# TP 10
Les structures cartésiennes

## Exercice 1

Dans cet exercice, on veut gérer des données sur des étudiants : chaque étudiant est caractérisé par son **nom** et par son **prénom** (chaînes de 20 caractères maximum) et par son **âge** (int).

**Question 1**
Définissez le type chaine20 et le type etudiant.

**Question 2**
Écrivez une procédure qui reçoit un étudiant en paramètre d'entrée et qui affiche ses caractéristiques, conformément à l'exemple suivant :

Etudiant : Jean ALBERT, 20 ans.

**Question 3**
Écrivez une procédure de saisie qui fournit en paramètre de sortie un étudiant après avoir initialisé ses caractéristiques par des lectures clavier.

**Question 4**
Écrivez une fonction `compare(etudiant e1, etudiant e2)` qui retourne :

- 0 si les deux étudiants ont le même âge,
- 1 si e1 est plus vieux que e2,
- -1 si e2 est plus vieux que e1.

**Question 5**
Écrivez un main qui teste les procédures et la fonction précédentes.

**Exercice 2**

Considérons les déclarations suivantes :

```
#define NM 12
typedef char t_chaine9[10];
typedef  struct{
      t_chaine9 c_nom;
      int       c_nbJours;
} t_mois;
typedef  t_mois t_tabMois[NM];

const t_tabMois  tMois= {     {"janvier",31},
                              {"fevrier",28},
                              {"mars",31},
                              {"avril",30},
                              {"mai",30},
                              {"juin",30},
                              {"juillet",30},
                              {"aout",30},
                              {"septembre",30},
                              {"octobre",30},
                              {"novembre",30},
                              {"decembre", 31}
                         };
```

**Question 1**

Écrire et tester une procédure qui affiche le nom des mois de 31 jours.

**Question 2**

Écrire et tester une fonction `int nombreJours(t_chaine9 nomMois)` qui délivre le nombre de jours du mois `nomMois` si `nomMois` est présent dans `tMois` *(constante globale)*, et `-1` (par convention) sinon.

Rappel : quand on commence la recherche, on ne sait pas combien de fois on va boucler, donc on utilise une boucle *tant que* avec 2 conditions.

**Question 3**

Écrire un programme utilisant la fonction précédente, qui :

- demande à l'utilisateur du programme le nom d'un mois,
- affiche le nombre de jours de ce mois si la chaîne tapée par l'utilisateur est bien un nom de mois, et un message d'erreur sinon.

**Exercice 3**

À l'occasion de l'épreuve du « contre-la-montre individuel » du Tour de France, on veut afficher le classement provisoire au fur et à mesure de l'arrivée des concurrents. Les concurrents partent **un par un**. À l'arrivée de chaque concurrent, on note le numéro de dossard et le temps mis pour effectuer le parcours.
On veut écrire un programme qui affiche le classement provisoire à l'arrivée de chaque concurrent. On utilisera donc un tableau de « concurrents à l'arrivée », un concurrent à l'arrivée étant défini par son numéro de dossard et son temps :

```
#define MAX 198   // 22 équipes de 9 coureurs = 198 coureurs
typedef struct{
      int c_numero;
      int c_temps;
}t_concurrent;
typedef t_concurrent t_tabconc[MAX];
```

**Question 1**

Écrire et tester une procédure

```
void insere( t_concurrent c, t_tabconc tc, int n )
```

qui insère le concurrent `c` à sa « vraie place » dans le tableau `tc` déjà rempli de *n* concurrents déjà classés. Pour cela, vous placerez *c* dans `tc[n]`, puis vous le ferez « remonter » à sa vraie place (méthode par permutation).

Pour permuter `tc[j]` et `tc[j-1]`, vous utiliserez une procédure

```
void permuter(t_concurrent* c1, t_concurrent* c2)
```

qui permute les concurrents `*c1` et `*c2`. Notez bien que la permutation se fait globalement, et non pas champ par champ.

**Question 2**

Écrire un programme qui lit au clavier le numéro et le temps de chaque concurrent qui vient d'arriver (l'utilisateur tape –1 à la place du numéro pour signaler la fin des données), et qui affiche le classement provisoire après l'arrivée de chaque concurrent.
Le programme utilisera la procédure `insere` de la question 1, et une procédure `affiche_classement ( t_tabconc tc, int n )` qui affiche le classement provisoire des `n` concurrents présents dans `tc`.

## Exercices complémentaires

### Exercice 4

Reprenons l'exerice 1

On dispose maintenant cette structure de date :

```
typedef struct {
	int jour;
	int mois;
	int annee;
} date;
```

Un étudiant est désormais caractérisé par une date de naissance à la place de son âge :

```
typedef struct {
	chaine20 nom;
	chaine20 prenom;
	date dateNaissance;
} etudiant;
```

**Question 1**
Écrivez une procédure `afficherAnnee(etudiant e)` qui affiche l'année de naissance de l'étudiant passé en paramètre.

**Question 2**
Écrivez une procédure de saisie qui fournit en paramètre de sortie un étudiant après avoir initialisé ses caractéristiques par des lectures clavier.

**Question 3**
Récrivez la fonction `compare(etudiant e1, etudiant e2)` qui retourne:

- 0 si les deux étudiants ont le même âge,
- 1 si `e1` est plus vieux que `e2`,
- -1 si `e2` est plus vieux que `e1`.

**Question 4**
Écrivez un main possédant deux variables de type `etudiant`, initialisées avec des constantes, et testez la procédure `afficher` et la fonction `compare`.

## Exercice 5

Reprenons l'exercice 3 : les concurrents passent toujours **un par un**, mais cette fois ils sont départagés suivant le principe défini comme suit (il ne s'agit plus du contre-la-montre individuel du Tour de France, mais par exemple d'un concours hippique) :

« *les concurrents sont départagés par des points de pénalité, et en cas d'égalité de points, par le temps mis pour accomplir l'épreuve* ».

Un concurrent est donc défini par son numéro de dossard, son nombre de points et son temps.
Le type *concurrent* devient :

```
typedef struct{
      int numero;
      int  points;
      int temps;
}t_concurrent;
```

**Question 1**
Écrire une fonction `compare` qui prend en paramètres deux concurrents, et qui délivre la valeur *ad hoc* du type énuméré comparaison, ce dernier étant défini comme suit :

```
typedef  enum {moins, egalite, mieux} t_comparaison;
```

N.B. La fonction retourne la valeur `mieux` si le premier concurrent passé en paramètre est mieux placé que le second concurrent passé en paramètre, `moins` s'il est moins bien placé, `egalite` autrement.

**Question 2**
Valider la fonction `compare` en utilisant un programme de test.

**Question 3**
Reprendre le programme de l'exercice 2, en utilisant la fonction `compare` pour classer les concurrents.

**Question 4**
Améliorer l'affichage des résultats en tenant compte du cas des *ex aequo* dans la procédure `affiche_classement` (là aussi on utilisera la fonction `compare`).