Structures de données : listes chaînées abstraites

Terminer le TP précédent

1 Listes abstraites

Dans les TP précédents, les structures de listes dépendaient du contenu. Comment les rendre indépendantes de leur contenu ?

Une liste doit obligatoirement permettre les fonctionnalités suivantes :
- créer une liste vide
- libérer une liste
- tester si une liste est vide
- ajouter un élément en tête de liste
- retirer la tête de liste

Généralement, on ajoute les fonctionnalités :
- consulter la tête de liste
- afficher la liste
- insérer un élément dans la liste selon un ordre
- trier la liste selon un ordre

1.1 Étude de programme

Créez un nouveau répertoire et ajoutez les programmes fournis en annexe :
- makefile.tp3
- main.c, le programme principal permettant de tester le comportement des listes, piles et files.
- debug.c et debug.h, les utilitaires permettant de vérifier que les `malloc` et `free` sont bien appariés.
- util.c et util.h
- liste.c et liste.h
- contenu.c et contenu.h

Compilez-le (utilisez `make -f makefile.tp3`) et s'il subsiste des *warnings*, essayez de les comprendre et de les supprimer.

Exécutez plusieurs fois le programme pour vous familiariser avec lui.

Si le mode `DEBUG` est activé (ligne `#define DEBUG` du fichier `debug.h`), les allocations et libérations mémoire apparaîtront dans la fenêtre d'exécution et seront stockées dans des fichiers dont les contenus seront comparés en fin de programme.

1.2 Étude du fichier `liste.c`

Étudiez les fichiers `liste.c` et `liste.h`.

Représentez la mémoire après le lancement du programme principal, le choix de `test_liste` et l'ajout en tête de liste de deux contenus aléatoires.

1.3 Modification des contenus de liste

Pour l'instant, les objets sont des structures (nom,age).

Modifiez les fichiers `contenu.h` et `contenu.c` pour transformer les objets en structure (nom, date de naissance, date de décès).

La génération aléatoire d'un contenu sera une chaîne aléatoire, une date aléatoire comprise entre 1000 et 1900 pour la naissance et entre la date de naissance et la date de naissance + 110 pour le décès.

La comparaison sur contenu se fera dans l'ordre sur les noms, puis les dates de naissance, puis les dates de décès.

La comparaison sur valeur se fera dans l'ordre sur les dates de naissance, puis sur les dates de décès, puis sur les noms).

L'affichage se fera sous la forme `XXXX 1800-1900`.

1.4 Modification de la fonction de tri de liste

Vous pouvez constater en mode `DEBUG` que le tri tel qu'il est écrit génère de nombreux `free alloc` qui semblent inutiles.

Étudiez et réécrivez le tri de liste pour qu'il ne fasse plus aucune opération `free` ou `alloc`

2 Les files

Ajoutez ce qu'il faut pour pouvoir traiter et tester des files.

```c
/* main.c */

/*
test de liste, piles et files
 */



#include <stdio.h>
#include "debug.h"
#include "util.h"
#include "contenu.h"
#include "liste.h"




void test_pile() {
    char rep[8];

    contenu *pc;
    pile *pp;
    pp=cree_pile();
    for(;;) {
        printf("\n\
1. Consultation du (S)ommet\n\
2. (E)mpiler\n\
3. (D)epiler\n\
\n\
0. (F)inir\n\
Choix : ");
        saisie_chaine(rep,2);
        switch(rep[0]) {
            case 'E' : case 'e' : case '2' :
                ajoute_pile(pp, cree_alea_contenu());

                printf("%-20s : ", "Ajout");
                affiche_contenu(sommet_pile(pp));
                printf("\n");

                printf("%-10s : \n", "Pile :");
                affiche_pile(pp, (void (*) (void *))affiche_contenu);
                printf("\n");

                break;

            case 'D' : case 'd' : case '3' :
                pc =  (contenu *)retire_pile(pp);

                printf("%-20s : ","Retrait" );
                if (pc) {
                    affiche_contenu(pc);
                    printf("\n");
                    libere_contenu(pc);
                }
                else {
                    printf(" Impossible"); printf("\n");
                }
```

```
                printf("%-10s : \n", "Pile :");
                affiche_pile(pp,(void (*) (void *))affiche_contenu);
                printf("\n");

                break;

            case 'S' : case 's' : case '1' :
                pc =(contenu *)sommet_pile(pp);

                printf("%-20s : ","Sommet pile");
                if (pc) {
                    affiche_contenu(pc);
                }
                else {
                    printf("Impossible");
                }
                printf("\n");

                break;

            case 'F' : case 'f' : case '0' :

                libere_pile(pp,(void(*)(void *))libere_contenu);
                printf("Fin de test pile \n");
                return ;
        }
    }
}


void test_liste() {
    char rep[8];
    int (*comp)(contenu *, contenu *); /* ptr sur fonction de comparaison */

    contenu *pc;
    liste *pl;

    pl=cree_liste();

    for(;;) {
        printf("\n\
1. (T)ete de liste\n\
2. (A)jout en tete\n\
3. (R)etirer tete\n\
4. (I)nserer dans l'ordre des noms\n\
5. Tri par (N)om\n\
6. Tri par (V)aleur\n\
\n\
0. (F)inir\n\
Choix : ");
        saisie_chaine(rep,2);
        switch(rep[0]) {
            case 'A' : case 'a' : case '2' :
                ajoute_en_tete_liste(pl,cree_alea_contenu());

                printf("%-20s : ","Ajout en tete : ");
                affiche_contenu(tete_de_liste(pl));
                printf("\n");

                printf("%-10s : \n", "Liste");
```

```
            affiche_liste(pl,(void(*)(void *))affiche_contenu);
            printf("\n");

            break;

        case 'I' : case 'i' : case '4' :
            pc = cree_alea_contenu();

            printf("%-20s : ","Insertion ordonnee sur nom : ");
            affiche_contenu(pc);
            printf("\n");

            insere_liste(pl,pc,(int(*)(void*,void*))compare_contenu);

            printf("%-10s : \n", "Liste");
            affiche_liste(pl,(void(*)(void *))affiche_contenu);
            printf("\n");

            break;

        case 'R' : case 'r' : case '3' :
            pc =  (contenu *)retire_en_tete_liste(pl);

            printf("%-20s : ","Retrait tete de liste" );
            if (pc) {
                affiche_contenu(pc);
                printf("\n");
                libere_contenu(pc);
            }
            else {
                printf(" Impossible");
                printf("\n");
            }

            printf("%-10s : \n", "Liste");
            affiche_liste(pl, (void(*)(void *))affiche_contenu);
            printf("\n");

            break;

        case 'T' : case 't' : case '1' :
            printf("%-20s : ","Tete de liste");

            pc =(contenu *)tete_de_liste(pl);

            if (pc) {
                affiche_contenu(pc);
            }
            else {
                printf("Impossible");
            }
            printf("\n");

            break;

        case 'N' : case 'n' : case '5' :

        case 'V' : case 'v' : case '6' :
            if (rep[0]=='v' || rep[0]=='V' || rep[0]=='6' ) {
                comp = compare_valeur;
```

```
                }
                else {
                    comp = compare_contenu;
                }

                printf("Tri de liste : \n");
                tri_liste(pl,(int(*)(void*,void*))comp);

                printf("%-10s : \n", "Liste");
                affiche_liste(pl,(void(*)(void *))affiche_contenu);
                printf("\n");

                break;

            case 'F' : case 'f' : case '0' :
                libere_liste(pl,(void(*)(void *))libere_contenu);

                printf("Fin\n");

                return ;
        }
    }
}


void test_file() {
    printf("a faire");
}


int main() {
    char rep[5];
    int fin = 0;
#ifdef DEBUG
init_debug();
#endif


    while (! fin) {
        printf("1. test (L)iste \n");
        printf("2. test (P)ile \n");
        printf("3. test (F)ile \n");

        printf("\n0. fin \nChoix : ");

        saisie_chaine(rep,2);

        switch(rep[0]) {
            case '1' : case 'l' : case 'L' :
                test_liste();
                break;
            case '2' : case 'p' : case 'P' :
                test_pile();
                break;
            case '3' :  case 'f' : case 'F' :
                test_file();
                break;

            case '0' :
                fin = 1;
            default :
```

```
                break;
        }
    }

#ifdef DEBUG
fin_debug();
#endif
    return 0;
}
```

```
/* debug.h */
/* pour le debugage */

#include <stdio.h> /* pour FILE */

#ifndef DEBUGH
#define DEBUGH

/* a commenter pour un mode sans debug d'alloc memoire */
#define DEBUG


#ifdef DEBUG
FILE *f_malloc;
FILE *f_free;

void init_debug();
void fin_debug();



#endif



void * mon_malloc(size_t t);
void mon_free (void *p);
#endif


/*


sort malloc.dbg > malloc.srt
sort free.dbg > free.srt
FC malloc.srt free.srt
    Comparaison des fichiers malloc.srt et free.SRT
    FC : aucune difference trouvee
*/
```

```c
/* debug.c */

#include <stdio.h>
#include <stdlib.h>
#include "debug.h"

void * mon_malloc(size_t t) {
    void *p=malloc(t);
#ifdef DEBUG
    fprintf(f_malloc, "%p\n",p);
    fprintf(stdout, "%40s %p\n","mode DEBUG malloc",p);
#endif
    return p;
}


void mon_free(void *p) {
#ifdef DEBUG
    fprintf(f_free, "%p\n",p);
    fprintf(stdout, "%50s %p\n","mode DEBUG free",p);
#endif
    free(p);
}



#ifdef DEBUG
void init_debug() {
f_malloc = fopen("__malloc.dbg","wb");
f_free   = fopen("__free.dbg","wb");
}

void fin_debug() {
    fclose(f_free);
    fclose(f_malloc);
    system("sort __malloc.dbg > __malloc.srt");
    system("sort __free.dbg > __free.srt");
    if (system("diff __malloc.srt __free.srt")== 0)
      fprintf(stderr,"mode DEBUG : fichiers d'alloc et free identiques\n");
    else
      fprintf(stderr,"mode DEBUG problem : fichiers d'alloc et free differents\n");
}

/*
sort __malloc.dbg > __malloc.srt
sort __free.dbg > __free.srt
FC __malloc.srt __free.srt
*/
#endif
```

```c
/* util.h */
/*
utilitaires de saisie et de generation aleatoire de chaines
de caracteres et d'entiers
*/

#ifndef UTILH
#define UTILH


/* saisie de chaine */
char *saisie_chaine(char *chaine, int taille);


/* generation de chaine aleatoire de 'taille' majuscules */
char *alea_chaine(char *chaine, int taille);

/* saisie d'entier */
int saisie_int(int *pn);

/* generation d'un entier aleatoire entre a (inclus) et b(exclu) */
void alea_int(int *pn, int a, int b);

#endif
```

```c
/* util.c */
/* utilitaires de saisie */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void purge_clavier() {

    while (getchar()!='\n');
}

char *saisie_chaine(char *chaine, int taille) {
    char *p;
    fgets(chaine, taille, stdin);
    if ((p=(char *)memchr(chaine,'\n',taille))!=NULL) {
        *p = '\0';
    }
    else {
        purge_clavier();
    }
    chaine[taille]='\0';
    return chaine;
}

char *alea_chaine(char *chaine, int taille) {
    int i;
    for (i=0;i<taille;++i) {
        chaine[i]=rand()%26+'A';
    }
    chaine[taille]='\0';
    return chaine;
}

int saisie_int(int *pn) {
    long int n;
    static char buffer[256];
    char c,*p;
    p=&c;
    c='A';
    saisie_chaine(buffer,256);
    n = strtol(buffer,&p,10);
    if (*p=='\0') {
        *pn = (int)n;
        return 1;
    }
    return 0;
}

void alea_int(int *pn, int a, int b) {
    /* aleatoire entre a (inclus) et b(exclu) */
    *pn = a+rand()%(b-a);
}
```

```c
/* contenu.h */
typedef struct {
    char *nom;
    int age;
} contenu;

contenu * cree_contenu(char *nom, int age);

#define N 10
contenu * cree_alea_contenu() ;


void affiche_contenu(contenu *pc);

void libere_contenu(contenu *pc);

int compare_contenu(contenu *pc1, contenu* pc2);

int compare_valeur(contenu *pc1, contenu* pc2);
```

```c
/* contenu.c */


#include <stdio.h> /* pour printf */

#include <string.h> /* pour strlen strcmp et strcpy */
#include <stdlib.h> /* pour rand */


#include "contenu.h"
#include "debug.h"
#include "util.h"


contenu * cree_contenu(char *nom, int age) {
    contenu *pc = (contenu*)mon_malloc(sizeof(contenu));
    pc->nom = (char *)mon_malloc(strlen(nom)+1);
    strcpy(pc->nom, nom);
    pc->age = age;
    return pc;
}

#define N 10
contenu * cree_alea_contenu() {
    char nom[N+1];
    int age;
    alea_chaine(nom,2+rand()%(N-2));
    alea_int(&age,0,130);
    return cree_contenu(nom, age);
}


void affiche_contenu(contenu *pc){
    printf("%-*s (%3d)    ",N,pc->nom, pc->age);
}

void libere_contenu(contenu *pc) {
    mon_free(pc->nom);
    mon_free(pc);
}

int compare_contenu(contenu *pc1, contenu* pc2) {
    int c = strcmp(pc1->nom,pc2->nom);
    return c?c:(pc1->age - pc2->age);
}
int compare_valeur(contenu *pc1, contenu* pc2) {
    int c = (pc1->age - pc2->age);

    return c?c:strcmp(pc1->nom,pc2->nom);
}
```

```c
/* liste.h */

#ifndef LISTEH
#define LISTEH

typedef void * liste;

liste* cree_liste();
void libere_liste(liste *l, void(* libere_contenu)(void *));
int est_vide_liste(liste *l);

void ajoute_en_tete_liste(liste *l, void *pc);
void *retire_en_tete_liste(liste *l);
void *tete_de_liste(liste *l);
void affiche_liste(liste *l, void (*aff)(void *)) ;
void insere_liste (liste *l, void *pc, int (*compare)(void *, void *)) ;
void tri_liste(liste *l, int (*compare)(void *, void *));


typedef liste pile;

pile * cree_pile() ;
void* retire_pile(pile *p) ;
void ajoute_pile (pile *p, void *c) ;
void libere_pile(liste *l, void(* lib_contenu)(void *)) ;
void * sommet_pile(pile *p) ;
void affiche_pile(pile *p, void (* aff)(void *)) ;


#endif
```

```c
/* liste.c */

#include <stdio.h>
#include <stdlib.h>

#include "debug.h"

#include "liste.h"

typedef struct maillon {
    void * contenu;
    struct maillon *svt;
} maillon;

/* cuisine interne. N'est jamais vu de l'exterieur */

static maillon *cree_maillon(void *pc, maillon *svt) {
    maillon *pm = (maillon*)mon_malloc(sizeof(maillon));
    pm->contenu = pc;
    pm->svt = svt;
    return pm;
}

static void libere_maillon(maillon *pm, void(* lib_contenu)(void *)) {
    void * pc = pm->contenu;
    mon_free(pm);
    if (lib_contenu!=NULL) {
        lib_contenu(pc);
    }
}

liste *cree_liste() {
    liste *l=(liste *)mon_malloc(sizeof(liste));
    *l = NULL;
    return l;
}

int est_vide_liste(liste *l) {
    return *l==NULL;
}


void libere_liste(liste *l, void(* lib_contenu)(void *)) {

    maillon *pm = (maillon*)*l, *qm ;
    while (pm != NULL) {
        qm = pm->svt;
        libere_maillon(pm,lib_contenu);
        pm = qm;
    }
    mon_free(l);

}


void ajoute_en_tete_liste(liste *l, void *pc) {
    maillon *pm = cree_maillon(pc, (maillon*) *l);
    *l = pm;
}

void *tete_de_liste(liste *l) {
```

```
    if (! est_vide_liste(l)) {
        return ((maillon *)*l)->contenu;
    }
    else {
        return NULL;
    }
}

void *retire_en_tete_liste(liste *l) {
    if (! est_vide_liste(l)) {
        maillon *pm = (maillon*)*l;
        void *pc = pm->contenu;
        *l = pm -> svt;
        libere_maillon(pm, NULL);
        return pc;
    }
    else {
        return NULL;
    }
}

void affiche_liste(liste *l, void (*aff)(void *)) {
    maillon *pm = (maillon*)*l;
    while (pm!=NULL) {
        aff(pm->contenu);
        pm = pm->svt;
    }
}

void insere_liste (liste *l, void *pc, int (*compare)(void *, void *)) {
if (est_vide_liste(l) || compare(pc, tete_de_liste(l))<=0 ) {
        ajoute_en_tete_liste(l, pc);
    }
    else {
        insere_liste((liste *)&(((maillon *)*l)->svt), pc, compare);
    }
}

void tri_liste(liste *l, int (*compare)(void *, void *)) {
    void *pc;
    if (est_vide_liste(l) )
        return;
    tri_liste((liste *)&(((maillon *)*l)->svt), compare);
    pc = retire_en_tete_liste(l);
    insere_liste(l, pc,compare);
}



pile * cree_pile() {
    return cree_liste();
}

void* retire_pile(pile *p) {
    return retire_en_tete_liste(p);
}
void ajoute_pile (pile *p, void *c) {
    ajoute_en_tete_liste(p, c);
}
```

```
void libere_pile(liste *l, void(* lib_contenu)(void *)){
    libere_liste(l, lib_contenu);
}

void * sommet_pile(pile *p) {
    return tete_de_liste(p);
}

void affiche_pile(pile *p, void (* aff)(void *)) {
    affiche_liste((liste *)p, aff);
}
```