S3:E7 – Listes doublement chaînées

LICENCE INFORMATIQUE : PROGRAMMATION NÉCESSAIRE

Stéfane

1 Introduction

Lors de ce TP on va s'intéresser aux listes doublement chaînées *générique* et *homogène* pour des *formulaires*. Chaque formulaire contient :

- un nom de produit, [1]
- sa quantité et
- son prix unitaire hors taxe.

```c
/**
Form of manufactured products modeled with:
- A product name (char *)
- A stock (int)
- A price before tax (double)
*/
#define len_max 20
struct form {
  char product[len_max];
  int stock;
  double pbt;
};

/** Delete a form */
void del_form(struct form * F);

/** Read a form from FILE */
struct form * read_form(FILE * fd, enum mode_t mode);

/** Write a form to FILE */
void write_form(struct form * F, enum mode_t mode, FILE * fd);

/** Get the product name of the form F */
char * get_product(struct form * F);

/** Get the stock of the form F */
int get_stock(struct form * F);

/** Get the price before tax of the form F */
double get_price(struct form * F);

/** Display a form on stdout stream */
void view_form(struct form * F);

/**  Is F1's product name less than or equal to F2's product name ? */
bool gt_form(struct form * F1, struct form * F2);
```

1. On utilise une chaîne de caractères *statique* afin de faciliter l'enregistrement des formulaires dans un fichier *binaire*.

La fonction `read_form(FILE * fd, enum mode_t mode)` remplit un formulaire depuis les données saisies dans le fichier désigné par `fd`.

Ce fichier est soit en mode texte soit en binaire.

D'où le second argument qui prend une des deux valeurs définies dans le type *énuméré* `num mode_t { TEXT, BINARY };` qui est écrit dans le fichier `db.h`

```c
struct form * read_form(FILE * fd, enum mode_t mode) {
  // fd is already open and mode is eq to TEXT or BINARY
  assert(fd != NULL);

  // create a form
  struct form * F = (struct form *) calloc(1, sizeof(struct form));
  assert(F != NULL);

  if(mode == TEXT) { // either fd is a text file
    fscanf(fd, " %s", F->product);
    F->product[len_max-1]='\0';
    fscanf(fd, " %d", &(F->stock));
    fscanf(fd, " %lf", &(F->pbt));
  } else { // BINARY, or fd is a binary file
    fread(F, sizeof(struct form), 1, fd);
  }
  return F;
}
```

Remarque.

La fonction qui appelle `read_form(fd, mode);` se charge d'ouvrir le fichier et de le fermer.

La fonction `void write_form(struct form * F, enum mode_t mode, FILE * fd)` fonctionne sur le même principe, à savoir que le fichier est ouvert et fermé par la fonction appelante (en mode texte ou en mode binaire). Elle *se contente* d'écrire un formulaire dans le fichier `fd`.

2 Listes doublement chaînées

Les listes doublement chaînées se présentent comme les listes simplement chaînées auxquelles on ajoute à chaque élément de liste un *pointeur* vers l'élément précédent.

Leur intérêt est une plus grande facilité de manipulation : insertion, suppression d'éléments. Les fonctions associées à ces deux TA sont les mêmes.

```c
/** Abstract type modeling a list element containing
- a form
- 2 pointers to its predecessor  and successor
*/
struct lst_elm_t {
  void * data;
  struct lst_elm_t * suc, * pred;
```

```
};

struct lst_elm_t * new_elmlist(void * data );

void del_elmlist(struct lst_elm_t * E, void (*ptrf) ());

struct lst_elm_t * get_suc(struct lst_elm_t * E);

struct lst_elm_t * get_pred(struct lst_elm_t * E);

void * get_data(struct lst_elm_t * E);

void set_suc(struct lst_elm_t * E, struct lst_elm_t * S);

void set_pred(struct lst_elm_t * E, struct lst_elm_t * P);

void set_data(struct lst_elm_t * E, void * data);

void view_elmlist(struct lst_elm_t * E, void (*ptrf)());
```

```
/** Abstract type for double-linked list modeled by
- 2 pointers pointing to the head and the tail of the list, resp.
- the number of element the list contains
*/
struct lst_t {
  struct lst_elm_t * head, * tail;
  int numelm;
};

/****************
  Constructors & co
  ****************/
struct lst_t * newLst();

void freeLst(struct lst_t * L, void (*ptrf) ());

bool emptyLst(struct lst_t * L);

/*
  Accessors & modifiers
*/
struct lst_elm_t * get_head(struct lst_t * L);

struct lst_elm_t * get_tail(struct lst_t * L);

/** Add on head */
void cons(struct lst_t * L, void * data);

/** Add on tail */
void queue(struct lst_t * L, void * data);

/** Insert data at place pointed by ptrf */
void ordered_insert(struct lst_t * L, void * data, bool (*ptrf)());

/** Display list on stdout stream */
void printLst(struct list * L, void (*ptrf)());
```

3 Travail personnel

La fonction principale effectue dans cet ordre :

1. Récupérer les formulaires stockés dans un fichier texte (`data/db.txt`) pour les ranger dans une liste,
2. Afficher la liste et écrire les formulaires cette liste dans un fichier *binaire*,
3. Supprimer la liste,
4. Créer à nouveau la liste en lisant les formulaires rangés dans le fichier binaire,
5. Afficher la liste,
6. Supprimer la liste.

```c
int main() {
  // form read from file function pointer
  struct form * (*ptr_read)(FILE *, enum mode_t) = &read_form;
  // form view function pointer
  void (*ptr_view)(struct form *) = &view_form;
  // form write into file function pointer
  void (*ptr_write)(struct form *, enum mode_t, FILE *) = &write_form;
  // form deletion function pointer
  void (*ptr_del)(struct form *) = &del_form;

  // create form list from text/binary file
  struct list * L = read_list(TEXT, (void * (*)()) ptr_read, ptr_del);
  // view form list
  view_list(L, ptr_view);

  // write form list into text/binary file
  write_list(L, BINARY, ptr_write);
  // delete form list
  del_list(L, ptr_del);

  // create form list from text/binary file
  L = read_list(BINARY, (void * (*)()) ptr_read, ptr_del);
  // view form list
  view_list(L, ptr_view);

  // delete form list
  del_list(L, ptr_del);
  return EXIT_SUCCESS;
}
```

Comme vous pouvez le constater en sus des formulaires et des listes vous avez aurez les outils de manipulations des fichiers textes à écrire (fichier `db.h`) :

```c
/* file mode : text or binary */
enum mode_t { TEXT, BINARY };

/** write a list into a text or binary file acording to the mode */
void write_list(struct list * L, enum mode_t mode, void (*ptrf)());

/** read a list from a text or binary file acording to the mode */
struct list * read_list(enum mode_t mode, void * (*ptrf)(), void (ptr_del)());
```

Le canevas des définitions des deux fonctions vous est donné (fichier `db.c`) :

```c
void write_list(struct list * L, enum mode_t mode, void (*ptrf)()) {
  FILE * fd;
  char fname[20];
```

```c

  if (mode == TEXT) {
    printf("\t\tWrite list to text file (.txt)\n\tfile name :");
    scanf("%s", fname);
    fd = fopen(fname, "wt");
  } else {
    printf("\t\tWrite list to binary file (.bin)\n\tfile name :");
    scanf("%s", fname);
    fd = fopen(fname, "wb");
  }
  assert(fd != NULL);

  /** TODO
    Vous devez parcourir la liste et écrire les formulaires qui y sont rangés grâce au pointeur de fonctions
  */
  fclose(fd);
}

struct list * read_list(enum mode_t mode, void * (*ptr_data)(), void (ptr_del)()) {
  /**
      on peut ajouter en argument un pointeur sur la fonction de comparaison des formulaires gt_form si l'on souhaite faire une insertion
            ordonnée (insert_ordered)
    */
  FILE * fd;
  char fname[20];

  if (mode == TEXT) {
    printf("\t\tRead list from text file (.txt)\n\tfile name :");
    scanf("%s", fname);
    fd = fopen(fname, "rt");
  }
  else {
    printf("\t\tRead list from binary file (.bin)\n\tfile name :");
    scanf("%s", fname);
    fd = fopen(fname, "rb");
  }
  assert(fd != NULL);

  /** TODO
    Vous devez parcourir l'ensemble du fichier pour y collecter les formulaires et les ranger dans la liste.
    ATTENTION : il est possible que vous créiez un élément de trop (un formulaire vide) en fin de liste.
    Il faut alors penser à le supprimer grâce au code suivant :
      E = get_tail (L);
      struct elmlist * T = getPred(E);
      set_suc(T, NULL);
      L->tail = T;
      del_elmlist(E, ptr_del);
    où ptr_del est le pointeur sur la fonction de suppression de la donnée (ici la donnée est un formulaire)
    }
  */
  fclose(fd);
  return L;
}
```

Enfin, vous aurez également besoin de définir la bibliothèque `outils` :

```c
/** Is *a greater than *b */
bool gt_int(int * a, int * b);

bool gt_double(double * x, double * y);

bool gt_string(char * s1, char * s2);



```

```c
/** Is *a less than *b */
bool lt_int(int * a, int * b);

bool lt_double(double * x, double * y);

bool lt_string(char * s1, char * s2);
```

Faites en sorte que l'ensemble fonctionne correctement.