# Structuri, câmpuri de biți

# Structuri

- Declarare și inițializare
- Declarare tipuri
- Accesul la membrii structurilor
- Pointeri la structuri
- Tablouri de structuri
- Atribuiri
- Funcții

# Ce este o structură?

- O structură este un tip de date care conține o listă de variabile (posibil de tipuri diferite) care sunt identificate printr-un singur nume și o locație de memorie.
- Variabilele conținute în structură se numesc câmpuri ale structurii
- o structură poate conține
  - câmpuri de același tip sau de tipuri diferite
  - câmpuri de tipuri simple sau compuse (structuri care conțin la rândul lor alte structuri)

# Declarare structură

struct numeStructura {tip_camp1 nume_camp1;
tip_camp2 nume_camp2;….};

structura trebuie definită utilizând cuvântul cheie struct a.i. compilatorul sa o poata folosi

# Exemplu de structură

- Punct2D – contine 2 câmpuri, x și y ce reprezintă coordonatele pe cele 2 dimensiuni

```
struct Punct2D{
        int x,y;
};
```

# typedef

- typedef ne permite să definim tipuri noi de date atât simple cât și compuse
- ex:
  - typedef unsigned char byte; //definește un tip byte care va avea aceeași
    //definiție cu unsigned char

  putem apoi declara variabile utilizând noul tip

  byte b1, b2; // declară 2 variabile de tip byte
- sintaxa
  - typedef definitie_tip nume_nou_tip;

# utilizare typedef in contextual definirii structurilor

**definitie structura fara typedef**

```
struct Punct2D{
        int x,y;
};
```

**definire structura cu typedef**

```
typedef struct {
        int x, y;
        }TPunct2D;
```

# Declarare variabila de tip structura

**definitie structura fara typedef**

struct nume_structura nume_variabila;

**definire structura cu typedef**

numetip nume_variabila;

```
struct Punct2D p;
```

```
TPunct2D p1;
```

# Initializari structuri

- modificam un pic structura pentru a putea sa ne jucam cu mai multe tipuri de campuri

```
typedef struct {
        int x, y;
        char nume[10];
        }TPunct2D;
```

....

```
TPunct2D p1={5,6,"abc"};
TPunct2D p2={.y=4,.nume="xyz",.x=6};
TPunct2D p3={5};
```

# initializari structuri

```
printf("p1 are coordonatele (%d,%d,%s)\n",p1.x,p1.y,p1.nume);
printf("p2 are coordonatele (%d,%d,%s)\n",p2.x,p2.y,p2.nume);
printf("p3 are coordonatele (%d,%d,%s)\n",p3.x,p3.y,p3.nume);
```

```
p1 are coordonatele (3,9,abc)
p2 are coordonatele (6,4,xyz)
p3 are coordonatele (5,0,)
```

```
TPunct2D p1={5,6,"abc"};
TPunct2D p2={.y=4,.nume="xyz",.x=6};
TPunct2D p3={5};
```

- p1={5,6,”abc”}; //sunt initializate campurile in ordine
- p2={.y=4, .nume=“abc”, .x=6}; //sunt initializate campurile explicit
- p3={5}; //campurile care nu sunt mentionate sunt initializate cu valorile default (0, \0
- TPunct2D p4; //p4 va avea toate campurile initializate in mod aleator

# Dimensiunea unei structuri

```
typedef struct {int x, y; char c; float f;} TTest1;
typedef struct {int x, y; float f; char c;} TTest2;

int main(void)
{
        TTest1 t1;
        TTest2 t2;
        printf("adrese campuri t1: %p %p %p %p\n", &t1.x,&t1.y,&t1.c,&t1.f);
        printf("adrese campuri t2: %p %p %p %p\n", &t2.x,&t2.y,&t2.c,&t2.f);
        printf("dim t1=%d\ndim t2=%d\n",sizeof(t1),sizeof(t2));
        return 0;
}
```

```
mint@mint ~/programare/c7 $ ./a.out
adrese campuri t1: 0xbf93a880 0xbf93a884 0xbf93a888 0xbf93a88c
adrese campuri t2: 0xbf93a890 0xbf93a894 0xbf93a89c 0xbf93a898
dim t1=16
dim t2=16
```

# adresare campuri

- daca avem o variabila de tip structura utilizam operatorul “.” pentru a accesa campurile
- utilizare: numeVariabila.numeCamp
- daca avem un pointer catre o variabila de tip structura utilizam operatorul “->” pentru a accesa campurile
- TPunct2D *ps;
- ps=&p1;
- ps->x=5;//atribuie campului x valoarea 5

# Exemplu adresare folosind “->”

```
TPunct2D p1={5,6,"abc"};
TPunct2D p2={.y=4,.nume="xyz",.x=6};
TPunct2D p3={5};
TPunct2D *ps;
```

```
ps=&p1;
printf("ps are coordonatele (%d,%d,%s)\n",ps->x,ps->y,ps->nume);
```

```
p1 are coordonatele (3,9,abc)
p2 are coordonatele (6,4,xyz)
p3 are coordonatele (5,0,)
ps are coordonatele (3,9,abc)
```

# Pointeri la structuri

- respecta aceleasi reguli ca si ceilalti pointeri☺
- trebuie sa-i initializam si sa avem grija sa facem conversii explicite cand este cazul
- pointeri la structuri vs. structuri care contin pointeri
  - cand avem un pointer la o structura folosim “->” pentru a accesa campurile
  - cand avem o structura care are un camp de tip pointer nu ne afecteaza tipul de operator folosit

# Functii ce prelucreaza structuri

- cand o structura este trimisa parametru unei functii se face o copie a zonei de memorie respective
- modificarile efectuate asupra structurii in functie nu vor afecta si structura originala

# Exemplu

```
void f(TPunct2D p)
{
        printf("adresa lui p: %p, p.x=%d, p.y=%d, p.nume=%s\n",
                &p,p.x,p.y,p.nume);
        strcpy(p.nume, "zzz");
}
```

in main

```
f(p1);
printf("p1 are adresa: %p si numele %s\n",&p1, p1.nume);
```

```
adresa lui p: 0xbfd1da70, p.x=3, p.y=9, p.nume=abc
p1 are adresa: 0xbfd1daa0 si numele abc
```

# Vectori de structuri

- Vezi program/tabla

# Structuri ce contin structuri

- Vezi program/tabla

# Sortarea unui vector de structuri dupa mai multe criterii

- Vezi program/tabla

# Campuri de biti

- structuri in care putem declara campurile ca avand un numar fix de biti (mai mic sau egal cu tipul de baza)
- putem gestiona separat bitii dintr-o zona de memorie
- ex:

```
typedef struct myCar{
unsigned int cc: 13;
unsigned int vmax: 9;
unsigned int anFab: 8; //ultimele 2 cifre
unsigned int vanduta: 1; //da/nu
}
```

# Exemplu campuri de biti

```c
#include<stdio.h>
typedef struct {
 unsigned short red:4;
 unsigned short green:4;
 unsigned short blue: 4;
 unsigned short other: 4;
} TColor;

int main()
{
        TColor c;
        c.red=21;
        c.green=5;
        printf("c=%d, c.red=%d, c.green=%d, c.blue=%d, c.other=%d\n",
        c,c.red,c.green, c.blue, c.other);
        return 0;
}
```

# Exemplu campuri de biti

```
p3.c: In function ‘main’:
p3.c:12:2: warning: large integer implicitly truncated to unsigned type [-Wover
flow]
  c.red=21;
  ^
```

```
mint@mint ~/programare/c7 $ ./a.out
c=46933, c.red=5, c.green=5, c.blue=7, c.other=11
```

# Union

- Tip de date special
- Similar cu struct
- Diferență: permite salvarea mai multe date la aceeași adresă de memorie
- Util: când folosiți doar unul dintre ele, dar diferă de la caz la caz

```
union Data {
        int i;
        float f;
        char str[20];
};
```

- O variabilă de tipul Data poate reține un int, un float sau un șir de caractere

# Union

- Se alocă memorie cât pentru câmpul de dimensiune maximă
- Accesul la câmpuri se face la fel ca la struct
- Vezi exemplu