Programarea Calculatoarelor

Cursul 3: Instrucţiuni

Ion Giosan
Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Instrucţiuni

- Reprezintă elementele fundamentale ale funcţiilor şi totodată comenzile date calculatorului
- Determină fluxul de control al programului
- Instrucţiuni de bază (simple)
  - Instrucţiunea expresie
  - Instrucţiunea vidă
  - Instrucţiunea alternativă
  - Instrucţiunea selectivă
  - Instrucţiuni repetitive
  - Instrucţiuni de salt
- Instrucțiuni compuse
  - Presupune gruparea mai multor instrucțiuni de tipul celor de bază

Program structurat în C

- Conţine numai trei mari tipuri de instrucţiuni de control al fluxlui
  - Instrucţiuni secvenţiale (compuse)
    - Execuția secvențială presupune execuția tuturor instrucțiunilor în ordinea în care apar scrise
    - Este exclusă utlizarea instrucțiunilor de salt
  - Instrucţiuni alternative şi selective
  - Instrucţiuni repetitive

- Observaţie
  - Programarea structurată presupune faptul că programele scrise nu utilizează instrucţiuni de salt

Instrucţiunea expresie

- Reprezintă orice expresie urmată de caracterul “punct şi virgulă”

expresie;

- Utilizată la atribuiri sau la apeluri de funcţii
  - Expresiile au de obicei efecte secundare adică schimbă valoarea unui operand
- Exemplu

```
#include <stdio.h>
int main()
{
   int a, b, c;
   printf("\nIntroduceti doi intregi, a si b\n");
   scanf("%d %d", &a, &b);
   c=(a > b)? a: b; /* instructiune expresie */
   printf("\nMaximul dintre a=%d si b=%d este c=%d\n", a, b, c);
   return 0;
}
```

Instrucţiunea vidă

- Este o instrucţiune care conţine doar caracterul **;**
- Nu are niciun efect
- Este utilă atunci când este necesară prezenţa unei instrucţiuni dar care să nu execute absolut nimic
  - De exemplu în instrucţiuni repetitive cu număr cunoscut de iteraţii
- Exemplu

```
/* Suma elementelor unui sir de numere intregi */
#include <stdio.h>
int main()
{
   int a[] = { 2, 5 ,8 };
   int n = sizeof(a) / sizeof(int);
   int i, s;
   for (i = 0, s = 0; i < n; s += a[i], i++);
   printf("\nSuma tuturor elementelor este %d\n", s);
   return 0;
}
```

Instrucţiunea compusă

- Este o secvenţă de instrucţiuni şi declaraţii cuprinse între acolade
  - Compilatorul va trata această secvenţă ca pe o singură instrucţiune
- Utilă atunci când sintaxa limbajului presupune o singură instrucţiune, dar programul trebuie să efectueze mai multe instrucţiuni
- Este numită şi instrucţiune bloc
- Format

```
{
        declaraţii;
        instrucţiuni;
}
```

Instrucţiunea alternativă *if*

- Ramifică fluxul de control pe una din maxim două alternative în funcţie de valoarea de adevăr a expresiei evaluate
- Forma generală

```
if ( expresie )
        instructiune_1
else
        instructiune_2
```

- **expresie**
  - Trebuie obligatoriu inclusă între paranteze rotunde
  - Trebuie să se evalueze la o valoare scalară
- Dacă **expresie** este adevărată (nenulă) atunci se execută **instructiune_1** altfel se execută **instrucţiune_2**
- Ramura **else** poate uneori să lipsească

Instrucţiunea alternativă *if* (exemplu)

```
/* Calculul radacinilor ecuatiei a*x^2+b*x+c=0 */
#include <stdio.h>
#include <math.h>
int main() {
   float a, b, c, delta, x1, x2;
   printf("\nIntroduceti coeficientii a, b, c\n");
   scanf("%f %f %f", &a, &b, &c);
   if (a !=0 ) { // instructiune compusa
     delta = b * b - 4 * a * c;
     if ( delta >= 0 ) { // instructiune compusa
        x1 = ( - b - sqrt( delta )) / ( 2 * a );
        x2 = ( - b + sqrt( delta )) / ( 2 * a );
        printf("\nEcuatia are radacinile reale x1=%g si x2=%g\n",
                                                    x1, x2);
     }
     else  { // instructiune compusa
        x1 = - b / ( 2 * a );
        x2 = sqrt( -delta ) / ( 2 * a );
        printf("\nEcuatia are radacinile complexe x1=%g-j*%g si
                              x2=%g+j*%g\n", x1, x2, x1, x2);
     }
   }
   else printf("\nEcuatia nu este de gradul doi\n");
  return 0; }
```

Instrucţiunea alternativă *if*

- Eroare frecventă
  - Confundarea operatorului de egalitate **==** cu operatorul de atribuire **=** în specificarea expresiei de test

Exemplu comparativ:

```
a = 2;
if  ( a == 10 )
    printf("a este 10 \n");
```

- Mesajul *a este 10* nu va fi afişat
- După testarea egalităţii folosind operatorul **==** se evaluează expresia de test la fals, adică la 0
  - 2 nu este egal cu 10

```
a = 2;
if  ( a = 10 )
    printf("a este 10 \n");
```

- Mesajul *a este 10* va fi afişat întotdeauna
- Expresia a=10 în *if*
  - a ia valoarea 10
  - Se evaluează la adevărat întrucât valoarea 10 este nenulă
  - Se trece la executarea instrucţiunii *printf*

Instrucţiunea selectivă *switch*

- Efectuează selecţia multiplă
- Este utilă atunci când expresia de evaluat poate avea mai multe valori posibile
- Forma generală

```
switch ( expresie )
{
  case C1:     instructiuni_1

  case C2:     instructiuni_2
  ...
  case Cn:     instructiuni_n

  default:     instructiuni_d  /* optional */
}
```

- Dacă valoarea expresiei testate **expresie** este egală cu una din constantele (etichetele) **Ci**, atunci fluxul de control sare direct la acea etichetă şi începe execuţia **instructiuni_i**
- Dacă valoarea expresiei testate **expresie** nu este egală cu niciuna din valorile **Ci** şi clauza **default** este prezentă, atunci fluxul de control sare la eticheta **default** şi se începe execuţia **instructiuni_d**

Instrucţiunea selectivă *switch*

- Constrângeri
  - **expresie** trebuie să poată fi evaluată la o valoare întreagă (poate fi inclusiv caracter, dar nu valori reale sau şiruri de caractere)
  - Valorile constantelor **case** notate **Ci** (numite şi etichete) trebuie să fie constante întregi (sau caracter), reprezentând o singură valoare
- Observaţii
  - Instrucțiunile care urmează după etichetele **case** nu trebuie incluse între acolade, deși pot fi mai multe instrucțiuni, iar ultima instrucțiune este de regulă instrucțiunea **break**
  - Dacă nu s-a întâlnit **break** la finalul instrucțiunilor de pe ramura (eticheta) pe care s-a intrat, atunci se continuă execuția instrucțiunilor de pe ramurile consecutive (fără verificarea etichetei) până când se ajunge la **break** sau la sfârșitul instrucțiunii **switch**, moment în care se iese din instrucțiunea **switch** și se trece la execuția instrucțiunii imediat următoare
  - Ramura **default** este opțională iar poziția relativă a acesteia printre celelalte ramuri nu este relevantă
  - Dacă nici o etichetă nu se potrivește cu valoarea expresiei testate și nu există ramura **default**, atunci instrucțiunea **switch** nu are nici un efect
  - Instrucţiunea switch ar putea fi reprezentată întotdeauna folosind mai multe instrucţiuni **if** cascadate
    - **switch** este mai rapid şi codul scris mai uşor de înţeles

Instrucţiunea selectivă *switch*

```
/* Evaluarea unei expresii aritmetice simple */
#include <stdio.h>
int main() {
   int operand1, operand2, result;
   char operatie;
   printf("Scrieti o expresie aritmetica simpla, cu doi intregi,
           operatia intre acestia si fara alte spatii\n");
   scanf("%d%c%d", &operand1, &operatie, &operand2);
   switch( operatie ) {
        case '+': result = operand1 + operand2;
                                                  break;
        case '-': result = operand1 - operand2;
                                                  break;
        case '*': result = operand1 * operand2;
                                                  break;
        case '/': result = operand1 / operand2;
                                                  break;
        default : printf("\nOperatia %c este invalida!\n", operatie);
                                                  return 1;
   }
   printf("\n%d %c %d = %d\n", operand1, operatie, operand2, result);
   return 0;
}
```

Instrucţiuni repetitive

- Se mai numesc şi instrucţiuni iterative sau ciclice
- Efectuează o serie de instrucțiuni în mod repetat fiind condiționate de o expresie de control care este evaluată la fiecare iterație
- Instrucțiunile iterative furnizate de limbajul C sunt
  - Instrucțiunea repetitivă cu testare inițială **while**
  - Instrucțiunea repetitivă cu testare finală **do-while**
  - Instrucțiunea repetitivă cu contor **for**

Instrucţiunea repetitivă *while*

- Execută în mod repetat o instrucțiune atâta timp cât expresia de control este evaluată la adevărat
- Forma generală

  while (expresie)
  instructiune
- Evaluarea expresiei de control **expresie** se efectuează întotdeauna la începutul fiecărei iteraţii
  - **while** este potrivit în implementarea de cicluri care au niciuna, una sau mai multe iteraţii
- Dacă valoarea **expresie** corespunde valorii logice adevărat atunci se execută corpului instrucțiunii, după care procedeul se reia
  - Acești pași se repetă până când expresia **expresie** va fi evaluată la fals, moment care va determina ieșirea din ciclu și trecerea la instrucțiunea imediat următoare instrucţiunii **while**
  - **instructiune** trebuie să modifice valoarea de adevăr a expresiei **expresie**, altfel apare fenomenul de ciclu infinit

Instrucţiunea repetitivă *while*

```
/* Calculul celui mai mare divizor comun si a celui mai mic
   multiplu comun pentru doua numere */

#include <stdio.h>
int main()
{
   int a, b, a1, b1, cmmdc, cmmmc, rest;

   printf("a="); scanf("%d", &a);
   printf("b="); scanf("%d", &b);
   /* gasirea cmmdc utilizand aloritmul lui Euclid */
   a1 = a; b1 = b;
   while ( (rest =a1 % b1) != 0 )
   {
        a1 = b1;
        b1 = rest;
   }
   cmmdc = b1;
   cmmmc = a * b / cmmdc;
   printf("a=%d b=%d cmmdc(a, b)=%d cmmmc(a, b)=%d\n", a, b,
                                                  cmmdc, cmmmc);
   return 0;
}
```

Instrucţiunea repetitivă *do-while*

- Execută în mod repetat o instrucțiune atâta timp cât expresia de control este evaluată la adevărat
- Forma generală

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

- Evaluarea expresiei de control **expresie** se efectuează întotdeauna la sfârşitul fiecărei iteraţii
  - **do-while** este potrivit în implementarea de cicluri care au cel puţin o iteraţie
- Se execută corpul instrucțiunii, iar apoi dacă valoarea **expresie** corespunde valorii logice adevărat atunci procedeul se reia
  - Acești pași se repetă până când expresia **expresie** va fi evaluată la fals, moment care va determina ieșirea din ciclu și trecerea la instrucțiunea imediat următoare instrucţiunii **do-while**
  - **instructiune** trebuie să modifice valoarea de adevăr a expresiei **expresie**, altfel apare fenomenul de ciclu infinit

Instrucţiunea repetitivă *do-while*

- Efectul este acelaşi cu cel al unui ciclu **while** dar care execută mai întâi **instructiune** o singură dată:

instructiune

while (expresie)
instructiune

Instrucţiunea repetitivă *do-while*

```
/* Citirea unui sir de numere si calculul mediei lor */
#include <stdio.h>
#include <stdlib.h>
#define NUMELEM 100
int main()
{
   float a[NUMELEM], media, suma = 0;
   int i = 0, n = 0;
   do {
     printf("\nNumarul de elemente din sir [<%d], n=", NUMELEM);
     scanf("%d", &n);
   }
   while (n < 1 || n > NUMELEM);
   printf("\nIntroduceti elementele sirului\n");
   do {
     printf("a[%2d]=", i+1); scanf("%f", &a[i]);
     suma += a[i];
     i++;
   } while (i < n);
   media = suma / n;
   printf("Media=%g\n", media);
   return 0;
}
```

Instrucţiunea repetitivă *for*

- Forma generală

```
for (expresii_initiale; expresie_control; expresii_ajustare)
        instructiune
```

- Execută în mod repetat **instructiune** atâta timp cât **expresie_control** este evaluată la adevărat
- Efectul este acelaşi cu cel al unui ciclu **while** astfel:

```
expresii_initiale;
while ( expresie_control )
{
        instructiune
        expresii_ajustare;
}
```

- Observaţii
  - **expresii_initiale**, **expresie_control**, **expresii_ajustare** pot lipsi, dar caracterul **;** care le desparte trebuie să fie întotdeauna prezent
  - **expresii_initiale** se execută o singură dată
  - **expresii_ajustare** sau **instructiune** trebuie să modifice valoarea de adevăr a expresiei **expresie_control**, altfel apare fenomenul de ciclu infinit
  - Instrucţiunea **for** este folosită de obicei pentru cicluri cu număr cunoscut de iteraţii

Instrucţiunea repetitivă *for*

```
/* Citirea unui sir de numere si calculul mediei lor */
#include <stdio.h>
#define NUMELEM 100
int main() {
   float a[NUMELEM], media, suma;
   int i, n;
   for ( n = 0; n < 1 || n > NUMELEM; scanf("%d", &n) )
        printf("\nNumarul de elemente din sir [<%d], n=", NUMELEM);

   printf("\nIntroduceti elementele sirului\n");
   for ( i = 0, suma = 0; i < n; i++ )
   {
        printf("a[%2d]=", i+1); scanf("%f", &a[i]);
        suma += a[i];
   }
   media = suma / n;
   printf("Media=%g\n", media);
   return 0;
}
```

Instrucțiunile de salt *continue* și *break*

- Pot fi utilizate numai în interiorul unor instrucțiuni repetitive (în interiorul ciclurilor)
  - Excepție face instrucțiunea **break** care poate fi folosită și în instrucțiunile selective (**switch**) pentru a preveni fluxul de control să treacă la următorul **case**
- **continue** cauzează abandonarea iterației curente (instrucțiunile rămase) în cicluri și trecerea la următoarea iterație astfel:
  - În instrucțiunea **for**, **expresii_ajustare** sunt executate iar apoi **expresie_control** este re-evaluată cu scopul de a trece la iterația următoare
  - În instrucțiunile **while** și **do-while** se trece la re-evaluarea expresiei de control cu scopul de a trece la iterația următoare
- **break** cauzează terminarea imediată a ciclului (ieșirea din acesta), execuția programului continuând cu instrucțiunea imediat următoare de după instrucțiunea repetitivă

Instrucțiunea de salt *goto* și funcția *exit*

- Instrucțiunea **goto**
  - Este utilizată pentru mutarea fluxului de control al programului dintr-un punct al unei funcții într-un alt punct etichetat al acesteia
  - Eticheta se scrie ca numele unui identificator urmat de caracterul **:**

    eticheta:
  - Formatul instrucțiunii

    goto eticheta;
- Funcția **exit**
  - Asemănătoare unei instrucțiuni de salt
  - Prototipul ei este conținut în fișierul antet **stdlib.h**
  - Realizează terminarea imediată a programului și returnarea unui anumit cod de eroare
    - Zero înseamnă terminare normală, orice altă valoare înseamnă un cod de eroare definit de programator

Instrucțiuni de salt - exemplu

```
/* Insumarea tuturor numerelor
prime dintr-un sir de numere
intregi, pana la intalnirea
primului numar multiplu de 100 */
#include <stdio.h>
#include <math.h>
int main()
{
     int v[]={640,2,29,1,49,
              33,23,800,47,3};
     int i, suma=0;
     int nr=sizeof(v)/sizeof(int);
     for (i=0; i<nr; i++)
     {
          if (v[i]%100==0)
               goto afisare_suma;
          if (v[i]<2)
               continue;
          int prim=1;
          int k;
          double epsilon=0.001;
          int limit=
            (int)(sqrt(v[i])+epsilon);
          for (k=2; k<=limit; k++)
               if (v[i]%k==0) {
                    prim=0;
                    break;
               }
          if (prim)
               suma+=v[i];
     }
 afisare_suma:
     printf("Suma este %d",suma);
          /* Rezultat afisat: 54 */
     return 0;
}
```

Exemplu fără utilizarea instrucțiunilor de salt => programare structurată

```
/* Acceasi problema dar fara a
utiliza break, continue si goto */
#include <stdio.h>
#include <math.h>
int main()
{
    int v[]={640,2,29,1,49,
             33,23,800,47,3};
    int suma=0;
    int i=0;
    int nr=sizeof(v)/sizeof(int);
    while (i<nr && v[i]%100!=0)
    {
        if (v[i]>=2)
        {
            int prim=1;
            int k=2;
            double epsilon=0.001;
            int limit=
              (int)(sqrt(v[i])+epsilon);
            while (prim && k<=limit)
                {
                  if (v[i]%k==0)
                     prim=0;
                  k++;
                }
            if (prim)
                suma+=v[i];
        }
        i++;
    }
    printf("Suma este %d",suma);
        /* Rezultat afisat: 54 */
    return 0;
}
```