



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

**An universitar 2016 – 2017
Semestrul I**

Curs 6



Agenda cursului

1. Tipuri structurate de date

- Tablouri uni- si bi- dimensionale
- Structuri, uniuni, câmpuri de biți, enumerări.

2. Pointeri

- definire
- aritmetica pointerilor
- pointeri si tablouri



1. Tablouri uni si bidimensionale

Sintaxa → tablouri unidimensionale

tip nume_variabila [dimensiune];

double tab [100] ;

0.3	-1.2	10	5.7	...	0.2	-1.5	1
0	1	2	3		97	98	99

tab [3] = 5.7;

int a[5];

3	-12	10	7	1
0	1	2	3	4

a [1] = 3;

char b1 [34];

A	&	*	+	...	c	M	#
0	1	2	3	...			

b1 [1] = '&;'



1. Tablouri uni si bidimensionale

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune] → **sizeof(nume) = sizeof (tip) * dimensiune ;**

```
double tab [100] ;

int a[5];

char b1 [34];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```

C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe

```
Stocarea unui tablou de elemente double = 800 octeti
Stocarea unui tablou de elemente int = 20 octeti
Stocarea unui tablou de elemente char = 34 octeti
```



1. Tablouri uni si bidimensionale

Tablouri unidimensionale → liste de informatii de acelasi tip, stocate in locatii de memorie contigue in ordinea indicilor.

```
11  
12 int a[5];  
13  
14 for(i=0;i<5;i++)  
15 printf("Adresa elem %d din tablou = %d \n",i,&a[i]);  
16  
17  
18 C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe  
19 Adresa elem 0 din tablou = 2686728  
20 Adresa elem 1 din tablou = 2686732  
21 Adresa elem 2 din tablou = 2686736  
22 Adresa elem 3 din tablou = 2686740  
23 Adresa elem 4 din tablou = 2686744
```

```
char b1 [34];  
  
for(i=0;i<5;i++)  
printf("Adresa elem %d din tablou = %d \n",i,&b1[i]);
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe  
Adresa elem 0 din tablou = 2686694  
Adresa elem 1 din tablou = 2686695  
Adresa elem 2 din tablou = 2686696  
Adresa elem 3 din tablou = 2686697  
Adresa elem 4 din tablou = 2686698
```



1. Tablouri uni si bidimensionale

Aplicatie Citirea si afisarea elementelor unui tablou unidimensional

```
int main()
{
    int a[100]; // tabloul a cu maxim 100 de elemente
    int i; // indicele cu care parcurg tabloul
    int n; // lungimea efectiva a tabloului

    printf("Cate elem introduceti? ");
    scanf("%d", &n);

    printf("\nDati elem. sirului: \n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    printf("\nAfisarea sirului: ");
    for(i=0;i<n;i++)
        printf("%d  ",a[i]);
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\t
Cate elem introduceti? 4

Dati elem. sirului:
12
45
-78
3

Afisarea sirului: 12 45 -78 3
```



1. Tablouri uni si bidimensionale

Sintaxa → tablouri bidimensionale

tip nume_variabila [dimensiune1][dimensiune2];

int a[3][5];

0	3	-12	10	7	1
1	10	2	0	-7	41
2	-3	-2	0	0	2
	0	1	2	3	4

a [1][4] = 41;



1. Tablouri uni si bidimensionale

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune1][dimensiune2] →

sizeof(nume) = sizeof (tip) * dimensiune1 * dimensiune2 ;

```
double tab [5][10] ;

int a[5][10];

char b1[5][10];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```

C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe

```
Stocarea unui tablou de elemente double = 400 octeti
Stocarea unui tablou de elemente int = 200 octeti
Stocarea unui tablou de elemente char = 50 octeti
```




1. Tablouri uni si bidimensionale

Aplicatie Citirea si afisarea elementelor unui tablou bidimensional

```
int main()
{
    int a[5][10]; // tabloul a cu maxim 5 linii si 10 coloane
    int i,j; // i = inice pt linii, j = indice pt coloane
    int n; // numarul de linii
    int m; // numarul de coloane

    printf("Dati nr de linii si de coloane: ");
    scanf("%d%d", &n, &m);

    printf("\nDati elem. matricei: \n");
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            scanf("%d",&a[i][j]);

    printf("\nAfisarea matricei: \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
}
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe
Dati nr de linii si de coloane: 3 4

Dati elem. matricei:
1 3 2 4 7 9 0 5 7 3 5 4

Afisarea matricei:
1 3 2 4
7 9 0 5
7 3 5 4

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



1. Tablouri uni si bidimensionale

Tablouri bidimensionale → stocarea in locatii de memorie

```
int main()
{
    int a[5][10]; // tabloul a cu maxim 5 linii si 10 coloane
    int i,j; // i = inice pt linii, j = indice pt coloane
    int n; // numarul de linii
    int m; // numarul de coloane

    printf("Dati nr de linii si de coloane: ");
    scanf("%d%d", &n, &m);
}
```

Linia 0: adrese → 2686520 ... **2696556**
Linia 1: adrese → 2686560 ... **2696596**
Linia 2: adrese → 2686600 ... **2696636**

```
printf("\nAfisarea adreselor elementelor matricei: \n");
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
        printf("%d ",&a[i][j]);
    printf("\n");
}
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe
Dati nr de linii si de coloane: 3
4
Afisarea adreselor elementelor matricei:
2686520 2686524 2686528 2686532
2686560 2686564 2686568 2686572
2686600 2686604 2686608 2686612
Process returned 0 (0x0)   execution time
Press any key to continue.
-
```



1. Tablouri uni si bidimensionale

Aplicatie – Interschimbarea de linii in matrice

```
int a[10][15], linii,coloane,i,j,x,y,aux;
```

```
/*citire matrice */
```

```
scanf("%d%d",&linii,&coloane);
```

```
for(i = 0; i<linii; i++)
```

```
    for (j = 0; j<coloane; j++)
```

```
        scanf("%d",&a[i][j]);
```

```
/*afisare matrice */
```

```
for(i = 0; i<linii; i++)
```

```
{
```

```
    for (j = 0; j<coloane; j++)
```

```
        printf("%d ",a[i][j]);
```

```
    printf("\n");
```

```
}
```

```
/*interschimbare linie x cu linie y */
```

```
scanf("%d%d",&x,&y);
```

```
for ( j = 0; j<coloane; j++)
```

```
    aux = a[x][j], a[x][j] = a[y][j], a[y][j]=aux;
```



1. Tablouri uni si bidimensionale

Aplicatie – Diagonala principala si secundara in matrice patratica

```
int a[10][15], n,i,j;
scanf("%d",&n);
for(i = 0; i<n; i++)
    for (j = 0; j<n; j++)
        scanf("%d",&a[i][j]);

for(i = 0; i<n; i++)
{
    for (j = 0; j<n; j++)
        printf("%d ",a[i][j]);
    printf("\n");
}
printf("\n");
```

```
/*** diagonal principala ***/
```

```
for ( i = 0; i<n; i++)
    printf("%d ", a[i][i]);
printf("\n");
```

```
/*** diagonal secundara ***/
```

```
for ( i = 0; i<n; i++)
    printf("%d ", a[i][n - i - 1]);
printf("\n");
```

5									
1	2	3	4	5	6	7	8	9	0
1	2	3	4	5					
6	7	8	9	0					
1	2	3	4	5					
6	7	8	9	0					
1	2	3	4	5					
1	7	3	9	5					
5	9	3	7	1					



2. Tipuri structurate de date

Limbajul C → crearea tipurilor uzuale in 5 moduri

1. Structura (**struct**) – grupeaza mai multe variabile sub acelasi nume; // tip de data **compozit**
2. Campul de biti – variatiune a structurii → acces usor la bitii individuali
3. Uniunea (**union**) – face posibil ca aceleasi zone de memorie sa fie definite ca doua sau mai multe tipuri diferite de variabile
4. Enumerarea (**enum**) – lista de constante intregi cu nume
5. Tipuri definite de utilizator (**typedef**) – defineste un nou nume pentru un tip existent



2. Tipuri structurate de date

Structuri

Grup de variabile unite sub acelasi nume

Sintaxa

```
struct <nume> {  
    < tip 1 >    <variabila 1>;  
    < tip 2 >    <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista de identificatori de tip struct;
```

Variabilele care fac parte din structura sunt denumite membri ai structurii (uzual numiti *elemente* sau *campuri*).



2. Tipuri structurate de date

Structuri

```
struct <nume> {  
    < tip 1 > <variabila 1>;  
    < tip 2 > <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista de identificatori de tip struct;
```

Obs:

1. Daca numele tipului lipseste, structura se numeste **anonima**. Daca lista identificatorilor declarati lipseste, s-a definit doar tipul structura. Cel putin una dintre aceste specificatii trebuie sa existe.

2. Daca <nume> este prezent → se pot declara noi variabile de tip structura **struct <nume> <lista noilor identificatori>;**

3. Referirea unui membru al unei variabile de tip structura → operatorul de selectie **(.)** intr-o expresie care precizeaza identificatorul variabilei si al campului.



2. Tipuri structurate de date

Structuri

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

Numele **adrese** identifica aceasta structura de date particulara.

struct adrese A;

Declara o variabila de tip adrese si ii aloca memorie

Exemple

Structura din memorie pentru variabila A de tip adrese

nume 30 octeti

strada 40 octeti

oras 20 octeti

jud 3

cod 4



2. Tipuri structurate de date

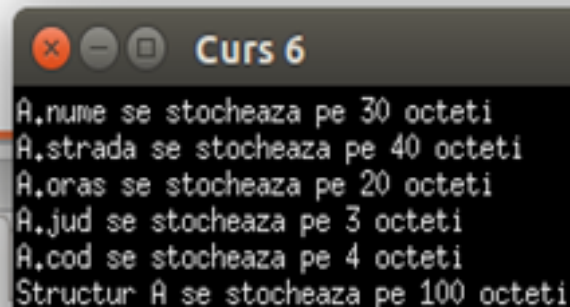
Structuri

Exemple

Compilatorul alocă memorie în plus pentru aliniere (multiplu de 4)

```
int main()
{
    struct adrese A;
    printf("A.numa se stocheaza pe %d octeti \n", sizeof(A.numa));
    printf("A.strada se stocheaza pe %d octeti \n", sizeof(A.strada));
    printf("A.oras se stocheaza pe %d octeti \n", sizeof(A.oras));
    printf("A.jud se stocheaza pe %d octeti \n", sizeof(A.jud));
    printf("A.cod se stocheaza pe %d octeti \n", sizeof(A.cod));
    printf("Structur A se stocheaza pe %d octeti \n", sizeof(A));
    return 0;
}
```

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



```
Curs 6
A.numa se stocheaza pe 30 octeti
A.strada se stocheaza pe 40 octeti
A.oras se stocheaza pe 20 octeti
A.jud se stocheaza pe 3 octeti
A.cod se stocheaza pe 4 octeti
Structur A se stocheaza pe 100 octeti
```



2. Tipuri structurate de date

Structuri

Exemple

Compilerul alocă memorie în plus pentru aliniere NUMAI CÂND
AVEM TIPURI DE DATE DIFERITE!

```
struct adrese A;  
printf("A.nume se stocheaza pe %d octeti \n", sizeof(A.nume));  
printf("A.strada se stocheaza pe %d octeti \n", sizeof(A.strada));  
printf("A.oras se stocheaza pe %d octeti \n", sizeof(A.oras));  
printf("A.jud se stocheaza pe %d octeti \n", sizeof(A.jud));  
// printf("A.cod se stocheaza pe %d octeti \n", sizeof(A.cod));  
printf("Structura A se stocheaza pe %d octeti \n", sizeof(A));  
return 0;
```

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

Curs 6

A.nume se stocheaza pe 30 octeti
A.strada se stocheaza pe 40 octeti
A.oras se stocheaza pe 20 octeti
A.jud se stocheaza pe 3 octeti
Structura A se stocheaza pe 93 octeti



2. Tipuri structurate de date

Structuri

Exemple

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

Defineste un tip de structura numit **adrese** si declara ca fiind de acest tip variabilele **A, B, C**

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

Declara o variabila numita **A** definita de structura care o precede.



2. Tipuri structurate de date

Structuri

Exemple

Accesul la membrii structurii

Se face prin folosirea
operatorului punct.

nume_variabila.nume_camp

```
A.nume = "Ionescu";  
B.Cod = 257;  
scanf("%d", &C.cod);
```

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

Atribuirea la nivel de structura

```
A.nume = "Ionescu"; A.strada="1 Mai";  
A.oras = "Bucuresti"; A.jud = "B"; A.cod = 1;
```

```
B = A;
```



2. Tipuri structurate de date

Structuri

Aplicatie

Se citesc de la tastatura / din fisier forme geometrice (triunghi, dreptunghi, cerc) date prin coordonatele (in plan) ale varfurilor (triunghi, dreptunghi), respectiv coordonatele centrului si ale unui punct de pe cerc.

1. Sa se afiseze primele n astfel de forme citite ordonate crescator dupa arie.
2. Sa se adauge / sa se stearga o forma geometrica (se definesc subprogramele ADD si DELETE).
3. Sa se afiseze doar acele forme geometrice dreptunghiulare care pot fi formate din exact 2 forme geometrice triunghiulare.



2. Tipuri structurate de date

Structuri

Aplicatie

- forme geometrice (triunghi, dreptunghi, cerc) date prin coordonatele (in plan) ale varfurilor (triunghi, dreptunghi), respectiv coordonatele centrului si ale unui punct de pe cerc.

```
struct punct
{
    int x,y;
};

struct forma_geometrica
{
    int np; // numarul de puncte
    punct *p; // pointer catre un vector de puncte
};
```



2. Tipuri structurate de date

Structuri

Aplicatie

```
int main()
{
    int n,i;
    forma_geometrica *fg; //vector de forme geometrice
    scanf("%d",&n); // numarul de forme geometrice dorite
    fg = (forma_geometrica*)malloc(n*sizeof(forma_geometrica));
    for(i = 0; i<n; i++)
        citire(&fg[i]);
    for(i = 0; i<n; i++)
        afisare(fg[i]);
    return 0;
}
```

```
"C:\Users\Ank\Desktop\Curs 10\bin\Debug\Curs 10.exe"
3
Dati tipul formei (nr de puncte): 2
Dati cele 2 puncte ce apartin cercului:
1 1 2 2
Dati tipul formei (nr de puncte): 2
Dati cele 2 puncte ce apartin cercului:
3 3 4 4
Dati tipul formei (nr de puncte): 4
Dati cele 4 puncte ce apartin dreptunghiului:
1 1 2 2 3 3 4 4
Cerc: (1,1) (2,2)
Cerc: (3,3) (4,4)
Dreptunghi: (1,1) (2,2) (3,3) (4,4)
```




2. Tipuri structurate de date

Structuri

Aplicatie

```
void citire(forma_geometrica *A)
{
    printf("Dati tipul formei (nr de puncte): ");
    scanf("%d",&(*A).np);
    (*A).p = (punct*)malloc((*A).np*sizeof(punct));
    if ((*A).np == 2)
    {
        printf("Dati cele 2 puncte ce apartin cercului: \n");
        scanf("%d%d%d%d",&(*A).p[0].x,&(*A).p[0].y,&(*A).p[1].x,&(*A).p[1].y);
    }
    if ((*A).np == 3)
    {
        printf("Dati cele 3 puncte ce apartin triunghiului: \n");
        scanf("%d%d%d%d%d%d",&(*A).p[0].x,&(*A).p[0].y,&(*A).p[1].x,&(*A).p[1].y,&(*A).p[2].x,&(*A).p[2].y);
    }
    if ((*A).np == 4)
    {
        printf("Dati cele 4 puncte ce apartin dreptunghiului: \n");
        scanf("%d%d%d%d%d%d%d%d",&(*A).p[0].x,&(*A).p[0].y,&(*A).p[1].x,&(*A).p[1].y,&(*A).p[2].x,&(*A).p[2].y,&(*A).p[3].x,&(*A).p[3].y);
    }
}
```




2. Tipuri structurate de date

Structuri

Aplicatie

```
void afisare(forma_geometrica A)
{
    if (A.np == 2)
        printf("Cerc: (%d,%d) (%d,%d) \n", A.p[0].x, A.p[0].y, A.p[1].x, A.p[1].y);
    if (A.np == 3)
        printf("Triunghi: (%d,%d) (%d,%d) (%d,%d) \n", A.p[0].x, A.p[0].y, A.p[1].x, A.p[1].y, A.p[2].x, A.p[2].y);
    if (A.np == 4)
        printf("Dreptunghi: (%d,%d) (%d,%d) (%d,%d) (%d,%d) \n", A.p[0].x, A.p[0].y, A.p[1].x, A.p[1].y, A.p[2].x,
```



2. Tipuri structurate de date

Campuri de biti

Caracteristica intrinseca a limbajului C.

Poate adauga mai multa structurare (posibil si eficienta).

Este efectiv un tip special de membru al unei structuri care defineste cat de lung trebuie sa fie campul, in biti.

Permite accesul la un singur bit.

Memorie limitata → stocarea mai multor variabile booleene intr-un singur octet.

Nu se poate obtine adresa unui camp de biti



2. Tipuri structurate de date

Campuri de biti

Forma generala

```
struct nume_generic {  
    tip nume_1 : lungime;  
    tip nume_2 : lungime;  
    .....  
    tip nume_n : lungime;  
} lista_variabile;
```

Tipul campului de biti poate fi doar: **int**, **unsigned** sau **signed**.

Campul de biti cu lungimea 1 → unsigned (un singur bit nu poate avea semn).

Unele compilatoare → doar unsigned.

Lungime → nr de biti dintr-un camp



2. Tipuri structurate de date

Campuri de biti

Exemplu

Restul impartirii la 16 dat prin valoare: 0 – 15 si tip: litera / cifra

```
struct rest_baza_16
{
    unsigned char valoare;
    unsigned char tip; // litera - codificata cu 1 sau cifra - 0
};

int main()
{
    struct rest_baza_16 A;
    printf("Structura A se stocheaza pe %d octeti \n", sizeof(A));
}

/*
struct adrese
{
    char nume[30];
}
```

Curs 6
Structura A se stocheaza pe 2 octeti



2. Tipuri structurate de date

Campuri de biti

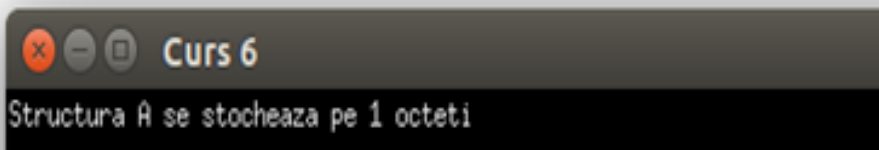
Exemplu

Restul impartirii la 16 dat prin valoare: 0 – 15 si tip: litera / cifra

```
struct rest_baza_16
{
    unsigned char valoare:4;
    unsigned char tip:1; // litera - codificata cu 1 sau cifra - 0
};
```

```
int main()
{
    struct rest_baza_16 A;
    printf("Structura A se stocheaza pe %d octeti \n", sizeof(A));
}

/*
struct adrese
{
    char nume[30];
```





2. Tipuri structurate de date

Campuri de biti

In aceeaasi structura pot fi combinate campuri de biti si membri normali.

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

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

Obs: Defineste o inregistrare despre salariat care foloseste doar un octet pentru a pastra 3 informatii: statutul, daca este angajat si impozitul.

Fara campul de biti, aceste informatii ar fi ocunat 3 octeti.



2. Tipuri structurate de date

Campuri de biti

Restrictii:

Nu se poate obtine adresa unui camp de biti.

Nu pot fi introduse in matrice.

Nu se stie daca vor fi rulate de la stanga la dreapta sau de la dreapta la stanga → depinde de echipament.



2. Tipuri structurate de date

Uniuni

Def: Locatie de memorie impartita in momente diferite intre doua sau mai multe variabile diferite, in general de tipuri diferite.

Forma generala

```
union nume_generic {  
    tip nume_1;  
    tip nume_2;  
    .....  
    tip nume_n;  
} variabile_uniuni;
```

Exemplu

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

union tip_u A;

In variabila A i si ch impart aceeaasi zona de memorie.

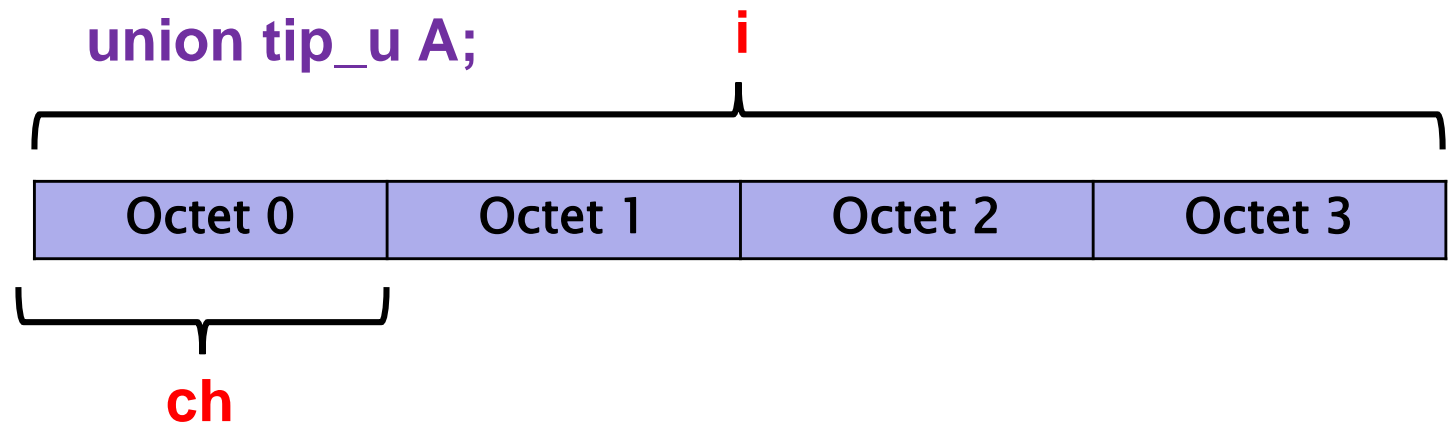
Cand este declarata o variabila de tip **union** compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia



2. Tipuri structurate de date

Uniuni

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



Utilizate:

Cand sunt necesari specificatori speciali de conversie (exemplu: utilizare union pentru a manevra bitii care formeaza o data **double**, pentru a-l modifica precizia sau a face anumite rotunjiri.



2. Tipuri structurate de date

Uniuni

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

int main()
{
    union tip_u A;
    printf("Union A se stocheaza pe %d octeti \n", sizeof(A));

    A.ch = '1';
    printf("A.ch = %d si A.i = %d \n", A.ch, A.i);

    A.i = 300;
    printf("A.ch = %d si A.i = %d \n", A.ch, A.i);
}
```

```
Curs 6
Union A se stocheaza pe 4 octeti
A.ch = 49 si A.i = 346498097
A.ch = 44 si A.i = 300
```



2. Tipuri structurate de date

Enumerari

Set de constante de tip intreg care specifica toate valorile permise pe care le poate avea o variabila de acel tip.

```
enum nume_generic { lista enumerarilor } lista_variabile;
```

Atat numele generic al enumerarii cat si lista de variabile sunt optionale.

Constanta unui element al enumerarii este fie asociata implicit, fie explicit. Implicit, primul element are asociata valoarea 0, iar pentru restul este valoarea_precedenta+1.



2. Tipuri structurate de date

Enumerari

Set de constante de tip intreg care specifica toate valorile permise pe care le poate avea o variabila de acel tip.

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};



2. Tipuri structurate de date

Enumerari

Declararea unei matrice de siruri si folosirea valorii enumerarii pe post de indice.

```
enum monede { penny, nickel, dime, quarter, jumătate_dolar, dolar };  
  
enum monede bani;  
  
int main()  
{  
    char nume[][15] = {"penny", "nickel", "dime", "quarter", "jumătate_dolar", "dolar"};  
    printf("%s", nume[bani]);  
}
```



Codul lucreaza corect doar daca nu este initializat nici un simbol, deoarece matricea de siruri trebuie sa aiba indici incepand cu 0.



2. Tipuri structurate de date

Enumerari

```
enum zile {luni, marti = 200, miercuri, joi, vineri, sambata, duminica};
```

```
int main()
```

```
{  
    printf("Ziua %d a saptamanii \n", luni);  
    printf("Ziua %d a saptamanii \n", marti);  
    printf("Ziua %d a saptamanii \n", miercuri);  
    printf("Ziua %d a saptamanii \n", joi);  
}
```

```
/*
```

```
enum monede { penny, nick
```

```
enum monede bani;
```

```
Curs 6  
Ziua 0 a saptamanii  
Ziua 200 a saptamanii  
Ziua 201 a saptamanii  
Ziua 202 a saptamanii
```



2. Tipuri structurate de date

Specificatorul typedef

Definirea explicita a noi tipuri de date.

Nu se declara o variabila sau o functie de un anumit tip, ci se asociaza un nume (sinonimul) tipului de date.

Sintaxa

typedef <definiție tip> <identificator>;

Exemple:

```
typedef unsigned int natural ;  
typedef long double tablou [100] ;  
tablou a, b, c;  
natural m,n,i;
```



3. Pointeri

Pointerii sunt foarte folositi in C deoarece:

Ofera posibilitatea de a modifica argumentele de apelare a functiilor

```
// void modificare (int *a, float b) {...}
```

Permit o alocare dinamica

```
// int *a = malloc (10 * sizeof(int));
```

Pot imbunatati eficienta anumitor rutine

Atentie la: Pointerii neinitializati.
Pointerii care contin valori neadeccvate.

Pot determina blocarea sistemului.
E usor sa fie folositi incorect → erori greu de depistat.



3. Pointeri

Un **pointer** este o variabila (alocata “static”) care contine o adresa de memorie.

Aceasta adresa este localizarea in memorie a unui alt obiect(de obicei o alta variabila) → Daca o variabila contine adresa alteia, prima se spune ca este **un pointer la** (indica pe) cea de-a doua.

Adresa de memorie	Variabila de memorie
1000	1003
1001	
1002	
1003	
1004	



Variable de tip pointer

Forma generala **tip** * **nume**;

variabila nume → adrese de zone de memorie alocate unor date de tipul tip.

* - operator de indirectare

semnifica faptul ca variabila este pointer la tipul respectiv.



3. Pointeri

Un pointer de tip **void** reprezinta doar o adresa de memorie a unui obiect oarecare:

- dimensiunea zonei de memorie indicate si interpretarea informatiei continute, nu sunt definite;
- poate apare in atribuirii, in ambele sensuri, cu pointeri de orice alt tip;
- poate fi convertit la orice alt tip de pointer fara a folosi conversie explicită de tip cu operatorul **cast** : **(tip *)** – **Nu e valabil pt celelalte tipuri!**



3. Pointeri

Conversie **void** < -- > tip (fara cast)

```
int x,*p,*r;
void *q;
p = &x;
printf("Pointer catre adresa lui x: %p \n",p);
q = p;
printf("Pointer catre int (p) -> pointer catre void (q): %p \n",q);
r = q;
printf("Pointer catre void (q) -> pointer catre int (r): %p \n",r);
```

```
Pointer catre adresa lui x: 0028FF10
Pointer catre int (p) -> pointer catre void (q): 0028FF10
Pointer catre void (q) -> pointer catre int (r): 0028FF10
```

Conversie **tip1** < -- > tip2

```
int x,*p,*r;
double *q;
p = &x;
printf("Pointer catre adresa lui x: %p \n",p);
q = (double *)p;
printf("Pointer catre int (p) -> pointer catre float (q): %p \n",q);
r = (int *)q;
printf("Pointer catre float (q) -> pointer catre int (r): %p \n",r);
```

```
Pointer catre adresa lui x: 0028FF10
Pointer catre int (p) -> pointer catre void (q): 0028FF10
Pointer catre void (q) -> pointer catre int (r): 0028FF10
```



3. Pointeri

Operatori speciali pentru pointeri: **&** si *****

& (operator unar) - adresa de memorie a operandului sau

Adresa variabilei x	Valoarea variabilei x
2686748	279

```
int x = 279;  
  
printf("Adresa lui x = %d \n\n\n", &x);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe
Adresa lui x = 2686748

*** (operator unar)** - complementul lui &; returneaza valoarea inregistrata la adresa care ii urmeaza

```
int x = 279;  
  
printf("Valoarea de la adresa lui x = %d \n\n\n", * &x);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe
Valoarea de la adresa lui x = 279



3. Pointeri

Adrese de memorie

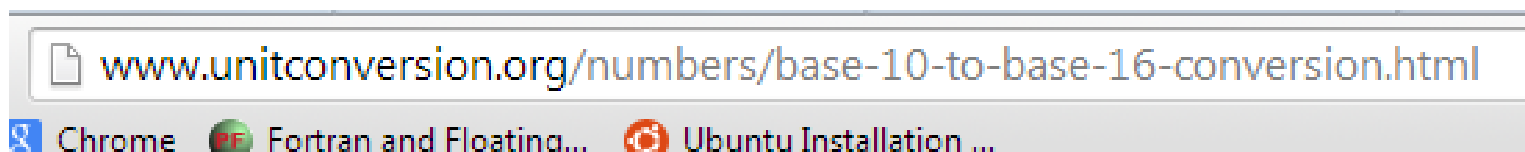
In C → specificator de format special pentru tiparirea valorilor reprezentand adresele de memorie.

%p

```
int a;  
printf("\n adresa lui a este %p ", &a);  
printf("\n adresa lui a in DECIMAL este %d", &a);  
printf("\n adresa lui a in HEXA este %x\n\n", &a);  
  
return 0;  
}
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
adresa lui a este 0028FF1C  
adresa lui a in DECIMAL este 2686748  
adresa lui a in HEXA este 28ff1c
```



(este o variantă portabilă).

base-10:

2686748

base-16:

28FF1C

Valorile și formatul adreselor de memorie depind de arhitectura calculatorului și de sistemul de operare sub care rulează.



3. Pointeri

Important!

Variabilele de tip pointer trebuie sa indice corect **tipul de date**.

Expl: pointer de tip int → compilatorul intelege ca la adresa pe care o contine pointerul exista o variabila de tip intreg si nu de alt tip.

```
float x,y;  
int *p;  
  
x = 12.34;  
p = &x;  
  
y = *p;  
  
printf("%f \n\n", y);
```

C:\Users\Ank\Desktop\Lab6\bin\
1095069824.000000

Inc corect

```
float x,y;  
float *p;  
  
x = 12.34;  
p = &x;  
  
y = *p;  
  
printf("%f \n\n", y);
```

C:\Users\Ank\Desktop\Lab6\bin\De
12.340000

Corect



3. Pointeri

Obs. Compilatorul nu da mesaj de eroare (eventual avertisment).

Deoarece p este pointer de tip intreg, vor fi transferati in y doar 4 octeti din informatie, nu toti 8 care formeaza in mod normal un numar in virgula mobila.

```
float x,y;  
int *p;  
  
x = 12.34;  
p = &x;  
  
y = *p;  
  
printf("%f \n\n", y);
```

C:\Users\Ank\Desktop\Lab6\bin\
1095069824.000000

Inc corect

```
float x,y;  
float *p;  
  
x = 12.34;  
p = &x;  
  
y = *p;  
  
printf("%f \n\n", y);
```

C:\Users\Ank\Desktop\Lab6\bin\De
12.340000

Corect



3. Pointeri

Referirea valorii unei variabile prin indirectare

```
int x, *p;

x = 419;
printf("Valoarea initiala a lui x = %d \n\n", x);

p = &x;

*p = -258;

printf("Valoarea lui x dupa indirectare = %d \n", x);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

Valoarea initiala a lui x = 419

Valoarea lui x dupa indirectare = -258

Programul asigneaza lui x o valoare INDIRECT, folosind pointerul p !



3. Pointeri

Instructiuni de atribuire pentru pointeri

```
int x, *p1, *p2;  
  
x = 419;  
p1 = &x;  
p2 = p1;  
  
printf("Adresa lui x prin &x = %p \n", &x);  
printf("Adresa lui x prin p1 = %p \n", p1);  
printf("Adresa lui x prin p2 = %p \n", p2);
```

p2 indica adresa variabilei initiale x.

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
Adresa lui x prin &x = 0028FF14  
Adresa lui x prin p1 = 0028FF14  
Adresa lui x prin p2 = 0028FF14
```

Asignarea valorii 888 lui x prin p2.

Toate valorile sunt modificate!

```
int x, *p1, *p2;
```

```
x = 419;  
p1 = &x;  
p2 = p1;  
*p2 = 888;
```

```
printf("Valoarea lui x = %d \n", x);  
printf("Valoarea de la p1 = %d \n", *p1);  
printf("Valoarea de la p2 = %d \n", *p2);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
Valoarea lui x = 888  
Valoarea de la p1 = 888  
Valoarea de la p2 = 888
```



3. Pointeri

Aritmetica pointerilor

Utilizare pointeri:

- expresii aritmetice
- asignari
- comparatii.

Nu toti operatorii pot avea pointeri ca operanzi!

Asupra pointerilor pot fi realizate operatii:

- incrementare (++), decrementare (--)
- adaugare (+ sau +=) sau scadere a unui intreg (- sau -=)
- scadere a unui pointer din alt pointer.



3. Pointeri

Aritmetica pointerilor

Initializarea pointerului *pv cu adresa primului element al unui tablou

```
int v[5];  
int *pv;  
  
pv = v; //  
  
printf("Adresa primului elem = %p \n\n", pv);  
  
pv = &v[0];  
  
printf("Adresa lui v[0] = %p \n\n", pv);
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

Adresa primului elem = 0028FF08

Adresa lui v[0] = 0028FF08

```
int *pv = v;  
pv = &v[0];
```

Adresa celorlalte
elemente ale vectorului:

```
for(i=0;i<5;i++)  
printf("&v[%d] = %p \n", i, &v[i]);
```

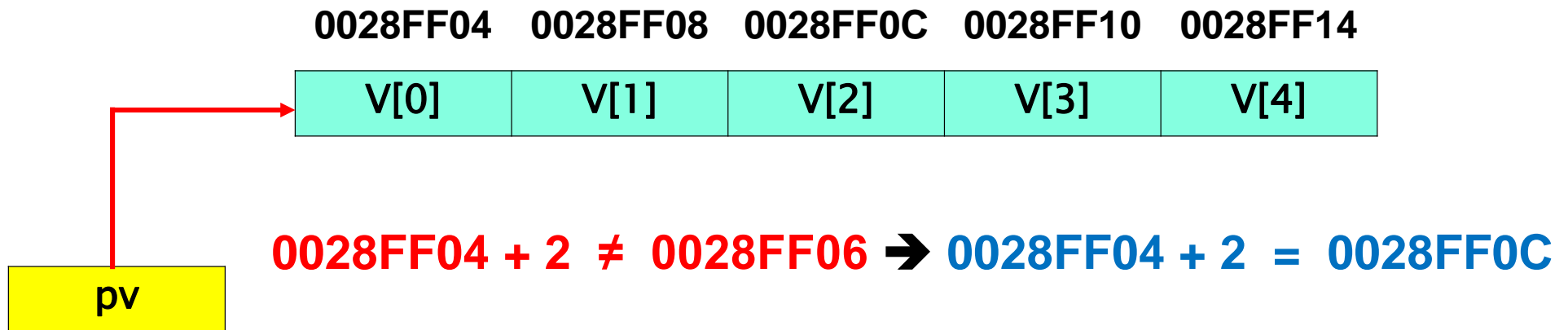
C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab

```
&v[0] = 0028FF04  
&v[1] = 0028FF08  
&v[2] = 0028FF0C  
&v[3] = 0028FF10  
&v[4] = 0028FF14
```

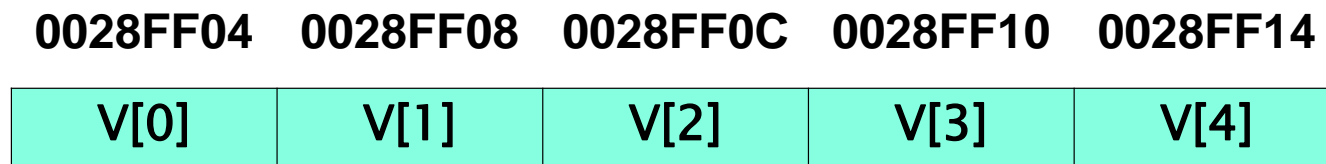


3. Pointeri

Aritmetica pointerilor



In aritmetica pointerilor adaugarea unui intreg la o adresa de memorie are ca rezultat o noua adresa de memorie!



```
int *pv=v;  
pv += 2;  
printf("pv = %p\n", pv);
```



3. Pointeri

Aritmetica pointerilor

```
int *pv=v;  
pv += 2;  
printf(" Adresa din pv dupa operatia pv += 2: %p\n", pv);  
pv -= 4;  
printf(" Adresa din pv dupa operatia pv -= 4: %p\n", pv);  
pv++;  
printf(" Adresa din pv dupa operatia pv++: %p\n", pv);  
++pv;  
printf(" Adresa din pv dupa operatia ++pv: %p\n", pv);  
pv--;  
printf(" Adresa din pv dupa operatia pv--: %p\n", pv);  
--pv;  
printf(" Adresa din pv dupa operatia --pv: %p\n", pv);
```

+ 8 bytes
-16 bytes
+ 4 bytes
+ 4 bytes
- 4 bytes
- 4 bytes

```
int *pv2 = &v[4];  
printf(" Rezultatul operatiei pv2 - pv: %d\n", pv2 - pv);
```

**diferenta = nr de obiecte
de acelasi tip care
despart cele 2 adrese**

```
Adresa din pv dupa operatia pv += 2: 0028FF08  
Adresa din pv dupa operatia pv -= 4: 0028FEF8  
Adresa din pv dupa operatia pv++: 0028FEFC  
Adresa din pv dupa operatia ++pv: 0028FF00  
Adresa din pv dupa operatia pv--: 0028FEFC  
Adresa din pv dupa operatia --pv: 0028FEF8  
Rezultatul operatiei pv2 - pv: 6
```



3. Pointeri

Aritmetica pointerilor – Compararea pointerilor

In general utilizata cand 2 sau mai multi pointeri indica acelasi obiect.

```
int x,y;  
int *px,*py;  
px = &x; py = &y;  
printf(" Adresa indicata de px: %p \n",px);  
printf(" Adresa indicata de py: %p \n",py);  
if(px<py)  
    printf("px indica o memorie mai mica decat py");  
else  
    printf("py indica o memorie mai mica decat px");
```

```
Adresa indicata de px: 0028FF14  
Adresa indicata de py: 0028FF10  
py indica o memorie mai mica decat px
```



3. Pointeri

Aritmetica pointerilor – Pointeri si tablouri

Reamintire:

1. Initializarea pointerului *pv cu adresa primului element al unui tablou

int *pv = v; pv = &v[0];

1. Adresarea celui de-al x-lea element din vectorul v

***(pv + x)**



Valoarea celui de-al x-lea element din vectorul v

***(pv + x) = v[x];**

```
int v[5]={10,20,30,40,50};  
int *pv = v;  
printf(" *(pv+2) = %d \n",*(pv+2));  
printf(" v[2] = %d \n",v[2]);
```

```
*(<pv+2>) = 30  
v[2] = 30
```



3. Pointeri

Aritmetica pointerilor – Pointeri si tablouri

1. $*(pv+x) \Leftrightarrow v[x]$
2. $\&v[x] = pv + x$
2. Daca pv este un pointer, acesta poate fi folosit cu un indice in expresii: $pv[i] = *(pv+i)$.

Concluzie: o expresie cu tablou si indice este echivalenta cu una scrisa ca pointer si distanta de deplasare.

Diferenta intre un nume de tablou si un pointer:

Un pointer este o variabila: $pv = v$ si $pv++$ **sunt expresii legale**

Un nume de tablou nu este o variabila: $v = pv$ si $v++$ **sunt expresii ilegale**



3. Pointeri

Indirectare multipla (pointeri catre pointeri)

Un pointer indica un al doilea pointer care indica o valoare tinta.

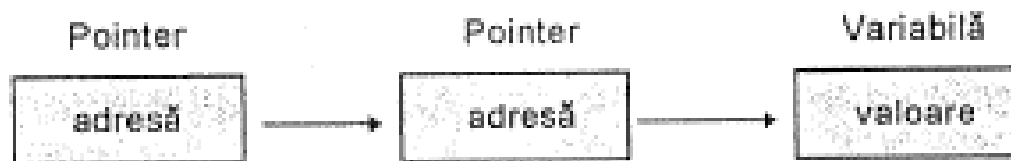
Nu se recomanda continuarea indirectarii. Rareori e necesar mai mult decat un pointer catre un pointer.

Declarare:

tip ** variabila;



Indirectare simplă



Indirectare multiplă



3. Pointeri

Indirectare multipla (pointeri catre pointeri) - Exemplu

```
int x, *p, **q;  
  
x = 10;  
p = &x;  
q = &p;  
  
printf("Adresa lui x stocata in p = %p \n\n", p);  
printf("Adresa adresei lui x stocata in q = %p \n\n", q);  
printf("Valoarea de la adresa adresei lui x prin **q = %d \n\n", **q);
```

```
Adresa lui x stocata in p = 0028FF18  
Adresa adresei lui x stocata in q = 0028FF14  
Valoarea de la adresa adresei lui x prin **q = 10
```



3. Pointeri

Aritmetica pointerilor – Aplicatii

Cum functioneaza urmatorul program?

```
int main()
{
    int i = 1, j = 5, *p = &i;
    *p = 2;
    (*(p = &j))++;
    printf("%d %d\n", i, j);

    return 0;
}
```



3. Pointeri

Aritmetica pointerilor – Aplicatii

Rezolvare:

```
int i = 1, j = 5;

int *p = &i;
printf(" *p = &i pointerul p are ca valoare adresa lui i \n");

*p = 2;
printf(" *p = 2 --> valoarea lui i devine %d prin indirectare \n",i);

(*(p = &j))++;
printf(" (*(p = &j))++ --> valoarea lui j se incrementeaza prin idirectare\n");

printf("Final: i = %d   j = %d",i,j);
```

```
*p = &i pointerul p are ca valoare adresa lui i
*p = 2 --> valoarea lui i devine 2 prin indirectare
(*(p = &j))++ --> valoarea lui j se incrementeaza prin idirectare
Final: i = 2      j = 6
```



3. Pointeri

Aritmetica pointerilor – Aplicatii

Cum functioneaza urmatorul program

```
int main()
{
    int v[10], n, i;
    int *pv;

    pv = v;

    scanf("%d", &n);
    for(i = 0; i < n; i++)
        scanf("%d", pv+i);

    for(i = 0; i < n; i++)
    {printf("%d ", *pv);
     pv++;}
```



3. Pointeri

Rezolvare:

Aritmetica pointerilor – Aplicatii

```
int main()
{
    int v[10],n,i;
    int *pv;

    pv = v;

    scanf("%d", &n);
    for(i = 0; i < n; i++)
        scanf("%d", pv+i);

    for(i = 0; i < n; i++)
        {printf("%d ",*pv);
         pv++;}
```

```
C:\Users\Ank\Desktop>
5
1
5
2
6
7
```

```
int main()
{
    int v[10],n,i;
    int *pv;

    pv = v;

    scanf("%d", &n);
    for(i = 0; i < n; i++)
        //      scanf("%d", pv+i);
        scanf("%d", &v[i]);

    for(i = 0; i < n; i++)
        //      {printf("%d ",*pv);
        //      pv++;}
        printf("%d ", v[i]);
    return 0;
}
```



3. Pointeri

Probleme ale pointerilor – Erori uzuale

Nimic nu va crea mai multe probleme decat un pointer gresit!

1. Pointerul neinitializat

```
int main()
{

    int x, *p;

    x = 10;
    *p = x;

    return 0;
}
```

Valoarea 10 este atribuita unei adrese necunoscute (pointerului p nu i s-a dat o valoare)

Valoarea lui x este scrisa intr-o adresa de memorie necunoscuta.

Problema scapa de obicei neobservata cand programul este mic, deoarece sunt sanse foarte mari ca p sa acceseze o adresa sigura, care nu afecteaza sistemul.



3. Pointeri

Probleme ale pointerilor – Erori uzuale

2. Neintelegerea modului de folosire a unui pointer

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

int main()
{
    int x, *p;

    x = 10;
    p = x;
    printf("%d", *p);
    return 0;
}
```

C:\Users\Ank\Desktop\Lab6\bin\De

Lab6.exe

Lab6.exe has stopped work

Windows can

```
int main()
{
    int x, *p;

    x = 10;
    *p = x;
    printf("%d", *p);
    return 0;
}
```

C:\Users\

10
Process r
Press any

```
int main()
{
    int x, *p;

    x = 10;
    p = &x;
    printf("%d", *p);
    return 0;
}
```

C:\Users\

10
Process r
Press any



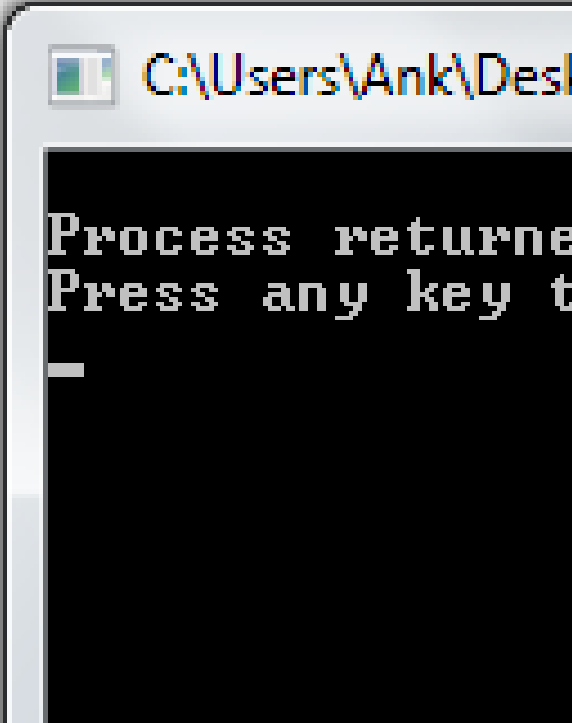
3. Pointeri

Probleme ale pointerilor – Erori uzuale

3. Presupunere incorecta asupra plasarii variabilelor

```
int main()
{
    char s[80], y[80];
    char *p1, *p2;

    p1=s;
    p2 = y;
    if (p1<p2)
    printf("P1 < P2");
    return 0;
}
```





3. Pointeri

Probleme ale pointerilor – Erori uzuale

4. Incrementarea pointerului peste limitele tablourilor

```
int main()
{
    int A[10], B[10];
    int *p, t;

    p = A;
    for(t = 0; t<20; t++)
        *p++ = t;
    return 0;
}
```

Presupunerea falsa: doua tablouri alaturate pot fi indexate ca unul singur prin simpla incrementare a pointerului peste granitele tablourilor.

Nu intotdeauna cele 2 tablouri vor fi declarate la adrese de memorie alaturate.

```
printf("Adresa sirului A: %d \n\n", &A[0]);
printf("Adresa sirului B: %d \n\n", &B[0]);
/*
```



3. Pointeri

Alocare dinamica – Tablouri uni si multi-dimensionale

```
int main() {  
  
    int *v, n, i;  
    printf("Dimensiunea efectiva = ");  
    scanf("%d", &n);  
  
    //    v = (int) malloc ( n *sizeof(int));  
  
    printf("\n Citirea elementelor sirului: \n");  
    for(i=0; i<n; i++)  
        scanf("%d", &v[i]);  
  
    printf("\n Afisarea elementelor sirului: \n");  
    for(i=0; i<n; i++)  
        printf("%d ", v[i]);  
  
    return 0;  
}
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
Dimensiunea efectiva = 4  
Citirea elementelor sirului:  
1
```

Lab6.exe

Lab6.exe has stopped working

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
Dimensiunea efectiva = 4  
Citirea elementelor sirului:  
1  
2  
3  
4  
  
Afisarea elementelor sirului:  
1 2 3 4
```

```
v = (int) malloc ( n *sizeof(int));  
  
printf("\n Citirea elementelor sirului: \n");  
for(i=0; i<n; i++)  
    scanf("%d", &v[i]);
```



3. Pointeri

Alocare dinamica – Tablouri uni si multi-dimensionale

```
int **a, n, m, i, j;
printf("nr linii = ");
scanf("%d", &n);
printf("nr coloane = ");
scanf("%d", &m);

a = (int *) malloc ( n *sizeof(int));

for(i=0; i<n; i++)
{
    a[i] = (int) malloc(m*sizeof(int));
    for (j = 0; j<m; j++)
        scanf("%d", &a[i][j]);
}

printf("\n Afisarea elementelor matricei: \n");
for(i=0; i<n; i++)
{
    for (j = 0; j<m; j++)
        printf("%d ", a[i][j]);
    printf("\n");
}
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.e

```
nr linii = 3
nr coloane = 4
1
2
3
4
5
6
7
8
9
0
1
2

Afisarea elementelor matricei:
1 2 3 4
5 6 7 8
9 0 1 2
```



Concluzii

1. S-au prezentat notiunile introductive referitoare la tipuri

structurate de date:

- Tablouri uni- si bi-dimensionale**
- Structuri**
- Enumerari**
- Campuri de biti**



Perspective

Cursul 7:

1. Siruri de caractere

2. Functii

- Declarare si definire. Apel. Transmiterea parametrilor**
- Pointeri la functii**