

PROGRAMA CURSULUI

❑ Introducere

- Algoritmi
- Limbaje de programare.

❑ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

❑ Fișiere text

- Funcții specifice de manipulare.

❑ Funcții (1)

- Declaraire și definire. Apel. Metode de transmitere a paramerilor. Pointeri la funcții.

❑ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetica pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

❑ Șiruri de caractere

- Funcții specifice de manipulare.

❑ Fișiere binare

- Funcții specifice de manipulare.

❑ Structuri de date complexe și autoreferite

- Definire și utilizare

❑ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.

CURSUL DE AZI

1. Structuri, uniuni, câmpuri de biți, enumerări.
2. Instrucțiuni de control

STRUCTURI

- Colecții de componente (variabile) înrudite și agregate sub un singur nume
- Pot conține componente (variabile) având diferite tipuri de date
- Frecvent utilizate pentru definirea de înregistrări care urmează a fi stocate în fișiere
- Combinate cu pointeri pot genera liste înlănțuite, stive, cozi, arbori, etc.

STRUCTURI

- sintaxa:

```
struct <nume> {  
    < tip 1 >  <variabila 1>;  
    < tip 2 >  <variabila 2>;  
    -----  
    < tip n >  <variabila n>;  
} lista_identificatori_de_tip_struct;
```

- variabilele care fac parte din structură sunt denumite membri (elemente sau câmpuri) ai structurii.

STRUCTURI

- Structura student cu variabile declarate

```
struct student {  
    int  numar_matricol;  
    char  nume[25];  
    char  CNP[14];  
    float  nota;  
} stud1,stud2;
```

- Structura student si variabile declarate ulterior

```
struct student {  
    int  numar_matricol;  
    char  nume[25];  
    char  CNP[14];  
    float  nota;  
};  
struct student stud1,stud2;
```

STRUCTURI

- Structura anonima cu variabile declarate

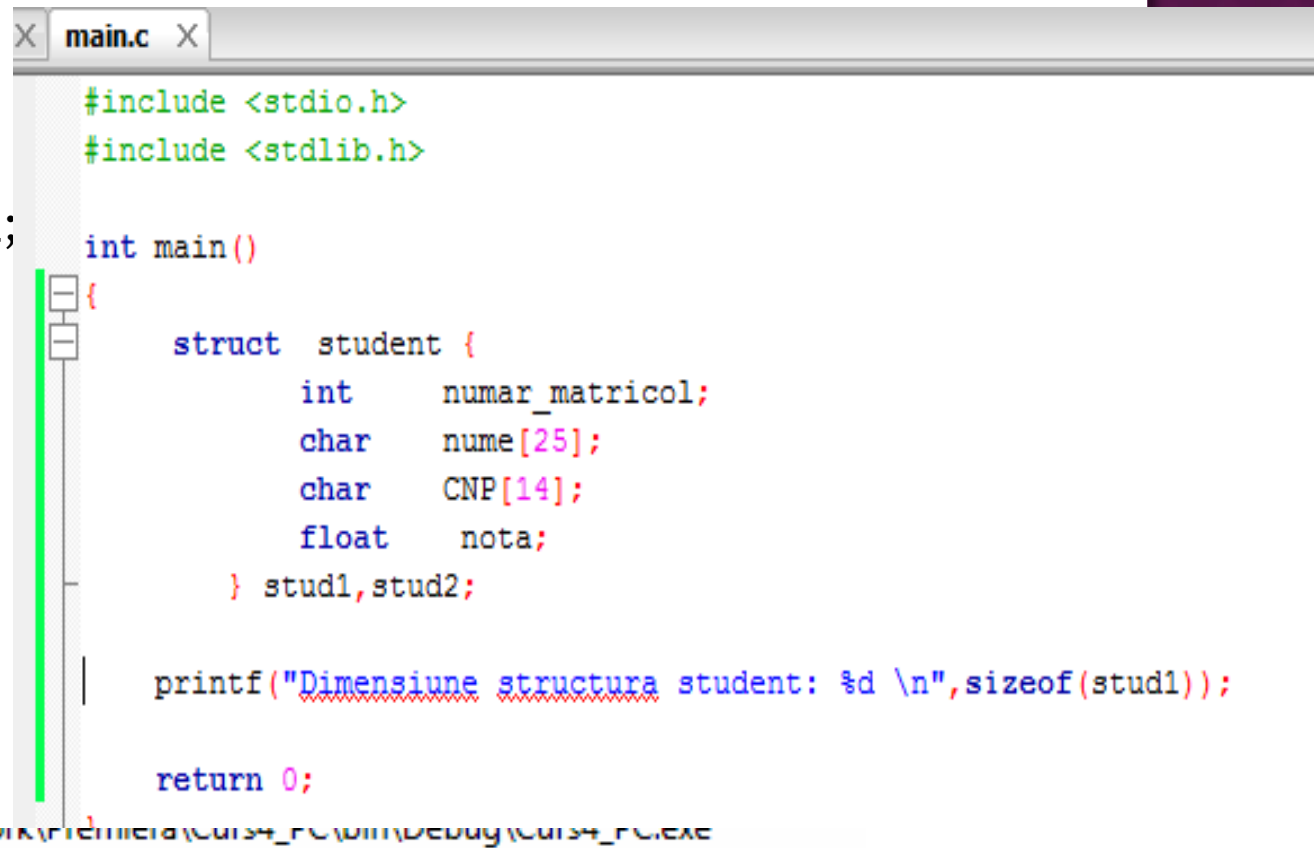
```
struct {  
    int  numar_matricol;  
    char  nume[25];  
    char  CNP[14];  
    float  nota;  
} stud1, stud2;
```

- Ulterior nu se mai pot declara alte variabile de acest tip
- Simpla definire a unei structuri nu ocupa memorie
- Variabilele declarate (instantele) ocupa memorie
- Memorie ocupata ~ suma dimensiunilor fiecarui camp
- Se aliniaza la multiplu de 4 octeti

STRUCTURI

❑ exemplu:

```
struct student{  
    int    numar_matricol;  
    char   nume[25];  
    char   CNP[14];  
    float  nota;  
} stud1,stud2;
```



```
main.c  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main()  
{  
    struct student {  
        int    numar_matricol;  
        char   nume[25];  
        char   CNP[14];  
        float  nota;  
    } stud1,stud2;  
  
    printf("Dimensiune structura student: %d \n",sizeof(stud1));  
  
    return 0;  
}
```

Dimensiune structura student: 48

Process returned 0 (0x0) execution time : 0.012 s
Press any key to continue.

STRUCTURI

- O structura nu poate contine un camp de tipul structurii insasi

```
struct student {  
    int  numar_matricol;  
    char  nume[25];  
    char  CNP[14];  
    struct student stud1;  
};
```

- O structura poate contine un camp de tipul pointer la tipul structurii insasi

```
struct student {  
    int  numar_matricol;  
    char  nume[25];  
    char  CNP[14];  
    struct student *stud1;  
};
```


STRUCTURI

Operatii permise cu structuri:

- accesul la membri structurii se face prin folosirea operatorului punct.
Ex.: `scanf("%f", & stud1.nota);`
- atribuire pentru variabile de tip structură: `stud1 = stud2;`
- Operator `sizeof()` si operator adresa

```
struct student{  
    int    numar_matricol;  
    char   nume[25];  
    char   CNP[14];  
    float  nota;  
} stud1,stud2;
```

CÂMPURI DE BIȚI

- tip special de membru al unei structuri care definește cât de lung trebuie să fie câmpul, în biți.
- permite accesul la un nivel bit.
- putem stoca mai multe variabile boolene într-un singur octet.
- nu se poate obține adresa unui câmp de biți.

CÂMPURI DE BIȚI

□ Sintaxa:

```
struct <nume> {  
    < tip 1 >  <variabila 1>: lungime;  
    < tip 2 >  <variabila 2>: lungime;  
    .....  
    < tip n >  <variabila n>: lungime;  
} lista_identificatori_de_tip_struct;
```

- lungime = numărul de biți dintr-un câmp.
- tipul câmpului de biți poate fi doar: **int**, **unsigned** sau **signed**.
- Atentie: câmpul de biți cu lungimea 1 se folosește la **unsigned** (un singur bit nu poate avea semn).

CÂMPURI DE BIȚI

```
struct student{  
    int    numar_matricol : 3;  
    char   nume[25];  
    char   CNP[14];  
    float  nota;  
    unsigned char admis: 1;  
} stud1,stud2;
```

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main()  
{  
    struct studentfull {  
        int    numar_matricol;  
        char   nume[25];  
        char   CNP[14];  
        float  nota;  
        unsigned char admis;  
    } st1,st2;  
  
    struct student {  
        int    numar_matricol:3;  
        char   nume[25];  
        char   CNP[14];  
        float  nota;  
        unsigned char admis:1;  
    } stud1,stud2;  
  
    printf("Dimensiuni: %d %d\n",sizeof(stud1),sizeof(st1));  
    return 0;}  

```

Dimensiuni: 48 52

Process returned 0 (0x0) execution time : 0.038 s
Press any key to continue.

CÂMPURI DE BIȚI

```
struct student{  
    int    numar_matricol : 3;  
    char   nume[25];  
    char   CNP[14];  
    float  nota;  
    unsigned char admis: 1;  
} stud1,stud2;
```

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main()  
{  
    struct student {  
        int    numar_matricol:3;  
        char   nume[25];  
        char   CNP[14];  
        float  nota;  
        unsigned char admis:1;  
    } stud1,stud2;  
  
    char m,a;  
    scanf("%d %d",&m,&a);  
  
    stud1.numar_matricol = m;  
    stud1.admis = a;  
  
    printf("%d %d \n",stud1.numar_matricol,stud1.admis);  
    return 0;}
```

```
4 1  
0 1
```

```
Process returned 0 (0x0)   execu  
Press any key to continue.
```

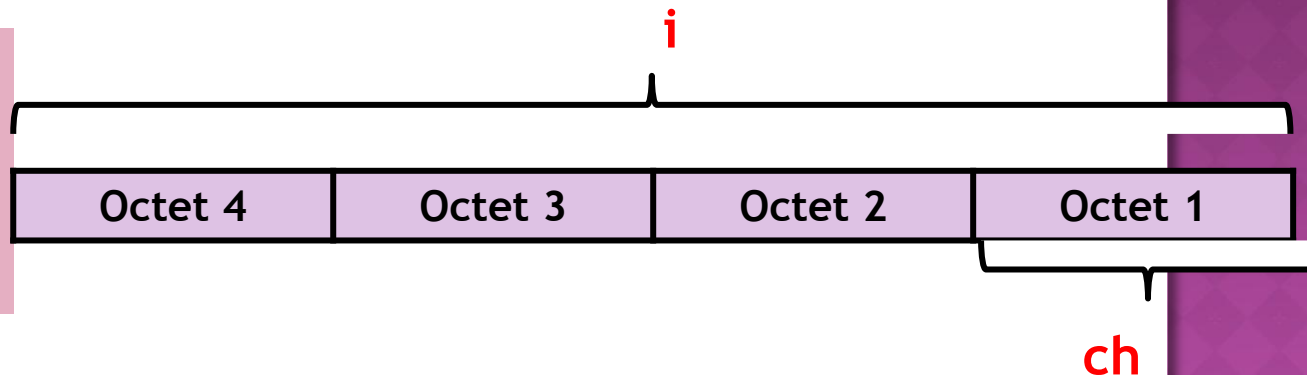
UNIUNI

- O uniune este o zonă de memorie care poate conține o varietate de componente la momente diferite de timp
- Conține numai o singură componentă (membru) la un anumit moment (doar ultimul membru accesat)
- Membrii unei uniuni partajează aceeași zonă de memorie
- Ajută la economisirea spațiului de memorie utilizat
- Zona de memorie rezervată are dimensiunea componentei care necesită cea mai multă memorie pentru reprezentare

UNIUNI

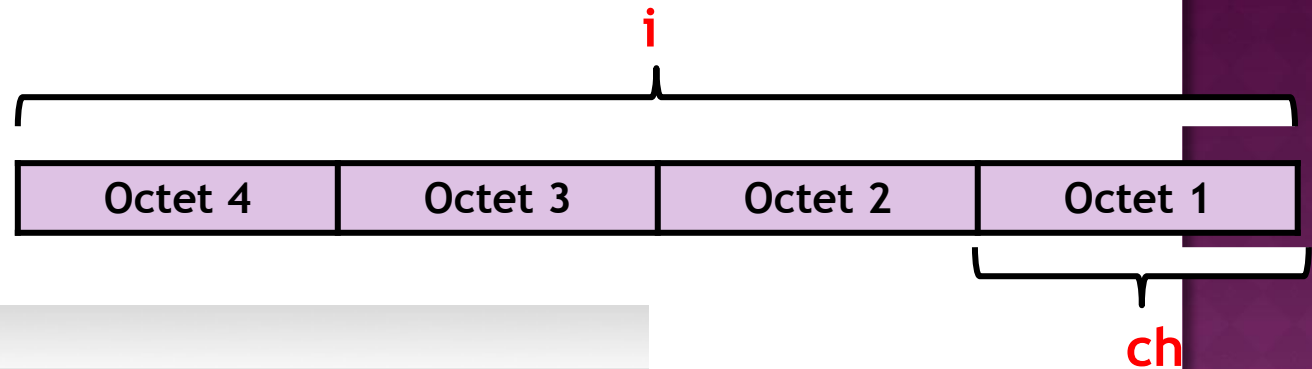
- Definire - la fel ca și o structură, dar folosind **union**
- Operațiile permise cu structuri sunt permise și cu uniuni
- Excepție: la inițializarea unei uniuni doar primul membru poate fi inițializat

```
union tip_u {  
    int i;  
    char ch;  
};
```



UNIUNI

□ esempiu



```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      union tip_u{
8          int i;
9          unsigned char ch;
10     };
11
12     union tip_u A;
13     printf("Dimensiune A = %d \n", sizeof(A));
14     A.ch = 10;
15     printf("A.ch = %d \n", A.ch);
16     printf("A.i = %d \n", A.i);
17
18     A.i = 300;
19     printf("A.ch = %d \n", A.ch);
20     printf("A.i = %d \n", A.i);
21
22
23     return 0;
24
25 }
```

Dimensiune A = 4
A.ch = 10
A.i = 10
A.ch = 44
A.i = 300

Process returned 0 (0x0) execution time : 0.007
Press ENTER to continue.

UNIUNI

Sa se implementeze o functie care foloseste o uniune pentru a inversa cei doi octeti ai unui intreg(reprezentat pe 2 octeti) citit de la tastatura. Programul principal va apela functia pentru a codifica sidecodifica un intreg dat.

Exemplu: $n = 20 \rightarrow$ 20 codificat este 5120
5120 decodificat este 20

UNIUNI

Sa
a ir
citi
per
Exe

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

int codificare(int n)
{
    union { char ch[2];
            int i;
        } uniune;

    uniune.i = n;

    uniune.ch[0] = uniune.ch[0]^uniune.ch[1];
    uniune.ch[1] = uniune.ch[0]^uniune.ch[1];
    uniune.ch[0] = uniune.ch[0]^uniune.ch[1];

    return uniune.i;
}

int main()
{
    int n;
    scanf("%d",&n);
    printf("%d",codificare(n));
    return 0;}
```

entru
x(teti)

UNIUNI

ru
i)

```
S #include <stdio.h>
a #include <stdlib.h>
C int codificare(int n)
P {
E     union { char ch[2];
        int i;
        } uniune;

    uniune.i = n;

    uniune.ch[0] = uniune.ch[0] ^ uniune.ch[1];
    uniune.ch[1] = uniune.ch[0] ^ uniune.ch[1];
    uniune.ch[0] = uniune.ch[0] ^ uniune.ch[1];

    return uniune.i;
}

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", codificare(n));
    return 0;
}
```

```
20
5120
Process returned 0 (0x0)   exec
Press any key to continue.
-
```

```
5120
20
Process returned 0 (0x0)   e
Press any key to continue.
-
```

ENUMERĂRI

- O enumerare conține un set de constante întregi reprezentate prin identificatori
- Permite folosirea unor nume sugestive pentru valori numerice
- Constantele sunt asemănătoare constantelor simbolice și au valori setate automat
- Valorile încep de la 0 și sunt incrementate cu 1
- Se pot seta valori explicite prin asignare cu operatorul =
- Numele constantelor trebuie să fie unice
- Variabilele de tip enumerare își pot asuma doar una din valorile constante din set
- Nu se poate garanta că reprezentarea pe tipul întreg a unei variabile de tipul enumerare poate fi folosită pentru a stoca alt întreg

ENUMERĂRI

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

int main()
{
    enum culoare{alb,negru=14,verde,albastru,rosu=30};

    printf("%d %d %d %d %d\n\n",alb,negru,verde,albastru,rosu);

    enum culoare x = negru;
    enum culoare y = albastru;
    int z = x+y;
    printf("%d %d %d\n\n",x,y,z);
    x = alb;
    x = 40000; // nu garanteaza ca se poate stoca corect valoarea in x
    printf("%d\n",x);

    return 0;
}
```

0 14 15 16 30

14 16 30

40000

Process returned 0 (0x0)
Press any key to continue.

ENUMERĂRI

De ce **enum**, in locul lui #define sau const?

Inserati *const_C* pe pozitia a 3-a din lista:

```
#define const_A 1  
#define const_B 2  
#define const_D 3  
#define const_E 4  
#define const_F 5  
... ..
```

SPECIFICATORUL TYPEDEF

- ❑ definește explicit noi tipuri de date.
- ❑ nu se declară o variabilă sau o funcție de un anumit tip, ci se asociază un nume (sinonimul) tipului de date.

❑ sintaxa:

typedef <definiție tip> <identificator>;

❑ **exemple:**

```
typedef int intreg;
```

```
typedef enum {false,true} boolean;
```

```
typedef struct {  
    doublereal;  
    doubleimag;  
}complex;
```

CURSUL DE AZI

1. Structuri, uniuni, câmpuri de biți, enumerări.
2. Instrucțiuni de control

INSTRUCȚIUNI DE CONTROL

□ reprezintă:

- elementele fundamentale ale funcțiilor
- comenzile date calculatorului
- determină fluxul de control al programului (ordinea de execuție a operațiilor din program)

□ instrucțiuni de bază

- instrucțiunea expresie
- instrucțiunea vidă
- instrucțiuni secvențiale/liniare
- instrucțiuni decizionale/selective simple sau multiple
- instrucțiuni repetitive/ciclice/iterative
- instrucțiuni de salt condiționat/necondiționat
- instrucțiunea return

INSTRUCȚIUNI DE CONTROL

- instrucțiuni compuse

- create prin combinarea instrucțiunilor de bază

- programare structurată

- instrucțiuni secvențiale
 - instrucțiuni decizionale
 - instrucțiuni repetitive

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

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 DECIZIONALE/SELECTIVE

- ❑ ramifică fluxul de control în funcție de valoarea de adevăr a expresiei evaluate
- ❑ limbajul C furnizează două instrucțiuni decizionale
 - ❑ instrucțiunea **if** – instrucțiune decizională simplă
 - ❑ instrucțiunea **switch** - instrucțiune decizională multiplă

INSTRUCȚIUNEA IF

- instrucțiunea selectivă fundamentală

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

- forma generală:

if (expresie)

 {bloc de instructiuni 1};

else

 {bloc de instructiuni 2};

- valoarea expresiei incluse între paranteze rotunde trebuie să fie un scalar

- dacă e nenulă se execută *blocul de instrucțiuni 1 (instrucțiunea compusă)*, altfel se execută *blocul de instrucțiuni 2*

INSTRUCȚIUNEA IF

- se evalueaza maximum si minimum dintre 2 intregi

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

INSTRUCȚIUNEA IF

- ❑ erori frecvente:

- ❑ confundarea operatorul de egalitate ==, cu operatorul de atribuire =

Exemple comparative:

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

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

INSTRUCȚIUNEA IF

- instrucțiuni IF imbricate

- pe oricare ramură poate conține alte instrucțiuni if

- forma generală:

if (expresie1)

if (expresie2) {bloc de instructiuni 1};

else {bloc de instructiuni 2};

else

{bloc de instructiuni 2};

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 în cascadă

□ forma generală:

```
if (expresie1) {bloc de instructiuni 1};  
else if (expresie2) {bloc de instructiuni 2};  
else if (expresie3) {bloc de instructiuni 3};  
...  
else {bloc de instructiuni N};
```

Exemplu: if (delta<0)

```
    printf("ecuatia nu are radacini reale");  
else if (delta==0)  
    printf ("ecuatia are 1 radacina reala egala cu %f", ...);  
else //cazul default - acopera toate celelalte p  
    printf("ecuatia are 2 radacini reale egale cu %f si %f", ...);
```

INSTRUCȚIUNEA SWITCH

- efectuează selecția multiplă
 - util când expresia de evaluat are mai multe valori posibile

- forma generală

```
switch (expresie){  
    case val_const_1: {bloc de instructiuni 1};  
    case val_const_2: {bloc de instructiuni 2};  
    .....  
    case val_const_n: {bloc de instructiuni N};  
    default: {bloc de instructiuni D};  
}
```

INSTRUCȚIUNEA SWITCH

- poate fi întotdeauna reprezentată prin instrucțiunea IF
 - de regulă prin instrucțiuni IF cascade
- **în cazul instrucțiunii switch fluxul de control sare direct la instrucțiunea corespunzătoare valorii expresiei testate**
- switch este mai rapid și codul rezultat mai ușor de înțeles

INSTRUCȚIUNEA SWITCH

```
char c;
int exit=0, print=1;
while(1)
{
    if(print)
    {
        printf("Alegeti una din urmatoarele optiuni:");
        printf("S - Start program\n");
        printf("O - Optiuni program\n");
        printf("X - Iesire program\n");
    }
    c=getchar();
    switch(c)
    {
        case 's':
        case 'S': printf("s-a executat programul\n"); print=1; break;
        case 'o':
        case 'O': printf("nici o optiune disponibila\n"); print=1; break;
        case 'x':
        case 'X': exit=1; break;
        default: print=0;
    }
    if(exit==1)
        break;
}
```

```
Alegeti una din urmatoarele optiuni:
S - Start program
O - Optiuni program
X - Iesire program
s
s-a executat programul
Alegeti una din urmatoarele optiuni:
S - Start program
O - Optiuni program
X - Iesire program
o
nici o optiune disponibila
Alegeti una din urmatoarele optiuni:
S - Start program
O - Optiuni program
X - Iesire program
X
nici o optiune disponibila
Alegeti una din urmatoarele optiuni:
S - Start program
O - Optiuni program
X - Iesire program
x
```

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`

INSTRUCȚIUNEA SWITCH

- ❑ 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

```
char c;
int exit=0, print=1;
while(1)
{
    if(print)
    {
        printf("Alegeti una din urmatoarele optiuni:\n");
        printf("S - Start program\n");
        printf("O - Optiuni program\n");
        printf("X - Iesire program\n");
    }
    c=getchar();
    switch(c)
    {
        case 's':
        case 'S': printf("s-a executat programul\n"); print=1; break;
        case 'o':
        case 'O': printf("nici o optiune disponibila\n"); print=1; break;
        case 'x':
        case 'X': exit=1; break;
        default: print=0;
    }
    if(exit==1)
        break;
}
```


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 testare inițială **for**

INSTRUCȚIUNEA WHILE

- ❑ execută în mod repetat o instrucțiune atâta timp cât expresia de control este evaluată la adevărat
- ❑ evaluarea se efectuează la începutul instrucțiunii
 - ❑ 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
- ❑ forma generală: while (expresie) {bloc de instrucțiuni}

INSTRUCTIUNEA WHILE

```
sumaNumere.c
1  #include <stdio.h>
2
3  int main()
4  {
5      int nr, i , suma;
6      printf("Introduceti un numar intreg: ");
7      scanf("%d",&nr);
8
9      i = 0; suma = 0;
10     while (i<=nr)
11     {
12         suma += i;
13         i++;
14     }
15     printf("Suma numerelor mai mici sau egale decat %d este: %d\n", nr, suma);
16
17     return 0;
18
19 }
20
```

❑ Observații

- ❑ valorile care participă în expresia de control să fie inițializate înainte
- ❑ evitare ciclului infinit

INSTRUCȚIUNEA WHILE

sumaNumere.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int nr, i , suma;
6      printf("Introduceti un numar intreg: ");
7      scanf("%d",&nr);
8
9      i = 0; suma = 0;
10     while ((i=i+1) && (i<=nr) && (suma+=i) );
11     printf("Suma numerelor mai mici sau egale decat %d este: %d\n", nr, suma);
12
13     return 0;
14
15 }
16
```

❑ Observații

- ❑ Dacă o expresie nu mai este adevărată nu se mai continuă cu evaluarea expresiilor următoare

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 {bloc de instrucțiuni} while (expresie);`
- **eroare frecventă:** omiterea caracterului punct și virgulă de la finalul instrucțiunii

INSTRUCTIUNEA DO-WHILE

```
#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;
}
```

Rezultatul unei r  ri a acestui program este:

```
Introduceti numarul de elemente (1 <= nr <= 100): 5
v[0]: 4
v[1]: 7
v[2]: 2
v[3]: 10
v[4]: 5
Suma elementelor vectorului este:      28
```

INSTRUCȚIUNEA FOR

- evaluarea expresiei de control se face la începutul fiecărei iterații

- forma generală:

```
for (expresii_init; expresie_ control; expresie_ajustare)  
    {bloc de instructiuni}
```

- poate fi întotdeauna transcrisă folosind o instrucțiune while:

```
expresii_init;  
while (expresie_control)  
    {bloc de instructiuni  
    expresii_ajustare;}
```

INSTRUCȚIUNEA FOR

```
// 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];
}
```

- ❑ instrucțiunea for permite ca elementul de ajustare din antetul instrucțiunii să cuprindă mai multe expresii
 - ❑ se poate ajunge chiar și la situația în care corpul instrucțiunii nu mai conține nici o instrucțiune de executat
 - ❑ 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);
```


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 GOTO

- instrucțiunea goto produce un salt la o etichetă predefinită în cadrul aceleiași funcții
- forma generală: goto eticheta
 - unde eticheta este definită în program
 - eticheta: instructiune

```
    int i=0;
eticheta:
    if (i%3!=0)
        printf("i=%d\n",i);
    i++;
    if (i<10)
        goto eticheta;
    return 0;
```

```
i=1
i=2
i=4
i=5
i=7
i=8
```

INSTRUCTIUNEA GOTO

```
i=0;

outer_next:
    if ( i >= NUM_ELEMENTS - 1 )
        goto outer_end;
    j = i+1;
inner_next:
    if ( j >= NUM_ELEMENTS )
        goto inner_end;
    if ( value[i] <= value[j] )
        goto no_swap;
    temp = value[i];
    value[i] = value[j];
    value[j] = temp;
no_swap:
    j += 1;
    goto inner_next;
inner_end:
    i += 1;
    goto outer_next;
outer_end:
    ;
```

INSTRUCȚIUNEA GOTO

Varianta fara GOTO:

```
for ( i = 0; i < NUM_ELEMENTS - 1; i += 1){  
    for ( j = i+1; j < NUM_ELEMENTS; j += 1){  
        if (value[i] > value[j]){  
            temp = value[i];  
            value[i] = value[j];  
            value[j] = temp;  
        }  
    }  
}
```

INSTRUCȚIUNEA GOTO

Situațiile în care GOTO este cel mai potrivit:
Evadarea din bucle imbricate

```
while ( conditie1 ) {  
    while ( conditie2 ){  
        while ( conditie3 ){  
            while ( conditie4 ){  
                if ( some_disaster )  
                    goto quit;  
            }  
        }  
    }  
}  
quit: ;
```

INSTRUCȚIUNEA RETURN

- ❑ se folosește pentru a returna fluxul de control al programului apelant dintr-o funcție (main sau altă funcție)
- ❑ are două forme:
 - ❑ `return;`
 - ❑ `return expresie;`