

# Instrucțiuni

---

CURS NR. 1/3

# Instrucțiuni

---



## Reprezintă

- Elementele fundamentale ale funcțiilor
- Comenzile date calculatorului
- Determină fluxul de control al programului

## Instrucțiuni de bază

- Instrucțiunea expresie
- Instrucțiunea vidă
- Instrucțiuni secvențiale
- Instrucțiuni selective
- Instrucțiuni iterative
- Instrucțiuni de salt

## Instrucțiuni compuse

- Create prin combinarea instrucțiunilor de bază

# Instrucțiunea expresie



Formată dintr-o expresie urmată de semnul ;

**expresie;**

Sunt cele mai frecvente expresii

- se bazează pe expresii de atribuire, aritmetice și de incrementare / decrementare
  - adică expresii care au efecte secundare: schimbă valoarea unui operand

Exemple:

```
a = 123;  
b = a + 5;  
b++;
```

Expresie vs. instrucțiune

**Expresie**

`i++`

`a=a-5`

**Instrucțiune**

`i++;`

`a=a-5;`

# Instrucțiunea vidă

---



O instrucțiune care constă doar din caracterul ;

- folosită în locurile în care limbajul impune existența unei instrucțiuni, dar programul nu trebuie să execute nimic

Cel mai adesea instrucțiunea vidă apare în combinație cu instrucțiunile repetitive

- Ex. Vezi instrucțiunea for

# Instrucțiunea compusă

---



Numită și **instrucțiune bloc**

Alcătuită prin gruparea mai multor instrucțiuni și declarații

- folosite în locurile în care sintaxa limbajului presupune o singură instrucțiune, dar programul trebuie să efectueze mai multe instrucțiuni
- Gruparea
  - Includerea instrucțiunilor între acolade, { }
  - Astfel compilatorul va trata secvența de instrucțiuni ca pe o singură instrucțiune
- { secvență de declarații și instrucțiuni }

# Instrucțiuni selective

---



ramifică fluxul de control în funcție de valoarea de adevăr a expresiei evaluate

C furnizează două instrucțiuni selective

- instrucțiunea **if** și
- instrucțiunea **switch**

# Instrucțiunea **if**



## instrucțiunea selectivă fundamentală

- permite selectarea **uneia dintre două alternative** în funcție de valoarea de adevăr a expresiei testate

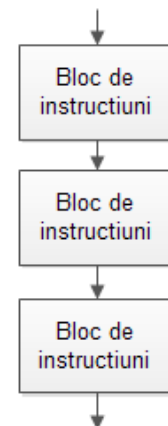
### Forma generală

```
if ( expresie ) instrucțiune1  
else instrucțiune2
```

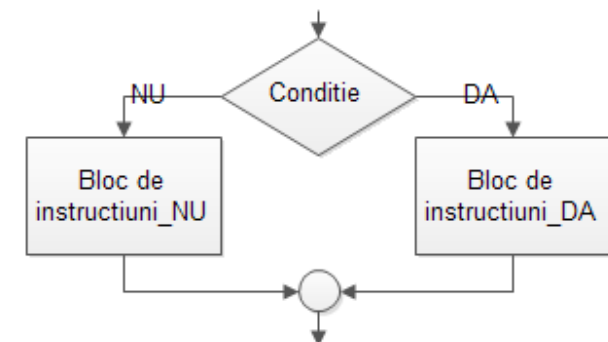
### Valoarea expresiei incluse între paranteze rotunde trebuie să fie **scalară**

- Dacă e nenulă se alege spre execuție *instrucțiune1*
- Altfel se alege spre execuție *instrucțiune2*

- **Structura secvențială** vs. **structura selectivă** (decizională)
  - Reprezentare comparativă folosind schema logică



Structura secvențială



Structura selectivă

# Instrucțiunea `if`

## Exemplu

```
/* Nume: selectie_if.c
 * Scop: exemplificarea instructiunii selective if
 */

#include <stdio.h>

int main()
{
    int luna;
    printf("Introduceti luna curenta ca si numar intreg: ");
    scanf("%d", &luna);

    if (luna < 7)
        printf("Suntem in prima jumatate a anului \n");
    else
        printf("Suntem in a doua jumatate a anului \n");

    return 0;
}
```

- Rezultatul unei rulări

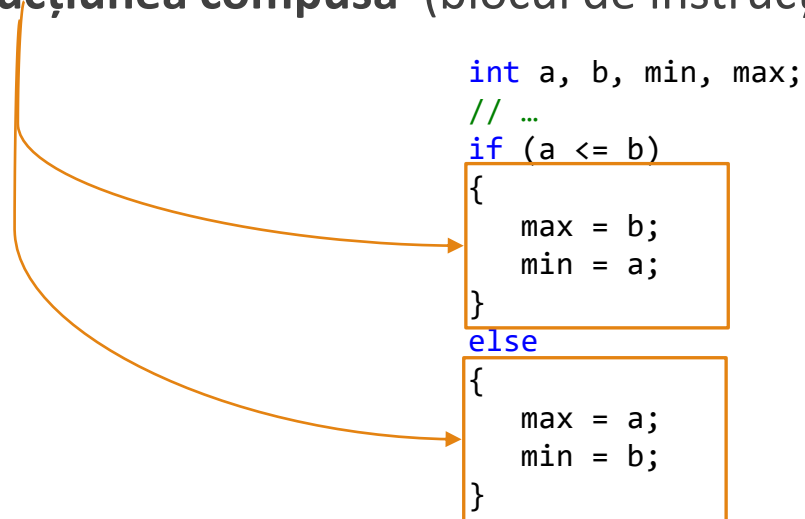
```
Introduceti luna curenta ca si numar intreg: 10
Suntem in a doua jumatate a anului
Press any key to continue . . .
```



# Instrucțiunea `if`

Dacă trebuie efectuate mai multe instrucțiuni pe oricare ramură

- folosim **instrucțiunea compusă** (blocul de instrucțiune)



```
int a, b, min, max;  
// ...  
if (a <= b)  
{  
    max = b;  
    min = a;  
}  
else  
{  
    max = a;  
    min = b;  
}
```

În unele cazuri ramura `else` poate lipsi

- Forma generală

`if ( expresie ) instrucțiune`

```
if (a % 2 == 0)  
    printf("%d este numar par \n", a);
```

# Instrucțiunea `if`



## Erorare frecventă

- confundarea operatorului de **egalitate** `==` cu operatorul de **atribuire** `=`

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

- mesajul `a este 10` nu va fi afișat
  - după testarea egalității folosind operatorul `==` se returnează 0
    - (2 nefiind egal cu 10)

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

- mesajul `a este 10` va fi afișat întotdeauna
- expresia `a = 10`
  - `a` ia valoarea 10
  - se evaluează la *adevărat* și se trece la executarea instrucțiunii `printf`

# Instrucțiunea `if`

## Instrucțiuni `if` **imbricate**

- Pe oricare ramură pot apare alte instrucțiuni `if`

- Forma generală

```
if ( expresie )  
    if ( expresie2 ) instrucțiune1  
    else instrucțiune2  
else instrucțiune3
```

- Exemplu

```
int a, b;  
// ...  
if (a <= b)  
    if (a == b)  
        printf("a = b");  
    else  
        printf("a < b");  
else printf("a > b");
```

# Instrucțiunea `if`

## Instrucțiuni `if` **cascadate**

- testează **succesiv mai multe condiții** implementând o variantă de selecție multiplă
- Forma generală

```
/* Nume: note.c
* Scop: calificativ asociat notei
*/

#include <stdio.h>

int main() {
    float nota;

    printf("Introduceti o nota in intervalul [1, 10]: ");
    scanf("%f", &nota);

    if (nota > 9 && nota <= 10)
        printf("Calificativul este: EXCELENT \n");
    else if (nota > 8 && nota <= 9)
        printf("Calificativul este: Foarte bine \n");
    else if (nota > 7 && nota <= 8)
        printf("Calificativul este: Bine \n");
    else if (nota > 5 && nota <= 7)
        printf("Calificativul este: Acceptabil \n");
    else printf("Calificativul este: Insuficient \n");

    return 0;
}
```

```
if ( expresie ) instrucțiune1
else if ( expresie2 ) instrucțiune2
else if ( expresie3 ) instrucțiune3
...
else instrucțiuneN
```

```
Introduceti o nota in intervalul [1, 10]: 8
Calificativul este: Bine
```

# Instrucțiunea switch



## Efectuează **selecția multiplă**

- Utilă când expresia de evaluat are mai multe valori posibile

Valoarea expresiei trebuie să fie întreagă

## Forma generală

```
switch (expresie)  
{  
case val_const_1: instrucțiuni1  
case val_const_2: instrucțiuni2  
...  
case val_const_N: instrucțiuniN  
default : instrucțiuniD  
}
```

Etichetele: constante întregi - inclusiv caractere

Atenție! Nu poate testa intervale !

## Poate fi întotdeauna reprezentată prin instrucțiunea if

- de regulă prin instrucțiuni if cascadate

În cazul instrucțiunii switch **fluxul** de controlul sare direct la instrucțiunea corespunzătoare valorii expresiei testate

- switch este mai rapid și codul rezultat mai ușor de înțeles

# Instrucțiunea switch

```
/* Nume: expresie_aritmetica.c
* Scop: evalueaza si afiseaza valoarea unei expresii aritmetice
*/

#include <stdio.h>

int main() {
    int nr1, nr2, rez;
    char op;

    printf("Introduceti o expresie aritmetica sub forma: nr1 operator nr2: ");
    scanf("%d %c %d", &nr1, &op, &nr2);

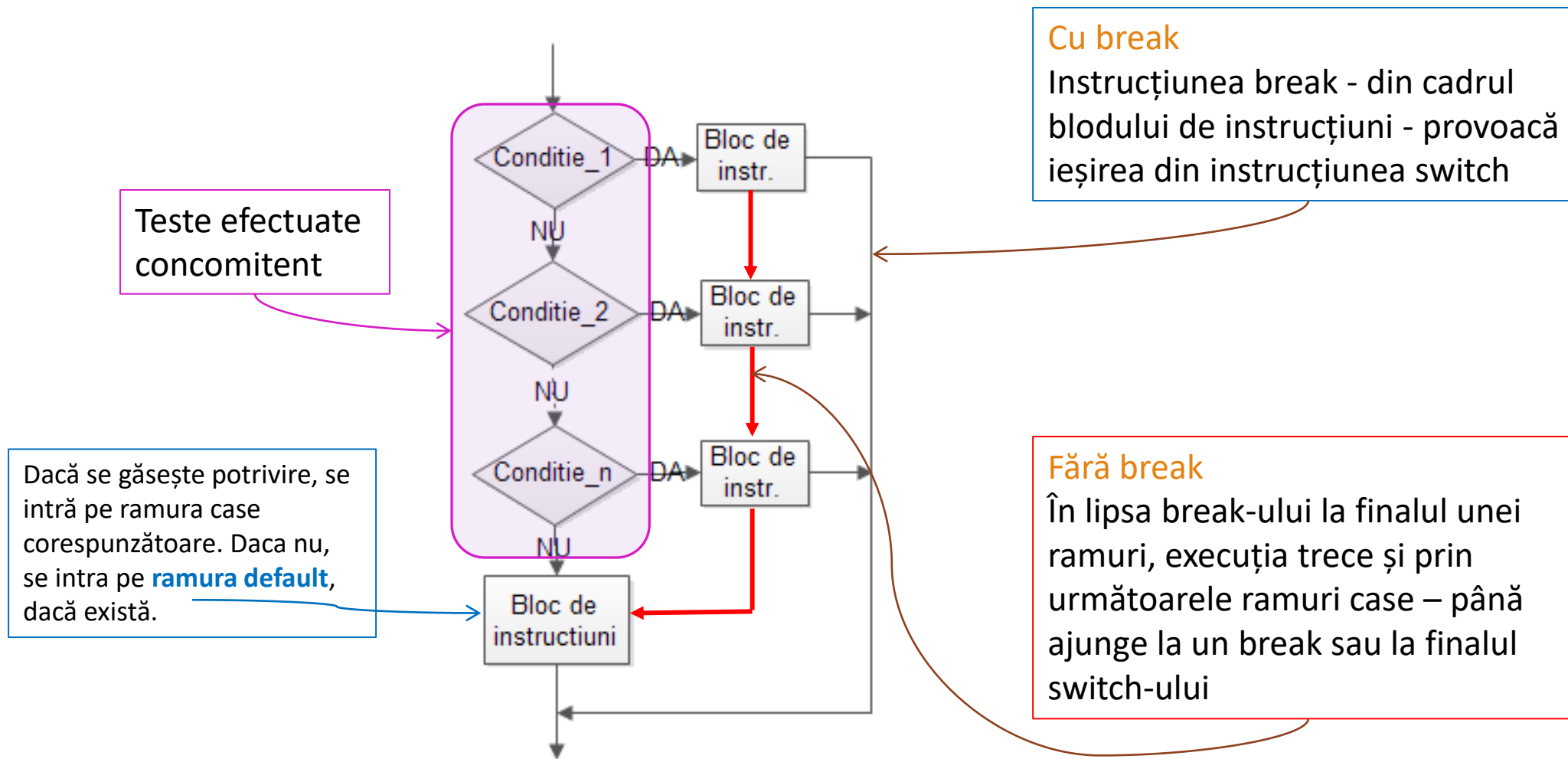
    switch (op)
    {
        case '+': rez = nr1 + nr2; break;
        case '-': rez = nr1 - nr2; break;
        case '*': rez = nr1 * nr2; break;
        case '/': rez = nr1 / nr2; break;
        case '%': rez = nr1 % nr2; break;
    }

    printf("Valoarea expresiei aritmetice introduse este: %d \n", rez);

    return 0;
}
```

```
Introduceti o expresie aritmetica sub forma: nr1 operator nr2: 3 + 12
Valoarea expresiei aritmetice introduse este: 15
```

# Instrucțiunea switch



# Instrucțiunea switch



## Mod de funcționare și constrângeri

- `expresie` se evaluează o singură dată la intrarea în instrucțiunea switch
- `expresie` trebuie să rezulte într-o **valoare întreagă** (poate fi inclusiv caracter, dar nu valori reale sau șiruri de caractere)
- **valorile din ramurile** case notate `val_ const_i` (numite și etichete) trebuie să fie **constante întregi** (sau caracter), reprezentând o singură valoare
- nu se poate reprezenta un interval de valori
- 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 valoarea expresiei se potrivește cu vreuna din valorile constante din ramurile case, atunci se vor executa instrucțiunile corespunzătoare acelei ramuri, altfel se execută instrucțiunea de pe **ramura default** (dacă aceasta există)
- dacă **nu s-a întâlnit break** la finalul instrucțiunilor de pe ramura 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



# Instrucțiunea switch



## Omiterea instrucțiunii break de la finalul unei ramuri case

- Accidentală - este o eroare frecventă
- Deliberată - permite fluxului de execuție să intre și pe ramura case următoare

```
#include <stdio.h>

int main()
{
    int an, luna, bisect = 0, nr_zile;

    printf("Introduceti luna si anul care va intereseaza: ");
    scanf("%d %d", &luna, &an);

    // pentru a determina daca luna februarie are 28 sau 29 de zile
    // determinam daca anul este bisect -> bisect devine 1
    if ((an % 4 == 0 && an % 100 != 0) || an % 400 == 0)
        bisect = 1;

    switch (luna)
    {
        case 1: case 3: case 5:
        case 7: case 8: case 10:
        case 12: nr_zile = 31; break;
        case 4: case 6: case 9:
        case 11: nr_zile = 30; break;
        case 2: if (bisect == 1) nr_zile = 29;
                else nr_zile = 28;
                break;
        default: printf("Luna trebuie sa fie in intervalul [1, 12] !");
    }

    printf("Numarul de zile din luna %d, anul %d, este: %d \n", luna, an, nr_zile);
    return 0;
}
```

Introduceti luna si anul care va intereseaza: 2 2016  
Numarul de zile din luna 2, anul 2016, este: 29

# Instrucțiuni repetitive



Sunt numite și **instrucțiuni 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**.

# Instrucțiunea `while`



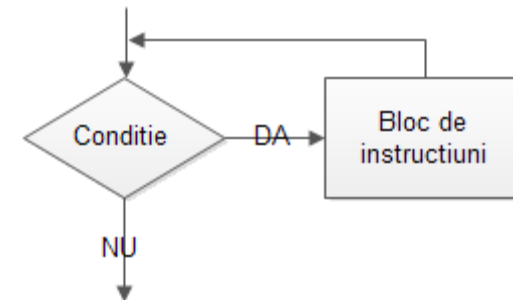
Execută în mod repetat o instrucțiune atâta timp cât expresia de control este evaluată la valoarea adevărat

Evaluarea se efectuează la începutul instrucțiunii și a fiecărei iterații

- Dacă rezultatul corespunde valorii logice adevărat
  - Se execută corpului instrucțiunii, după care se revine la testarea expresiei de control
- Acești pași se repetă până când expresia va fi evaluată la fals
  - Acesta va determina ieșirea din instrucțiune și trecerea la instrucțiunea imediat următoare

Foma generală

```
while ( expresie ) instrucțiune
```



# Instrucțiunea while

## Observații

- Valorile care participă în expresia de control trebuie să fie inițializate înainte
- Evitarea ciclului infinit

## Exemplu

```
/* Nume: suma_numere.c
 * Scop: calculeaza suma numerelor mai mici decat n
 */

#include <stdio.h>

int main() {
    int nr, i, suma;

    printf("Introduceti un numar intreg: ");
    scanf("%d", &nr);

    i = 0; suma = 0;
    while (i <= nr) {
        suma += i;
        i++;
    }
    printf("Suma numerelor mai mici decat %d este: %d \n", nr, suma);

    return 0;
}
```

Introduceti un numar intreg: 12  
Suma numerelor mai mici decat 12 este: 78

# Instrucțiunea while

## Observații

- Valorile care participă în expresia de control trebuie să fie inițializate înainte
- Evitarea ciclului infinit

## Exemplu

```
/* Nume: suma_numere.c
 * Scop: calculeaza suma numerelor mai mici decat n
 */

#include <stdio.h>

int main() {
    int nr, i, suma;

    printf("Introduceti un numar intreg: ");
    scanf("%d", &nr);

    i = 1; suma = 0;
    while (i < nr) {
        suma += i;
        i++;
    }
    printf("Suma numerelor mai mici decat %d este: %d \n", nr, suma);

    return 0;
}
```

```
Introduceti un numar intreg: 12
Suma numerelor mai mici decat 12 este: 78
```

# Instrucțiunea do-while



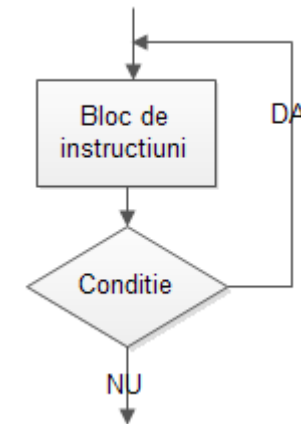
Efectuează în mod repetat o instrucțiune atâta timp cât expresia de control este adevărată

Evaluarea se face la finalul fiecărei iterații

- corpul instrucțiunii este executat cel puțin o dată

Forma generală

```
do instrucțiune while ( expresie ) ;
```



Eroare frecventă: omiterea caracterului punct și virgulă de la finalul instrucțiunii

# Instrucțiunea do-while

## Exemplu

```
/* Nume: suma_vector.c
 * Scop: citeste elementele unui vector si afiseaza suma lor
 */

#include <stdio.h>
#define N 100

int main() {
    int nr, i, suma;
    int v[N];

    do
    {
        printf("Introduceti numarul de elemente (1 <= nr <= 100): ");
        scanf("%d", &nr);
    } while (nr<1 || nr >100);

    i = 0; suma = 0;
    do {
        printf("v[%d]: ", i);
        scanf("%d", &v[i]);
        suma += v[i];
        i++;
    } while (i<nr);

    printf("Suma elementelor vectorului este: %5d\n", suma);

    return 0;
}
```

```
Introduceti numarul de elemente (1 <= nr <= 100): 10
v[0]: 1
v[1]: 4
v[2]: 7
v[3]: 2
v[4]: 9
v[5]: 10
v[6]: 11
v[7]: 23
v[8]: 79
v[9]: 6
Suma elementelor vectorului este:    152
```

# Instrucțiunea for



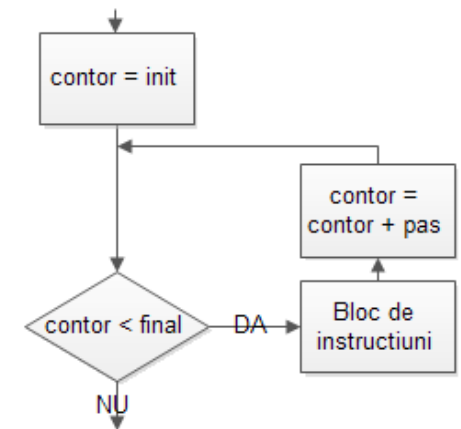
evaluarea expresiei de control se face la începutul fiecărei iterații

Forma generală

```
for ( expresii_init ; expresie_control ; expresii_ajustare ) instrucțiune
```

poate fi întotdeauna transcrisă folosind o instrucțiune while

```
expresii_init;  
while ( expresie_control )  
{ instrucțiune  
  expresii_ajustare;  
}
```





# Instrucțiunea for



Instrucțiunea for permite ca elementul de ajustare din antetul instrucțiunii să cuprindă mai multe expresii

```
// citirea si insumarea elementelor din vectorul de intregi cu for

for (i = 0, suma = 0; i < nr; i++)
{
    printf("v[%d]: ", i);
    scanf("%d", &v[i]);
    suma += v[i];
}
```

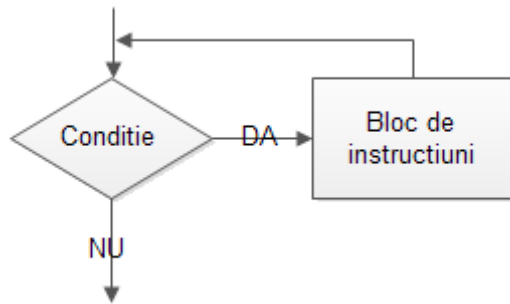
- se poate ajunge chiar și la situația în care corpul instrucțiunii nu mai conține nici o instrucțiune de executat
  - Atunci se folosește **instrucțiunea vidă** (punct și virgulă) pentru a indica sfârșitul instrucțiunii for

```
// insumarea elementelor din vectorul de intregi cu for

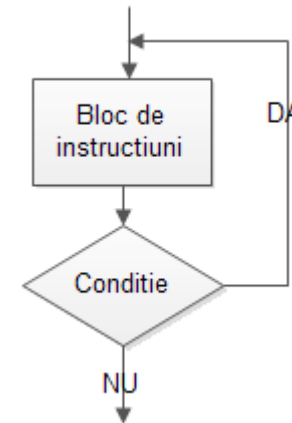
for (i = 0, suma = 0; i < nr; suma += v[i], i++);
printf("Suma elementelor este: %d", suma);
```

# Recapitulare instrucțiuni repetitive

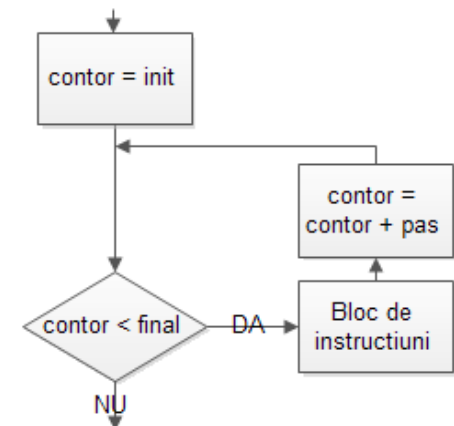
`while ( expresie ) instrucțiune`



`do instrucțiune while ( expresie );`



`for ( expresii_init ; expresie_control ; expresii_ajustare ) instrucțiune`



# Instrucțiunile `break`, `continue` și `goto`



## Realizează **salturi**

- întrerup controlului secvențial al programului și continuă execuția dintr-un alt punct al programului sau chiar provoacă ieșirea din program

Instrucțiunea `break` provoacă **ieșirea din instrucțiunea curentă**

Instrucțiunea `continue` provoacă **trecerea la iterația imediat următoare** în instrucțiunea repetitivă

Instrucțiunea `goto` produce un **salt la o etichetă predefinită** în cadrul aceleiași funcții

# Instrucțiunea break



## Instrucțiunea break provoacă ieșirea din instrucțiunea curentă

- produce un salt din punctul unde este apelată în punctul imediat următor instrucțiunii
- Se aplică în cazul instrucțiunilor repetitive (while, do-while, for) și switch
- Exemplu: permite părăsirea ciclului în momentul în care rezultatul este deja determinat deși nu s-au parcurs toate iterațiile

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


int main() {
    int nr, i;

    do {
        printf("Introduceți numărul natural > 2 pentru a testa dacă este prim: ");
        scanf("%d", &nr);
    } while (nr < 2);

    i = 2;
    while (i < sqrt(nr)) {
        if (nr % i == 0) break;
        i++;
    }

    if (i >= sqrt(nr))
        printf("Numărul %d este prim \n", nr);
    else
        printf("Numărul %d (divizibil cu %d) NU este prim \n", nr, i);

    return 0;
}
```



Salt după instrucțiunea while

# Instrucțiunea break

## Instrucțiunea `break` provoacă ieșirea din instrucțiunea curentă

- produce un salt din punctul unde este apelată în punctul imediat următor instrucțiunii
- Se aplică în cazul instrucțiunilor repetitive (`while`, `do-while`, `for`) și `switch`
- **Exemplu:** permite părăsirea ciclului în momentul în care rezultatul este deja determinat deși nu s-au parcurs toate iterațiile

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

int main() {
    int nr, i;

    do {
        printf("Introduceti numarul natural > 2 pentru a testa daca este prim: ");
        scanf("%d", &nr);
    } while (nr < 2);

    i = 2;
    while (i < sqrt(nr)) {
        if (nr % i == 0) break;
        i++;
    }

    if (i >= sqrt(nr))
        printf("Numarul %d este prim \n", nr);
    else
        printf("Numarul %d (divizibil cu %d) NU este prim \n", nr, i);

    return 0;
}
```

```
Introduceti numarul natural > 2 pentru a testa daca este prim: 291
Numarul 291 (divizibil cu 3) NU este prim
```

# Instrucțiunea `continue`



Se aplică doar în cadrul instrucțiunilor ciclice unde provoacă trecerea la iterația imediat următoare, fără a termina iterația curentă

- Atenție la incrementarea contorului – pentru trecerea la iterația următoare în cadrul `while` și `do-while`

## Exemplu

- suma elementelor unui vector de întregi care sunt divizibile cu 3

```
#include <stdio.h>

int main() {
    int nr, i, suma, v[100];

    printf("Introduceti numarul de elemente (1 <= nr <= 100): ");
    scanf("%d", &nr);

    for (i = 0; i < nr; i++) {
        printf("v[%d]: ", i);
        scanf("%d", &v[i]);
    }

    printf("\nElementele divizibile cu 3 sunt: \n");
    for (i = 0, suma = 0; i < nr; i++) {
        if (v[i] % 3 != 0) continue;
        printf("v[%d]: %d\n", i, v[i]);
        suma += v[i];
    }

    printf("\nSuma elementelor divizibile cu 3 este: %d", suma);

    return 0;
}
```

Salt la finalul iterației curente

# Instrucțiunea `continue`

Se aplică doar în cadrul instrucțiunilor ciclice unde provoacă trecerea la iterația imediat următoare, fără a termina iterația curentă

- Atenție la incrementarea contorului – pentru trecerea la iterația următoare în cadrul `while` și `do-while`

## Exemplu

- suma elementelor unui vector de întregi care sunt divizibile cu 3

```
#include <stdio.h>

int main() {
    int nr, i, suma, v[100];

    printf("Introduceti numarul de elemente (1 <= nr <= 100): ");
    scanf("%d", &nr);

    for (i = 0; i < nr; i++) {
        printf("v[%d]: ", i);
        scanf("%d", &v[i]);
    }

    printf("\nElementele divizibile cu 3 sunt: \n");
    for (i = 0, suma = 0; i < nr; i++) {
        if (v[i] % 3 != 0) continue;
        printf("v[%d]: %d\n", i, v[i]);
        suma += v[i];
    }

    printf("\nSuma elementelor divizibile cu 3 este: %d", suma);

    return 0;
}
```

```
Introduceti numarul de elemente (1 <= nr <= 100): 5
v[0]: 4
v[1]: 9
v[2]: 13
v[3]: 2
v[4]: 21

Elementele divizibile cu 3 sunt:
v[1]: 9
v[4]: 21

Suma elementelor divizibile cu 3 este: 30
```

# Instrucțiunea `continue`



## Atenție la incrementarea contorului

- pentru trecerea la iterația următoare în cadrul `while` și `do-while`

## Exemplu

- rescrierea ciclului de însumare din exemplul anterior folosind o instrucțiune `while` în loc de instrucțiunea `for`

```
// ...  
printf("\nElementele divizibile cu 3 sunt: \n");  
  
i = 0, suma = 0;  
  
while (i < nr) {  
    if (v[i] % 3 != 0) {  
        i++;  
        continue;  
    }  
    printf("v[%d]: %d\n", i, v[i]);  
    suma += v[i];  
    i++;  
}
```

Programatorul trebuie să se asigure că valoarea contorului este incrementată înainte de întreruperea iterației curente



# Instrucțiunea goto



produce un salt necondiționat la o etichetă predefinită, aflată în cadrul aceleiași funcții

- Se recomandă evitarea instrucțiunii goto, deoarece face dificilă trasarea fluxului de control și modificarea programului
- Codul poate fi întotdeauna rescris pentru a nu folosi goto
- Exemplu
  - caută primul element dintr-un vector de întregi, care are cel puțin două cifre și toate cifrele au valori identice

```
// cautare element
i = 0;
while (i < nr_elem)
{
    if (v[i] <= 10) goto respins;
    nr = v[i];          // pastram v[i] nealterat
    val = nr % 10;      // valoarea primului rest
    nr = nr / 10;
    while (nr != 0) {
        rest = nr % 10;
        if (rest != val) goto respins;
        nr = nr / 10;
    }
    printf("Primul element > 10 cu cifrele identice: %d", v[i]);
    break;

    respins: i++;
}
```

# Recapitulare Instrucțiuni

---

## Instrucțiunile reprezintă

- Elementele fundamentale ale funcțiilor
- Comenzile date calculatorului
- Determină fluxul de control al programului

## Instrucțiuni de bază

- Instrucțiunea expresie
- Instrucțiunea vidă
- Instrucțiuni secvențiale
- Instrucțiuni selective
- Instrucțiuni iterative
- Instrucțiuni de salt

## Instrucțiuni compuse

- Create prin combinarea instrucțiunilor de bază

# Reguli și recomandări



- Folosim acolade pentru corpul instrucțiunilor
  - chiar dacă este o singură instrucțiune în corp

În loc de:

```
for (i = 0, suma = 0; i < nr; i++)  
    suma += v[i];
```



```
for (i = 0, suma = 0; i < nr; i++) {  
    suma += v[i];  
}
```

Pentru

- **delimitarea și gruparea clară a instrucțiunilor**
- **fiabilitate la modificări**
  - Extindere ulterioară a corpului

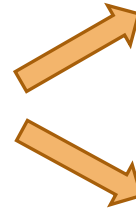
# Reguli și recomandări



- Să nu confundăm operatorul de atribuire (=) cu operatorul de egalitate (==) în contextele următoare
  - În expresia de control al instrucțiunilor if, while, do ... while,
  - Al doilea operand din for
  - Primul, al doilea sau al treilea operand din ? :
  - Oricare operand al operatorilor && și ||
  - Al doilea operand al operatorului ,

De evitat:

```
if (a = b) {  
    /* ... */  
}
```



```
if (a == b) {  
    /* ... */  
}
```

**Corect:**

Dacă se dorește testarea egalității,  
și nu atribuire

```
if ((a = b) != 0) {  
    /* ... */  
}
```

**Corect:**

Dacă atribuirea este  
intenționată

- Dacă unul din operanzi este o constantă, putem evita atribuirea nedorită prin plasarea constantei în partea stângă a operatorului

Eroare  
sesizată  
de compilator:

```
if (3 = a) {  
    /* ... */  
}
```



```
if (3 == a) {  
    /* ... */  
}
```

**Corect:**

Odată sesizată eroarea este remediată  
simplu

# Surse bibliografice

---

- K. N. King, C Programming – A Modern Approach, 2nd edition, W. W. Norton & Co., 2008
  - Capitolele 5 și 6
- Deitel & Deitel, C How to Program, 6th edition, Pearson, 2009
  - Capitolul 4