enum,
union et typedef
en langage C

1. Les énumérations

a. *Définition*

L'énumération est un type de données défini par l'utilisateur. Elle est notamment utilisée pour attribuer des noms (et donc favoriser la maintenance du code) à des constantes entières.

Syntaxe :

```
enum <def_type> {const1, const2, …};
```

Par défaut, `const1` prend la valeur 0 et ainsi de suite mais il est possible d'attribuer individuellement des valeurs à chaque constante.

Exemple :

```
enum Feu_Tricolore {VERT, ORANGE, ROUGE};
enum booleen {VRAI=1, FAUX=-1};

enum Feu_Tricolore F;
```

2. Les unions

b. *Définition*

Il est parfois nécessaire de manipuler des variables auxquelles on désire affecter des valeurs de type différents. Ceci peut se réaliser en langage C grâce au mécanisme des ***unions***.
Une définition d'union a la même syntaxe qu'une définition de structure, le mot-clé **struct** étant remplacé par le mot-clé **union.**

Syntaxe :

```
union <def_type> {
  <members>
} union_name;
```

Exemple :

```
union nombre {
    int i;
    float f;
} ;
```

La différence fondamentale avec les structures est la suivante.
Tous les champs d'une **structure** peuvent posséder en même temps une valeur. Dans une variable de type **union**, à un instant donné, seul un champ peut prendre une valeur.
Ainsi, dans l'exemple précédent si on déclare :

```
union nombre union_name;
```

union_name pourra prendre soit une valeur entière, soit une valeur réelle mais pas les deux en même temps !

c. *Accès aux champs*

L'accès se fait par l'opérateur "." (qui sert aussi à accéder aux champs des structures).
Dans l'exemple précédent, on pourra écrire :

```
n.i = 10;    ou encore n.f = 10.1;
```

Syntaxe :

```
union_name.membre OU union_pointer->membre;
```

d. *Utilisation des unions*

Lorsqu'on manipule des unions, le programmeur n'a aucun moyen de connaître à un moment donne quel est le champ de l'union qui possède une valeur !
Pour être utilisable, l'union doit donc être associée à une variable dont le but sera d'indiquer le champ de l'union valide.
En règle générale, on utilise une structure.

Exemple :

```
#define ENTIER 0
#define REEL   1
Page 2

struct mathematique {
  int type;
  union {
    int i;
    float f;
  } u;
};

struct mathematique A;
A.type= ENTIER;
A.u.i = 10;
```

3. Les types de données

Il existe un moyen en langage C de donner un nom à un nouveau type avec le mot clé **typedef.**

Syntaxe :

```
typedef <def_type> nouveau_type;
```

Exemple :

```
typedef struct {
  char nom[30] ;
  char prenom[30) ;
} personne, *p_personne ;
```

Ici, on déclare deux nouveaux types : `personne` (la structure) et `p_personne` (pointeur sur la structure personne).
Ces types sont ensuite utilisables dans des déclarations de variables comme types de base.

Exemple :

```
personne P, *ptr_p ; /* P est de type personne et ptr_p, pointeur vers personne */
```