Programarea Calculatoarelor
Curs 8

Iulian Năstac

Recapitulare

Instrucţiuni
1. Introducere

- **Definiţie**: o instrucţiune reprezintă o porţiune a programului care poate fi executată.

- Aceasta specifică o acţiune de un anumit tip.

Recapitulare

Standardul ANSI C împarte instrucţiunile în următoarele grupe:

- *Selecţie* (**if** şi **switch** → se mai numesc şi *instrucţiuni condiţionale*)
- *Iterare* (**while**, **for** şi **do – while** → denumite uneori *instrucţiune de buclare*)
- *Salt* (**break**, **continue**, **goto**, **return**)
- *Etichetă* (**case** şi **default** (adiţionate la *switch*) şi **etichetele** (la *goto*))
- *Expresie* (instrucţiuni compuse dintr-o expresie validă)
- *Bloc* (sunt blocuri de cod sau instrucţiuni compuse; un bloc începe cu **{** şi se termină cu **}**)

2. Instrucţiuni de selecţie

- În cadrul instrucţiunilor de selecţie (sau condiţionale) întâlnim două tipuri distincte: instrucţiunea ***if ... else*** şi instrucţiunea ***switch***.
- Ca observaţie, în anumite condiţii, o alternativă a lui ***if*** este operatorul condiţional (***? :***).

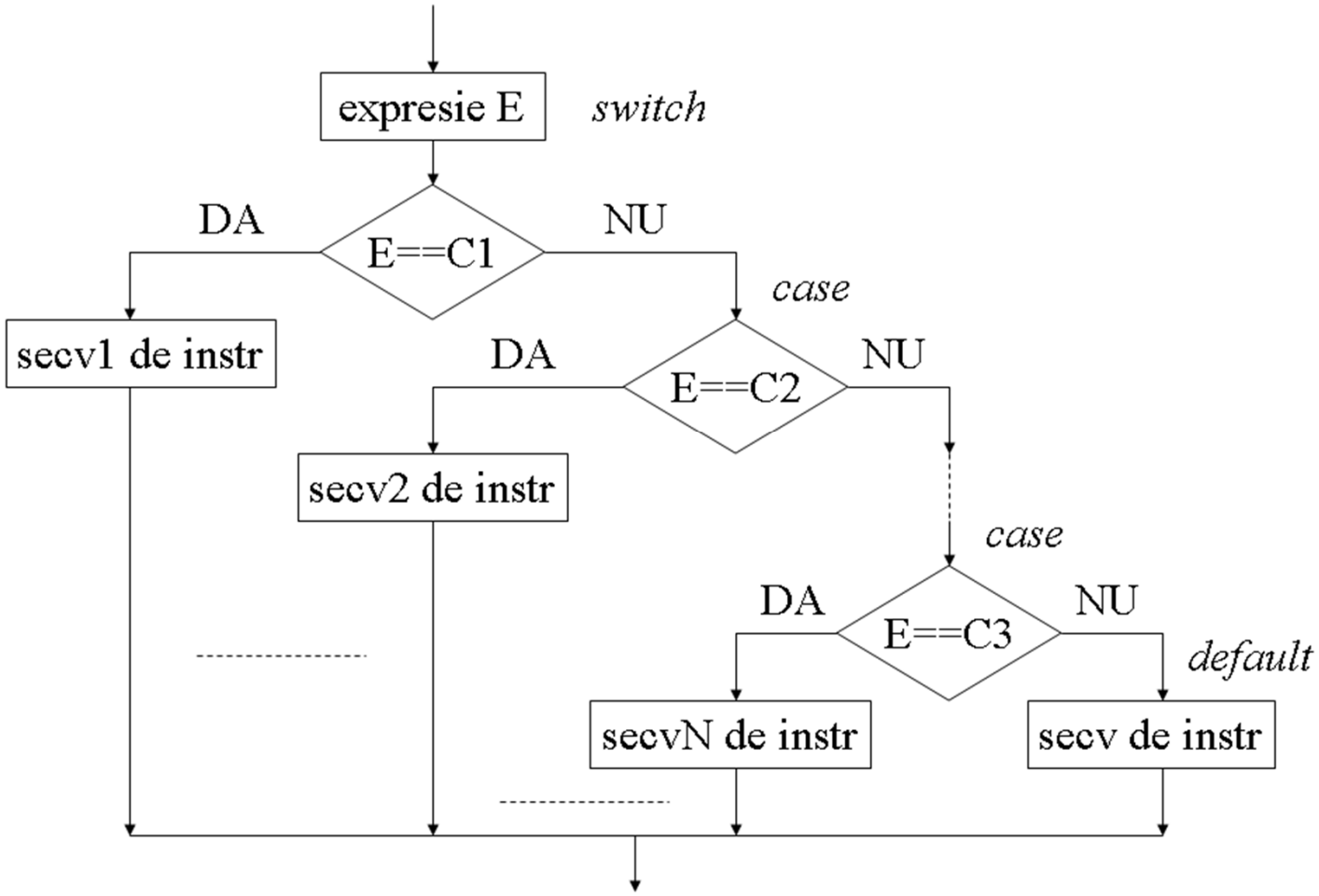
2.1.3. Alternativa condiţională (? :)

Se poate utiliza operatorul condiţional (*? :*) pentru a înlocui instrucţiunea *if-else* în maniera următoare:

if (condiţie) expresie1;
else expresie2; ⇔ *condiţie ? expresie1 : expresie2;*

Observaţie: În cazul acestei substituţii, subiectul atât pentru ***if*** cât şi pentru ***else*** trebuie să fie o singură expresie şi nu o altă instrucţiune.
Operatorul condiţional (**? :**) se mai numeşte şi *operator liniar* deoarece el are trei elemente asupra cărora operează.

O schemă logică pentru *switch*



Se observă asemănarea cu *scara if-else-if*, numai că spre deosebire de aceasta, *switch* se execută mai rapid.

3. Instrucţiuni de iterare

Definiţie: *instrucţiunile de iterare (bucle) permit ca un set de instrucţiuni să se execute repetat până se îndeplineşte o anumită condiţie.*

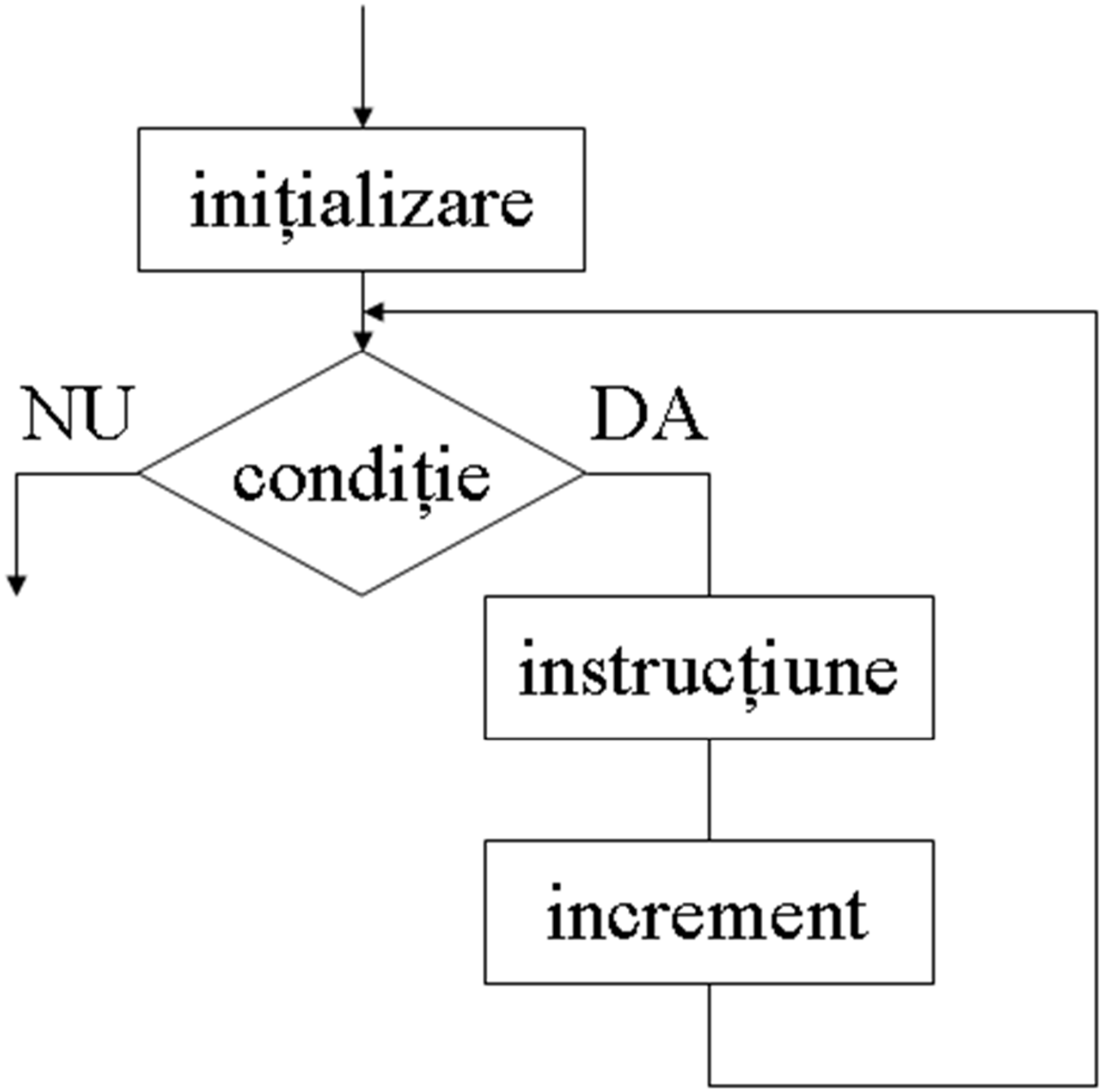
3.1. Bucla *for*

Este întâlnită în majoritatea limbajelor de programare. În *C* are o putere şi o flexibilitate maximă.

Format:

for (iniţializare; condiţie; increment) instrucţiune;

Recapitulare



3.2. Bucla *while*

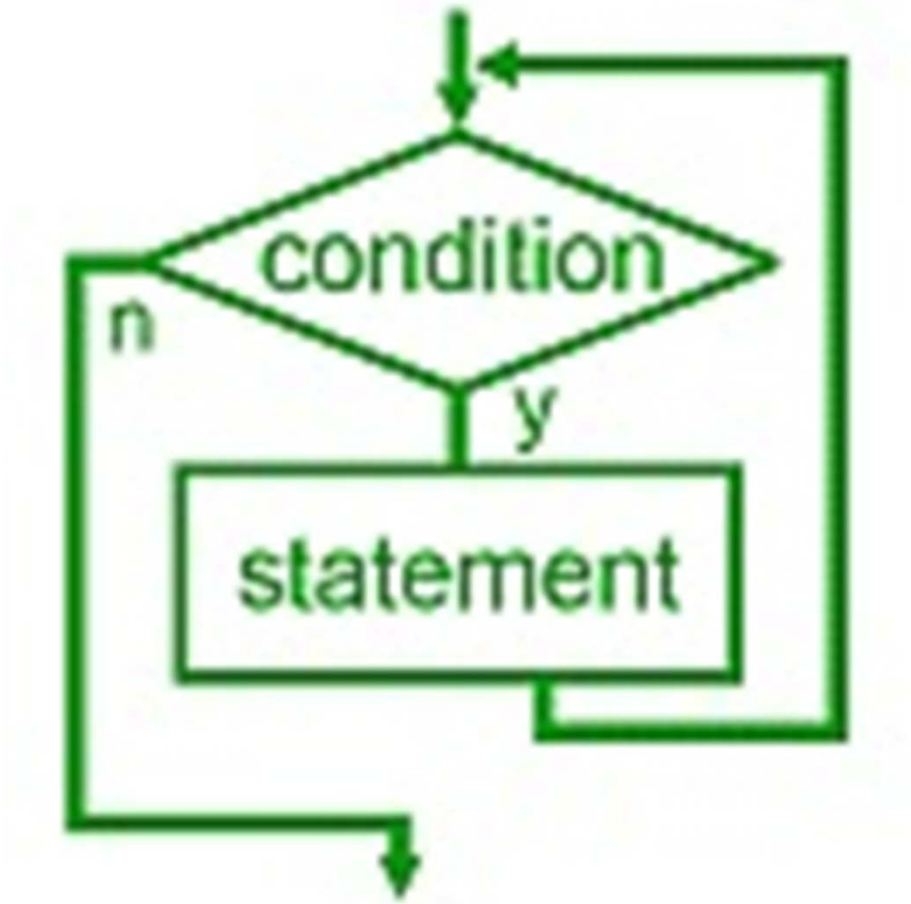
Format:

while(condiţie) instrucţiune;

unde:
- ***instrucţiune*** este o instrucţiune vidă, simplă sau un bloc de instrucţiuni.
- ***condiţie*** este orice expresie (va fi adevărată pentru valori ≠ 0).

Bucla ***while*** se reia atât timp cât condiţia testată este adevărată.

Schema bloc pentru instrucțiunea **while**



Echivalenţa *for* ⇔ *while*:

for (exp1; exp2; exp3) instrucţiune;

⇔

```
exp1;
while (exp2)
        { instrucţiune;
          exp3;
        }
```

Observaţii privind utilizarea lui *while*:

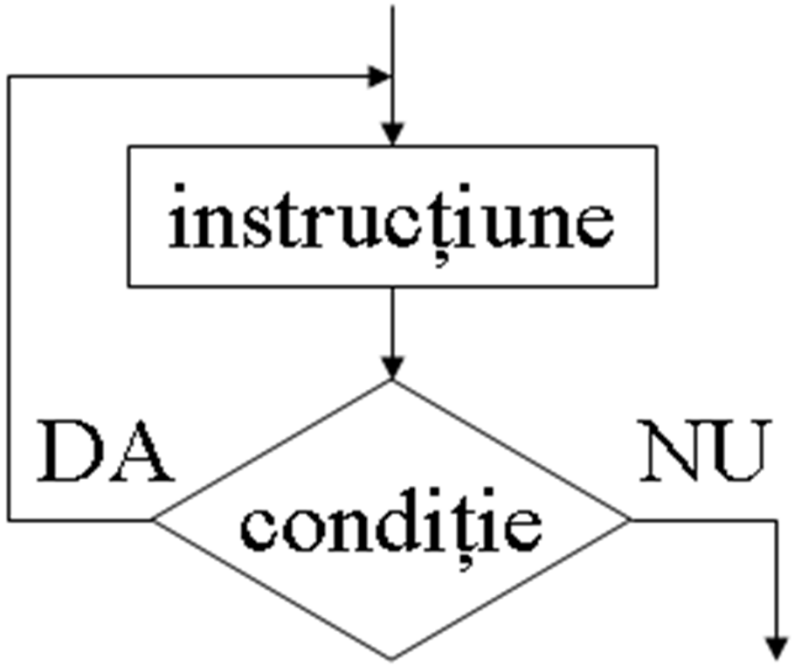
1) Condiţia este testată la început.

2) Dacă condiţia este iniţial falsă nu se pătrunde în buclă.

3.3. Bucla *do - while*

Format:

```
do
{ instrucțiune; // sau set instrucţiuni;
} while(condiţie);
```



Tema:

Rescrieți problema cu numere magice astfel încât programul să ceară reintroducere unui număr până la ghicirea celui ales de calculator prin funcția rand().

4. Instrucţiuni de salt

În cadrul instrucţiunilor de salt întâlnim:

- ***return*** → poate fi plasată oriunde în program.
- ***goto*** → poate fi plasată oriunde în program.
- ***break*** → în interiorul instrucţiunilor de buclare sau *switch*.
- ***continue*** → în interiorul instrucţiunilor de buclare.

4.1. Instrucţiunea *return*

Este folosită pentru revenirea dintr-o funcţie.

Format:

return expresie;

unde *expresie* este prezentă doar dacă funcţia este declarată ca returnând o valoare. Valoarea expresiei este convertită la tipul funcţiei.

Observaţie: O funcţie declarată cu ***void*** poate să nu conţină nici o instrucţiune ***return***.

4.2. Instrucţiunea *goto*

Format: ***goto etichetă;***

...

etichetă;

Observaţii:

- Este o instrucţiune evitată deoarece prin abuz se creează programe neportabile;
- Nu există situaţii în programare care să necesite imperios instrucţiunea *goto*. Eticheta este un specificator valid care trebuie să se găsească în aceeaşi funcţie ca şi *goto* care o utilizează. Nu se poate sări între funcţii.
- Eticheta poate fi plasată înainte sau după *goto*.

Exemplu:

```
x=1;
bucla 1;
        x++;
if (x<100) goto bucla1;
```

4.3. Instrucţiunea *break*

Are două utilizări:
- încheie un case dintr-un switch;
- determină ieşirea dintr-o buclă.

Exemplu:

```
...
        while(1)
                {
                   printf("\n\nMenu:\n");
                   /* se executa niste operatii dintr-un meniu...*/
                   …
                   printf ("\nDoriti sa parasiti acest meniu ?\n");
                   if ((c=getche())=='d') break;
                }
...
```

Observaţie: Un *break* determină ieşirea doar din bucla cea mai interioară.

Exemplu:
Se afişează numerele de la 1 la 10 de 100 de ori.

```
...
        int t, contor;
        for(t=0; t<100; t++)
                { contor=1;
                  for( ; ; )
                   { printf(" %d",contor);
                          contor++;
                          if(contor==11)
                            {
                              printf("\n");
                              break;
                            }
                    }
                }
...
```

Funcţia ***exit()***

Funcţia *exit()* se prezintă ca o instrucţiune de tip *break* generalizată.
Această funcţie determină ieşirea dintr-un program.
Format:

void exit(cod de întoarcere);

Observaţii referitoare la parametrul funcţiei *exit*:
este de tip *int*;
"0" este utilizat pentru a semnala terminarea normală a programului.
Funcţia *exit()* poate fi utilizată atunci când programul ia o turnură nefavorabilă.
Exemplu: Secvenţă dintr-un program care necesită un adaptor grafic special.

```
...
void main()
        { if(!grafica_virtuala()) exit(1);
        ... /* play */
        }
```

4.4. Instrucţiunea *continue*

Instrucţiunea ***continue*** se foloseşte numai în cadrul buclelor. Această instrucţiune forţează trecerea la următoarea iteraţie a buclei, ignorând restul codului din iteraţia curentă.

Efectul acestei instrucţiuni este următorul:

a. în interiorul buclei ***for***, ***continue*** determină trecerea la secvenţa de incrementare și apoi execuţia testului de condiţionare.

b. pentru buclele ***while*** şi ***do-while*** controlul programului este trecut testului de condiţionare.

Exemplul 1

Un program care afişează de la 1 la 100 toţi factorialii numerelor. Programul afişează succesiuni de 15 numere după care aşteaptă o tastă pentru afişarea următorilor 15 factoriali.

```
# include <stdio.h>
# include<conio.h>
int main()
{       int i,j,k;
        double f;
        for(i=1;i<=100;i++)
        { for(f=1.0, j=2;j<=i;j++)
                f*=j;
          printf("\n %d factorial este %g",i,f);
          k=i%15;
          if (k) continue;
          getch();
        }
        getch();
}
```

Exemplul 2

Codificarea unui mesaj schimbând toate caracterele care se tastează cu litera următoare în cod ASCII (de exemplu A devine B). Programul se opreşte când se tastează *$*.

```
...
        char gata,ch;
        …
        gata=0;
        while (!gata)
        {  ch=getch();
           if(ch=='$')
                  { gata=1;
                    continue;
                  }
        putchar(ch+1);
        }
...
```

5. Instrucţiuni etichetă

Acestea reprezintă etichete valide întâlnite pe parcursul derulării unui program.

În limbajul C, instrucţiunile etichetă sunt de două tipuri:

-***case*** şi ***default*** - discutate în cadrul instrucţiunii *switch*;

- etichetele - discutate în cadrul instrucţiunii *goto*.

6. Instrucţiuni de tip expresie

O *instrucţiune de tip expresie* reprezintă orice expresie validă urmată de punct şi virgulă (**;**). În această categorie intră şi *instrucţiunea vidă*.

Instrucţiunea vidă se reduce la caracterul punct şi virgulă (**;**). Ea nu are nici un efect, însă se utilizează frecvent în cadrul anumitor structuri alternative şi repetitive.

7. Instrucţiuni bloc

O instrucţiune bloc (sau compusă) reprezintă o succesiune de instrucţiuni (de diferite naturi) incluse între acolade.

Format:

```
{
   declaraţii;
   succesiune de instrucţiuni
}
```

Instrucţiunile bloc sunt grupuri de instrucţiuni care sunt tratate ca o unitate.
Observaţii:

Orice *switch* este urmat de o instrucţiune bloc.
Cel mai frecvent se folosesc instrucţiunile bloc pentru a crea o instrucţiune multiplă ca obiect al unei alte instrucţiuni.

Cap. Programarea procedurală

Introducere

Programarea procedurală este o caracteristică a limbajelor de programare de nivel înalt. Dacă se doreşte executarea unui anumit set de instrucţiuni, cu date diferite sau în locuri diferite, acestea se grupează într-o subrutină care va fi apelată printr-un salt ori de câte ori este nevoie. Acest salt este cu revenire la instrucţiunea următoare celei din care s-a făcut saltul şi de aceea el diferă de salturile realizate cu instrucţiunea *goto.*

Acest tip de secvenţă cu organizarea de mai sus poartă diverse denumiri în cadrul limbajelor de programare: *subprogram*, *subrutină*, *procedură*, *funcţie*, etc.

În principiu toate limbajele de programare conţin două tipuri (sau categorii) de proceduri:

a) proceduri care definesc o valoare de revenire;

b) proceduri care nu definesc o valoare de revenire.

Observaţii:

- Procedurile din categoria a) se numesc în general funcţii;
- În limbajul C toate subrutinele sunt denumite funcţii indiferent dacă întorc sau nu o anumită valoare (există funcţii declarate ca fiind de tip *void,* adică nu întorc nici o valoare).
- Limbajul C livrează o serie de funcţii care au o utilizare frecventă în programare. Ele se păstrează într-un fişier special (în format *obj,* adică compilat) care se adaugă în faza de editare. Acestea sunt funcţiile standard de bibliotecă (fişiere de extensie *h)*. De exemplu funcţiile de tip intrare/ieşire (*printf()*, *scanf()*) se găsesc în *stdio.h*, funcţiile matematice elementare (*sqrt*, *sin*, *cos*, *atan*, *log*) în *math.h*, etc. Prototipurile funcţiilor standard se includ în program înaintea apelurilor lor prin directiva *#include*.

În limbajul C folosirea funcţiilor în cadrul unui program implică:

- declararea funcţiilor;

- definirea funcţiilor.

1. Declararea funcţiilor

În declararea funcţiilor se pot folosi două stiluri:

a) stilul clasic: specifică doar numele funcţiei şi tipul valorii returnate:

tip nume_funcţie() ;

Nu există informaţii asupra parametrilor, astfel încât nu se pot face verificări de eroare.

b) stilul modern: se introduc şi informaţii despre parametrii funcţiei:

tip nume_funcţie(inf_p1, inf_p2, ... etc.);

unde *inf_p* declară *tipul* şi *numele variabilei* folosită ca argument al funcţiei.

2. Definirea funcţiilor

a) Stilul clasic

```
tip nume_funcţie(nume_parametrii)
        definiţii parametrii
        {
        ...
        }
```

b) Stilul modern

```
tip nume_funcţie(inf_p, inf_p, ...)
        {
        ...
        }
```

Prin *inf_p* se declară *tipul* şi *numele* fiecărui parametru.

Observaţii:

1. Cele două stiluri definesc etape de dezvoltare ale limbajului C. Majoritatea compilatoarelor de astăzi nu utilizează decât programe scrise în stilul modern. Programele scrise în stilul clasic pot fi uşor corectate .

2. În cazul definirii funcţiilor, prima linie va fi identică cu prototipul funcţiei (de la declarare) cu observaţia că la definire nu apare caracterul punct şi virgulă (;).

3. În stadiul actual putem spune că logica scrierii unui program în "***C***" este următoarea:

- includerile de fişiere header;
- declaraţiile de variabile şi constante globale;
- declaraţiile de funcţii;
- definirea funcţiei *main*;
- definirea tuturor funcţiilor declarate.

Observaţii (continuare):

4. Prin declararea funcţiilor (a prototipurilor), compilatorul face verificări preliminare asupra parametrilor funcţiei.

5. Ordinea de declarare a funcţiilor se face în ordinea apelării lor de către program.

6. Se poate urma şi logica din Pascal, renunţând la declararea funcţiilor în cazul în care definirea funcţiilor se face înainte de definirea funcţiei *main,* şi fără a se mai face declararea.

3. Sfera de influenţă a funcţiilor

Definiţie: *sfera de influenţă* a unui limbaj înseamnă totalitatea regulilor ce stabilesc modul în care o secvenţă de cod influenţează sau are acces la o altă secvenţă de cod sau de date.

În limbajul C, în cadrul *sferei de influenţă* sunt strict respectate următoarele **principii**:

1. Fiecare funcţie conţine un bloc de cod discret propriu acelei funcţii şi nici o instrucţiune din altă funcţie nu poate să aibă acces la el decât printr-un apel al funcţiei. Nu se poate folosi ***goto*** pentru salt între funcţii.

2. Dacă o funcţie nu utilizează variabile sau date globale ea nu poate afecta alte părţi ale programului. Codul şi datele dintr-o funcţie nu pot interacţiona direct cu codul şi datele din altă funcție.

3. În *C toate funcţiile au acelaşi nivel al sferei de influenţă.* C nu este tehnic vorbind un limbaj structurat în blocuri, deoarece nu poate fi definită o funcţie în altă funcţie (dar o funcţie o poate apela pe alta).

Principii (continuare)

4. Variabilele care sunt definite într-o funcţie sunt numite *variabile locale*. O variabilă locală este creată atunci când se intră în acea funcţie şi este distrusă la ieşire. Deci variabilele locale nu îşi păstrează valoarea între apelările funcţiei. Singura excepţie sunt variabilele declarate cu *static* care nu se distrug la părăsirea funcţiei dar sunt limitate ca sferă de influenţă la interiorul funcţiei.

Principii (continuare)

5. Vrem să apelăm o funcţie dintr-o altă funcţie printr-un argument ce este o variabilă.

Exemplu:

```
tip f1 (    )
{ tip x;
   ...
   f2(x);
   ...
}
```

Funcţiei *f2* îi este transmisă o copie a valorii argumentului. Ceea ce se întâmplă în interiorul funcţiei *f2* nu are efect asupra variabilei folosite pentru apelare (adică *x* din funcţia *f1*).

6. Funcţii diferite pot avea variabile locale cu nume identice ceea ce nu duce la influenţarea reciprocă a funcţiilor.

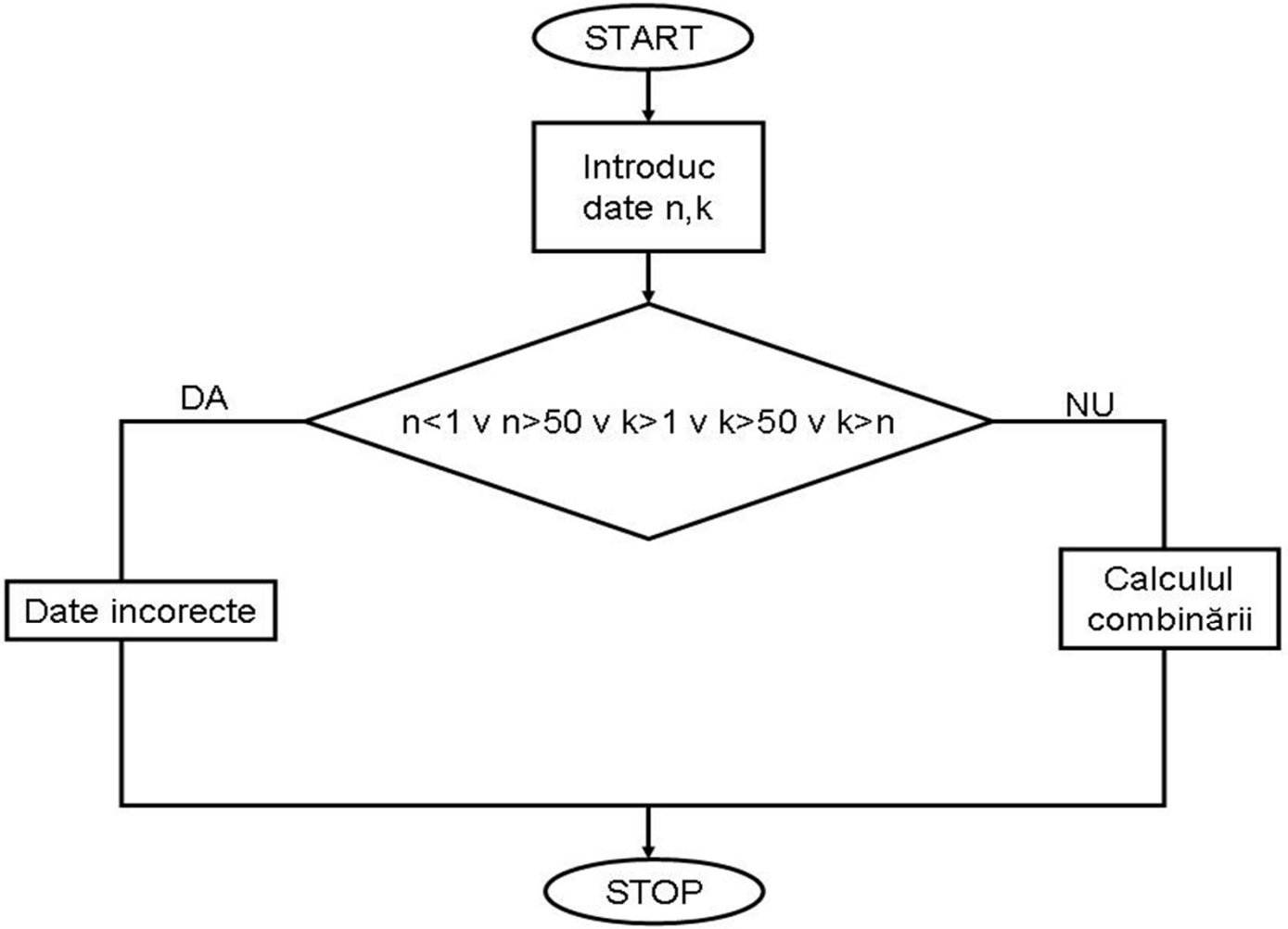
Exemplu:

Vom scrie un program care cere introducerea variabilelor n şi k de tip întreg de la tastatură şi verifică dacă acestea aparţin intervalului [1,50]. Programul calculează şi furnizează rezultatul:

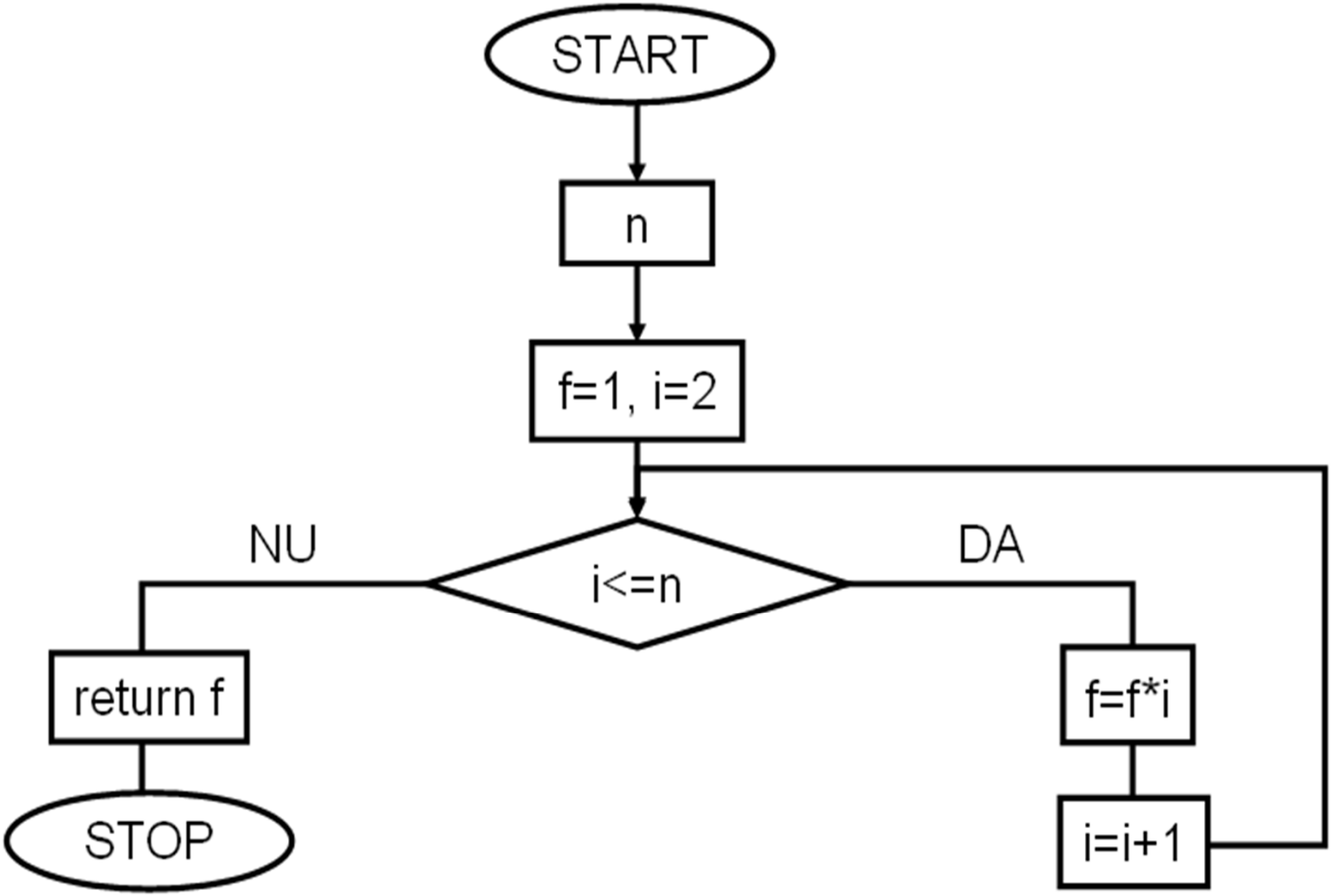
$$C_n^k = \frac{n!}{k!(n-k)!}$$

Se impune condiţia k<n.

Schema logică a unui program detaliază, în general, doar funcţia *main*.



Schema logică a funcţiei *fact* este:



```
#include <stdio.h>
double fact(int n);
int main()
{ long int k,n; /* pentru conversia rezultatului la long int */
  printf("\n introduceti pe n=");
  scanf("%d",&n);
  printf("\n introduceti pe k=");
  scanf("%d",&k);
  if (n<1||n>50||k<1||k>50||k>n)
          printf("\n Date incorecte");
  else printf("\n Rezultatul = %g", fact(n)/(fact(k)*fact(n-k)));
}
double fact(int n)
{ double f;
  int i;
  for (i=2,f=1.0;i<=n;i++)
          f*=i;
  return f;
}
```

4. Apelarea prin argumentele unei funcţii

În limbajul C apelarea unei funcţii se poate face (relativ la natura argumentelor sale) în două feluri:

- prin valoare;

- prin referinţă.