Programarea Calculatoarelor
Curs 7 - Instrucțiuni

Iulian Năstac

Recapitulare

Operatori în C

- Limbajele de programare utilizează un set de operatori cu o sintaxă proprie.

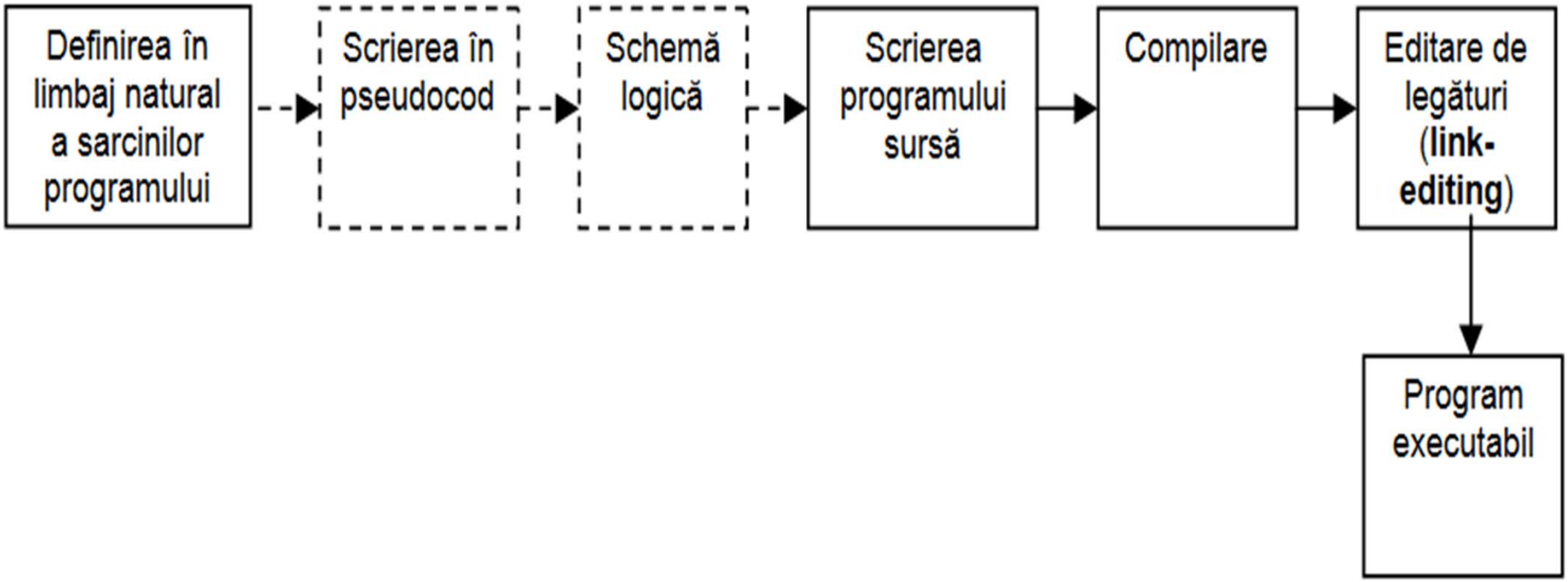
- Limbajul de programare C conține un număr fix de operatori predefiniți.

Recapitulare

1	**() [] -> . ::**	grupare, matrici, acces
2	**! ~ - + * & sizeof *type cast* ++x - -x**	Operatori unari, sizeof și casts
3	*** / %**	Multiplicare, împărțire, modulo
4	**+ -**	Adunare și scădere
5	**<< >>**	Deplasare stânga și dreapta
6	**< <= > >=**	Comparație inegalitate
7	**== !=**	Comparație egalitate
8	**&**	ȘI logic (pe biți)
9	**^**	SAU exclusiv (pe biți)
10	**|**	SAU (pe biți)
11	**&&**	ȘI logic
12	**||**	SAU logic
13	**? : = += -= *= /= %= &= |= ^= <<= >>=**	Expresia condițională și operatorii de atribuire
14	**,**	Operatorul virgulă

Recapitulare

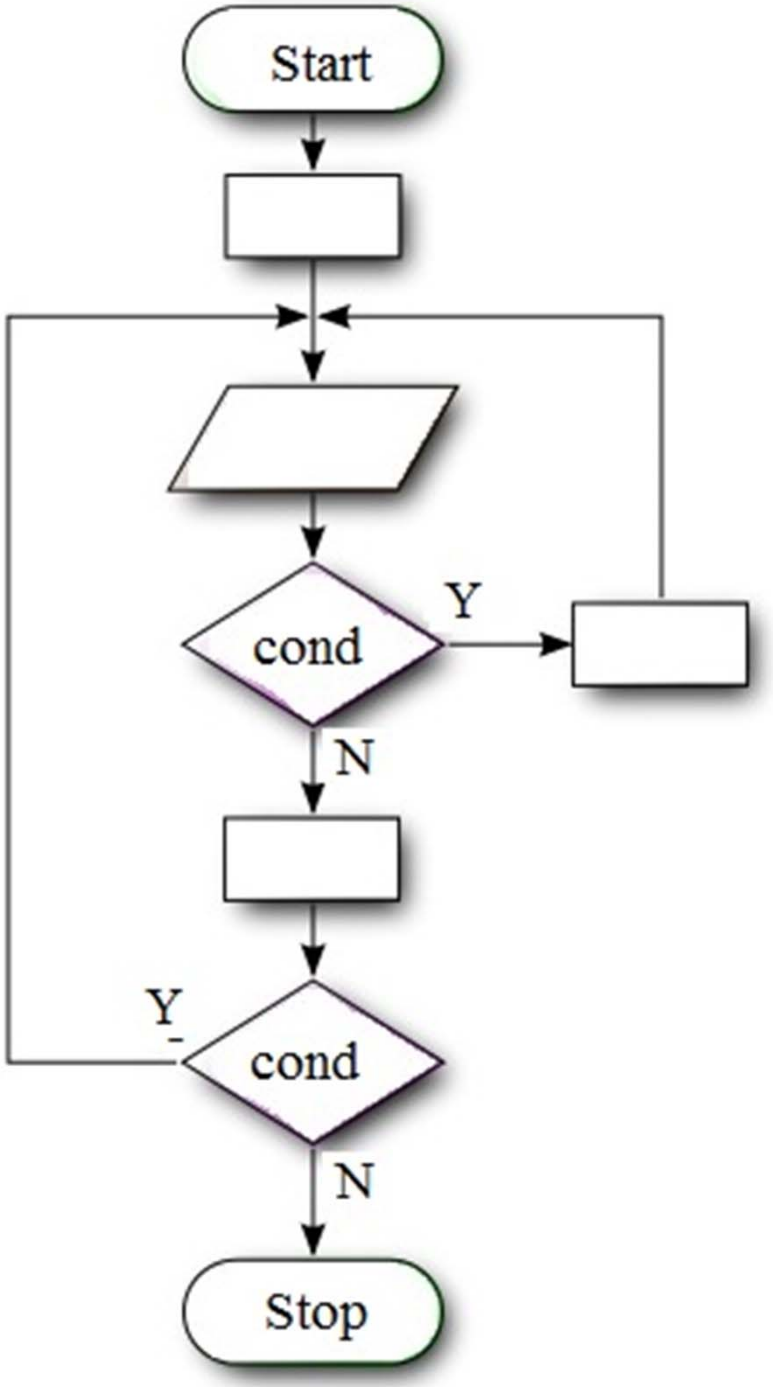
În funcţie de dimensiunile unui program acesta poate parcurge mai multe etape:



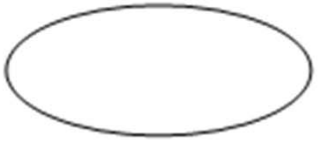
Recapitulare

Diagrame logice (Scheme logice)

- Schemele logice (sau bloc) sunt scheme organizate pe blocuri astfel încât să fie respectat principiul de secvenţialitate al unui program.
- *Principiul de secvenţialitate (Von Neumann)*: Trecerea unui program într-o altă etapă se poate face prin terminarea celei precedente sau printr-o condiţie impusă de aceasta.
- Nu se pot desfăşura două etape simultan într-un sistem de calcul secvenţial.



Formate grafice întâlnite în cadrul unei scheme logice:



← alocate proceselor de - început (START)
- sfârşit (STOP)

← leagă blocurile între ele

← bloc de operaţie (calcul) poate conţine o instrucţiune, un grup de instrucţiuni, un bloc de instrucţiuni, o buclă, o funcţie, fişier de funcţii etc.

← bloc de decizie

Instrucţiuni
1. Introducere

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

- Aceasta specifică o acţiune de un anumit tip.

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 **}**)

Adevărat şi fals în C

- Multe instrucţiuni în C se bazează pe o expresie de condiţionare care determină cursul acţiunilor următoare. O expresie condiţională este evaluată ca adevărată sau falsă.
- În C, spre deosebire de alte limbaje, este adevărată orice valoare diferită de zero (inclusiv numerele negative). O valoare falsă este 0. Conceptul de adevărat şi fals permite o mare varietate de rutine care pot fi codate eficient.

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 (***? :***).

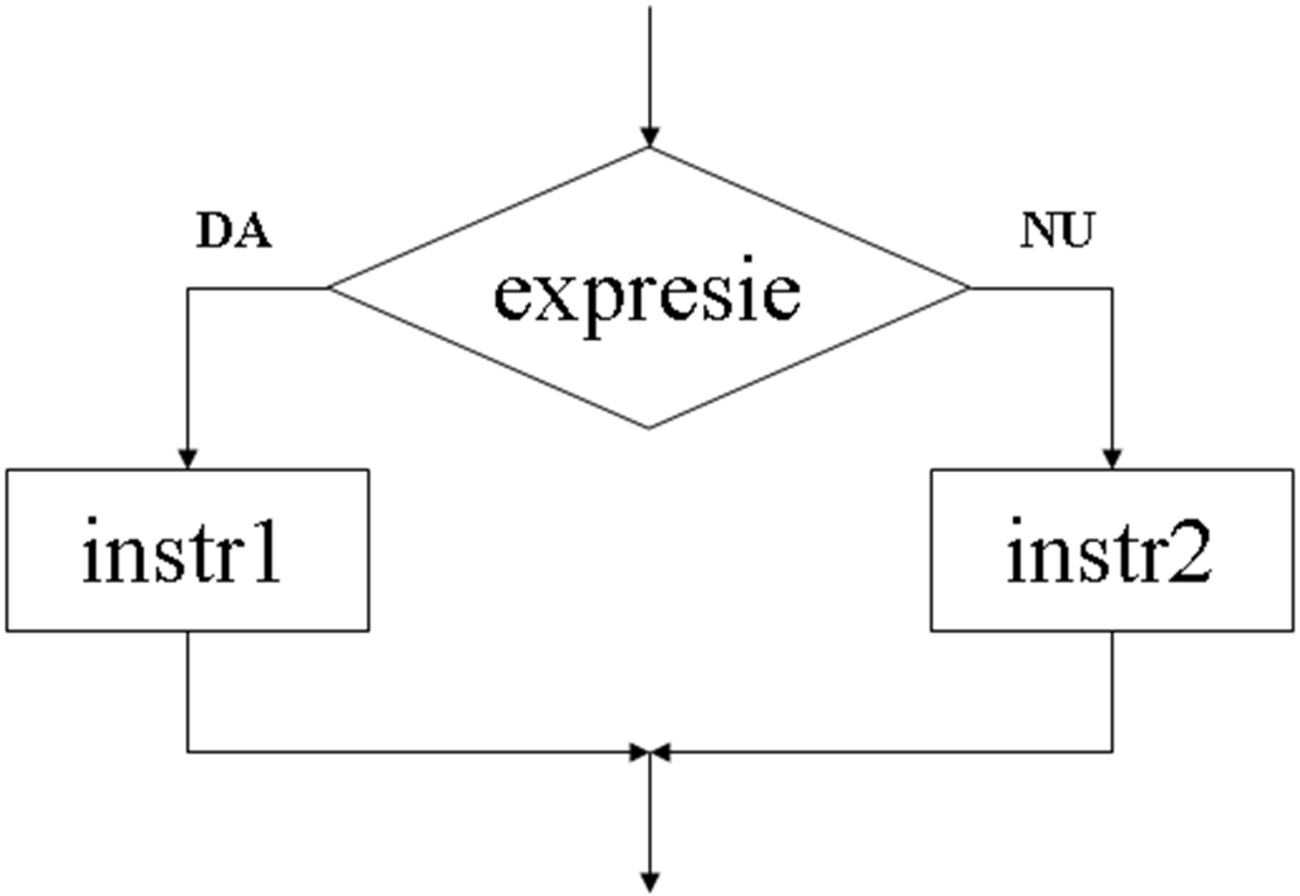
2.1. Instrucţiunea *if*

Forma generală:

if (expresie) instrucţiune1;

else instrucţiune2;

Schema bloc



Observaţii:

1) Nu se execută niciodată amândouă instrucţiunile 1 şi 2, ci numai una dintre ele. Dacă se execută amândouă înseamnă că s-a strecurat un ***";"*** unde nu trebuie (după ***condiție*** sau după ***else***).

2) ***Expresia de condiţionare*** care controlează pe ***if*** va determina un rezultat scalar (*întreg*, *caracter*, *pointer* sau *în virgulă mobilă* → ultimul mai rar folosit în astfel de scopuri).

Exemplu:

Versiunea simplă a jocului *“ghiceşte numărul magic”*.

Se afişează *corect* dacă jucătorul ghiceşte numărul magic. Pentru alegerea numărului magic se utilizează o funcţie *rand()* de generare de numere aleatoare care returnează un număr arbitrar din intervalul [0, RAND_MAX] unde RAND_MAX = 32.767 sau chiar mai mare.

rand() se găseşte în fişierul header *stdlib.h*.

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

int main ()
{ int magic;                    /* număr magic */
  int ghici;                    /* număr jucător */
  magic = rand();               /* se generează numărul magic */
  printf ("\n Ghiceşte numărul magic: ");
  scanf ("%d", &ghici);
  if (ghici==magic) printf ("\n Corect");
  else printf ("Greşit");                  /* poate lipsi */
  getch();
}
```

2.1.1. Instrucţiunea *if* imbricat

- *Un **if imbricat** este un **if** care este obiectul unui alt **if** sau al unui **else**.*

- Observaţii:
  - în programare *if imbricat* este foarte uzual.
  - un *else* se leagă întotdeauna de ultimul *if* întâlnit din acelaşi bloc.

Ex:
```
if (i)  { if (i) instr1;
            if (k) instr2;
            else instr3;
          }
        else instr4;
```

Observaţie: Standardul ANSI C permite un număr de cel puţin 15 nivele de imbricare. În practică, majoritatea compilatoarelor permit mult mai multe. Însă imbricarea excesivă îngreunează înţelegerea algoritmului.

Exemplu: Extindem programul precedent cu un nivel de imbricare

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

int main()
{
  int magic;                    /* număr magic */
  int ghici;                    /* număr jucător */
  magic = rand();               /* se generează numărul magic */
  printf ("\n Ghiceşte numărul magic: ");
  scanf ("%d", &ghici);
  if (ghici==magic) printf ("\n Corect");
  else    { printf ("Greşit. ");
            if (ghici>magic) printf("Număr prea mare \n");
            else printf ("Număr prea mic \n");
          }
}
```

2.1.2. Scara *if – else – if*

Scara *if – else – if* este o construcţie uzuală în cazul unei selecţii multiple.

Forma sa generală este:

```
if (exp1) instr1;
else   if (exp2) instr2;
       else   if (exp3) instr3;
              else instr4;
```

Observaţie: Un ***else*** se leagă de ultimul ***if*** întâlnit.

Exemplu:
Simplificarea programului precedent

```
# include <stdio.h>
# include <stdlib.h>
void main ()
{
  int magic;                    /* număr magic */
  int ghici;                    /* număr jucător */
  magic = rand();               /* se generează numărul magic */
  printf ("\n Ghiceşte numărul magic: ");
  scanf ("%d", &ghici);
  if (ghici==magic) printf ("\n Corect");
  else   if (ghici>magic) printf("prea mare \n");
         else printf ("prea mic \n");
}
```

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.

Exemple:

1) *if (x>9) y=100;*
else y=200; ⇔ *y=x>9 ? 100:200;*

2) Se pot folosi în loc de expresii şi funcţii care nu sunt de tip *void* (care întorc expresii):
t ? f1(t)+f2(t) : printf("\n s-a introdus 0");

Programul cu numere magice:

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

 int main()
{ int magic;
 int ghici;
 magic = rand();
 printf ("\n Ghiceşte numărul magic: ");
 scanf ("%d", &ghici);
 if (ghici==magic) printf ("\n Corect");
 else (ghici>magic) ? printf("prea mare\n") : printf("prea mic\n");
}
```

2.2. Instrucţiunea *switch*

Este o instrucţiune de condiţionare multiramură care testează succesiv valoarea unei expresii faţă de o listă de constante de tip *caracter* sau *întreg*.

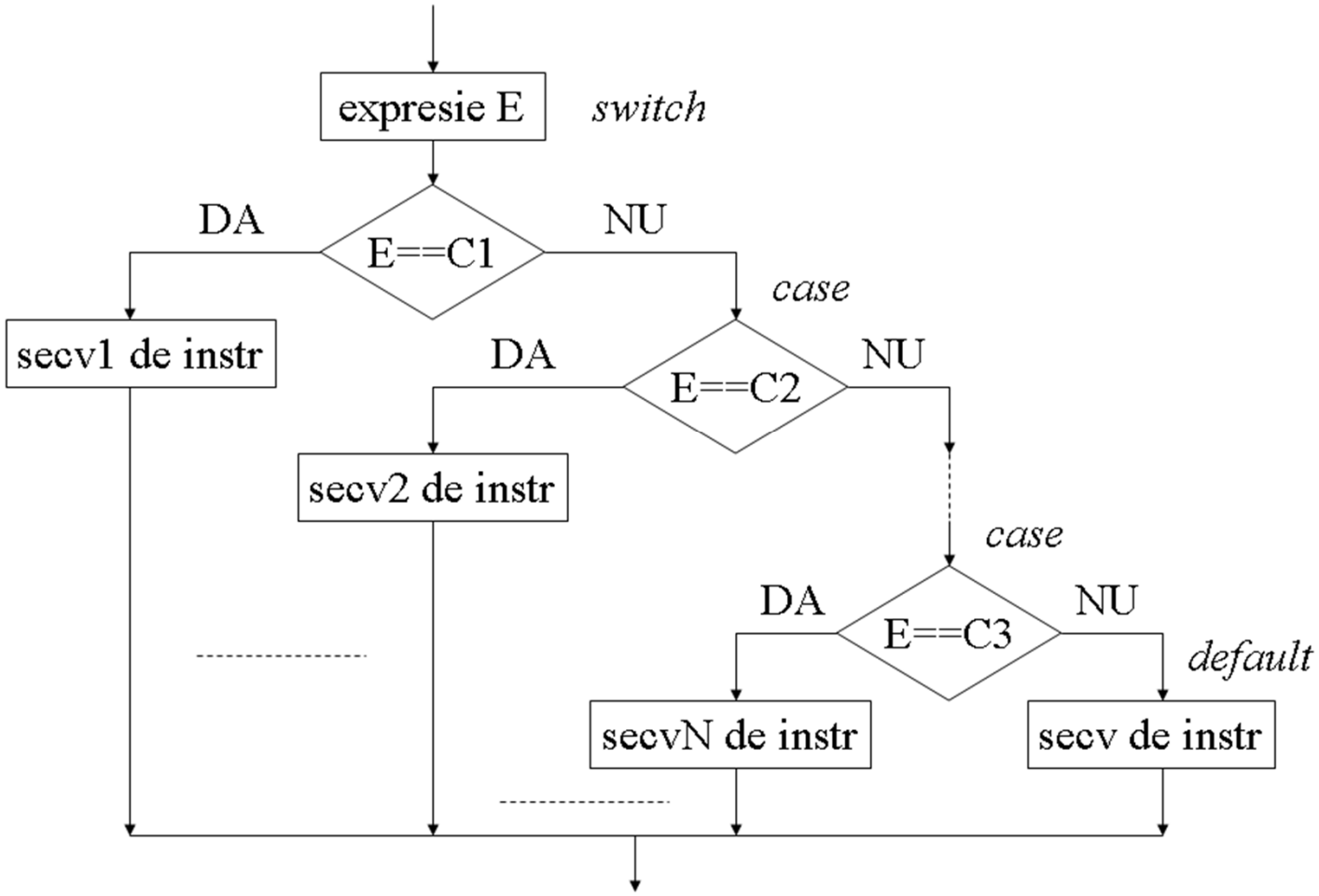
Forma generală:

```
switch (expresie)
        { case const1: secvenţă de instrucţiuni;
                break;
          case const2: secvenţă de instrucţiuni;
                break;
          …
          case constN: secvenţă de instrucţiuni;
                break;
          default: secvenţă de instrucţiuni;
        }
```

Observaţii:

1) Prezenţa instrucţiunii ***break*** (instrucţiune de salt) în cadrul fiecărei secvenţe *case* este importantă. Lipsa ei (funcţiei de compilator) duce la evoluţii anormale ale programului.
2) ***default*** este opţional.
3) Standardul ANSI C stipulează: compilatoarele de ***C*** să permită pentru o instrucţiune ***switch*** cel mult 257 instrucţiuni de tip *case*. Practic însă nu există programe care să folosească atât de multe *case*-uri.
Standardul propus pentru ANSI C++ recomandă un număr maxim de până la 16.384 de instrucţiuni ***case*** pentru un ***switch***.
4) Deşi ***case*** este o instrucţiune etichetă, ea nu poate exista de una singură, în afara instrucţiunii ***switch***.
5) Instrucţiunea ***switch*** este deseori folosită pentru a prelucra comenzile de la tastatură (cum ar fi selecţia dintr-un meniu).

O schemă logică pentru ***switch***



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

Exemplu: O secvenţă de program care adună sau scade două numere pentru + respectiv –. Nu se efectuează nici o operaţie la tastarea * sau / .

```
...
float x, y;
char op;
…
scanf("%f %f", &x, &y);
op=getch();
switch(op);
{ case '+': printf("\n Rezultat adunare: %f \n", x+y);
            break;
  case '-': printf("\n Rezultat scădere: %f \n", x-y);
            break;
  case '*':
  case '/': printf("\n Nu s-a efectuat operaţia \n");
            break;
  default: printf("\n Eroare \n");
 }
…
```

Observaţii:

1) Pot exista instrucţiuni ***case*** care nu au asociate secvenţe de instrucţiuni. Când apare această situaţie, execuţia trece la următorul ***case***.
2) Execuţia unei instrucţiuni continuă cu următorul ***case*** dacă nu este prezentă instrucţiunea ***break***. Aceasta din urmă previne repetarea inutilă a instrucţiunilor rezultând un cod mai eficient.

2.2.1. Instrucţiunea *switch* imbricată

În cadrul unor programe poate exista un *switch* inclus într-o secvenţă de instrucţiune a unui alt *switch* exterior. Chiar dacă unele constante *case* din *switch*-ul interior şi din cel exterior conţin valori comune, nu apar conflicte.

```
...
switch(x)
  { case 1: switch(y)
                { case 0: printf("mesaj1");
                          break;
                  case 1: procesat(x,y);
                }
        break;
    case 2: …;
  }
```

3. Instrucţiuni de iterare

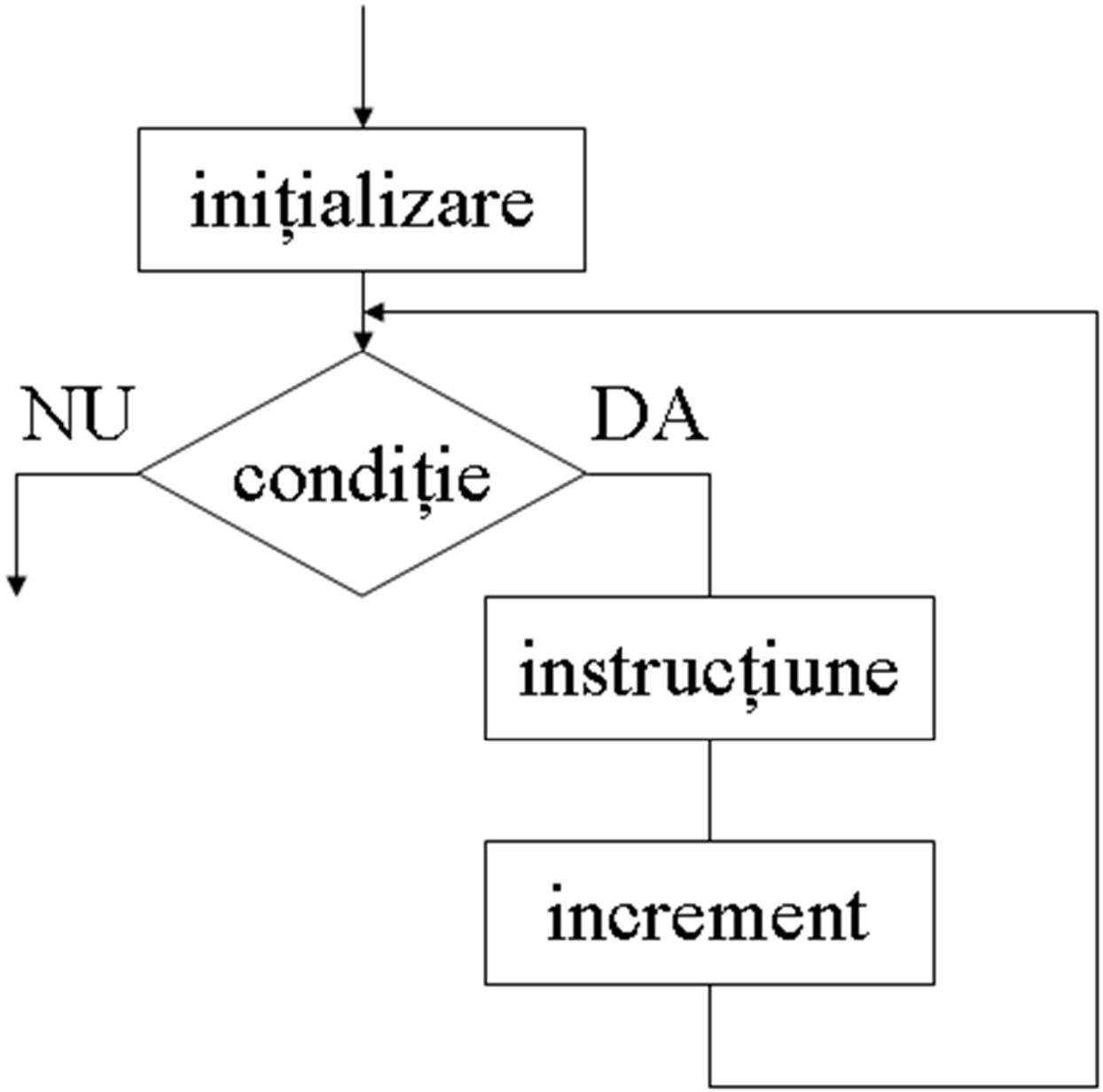
Definiţie: *instrucţiunile de iterare (bucle) permit ca un set de instrucţiuni să se execute repetat atât timp cât se îndeplineşte o anumită condiţie.*

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;


inițializare
NU
condiție
DA
instrucțiune
increment

Observaţii:

1) Atât ***iniţializarea***, cât şi ***condiţie*** şi ***increment*** sunt opţionale.

2) ***Instrucţiune*** poate fi instrucţiune nulă, simplă sau multiplă.

Exemplu:

```
for (x=100; x!=65; x-=5)
                { y=x*x;
                  printf(“ \n Patratul lui %d, este %f ”, x, y);
                }
```

3.1.1. Variaţii ale buclei *for*

Variaţii de la bucla *for* standard se pot obţine folosind operatorul virgulă.

Exemple:

1)
```
...
        for (x=0, y=0 ;  x+y<10 ; ++x)
        { … }
```

2)
```
...
for (i=1, j=rand() ;  i<j ;  i++, j--)
        { … }
```

3) Utilizarea pentru accesul într-un program cu parolă. Utilizatorul poate încerca de maximum trei ori să introducă parola.

```
...
char sir[20];
int x;
…
for (x=0; x<3 && strcmp(sir, “parola”); x++)
        { printf(“\n Introduceti parola:”);
          gets(sir);
        }
if(x==3) exit(1); /* altfel se continua programul */
…
}
```

4) Unele părţi din definiţia generală pot lipsi.

```
...
for (x=0; x!=123; ) scanf(“%d”, &x);
```


În acest ultim exemplu dacă se tastează 123 se iese din buclă.

3.1.2. Bucla *for* infinită

Format: *for(; ;)* ***instructiune****;*

Exemplu:
for(; ;) printf(“Această bucla va rula la infinit \n”);

Observaţii:
1) Când expresia de condiţionare este absentă, se presupune că este adevărată.
2) Construcţia *for(; ;)* nu garantează neapărat o adevărată buclă infinită deoarece instrucţiunea *break* folosită în interiorul său determină încheierea imediată a acesteia.

```
...
ch='\0';
for( ; ; )
        { ch=getchar();
          if(ch=='A') break;
        }/* bucla se poate repeta la indefinit până când
                utilizatorul va scrie A de la tastatură */
...
```

3.1.3. Bucla *for* fără corp

Se poate folosi pentru mărirea eficienţei unor algoritmi sau pentru a crea bucle de întârziere.

Exemplu: bucla de întârziere:

```
...
for (t=0; t<N; t++);
...
```

3.2. Bucla *while*

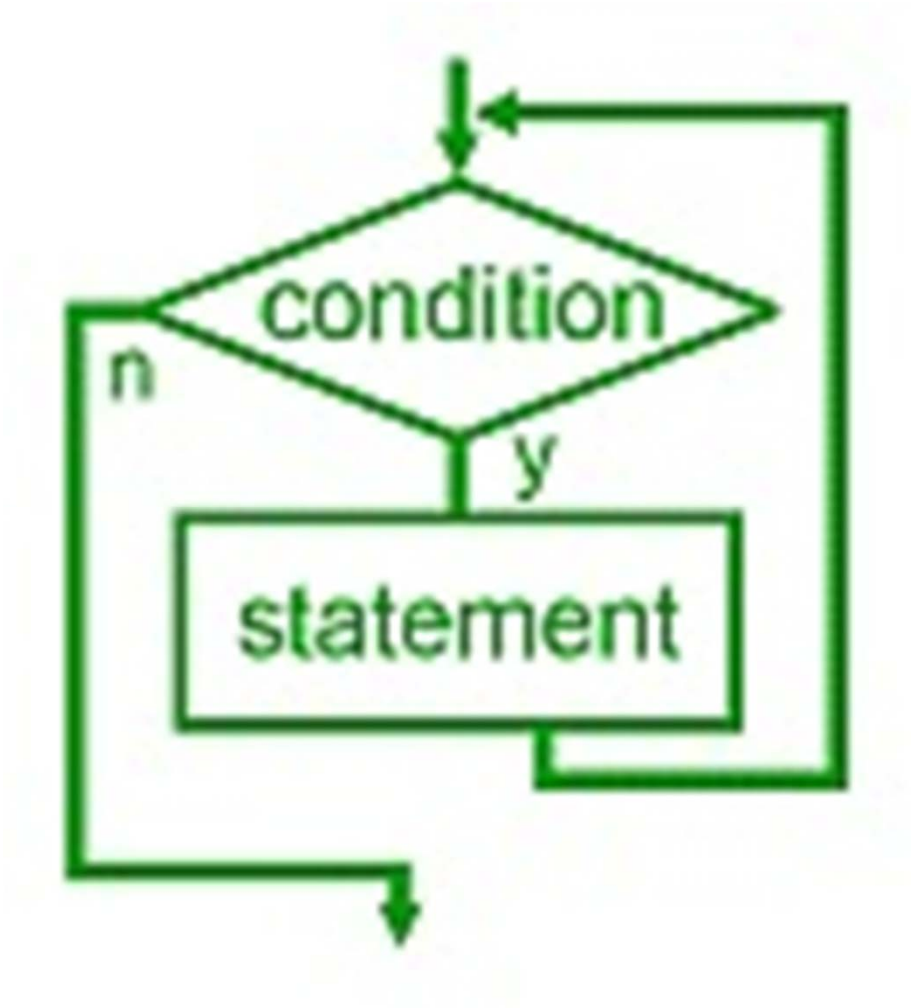
Format:

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.

Exemple:

1)
```
char ch;
ch='\0';
…
while (ch!='A') ch=getchar();
…
```

Bucla se repetă până când utilizatorul introduce ‘A’ de la tastatură.

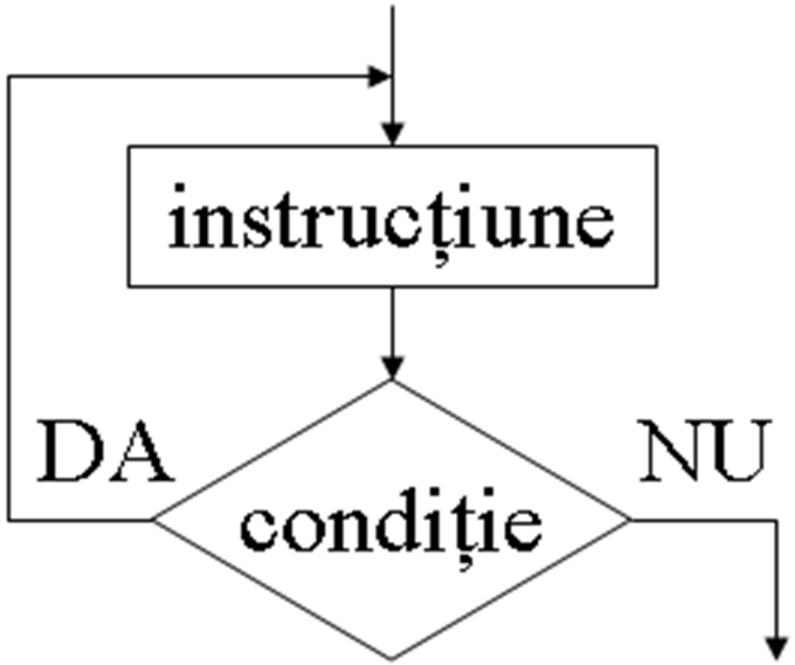
2) Nu este necesar să existe vreo instrucţiune în cazul buclei *while*:

```
while ((ch=getchar()) != 'A');
```

3.3. Bucla *do - while*

Format:

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



Spre deosebire de *for* şi *while*, bucla *do-while* verifică condiţia la sfârşit. În consecinţă, bucla *do-while* se execută cel puţin o dată. Bucla se repetă până ce condiţia devine falsă.

Exemplu: selectare dintr-un meniu

```
...
char ch;
…
printf("\n 1. Opţiunea 1 ");
printf("\n 2. Opţiunea 2 ");
printf("\n 3. Opţiunea 3 ");
printf("\n Introduceţi o opţiune ");
do   { ch=getchar();
       switch(ch);
          { case '1': <secvenţa 1>; break;
            case '2': <secvenţa 2>; break;
            case '3': <secvenţa 3>; break;
          }
     } while (ch!='1' && ch!='2' && ch!='3');
…
```

Temă:

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.