L3 - SRI
CC2 - Informatique
Décembre 2021 – 1 heure 30
(documents autorisés)

Pour être comptabilisée, toute réponse devra être justifiée.

1 Compréhension d'un programme C (8 points)

Soit le programme C suivant :

Type

```
typedef struct ettruc {
    int * champ1 ;
    int champ2 ;
    int champ3 ;
    struct ettruc * champ4 ; } * Truc ;
```

Fonctions

```
void fonction1 (Truc t, int n) {
    t = (Truc)malloc(sizeof(struct ettruc)) ;
    t->champ1 = (int *) malloc(sizeof(int)*n) ;
    t->champ2 = n ;
    t->champ3 = 0 ;
    t->champ4 = NULL ;
    }
Truc fonction2 (Truc t, int v) {
    if (t->champ2 == t->champ3) {
        Truc taux = (Truc)malloc(sizeof(Truc)) ;
        taux->champ1 = (int *) malloc(sizeof(int)*t->champ2) ;
        taux->champ2 = t->champ2 ;
        taux->champ3 = 0 ;
        taux->champ4 = t ;
        t = taux ;
        }
    *((t->champ1)+champ3) = v ;
    t->champ3 ++ ;
    return t ;
    }
void fonction3 (Truc t) {
    if (t != NULL) {
         fonction3(t.champ4) ;
         for (int i = 0; i < t.champ3; i++)
             printf("%d ",*((t->champ1)+i)) ;
         }
    }
```

Programme de test

```
int main() {
  Truc t ;
  fonction1(t,2) ;                          // etape 1
  printf("t : ") ;  fonction3(t) ;          // etape 2
  t = fonction2(t,10) ;                     // etape 3
  t = fonction2(t,20) ; t = fonction2(t,0) ;  // etape 4
  printf("t : ") ;  fonction3(t) ; }        // etape 5
```

1. Réécrire l'extrait de code `*((t->champ1)+i)` en considérant que `t->champ1` est un tableau et en utilisant les notations sur les tableaux.
2. Expliquer ce qu'il se passe lors des passages de paramètre pour les étapes 1 et 3 et quels sont les impacts sur la variable `t` du `main`.
3. En utilisant votre réponse à la question précédente, analyser tout le code en identifiant :
   — les erreurs éventuelles,
   — les corrections possibles pour ces erreurs
   — et le résultat des affichages après correction.
4. Y-a-t-il des fonctions récursives ? Si oui, dire lesquelles et donner leur type (primitive, terminale, transformable terminale) en justifiant votre réponse.

2 Problème (12 points)

On veut implémenter une version en langage C des ArrayList [1] contenant des éléments de type TElem. Cette implémentation va se faire sous la forme d'une liste chaînée de tableaux d'éléments (chaque tableau étant de la même taille et cette taille étant définie lors de l'initialisation).

1. Donner un type C permettant de simuler une ArrayList<TElem> sachant que le type TElem des éléments est considéré comme connu.

Puis écrire en C les fonctions suivantes **(une attention particulière devra être portée à la robustesse et à l'efficacité de votre code, ainsi qu'aux aspects "consommation mémoire")** :

2. `initArrayList` : prend en paramètre la taille pour les tableaux et renvoie une ArrayList <TElem> ne contenant pour l'instant aucun élément,
3. `addArrayList` : prend en paramètres une ArrayList<TElem> et un élément de type TElem et renvoie l'ArrayList<TElem> passée en paramètre dans laquelle on aura ajouté en fin de liste l'élément passé en paramètre
4. `printArrayList` : prend en paramètre une ArrayList<TElem> et affiche son contenu de la première valeur à la dernière ; on considère qu'il existe une fonction `void afficheElement (TElem e)` qui se charge de l'affichage d'un élément donné
5. `isEmptyArrayList` : prend en paramètre une ArrayList<TElem> et renvoie *true* si cette liste est vide et *false* sinon
6. `sizeArrayList` : prend en paramètre une ArrayList<TElem> et renvoie le nombre d'éléments stockés dans cette liste (on ne demande pas ici une version très optimisée)
7. `getArrayList` : prend en paramètres une ArrayList<TElem> et un entier i et renvoie l'élément qui est en position i dans la liste (i devra être ≥ 0 et $<$ à la taille de la liste) ; la position 0 correspondra au premier élément de la liste
8. `removeArrayList` : prend en paramètres une ArrayList<TElem> et un entier i et renvoie l'ArrayList<TElem> passée en paramètre de laquelle on aura enlevé l'élément qui était en position i (i devra être ≥ 0 et $<$ à la taille de la liste). **Attention, votre solution devra optimiser en priorité le temps d'exécution de la fonction, puis dans un second temps la place mémoire, tout en évitant les fuites mémoire !**
9. Expliquer comment votre code est organisé en terme de découpage en plusieurs fichiers. Vous considérerez que les fichiers `elem.h` et `elem.c` définissant l'unité pour le type TElem sont donnés.

1. Structure de données bien connue des programmeurs Java !

Corrigé

Exo 1 : Compréhension d'un programme C (8 points)

Soit le programme C suivant :

////////////// Type ///////////////

```
typedef struct ettruc {
    int * champ1 ;
    int champ2 ;
    int champ3 ;
    struct ettruc * champ4 ; } * Truc ;
```

////////////// Fonctions ///////////////

```
void fonction1 (Truc t, int n) {
    t = (Truc)malloc(sizeof(struct ettruc)) ;
    t->champ1 = (int *) malloc(sizeof(int)*n) ;
    t->champ2 = n ;
    t->champ3 = 0 ;
    t->champ4 = NULL ;
    }
Truc fonction2 (Truc t, int v) {
    if (t->champ2 == t->champ3) {
        Truc taux = (Truc)malloc(sizeof(Truc)) ;
        taux->champ1 = (int *) malloc(sizeof(int)*t->champ2) ;
        taux->champ2 = t->champ2 ;
        taux->champ3 = 0 ;
        taux->champ4 = t ;
        t = taux ;
        }
    *((t->champ1)+champ3) = v ;
    t->champ3 ++ ;
    return t ;
    }
void fonction3 (Truc t) {
    if (t != NULL) {
         fonction3(t->champ4) ;
         for (int i = 0; i < t.champ3; i++)
             printf("%d ",*((t.champ1)+i)) ;
         }
    }
```

////////////// Programme de test ///////////////

```
int main() {
  Truc t ;
  fonction1(t,2) ;                              // etape 1
  printf("t : ") ;  fonction3(t) ;              // etape 2
  t = fonction2(t,10) ;                         // etape 3
  t = fonction2(t,20) ; t = fonction2(t,0) ;    // etape 4
  printf("t : ") ;  fonction3(t) ;              // etape 5
```

1. Réécrire l'extrait de code `*((t->champ1)+i)` en considérant que `t->champ1` est un tableau et en utilisant les notations sur les tableaux.

   Solution : *On aura :* `t->champ1[i]`

2. Expliquer ce qu'il se passe lors des passages de paramètre pour les étapes 1 et 3 et quels sont les impacts sur la variable `t` du `main`.

Solution : *Etape 1 : Passage par valeur, donc la valeur du* `t` *du* `main` *est recopiée dans le* `t` *de* `fonction1` *; puis ce* `t` *de* `fonction1` *est maj par* `fonction1` *mais comme ce n'est qu'une recopie du* `t` *du* `main`*, le* `t` *du* `main` *ne change pas de valeur.*
Etape 3 : on a la même chose mais comme il y a un `return` *du* `t` *de* `fonction2` *à la fin de l'exécution de* `fonction2` *et que le* `t` *du* `main` *reçoit cette valeur de retour, du coup, le* `t` *du* `main` *va être modifié par l'exécution de* `fonction2`

3. En utilisant votre réponse à la question précédente, analyser tout le code en identifiant :
   — les erreurs éventuelles,
   — les corrections possibles pour ces erreurs,

   Solution : *Plusieurs erreurs de syntaxe et de sémantique :*

   (a) ds `fonction1`*, 1 erreur de sémantique : il faudrait que la maj du* `t` *de* `fonction1` *ait un impact sur le* `t` *du* `main` *; donc 2 solutions :*
   — *soit on fait un* `return t ;` *et on modifie l'appel dans le* `main` *par* `t = fonction1(t,2) ;`
   — *soit on rajoute une* `*` *devant le* `t` *ds les paramètres et sur la ligne du* `malloc`*, on remplace chaque autre* `t` *dans le corps par* `(*t)` *et on change l'appel ds le* `main` *par* `fonction1(&t,2) ;`

   (b) ds `fonction2`*, 2 erreurs de syntaxe : ds le* `sizeof` *le type est erroné (il faut* `struct ettruc`*) et dans l'expression* `*((t->champ1)+champ3)` *l'utilisation de* `champ3` *n'est pas possible (il faut mettre* `t->champ3`*)*

   (c) ds `fonction3`*, erreurs de syntaxe :* `t` *étant un pointeur sur une structure l'accès aux champs se fait avec* `->` *et pas avec* `.`

   Ce code crée une liste chainée composée de tableaux de 2 cases. Dans la première cellule, on place les valeurs 10, puis 20. Puis comme le tableau de la première cellule est plein, on crée une seconde cellule dans laquelle on met 0. L'affichage se fait ensuite dans l'ordre des valeurs insérées
   Bonus : la structure de données `Truc` *correspond à* ArrayList *qu'on implémente ds l'exo 2.*

   — et le résultat des affichages après correction.

   Solution : *Le premier affichage donne : (la liste est vide)*
   `t :`
   Le premier affichage donne : (la liste contient 3 valeurs)
   `t :10 20 0`

4. Y-a-t-il des fonctions récursives ? Si oui, dire lesquelles et donner leur type (primitive, terminale, transformable terminale) en justifiant votre réponse.

   Solution : *Il y a 1 fonction récursive :* `fonction3`*. Elle est primitive puisqu'elle n'apparaît pas ds les arguments de l'appel récursif. Elle n'est pas terminale puisqu'il reste des actions à faire après l'appel récursif (la boucle* `for`*). Et elle n'est pas transformable-terminale puisqu'on ne peut pas exhiber une loi associative liant l'appel et ce qui reste à faire après l'appel.*

Exo 2 : Problème (12 points)

On veut implémenter une version en langage C des ArrayList [2] contenant des éléments de type TElem. Cette implémentation va se faire sous la forme d'une liste chaînée de tableaux d'éléments (chaque tableau étant de la même taille et cette taille étant définie lors de l'initialisation).

1. Donner un type C permettant de simuler une ArrayList<TElem> sachant que le type TElem des éléments est considéré comme connu.

   Solution : *Le type de l'exo1 peut être utilisé ici après une légère adaptation (meilleur choix des identificateurs) :*

2. Structure de données bien connue des programmeurs Java !

```
typedef struct etAList {
    TElem * tabElements ;          // le tableau des elements
    int tailleTab ;                // la taille de ce tableau
    int nbElements ;               // le nb d'elements significatifs ds le tableau
    struct etAList * celSuivante ; // l'accès à la cellule suivante de la liste
    } * ArrayList ;
```

Puis écrire en C les fonctions suivantes **(une attention particulière devra être portée à la robustesse et à l'efficacité de votre code, ainsi qu'aux aspects "consommation mémoire")** :

2. `initArrayList` : prend en paramètre la taille pour les tableaux et renvoie une ArrayList <TElem> ne contenant pour l'instant aucun élément,

   Solution : *Il s'agit ici de l'adaptation de la* `fonction1` *de l'exo1 :*

   ```
   ArrayList initArrayList  (int n) {
       ArrayList l = (ArrayList)malloc(sizeof(struct etAList)) ;
       l->tabElements = (TElem *) malloc(sizeof(TElem)*n) ;
       l->tailleTab = n ;
       l->nbElements = 0 ;
       l->celSuivante = NULL ;
       }
   ```

3. `addArrayList` : prend en paramètres une ArrayList<TElem> et un élément de type TElem et renvoie l'ArrayList<TElem> passée en paramètre dans laquelle on aura ajouté en fin de liste l'élément passé en paramètre

   Solution : *Il s'agit ici de l'adaptation de la* `fonction2` *de l'exo1 :*

   ```
   ArrayList addArrayList (ArrayList l, TElem v) {
       if (l->tailleTab == l->nbElements) {
           ArrayList laux = (ArrayList)malloc(sizeof(struct etAList)) ;
           laux->tabElements = (int *) malloc(sizeof(int)*l->tailleTab) ;
           laux->tailleTab = l->tailleTab ;
           laux->nbElements = 0 ;
           laux->celSuivante = l ;
           l = laux ;
           }
       *((l->tabElements)+(l->nbElements)) = v ;
                       // ou bien l->tabElements[l->nbElements] = v
       l->nbElements ++ ;
       return l ;
       }
   ```

4. `printArrayList` : prend en paramètre une ArrayList<TElem> et affiche son contenu de la première valeur à la dernière ; on considère qu'il existe une fonction `void afficheElement (TElem e)` qui se charge de l'affichage d'un élément donné

   Solution : *Il s'agit ici de l'adaptation de la* `fonction3` *de l'exo1 :*

   ```
   void printArrayList (ArrayList l) {
       if (l != NULL) {
            printArrayList(l->celSuivante) ;
            for (int i = 0; i < l->nbElements; i++)
                afficheElement(l->tabElements[i]) ;
            }
       }
   ```

5. `isEmptyArrayList` : prend en paramètre une ArrayList<TElem> et renvoie *true* si cette liste est vide et *false* sinon

   Solution : *Une liste sera vide si elle n'a pas de suivant et que le nb d'éléments significatifs du tableau est 0. On peut aussi ajouter le cas où elle serait = à NULL.*

```
int isEmptyArrayList(ArrayList l) {
   return (l == NULL || (l->celSuivante == NULL && l->nbElements == 0)) ;
   }
```

6. `sizeArrayList` : prend en paramètre une ArrayList<TElem> et renvoie le nombre d'éléments stockés dans cette liste (on ne demande pas ici une version très optimisée)

   Solution : *Ici 2 solutions : soit on fait la somme du nb d'élements de chaque cellule, soit on ajoute un champ à la structure comptabilisant ce nb (donc modif du type). Ici on va juste donner la version 1.*

```
int sizeArrayList(ArrayList l) {
   if (l == NULL) return 0 ;
   return l->nbElements + sizeArrayList(l->celSuivante) ;
   }
```

7. `getArrayList` : prend en paramètres une ArrayList<TElem> et un entier i et renvoie l'élément qui est en position i dans la liste (i devra être ≥ 0 et $<$ à la taille de la liste) ; la position 0 correspondra au premier élément de la liste

   Solution : *Attention, la liste étant construite ici comme une pile, on a d'abord accès aux derniers éléments !*

```
TElem getArrayList(ArrayList l, int i) {
   int posFinTab = sizeArrayList(l) - 1 ; // correspondra tjrs a la position
                                          // du dernier element du tableau
                                          // de la cellule courante
   while (1) {
      if (i < posFinTab - l->nbElements +1) {
          // posFinTab - l->nbElements +1 correspond au début du tableau
          // la case i n'est pas ds la cellule courante
          posFinTab = posFinTab - l->nbElements ;
          l = l->celSuivante ;
          }
      else  {
          // la case i est ds la cellule courante
          return (l->tabElements[l->nbElements - posFinTab + i -1]);
          }
      }
   }
```

8. `removeArrayList` : prend en paramètres une ArrayList<TElem> et un entier i et renvoie l'ArrayList<TElem> passée en paramètre de laquelle on aura enlevé l'élément qui était en position i (i devra être ≥ 0 et $<$ à la taille de la liste). **Attention, votre solution devra optimiser en priorité le temps d'exécution de la fonction, puis dans un second temps la place mémoire, tout en évitant les fuites mémoire !**

   Solution : *Puisqu'on cherche à optimiser, on va donc réduire au max les décalages. On se contentera de faire les décalages ds la cellule courante uniquement et à rechaîner si jamais cette cellule ne contient plus rien de significatif. On reprend la même architecture de code que celle pour le get.*

```
ArrayList removeArrayList(ArrayList l, int i) {
   ArrayList lc = l ;                     // acces cellule courante
   ArrayList lplc = NULL ;                // acces cellule précédente de la lc
   int posFinTab = sizeArrayList(l) - 1 ; // correspondra tjrs a la position
                                          // du dernier element du tableau
                                          // de la cellule courante
   while (1) {
      if (i < posFinTab - lc->nbElements +1) {
          // posFinTab - l->nbElements +1 correspond au début du tableau
          // la case i n'est pas ds la cellule courante
          posFinTab = posFinTab - lc->nbElements ;
```

```
            lplc = lc ;
            lc = lc->celSuivante ;
            }
        else  {
            // la case i est ds la cellule courante
            if (lc->nbElements == 1) {
               // un seul element ds la cellule : celui qu'on remove
               if (lplc != NULL)
                  lplc->celSuivante = lc->celSuivante ;
               else l = lc->celSuivante ;
               free(lc->tabElements) ;
               free(lc) ;
               break ;
               }
            else {
               // decalage à faire jusqu'à la position correspondant à i
               for (int j=lc->nbElements-1;j>=lc->nbElements-posFinTab+i;j--)
                  lc->tabElements[j-1] = lc->tabElements[j] ;
               lc->nbElements -- ;
               break ;
               }
            }
        }
    return l ;
    }
```

9. Expliquer comment votre code est organisé en terme de découpage en plusieurs fichiers. Vous considérerez que les fichiers `elem.h` et `elem.c` définissant l'unité pour le type TElem sont donnés.

   Solution : *Il va falloir ici faire de la compilation séparée. On a donc besoin de :*
   - *un fichier* `arrayList.h` *contenant l'*`include` *de* `elem.h` *puis le type* `ArrayList` *et les prototypes de toutes les fonctions sur les* ArrayList
   - *un fichier* `arrayList.c` *contenant l'*`include` *de* `arrayList.h` *puis le corps de toutes les fonctions sur les* ArrayList
   - *un fichier de tests avec l'*`include` *de* `arrayList.h` *et le* `main` *serait aussi utile.*