Programmation C (E. Lazard)
Examen du 22 janvier 2013

CORRECTION

(durée 2h)

I. Chaînes de caractères

On possède un fichier texte dans lequel sont répartis des nombres entiers positifs. On souhaite les extraire et les afficher. Chaque ligne du fichier est composée de caractères quelconques parmi lesquels peuvent se glisser 0, 1 ou plusieurs nombres (deux nombres sont forcément séparés par au moins un caractère non-numérique).

1. Écrivez une fonction

   ```
   int *getInts(FILE *f, int *nbr);
   ```

   qui reçoit comme paramètres :

   - un pointeur `f` sur un fichier ouvert dont on lira les lignes ;
   - un pointeur `nbr` sur un entier dans lequel on mettra le total de nombres lus ;

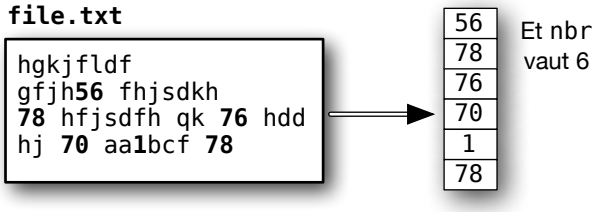
   et qui renvoie un tableau d'entiers dans lequel on stockera les nombres lus. On demande à ce que ce tableau soit le plus petit possible.

   On garantit que chaque ligne fait moins de 100 caractères et qu'il y a moins de 100 nombres au total.

   On pourra s'aider des fonctions suivantes :

   - `int fgets(char *s, int n, FILE *in);` qui lit une ligne d'au plus `n`-1 caractères du fichier dont le pointeur est dans le paramètre `in` et la met dans le tableau `s` (qui doit exister et être au moins de taille `n`). Renvoie 0 si on est arrivé à la fin du fichier ;
   - `int isdigit(char c);` qui teste si le caractère c est numérique. Renvoie 1 dans ce cas et 0 sinon ;
   - `int atoi(char *s);` qui renvoie la valeur numérique d'un nombre écrit en caractères et s'arrête au premier caractère non-numérique rencontré. Ainsi, les trois chaînes `"42"`, `"42a"` et `"42 6"` ont toutes 42 comme valeur numérique (si la chaîne ne commence pas par un chiffre ou par `'-'`, `atoi()` renvoie 0).

   Voici un exemple où appliquer la fonction sur le fichier de gauche (les nombres sont en gras pour bien les identifier dans l'exemple) renvoie le tableau de droite.



2. Écrivez un `main()` qui ouvre un fichier `"file.txt"`, appelle la fonction ci-dessus et affiche tous les nombres récupérés.

Corrigé :

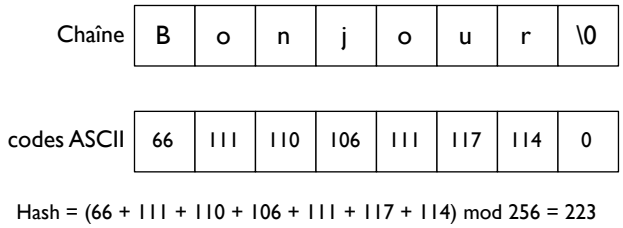
Listing 1. ***Tableau d'entiers***

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int *getInts(FILE *f, int *nbr) {
    char tmp[100];    /* stockage temporaire de la ligne */
    char *ptr;       /* parcours de la ligne            */
    int tmpTab[100]; /* tableau temporaire des nombres  */
    int *tab;        /* pour le tableau final           */
    int i;
    *nbr = 0;
    while (fgets(tmp, 1000, f)) { /* lire une ligne */
        ptr = tmp;
        while (*ptr) {
            if (isdigit(*ptr)) { /* on cherche un chiffre */
                tmpTab[(*nbr)++] = atoi(ptr); /* transformé en entier */
                while (isdigit(*ptr))
                    ptr++; /* on se déplace après le nombre */
            } else
                ptr++; /* prochain caractère */
        }
    }
    /* il faut maintenant créer un tableau de la bonne taille */
    tab = malloc(*nbr * sizeof(int));
    for (i = 0; i < *nbr; i++)
        tab[i] = tmpTab[i];
    /* on peut aussi écrire memcpy(tab, tmpTab, *nbr * sizeof(int)); */
    return tab;
}

int main(int argc, char **argv) {
    int *tab;
    int nbr, i;

    FILE *f = fopen("file.txt", "r");
    tab = getInts(f, &nbr);
    for (i = 0; i < nbr; i++)
        printf("%d\n", tab[i]);
    return 0;
}
```

II. Hachage de chaînes

On souhaite gérer une table de hachage pour des chaînes de caractères afin de permettre une recherche rapide d'une chaîne déjà stockée.

1. Le premier travail consiste à écrire une fonction de hachage renvoyant un octet non signé (entre 0 et 255, stocké dans un caractère non-signé) à partir d'une chaîne de caractères. Pour ce faire, on décide d'utiliser le code ASCII de chaque caractère : ceux-ci sont additionnés (l'addition s'effectuant via des caractères non-signés, on garantit ainsi que le résultat le sera également, les chiffres débordants étant éliminés).



Écrivez une fonction

```
unsigned char hash(char *str);
```

qui renvoie la valeur de hachage calculée à partir de la chaîne passée en paramètre.

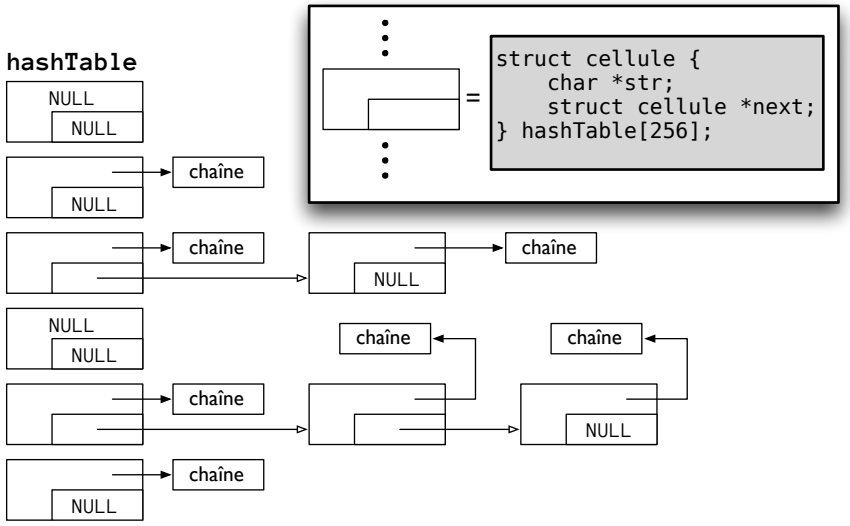
2. Le principe de la table de hachage est ensuite de stocker les chaînes dans une table indexée par la valeur de hachage de la chaîne. La table est un tableau de `char *` pointant sur une copie de la chaîne. Ainsi, une copie de la chaîne `"Bonjour"` sera stockée dans l'élément 223 de la table. On supposera pour l'instant qu'il n'y a jamais de collision (chaque chaîne a une valeur de hachage différente et deux chaînes ne peuvent pas occuper la même case du tableau).
   En supposant que l'on dispose d'un tableau `char *hashTable[256]` dont chaque élément est initialisé à `NULL`, écrivez une fonction

   ```
   void insere(char *str);
   ```

   qui insère une copie de la chaîne dans le tableau de hachage.
   Écrivez une fonction

   ```
   int recherche(char *str);
   ```

   qui cherche dans le tableau s'il existe déjà un exemplaire de la chaîne passée en paramètre et renvoie 1 si c'est le cas (et 0 sinon).
   On pourra utiliser les fonctions suivantes :
   - `int strcmp(char *s1, char *s2);` permet de comparer des chaînes de caractères et renvoie 1 si $s1 > s2$, 0 si $s1 = s2$ et -1 si $s1 < s2$;
   - `char *strdup(char *);` permet de dupliquer une chaîne de caractères passée en argument en allouant la place mémoire nécessaire.
3. *(plus difficile)* On autorise maintenant les collisions (deux chaînes différentes peuvent avoir la même valeur de hachage). Pour pouvoir stocker toutes les chaines ayant la même valeur de hachage, chaque élément du tableau devient une cellule d'une liste chaînée :



Ex : Le tableau est créé avec une première cellule vide. Une première chaîne peut être stockée dans cette cellule. Si une deuxième chaîne est insérée au même indice (elle a la même valeur de hachage

que la première), une nouvelle cellule est créée, chaînée à la première.
Écrivez une fonction

```
void insere2(char *str);
```

qui insère une copie de la chaîne dans le tableau de hachage, en tenant compte des collisions.
Écrivez une fonction

```
int recherche2(char *str);
```

qui cherche dans le tableau s'il existe déjà un exemplaire de la chaîne passée en paramètre et renvoie 1 si c'est le cas (et 0 sinon).

Corrigé :

Listing 2. ***hash***

```
struct cellule {
    char *str;
    struct cellule *next;
} hashTable[256];

unsigned char hash(char *str) {
    unsigned char h = 0;
    int i;
    if (str)
        while (*str)
            h += *(str++);
    return h;
}

void insere(char *str) {
    unsigned char h = hash(str);
    hashTable[h] = strdup(str);
}

int recherche(char *str) {
    unsigned char h = hash(str);
    if ((hashTable[h] == NULL) || (strcmp(hashTable[h], str) != 0))
        return 0;
    else
        return 1;
}

void insere2(char *str) {
    unsigned char h = hash(str);
    struct cellule *p = &hashTable[h];
    while (p->next != NULL)
        p = p->next; /* on va au bout de la liste */
    if (p->str != NULL) { /* cellule occupée */
        p->next = malloc(sizeof(struct cellule));
        p = p->next;
    }
    p->str = strdup(str);
    p->next = NULL;
}

int recherche2(char *str) {
    unsigned char h = hash(str);
    struct cellule *p = &hashTable[h];
```

```
    while ((p != NULL) && (p->str != NULL)) {
        if (strcmp(p->str, str) == 0)
            return 1;
        else
            p = p->next;
    }
    return 0;
}
```

III. Nombres

Écrivez une fonction

```
void afficheBase(unsigned int val, unsigned short base);
```

qui affiche une chaîne de caractères correspondant à la valeur numérique contenue dans `val` mais dans la base indiquée (qu'on supposera inférieure ou égale à 16). Exemples : `afficheBase(235, 10)` affichera `"235"`, `afficheBase(235, 7)` affichera `"454"`, `afficheBase(235, 11)` affichera `"1a4"`.

Corrigé :

Listing 3. ***afficheBase***

```
void afficheBase(unsigned int val, unsigned short base) {
/* On construit la chaîne puis on l'affiche dans l'ordre inverse */
    char str[32]; /* On a 32 bits au maximum */
    int index = 0;
    int modulo;

    do {
        modulo = val%base;
        val /= base;
        str[index++] = (modulo <= 9) ? '0' + modulo : 'a' + modulo - 10;
    } while (val > 0);
    while (index > 0)
        printf ("%c", str[--index]);
}
```

Ou une autre solution récursive :

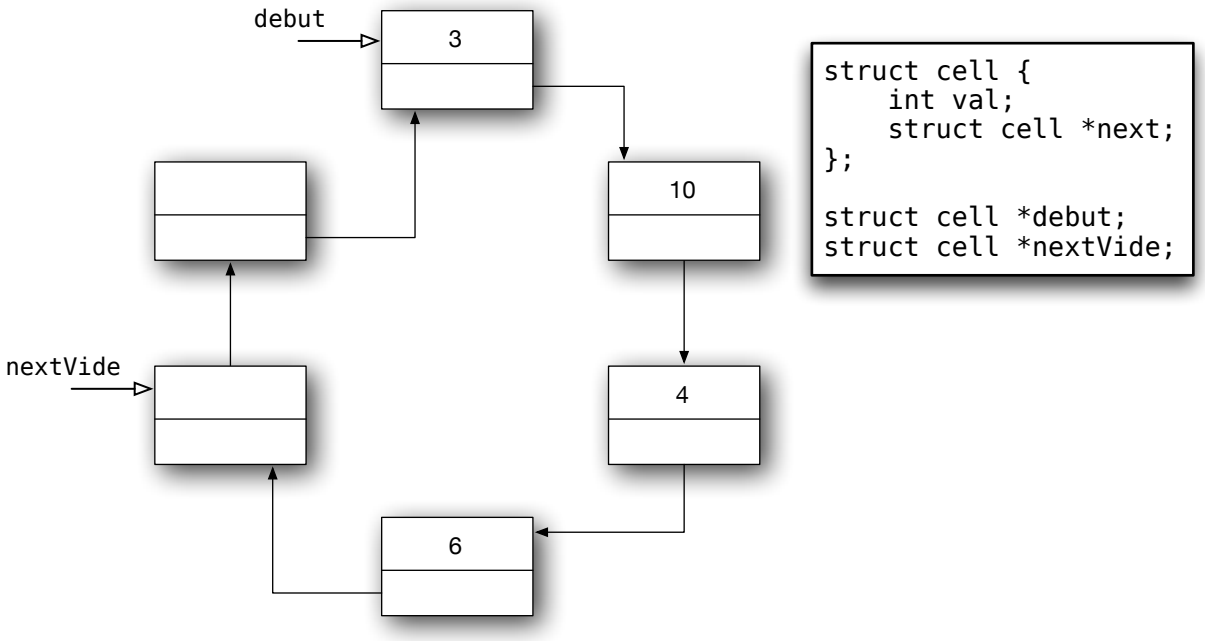
Listing 4. ***afficheBase***

```
void afficheBase(unsigned int val, unsigned short base) {
    int modulo;

    if (val/base != 0)
        afficheBase(val/base, base);
    modulo = val%base;
    printf("%c", (modulo <= 9) ? '0' + modulo : 'a' + modulo - 10);
}
```

IV. File

On souhaite gérer une file d'attente FIFO de valeurs entières (pensez à un numéro de tickets d'attente dans une boutique). Avec cette file, on doit pouvoir récupérer le premier numéro de la file, et insérer un numéro en queue de file. Il est possible de gérer cette file d'attente avec une structure de liste chaînée mais

cela oblige à supprimer une cellule lors du retrait d'un ticket et à en créer une lors de l'insertion, opérations qui peuvent être coûteuses en temps. On décide plutôt d'utiliser une liste circulaire où les cellules sont créées dès le départ et où deux pointeurs indiquent respectivement la première cellule occupée (le début de la file) et la première cellule vide (fin de la file) :



Récupérer le premier élément de la liste revient à prendre sa valeur puis à déplacer le pointeur `debut` ; insérer un élément revient à initialiser la cellule puis à déplacer le pointeur `nextVide`. L'avantage de cette structure est qu'elle ne nécessite aucune création de cellule mais simplement des déplacements de pointeurs.

- `debut` sera toujours initialisé, même si la file est vide ;
- si le pointeur `nextVide` est nul, cela signifie que la file est pleine.

1. Écrivez une fonction
   ```
   void init(int taille);
   ```
   qui crée une structure de file avec autant de cellules qu'indiqué et initialise les pointeurs globaux.
2. Écrivez une fonction
   ```
   int estVide(void);
   ```
   qui teste si la file est vide et renvoie 1 dans ce cas et 0 sinon.
3. Écrivez une fonction
   ```
   int put(int val);
   ```
   qui insère la valeur `val` en fin de file ; la fonction renvoie 1 si l'insertion a pu se faire et 0 sinon.
4. Écrivez une fonction
   ```
   int getNext(int *val);
   ```
   qui récupère la cellule en tête de file et met sa valeur dans la variable passée en paramètre par référence ; la fonction renvoie 1 si la valeur a correctement été récupérée et 0 sinon.

Corrigé :

Listing 5. ***file***

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct cell {
    int val;
    struct cell *next;
};

struct cell *debut = NULL;
struct cell *nextVide = NULL;

void init(int taille) {
    struct cell *p;
    int i;

    if (taille == 0) {
        debut = nextVide = NULL;
        return;
    } /* on crée une première cellule */
    p = nextVide = debut = malloc(sizeof(struct cell));
    /* puis toutes les autres */
    for (i = 1; i < taille; i++) {
        p->next = malloc(sizeof(struct cell));
        p = p->next;
    }
    /* et on termine en bouclant sur la première cellule */
    p->next = debut;
}

int estVide(void) {
    return nextVide == debut;
}

int put(int val) {
    if (nextVide == NULL)
        return 0; /* pas possible si la file est pleine */
    nextVide->val = val;
    nextVide = nextVide -> next;
    /* est-elle pleine maintenant ? */
    if (nextVide == debut)
        nextVide = NULL;
    return 1;
}

int getNext(int *val) {
    if (estVide())
        return 0;
    *val = debut->val;
    /* si la file était pleine, elle ne l'est plus */
    if (nextVide == NULL)
        nextVide = debut;
    /* on décale le pointeur */
    debut = debut->next;
    return 1;
}
```