I2010 - Langage C

Chap. 9:
Les types utilisateurs

Définition de type

- Convention: un identificateur de type commence toujours par une majuscule
- Les types utilisateur sont définis après les directives du préprocesseur, généralement dans les fichiers d'entête

Structures

- **type composé** permettant de regrouper des données – nommées ***champs*** – pouvant être de types différents dans une même entité

Définition de structure

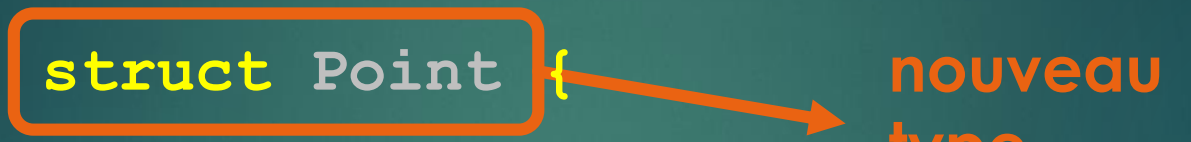
- définition de type structure:

```
struct Point {
    int abscisse;
    int ordonnee;
};
```

Définition de structure

- définition de type structure:

```
struct Point {
    int abscisse;
    int ordonnee;
};
```



Définition de structure

- définition de type structure:

```
struct Point {
    int abscisse;
    int ordonnee;
};
```

Déclaration de variable

- avec ou sans initialisation

```
struct Point p1;

struct Point p2 = {0,0};

struct Point * ptr = &p2;
```

Accès aux champs

- nom de la variable et nom du champs introduit par « . » ou « **->** »

Niveau de priorité	Opérateur	description	Associativité
17	[]	indice de tableau	gauche
	(...)	appel de fonction	
	.	sélection de membre	
	->	sélection de membre par déréférencement	
16	++	post-incrémentation	gauche
	--	post-décrémentation	

```
p1.abscisse = 5;
(*ptr).ordonnee = 8;
ptr->ordonnee = 8;
```

Opérations sur les structures


```
ptr = &p1      // adresse d'une structure
int* p = &p1.abscisse  // adresse d'un champ
p2 = p1        // affectation d'une structure
void fct (struct Point p) // paramètre
struct Point fct (...)  // valeur de retour
```

Structure récursive

- C doit connaître la taille d'un type à la compilation !

```
struct Noeud {
    int valeur;
    struct Noeud suivant;
};
```

Structure récursive

- C doit connaître la taille d'un type à la compilation !

```
struct Noeud {
    int valeur;
    struct Noeud suivant;
};
```

erreur:
récursivité interdite !

Structure récursive

- C doit connaître la taille d'un type à la compilation !

  → utilisation d'un pointeur

```
struct Noeud {
    int valeur;
    struct Noeud * suivant;
};
```

correct:
taille connue

Énumération

- type construit à partir d'un **ensemble de valeurs** spécifiées dans la définition du type

Définition d'énumération

- définition de type énuméré:

```
enum Couleur { ROUGE, VERT, BLEU };
```

Définition d'énumération

- définition de type énuméré:

```
enum Couleur { ROUGE, VERT, BLEU };
```

nouveau type énuméré

Définition d'énumération

- définition de type énuméré:

```
enum Couleur { ROUGE, VERT, BLEU };
```

ensemble des valeurs

Définition d'énumération

- définition de type énuméré:

```
enum Couleur { ROUGE, VERT, BLEU };
```

- identificateur des valeurs énumérées en majuscules (= constantes)
- énumération = type entier !

```
ROUGE = 0
JAUNE = 1
VERT  = 2
...
```

Définition d'énumération

- définition de type énuméré:

```
enum Couleur { ROUGE, VERT, BLEU };
```

- identificateur des valeurs énumérées en majuscules (= constantes)
- énumération = type entier !

  ⇒ usage fréquent dans des *switch* (branchement multiple en fonction d'une valeur entière)

Utilisation

- déclaration de variable

```
enum Couleur maCouleur, maFavorite;
```

- affectation

```
maCouleur = ROUGE;
```

- opérations

```
maCouleur++;  // reçoit la couleur JAUNE
```

typedef

- définition d'un synonyme d'un type existant

```
typedef unsigned int size_t;
```

typedef

- définition d'un synonyme d'un type existant

typedef

- définition d'un synonyme d'un type existant

```
typedef unsigned int size_t;
```

typedef

- définition d'un synonyme d'un type existant

```
typedef unsigned int size_t;
```

nouvel identificateur de type

typedef

- définition d'un synonyme d'un type existant

```
typedef unsigned int size_t;
```

- souvent utilisé pour renommer les types complexes t.q. structures et énumérations

```
typedef struct Point {
   int abscisse;
   int ordonnee;
} Point;
typedef enum Couleur Couleur;
```

```
Point p;
Couleur c;
```

typedef

- définition classique d'un type booléen (non existant en ANSI C) :

```
typedef enum {FALSE, TRUE} Boolean;
...
Boolean valid = TRUE;
```