Langage de programmation L2

Le langage C

TD4

Type énumération

Un type énumération est un type particulier, définit par le développeur. Il est définit par son nom et la liste complète de toutes les valeurs autorisées. Par exemple, le type énuméré suivant se nomme t_couleur et les valeurs possibles pour une variable de type t_couleur sont bleu, vert et jaune :

```
enum t_couleur { bleu, vert, jaune  } ;
```

On peut alors déclarer une variable de type couleur et lui affecter l'une des valeurs autorisées :

```
enum t_couleur cou = vert ;
```

Pour simplifier l'écriture des déclarations, on peut utiliser typedef, comme pour les structures :

```
typedef enum { bleu, vert jaune } t_couleur ;

t_couleur cou = vert ;
```

En terme de représentation en mémoire, une valeur d'un type énuméré est un entier. La première valeur d'énumé est 0, la second, 1, etc. L'instrcution suivante est donc valide :

```
printf(«%d », cou) ; // Affiche 1 sur la sortie standard
```

Exercice

On veut pouvoir afficher le nom d'une couleur associé à une valeur d'énuméré. La solution est d'utiliser un tableau de chaines de caractères. Pour le type t_couleur, on peut déclarer le tableau suivant :

```
char * nom_couleurs [ ] = { « bleu », « vert », « jaune » } ;
```

Ecrire la fonction *couleur_affiche*, qui prend une couleur en argument et affiche le nom de la couleur :

```
couleur_affiche (t_couleur c) ;
```

Type union

Une union est un type structure particulier qui permet de stocker différentes valeurs à la même adresse mémoire. Le bénéfice est qu'il est possible d'utiliser la même zone mémoire pour des utilisations différentes.

La définition et l'utilisation d'un type union est identique à celle d'une structure (au mot clé près) :

```
typedef union {
      int i ;
      char s [125] ;
} t_foo;

t_foo crazy, mob ;

crazy.i = 2 ;
printf («%d », crazy.i) ; // Affiche 2

strcpy(mob.s, « mobylette ») ;
printf(«%s », mob.s) ; // Affiche mobylette
```

Exercice

Le programme suivant compile et donne un résultat bizarre. Expliquez le résultat.

```
#include <stdio.h>
#include <string.h>

typedef union
{
   int i;
   float f;
   char  str[20];
} t_data;

int main( )
{
   t_data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "Mobylette");

   printf( "data.i : %d\n", data.i); // Affiche 1917853763
   printf( "data.f : %f\n", data.f); // Affiche 412236058032779486045368.000000
   printf( "data.str : %s\n", data.str); // Affiche Mobylette

   return 0;
}
```

Application

Voici des structures *C* développées pour manipuler des figures géométriques.

```
typedef struct {
  int x, y, cote;
} t_carre;

typedef struct {
  int x, y, longueur, largeur;
} t_rectangle;
```

Un carré se manipule donc par les coordonnées x, y du coin haut-gauche et par la valeur de son coté, un rectangle, par x, y, sa longueur et sa largeur.

Fonctions pour les carrés et les rectangles

Mettre en oeuvre les fonctions suivantes pour les carres:

- *void carre_creer(t_carre * self, int x, int y, int cote)*

  permet d'initialiser un carre;
- *void carre_detruire(t_carre * self)*

  permet de détruire un carre;
- *void carre_modifier(t_carre * self, int x, int y, int cote)*

  permet de modifier un carre;
- *void carre_afficher(t_carre * self)*

  permet d'afficher le contenu d'un carre sur la sortie standard;

- *void carre_copier(t_carre * self, t_carre * a_copier)*

  permet de copier un carre;

Pour la suite, on considère que les mêmes fonctions sont aussi mises en œuvre pour les rectangles.

Gestion d'un tableau de formes géométriques

Voici une structure permettant le stockage d'une liste de formes géométriques.

Le type *t_forme* permet de manipuler un carre ou un rectangle : si la valeur de tfg est *_carre_*, alors, le pointeur geo doit pointer sur une structure de type *t_ carre* et si la valeur de tfg est _rectangle_, alors, geo doit pointer sur une structure de type *t_rectangle*.

```
typedef enum { _carre_, _rectangle_ } type_fg;

typedef struct _forme {
  type_fg tfg;
  void * geo;
} t_forme;

typedef struct {
      int nb ;
      t_forme ** tab ;
} t_tab_formes ;
```

Mettre en œuvre un programme qui demande à l'utilisateur quel type de forme ajouter (rectangle ou carre) et qui en fonction de ce qui est demandé, crée la forme géométrique et l'ajoute à un *t_tab_formes*.

L'utilisateur pour aussi choisir d'afficher toutes les formes contenues.