TIPURI DE DATE ABSTRACTE
(DEFINITE DE CĂTRE PROGRAMATOR)

STRUCTURI

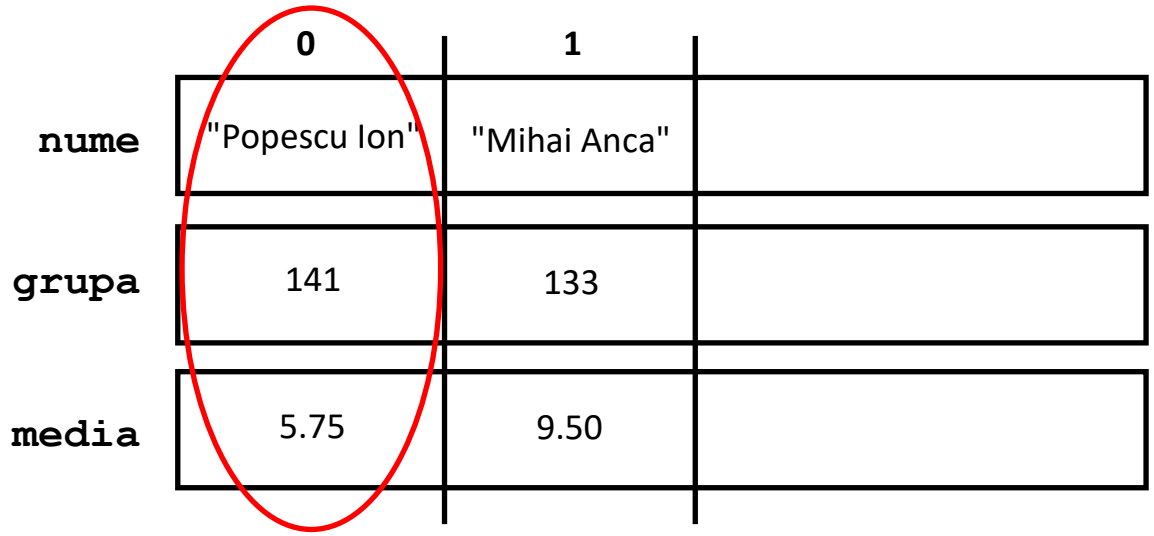
Date: Entitatea Student –> atributele (*nume, grupa, media*)

Trebuie să scriu un program care să-mi furnizeze informații despre promovabilitatea studenților dintr-o facultate.

- ***nume* –> șir de caractere**
- ***grupa* –> număr întreg**
- ***media* –> număr real**

Modelare 1: câte un tablou pentru fiecare atribut

```
char nume[100][21];
int grupa[100];
float media[100];
```



Student 0 -> (nume[0], grupa[0], media[0])

Student 1 -> (nume[1], grupa[1], media[1])

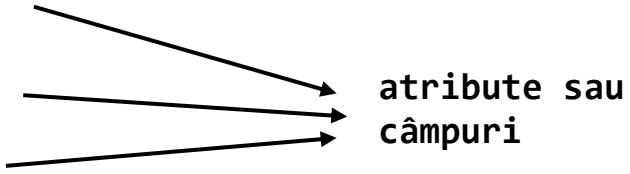
.....

Student i -> (nume[i], grupa[i], media[i])

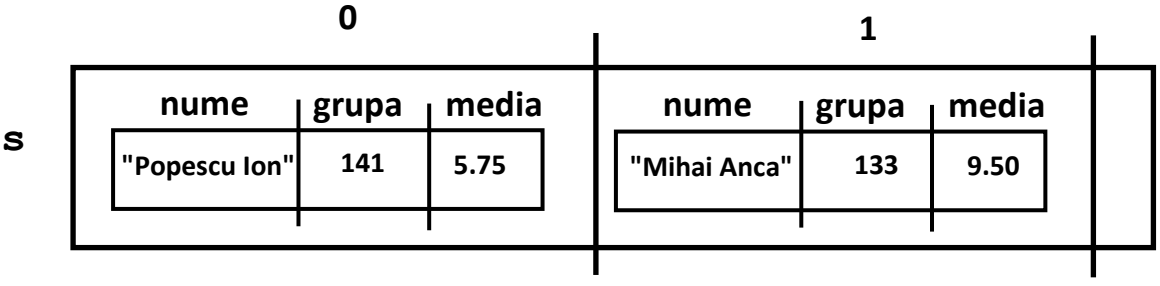
Modelare 2: un tablou de structuri

Structură = tip de date abstract (definit de către programator) care permite grupare unor date, posibil de tipuri diferite, într-un singur tip de date nou

```
struct Student
{
    char nume[21];
    int grupa;
    float media;
};
```




```
struct Student s[100]; //un tablou cu 100 de
                       //elemente de tip Student
```



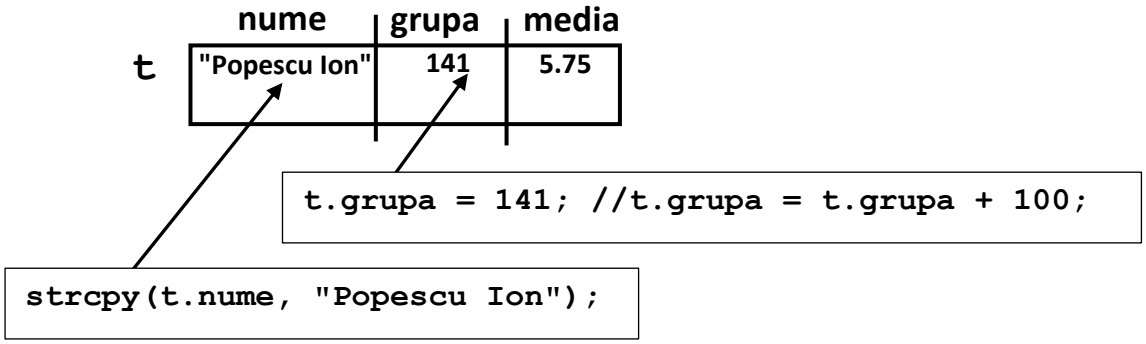
Student 0 -> s[0] -> (s[0].nume, s[0].grupa, s[0].media)

Student 1 -> s[1] -> (s[1].nume, s[1].grupa, s[1].media)

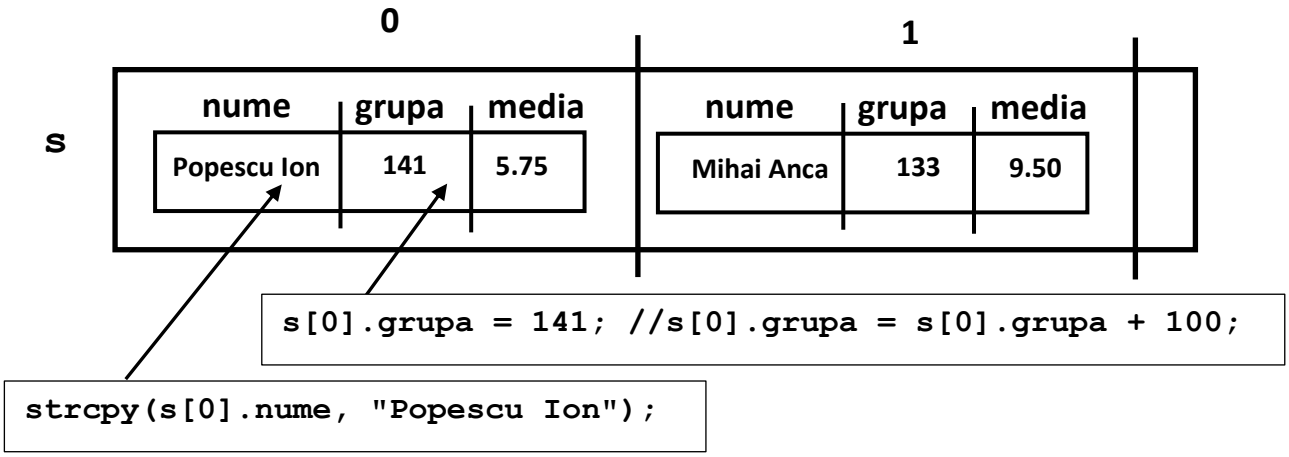
……

Student i -> s[i] -> (s[i].nume, s[i].grupa, s[i].media)

- `struct Student t;`
-



- `struct Student s[100];`



- **Cuvântul cheie `typedef` – permite definirea unui tip nou de date:**

```
typedef unsigned int UINT;
typedef int VECTOR[100];
typedef float MATRICE[100][20];
.............................
UINT x = 10;
unsigned int y = 20;
```

```
VECTOR v;
MATRICE a;
```

```
typedef struct
{
     char nume[21];
     int grupa;
     float media;
}Student;
.....................
Student s; //nu mai este nevoie de struct în față
```

Structură anonimă!!!

Typedef redenumește structura anonimă în "Student"!!!

➢ **Declararea unor variabile în momentul definirii unei structuri:**

```
struct Student
{
     char nume[21];
     int grupa;
     float media;
}s,t; //s și t sunt două variabile de tip Student
```

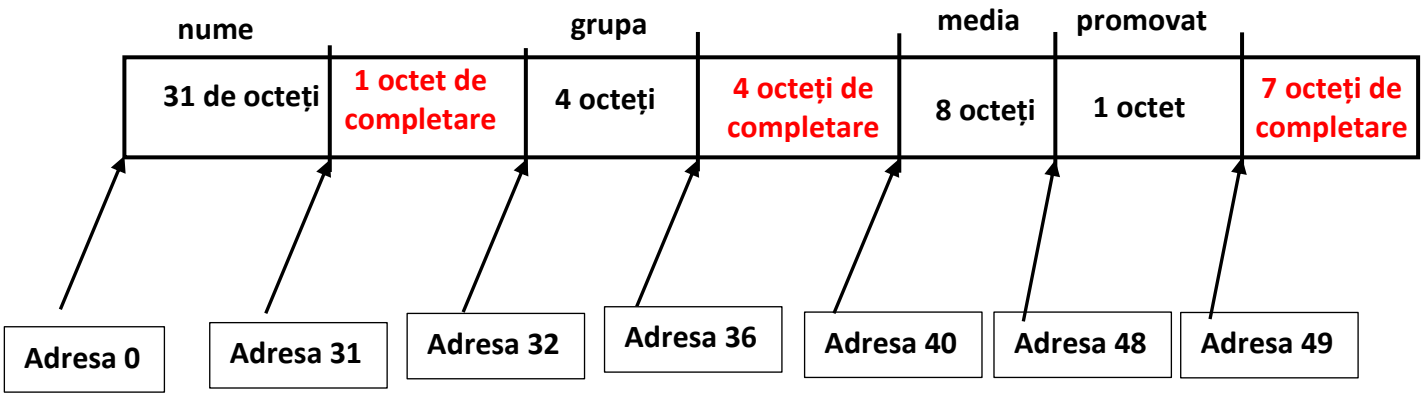
➢ **Dimensiunea variabilelor de tip structură**

Reguli de aliniere în memorie a câmpurilor unei structuri:

1. **primul câmp se consideră că începe la adresa 0;**
2. **orice câmp trebuie să înceapă la o adresă multiplu de dimensiunea sa în octeți, eventual adăugând octeți de completare înaintea sa (padding);**
3. **dimensiunea unei structuri trebuie să fie multiplu de dimensiunea maximă în octeți a unui câmp al său, eventual adăugând octeți de completare (padding) la sfârșitul său.**

Exemplu:

```
typedef struct
{
    char nume[31];   //31 de octeti
    int grupa;       //4 octeti
    double media;    //8 octeti
    char promovat;   //1 octet
}Student;            //total = 31+4+8+1 = 44 de octeti
```



`sizeof(Student) = 31+1+4+4+8+1 = 49 de octeți`

49 NU este multiplu de `sizeof(media)`=8, deci se vor mai adăuga 7 octeți de completare după structură, astfel încât dimensiunea sa totală să devină multiplu de 8 (regula 3)!

```
sizeof(Student) = 49+7 = 56 de octeți
Randament = 44/56 = 0.79 = 79%
```

Soluție de îmbunătățire a randamentului: se declară câmpurile structurii în ordinea descrescătoare a dimensiunilor lor în octeți!

```
typedef struct
{
    double media;    //8 octeti
    int grupa;       //4 octeti
    char nume[31];   //31 de octeti
    char promovat;   //1 octet
}Student;            //total = 31+4+8+1 = 44 de octeti
```

`sizeof(Student) = 8+4+31+1 = 44 de octeți` **(nu mai sunt necesari octeți de completare între câmpuri – regula 2)**

44 NU este multiplu de `sizeof(media)`=8, deci se vor mai adăuga 4 octeți de completare după structură, astfel încât dimensiunea sa totală să devină multiplu de 8 (regula 3)!

```
sizeof(Student) = 44+4 = 48 de octeți
Randament = 44/48 = 0.92 = 92%
```

- **Inițializarea variabilelor de tip structură:**

```
typedef struct
{
    double media;
    int grupa;
    char nume[31];
    char promovat;
}Student;

int main()
{
    Student s = {9.50, 133, "Popescu Ion", 'D'};

    printf("Nume: %s\n", s.nume);
    printf("Grupa: %d\n", s.grupa);
    printf("Media: %.2f\n", s.media);
    printf("Promovat: %c\n", s.promovat);

    return 0;
}
```

Inițializările câmpurilor se realizează în ordine în care au fost declarate câmpurile structurii!!!

Pentru a inițializa doar anumite câmpuri se folosesc punctatorii:

```
Student s = {.nume = "Popescu Ion", .grupa = 133};
```

➢ Declararea structurilor complexe:

```
typedef struct
{
    char strada[101];
    int numar;
    char oras[51];
}Adresa;

typedef struct
{
    double media;
    int grupa;
    char nume[31];
    char promovat;
    Adresa adresa;
}Student;

typedef struct
{
    double salariu;
    char nume[31];
    Adresa adresa;
}Angajat;

.....................................................
Student s = {.nume = "Popescu Ion", .grupa = 133};

strcpy(s.adresa.oras, "Bucuresti");
```

sau

```
typedef struct
{
    double media;
    int grupa;
    char nume[31];
    char promovat;

    struct
    {
        char strada[101];
        int numar;
```

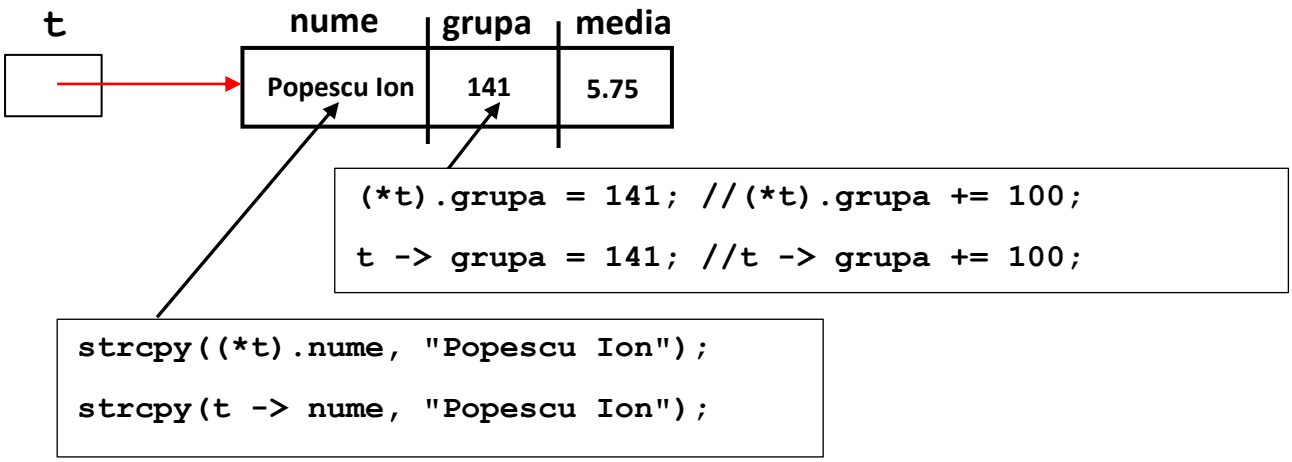
```
        char oras[51];
    }adresa;
}Student;

.....................................................
Student s = {.nume = "Popescu Ion", .grupa = 133};

strcpy(s.adresa.oras, "Bucuresti");
```

ALOCAREA DINAMICĂ A STRUCTURILOR

```
Student *t;
...........
t = (Student*)malloc(sizeof(Student));
...........
(*t).grupa = 141;
...........
free(t);
```



Observație: Alocarea dinamică a unei structuri NU implică și alocarea dinamică a câmpurilor sale!!!

```
typedef struct
{
    char *nume;
```

```
    int grupa;
    float media;
} Student;

Student *t;
...........
t = (Student*)malloc(sizeof(Student));
t->nume = (char*)malloc(21);
strcpy(t->nume, "Popescu Ion");
...........
free(t->nume);
free(t);
```

UNIUNI

Uniune = tip de date abstract (definit de către programator) cu ajutorul căruia se pot memora valori de tipuri diferite prin excludere reciprocă (la un moment dat, se poate utiliza un singur câmp dintre cele definite)

```
union Numar
{
    int x;
    double y;
    unsigned char z;
};
```

Observație: Dimensiunea unei uniuni este egală cu dimensiunea maximă a unui câmp al său.

Exemplu:

```c
#include<stdio.h>

typedef union
{
    int x;
    double y;
    unsigned char z;
}Numar;

int main()
{
    Numar t;

    t.z = 'A';
    printf("Caracterul: %c\n\n", t.z);

    t.x = 1234567890;
    printf("Numarul intreg: %d\n", t.x);
    printf("Caracterul: %c\n\n", t.z);

    t.y = 3.14;
    printf("Numarul real: %f\n", t.y);
    printf("Numarul intreg: %d\n", t.x);
    printf("Caracterul: %c\n", t.z);

    return 0;
}
```

```
C:\Users\Ours\Desktop\Test_C\bin\Debug\Test_C.exe
Caracterul: A

Numarul intreg: 1234567890
Caracterul: Ê

Numarul real: 3.140000
Numarul intreg: 1374389535
Caracterul: ▼

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

t	x					1234567890			
						01001001	10010110	00000010	11010010
	y								
	z								11010010 Ê

t	x					1374389535			
						01010001	11101011	10000101	00011111
	y	01000000	00001001	00011110	10111000	01010001	11101011	10000101	00011111
	z								00011111 ▼

[Double (IEEE754 Double precision 64-bit) (binaryconvert.com)]

Exemplu: Definiți o structură Automobil care să permită memorarea informațiilor despre un automobil.

Rezolvare:
Mai întâi, vom defini o enumerare Tip_automobil corespunzătoare tipului unui automobil (autoturism, autobuz sau autocamion):

```
typedef enum {Autoturism, Autobuz, Autocamion}
                                                Tip_automobil;
```

În afara unor caracteristici comune tuturor automobilelor (marcă, culoare, preț, capacitate cilindrică, anul fabricației etc.), mai există și unele caracteristici specifice doar anumitor tipuri de automobile, cum ar fi, de exemplu, capacitatea de transport. În cazul unui autoturism sau autobuz capacitatea de transport este indicată prin numărul maxim de pasageri, iar în cazul unui autocamion prin greutatea maximă pe care acesta o poate transporta. Pentru a evita păstrarea unor informații redundante (pentru un autoturism sau un autobuz este irelevantă greutatea maximă pe care o poate transporta, iar pentru un autocamion este irelevant numărul maxim de pasageri) și, implicit, pentru a minimiza spațiul de memorie utilizat, în cadrul structurii Automobil vom defini capacitatea sa folosind o uniune anonimă:

```
typedef struct
{
    Tip_automobil tip;
    //char tip[21];
    union
    {
        int nr_max_pasageri;
        float greutate_maxima;
    };
    float pret, capacitate_cilindrica;
    int an_fabricatie;
    char marca[20];
}Automobil;

.............

Automobil a;
a.tip = Autoturism;
```

ENUMERĂRI

Enumerare = tip de date abstract (definit de către programator) cu ajutorul căruia se pot defini seturi de constante întregi, implicit inițializate cu valori consecutive, începând cu valoarea 0.

```c
#include<stdio.h>

enum Zile{Luni = 1, Marti, Miercuri, Joi, Vineri = 100, Sambata, Duminica};

int main()
{
    enum Zile z;

    z = Luni;
    printf("z = %d\n", z);

    z = Marti;
    printf("z = %d\n", z);

    z = Vineri;
    printf("z = %d\n", z);

    z = Sambata;
    printf("z = %d\n", z);

    return 0;
}
```

```
C:\Users\Ours\Desktop\Test_C\bin\Debug\Test_C.exe
z = 1
z = 2
z = 100
z = 101

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

Observație: Valorile unei variabile de tip enumerare NU sunt restricționate la setul de valori precizat (nu se semnalează nicio eroare dacă variabila respectivă primește o valoare din afara setului de valori).

```c
#include<stdio.h>

typedef enum {Luni = 1, Marti, Miercuri, Joi, Vineri, Sambata, Duminica} Zile;

int main()
{
    Zile z;

    z = Luni + 100;
    printf("z = %d\n", z);

    return 0;
}
```

```
C:\Users\Ours\Desktop\Test_C\bin\Debug\Test_C.exe
z = 101

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

Exemplu:

```
typedef enum {PORNIT=1, IN_ASTEPTARE=107, IN_MISCARE, OPRIT=0}
                                                        Stare_robot;

int main()
{

    printf("Dimensiune Stare_robot: %d\n", sizeof(Stare_robot));

    Stare_robot sr = 101;
    printf("Starea robotului: ");

    switch(sr)
    {
        case PORNIT:
            printf("Pornit\n");
            break;

        case OPRIT:
            printf("Oprit\n");
            break;

        case IN_MISCARE:
            printf("In miscare\n");
            break;

        case IN_ASTEPTARE:
            printf("In asteptare\n");
            break;

        default:
            printf("Stare eronata!");
    }

    return 0;
}
```