Programmation C (E. Lazard)
Examen du 7 février 2014

CORRECTION

(durée 2h)

I. Expressions

Donner les résultats affichés par le programme suivant :

```
int f1(int i)  { return i--;                    }
int f2(int i)  { return ++i;                    }
int f3(int *i) { ++(*i); return (*i)++;         }
int f4(int *i) { printf("%d\n",*i==1); return *i; }
int f5(int *i) { printf("%d\n",*i=2);  return *i; }
int f6(int i)  { return i+1;                    }
int main() {
        int a, b;
        a=f1(1);  b=f2(2);  printf("a=%d, b=%d\n",a,b);
        a=f3(&b); printf("a=%d, b=%d\n",a,b);
        b=f4(&b);
        a=f5(&a);
        b=f6(b);    printf("a=%d, b=%d\n",a,b);
}
```

Corrigé :

- `f1` renvoie la valeur de `i`, la variable locale est modifiée.
- `f2` renvoie la valeur de `i+1`, la variable locale est modifiée.
- `f3` incrémente `b` de 2 mais retourne la valeur de `b+1`.
- `f4` affiche le résultat du test `*i==1` (0 ou 1) ; ne modifie pas `*i`.
- `f5` affiche toujours 2, modifie `a`, renvoie `*i` qui vaut 2.
- `f6` ne modifie pas `i`, retourne la valeur de `i+1`.

Ce programme affiche donc :

```
a=1, b=3
a=4, b=5
0
2
a=2, b=6
```

II. Chaîne

Le but de cet exercice est de réécrire un certain nombre de fonctions de traitement de chaînes de caractères semblables à celles disponibles dans la bibliothèque `string`. On ne pourra donc pas utiliser de fonctions déclarées dans `string.h` à l'exception de `strlen()`.

1. Écrire une fonction

   ```
   void myStrcat(char *s1, char *s2);
   ```

   qui concatène la chaîne `s2` à la chaîne `s1` (c'est-à-dire met `s2` à la suite de `s1`). On supposera que `s1` et `s2` sont bien définies est qu'il y a la place pour recopier `s2`.

2. Écrire une fonction

   `int myStrspn(char *s1, char *s2);`

   qui renvoie la longueur du plus long préfixe de `s1` ne contenant que des caractères se trouvant dans la chaîne de `s2`. Par exemple `strspn("3456ART","0123456789")` renvoie 4.

3. Écrire une fonction

   `int countChar(char *str, char c);`

   qui renvoie le nombre de fois que le caractère `c` se trouve dans la chaîne `str`.
   Par exemple `countChar("anticonstitutionnellement", 't')` renvoie 5.

4. Écrire une fonction

   `int countMultChar(char *str, char *cc);`

   qui renvoie le nombre de fois qu'un caractère quelconque de la chaîne `cc` se trouve dans la chaîne `str`. Par exemple `countMultChar("anticonstitutionnellement", "ton")` renvoie 12. On supposera que tous les caractères de `cc` sont différents ; on pourra utiliser la fonction de la question précédente.

Corrigé :

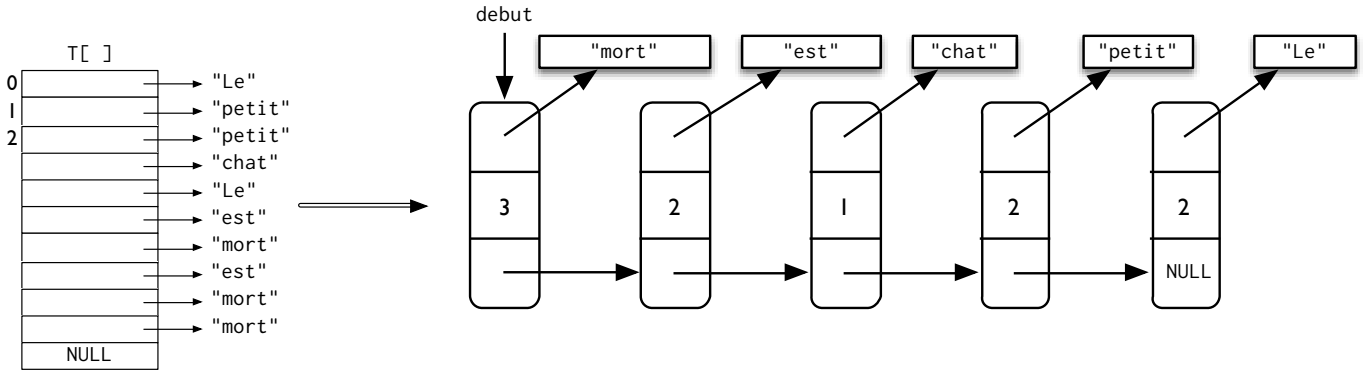
Listing 1. ***chaînes***

```c
void myStrcat(char *s1, char *s2) {
    int x = strlen(s1);
    do {
        *(s1++ + x) = *(s2);
    } while (*s2++ != '\0');
}

int myStrspn(char *s1, char *s2) {
    char c;
    int i = 0, j;

    while ((c = s1[i]) != '\0') {
    /* Pour chaque élément de s1, */
        j = 0;
        while (s2[j] != '\0')
        /* On regarde s'il apparaît dans s2 */
            if (s2[j] == c)
                break; /* Si oui, on passe à l'élément suivant de s1 */
            else
                j++;
        if (s2[j] == '\0')
            break; /* On a fait tout s2 en vain, le préfixe s'arrête là, */
                   /* On quitte le while() sur s1. */
        i++;
    }
    return i;
}

int countChar(char *str, char c) {
    int n = 0;
    while (*str != '\0')
        if (*(str++) == c)
            n++;
    return n;
}
```

```c
int countMultChar(char *str, char *cc) {
    int i, n = 0;
    for (i = 0; i < strlen(cc); i++)
        n += countChar(str, cc[i]);
    return n;
}
```

III. Tableau de Chaînes

On souhaite transformer un tableau de chaînes de caractères (contenant éventuellement des doublons) en liste chaînée où chaque cellule indique la chaîne ainsi que son nombre d'occurrences dans le tableau. La figure ci-dessous donne un exemple de ce qu'il faut faire.



On pourra utiliser la fonction `int strcmp(char *s1, char *s2);` qui permet de comparer des chaînes de caractères et renvoie 1 si `s1`>`s2`, 0 si `s1`=`s2`, et −1 si `s1`<`s2`.

1. Définir une structure de cellule de liste chaînée `struct cellule {...}` permettant de stocker la chaîne, son nombre d'occurrences ainsi qu'un pointeur sur la cellule suivante.
2. Écrire une fonction

   ```c
   struct cellule *findStr(struct cellule *debut, char *str);
   ```

   qui parcourt la liste pointée par le pointeur `debut` (pouvant être nul) et cherche une chaîne égale à `str`. Si une cellule ayant cette chaîne est trouvée, la fonction renvoie un pointeur sur la cellule correspondante ; dans le cas contraire, la fonction renvoie `NULL`.
3. Écrire une fonction

   ```c
   struct cellule *tab2liste(char *T[]);
   ```

   qui crée une liste chaînée (et renvoie un pointeur sur son début) à partir du tableau de chaînes passé en paramètre. Chaque cellule doit indiquer la chaîne ainsi que son nombre d'occurrences dans le tableau. L'ordre des chaînes dans la liste n'a aucune importance.

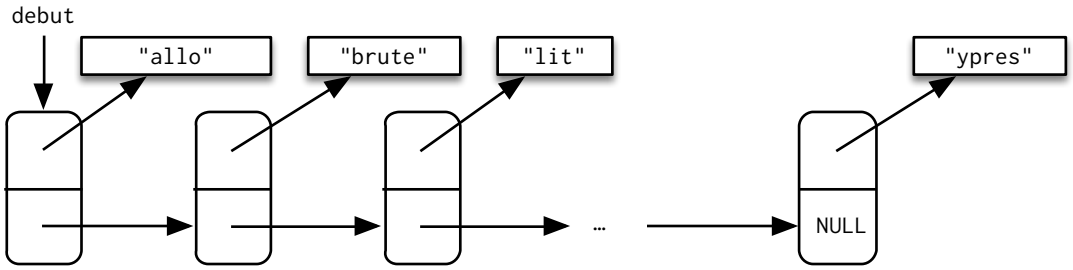
Corrigé :

Listing 2. ***Tableau vers liste***

```c
struct cellule {
    int decompte;
    char *str;
    struct cellule *next;
};

struct cellule *findStr(struct cellule *debut, char *str) {
    while (debut != NULL) {
        if (strcmp(str, debut->str) == 0)
            break;
        else
            debut = debut->next;
    }
    return debut;
}

struct cellule *tab2liste(char *T[]) {
    struct cellule *debut = NULL;
    struct cellule *p;
    int i = 0;

    while (T[i] != NULL) { /* On prend tous les éléments du tableau */
        if ((p = findStr(debut, T[i])) != NULL)
            p->decompte++; /* S'il est déjà dans la liste, on incrémente le compteur */
        else {
        /* Sinon on crée une nouvelle cellule et on l'insère en tête
         * (pour simplifier, l'ordre étant quelconque) */
            p = malloc(sizeof(struct cellule));
            p->decompte = 1;
            p->str = T[i];
            p->next = debut;
            debut = p;
        }
        i++;
    }
    return debut;
}
```

IV. Liste chaînée

On souhaite gérer une liste simplement chaînée de chaînes de caractères de telle sorte que la liste soit toujours triée.



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.

Dans toutes les fonctions suivantes, la liste peut être vide (et dans ce cas, le pointeur de tête `debut` passé en paramètre vaut `NULL`). Les fonctions pouvant modifier le pointeur de début de liste devront renvoyer la nouvelle valeur du pointeur.

1. Définir la structure correspondante `struct cellule {...}`.
2. Écrire une fonction `int compte(struct cellule *debut);` qui compte le nombre d'éléments de la liste.
3. Écrire une fonction `struct cellule *insere(struct cellule *debut, char *str);` qui insère une nouvelle chaîne au bon endroit dans la liste.
4. Écrire une fonction `struct cellule *supprime(struct cellule *debut, char *str);` qui supprime de la liste une chaîne passée en argument si elle s'y trouve.
5. Écrire une fonction `void clean(struct cellule *debut);` qui efface proprement la liste en supprimant tous ses éléments de la mémoire.

Corrigé :

Listing 3. ***liste chaînée de chaînes***

```c
struct cellule {
    char *chaine;
    struct cellule *next;
};

int compte(struct cellule *debut) {
    int total = 0;
    while (debut != NULL) {
        total++;
        debut = debut->next;
    }
    return total;
}
```

```c
struct cellule *insere(struct cellule *debut, char *str) {
    struct cellule *p = debut;
    struct cellule *new;
    /* Créons tout d'abord une nouvelle cellule */
    new = malloc(sizeof(struct cellule));
    new->chaine = strdup(str);
    new->next = NULL;
    if (p == NULL)
        debut = new; /* liste vide */
    else if (strcmp(str, p->chaine) < 0) {
        new->next = p;
        debut = new; /* ajout en tête */
    } else {
       /* Sinon on va à la bonne place (qui peut être la fin de la liste) */
        while (p->next && strcmp(str, p->next->chaine) > 0)
            p = p->next;
        new->next = p->next;
        p->next = new;
    }
    return debut;
}

struct cellule *supprime(struct cellule *debut, char *str) {
    struct cellule *prec = NULL, *p = debut;
    int val;
    /* Parcourons la liste */
    while (p != NULL) {
        val = strcmp(str, p->chaine);
        if (val < 0)
            break; /* on arrive à une chaîne plus grande */
        if (val == 0) { /* Trouvé ! */
            if (prec != NULL) /* On n'est pas au debut de la liste */
                prec->next = p->next;
            if (p == debut) /* Si on supprime la première cellule */
                debut = p->next;
            free(p->chaine); free(p);
            break;
        } else { /* cellule suivante */
            prec = p;
            p = p->next;
        }
    }
    return debut;
}

void clean(struct cellule *debut) {
    struct cellule *p;
    while ((p = debut) != NULL) {
        debut = debut->next;
        free(p->chaine); free(p);
    }
}
```