S3:E3 — Listes d'entiers

LICENCE INFORMATIQUE : PROGRAMMATION NÉCESSAIRE

Stéfane

1 Introduction

Lors de cet épisode vous allez construire la bibliothèque des listes *simplement chaînées* d'entiers.

Commencez par récupérer le fichier `intlist.c` sur *Arche*.

Ce fichier contient quatre grandes parties :

- Les directives de compilation,
- Les déclarations de types abstraits (TA) et des fonctions publiques,
- L'algorithme, c-àd. la fonction `main()` en C,
- Les définitions des fonctions publiques.

1.1 Les directives de compilation

```c
/**
 * @note Compiler avec
 *    gcc -Wall -std=c11 -o intlist intlist.c
 *
 * @brief Conception d'un type pour les listes d'entiers
 *        et des fonctions permettant de les manipuler
 *
 * @todo Complétez ce fichier puis, séparez les éléments afin qu'apparaissent :
 *  + Un fichier d'entêtes (.h) de la bibliothèque liste d'entiers,
 *  + Un fichier de définitions (.c) correspondant aux déclarations du fichier d'entêtes
 *  + Un fichier main.c ne contenant que la fonction principale.
 *
 * @todo Construisez le Makefile correspondant
 */
#include <stdlib.h> // librairie standard
#include <stdio.h> // librairie input/output
#include <stdbool.h> // librairie du type booléen
#include <assert.h> // librairie d'assertions
```

1.2 Les déclarations des TA et des fonctions publiques

```c
/** @brief Le type d'un élément de liste:
 * + x - un entier,
 * + suc - un pointeur sur son successeur (ou NULL s'il n'y en a pas)
 */
struct lst_elm_t {
  int x;
  struct lst_elm_t * suc;
};
/***
 * DÉCLARATION DES FONCTIONS PUBLIQUES
 ***/
/** @brief Création d'un nouvel élément de liste d'entiers */
struct lst_elm_t * new_lst_elm(int value);
/** @brief Suppression d'un élément de liste d'entiers */
void del_lst_elm_t(struct lst_elm_t ** ptrE);
/** @ brief Accéder au champ x de l'élément de liste d'entiers */
int getX ( struct lst_elm_t * E );
/** @ brief Accéder au champ suc de l'élément de liste d'entiers */
struct lst_elm_t * getSuc ( struct lst_elm_t * E );
/** @brief Modifier la valeur du champ x de l'élément de liste d'entiers */
void setX (struct lst_elm_t * E, int v );
/** @brief Modifier la valeur du champ suc de l'élément de liste d'entiers */
void setSuc ( struct lst_elm_t * E, struct lst_elm_t * S );

/** @brief Le type d'une liste :
  * + head - le premier élément de la liste
  * + tail - le dernier élément de la liste
  * + numelm - le nombre d'élément dans la liste
*/
struct lst_t {
  struct lst_elm_t * head;
  struct lst_elm_t * tail;
  int numelm;
};
/***
 * DÉCLARATION DES FONCTIONS PUBLIQUES
 ***/
/** @brief Construire une liste vide */
struct lst_t * new_lst();
/** @brief Libèrer la mémoire occupée par la liste */
void del_lst(struct lst_t ** ptrL );
/** @brief Vérifier si la liste L est vide ou pas */
bool empty_lst(const struct lst_t * L);
/** @brief Ajouter en tête de la liste L la valeur v */
void cons(struct lst_t * L, int v);
/** @brief Visualiser les éléments de la liste L */
void print_lst(struct lst_t * L );
```

1.3 La fonction principale `main`

```c
/*** ALGORITHME (FONCTION PRINCIPALE) ***/
int main() {
  int v;
  struct lst_t * L = new_lst();
  scanf( "%d", &v );
  while( v > 0 ) {
    cons(L,v);
    scanf("%d",&v);
  }
  print_lst(L);
  del_lst(L);
  return EXIT_SUCCESS;
}
```

1.4 Les définitions des fonctions

```c
/***
  * DÉFINITIONS DES FONCTIONS
  ***/
/** @todo Définissez les fonctions publiques :
 * + new_lst_elm
 * + del_lst_elm
 * + getX
 * + getSuc
 * + setX
 * + setSuc
 * des éléments de liste d'entiers.
 **/
struct lst_t * new_lst() {
  /**
   * @note : calloc fonctionne de manière identique à malloc
   * et de surcroît met à NULL(0) tous les octets alloués
   **/
  struct lst_t * L = (struct lst_t *)calloc(1,sizeof(struct lst_t));
  assert(L);
  return L;
}
void del_lst(struct lst_t ** ptrL ) {
  /** @todo */
}
bool empty_lst(const struct lst_t * L ) {
  assert(L);// L doit exister !
  return L->numelm == 0;
}
void cons(struct lst_t * L, int v) {
  /** @todo */
}
void print_lst(struct lst_t * L ) {
  printf( "[ " );
  for( struct lst_elm_t * E = L->head; E; E = E->suc) {
    printf( "%d ", E->x );
  }
  printf( "]\n\n" );
}
```

2 Travaux dirigés

2.1 Complétez le code

Vous allez compléter les définitions des fonctions et vérifier leur bon fonctionnement grâce à la fonction principale `main`.

2.2 Séparez les entêtes des définitions

Dès que l'ensemble des fonctions est correct, il ne vous reste plus qu'à découper le code :

- En fichiers de déclarations (`*.h`) à ranger dans un dossier `include` :
  `lst.h` et `lst_elm.h` ;
- En fichiers de définitions (`*.c`) à ranger dans le dossier `src` :
  `lst.c`, `lst_elm.c` et `main.c`.

Ensuite vous récupérez le `Makefile` de l'épisode S3:E2 que vous modifiez en conséquence.

2.3 Ajoutez des fonctions privée et publique

Complétez les TA précédents avec les fonctions `insert_after` et `insert_ordered`.

2.3.1 La fonction `insert_after`

La fonction `insert_after(struct lst_t * L, const int value, struct lst_elm_t * place)` prend trois arguments : La liste `L` est modifiée par l'insertion de la valeur entière `value` *après* l'emplacement désigné par `place`.

Hypothèse.
Par convention:

- *Soit `place` est `NULL` et alors l'insertion se fait en tête de la liste `L`,*
- *Soit `place` désigne* ***obligatoirement*** *un élément de la liste `L`.*

Définition 2.1 (Fonction privée).
Cette fonction est *privée* c-à-d. qu'elle n'est accessible que par les fonctions publiques du TA `lst_t`. Pour cette raison sa *déclaration* est faite en tête du fichier `lst.c` (et non dans le fichier `lst.h`). Et sa *définition* est faite **après** les définitions de *toutes* les fonctions publiques du TA `lst_t`.

2.3.2 La fonction `insert_ordered`

La fonction `insert_ordered(struct lst_t * L, const int value)` insère dans l'ordre *croissant* la valeur entière `value` dans la liste `L`.

Vous rencontrerez plusieurs situations :

Soit la liste `L` est vide.

L'insertion revient à faire une insertion en tête,

Soit `v` est plus petite que la première valeur entière de `L`.

L'insertion revient également à faire une insertion en tête,

Soit `value` est plus grande que la dernière valeur de `L`.

L'insertion revient à faire une insertion en queue,

Dans les autres cas.

La fonction `insert_ordered` recherche la *place après* laquelle `value` doit être insérée.

Une fois la place trouvée, la fonction appelle `insert_after` avec les bons paramètres.