Examen de TD n°2

La durée de l'examen est de 60 minutes, les notes de cours et de TD sont autorisées. La notation prendra en compte la présentation et la clarté des explications (programmes commentés, dessins explicatifs, etc.). L'examen sera noté sur 10 points et le barème est donné uniquement à titre indicatif. Il pourra être modifié lors de la notation finale.

Cet examen porte sur la représentation de graphes orientés en C. Un graphe orienté $G = (S, A)$ est constitué d'un ensemble S de sommets et d'un ensemble A d'arêtes (c'est à dire de couples de sommets). On peut donc considérer qu'un sommet est connecté à un ensemble d'autres sommets (voir figure 1).

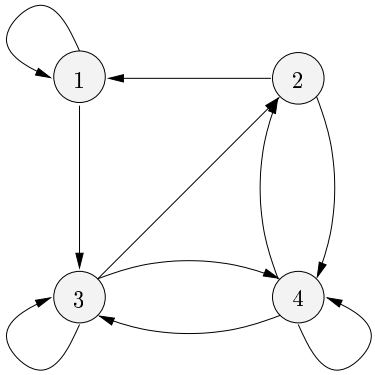

FIG. 1 – Un exemple de graphe orienté

Sur le graphe de la figure 1, les voisins du sommet étiqueté par 3 sont donc lui-même, le sommet étiqueté par 2 et le sommet étiqueté par 4.

Les types utilisés dans cette épreuve seront les suivants :

```
1  typedef struct s_sommet_maillon *p_sommet_maillon_t;
2  typedef p_sommet_maillon_t       sommet_liste_t;
3  typedef sommet_liste_t          *p_sommet_liste_t;
4  typedef struct s_sommet         *sommet_t;
5
6  typedef struct s_sommet_maillon {
7    sommet_t noeud;
8    sommet_liste_t suivant;
9  } sommet_maillon_t;
10
11 typedef struct s_sommet {
12   int etiquette;
13   sommet_liste_t voisins;
14 } s_sommet_t;
```

Un sommet est donc composé d'une étiquette de type `int` et d'une liste de sommets qui sont ses voisins.

▷ **Question 1. (0.5 points)** Écrire une fonction en C `nil` dont le prototypage est :

```
1    sommet_liste_t nil()
```

et qui renvoie une liste de sommets vide.

Réponse.

```
sommet_liste_t nil()
{
  return NULL;
}
```

□

▷ **Question 2. (1 points)** Écrire une fonction en C `cons` dont le prototypage est :

```
   sommet_liste_t cons(sommet_t sommet, sommet_liste_t sommet_liste)
```

et qui renvoie une liste dont le premier élément contient le sommet passé en argument et dont les éléments suivants sont ceux de la liste passée en argument.

Réponse.

```
sommet_liste_t cons(sommet_t sommet, sommet_liste_t sommet_liste)
{
  sommet_liste_t tete = NULL;

  if(sommet) {
    tete=malloc(sizeof(sommet_maillon_t));
    tete->noeud = sommet;
    tete->suivant = sommet_liste;
  }
  return tete;
}
```

□

▷ **Question 3. (1 points)** Écrire une fonction en C `car` dont le prototypage est le suivant :

```
   sommet_t car(sommet_liste_t liste)
```

et qui renvoie le sommet contenu dans le premier élément de la liste si cette dernière est non vide et qui affiche un message d'erreur et sort du programme sinon.

Réponse.

```
sommet_t car(sommet_liste_t liste)
{
  if(liste == nil()) {
    printf("Le car d'une liste vide, c'est pas terrible !!\n");
    exit(1);
  }
  return liste->noeud;
}
```

□

▷ **Question 4. (1 points)** Écrire une fonction en C `cdr` dont le prototypage est le suivant :

```
   sommet_liste_t cdr(sommet_liste_t liste)
```

et qui renvoie la liste vide si elle est appliquée à la liste vide et la liste privée de son premier élément sinon.

Réponse.

```
sommet_liste_t cdr(sommet_liste_t liste)
{
  if(liste == nil()) {
    printf("Le cdr d'une liste vide... disons vide par convention !\n");
    return(nil());
  }
  return liste->suivant;
}
```

□

▷ **Question 5. (1.5 points)** Écrire une fonction en C `new_sommet` dont le prototypage est le suivant :

```
    sommet_t new_sommet(int etiquette)
```

et qui créée un sommet étiqueté par la valeur passée en argument et qui ne possède pas de voisin.

Réponse.

```
sommet_t new_sommet(int etiquette) {
  sommet_t new = malloc(sizeof(s_sommet_t));

  new->etiquette = etiquette;
  new->voisins = nil();
  return new;
}
```

□

▷ **Question 6. (2 points)** Écrire une fonction en C `add_voisin` dont le prototypage est le suivant :

```
    void add_voisin(sommet_t sommet,sommet_t voisin)
```

et qui ajoute le sommet `voisin` à la liste des voisins du sommet `sommet`.

Réponse.

```
void add_voisin(sommet_t sommet,sommet_t voisin) {
  if(sommet) {
    sommet->voisins = cons(voisin,sommet->voisins);
  } else {
    printf("Attention, add_voisin devrait modifier un sommet vide\n");
    exit(1);
}
```

□

▷ **Question 7. (2 points)** Écrire une fonction en C `affiche_voisins` dont le prototypage est le suivant :

```
    void affiche_voisins(sommet_t sommet)
```

et qui affiche d'abord l'étiquette du sommet si celle-ci existe, puis un " :" puis les étiquettes de ses voisins séparées par des espaces.

Réponse.

```
void affiche_voisins(sommet_t sommet) {
  sommet_liste_t voisin=NULL;

  if(sommet) {
    printf("%d : ", sommet->etiquette);
    voisin=sommet->voisins;
    while(voisin) {
      printf("%d ", car(voisin)->etiquette);
      voisin=cdr(voisin);
    }
    printf("\n");
  }
}
```

□

▷ **Question 8. (1 points)** Écrire une fonction principale en C (un `main()` qui crée le graphe de la figure 1 et affiche la liste des voisins de chacun des sommets. *Un point supplémentaire pour ceux qui sont capables de m'écrire ce que leur programme affiche.*

Réponse.

```
int main()
{
  sommet_t s1,s2,s3,s4;

  s1=new_sommet(1);
  s2=new_sommet(2);
  s3=new_sommet(3);
  s4=new_sommet(4);

  add_voisin(s1,s1);
  add_voisin(s1,s3);

  add_voisin(s2,s1);
  add_voisin(s2,s4);

  add_voisin(s3,s2);
  add_voisin(s3,s3);
  add_voisin(s3,s4);

  add_voisin(s4,s2);
  add_voisin(s4,s3);
  add_voisin(s4,s4);

  affiche_voisins(s1);
  affiche_voisins(s2);
  affiche_voisins(s3);
```

```
27   affiche_voisins(s4);
28
29   return 0;
30 }
```

En ce qui concerne la sortie affichée par ce programme, il faut juste faire attention à l'ordre dans lequel on insère les voisins d'un sommet...

```
moby:~/Work/Documents/Enseignements/DEUG-MIAS-C $ ./graphe
1 : 3 1
2 : 4 1
3 : 4 3 2
4 : 4 3 2
```

□