /, %) și se afișează rezultatul acesteia.

```
#include <stdio.h>

void main()
{   int a, b;    // variabile folosite pentru citirea celor doi întregi
    char op;    // variabila op este folosită pentru citirea ca și caracter a tipului de operație dorit

    printf("Introdu doua valori intregi:");
    printf("\na=");
    scanf("%d", &a);
    printf("\nb=");
    scanf("%d", &b);
    printf("Introdu un operator aritmetic, simplu, binar (+, -, *, /, %) :");

    fflush(stdin);    // golirea buffer-ului de intrare astfel încât citirea următoare, a
                     // caracterului care desemnează operația, să se facă corect
    scanf("%c", &op);    // citirea se face în format caracter

    switch (op)        // expresia folosită pentru selecție este chiar valoarea
                       // caracterului citit de la tastatură
    {
        case ('+'): printf("\n Operatia de adunare - rezultat: %d", a+b);
                     break;
        case ('-'): printf("\n Operatia de scadere - rezultat: %d", a-b);
                     break;
        case ('*'): printf("\n Operatia de inmultire - rezultat: %d", a*b);
                     break;
        case ('/'): printf("\n Operatia de impartire - rezultat: %d", a/b);
                     break;
        case ('%'): printf("\n Operatia de impartire intreaga - rest: %d", a%b);
                     break;
        default:    printf("\nOperatie ilegala!");
    }
}
```



Secvența switch care selectează operația aritmetică ce se efectuează implementează schema logică din Fig. 5.7.

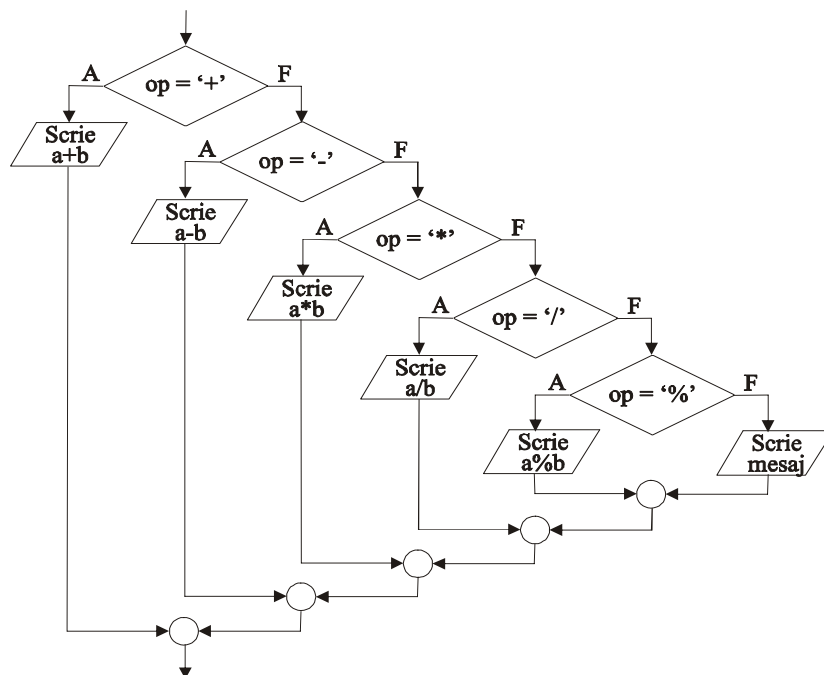


Fig.5.7. Schema logică pentru secvența de selecție a operației



### Observații

- Selecția multiplă se poate implementa, așa cum am văzut, folosind instrucțiunea if/if-else, dar, dacă selecția este controlată de valoarea unei singure expresii, este mai eficient și mai clar să se folosească instrucțiunea switch.
- Instrucțiunea switch evaluează doar egalitatea de valori, în timp ce if poate evalua orice tip de expresie relațională sau logică.
- Dacă în instrucțiunea switch constantele sunt de tip caracter, automat sunt convertite la întreg prin codul ASCII corespunzător.
- Într-o instrucțiune switch nu pot exista mai multe constante case cu valori identice.

Se pot asocia mai multe etichete scrise consecutiv cu aceleași secvențe de instrucțiuni. Această situație se regăsește în exemplul următor.

Se dorește asocierea caracterelor ce reprezintă o cifră hexazecimală cu valoarea numerică corespunzătoare în baza 10. Caracterele pot fi: 0, 1, 2,...9, a, A, b, B..., f, F.

```

char cifra;
int i;
//...
switch (cifra)
{case '0': i=0; break;
 case '1': i=1; break;

```

```
//...
case '9': i=9; break;
case 'a': // dacă cifra are ca valoare caracterul 'a' sau 'A'
case 'A': i=10; break; // i primește aceeași valoare, 10
case 'b': // dacă cifra are ca valoare caracterul 'b' sau 'B'
case 'B': i=11; break; // i primește aceeași valoare, 11
case 'c': // dacă cifra are ca valoare caracterul 'c' sau 'C'
case 'C': i=12; break; // i primește aceeași valoare, 12
case 'd': // dacă cifra are ca valoare caracterul 'd' sau 'D'
case 'D': i=13; break; // i primește aceeași valoare, 13
case 'e': // dacă cifra are ca valoare caracterul 'e' sau 'E'
case 'E': i=14; break; // i primește aceeași valoare, 14
case 'f': // dacă cifra are ca valoare caracterul 'f' sau 'F'
case 'F': i=15; // i primește aceeași valoare, 15
// fiind ultima instrucțiune din secvența switch, nu
// este necesară instrucțiunea break
}
```



## Exemple

```

/*****
Exemplul 5.1. - Se scrie un program pentru rezolvarea ecuației de gradul II:  $a \cdot x^2 + b \cdot x + c = 0$ 
*****/
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>

void main()
{float a, b, c, delta, x1, x2;
 printf("coeficientul lui x^2:");
 if (scanf("%f", &a)!=1)
 {printf("Coeficientul lui x^2 este incorect!\n");
 getch();
 return; // se poate folosi si apelul functiei: exit(0);
 }

 printf("coeficientul lui x:");
 if (scanf("%f", &b)!=1)
 {printf("Coeficientul lui x este incorect!\n");
 getch();
 return; // exit(0);
 }

 printf("termenul liber:");
 if (scanf("%f", &c)!=1)
 {printf("Termenul liber este incorect!\n");
 getch();
 return; // exit(0);
 }
 if (a==0 && b==0 && c==0)

```

```

{printf("Ecuatie nedeterminata !\n");
 getch();
 return;    // exit(0);
}

if (a==0 && b==0)
{printf("Ecuatie imposibila !\n");
 getch();
 return;    // exit(0);
}

if (a==0)
{printf("Ecuatie de grad I");
 printf("\nSolutia este: x=%.2f\n", -c/b);
 getch();
 return;    // exit(0);
}

delta= b*b - 4*a*c;
if ( delta<0)
{printf("radacini complexe\n");
 printf("x1= %.2f + i %.2f\n", -b/(2*a), sqrt(-delta)/(2*a));
 printf("x2= %.2f - i %.2f\n", -b/(2*a), sqrt(-delta)/(2*a));
 getch();
 return;    // exit(0);
}

if ( delta==0 )
{printf("Radacini reale si egale:\n");
 printf("x1=x2=%.2f\n", -b/(2*a));
 getch();
 return;    // exit(0);
}

printf("Radacini reale si distincte:");
printf("\nx1=%.2f\n", (-b+sqrt(delta))/(2*a));
printf("\nx2=%.2f\n", (-b-sqrt(delta))/(2*a));
getch();
}

```

/\* \*\*\*\*\*

**Exemplu1 5.2.** - Se scrie un program care citește de la tastatură un întreg. Corespunzător valorii acestuia se afișează denumirea zilei corespunzătoare din săptămână.

\*\*\*\*\* /

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

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

```
main()
```

```
{
```

```
int zi;
```

```
printf("\nIntrodu un numar <=7 :");
```

```
scanf("%d", &zi);
```

```
switch (zi)
```

```

        {
            case 1: printf("\nZiua %d este Luni", zi);break;
            case 2: printf("\nZiua %d este Marti", zi);break;
            case 3: printf("\nZiua %d este Miercuri", zi);break;
            case 4: printf("\nZiua %d este Joi", zi);break;
            case 5: printf("\nZiua %d este Vineri", zi);break;
            case 6: printf("\nZiua %d este Sambata", zi);break;
            case 7: printf("\nZiua %d este Duminica", zi);break;
            default: printf("\nValoarea introdusa este incorecta !");
        }
    printf("\n");
    getch();
}

```



### Întrebări. Exerciții. Probleme.

1. Să se scrie un program în care se declară o variabilă de tip int pentru care se citește valoare de la tastatură. Să se afișeze un mesaj care să indice dacă valoarea este pară sau impară.
2. Să se scrie un în care se declară trei variabile de tip float pentru care se citesc valori de la tastatură. Să se verifice dacă valorile pot reprezenta laturile unui triunghi și dacă da, să se calculeze și să se afișeze aria triunghiului.
3. Sa se scrie un program in care se citeste de la tastatura o data calendaristica (zi, luna, an - citite ca valori întregi). Sa se afișeze data, pentru luna fiind afișata denumirea. De exemplu, data 15.10.2009 va fi afișată 15 octombrie 2009. In secvența de afisare se va folosi instrucțiunea switch.
4. Să se scrie un program în care se declară două variabile de tip int pentru care se citesc valori de la tastatură. Se selectează tipul operației care se dorește a se efectua între cele două valori prin apăsarea simbolului asociat operației ( +, -, \*, /, %).