Structures de données : listes chaînées et files (corrigé)

1 Files

1.1 La structure `contenu`

Exercice 1. Définition et fonctions d'un type `contenu`

Soit la structure suivante :

```
#define TAILLEMAX 10
typedef struct {
    char tab[TAILLEMAX];
} contenu;
```

Le tableau de caractères `tab` peut contenir soit une chaîne de `TAILLEMAX-1` caractères ou moins terminée par un `'\0'`, soit `TAILLEMAX` caractères

Définir une fonction d'affichage `void afficher_contenu(contenu *pc)` qui affiche les caractères du champ `tab` du contenu pointé par `pc` jusqu'à, soit rencontrer un `'\0'`, soit en avoir affiché `TAILLEMAX`.

Corrigé

```
**************************************************************************
void afficher_contenu(contenu *pc) {
    int i;
    for (i = 0; i < TAILLEMAX; ++i) {
        if (pc->tab[i] == '\0') {
            break;
        }
        else {
            putchar(pc->tab[i]);
        }
    }
}
**************************************************************************
```

Définir une fonction de saisie `void saisir_contenu(contenu *pc)` qui stocke au maximum 10 caractères dans le champ `tab` du contenu pointé par `pc`, marque un `'\0'` en place d'un éventuel `'\n'`, et vide le buffer clavier.

Corrigé

```
**************************************************************************
void saisir_contenu(contenu *pc) {
    int i,c;
    for (i = 0; i < TAILLEMAX && ((c = getchar()) != '\n'); ++i) {
        pc->tab[i]=c;
    }
    /* si l'on n'a pas atteint TAILLEMAX */
    if (i < TAILLEMAX ) { /* on marque la fin de la chaine */
        pc->tab[i] = '\0';
    }
    else { /* on vide le buffer clavier */
        while (getchar() != '\n');
    }
}
**************************************************************************
```

Définir une fonction de comparaison `int comparer_contenu(contenu *pc1, contenu *pc2)` retournant un entier négatif si le champ `tab` du contenu pointé par `pc1` est avant celui de `pc2` par ordre alphabétique, un entier positif s'il est après et `0` en cas d'égalité.

Corrigé

```
*****************************************************************************
int comparer_contenu(contenu *pc1, contenu *pc2) {
    int i;
    for (i = 0; i < TAILLEMAX; ++i) {
        if (pc1->tab[i] != pc2->tab[i]) {
            return pc1->tab[i] - pc2->tab[i];
        }
        if (pc1->tab[i] == '\0') {
            return 0;
        }
    }
    return 0;
}
*****************************************************************************
```

Définir une fonction de comparaison `int comparer_chaine (contenu *pc1, char *s)` retournant un entier négatif si le champ `tab` du contenu pointé par `pc1` est avant `s` par ordre alphabétique, un entier positif s'il est après et `0` en cas d'égalité.

Corrigé

```
*****************************************************************************
int comparer_chaine(contenu *pc, char *s) {
    int i;
    for (i = 0; i < TAILLEMAX; ++i) {
        if (pc->tab[i] !=s [i]) {
            return pc->tab[i] - s[i];
        }
        if (pc->tab[i] == '\0') {
            return 0;
        }
    }
    return 0;
}
*****************************************************************************
```

Définir une fonction d'affectation `void affecter_contenu(contenu *pcdest, contenu *pcsource)` qui recopie dans le champ `tab` du contenu pointé par `pdest` les caractères du champ `tab` du contenu pointé par `psource`

Corrigé

```
*****************************************************************************
void affecter_contenu(contenu *pcdest, contenu *pcsource) {

    /*******************************************
    exemple de one liner (a eviter en regle generale,
    mais a devoir comprendre).
    La 2e condition de continuation de la boucle
            pcdest->tab[i] = pcsource->tab[i]
    effectue l'affectation (c'est bien = et non == qui est ecrit) et sera
    evaluee comme FAUX si l'affectation concernee vaut 0 ('\0')
    Le corps de la boucle est vide.

    Equivalent a :

    for (i = 0; i < TAILLEMAX; ++i) {
        pcdest->tab[i] = pcsource->tab[i];
        if (pcdest->tab[i] == '\0') {
            break;
        }
    }
    ******************************************/
    int i;
    for (i = 0; i < TAILLEMAX && (pcdest->tab[i] = pcsource->tab[i]); ++i);
}
*****************************************************************************
```

1.2 Les files

Exercice 2. Définition et fonctions d'une file

Avec le type `contenu` ci-dessus, définir tous les types et toutes les fonctions nécessaires pour que le programme principal suivant fonctionne :

```
int main() {
    contenu txt;

    file *pf = creer_file();

    for(; ; ) {
        printf("Ajouter un texte a la file (! pour retirer, * pour finir) : ");
        saisir_contenu(&txt);
        if (comparer_chaine(&txt, "*") == 0)
            break;
        if (comparer_chaine(&txt, "!") == 0) {
            if (est_vide_file(pf)) {
               printf("Retrait imposible. File vide");
            }
           else {
              defiler_file(pf, &txt);
            }
            afficher_file(pf);
            printf("\n");
       }
        else {
            enfiler_file(pf, &txt);
            afficher_file(pf);
            printf("\n");
        }
    }
    liberer_file(pf);
    getchar();
    return 0;
}
```

Il devra produire un résultat similaire à l'exemple d'exécution ci-dessous :

```
Ajouter un texte a la file (! pour retirer, * pour finir) : riri
<riri>
Ajouter un texte a la file (! pour retirer, * pour finir) : fifi
<riri,fifi>
Ajouter un texte a la file (! pour retirer, * pour finir) : loulou
<riri,fifi,loulou>
Ajouter un texte a la file (! pour retirer, * pour finir) : !
<fifi,loulou>
Ajouter un texte a la file (! pour retirer, * pour finir) : !
<loulou>
Ajouter un texte a la file (! pour retirer, * pour finir) : !
<>
Ajouter un texte a la file (! pour retirer, * pour finir) : !
Retrait imposible. File vide<>
Ajouter un texte a la file (! pour retirer, * pour finir) : nafnaf
<nafnaf>
Ajouter un texte a la file (! pour retirer, * pour finir) : nifnif
<nafnaf,nifnif>
Ajouter un texte a la file (! pour retirer, * pour finir) : noufnouf
<nafnaf,nifnif,noufnouf>
Ajouter un texte a la file (! pour retirer, * pour finir) : !
<nifnif,noufnouf>
Ajouter un texte a la file (! pour retirer, * pour finir) : *
```

Réfléchir à un moyen de vérifier que tous les `malloc` sont bien appariés par des `free` lors de l'exécution du programme.

Si vous avez écrit tous les programmes dans le même fichier `.c`, réfléchissez à le découper en plusieurs fichiers sources `.c` et en-têtes `.h`, de manière à regrouper les programmes par fonctionnalités (file, contenu, tests).

Corrigé

****

Le fichier d'en-tête

```
/*****************************************************************************
tp2_file.h
*****************************************************************************/
#ifndef _TP2_FILE_H

#define _TP2_FILE_H


/* un maillon est compose d'un pointeur sur contenu
et d'un pointeur suivant
              -----------
             |           |
       ---> |  contenu  |
      /     |___________|
 ____/______             ___________
|    / |     |          |     |     |
|  o   |  o--------------> o   |  o --------->
|_____|_____|          |_____|_____|
  pc    svt

*/
typedef struct maillon {
    contenu *pc;
    struct maillon *svt;
} maillon;

/* une file est composee de deux pointeurs sur maillon : tete et queue

              -----------      -----------      -----------
             |     |     |    |     |     |    |     |   / |
        --->|  o  |  o------>|  o  |  o------>|  o  |  /  |
       /    |_____|_____|    |_____|_____|    |_____|_/___|
      /                                         ^
 ____/______                                    |
|    / |     |                                  |
|  o   |  o-------------------------------------
|_____|_____|
  tete queue

*/

typedef struct {
    maillon *tete, *queue;
} file;


maillon *allouer_maillon();
void liberer_maillon(maillon *pm) ;


file *allouer_file();
void init_file(file *pf);

file* creer_file ();
void liberer_file (file *pf);

int est_vide_file(file *pf);

int defiler_file(file *pf, contenu *pc);
int enfiler_file(file *pf, contenu *pc);

void afficher_file(file *pf);
```

```
#endif
```

et le fichier source

```
/*****************************************************************************
tp2_file.c
*****************************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include "tp2_contenu.h"
#include "tp2_file.h"


file *allouer_file()  {
    file *pf = malloc(sizeof(file));
    return pf;
}

void init_file(file *pf) {
    pf->tete = pf->queue = NULL;
}

/* creer une file : allouer la memoire necessaire et
initialiser les deux pointeurs tete et queue a NULL
*/
file* creer_file (){
    file *pf = allouer_file();
    init_file(pf);
    return pf;
}



maillon *allouer_maillon() {
    maillon *pm = malloc(sizeof(maillon));
    pm->pc = malloc(sizeof(contenu));
    return pm;
}

void liberer_maillon(maillon *pm) {
    free(pm->pc);
    free(pm);
}

void liberer_file (file *pf) {
    maillon *pm = pf->tete, *pc;
    while (pm != NULL) {
        pc = pm->svt;
        liberer_maillon(pm);
        pm=pc;
    }
    free(pf);
}

int est_vide_file(file *pf) {
    return pf->tete == NULL;
}

int defiler_file(file *pf, contenu *pc) {
    maillon *pm = pf->tete;

    pf->tete = pm->svt;
    if (pf->tete == NULL) {
        pf->queue = NULL;
    }

    affecter_contenu(pc, pm->pc);

    liberer_maillon(pm);
```

```
    return 0;
}

int enfiler_file(file *pf, contenu *pc) {
    maillon *pm = allouer_maillon();

    affecter_contenu(pm->pc, pc);
    pm->svt = NULL;

    if (pf->tete == NULL) {
        pf->tete = pf->queue=pm;
    }
    else {
        pf->queue->svt = pm;
        pf->queue = pm;
    }
    return 0;
}


void afficher_file(file *pf) {
    maillon *pm = pf->tete;
    printf("<");

    while (pm != NULL) {
        if (pm != pf->tete) {
            printf("%c", ',');
        }
        afficher_contenu(pm->pc);

        pm = pm->svt;
    }
    printf(">\n");
}
```

****

2 Le tri par éclatement/fusion

Le but de cette section est d'écrire un tri par éclatement-fusion (*mergesort*, **le tri** sur les listes chaînées) dont le principe est le suivant :

- si la liste a au moins deux éléments, scinder la liste en deux listes : une liste contenant un maillon sur deux à partir de la tête de liste initiale et une liste contenant un maillon sur deux à partir du suivant de la tête de liste initiale ;
- trier récursivement chacune des deux listes ;
- fusionner les deux listes triées en une seule.

Ce tri est en $n \log(n)$ et a l'énorme avantage de laisser les éléments en place (seuls les chaînages sont modifiés).

Voici par exemple les opérations nécessaires à trier par ordre alphabétique les éléments d'une liste :

U	→	Y	→	T	→	R	→	E	→	Z	→	A
U	→	T	→	E	→	A		Y	→	R	→	Z
U	→	E		T	→	A		Y	→	Z		R
U		E		T		A		Y		Z		R
E	→	U		A	→	T		Y	→	Z		R
A	→	E	→	T	→	U		R	→	Y	→	Z
A	→	E	→	R	→	T	→	U	→	Y	→	Z

Exercice 3. Créer une liste chaînée

Avec la structure

```
typedef struct _m {
    int n;
    struct _m *svt;
} maillon, *liste;
```

créez une liste de 10 000 éléments entiers aléatoires.

Exercice 4. Scinder une liste

Écrire la fonction qui scinde une liste en deux : la liste des éléments de rang pair dans la liste initiale, et celle des éléments de rang impair.

```
void scission(liste lst, liste *p_lst1, liste *p_lst2;
```

Corrigé

```
*************************************************************************
void scission(liste lst, liste *p_lst1, liste *p_lst2) {
    if (lst == NULL || lst->svt == NULL) {
        *p_lst1 = lst;
        *p_lst2 = NULL;
        return;
    }
    liste lst1=NULL, lst2=NULL;
    scission(lst->svt->svt, &lst1, &lst2);

    lst->svt->svt = lst2;
    *p_lst2 = lst->svt;
    lst->svt = lst1;
    *p_lst1 = lst;
}
*************************************************************************
```

Exercice 5. Fusionner deux listes triées

Écrire la fonction qui fusionne deux listes triées en une seule liste triée.

```
void fusion(liste lst1, liste lst2, liste *p_lst);
```

Corrigé

```
*************************************************************************
void fusion(liste lst1, liste lst2, liste *p_lst) {

    liste lst;
    if (lst1 == NULL)
        *p_lst = lst2;
    else if (lst2 == NULL)
        *p_lst = lst1;
    else {
        if (lst1->n < lst2->n) {
            fusion(lst1->svt, lst2, &lst);
            lst1->svt = lst;
            *p_lst = lst1;
        }
        else {
            fusion(lst1, lst2->svt, &lst);
            lst2->svt = lst;
            *p_lst = lst2;
        }
    }
}
*************************************************************************
```

Exercice 6. Tri fusion

Écrire la fonction qui trie une liste sur elle-même.

```
void tri(liste * p_lst);
```

Corrigé

```
**************************************************************************
void tri(liste * p_lst) {
    liste lst1, lst2;
    if (*p_lst == NULL ||       (*p_lst)->svt == NULL)
      return;
    scission(*p_lst, &lst1, &lst2);

    tri(&lst1);
    tri(&lst2);
    fusion(lst1, lst2, p_lst);
}
**************************************************************************
```

3 Le codage de Huffman (1)

Exercice 7. Liste des fréquences

Écrivez la fonction `void calcule_freq(FILE *f, unsigned long int t[])` qui remplit un tableau `t` des 256 fréquences d'octets du fichier pointé par `f`.

Créez la liste des fréquences avec les contenus suivants :

```
typedef struct {
    unsigned char c;
    unsigned long int freq;
} contenu;
```

et triez la par ordre de fréquence croissante. Testez-le avec le fichier dico.txt.

Corrigé

```
**************************************************************************
/*************************************************************************
tp2_huffman1.c
*************************************************************************/

#include <stdio.h>
#include <limits.h>

#define NB_OCTETS 1<<CHAR_BIT

typedef struct {
    unsigned char c;
    unsigned long int freq;
} contenu;

void calcule_freq(FILE *f, contenu t[]) {
    int i,b;
    for (i = 0; i < NB_OCTETS; ++i) {
        t[i].c = i;
        t[i].freq = 0;
    }
    fseek(f, 0, SEEK_SET);
    while ((i = fgetc(f)) != EOF)
            ++(t[i].freq);
}

void tri(contenu t[], int n) {
    /* un bubble sort */
```

```
    int i, trie = 0;
    contenu x;
    while (!trie ) {
        trie = 1;
        for (i = 1; i < n; ++i) {
            if (t[i].freq < t[i - 1].freq) {
                x = t[i];
                t[i] = t[i-1];
                t[i-1] = x;
                trie = 0;
            }
        }
        --n;
    }
}

void qs(contenu t[], int n) {
    /* un quick sort */
    int k;
    if (n >= 2) {
        k = partition(t, n);
        qs(t, k);
        qs(t+k+1, n-k-1);
    }
}

int partition(contenu t[], int n) {
    int d = 1,f = n;
    contenu x;
    while (d < f) {
        if (t[d].freq > t[0].freq) {
            --f;
            x = t[d];
            t[d] = t[f];
            t[f] = x;
        }
        else {
            ++d;
        }
    }
    --d;
    x = t[d];
    t[d] = t[0];
    t[0] = x;
    return d;
}

void tri2(contenu t[], int n) {
    qs(t, n);
}

void affiche_freq(contenu freq[], int n) {
    int i;
    for (i = 0; i < n; ++i) {
       if (i%4 == 0) printf("\n");
       if (freq[i].freq > 0 && freq[i].c >= 32) {
            printf(" %c ", freq[i].c);
        }
        else {
            printf("    ");
        }
        printf("(%02x) %6lu\t", freq[i].c, freq[i].freq);
     }
}

int main() {
    contenu freq[NB_OCTETS];
    int i;
    FILE *f = fopen("dico.txt", "rb");
    if (f == NULL){
```

```
        fprintf(stderr, "Erreur d'ouverture fichier \n");
        getchar();
        return 1;
    }
    printf("Lecture\n");
    calcule_freq(f, freq);
    fclose(f);


    printf("Affichage des frequences\n");
    affiche_freq(freq, NB_OCTETS);

    printf("\n\n\n");
    printf("Tri des frequences\n");
    tri2(freq, NB_OCTETS);
    printf("Affichage des frequences triees\n");
    affiche_freq(freq, NB_OCTETS);

    getchar();

    return 0;
}
```

****