UFR d'Ingénierie

18 février 2020

L1 (2019-2020)

Éléments de programmation en C (LU1IN002)

TD

Semaines 7 à 11

Semaine 7 - TD

Objectifs

— Structures
— Tableaux de structures

Exercices

Exercice 26 – Structures et fonctions

L'objet de base que nous considérons dans cet exercice est le point. Il est défini par ses coordonnées x et y (de type entier) et une couleur d'affichage (de type chaîne de caractères).

Question 1
Expliquez pourquoi un point ne peut pas être représenté par un tableau à trois éléments. Quel type est alors le plus adapté ?

Question 2
Définissez le type `point` permettant de représenter un point. La taille de la chaîne de caractères représentant la couleur sera limitée à 10 caractères (n'oubliez pas de réserver la place pour le caractère '`\0`' qui signale la fin de chaîne). Déclarez et initialisez ensuite un point nommé `mon_point`, situé aux coordonnées `x=3`, `y=4` et de couleur rouge. Pour recopier une chaîne de caractères, vous devrez utiliser la fonction `strcpy` qui prend en paramètres deux chaînes de caractères et qui recopie la valeur de la deuxième chaîne dans la première.

Question 3
Nous voulons maintenant pouvoir décrire des rectangles dont les côtés sont horizontaux et verticaux. Un rectangle est alors déterminé par les coordonnées de deux coins opposés (en bas à gauche et en haut à droite), la couleur d'affichage des traits et celle du fond. Définissez une structure `rectangle` qui utilise la structure `point`.

Question 4
Déclarez et initialisez une variable `mon_rectangle` de type `rectangle` dont les deux sommets opposés sont respectivement aux coordonnées (100, 200) et (300, 2), la couleur d'affichage des traits et des points est le rouge et la couleur du fond est le blanc. Nous vous rappelons que le repère associé à une fenêtre a son origine dans le coin en haut à gauche (voir figure 1).

FIGURE 1 – Repère associé à une fenêtre

Question 5
Écrivez la fonction `point_dans_rectangle` qui prend un point et un rectangle en paramètres et qui renvoie 1 (vrai) si le point est dans le rectangle (les bords du rectangle sont dans le rectangle) et 0 (faux) sinon.

Question 6

Écrivez la fonction `intersection_rectangles` qui prend en paramètres deux rectangles et qui renvoie le rectangle se trouvant à l'intersection des deux. Nous ferons l'hypothèse que l'intersection des deux rectangles passés en paramètre n'est pas vide. La couleur associée à tous les éléments de ce nouveau rectangle sera le noir (couleur des points, des traits et du fond).

Si les deux rectangles `r1` et `r2` de la figure 2 sont passés en paramètre, la fonction `intersection_rectangles` doit renvoyer le rectangle gris.

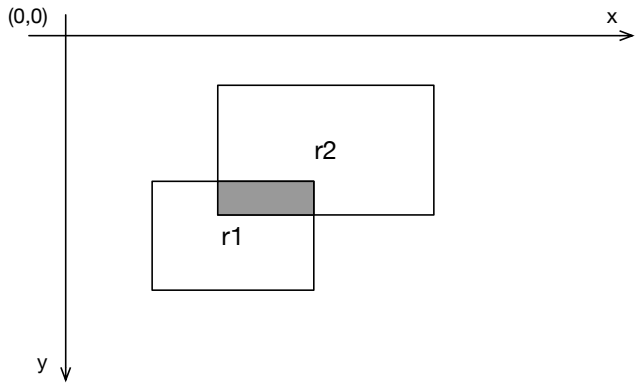

FIGURE 2 – Intersection de deux rectangles

Pour écrire la fonction `intersection_rectangles`, vous supposerez que vous avez à votre disposition la fonction `minimum` (resp. `maximum`) qui prend deux entiers en paramètres et qui renvoie le plus petit (resp. grand) des deux.

Question 7

Donnez l'évolution de la pile d'exécution lors de l'exécution du `programme` suivant.

```
#include <stdio.h>

/* déclaration des types point et rectangle */

/* définitions des fonctions minimum, maximum
   et intersection_rectangles */

int main(){
  rectangle rectangle1;
  rectangle rectangle2;
  rectangle rectangle3;

  /* initialisation rectangle1
  point_bas_gauche = (100,300,"noir")
  point_haut_droite = (300,200,"noir")
  couleur_fond = "bleu"
  couleur_trait = "jaune" */

  /* initialisation restangle2
  point_bas_gauche = (200,250,"noir")
  point_haut_droite = (350,100,"noir")
  couleur_fond = "rouge"
  couleur_trait = "vert" */

  rectangle3=intersection_rectangles(rectangle1,rectangle2);

  return 0;
```

```
}
```

Exercice 27 – Gestion de stock

Nous considérons la gestion du stock d'un commerce. Chaque article du stock est représenté par une référence (un entier), un prix (un flottant) et une quantité (un entier correspondant au nombre d'exemplaires en stock). Le stock est représenté par un tableau d'articles. Nous considérons que la taille du tableau correpond exactement au nombre d'articles référencés.

Question 1
Définissez le type `article` qui permet de définir un article du stock.

Question 2
Soit la déclaration **`#define`** `TAILLE_STOCK 40` qui permet de définir le nombre d'articles référencés (40 dans ce cas). Déclarez le tableau `stock` qui permettra de contenir tout le stock.

Question 3
Écrivez la fonction `augmentationPrix` qui permet d'augmenter le prix d'un article en appliquant le taux d'augmentation passé en paramètre. La fonction ne renvoie rien, elle doit donc avoir comme deuxième paramètre une information permettant d'accéder à l'article dont on veut modifier le prix.

Question 4
Écrivez la fonction `augmentationGenerale` qui permet d'augmenter le prix de tous les articles en appliquant le taux d'augmentation passé en paramètre. Cette fonction doit faire appel à la fonction `augmentationPrix`.

Question 5
Donnez l'évolution de la pile d'exécution lors de l'exécution du `programme` suivant.

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

#define TAILLE_STOCK 2

/* déclaration du type article */

/* définition des fonctions augmentationPrix et
   augmentationGenerale */



int main() {
  article stock[TAILLE_STOCK]={{152,10.5,10},{432,24.3,50}};


  augmentationGenerale(stock,TAILLE_STOCK,10.0);

}
```

Question 6
Écrivez la fonction `validationReferences` qui prend en paramètre un stock et qui renvoie 1 si toutes les références sont différentes et 0 sinon.

Semaine 8 - TD

Objectifs

— Introduction aux listes chaînées
— Représentation de listes d'entiers
— Les différents parcours de liste

Exercices

Dans les exercices de cette semaine, nous allons implémenter en langage C les opérations et fonctions que vous réalisiez en Python sur les listes. Dans un premier temps nous nous intéresserons uniquement aux listes d'entiers.

Les exercices s'appuieront sur la structure de données suivante :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
  int donnee;
  cellule_t *suivant;
};
```

Exercice 28 – Représentation et affichage d'une liste

L'objectif de cet exercice est de manipuler une structure simple de liste, de comprendre sa représentation en mémoire et de voir un premier parcours de liste.

Question 1

Écrivez une fonction `Creer_cellule` qui alloue et renvoie un pointeur vers une `cellule_t` dont le champ `donnee` sera initialisé avec un entier `d` donné en argument et le champ `suivant` à `NULL`.

Soit le programme suivant :

```
void Afficher_liste_int(cellule_t *liste){
/* Affiche les champs donnee des elements de la liste */
   cellule_t *cell = liste;
   while (cell != NULL) {
      printf("%d ", cell->donnee);
      cell = cell->suivant;
   }
   printf("\n");
}

int main() {
   cellule_t *nCell1=NULL, *nCell2=NULL, *nCell3=NULL;

   nCell1 = Creer_cellule(1);
   nCell2 = Creer_cellule(2);
   nCell3 = Creer_cellule(3);
   /* Breakpoint 1 ICI, Dessiner la memoire, tas et pile*/

   nCell1->suivant = nCell2;
   nCell2->suivant = nCell3;
```

```
    /* Breakpoint 2 ICI, Dessiner la memoire, tas et pile*/
    /* Afficher la liste en partant de chaque variable, definir la tete et la queue */
    Afficher_liste_int(nCell1);
    return 0;
}
```

Question 2
Représentez l'état de la mémoire (pile et tas) avant l'exécution de l'instruction **`return`** de la fonction `Creer_cellule` lors de l'appel `Creer_cellule(1)`, puis après l'exécution de l'appel.

Question 3
Représentez l'état de la mémoire (pile et tas) au *breakpoint 1* puis au *breakpoint 2*. Que fait ce programme ?

Question 4
Qu'affiche ce programme ?
Qu'affiche-t-il si l'appel à `Afficher_liste_int(nCell1)` devient `Afficher_liste_int(nCell2)` ?

Question 5
Que se passe-t-il si on ajoute les lignes suivantes à la fin du programme ?

```
nCell1->suivant = nCell3;
Afficher_liste_int(nCell1);
```

Question 6
Que se passe-t-il si on ajoute ensuite les lignes suivantes à la fin du programme :

```
nCell3->suivant = nCell2;
Afficher_liste_int(nCell1);
```

Question 7
Ré-écrivez la fonction `main` telle qu'elle a été donnée en question 2 mais avec une seule variable de type `cellule_t*`, variable que nous nommerons `tete`.

Exercice 29 – Parcours de liste

Question 1
Écrivez une fonction `len` qui prend en paramètre une liste de `cellule_t` et qui renvoie son nombre d'éléments. Ajoutez à la fonction `main`, du corrigé de la denière question de l'exercice précédent, une instruction permettant de tester votre fonction `len`.

Question 2
Écrivez une fonction `existe` qui prend en paramètre une liste de `cellule_t` et un entier `val`. La fonction renvoie `0` (faux) si la valeur `val` n'existe pas dans la liste et `1` sinon.

Question 3
Modifiez la fonction `existe`, pour qu'elle renvoie un pointeur vers le premier élément de valeur `val` dans la liste. La fonction renvoie `NULL` si aucun élément n'a cette valeur. Vous appellerez cette nouvelle fonction `Renvoyer_element_debut`. Donnez des instructions permettant de vérifier que la fonction a bien renvoyé un pointeur sur un élément ayant la bonne valeur.

Question 4
Modifiez la fonction `Renvoyer_element_debut` pour qu'elle renvoie un pointeur vers le **dernier** élément de valeur

`val` dans la liste. La fonction renvoie `NULL` si aucun élément n'a cette valeur. Vous appellerez cette nouvelle fonction `Renvoyer_element_fin`.

Comment faire pour tester que la fonction `Renvoyer_element_debut` renvoie bien le premier élément rencontré et la fonction `Renvoyer_element_fin` le dernier ?

Question 5

Dans le cas d'une liste non vide, quelle information vous permet de savoir qu'un élément est le dernier de la liste ?

Écrivez une fonction `Renvoyer_dernier_element` qui renvoie un pointeur vers le dernier élément de la liste. La fonction renvoie `NULL` si la liste est vide. Complétez la fonction `main` pour tester votre fonction.

Question 6

Écrivez une fonction `Modifier_element(int val, int pos, cellule_t* liste)` qui affecte la valeur `val` à l'élément en position `pos` de la liste donnée en argument. Si la liste comporte moins de `pos` éléments (par analogie avec les tableaux, on suppose que le premier élément correspond à la position `0`), elle n'est pas modifiée. Nous ferons l'hypothèse que `pos` est supérieur ou égal à 0.

Semaine 9 - TD

Objectifs

— Ajout d'un élément dans une liste
— Suppression d'un élément d'une liste
— Parcours simultané de plusieurs listes

Exercices

Les fonctions que nous allons écrire cette semaine modifient la liste sur laquelle elles s'appliquent soit en y ajoutant un élément soit en en supprimant un. Si l'ajout se fait en tête de liste ou si le premier élément de la liste est supprimé, la tête de liste est modifiée, il faut donc récupérer sa nouvelle valeur. Ces fonctions vont donc prendre en paramètre un pointeur sur un élément de liste (le premier élément de la liste avant modification) et retourner un pointeur sur un élément de liste (le nouveau premier élément de la liste modifiée).
Attention, ces fonctions modifient la liste initiale et renvoient le pointeur sur le premier élément de la liste après modification. La liste initiale n'existe plus après appel à l'une de ces fonctions, elle a été modifiée.

Exercice 30 – Insertions

Dans cet exercice, nous allons commencer à construire des listes. Une liste se construit en ajoutant des éléments à une liste existante, éventuellement vide.

Il existe 3 types d'insertion dans les listes : au début (en tête), à la fin (en queue) ou à une position spécifique donnée. Nous allons implémenter chacun des 3 cas.

Nous ferons appel à la fonction `Creer_cellule` qui alloue et renvoie un pointeur vers une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` donné en argument et le champ `suivant` à `NULL`.

Question 1

Écrivez une fonction `Inserer_tete` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en tête de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

Question 2

Écrivez une fonction `Inserer_fin_it` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en queue de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

Question 3

Qu'affiche la fonction `main` suivante ?

```
int main() {
   cellule_t *tete = NULL;
   int i;

   for(i = 0; i < 5; i++) {
      tete = Inserer_tete(i,tete);
   }
   Afficher_liste_int(tete);
   for(i = 5; i < 10; i++) {
      tete = Inserer_fin_it(i,tete);
   }
   Afficher_liste_int(tete);
```

```
    return 0;
}
```

Question 4

Écrivez une fonction `Inserer_en_pos(int d, int pos, cellule_t *liste)` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec l'entier `d` passé en argument. La fonction insère la nouvelle cellule à la position `pos` de `liste` et renvoie un pointeur vers la nouvelle liste ainsi formée. Nous ferons l'hypothèse que `pos` est supérieur ou égal à 0.

Par analogie avec les indices de tableaux, on notera `0` la position du premier élément. Si la valeur `pos` est supérieure au nombre d'éléments dans la liste, l'élément sera ajouté à la fin.

Par exemple, si on insère un élément de valeur 15 en position 5 de la liste `1 2 3 4 5 6 7 8 9 10`, la liste résultante contiendra les valeurs `1 2 3 4 5 15 6 7 8 9 10`.

La Figure 3 donne un état de la mémoire au cours de l'exécution d'un programme. @*ti* représente l'adresse dans le tas d'une zone allouée pour contenir une `cellule_t`).

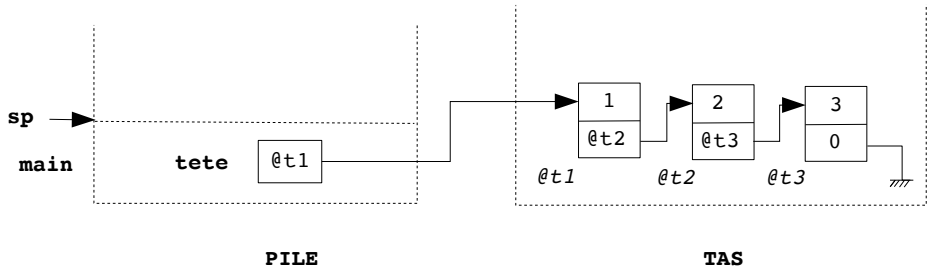

FIGURE 3 – état mémoire à l'appel de `Inserer_en_pos`

Question 5

On suppose qu'à ce moment, on exécute l'instruction `tete = Inserer_en_pos(-1, 2, tete)`. Représentez l'état de la mémoire à l'entrée de la fonction, puis juste après sa terminaison.

Question 6

Ajoutez dans la fonction `main` les instructions permettant de vérifier que votre fonction se comporte correctement dans tous les cas.

Exercice 31 – Suppression

L'objectif de cet exercice est de savoir détruire (désallouer) des éléments de listes ou des listes entières.

Question 1

Écrivez une fonction `cellule_t* Liberer_liste(cellule_t* liste)` qui libère (désalloue) tous les éléments de la liste `liste`. Cette fonction renverra `NULL` pour garantir que la variable de la fonction appelante qui récupère le résultat ne pointe pas sur une zone mémoire désallouée.

On considère l'état mémoire de la Figure 4, obtenu après l'appel de fonction `tete = Liberer_liste(tete)` (`tmp` est la variable locale qui mémorise l'adresse de l'élément à désallouer).

Question 2

Représentez l'état de la mémoire à la fin de la première itération (désallocation du premier élément).

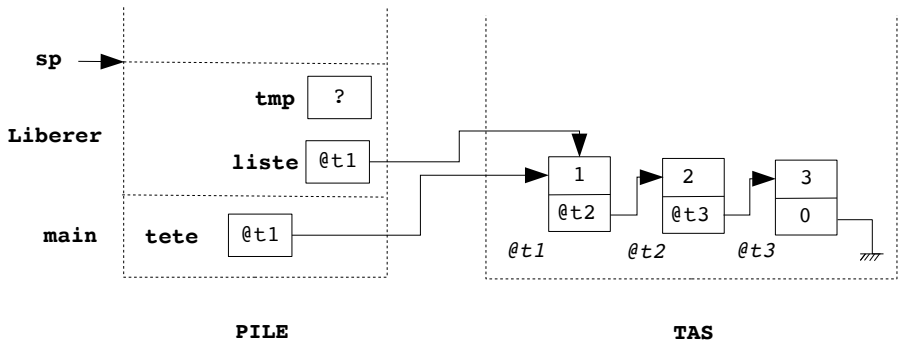

FIGURE 4 – état mémoire à l'entrée de la fonction `Liberer_liste`

Question 3

Écrivez une fonction `cellule_t* Supprimer_en_pos(int pos, cellule_t *liste)` qui supprime (désalloue) l'élément de la liste en position `pos`. Si l'indice est trop grand, la fonction ne modifie pas la liste. Nous faisons l'hypothèse que `pos` est supérieur ou égal à 0.

Exercice 32 – Parcours de plusieurs listes

Dans cet exercice, nous allons allons écrire des fonctions nécessitant de parcourir plusieurs listes simultanément.

Question 1

Écrivez une fonction `listes_identiques` qui prend en paramètre deux listes d'entiers et qui renvoie 1 si les listes sont identiques, 0 sinon. Deux listes son identiques si elles contiennent exactement les mêmes éléments et dans le même ordre.

Question 2

Écrivez une fonction `listes_incluses` qui prend en paramètre deux listes d'entiers et qui renvoie 1 si la première liste est incluse dans le seconde (ou égale), 0 sinon. Nous ferons l'hypothèse que les listes sont triées en ordre croissant.

Semaine 10 - TD

Objectifs

— Récursivité
— Extraction d'une sous-liste

Exercices

Exercice 33 – Manipulations récursives sur les listes

Une liste est une structure intrinsèquement récursive puisqu'elle est composée d'un élément suivi d'une liste (ou l'inverse...). Beaucoup de fonctions s'écrivent donc assez naturellement sous forme récursive. Nous allons appliquer cette approche à quelques unes des opérations que nous avons déjà programmées.

Question 1
Écrivez une fonction **récursive** `cellule_t* Renvoyer_element_debut_rec(int val, cellule_t* liste)` qui renvoie l'adresse de la première occurrence de `val` dans la liste. La fonction renvoie `NULL` si la valeur recherchée n'est pas dans la liste.

Question 2
Écrivez une fonction **récursive** `cellule_t *Inserer_fin_rec(int d, cellule_t *liste)` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en fin de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

On considère la fonction `main` suivante, l'état de la mémoire avant l'appel à `Inserer_fin_rec` est donné en Figure 5.

```
int main() {
   cellule_t *tete = NULL;

   tete = Creer_cellule(1);
   tete->suivant = Creer_cellule(2);

   tete = Inserer_fin_rec(3, tete);
   Afficher_liste_int(tete);
   return 0;
}
```

Question 3
Déroulez l'exécution de l'appel à `Inserer_fin_rec(3, tete)`

Question 4
Écrivez une fonction **récursive** `Supprimer_en_pos_rec(int pos, cellule_t *liste)` qui supprime (désalloue) l'élément de la liste en position `pos`. Si l'indice est trop grand, aucune désallocation n'est effectuée.

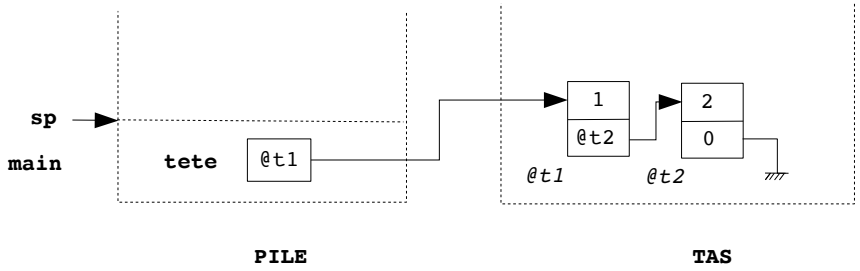

FIGURE 5 – état mémoire avant l'appel à `Inserer_fin_rec`

Exercice 34 – Extraction d'une sous liste

Nous allons extraire une sous-liste d'une liste de deux façons :
- en construisant une nouvelle liste contenant les éléments que nous souhaitons garder
- en supprimant de la liste les éléments que l'on ne souhaite pas garder.

Question 1

Écrivez une fonction **itérative** `Creer_liste_positifs` qui prend en paramètre une liste d'entiers, qui construit la liste contenant les éléments strictements positifs de la liste initiale et qui renvoie la tête de cette nouvelle liste (qui sera égale à `NULL` si aucun entier de la liste n'est strictement positif). La liste passée en paramètre n'est pas modifiée. Vous appellerez la fonction `Inserer_tete` pour ajouter un élément en tête de liste.

Question 2

Donnez une version **récursive** de la fonction `Creer_liste_positifs`. Vous appellerez la fonction `Inserer_tete` pour ajouter un élément en tête de liste.

Question 3

Quelle différence sur le résultat obtenu constatez-vous entre les deux fonctions précédentes ?

Question 4

Écrivez une fonction `Garder_positifs` qui prend en paramètre une liste d'entiers et qui supprime de cette liste tous les éléments qui ne sont pas strictement positfs. La fonction renvoie la nouvelle tête de liste.

Semaine 11 - TD

Objectifs

— Listes
— Révisions

Exercices

Exercice 35 – Parcours de listes de types différents

Dans cet exercice nous allons construite un multi-ensemble trié par ordre croissant des valeurs à partir d'une liste d'entiers triée elle aussi par ordre croissant.

Les éléments d'une liste d'entiers sont du type suivant :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
  int donnee;
  cellule_t *suivant;
};
```

Ceux d'un multi-ensemble sont du type suivant :

```
typedef struct _element_t element_t;
struct _element_t{
  int valeur;
  int frequence;
  element_t *suivant;
};
```

Question 1

Écrivez la fonction `Creer_multi_ensemble_liste` qui prend en paramètre une liste d'entiers triés en ordre croissant(un même entier peut apparaître plusieurs fois) et crée le multi-ensemble correspondant triée par ordre croissant des valeurs. La fonction renvoie un pointeur sur le premier élément du multi-ensemble.