UNIVERSITÉ DE PARIS-DAUPHINE — DE MI2E 2e année

Programmation C (E. Lazard)
Examen du 23 janvier 2012

CORRECTION

(durée 2h)

I. Expressions

1. Donnez les résultats affichés par le programme suivant :

Listing 1. *Expressions*

```
int valeurs[3] = {5, 10, 11};

main() {
    int *p = valeurs;
    int **q = &p;
    printf("%d\n", +++(++(*q)));
    printf("%d\n", (*--(*q))++);
    printf("%d\n", (*(*q)++)++);
    printf("%d\n", (*(p+1))-*((*q)-- -1));
}
```

2. Le programme donne-t-il toujours les mêmes résultats avec tous les compilateurs ? Et si on change les valeurs du tableau ?

Corrigé :

1. – `++(*q)` incrémente `p` puis le renvoie ; on a donc un pointeur sur la deuxième case du tableau. Ce pointeur est déréférencé, la case est incrémentée (elle passe à 11) puis renvoyée.
   – `--(*q)` décrémente `p` puis le renvoie ; il repointe donc sur la première case. Ce pointeur est déréférencé, la valeur renvoyée (donc 5) puis la valeur est incrémentée et passe à 6.
   – on renvoie `p` puis on le décale sur la deuxième case. La valeur renvoyée (pointeur sur la première case) est déréférencé, la valeur renvoyée (donc 6) puis la valeur est incrémentée et passe à 7.
   – La valeur de la première opérande est 11 (troisième case). Pour la deuxième, on récupère `p` (deuxième case, puis la première) et on retranche un, donc on récupère la valeur de la première case (7). $11 - 7$ affiche 4.

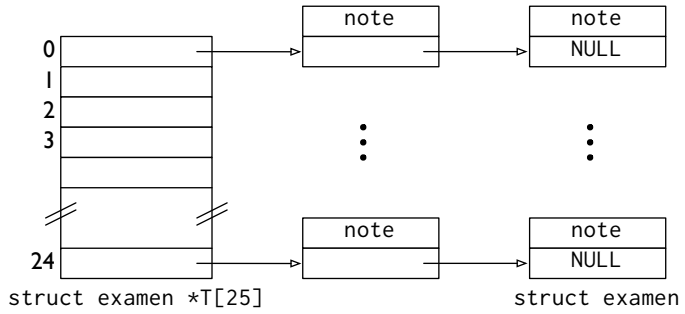
   Ce programme affiche donc :

```
11
 5
 6
 4
```

2. Le problème provient de l'évaluation de la dernière expression. Si `*(p+1)` est évalué en premier, cela sera toujours la valeur de la troisième case. En revanche, si `(*q)--` intervient d'abord, le pointeur `p` est décrémenté et `*(p+1)` pointera à ce moment-là sur la deuxième case. Avec les valeurs du tableau, cela fera $11 - 7$ dans les deux cas mais pas si on change les valeurs du tableau. Ce problème est dû à l'indétermination de l'ordre d'évaluation des opérandes d'une expression.

II. Listes chaînées

On souhaite gérer les résultats aux examens d'une classe de 25 élèves. Ces 25 élèves portent tous un numéro (de 0 à 24) et pour chacun, on mémorise ses résultats dans une liste chaînée d'entiers.




```
struct examen {
    int note;
    struct examen *next;
};

struct examen *T[25];
```

Chaque case du tableau `T[]` contient un pointeur sur la première cellule (contenant la note du premier examen) pour chaque élève. On pourra supposer que chaque élément de ce tableau est initialisé à `NULL` avant son utilisation. Chaque examen est noté par un entier entre 0 et 20, ou par la note -1 si l'élève est absent. On est donc sûr que toutes les listes chaînées sont de la même taille (autant de notes que d'examens).

1. Écrivez deux fonctions

   ```
   int maxEleve(int eleve);

   float moyEleve(int eleve);
   ```

   qui renvoient respectivement la note maximum et la moyenne des notes calculées sur l'ensemble des examens passés par l'élève (dont le numéro entre 0 et 24 est passé en argument). S'il n'y a aucun examen ou si l'élève n'en a passé aucun, elles renverront -1.
2. Écrivez une fonction

   ```
   int maxExamen(int numero);
   ```

   qui renvoie la note maximum attribuée lors d'un examen dont le numéro est passé en argument. Le premier examen a pour numéro 0 et ainsi de suite. Si aucun élève n'était présent à l'examen, la fonction renvoie -1. Pour simplifier, on pourra supposer que le numéro passé en argument est toujours correct, c'est-à-dire qu'il fait référence à un examen existant dans la liste.

Corrigé :

Listing 2. ***Listes chaînées***

```
struct examen {
    int note;
    struct examen *next;
};

struct examen *T[25];

int maxEleve(int eleve) {
    int max = -1;
    struct examen *p = T[eleve];
    while (p != NULL) {
        if (p->note > max)
            max = p->note;
        p = p->next;
    }
    return max;
}

float moyEleve(int eleve) {
    int somme = 0;
    int nbr = 0;
```

```
    struct examen *p = T[eleve];
    while (p != NULL) {
        if (p->note != -1) {
            somme += p->note;
            nbr++;
        }
        p = p->next;
    }
    if (nbr == 0)
        return -1;
    else
        return ((float) somme)/nbr;
}

int maxExamen(int numero) {
    int eleve, compteur;
    int max = -1;
    struct examen *p;
    for (eleve = 0; eleve < 25; eleve++) {
        p = T[eleve];
        compteur = numero;
        while ((compteur--) > 0)
            p = p->next;
        if (p->note > max)
            max = p->note;
    }
    return max;
}
```

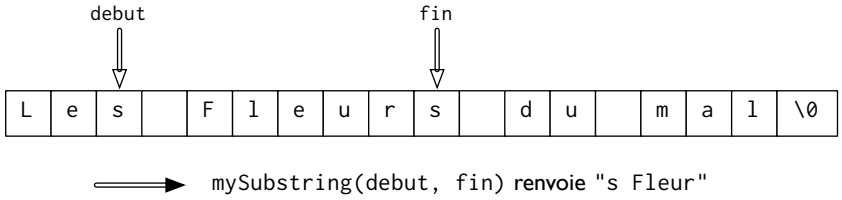
III. Chaîne de caractères

Le but de cet exercice est de découper une chaîne de caractères suivant les mots. On ne pourra pas utiliser de fonctions déclarées dans `string.h`*, à l'exception de* `strlen()` *si nécessaire.*

1. Écrivez une fonction

   `char *mySubstring(char *debut, char *fin);`

   qui extrait une sous-chaîne d'une chaîne à partir de deux pointeurs `debut` et `fin`. Le premier pointeur, `debut`, pointe sur le premier caractère à extraire, le second pointeur, `fin`, pointe sur le premier caractère à ne pas extraire.

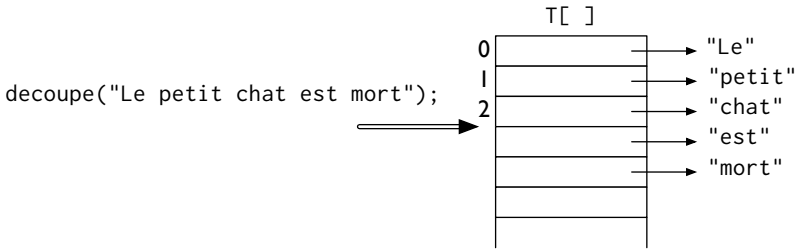


2. *On supposera que l'on dispose en variable globale d'un tableau de pointeurs de caractères,* `char *T[]` *que l'on utilisera pour stocker les pointeurs sur les mots extraits ; on supposera également que ce tableau est toujours assez grand pour tout stocker.*

   Écrivez une fonction

   `int decoupe(char *str);`

   qui reçoit une chaîne en argument et remplit le tableau `T[]` avec une copie de chaque mot. La fonction renvoie le nombre de mots stockés.



On supposera qu'il y a exactement un caractère espace entre chaque mot, et qu'il n'y en a ni en tête de la chaîne, ni en queue.

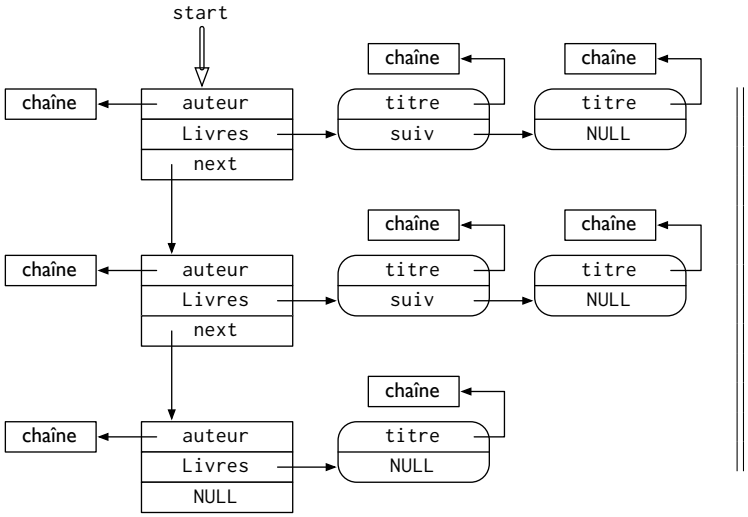
Corrigé :

Listing 3. ***Chaînes***

```
char *T[10];

char *mySubstring(char *debut, char *fin){
    char *str, *temp;
    int longueur = fin - debut + 1; /* ne pas oublier la place pour le \0 final */

    if (!fin || !debut || (longueur < 1))
        return NULL; /* mauvais paramètres */
    temp = str = malloc(longueur * sizeof(char));
    while (debut != fin)
        *(temp++) = *(debut++);
    *temp = '\0';
    return str;
}


int decoupe(char *str) {
    char *debut = str;
    int comp = 0;

    while (*str != '\0') {
        while ((*str != ' ') && (*str != '\0'))
            str++;
        T[comp++] = mySubstring(debut, str);
        if (*str == ' ') {
            str++;
            debut = str;
        }
    }
    return comp;
}
```

IV. Double liste chaînée

On souhaite gérer le catalogue d'une librairie dans lequel chaque livre est identifié par son auteur et son titre. La structure de données choisie est la suivante : nous avons une liste chaînée d'auteurs dont chaque élément pointe vers une liste chaînée d'ouvrages.



Les définitions sont les suivantes :

```
typedef struct cellAuteur {
    char *auteur;
    struct cellLivre *Livres;
    struct cellAuteur *next;
} cellAuteur;

typedef struct cellLivre {
    char *titre;
    struct cellLivre *suiv;
} cellLivre;

 /* la première cellule */
 cellAuteur *start;
```

Dans toutes les fonctions à écrire, on supposera :

- que la liste d'auteurs peut être vide (`start` peut valoir `NULL` – il faudra donc le modifier si nécessaire) ;
- que tous les paramètres sont corrects ; autrement dit, un argument pointeur supposé être une chaîne ne sera pas nul ;
- qu'il n'existe pas d'auteur sans livre dans le catalogue ;
- qu'aucune des listes n'a besoin d'être 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 *s);` permet de dupliquer une chaîne de caractères passée en argument en allouant dynamiquement la place mémoire nécessaire.

1. Écrivez une fonction

   ```
   cellAuteur *existe(char *strAuteur);
   ```

   qui teste si un auteur existe dans la liste et dans ce cas renvoie un pointeur sur sa cellule de la liste (et `NULL` sinon).
2. Écrivez une fonction

   ```
   int compteOuvrage(char *strAuteur);
   ```

   qui compte le nombre de livres d'un auteur (renvoie 0 si l'auteur n'existe pas).
3. Écrivez une fonction

   ```
   void add(char *strAuteur, char *strTitre);
   ```

   qui ajoute dans le catalogue un livre de l'auteur indiqué.
4. Écrivez une fonction

   ```
   void supprimer(char *strAuteur);
   ```

   qui supprime un auteur et tous ses livres du catalogue.

Corrigé :

Listing 4. ***Catalogue de livres***

```
cellAuteur *existe(char *strAuteur){
    cellAuteur *p = start;
    while ((p != NULL) && (strcmp(p->auteur, strAuteur) != 0))
        p = p->next;
    return p;
}

int compteOuvrage(char *strAuteur){
    int comp = 0;
    cellLivre *q;
    cellAuteur *p = existe(strAuteur);
    if (p == NULL)
        return 0;
    q = p->Livres;
    while (q != NULL) {
        comp++;
        q = q->suiv;
    }
    return comp;
}

void add(char *strAuteur, char *strTitre){
    cellLivre *q = malloc(sizeof(cellLivre));
    cellAuteur *p = existe(strAuteur);
    q->titre = strdup(strTitre);
    if (p != NULL) {  /* liste pas triée, on insère en tête */
        q->suiv = p->Livres;
        p->Livres = q;
    } else {  /* liste pas triée, on insère en tête */
        p = malloc(sizeof(cellAuteur));
        p->auteur = strdup(strAuteur);
        p->next = start;
        start = p;
        p->Livres = q;
        q->suiv = NULL;
    }
}

void supprimer(char *strAuteur){
    cellLivre *q, *n;
    cellAuteur *pp, *p = existe(strAuteur);
    if (p != NULL) {
        q = p->Livres;
        while ((n = q) != NULL) {
            q = q->suiv;
            free(n->titre);
            free(n);
        }
        free(p->auteur);
        if (start == p)
            start = p->next;
        else {
            pp = start;
            while (pp->next != p)
                pp = pp->next;
            pp->next = p->next;
        }
        free(p);
    }
}
```