L3 - SRI
CC2 - Informatique
Décembre 2017 – 1 heure 30
(documents autorisés)

Pour être comptabilisée, toute réponse devra être justifiée.

1 Compréhension d'un programme C (10 points)

Soit le programme C suivant :

```
// definitions pour les types
// ////////////////////////////
typedef struct etiq1 {
   int champ1 ;
   struct etiq1 * champ2 ;
   } Type1 ;
typedef Type1 * Type2;
typedef  struct etiq2 {
   Type2 champ1 ;
   int champ2 ;
   } Type3 ;

// fonctions pour le Type2
// ////////////////////////////
void fonction1Type2(Type2 o) {
   *o = NULL ;
   }
Type2 fonction2Type2(Type2 o, int x) {
   Type2 o2 = (Type2)malloc(sizeof(Type1)) ;
   assert(o2 != NULL) ;
   o2.champ1 = x ;
   o2.champ2 = o ;
   return o2 ;
   }
int fonction3Type2(Type2 o) {
   return (o == NULL) ;
   }


// fonctions pour le Type3
// ////////////////////////////
void fonction1Type3(Type3 * o) {
   o->champ1 = NULL ;
   o->champ2 = 0 ;
   }
Type3 fonction2Type3(Type3 o, int x) {
   o.champ1 = fonction2Type2(o.champ1,x) ;
   o.champ2 ++ ;
   return o ;
   }
```

```
void fonction3Type3(Type3 o) {
   Type2 o2 = o.champ1 ;
   for (int i = 0 ; i <= o.champ2; i++) {
      printf("%d ",o2->champ1) ;
      o2 = o2->champ2 ;
      }
   }
```

```
// le programme de test
// ////////////////////////
int main () {
  Type3 o ;

  fonction1Type3(&o) ;

  o = fonction2Type3(o,10);
  o = fonction2Type3(o,20);
  o = fonction2Type3(o,30);
  o = fonction2Type3(o,40);

  fonction3Type3(o) ;
}
```

1. Il y a 4 erreurs dans ce programme (3 provoquant une erreur de compilation et 1 risquant de provoquer une erreur à l'exécution). Trouvez-les et expliquez comment les corriger.
2. Une fois le programme corrigé, décrivez son exécution et donnez les résultats obtenus.
3. En vous inspirant des structures de données vues en TD et en TP, dites ce que représentent les `Type1` et `Type2`. Proposez alors des noms plus explicites pour chaque type, chaque champ de structure et chaque fonction.
4. Transformez la fonction `fonction3Type3` de manière à parcourir la structure de données de manière récursive terminale.

2 Parcours en largeur/profondeur d'un graphe (10 points)

On veut faire l'implémentation en C de l'algorithme de parcours en largeur/profondeur pour un graphe orienté noté G :

Algorithm 1: Parcours en largeur[profondeur] d'un graphe

Input: G : graphe connexe orienté
i_0 : sommet de G à partir duquel on commence le parcours
Data: $OUVERT$: sommets en attente d'être traités
$FERMÉ$: sommets déjà traités
i : sommet courant
Result: A : l'arborescence représentant le parcours
begin
  initialiser $OUVERT$ et $FERMÉ$ à vide;
  placer le sommet initial i_0 sans père dans $OUVERT$;
  initialiser A à vide ;
  while OUVERT *n'est pas vide* **do**
    $i \leftarrow$ le premier élément de $OUVERT$;
    supprimer i de $OUVERT$;
    if i *n'est pas dans* FERMÉ **then**
      mettre les successeurs de i qui $\notin FERMÉ$ en queue[tête] de $OUVERT$ (en mémorisant que i est leur père);
      mettre i dans A avec l'arc le liant à son père;
      mettre i dans $FERMÉ$;
  return A ;

Pour réaliser ce travail, vous disposerez des unités suivantes (que vous n'aurez donc pas à écrire) :

```
// fichier : sommet.h
// Unité Sommet
// ///////////////
typedef struct etSommet * Sommet ;

// récupération du père du sommet s
Sommet getPere(Sommet s) ;
// mise à jour du père de s avec p
Sommet setPere(Sommet s, Sommet pere) ;
```

```
// fichier : graphe.h
// Unité Graphe
// ///////////////
typedef struct etGraphe * Graphe ;

// creation d'un graphe vide
Graphe initGraphe() ;
// ajout du sommet s au graphe g et récupération du résultat
Graphe addSommet(Graphe g, Sommet s) ;
// ajout de l'arc (depart,arrivee) au graphe g et récupération du résultat
Graphe addArc(Graphe g, Sommet depart, Sommet arrivee) ;
// récupération des successeurs de s dans g
Liste successeurs(Graphe g, Sommet s) ;
```

```
// fichier : liste.h
// Unité Liste (de Sommet)
// ////////////////
typedef struct etListe * Liste ;

// création d'une liste vide
Liste initListe() ;
// récupération du premier élément de la liste
Sommet getTopListe(Liste l) ;
// suppression du premier élément de la liste et récupération du résultat
Liste removeTopListe(Liste l) ;
// ajout du sommet s en tête de la liste l et récupération du résultat
Liste addSommetEnTete(Liste l, Sommet s) ;
// ajout du sommet s en queue de la liste l et récupération du résultat
Liste addSommetEnQueue(Liste l, Sommet s) ;
// vérification de l'état de la liste l
//        (renvoie 1 si la liste l est vide et 0 sinon)
int estVide(Liste l)
// vérification de l'appartenance du sommet s à la liste l
//        (renvoie 1 si s est dans l et 0 sinon)
int appartient(Liste l, Sommet s)
```

Votre travail sera le suivant :

1. Utilisez l'unité `Liste` pour définir une unité `Pile`. Pour cela, vous donnerez le fichier header (`pile.h`) et le fichier source (`pile.c`) avec le code des fonctions `creerPile`, `empiler`, `depiler`, `getTopPile`, `pileEstVide` et `appartientPile`.
2. Utilisez l'unité `Liste` pour définir une unité `File`. Pour cela, vous donnerez le fichier header (`file.h`) et le fichier source (`file.c`) avec le code des fonctions `creerFile`, `enfiler`, `defiler`, `getTopFile`, `fileEstVide` et `appartientFile`.
3. Donnez le code C correspondant à la traduction de l'algorithme pour sa version *en largeur* en utilisant une des deux unités définies précédemment (`Pile` ou `File`).
4. Donnez le code C correspondant à la traduction de l'algorithme pour sa version *en profondeur* en utilisant une des deux unités définies précédemment (`Pile` ou `File`).