INFORMATIQUE 2

I. STRUCTURES
ENUMERATIONS
TYPES

Eva ANSERMIN & Romuald GRIGNON

TECH

Les différents types en programmation

- Nous avons vu jusqu'à maintenant les types "de base" (int, float, char, double).

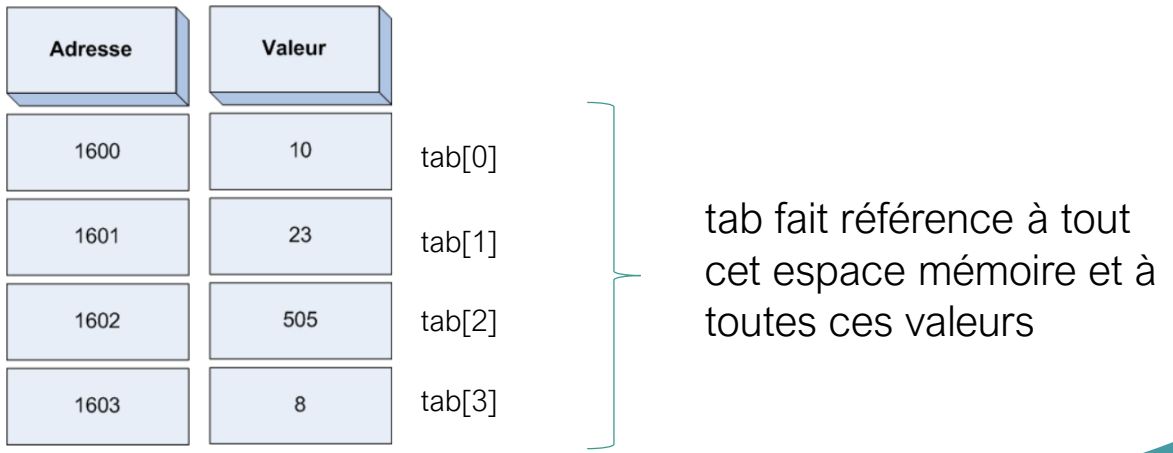
- Les différents types sont répartis en deux categories :
  - Les types **primitifs** : la variable contient directement une valeur (c'est le cas de int, float etc…)
  - Les **types références**: la variable fait référence à une ou plusieurs valeurs stockées en mémoire.

Les différents types en programmation

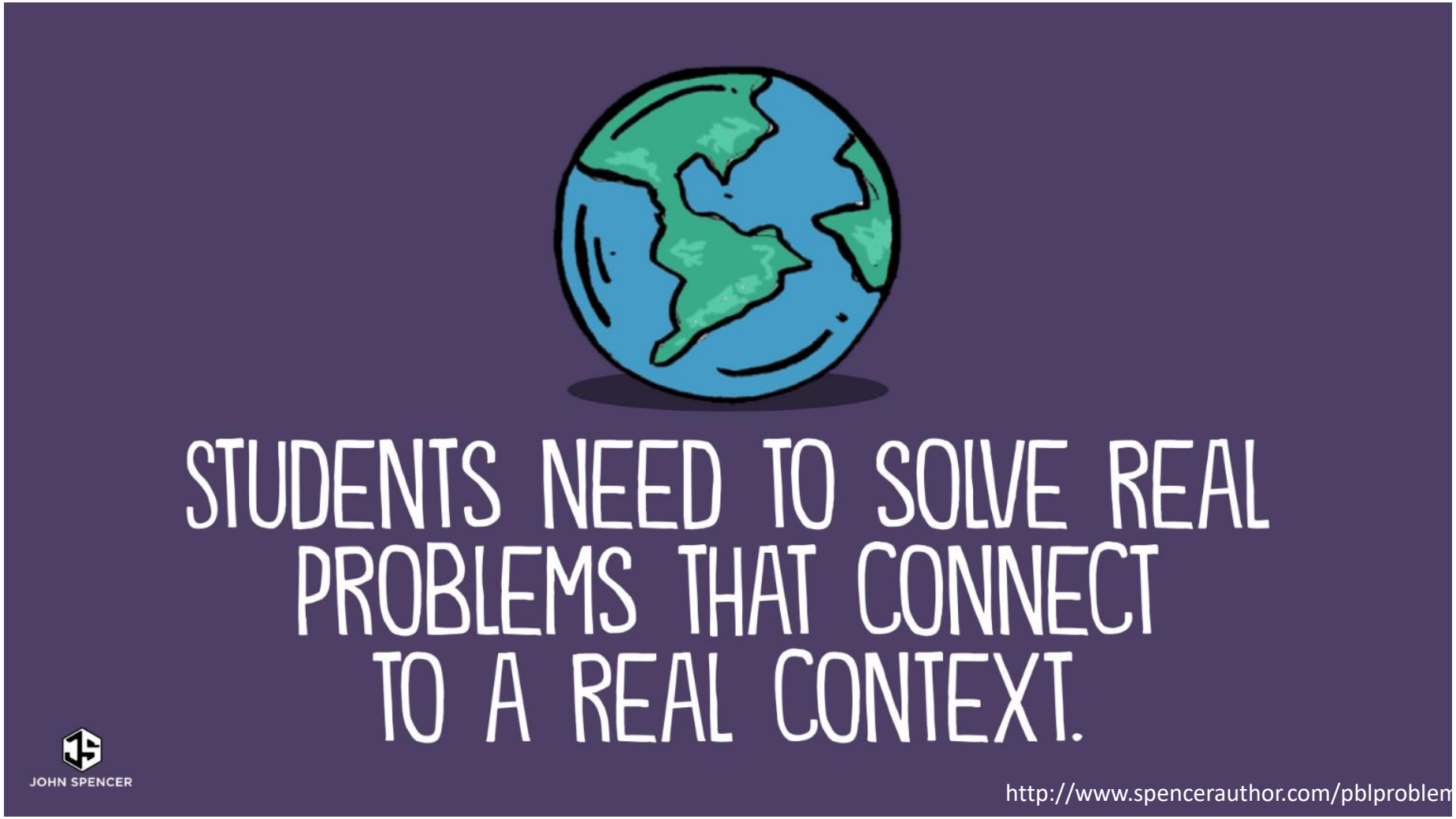
- Nous avons vu jusqu'à maintenant les types "de base" (int, float, char, double).

- Les différents types sont répartis en deux categories :
  - Les types **primitifs** : la variable contient directement une valeur (c'est le cas de int, float etc…)
  - Les **types références**: la variable fait référence à une ou plusieurs valeurs stockées en mémoire.
    - Ex : les tableaux !

tab ← {10, 23, 505 ,8}



Le monde n’est pas fait d’entiers et de réels….



Le monde n’est pas fait d’entiers et de réels….

- En géométrie, nous manipulons des **coordonnées**, des **rectangles**, des **droites** ... ;
- Un magasin gère des **produits**, des **clients**, des **factures**, des **fournisseurs** ;
- Dans une médiathèque, il y a des **documents**, des **usagers**, qui **louent**, **réservent** ces documents ...
- Exemple : On souhaite faire un programme qui va gérer des étudiants. Chaque étudiant possède :
  - un nom
  - un prénom
  - un numéro d’étudiant
  - un groupe
  - une ou plusieurs notes…

Exemple : gestion des étudiants

```
Programme gestion_etudiants
      VARIABLE
      nom_etudiant1, nom_etudiant2 : chaîne de caractères
      pnom_etudiant1, pnom_etudiant2 : chaîne de caractères
      num_etudiant1, num_etudiant2 : entier
      gp_etudiants1, gp_etudiant2 : entier
      note_etudiant1, note_etudiant2 : réel

      DEBUT
      …
```

Une première définition des structure

Un **type structuré** représente une donnée **complexe**, composée de plusieurs **valeurs** pouvant être de types **différents** :

- Un nombre *C*omplexe possède une partie réelle et une partie imaginaire ;
- Une *Coordonnée* est composée de coordonnées réelles (x, y) dans un espace de dimension 2, de coordonnées réelles (x, y, z) pour une *Coordonnée* en 3D, etc. ;
- Une *Date* est définie par un jour, un mois, une année, une heure, des minutes, secondes, millisecondes ;
- Une *Adresse* est définie par un numéro et un nom de rue, un code postal et une ville, etc.

Une définition étendue des structures

- Un type structuré définit un **type de donnée** (au même titre qu'entier, chaîne ou réel) ;
- À la différence des variables de types *primitifs*, une variable d'un tel type structuré contient **plusieurs valeurs** stockées dans des **champs** (ou **membres** ou **attributs**) ;
- Les variables de type structuré sont appelées parfois **enregistrements**.
- C'est le programmeur qui definit la structure, lui donne un nom et définit ses champs.

Exemple simple

- Une *Coordonnée* dans un espace 2D est définie par son abscisse et son ordonnée.
- On va construire une structure appellee *Coordonnée*, qui sera definit par deux champs: l'abscisse et l'ordonnée.

```
STRUCTURE Coordonnée
    abscisse : réel
    ordonnée : réel
FIN STRUCTURE
```

Exemple simple

- *Coordonnée* est maintenant un nouveau type!

```
STRUCTURE Coordonnée // définition de la structure
      abscisse : réel
      ordonnée : réel
FIN STRUCTURE

VARIABLE // déclaration des variables
      a: entier
      x : Coordonnée // x et y sont des ‘Coordonnée’
      y : Coordonnée
```

Exemple simple

- *Coordonnée* est maintenant un nouveau type!

```
STRUCTURE Coordonnée // définition de la structure
      abscisse : réel
      ordonnée : réel
FIN STRUCTURE

VARIABLE // déclaration des variables
      a: entier
      x : Coordonnée // x et y sont des 'Coordonnée'
      y : Coordonnée
```

- Les *Coordonnées* sont des variables possédant deux attributs de types réels: abscisse et ordonnée.

Exemple simple

```
STRUCTURE Coordonnée // définition de la structure
      abscisse : réel
      ordonnée : réel
FIN STRUCTURE

VARIABLE // déclaration des variables
      a: entier
      x : Coordonnée // x et y sont des 'Coordonnée'
      y : Coordonnée
DEBUT
      x.abscisse <- 3 // acceder au champs
      x.ordonnée <- 5,5
      y.abscisse <- x.abscisse +2
      …
```

Syntaxe

- Définition du type structuré

```
STRUCTURE nomType
        nomChamp1 : TYPE1,
        nomChamp2 : TYPE2,
                ....,
        nomChampn : TYPEn
FIN STRUCTURE
```

- Déclaration de variable

```
elt1, elt2 : nomType
```

- Manipulation :

Accéder au champ $nomChamp_i$ de la variable elt1 se fait selon : `elt1.nomChamp`$_i$

Manipulation des structure

- Chaque champ d'une occurrence de la structure peut être considérée comme une variable indépendante.

- Les opérations se font champ par champ ! (comme les cases pour les tableaux)

  - Exemple : x, y et z sont des *Coordonnée*

~~z ← x+y~~

z. abscisse ← x. abscisse + y. abscisse
z.ordonnee ← x.ordonnee + y.ordonnee

Manipulation des structure

- Chaque champ d'une occurrence de la structure peut être considérée comme une variable indépendante.

- Les opérations se font champ par champ ! (comme les cases pour les tableaux)

- On a par contre le droit d'écrire :

$$z \leftarrow x$$

Les valeurs des champs de z seront égales à ceux de x

Constructeur

- Constructeur de variable de type structuré (permet de valider les préconditions sur le type et de retourner la variable ou bien de terminer le programme)

```
FONCTION creerType(val1 : TYPE1, val2 : TYPE2, ...) : nomType
VAR res : nomType
DÉBUT
    SI (condition(val1, val2, ...)) ALORS
      écrire ("Un message d'erreur explicite")
      erreur() // met fin au programme en erreur
    FIN SI
    res.nomChamp1 <- val1
      res.nomChamp2 <- val2
            ....
      retourner res
FIN
```

Exemple simple

```
STRUCTURE Coordonnée
        abscisse : réel
        ordonnée : réel
FIN STRUCTURE


FONCTION creerCoordonnée(x, y: reel) :
Coordonnée
VAR p : Coordonnée
DÉBUT
      p.abscisse <- x
      p.ordonnee<- y
      RETOURNER p
FIN
```

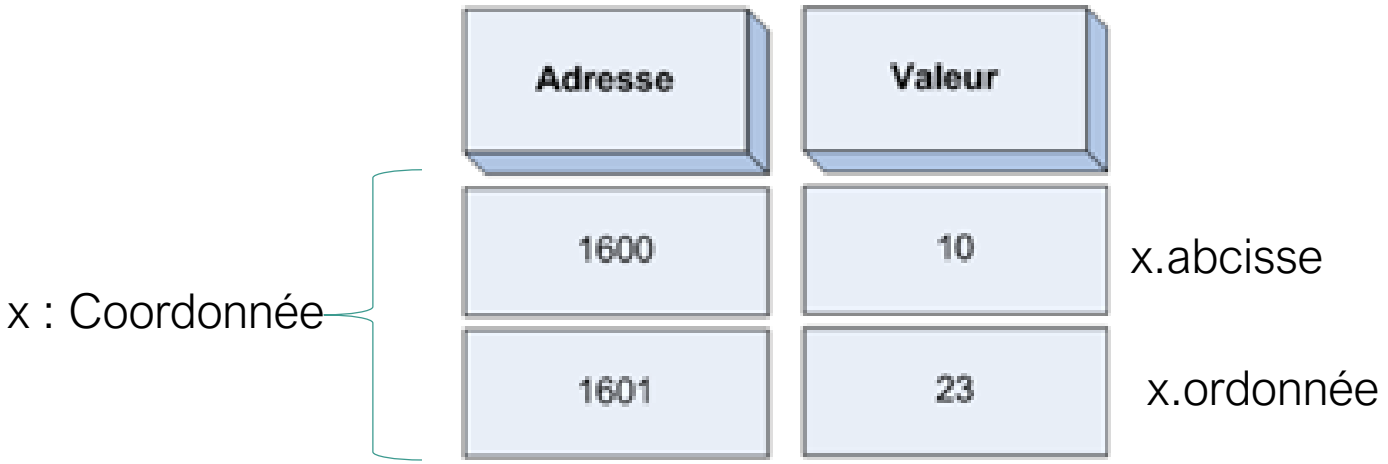
```
VARIABLE
        x : Coordonnée
        a : réel
        b : réel

DEBUT
        ECRIRE(“donner les
coordonnées du Coordonnée”)
        LIRE(a)
        LIRE(b)
        x<- creerCoordonnée(a,b)
FIN
```

Remarques

- **Le nom des structures commence toujours par une majuscule.**

- Il ne faut pas confondre la structure (le type créé) et les variables de cette structure.

- Les différents champs d'une variable de type structure sont à la suite dans la mémoire (comme les cases d'un tableau).



Un exemple plus compliqué…

- On va écrire la structure Etudiant.
- Chaque étudiant a :

```
STRUCTURE Etudiant




FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.
- Chaque étudiant a : un nom

```
STRUCTURE Etudiant
        nom : chaîne de caractères



FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.
- Chaque étudiant a : un nom, un prénom

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères



FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.

- Chaque étudiant a : un nom, un prénom, un numéro etudiant

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier


FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier

FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, des notes

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
        note1 : reel

FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.
- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, N notes

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
        note1 : reel
        note2: reel
        …
        noteN : reel
FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, N notes

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
        note[N] : tableau de rééls

FIN STRUCTURE
```

Un exemple plus compliqué…

- On va écrire la structure Etudiant.

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, N notes, une adresse

```
STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réél
        adresse : ???
FIN STRUCTURE
```

Un exemple plus compliqué…

- Une adresse est composée de plusieurs informations : le nom de la rue, le numéro de la maison, la ville, le code postal…

- Une structure semble donc appropriée pour représenter une adresse

```
STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE
```

Un exemple plus compliqué…

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, N notes, une adresse

```
STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE


STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réels
        add : Adresse
FIN STRUCTURE
```

Un exemple plus compliqué…

- Chaque étudiant a : un nom, un prénom, un numéro etudiant, un groupe, N notes, une adresse

```
STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE

STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réels
        add : Adresse
FIN STRUCTURE
```

Pour indiquer le code postal d'un Etudiant:

```
VARIABLE
        etu1 : Etudiant
DEBUT
        etu1.add.code <- 95000
```

Se lit : " le code de l'adresse de l'étudiant etu1"

Un exemple plus compliqué…

- Et pour gérer la promotion de 330 élèves?

```
STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE


STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réels
        add : Adresse
FIN STRUCTURE
```

Un exemple plus compliqué…

- Et pour gérer la promotion de 330 élèves?

```
Constante NB_ETU <- 330

STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE

STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réels
        add : Adresse
FIN STRUCTURE
```

```
VARIABLE
        prom[NB_ETU] : tableau d'Etudiant
DEBUT
```

Un exemple plus compliqué…

- Et pour gérer la promotion de 330 élèves?

```
Constante NB_ETU <- 330

STRUCTURE Adresse
        num : entier
        rue : chaîne de catactères
        ville : chaîne de caractères
        code : entier
FIN STRUCTURE

STRUCTURE Etudiant
        nom : chaîne de caractères
        prenom : chaîne de catactères
        num : entier
        groupe : entier
         note[N] : tableau de réels
        add : Adresse
FIN STRUCTURE
```

```
VARIABLE
        prom[NB_ETU] : tableau d'Etudiant
DEBUT
        prom[0].nom<-"Dupont"
        prom[5].groupe <- 2
        prom[1].add.num <- 56
        prom[1].num<- 225784
        …
```

Les structure en C : syntaxe

- Déclaration d'une **structure**

```
struct NomStructure {
    type champ1 ;
    type champ2 ;
    …
};
```

- Utilisation d'une **structure**

```
struct NomStructure nomVariable;
```

- Manipulation :

```
nomVariable.champ
```

Les structure en C : syntaxe

- Exemple

```
struct Coordonnée{
      float abscisse;
      float ordonnee;
};
int main(){
      struct Coordonnée c1;
      scanf("%f",&c1.abcisse);
      scanf("%f",&c1.ordonnee);
      printf( "Les coordonnées de c1 sont (%f, %f) \n",
               c1.abcisse,
               c1.ordonnee );
      return 0;
}
```

Les structure en C : syntaxe

- Exemple

```
//Attention à l'ordre !!
struct Adresse{
        int num;
        char rue[200]; // Tableau de caractères
        char ville[200];
        int code;
};

struct Etudiant{
        char nom[200];
        char prenom[200];
        int groupe;
        int num;
        struct Adresse add; // Adresse doit etre décrite avant
};
```

Types équivalents

- Permet de renommer un type existant
- Permet de simplifier une écriture, de gagner en expressivité
- Exemple :

```
typedef type nouveau_nom ;

typedef type nouveau_nom1, nouveau_nom2, ... ;
```

Exemple

```
// Renommage
typedef float reel ;
typedef int entier ;

// Utilisation
entier a ;
reel x ;
```

Types équivalents

- Déclaration d'une structure + renommage en type :

```
// Dorénavant, nous l'écrirons comme ceci
typedef struct {
    char nom[200] ;
    char prenom[200] ;
} Auteur;

// Utilisation
Auteur tolkien, rowling ;
```

- Plus besoin d'utiliser struct à chaque déclaration !

Énumération

- Une **énumération** définit un nouveau type permettant à une variable de prendre un nombre **fini de valeurs** ;
- Exemples :
  - mois : JANVIER, FÉVRIER, MARS, ...
  - jours : LUNDI, MARDI, MERCREDI, ...
  - couleurCarte : PIQUE, COEUR, CARREAU, TREFLE
  - feuTtricolore : ROUGE, ORANGE, VERT
  - booléen : VRAI, FAUX
  - ...

Syntaxe

- Déclaration d'une **énumération**

```
enum nomEnumeration { VAL1, VAL2, …, VALN };
```

ou plutôt :

```
typedef enum {
    VAL1, VAL2, …, VALN
} nomEnumeration;
```

- Utilisation d'une **énumération**

```
enum nomEnumeration nomVariable;
```

ou plutôt :

```
nomEnumeration nomVariable;
```

Exemple

```
typedef enum { PIQUE, COEUR, CARREAU, TREFLE } Couleur;

typedef struct {
     char valeur ; /* 1->10, 11: valet, 12: dame, 13: roi */
     Couleur c_couleur ;
} Carte;

typedef Carte Deck[52];  // Deck definit un lot de 52 Cartes

/* Utilisation */
Deck jeu ;
jeu[0].valeur = 1;
jeu[0].c_couleur = TREFLE;
```

Énumération : Sous le capot

- Une valeur est affectée à chaque valeur de l'énumération ;
- Première valeur : 0 ;
- Itération pour les autres (+1 par défaut) ;
- Il est possible de :
  - démarrer avec une autre valeur,
  - donner une valeur différente à un élément de l'énumération:

```
typedef enum {
    FAIBLE = 10, MOYEN = 50, FORT = 100
}Volume;
```

Pour completer

- Enumeration des types de rue pour completer la structure Adresse:

```
typedef enum {
    RUE, BOULEVARD, IMPASSE, CHEMIN, AVENUE
} type_rue;
```

Pour completer

```
typedef enum {
    RUE, BOULEVARD, IMPASSE, CHEMIN, AVENUE
} Type_rue;

typedef struct{
        int num;
        Type_rue style; // Type de rue de l'adresse
        char rue[200];  // Nom de la rue
        char ville[200];
        int code;
} Adresse;

typedef struct {
        char nom[200];
        char prenom[200];
        int groupe;
        int num;
        Adresse add;
} Etudiant;
```

Structure générale d'un fichier .c

#Instructions du préprocesseur
(inclusion bibliothèque)
+ **définition constante**

Défintions structures
+ typedef

Code de la fonction 1

Code de la fonction 2

....

Code de la fonction main

Conclusion

- Il est possible de créer/personnaliser ses propres types.
- Une structure est composée de sous-variables appellées **champs ou attributs ou membres**.
- Pour acceder à un membre d'une variable de type structure:
  - nomVariable.membre
- L'énumeration est un type qui permet de prendre un valeur définie parmi d'autres.
- On peut faire de l'imbrication de structures/énumerations pour créer des types complexes.