

Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 4 / 23.10.2023

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: pointeri, tablouri, șiruri de caractere, **structuri, uniuni, câmpuri de biți, enumerări**
- **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. Tipuri derivate de date:

- a. structuri,
- b. uniuni,
- c. câmpuri de biți,
- d. enumerări,
- e. tipuri definite de utilizatori

2. Instrucțiuni de control

Tipuri de date structurate

Limbajul C permite creare tipurilor de date în 5 moduri:

- a. **structura (**struct**)** – grupează mai multe variabile sub același nume;
- b. **câmpul de biți (variațiune a structurii)** → acces ușor la biții individuali
- c. **uniunea (**union**)** – face posibil ca aceleași zone de memorie să fie definite ca două sau mai multe tipuri diferite de variabile
- d. **enumerarea (**enum**)** – listă de constante întregi cu nume
- e. **tip definit de utilizator (**typedef**)** – definește un nou nume pentru un tip existent

a. Structuri

❑ variabile grupate sub același nume.

❑ 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

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

Obs. 1:

- ✓ dacă numele structurii (<nume>) lipsește, structura se numește **anonimă**.
- ✓ Dacă lista identificatorilor declarați lipsește, se definește doar tipul structură.
- ✓ Cel puțin una dintre aceste specificații trebuie să existe.

Obs. 2: dacă <nume> este prezent → se pot declara noi variabile de tip structura:

```
struct <nume> <lista noilor identificatori>;
```

Obs. 3: referirea unui membru al unei variabile de tip structură → **operatorul de selecție punct** . care precizează identificatorul variabilei și al câmpului.

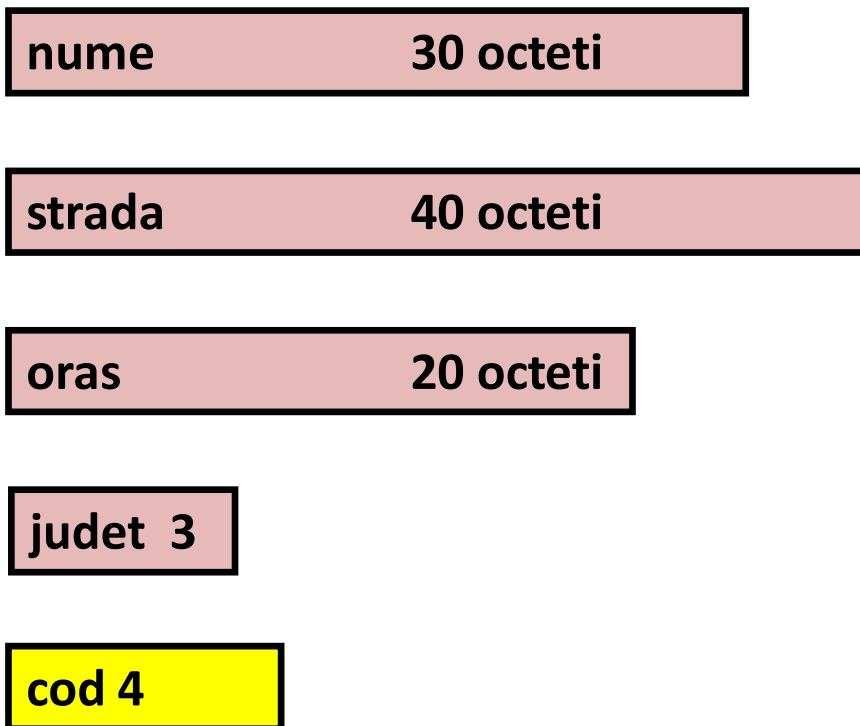
Structuri. Exemplu

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
    int cod;};
```

struct adrese A;

- ❑ numele **adrese** identifică aceasta structură de date particulară
- ❑ **struct adrese A;** declară o variabilă de tip adrese și îi alocă memorie

Structura din memorie pentru variabila A de tip adrese



Structuri. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      struct adrese {
6          char nume[30];
7          char strada[40];
8          char oras[20], judet[3];
9          int cod;
10     };
11     struct adrese A;
12
13     printf("%d\t", (int)sizeof(A.nume));
14     printf("%d\t", (int)sizeof(A.cod));
15     printf("%d", (int)sizeof(A));
16
17     return 0;
18 }
```

Ce afișează programul?

30

4

100

*Compilerul alocă
memorie în plus pentru
alinieare (multiplu de 4)*

Structuri. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      struct adrese {
6          char nume[30];
7          char strada[40];
8          char oras[20], judet[6];
9          int cod;
10     };
11     struct adrese A;
12
13     printf("%d\t", (int)sizeof(A.nume));
14     printf("%d\t", (int)sizeof(A.cod));
15     printf("%d", (int)sizeof(A));
16
17     return 0;
18 }
```

Ce afișează programul?

30

4

100

Structuri. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      struct adrese {
6          char nume[30];
7          char strada[40];
8          char oras[20], judet[3];
9      };
10     struct adrese A;
11
12     printf("%d\t", (int)sizeof(A.nume));
13     printf("%d\t", (int)sizeof(A));
14
15     return 0;
16 }
```

Ce afișează programul?

30

93

Compilerul alocă memorie în plus pentru aliniere numai când avem tipuri de date diferite.

Structuri. Exemplu

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20];  
    char judet[3];  
    int cod;  
} A, B, C;
```

- Definește un tip de structură numit **adrese**
- Declară ca fiind de acest tip variabilele **A, B, C**

```
struct {  
    char nume[30];  
    char strada[40];  
    char oras[20];  
    char judet[3];  
    int cod;  
} A;
```

- Declară o variabilă numită **A** definită de structura care o precede.

Structuri. Exemplu

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20];  
    char judet[3];  
    int cod;  
} A, B, C;
```

- Definește un tip de structură numit **adrese**
- Declară ca fiind de acest tip variabilele **A, B, C**

❑ **accesul la membrii structurii**
se face prin folosirea
operatorului punct:

nume_variabila.nume_camp

❑ **citirea:**

scanf("%d", &C.cod);

❑ **atribuiri pentru variabile de
tip structură:**

B = A;

b. 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 singur bit;
- ❑ putem stoca mai multe variabile boolene într-un singur octet;
- ❑ nu se poate obține adresa unui câmp de biți;
- ❑ **adaugă mai multă structurare.**

❑ **Sintaxa:**

```
struct <nume> {
```

```
    < tip 1 >          <variabila 1>: lungime;
```

```
    < tip 2 >          <variabila 2>: lungime;
```

```
    -----
```

```
    < tip n >          <variabila n>: lungime;
```

```
} lista_identificatori_de_tip_struct;
```

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;
```

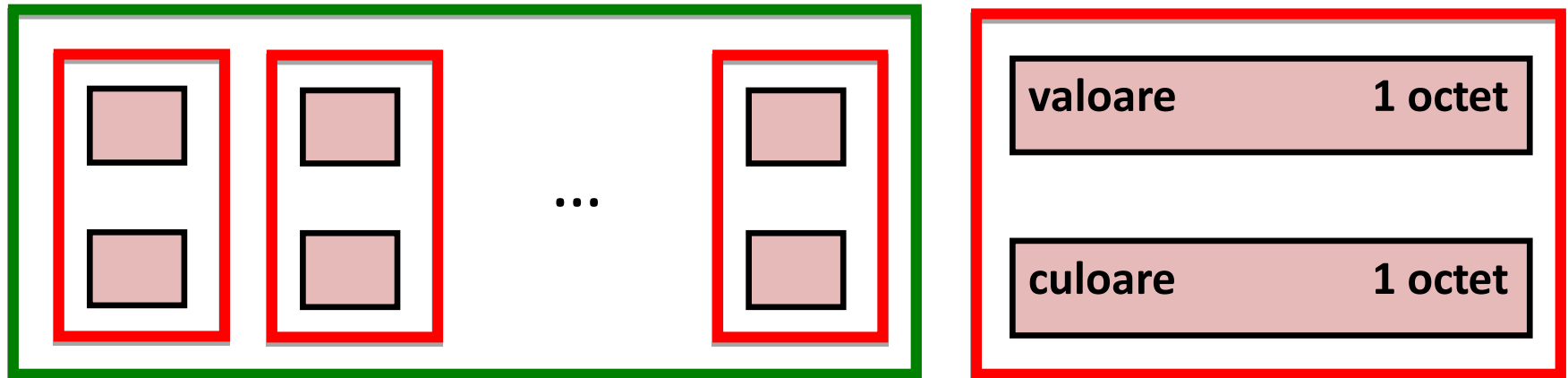
Observații:

- ❑ tipul câmpului de biți poate fi doar: **int**, **unsigned** sau **signed**;
- ❑ câmpul de biți cu lungimea 1 → **unsigned** (un singur bit nu poate avea semn);
- ❑ unele compilatoare → **doar unsigned**;
- ❑ lungime → **numărul de biți dintr-un câmp**.

Câmpuri de biți. Exemplu

Joc de cărți: reprezentăm o carte printr-o structură:

```
struct carteJoc {  
    unsigned char valoare; // valori între 1 și 13  
    unsigned char culoare; // valori între 1 și 4  
};  
struct carteJoc PachetCartiJoc[52];
```



Câmpuri de biți. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      struct carteJoc{
6          unsigned char valoare;
7          unsigned char culoare;
8      }A;
9      struct carteJoc PachetCartiJoc[52];
10
11     printf("%d\t", (int)sizeof(A));
12     printf("%d\t", (int)sizeof(PachetCartiJoc));
13
14     return 0;
15 }
```

Ce afișează programul?

2 104

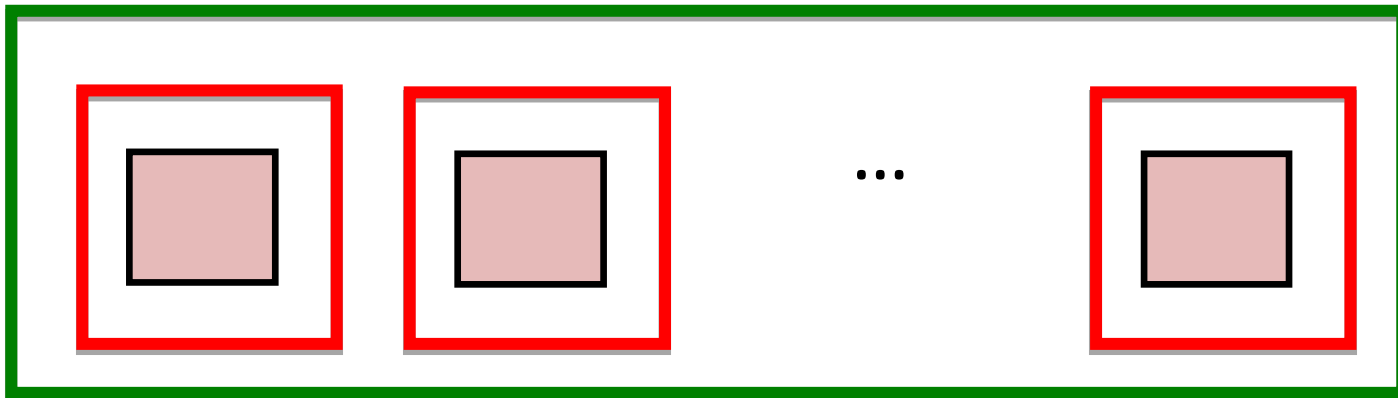
Dimensiune A este 2

Dimensiune PachetCartiJoc este 104(=2*52)

Câmpuri de biți. Exemplu

Joc de cărți: reprezentăm o carte printr-o structură:

```
struct carteJoc {  
    unsigned char valoare: 4; // 4 biți  
    unsigned char culoare: 2; // 2 biți  
};  
struct carteJoc PachetCartiJoc[52];
```



Câmpuri de biți. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      struct carteJoc{
6          unsigned char valoare:4;
7          unsigned char culoare:2;
8      }A;
9      struct carteJoc PachetCartiJoc[52];
10
11     printf("%d\t", (int)sizeof(A));
12     printf("%d\t", (int)sizeof(PachetCartiJoc));
13
14     return 0;
15 }
```

Ce afișează programul?

1 52

Dimensiune A este 1

Dimensiune PachetCartiJoc este 52

Obs: Foloseste doar un octet pentru a păstra două informații: valoare și culoare

Câmpuri de biți. Exemplu

□ în aceeași structură putem combina câmpuri de biți și membri normali

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20];  
    char jud[3];  
    int cod;  
};
```

```
struct angajat {  
    struct adrese A;  
    float plata;  
    unsigned statut: 1; // activ sau intrerupt  
    unsigned plata_oras: 1; // plata cu ora  
    unsigned impozit: 3; // impozit rezultat  
};
```

Obs: definește o înregistrare despre salariat care *folosește doar un octet pentru a păstra 3 informații*: statutul, dacă este platit cu ora și impozitul.

□ fără câmpul de biți, aceste informații ar fi ocupat 3 octeți.

c. Uniuni

- ❑ tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie;
- ❑ membrii unei uniuni au de obicei tipuri diferite.

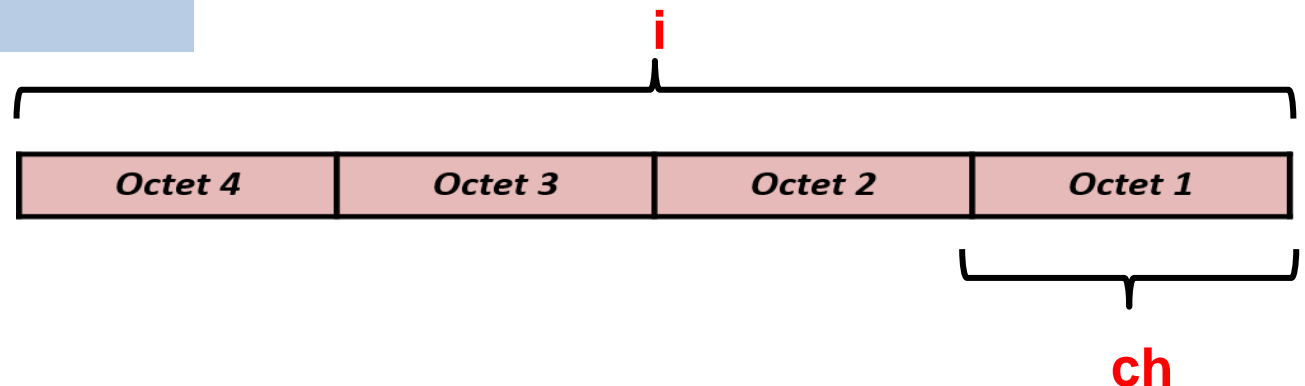
Sintaxa:

```
union <nume> {  
    < tip 1 >  <variabila 1>;  
    < tip 2 >  <variabila 2>;  
    -----  
    < tip n >  <variabila n>;  
} lista_identificatori_tip_union;
```

Uniuni. Exemplu

- ❑ tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie;
- ❑ membrii unei uniuni au de obicei tipuri diferite;
- ❑ când este declarată o variabilă de tip **union** compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia

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



Uniuni. Exemplu

main.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     union tip_u {
6         int i;
7         char ch;
8     };
9     union tip_u A;
10    printf("%d\n", (int)sizeof(A));
11
12    A.ch=10;
13    printf("%d\t%d\n", A.ch,A.i);
14
15    A.i=296;
16    printf("%d\t%d\n", A.ch,A.i);
17
18    return 0;
19 }
```

Ce afișează programul?

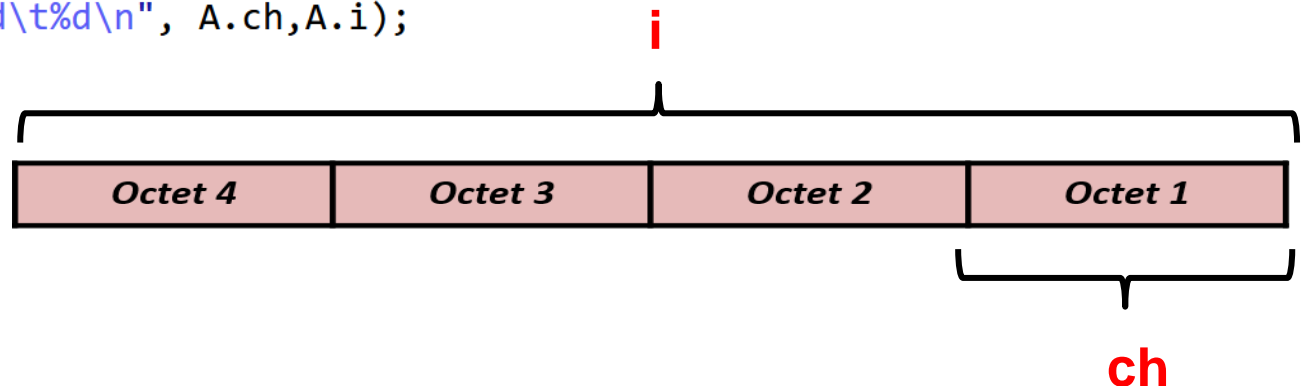
4

10

10

40

296



d. Enumerări

- ❑ mulțime de constante de tip întreg care specifică toate valorile permise pe care le poate avea o variabilă de acel tip
- ❑ atât numele generic al enumerării cât și lista de variabile sunt optionale
- ❑ constanta unui element al enumerării este fie asociată implicit, fie explicit. Implicit, primul element are asociată valoarea 0, iar pentru restul este valoarea precedentă+1.

Sintaxa:

```
enum <nume> {  
    lista enumerarilor  
} lista variabile;
```

Enumerări. Exemplu

`enum {a, b, c, d};` → a = 0, b = 1, c = 2, d = 3

`enum {a, b, c=7, d};` → a = 0, b = 1, c = 7, d = 8

`enum {a=4, b=-3, c=9, d=-8};`

main.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     enum zile {luni, marti, miercuri, joi, vineri, sambata, duminica};
6
7     printf("\tAzi este ziua cu numarul %d din saptamana.\n", joi);
8
9     return 0;
10 }
```

input

Azi este ziua cu numarul 3 din saptamana.

Enumerări. Exemplu

`enum {a, b, c=7, d};` $\rightarrow a = 0, b = 1, c = 7, d = 8$

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      enum zile {luni=1, marti, miercuri, joi, vineri, sambata, duminica};
6
7      printf("\tAzi este ziua cu numarul %d din saptamana.\n", joi);
8
9      return 0;
10 }
```

<



input

Azi este ziua cu numarul 4 din saptamana.

Enumerări. Exemplu

main.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5  enum zile {luni=1, marti, miercuri,
6             joi, vineri, sambata, duminica}azi, maine;
7  azi=joi;
8  printf("\tAzi este ziua cu numarul %d din saptamana.\n", azi);
9  maine=azi+1;
10 printf("\tMaine este ziua cu numarul %d din saptamana.\n", maine);
11 return 0;
12 }
```

input

Azi este ziua cu numarul 4 din saptamana.
Maine este ziua cu numarul 5 din saptamana.

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 unsigned int **natural**;

typedef long double **tablouNumereReale [100]** ;

tablouNumereReale a, b, c;

natural m, n, i;

Exemplu subiect examen

1. Definim regulile unui joc de carti
2. Definim cateva tipuri de date structurate care sa va ajute in implementare (carti, jucatori, etc.)
3. Se da main-ul/prototipul functiilor
4. Trebuie sa implementati functiile

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  struct carteJoc {
6      unsigned char valoare : 4;
7      unsigned char culoare : 2;
8  };
9
10 typedef struct {
11     struct carteJoc cartiJucator[4];
12     struct carteJoc cartiLuate[32];
13     int nrPuncte;
14 } jucator;
15
16 typedef struct {
17     struct carteJoc cartiPachet[32];
18     int carteCurenta;
19     int nrTotalCarti;
20 } pachetCarti;
21
22 void citesteCartiPachet(pachetCarti *p, char *numeFisier);
23 void amestecaPachet(pachetCarti *p);
24 void imparteCartiDinPachet(pachetCarti *p, jucator *j1, jucator *j2);
25 void afiseazaCarti(jucator j, char* numeJucator);
26 int joacaORunda(jucator* j1, jucator* j2, int* nrCartiJucate);
27 void numaraPunctele(jucator *j1, jucator* j2);
28
```

```
28
29  int main()
30  {
31      pachetCarti pachet;
32      char* numeFisier ="carti.in";
33      citesteCartiPachet(&pachet,numeFisier);
34      amestecaPachet(&pachet);
35      jucator Player, CPU;
36      //imparte 4 carti din pachet
37      imparteCartiDinPachet(&pachet,&Player,&CPU);
38      afiseazaCarti(Player,"Player");
39      afiseazaCarti(CPU,"CPU");
40
41
42      int cateCartiAuFostJucate=0;
43      jucator *j1=&Player,*j2=&CPU;
44      //se joaca cat timp mai sunt carti in pachet
45      while(pachet.carteCurenta < pachet.nrTotalCarti)
46      {
47          int castigatorMana = joacaORunda(j1,j2,&cateCartiAuFostJucate);
48          //cine a castigat mana
49          if(castigatorMana == 1) //a castigat j1
50              imparteCartiDinPachet(&pachet,j1,j2);
51          else
52              imparteCartiDinPachet(&pachet,j2,j1);
53      }
54      numaraPunctele(j1,j2);
55      return 0;
56
57  }
```


Cursul de azi

1. Tipuri derivate de date: structuri, uniuni, câmpuri de biți, enumerări, tipuri definite de utilizatori
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

1. instrucțiunea expresie
2. instrucțiunea vidă
3. instrucțiuni secvențiale/liniare
4. instrucțiuni decizionale/selective simple sau multiple
5. instrucțiuni repetitive/ciclice/iterative
6. instrucțiuni de salt condiționat/necon condiționat
7. instrucțiunea return

Instrucțiuni de control

- instrucțiuni compuse

- create prin combinarea instrucțiunilor de bază

- programare structurată

- **Teorema Böhm-Jacopini:** fluxul de control poate fi exprimat folosind doar trei tipuri de **instrucțiuni de control:**

- instrucțiuni **secvențiale**

- instrucțiuni **decizionale**

- instrucțiuni **repetitive**

1. Instrucțiunea expresie

- ❑ formată dintr-o expresie urmată de semnul ;
- ❑ cele mai frecvente se bazează pe expresii de atribuire, aritmetice și de incrementare / decrementare, adică expresii care au efecte secundare: schimbă valoarea unui operand

Example:

```
a=123;
```

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

```
b++;
```

- ❑ expresie vs. instrucțiune:

Expresie

```
i++
```

```
a=a-5
```

Instrucțiune

```
i++;
```

```
a=a-5;
```

2. 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
 - ❑ 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 }*

3. 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*

- ramura else poate lipsi

Instrucțiunea IF. Exemplu

Enunț: Se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

```
seminar1.c
1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int a,b;
7      scanf("%d %d",&a,&b);
8      if (a==0)
9      {
10         if (b==0)
11         {
12             printf("0 la puterea 0,caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     printf("%d \n", (int)pow(a%10,b%4+4) % 10);
24
25     return 0;
26 }
27
```

Instrucțiunea IF. Erori frecvente:

- ❑ neincluderea acoladelor:

```
if (a>b)
```

```
    a=a+b;
```

```
    b=a;           //întotdeauna se va executa instrucțiunea b=a;
```

- ❑ 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

Instrucțiuni IF imbricate

- o instrucțiune if poate conține alte instrucțiuni if pe oricare ramură
- 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");
```

Instrucțiuni IF în cascadă

- ▣ testează succesiv mai multe condiții implementând o variantă de selecție multiplă

- ▣ 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};
```

```
#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");  
    else if ( nota > 8 && nota <= 9)  
        printf("Calificativul este: Foarte bine");  
    else if ( nota > 7 && nota <= 8)  
        printf("Calificativul este: Bine");  
    else if ( nota > 5 && nota <= 7)  
        printf("Calificativul este: Suficient");  
    else printf("Calificativul este: Insuficient");  
  
    return 0;  
}
```

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

```
#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;
}
```

Rezultatul unei rulări a acestui program este:

```
Introduceti o expresie aritmetica sub forma: nr1 operator nr2: 4 + 7
Valoarea expresiei aritmetice introduse este: 11
```

Instrucțiunea SWITCH. Exemplu

Enunț: Se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

```
seminar1_2.c
1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int a,b;
7      scanf("%d %d",&a,&b);
8      if (a==0)
9      {
10         if (b==0)
11         {
12             printf("0 la puterea 0,caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     a = a%10;
24     switch (b%4)
25     {
26     case 0: printf("%d\n",a*a*a*a % 10); break;
27     case 1: printf("%d\n",a); break;
28     case 2: printf("%d\n",a*a % 10); break;
29     case 3: printf("%d\n",a*a*a % 10); break;
30     }
31     return 0;
32 }
```

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

- ❑ 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

- mod de funcționare și constrângeri (continuare):
 - 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 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

```
...
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] !");
}
}
```

Instrucțiunea SWITCH

```
seminar1_2.c x
1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int a,b;
7      scanf("%d %d",&a,&b);
8      if (a==0)
9      {
10         if (b==0)
11         {
12             printf("0 la puterea 0,caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     a = a%10;
24     switch (b%4)
25     {
26     case 0: printf("%d\n",a*a*a*a % 10);
27     case 1: printf("%d\n",a);
28     case 2: printf("%d\n",a*a % 10);
29     case 3: printf("%d\n",a*a*a % 10);
30     }
31     return 0;
32 }
```

Ce afișează programul?

12 33
2
4
8

5. 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:
 - a) instrucțiunea repetitivă cu testare inițială **while**
 - b) instrucțiunea repetitivă cu testare finală **do-while**
 - c) 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 corpul 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}
```

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<=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
```

Observație

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

Instrucțiunea WHILE

Ce afișează următoarea secvență de cod?

```
unsigned int i=3;  
while (i>=0){  
    printf("%d\n",i);  
    i--;  
}
```

CICLEAZĂ!

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

Instrucțiunea 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);
```

Rezultatul unei rulări a acestui program este:

```
    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): 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);
```

Instrucțiunea FOR

Ce afișează următoarea secvență de cod?

```
int i=0;  
for(; i<=5; i++);  
    printf("%d", i);
```

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`
 - ❑ eticheta este definită în program
 - ❑ eticheta: instrucțiune

```
    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
```

Instrucțiunile break, continue și goto

```
//Insumarea tuturor numerelor prime
dintr-un vector de 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 suma=0;
    int i;
    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);
return 0;
}
```


Instrucțiunile break, continue și goto

```
//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;
```

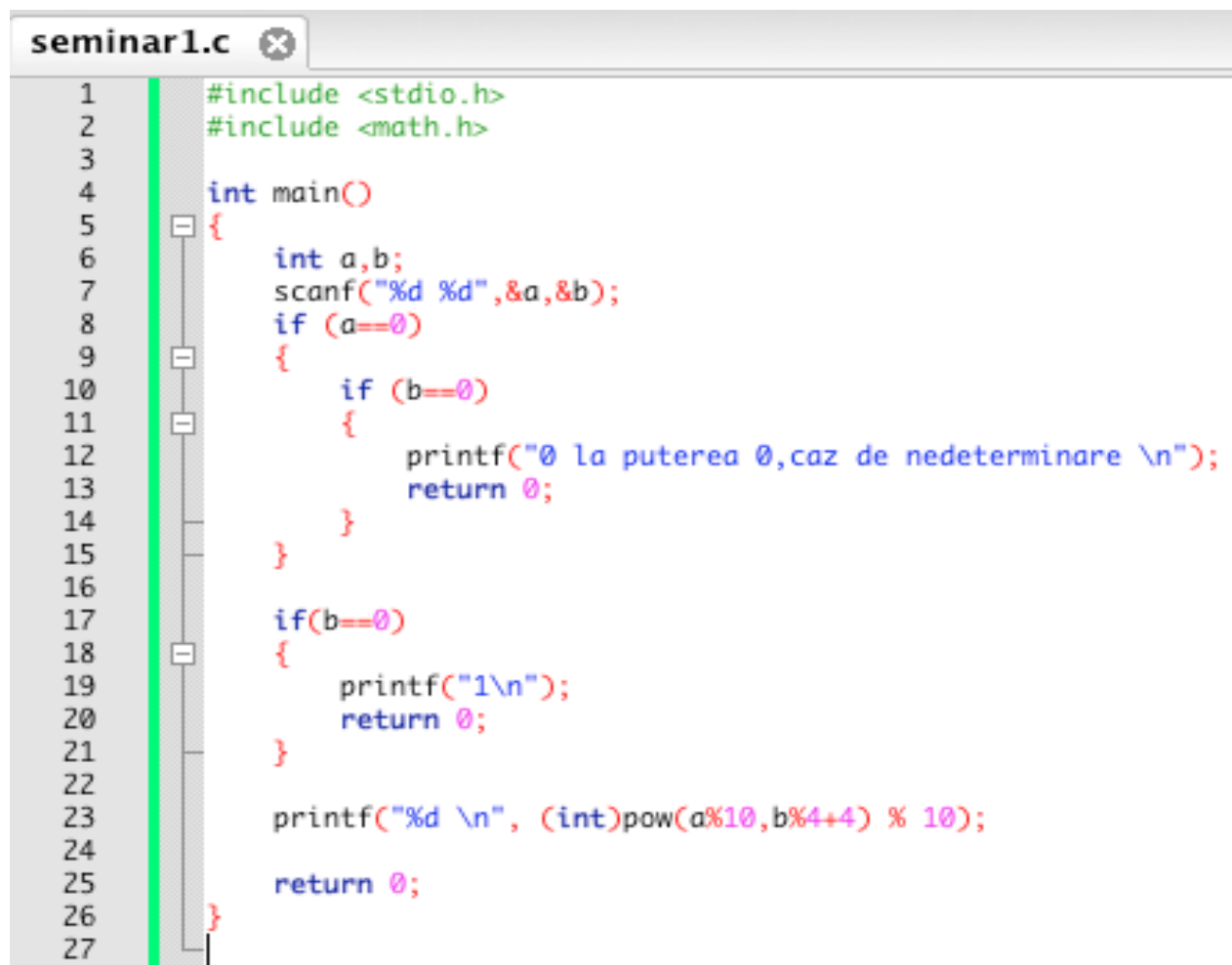
```
            int limit= (int) (sqrt(v[i])
                             +epsilon);
            while (prim && k<=limit)
            {
                if (v[i]%k==0)
                    prim=0;
                k++;
            }
            if (prim)
                suma+=v[i];
        } i++;
    }
    printf("Suma este %d",suma);
    return 0;
}
```

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;



```
seminar1.c
1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int a,b;
7      scanf("%d %d",&a,&b);
8      if (a==0)
9      {
10         if (b==0)
11         {
12             printf("0 la puterea 0,caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     printf("%d \n", (int)pow(a%10,b%4+4) % 10);
24
25     return 0;
26 }
27
```

Cursul 4

- 1. Tipuri derivate de date: structuri, uniuni, câmpuri de biți, enumerări**
- 2. Instrucțiuni de control**

Cursul 5

1. Directive de preprocesare. Macrodefiniții.
2. Funcții de citire/scriere.
3. Etapele realizării unui program C.