Programare II
Limbajul C/C++

CURS 5

Curs anterior

- Pointeri constanți
  - Tablouri
- Șiruri de caractere
- Alocarea dinamică a memoriei
- Pointeri la funcții
- Bibliografie
  - Kernighan B. and D. Ritchie - The C Programming Language - capitolul 5

Curs curent

- Structuri
- Uniuni
- Bit-vector
- Operatori pe biți
- Bibliografie
  - Kernighan B. and D. Ritchie - The C Programming Language - capitolul 6

Structuri. Uniuni. Bit-vector

- Cum structurăm date care au sens împreună?

- Putem defini noi structuri de date?
  - struct

  - union

Structuri

- Gruparea mai multor variabile care au sens împreună

- Exemplu
  - Informațiile despre o dată calendaristică

```
struct Data{
  int zi;
  int luna;
  int an;
};
```

  - Informațiile despre o persoană

```
struct Persoana{
  struct Data dataNastere;
  char * nume;
  struct Persoana **rude;
};
```

Structuri

- Sintaxă
  - `struct [nume] { declarații de variabile} [lista de variabile];`
- Definire

```
struct coordonate { int latitudine, longitudine;};
```

- Declarare variabile

```
struct coordonate timisoara;
```

- Inițializare

```
struct coordonate timisoara = {45, 21};
```

Structuri. Accesarea membrilor

```
struct coordonate { int latitudine, longitudine;};
```

- Variabilă de tipul structurii

```
struct coordonate timisoara;
timisoara.latitudine = 45;
timisoara.longitudine = 23;
```

- Pointer de tipul structurii

```
struct coordinate *bucuresti;
bucuresti = (struct coordonate*)malloc(sizeof(struct coordonate));
bucuresti->latitudine = 45;
(*bucuresti).longitudine = 25;
```

typedef

- Mecanism prin care se pot da noi nume unor tipuri de date
  - Tipurile noi sunt aliasuri ale tipurilor de date referite
  - Sporesc claritatea programului
  - Sintaxa:

    typedef tip_existent tip_nou;
- Exemple

```
typedef unsigned char uint8_t;
typedef char int8_t;

typedef unsigned int uint32_t;
typedef int int32_t;
```

typedef

- Exemple

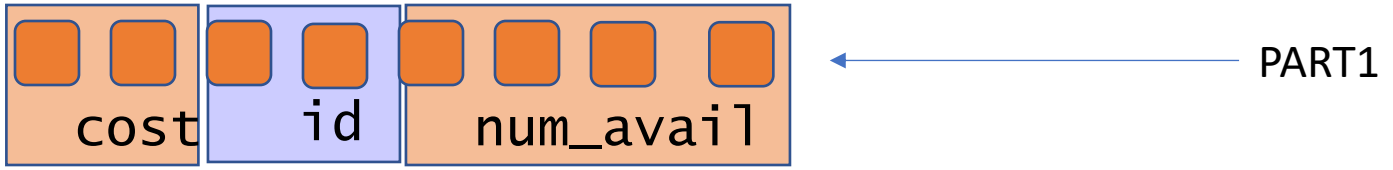
```
struct Persoana{
  Data dataNastere;
  char * nume;
  struct Persoana **rude;
};

typedef struct Persoana Persoana;

Persoana popescu, *ionescu;
```

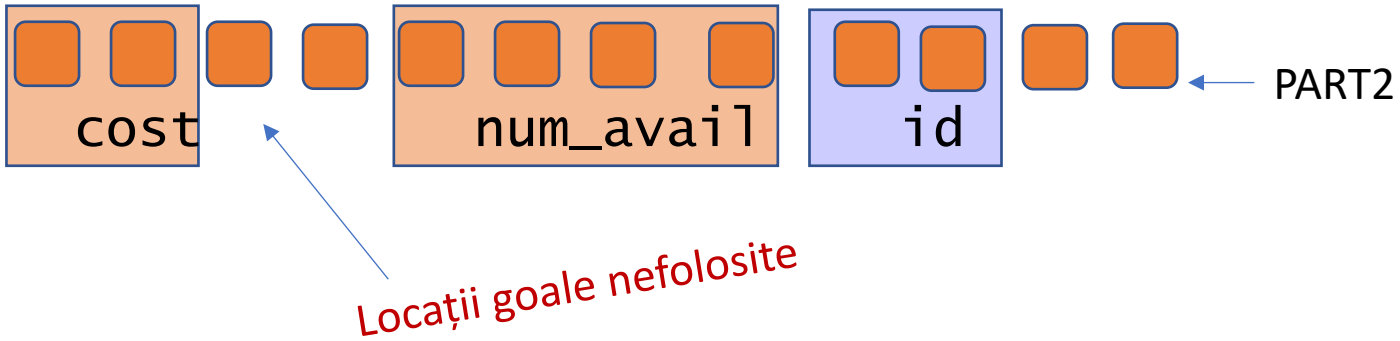
```
typedef struct {
  Data dataNastere;
  char * nume;
  struct Persoana **rude;
} Persoana;
```

```
typedef struct Persoana{
  Data dataNastere;
  char * nume;
  struct Persoana **rude;
} Persoana;
```

Spațiu de memorie ocupat

- Dimensiunea în memorie este egala cu cel puțin suma tipurilor câmpurilor
- Alinierea
  - Aliniere 4 bytes / 8 bytes in functie de processor (32 bits / 64 bits)
  - Se face automat de catre compilator pentru a optmiza numarul de cicluri procesor necesare accesarii datelor
  - Configurabil prin folosirea extensiilor compilatorului: __attribute__((aligned(Nr_bytes))), __declspec((aligned(Nr_bytes)), __attribute__((packed))






```
#define N 2

struct COST {
   char currency_type[N];
};

struct PART1 {
   struct COST cost;
   char id[N];
   int num_avail; ;
};

struct PART2 {
   struct COST cost;
   int num_avail;
   char id[N];
};
```

Bit-field

- Dacă spațiul structurii trebuie gestionat cu grijă
  - Numărul după „:” reprezintă lungimea în biți
  - Variabile ar trebui să fie declarate `unsigned int`

- Exemplu

```
struct CHAR { unsigned ch: 7;
              unsigned font: 6;
              unsigned size: 19; };
```

Structuri

Definiți o structură de date care să conțină numele unui student, anul de studiu. Definiți funcții care permit citirea unei structuri, afișarea unui tablou de structuri si modificarea conținutului structurii.

Uniuni

- Asemănătoare cu structurile, insa:
  - Doar un membru este activ la un moment dat
  - Spațiul de memorie ocupat este dat câmpul care ocupa spațiul de memorie maxim dintre câmpurile structurii de tip union
- Programul trebuie sa gestioneze corect valoarea stocata in uniune pentru a o recupera
- Exemplu

Uniuni

```
typedef struct {
    enum { Int, Float, Char} type;
    union {
        int iValue;
        float fValue;
        char cValue;
    } value;
} VARIABLE;
```

```
void printValue(VARIABLE var)
{
    switch (var.type)
    {
        case Int:
            printf("var=%d\n", var.value.iValue);
            break;
        case Float:
            printf("var=%f\n", var.value.fValue);
            break;
        case Char:
            printf("var=%c\n", var.value.cValue);
            break;
        default:
            printf("var not defined\n");
    }
}
```

Operatori pe biți

Operator	Descriere	Tip operator	Exemple

&	AND	Binar	255 & 2 == 2; // (1111 1111) & (0000 0010)
			2 & 4 == 0; // (0000 0010) & (0000 0100)
			4 & 4 == 4;
			17 & 1 == 1; // X & 1 reprezinta restul impartirii lui X la 2

|	OR	Binar	2 | 4 == 6; // (0000 0010) | (0000 0100)
			3 | 4 == 7; // (0000 0011) & (0000 0100)
			4 | 4 == 4;

^	XOR (sau exclusiv)	Binar	4 ^ 4 == 0;
			2 ^ 4 == 6 ; // (0000 0010) ^ (0000 0100)
			6 ^ 2 == 4; // (0000 0110) ^ (0000 0010)

~	NOT (inversare biți)	Unar	unsigned char A = 0, B = 128;
			~A == 255; // (0000 0000) => (1111 1111)
			~B == 127; // (1000 0000) => (0111 1111)

Operatori pe biți

Operator	Descriere	Tip operator	Exemple

<<	Deplaseaza la stanga pe biți
	Biții noi adaugați se pun 0
		Binar	N << 1; // inmultire cu 2
			1 << N; // 2 ^ N

>>	Deplaseaza la dreapta pe biți Biții mai puțin semnificativi se pierd	Binar	N >> 1; // impartire intreaga la 2

- Exemplu

char A = 1;

char B = 2;

int C = (A << 8) | B;

char AA = C >> 8; // AA = A

char BB = C & 0xFF; // BB = B

Despre ce vom discuta?

- Programare orientată obiect. Abstractizarea datelor
- Diferențe / noutăți C – C++
- Clase. Obiecte
  - Specificatori de acces
  - Constructori
  - Destructori
- Bibliografie
    - Helbert Schildt- The Complete References C++- capitolele 11, 12, 13

ÎNTREBĂRI