TP 9 : Listes de cartes

Notions : Type abstrait, Listes, Structures, Allocation dynamique, pointeurs

1 Structures de données

L'objectif de cette séance est de définir les fonctions de base sur les listes de cartes que nous réutiliserons par la suite.

Au lieu d'avoir une liste de réels comme en cours, nous allons avoir des listes de cartes. Une carte sera définie par une structure de type `element_t` ou `CARTE`.

Une carte est définie par son rang (1,2,3,4... valet, dame, roi), sa couleur (cœur, carreau, trèfle, pique) et sa disposition (visible ou cachée). Une carte est donc définie par les champs :

- rang : la valeur (as, 2,..roi) : un entier
- couleur : trèfle (0), carreau (1), cœur (2), pique (3) : c'est un entier avec le codage indiqué entre parenthèse.
- visible : face visible (1) ou cachée (0) : un entier

Pour la couleur, on peut utiliser un type énuméré qui permet d'utiliser ensuite les constantes symboliques `TREFLE`, `CARREAU`, `CŒUR`, `PIQUE` au lieu des valeurs 0,1,2,3 comme valeurs respectives. Il est défini par : `typedef enum { TREFLE, CARREAU, CŒUR, PIQUE} COULEUR;` .

Le fichier d'entête `element.h` contient la définition des types `COULEUR` et `element_t`. Le type `CARTE` est identique au type `element_t`. Ce fichier contient aussi les prototypes des fonctions qui sont dépendantes de la nature des éléments de la liste, ici des cartes, et qui sont nécessaires au fonctionnement de nos listes. La partie contenant la définition des types est la suivante :

```
#ifndef _ELEMENT
#define _ELEMENT

typedef enum { TREFLE,CARREAU,COEUR,PIQUE} COULEUR;
typedef struct { int rang; char visible; COULEUR couleur; } CARTE;
typedef CARTE element_t;

// Prototypes des fonctions.

#endif
```

Le fichier d'entête `list.h` contient les types définissant un maillon et une liste. Un maillon contient donc une partie `val` qui est une carte, et une partie `next` qui est un pointeur sur le maillon suivant. Ce fichier contient aussi les prototypes des fonctions sur les listes.

Le type `link_t` représente un maillon, le type `list_t` représente un pointeur sur un maillon, donc une liste.

La partie contenant la définition des types est la suivante :

```
// Fichier list.h
#ifndef _LIST_H
#define _LIST_H
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
```

```
#include "element.h"

// 1. Définition des types maillon (link_t) et liste (list_t)
typedef struct _link {
  element_t val; /* un element de la liste*/
  struct _link *next; /* l'adresse du maillon suivant */
} link_t, *list_t;

//2. Prototype des fonctions

#endif
```

2 Fichiers à télécharger

Télécharger le fichier `tp9.zip`. Ce fichier contient les fichiers suivants :

`element.h`. Ce fichier contient la définition des types `COULEUR`, `element_t` et `CARTE` ainsi que les prototypes des fonctions associées :

```
void element_print (CARTE e); // Affichage d'une carte
void element_init (CARTE* e); // Initialisation d'une carte à 0,0,0
CARTE element_empty();  // Creation d'une carte vide (0,0,0)
int element_is_empty(CARTE e); // Test si la carte est la carte vide
int element_equal(CARTE*, CARTE*); // Test de l'egalite de 2 cartes
#endif
```

`element.c`. Ce fichier contient le code source des fonctions d'initialisation, d'affichage d'une carte, de création d'une carte vide et de test de l'égalité entre 2 cartes.

`list.h`. Ce fichier contient les définitions du type `list_t` et les prototypes des fonctions que nous fournissons.

`list.c`. Ce fichier contient le code source des fonctions que nous fournissons, ainsi que l'entête des fonctions que nous vous demandons d'écrire. Les fonctions déjà codées sont les suivantes :

- `list_t list_new(void);` : création d'une liste vide
- `int list_is_empty(list_t);` : teste si une liste est vide ou pas
- `void list_print(list_t);` : affiche le contenu d'une liste. Suppose que la fonction `element_print` existe.
- `list_t list_add_first(element_t, list_t l);` : Ajoute un element en tete de la liste et retourne la nouvelle liste.
- `list_t list_del_first(list_t) ;` Supprime le premier element de la liste et retourne la nouvelle liste.
- `list_t list_find(element_t, list_t);` qui recherche si une carte est présente dans la liste. Elle retourne le pointeur sur le maillon contenant la carte si cette dernière est présente, `NULL` sinon.
- `int list_length(list_t);` qui compte et retourne le nombre de cartes de la liste

3 Fonctions simples

1. Ecrire, dans un fichier `mytest.c`, un programme principal (fonction `main()`) qui crée une liste, ajoute l'as de coeur, puis le 3 de pique à la liste et affiche ensuite la liste. La compilation se fera par la commande `clang mytest.c list.c element.c -o mytest` et l'execution par `./mytest`
2. Compléter le fichier `list.c` avec les fonctions suivantes ;

— `list_t list_count(element_t, list_t);` qui compte et retourne le nombre de fois qu'une carte est présente dans la liste.

— `list_t list_add_last(element_t, list_t);` qui ajoute une carte en fin de liste. Elle retourne la nouvelle liste.

— `list_t list_delete(list_t);` qui libère tous les chainons de la liste. Vous pouvez utiliser la fonction `list_del_first`. Elle retourne une liste vide.

3. Compiler le fichier contenant les tests simples `test_list0.c` avec la commande `make test_list0`, exécuter ce programme avec la commande `./test_list0` et vérifier la cohérence du résultat avec le contenu du fichier `test_list0.c`.

   Le programme doit s'executer sans erreurs (`segmentation fault`) et le résultat affiché doit être cohérent avec la sémantique du programme. A vous de vérifier ce que ce programme doit faire.

   L'affichage normal de ce programme est donné dans le fichier `resultatTestListe0.txt`. Vous pouvez comparer votre sortie écran à ce fichier à l'aide de la commande Linux `diff` qui indique si 2 fichiers sont différents et à quel endroit, par la suite des 2 commandes suivante :

   (a) `./test_list0 > mesResultats.txt`

   (b) `diff resultatTestListe0.txt mesResultats.txt`

Le fichier `test_list0.c` est donné ci dessous :

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main() { CARTE c; list_t l1=NULL,l2=NULL, p=NULL;
    l1=list_new(); l2=list_new();
    printf("-----------------------------------------------\n");
    printf("test de l'ajout en tete d'une carte et de l'affichage\n");
    printf("-----------------------------------------------\n");
    printf("Nombre de cartes : %d\n",list_length(l1));
    printf("-------------------------------\n");
    printf("Ajout en tete de l'as de coeur\n");
    c.rang=1; c.couleur=COEUR; l1=list_add_first(c,l1);
    printf("Liste l1 : ");    list_print(l1);
    printf("Nombre de cartes : %d\n",list_length(l1));
    printf("-------------------------------\n");
    printf("Ajout en tete du roi de coeur\n");
    c.rang=13;    c.couleur=COEUR;  l1=list_add_first(c,l1);
    printf("Liste l1 : ");    list_print(l1);
    printf("Nombre de cartes : %d\n",list_length(l1));
    printf("-------------------------------\n");
    printf("Ajout en tete de l'as de pique\n");
    c.rang=1;    c.couleur=PIQUE;  l1=list_add_first(c,l1);
    printf("Liste l1 : ");    list_print(l1);
    printf("Nombre de cartes : %d\n",list_length(l1));
    printf("-------------------------------\n");
    printf("Ajout en tete du roi de pique\n");
    c.rang=13;    c.couleur=PIQUE; l1=list_add_first(c,l1);
    printf("Liste l1 : ");    list_print(l1);
    printf("Nombre de cartes : %d\n",list_length(l1));
    printf("-------------------------------\n");
    printf("Ajout en tete de l'as de pique\n");
    c.rang=1;    c.couleur=PIQUE; l1=list_add_first(c,l1);
    printf("Liste l1 : ");    list_print(l1);
    printf("Nombre de cartes : %d\n",list_length(l1));

    printf("\n-----------------------------------------------\n");
    printf("Test du nombre de carte donnee dans une liste \n");
    printf("-----------------------------------------------\n");
    printf("Test du nombre d'as de pique \n");
    printf("-------------------------------\n");
    c.rang=1;    c.couleur=PIQUE;
    int nb;
    nb=list_count(c,l1);
    printf("Nombre de "); element_print(c);
    printf("\ndans la liste : "); list_print(l1);
```

```c
printf("La carte est presente %d fois\n",nb);

printf("-------------------------------\n");
printf("Test du nombre de 10 de pique carte\n");
c.rang=10;    c.couleur=PIQUE;
nb=list_count(c,l1);
printf("Nombre de "); element_print(c);
printf("\ndans la liste : "); list_print(l1);
printf("La carte est presente %d fois\n",nb);

l2=list_new();
printf("\n----------------------------------------------\n");
printf("Nouvelle Liste : test de l'ajout en queue d'une carte \n");
printf("----------------------------------------------\n");
printf("Ajout en queue de l'as de coeur\n");
c.rang=1;    c.couleur=COEUR;   l2=list_add_last(c,l2);
printf("Liste l2 : ");    list_print(l2);
printf("-------------------------------\n");
printf("Ajout en queue du roi de coeur\n");
c.rang=13;    c.couleur=COEUR;   l2=list_add_last(c,l2);
printf("Liste l2 : ");    list_print(l2);
printf("-------------------------------\n");
printf("Ajout en queue de l'as de pique\n");
c.rang=1;    c.couleur=PIQUE;   l2=list_add_last(c,l2);
printf("Liste l2 : ");    list_print(l2);
printf("-------------------------------\n");
printf("Ajout en queue du roi de pique\n");
c.rang=13;    c.couleur=PIQUE;   l2=list_add_last(c,l2);
printf("Liste l2 : ");    list_print(l2);


printf("\n----------------------------------------------\n");
printf("test de la suppression en tete d'une carte \n");
printf("----------------------------------------------\n");
printf("Suppression en tete\n");
if (!list_is_empty(l1)) l1=list_del_first(l1);
printf("Liste l1 : ");    list_print(l1);
printf("-------------------------------\n");
printf("Suppression en tete\n");
if (!list_is_empty(l1)) l1=list_del_first(l1);
printf("Liste l1 : ");    list_print(l1);
printf("-------------------------------\n");
printf("Suppression en tete\n");
if (!list_is_empty(l1)) l1=list_del_first(l1);
printf("Liste l1 : ");    list_print(l1);
printf("-------------------------------\n");
printf("Suppression en tete\n");
if (!list_is_empty(l1)) l1=list_del_first(l1);
printf("Liste l1 : ");    list_print(l1);
printf("-------------------------------\n");
printf("Suppression en tete\n");
if (!list_is_empty(l1)) l1=list_del_first(l1);
printf("Liste l1 : ");    list_print(l1);

printf("\n----------------------------------------------\n");
printf("Liberation de toutes les listes\n");
printf("----------------------------------------------\n");
l1=list_delete(l1);     l2=list_delete(l2);

printf("\n----------------------------------------------\n");
printf("Test de la presence d'une carte dans une liste vide\n");
printf("----------------------------------------------\n");
c.rang=10;    c.couleur=PIQUE;
p=list_find(c,l1);
printf("Recherche de "); element_print(c);
printf("\ndans la liste l1 : "); list_print(l1);
if (p==NULL) printf("La carte n'est pas presente\n");
else {printf("La carte est presente, c'est une erreur\n"); exit(EXIT_FAILURE);}

exit(EXIT_SUCCESS);
```

```
}
```

4 Fonctions de copie

1. Ajouter les fonctions suivantes dans le fichier `list.c` et leur prototype dans `list.h` :
   (a) `list_t list_concat(list_t l1, list_t l2);` : ajoute la liste l2 à la suite de la liste l1 **sans** recréer de nouveaux éléments. La fonction retourne la nouvelle liste ainsi formée.
   (b) Que deviennent alors les listes l1 et l2 ?
   (c) `list_t list_copy(list_t l);` : crée une nouvelle liste qui est une copie de la liste `l`. Tous les éléments de la liste `l` sont dupliqués.
2. Compiler le fichier `test_list1.c` contenant un programme principal permettant de tester ces différentes fonctions avec la commande `make test_list1`
3. L'affichage normal de ce programme est donné dans le fichier `resultatTestListe1.txt`. Vous pouvez comparer votre sortie écran à ce fichier à l'aide de la commande Linux `diff` comme précédemment.
4. Expliquer les résultats obtenus pour les copies et concaténation de liste obtenus ci dessus

Le fichier `test_list1.c` est donné ci-dessous.

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main() { CARTE c; list_t l1,l2,l3,l4,l5,l6,l7;
  l1=list_new(); l2=list_new();
  l3=list_new(); l4=list_new();
  l5=list_new(); l6=list_new();
  l7=list_new();

  printf("\n-----------------------------------------------\n");
  printf("Test des copies et concatenations\n");
  printf("-----------------------------------------------\n");

  c.rang=11; c.couleur= COEUR; l1=list_add_first(c,l1);
  c.rang=8; c.couleur= COEUR; l1=list_add_first(c,l1);
  c.rang=7; c.couleur= CARREAU; l1=list_add_first(c,l1);
  c.rang=12; c.couleur= PIQUE; l1=list_add_first(c,l1);
  c.rang=9; c.couleur= PIQUE; l1=list_add_first(c,l1);
  printf("Liste l1: "); list_print(l1);
  printf("-------------------------------\n");

  c.rang=10; c.couleur= CARREAU; l2=list_add_first(c,l2);
  c.rang=1; c.couleur= TREFLE; l2=list_add_first(c,l2);
  c.rang=3; c.couleur= CARREAU; l2=list_add_first(c,l2);
  c.rang=5; c.couleur= PIQUE; l2=list_add_first(c,l2);
  printf("Liste l2: "); list_print(l2);
  printf("-------------------------------\n");

  l4=list_copy(l1);
  printf("\n-----------------------------------------------\n");
  printf("Copie de l1 dans l4\n");
  printf("l1 et l4 ont des maillons differents\n");
  printf("-----------------------------------------------\n");
  printf("Liste l1: "); list_print(l1);
  printf("Liste l4: "); list_print(l4);

  l5=l1;
  printf("\nAffectation de la liste l1 dans l5\n");
  printf("l1 et l5 partagent les memes maillons\n");
  printf("-------------------------------\n");
  printf("Liste l5: "); list_print(l5);
```

```
  printf("Liste l1: "); list_print(l1);

  printf("\n-----------------------------------------------\n");
  printf("Concatenation de l2 a la suite de l1 dans l3 \n");
  printf("l3 partage ses maillons avec l1 et l2\n");
  printf("-----------------------------------------------\n");
  l3=list_concat(l1,l2);
  printf("Liste l1: "); list_print(l1);
  printf("Liste l2: "); list_print(l2);
  printf("Liste l3: "); list_print(l3);
  printf("Liste l4: "); list_print(l4);
  printf("Liste l5: "); list_print(l5);

  printf("\n-----------------------------------------------\n");
  printf("Concatenation d'une liste vide avec une autre liste \n");
  printf("-----------------------------------------------\n");
  l7=list_concat(l6,l2);
  printf("Liste l6: "); list_print(l6);
  printf("Liste l2: "); list_print(l2);
  printf("Liste l7: "); list_print(l7);

  printf("\n-----------------------------------------------\n");
  printf("Liberation de toutes les listes\n");
  printf("-----------------------------------------------\n");
  printf("Liberation de l1\n");
  l1=list_delete(l1);
  printf("Ne pas liberer l2 car l1 est maintenant chainee avec l2 : l3=list_concat(l1,l2)\n");
  printf("Ne pas liberer l3 car elle est deja liberee a travers l1 \n");
  printf("Liberation de l4\n");
  l4=list_delete(l4);
  printf("Ne pas liberer l5 car elle est deja liberee a travers l1\n");
  printf("Ne pas liberer l6 car elle est vide\n");
  printf("Ne pas liberer l7 car elle est deja liberee a travers l2\n");
  exit(EXIT_SUCCESS);
}
```

5 Fonction de Suppression

1. Ajouter la fonction suivante dans le fichier `list.c` et son prototype dans le fichier `list.h` : `list_t list_remove_n(int n, list_t l);` : supprime la carte en n^{ieme} position dans la liste l et retourne la nouvelle liste. Attention : il faut être sur le maillon $n-1$ pour supprimer le maillon n et chainer le maillon $n-1$ avec le maillon $n-1$. Attention : Les positions démarrent à 1.
2. Compiler le fichier `test_list2.c` contenant un programme principal permettant de tester ces différentes fonctions. Les résultats seront comparés au fichier `resultatTestListe1.txt`.

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main() { CARTE c; list_t l1; int n;
  l1=list_new();
  printf("-----------------------------------------------\n");
  printf("Test de la suppression en position\n");
  printf("-----------------------------------------------\n");
  c.rang=11; c.couleur= COEUR; l1=list_add_first(c,l1);
  c.rang=8; c.couleur= COEUR; l1=list_add_first(c,l1);
  c.rang=7; c.couleur= CARREAU; l1=list_add_first(c,l1);
  c.rang=12; c.couleur= PIQUE; l1=list_add_first(c,l1);
  c.rang=9; c.couleur= PIQUE; l1=list_add_first(c,l1);
  c.rang=10; c.couleur= CARREAU; l1=list_add_first(c,l1);
  c.rang=1; c.couleur= TREFLE; l1=list_add_first(c,l1);

  printf("Liste l1: "); list_print(l1);
  n=4;
```

```
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=4;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=-1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=10;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  n=1;
  printf("Suppression de la position %d\n",n);
  printf("------------------------------\n");
  l1 = list_remove_n(n,l1);
  printf("Liste l1: "); list_print(l1);

  printf("-----------------------------------------------\n");
  printf("Liberation des listes\n");
  printf("-----------------------------------------------\n");
  l1=list_delete(l1);
}
```

6 Facultatif : liste triée à l'insertion

1. Ajouter la fonction suivante dans le fichier `list.c` et son prototype dans le fichier `list.h` :
   `list_t list_add_sort(element_t e, list_t l);` : ajoute une carte en respectant l'ordre défini par la valeur de la carte : l'as est inférieur au 2, qui est lui-même inférieur au 3, etc...

2. Compiler le fichier `test_list3.c` contenant un programme principal interactif permettant de tester ces différentes fonctions. Ajouter les tests pour les nouvelles fonctions. Compiler et tester le programme en ajoutant, affichant et supprimant des cartes. Essayer de prévoir les différents cas particuliers : ajout en tete dans une liste vide, ajout en queue dans une liste vide, suppression en tete d'une liste vide...

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main() { CARTE c; list_t l1;
  char rep; int n;
  l1=list_new();
  do {
    printf("quitter(0); Ajouter en tete(1); Ajouter en queu(2); Supprimer en tete(3);
   visualiser_liste(4); Supprimer en n (5)"); fflush(stdout);
    rep=getchar();
    switch(rep) {
      case '1': printf("Valeur(1..13) et couleur (0..3) \n"); scanf("%d %d",&(c.rang),&(c.couleur));
                if (c.rang>=1 && c.rang<=13 && c.couleur>=TREFLE && c.couleur<=PIQUE) l1=
   list_add_first(c,l1);
    else printf("Erreur de saisie %d %d\n",c.rang,c.couleur);
                break;
      case '2': printf("Valeur(1..13) et couleur (0..3) \n"); scanf("%d %d",&(c.rang),&(c.couleur));
                if (c.rang>=1 && c.rang<=13 && c.couleur>=TREFLE && c.couleur<=PIQUE) l1=
   list_add_last(c,l1);
    else printf("Erreur de saisie %d %d\n",c.rang,c.couleur);
                break;
      case '3': l1=list_del_first(l1); break;
      case '4': list_print(l1); break;
      case '5': printf("Quelle est la position a supprimer ? \n");
            scanf("%d",&n);
            if (n>=0)
                l1=list_remove_n(n,l1);
            break;
    }
    getchar();
  } while (rep !='0');
  printf("Liberation des listes\n");
  l1=list_delete(l1);
}
```