Programmation C (E. Lazard)
Examen du 28 janvier 2009

CORRECTION

(durée 2h)

I. Déclarations

Indiquez si les déclarations suivantes sont correctes et si c'est le cas, donnez leur signification.

```
int [] t ;
char str[5] = "hello" ;
float *p ;
int **i ;
```

Corrigé :

- `int [] t ; // FAUX`
- `char str[5] = "hello" ; // la déclaration est correcte mais il y aura un problème car il manque la place pour le 0 final`
- `float *p ; // pointeur sur un flottant`
- `int **i ; // pointeur sur pointeur sur entier`

II. Chaîne

Écrivez une fonction `int myatoi(char *)` qui transforme une chaîne de caractères en sa valeur numérique et la renvoie. Celle-ci est la valeur du groupe de chiffres débutant la chaîne, avec un éventuel signe '-' ou '+' comme premier caractère. Si la chaîne ne débute ni par -, ni par +, ni par un chiffre, la fonction renvoie zéro. On supposera que la valeur numérique tient toujours dans un entier. On pourra utiliser la fonction `int isdigit(char)` qui renvoie 1 si le paramètre est un caractère numérique (entre '0' et '9').

Exemples : `"23ab"` renvoie la valeur 23, `"-12x"` renvoie -12, `"-a54b"`, `"0y("` et `"xx6"` renvoient tous 0.

Corrigé :

```
int myatoi(char *str) {
    int val = 0 ;
    int index = 0 ;
    int positif = 1 ;
    char c ;
    if (str[0] == '-') {
        positif = -1 ;
        index ++ ;
    } else if (str[0] == '+')
        index++ ;
    while ( isdigit( c=str[index++] ) )
        val = 10*val+c-'0' ;
    return val*positif ;
}
```

III. Tableaux

Écrivez un programme qui transfert un tableau d'entiers T à deux dimensions L et C dans un tableau X à une dimension $L \times C$. Le programme aura la structure suivante :

- définition des deux dimensions L et C dans un `#define` ;
- boucle de saisie clavier des valeurs de T ;
- construction *dynamique* (par un `malloc`) du tableau X ;
- Recopie des valeurs de T vers X, ligne par ligne ;
- affichage du tableau X.

Corrigé :

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

#define L 3
#define C 4

void main() {
    int T[L][C];
    int *X;
    int i,j;

    /* saisie des valeurs de T */
    for (i=0 ; i<L ; i++)
        for (j=0 ; j<C ; j++) {
                printf("valeur [%d,%d] =", i, j);
                scanf("%d", &T[i][j]) ;
        }

    /* allocation  de X */
    X = (int *) malloc(L*C*sizeof(int));
    for (i=0 ; i<L ; i++)
        for (j=0 ; j<C ; j++)
            X[i*C+j] = T[i][j];

    /* affichage de X */
    for (i=0 ; i<L*C ; i++)
        printf("%d ", X[i]) ;
}
```

IV. Pile

On souhaite développer une pile de chaînes de caractères (plus précisément de pointeurs `char *`) sous la forme d'une liste simplement chaînée. On accédera à la pile à l'aide d'un pointeur sur la première cellule vide à son sommet.

- Définissez la structure d'un élément de la pile `struct cell`.

– Écrivez le code des fonctions suivantes :
  – `struct cell *creer(void) ;` : crée une pile vide et renvoie un pointeur sur la première cellule vide au sommet.
  – `int estVide(struct cell *);` ; teste si la pile dont on indique le sommet est vide (dans ce cas renvoie 1, sinon 0). Une pile vide signifie qu'il n'y a pas d'élément empilé.
  – `void push(struct cell **, char *) ;` : empile le pointeur de chaîne indiqué (second paramètre - en le dupliquant avec `strdup()`) en modifiant le pointeur indiquant le sommet de la pile (premier paramètre).
  – `char *pop(struct cell **) ;` : dépile le pointeur de chaîne au sommet (et le renvoie) et modifie le pointeur indiquant le sommet de la pile (paramètre).

Corrigé :

```
struct cell {
    char * str ;
    struct cell * next ;
} ;

struct cell *creer(void) {
    struct cell *p = (struct cell *) malloc(sizeof(struct cell)) ;
    p->str = NULL ;
    p->next = NULL ;
    return p ;
}

int estVide(struct cell *sommet) {
    if ( (sommet == NULL) || (sommet->next == NULL) )
        return 1 ;
    else
        return 0 ;
}

void push(struct cell **p, char *chaine) {
    if ( (p==NULL) || (*p==NULL))
        return ;
    (*p)->str = strdup(chaine) ;
    struct cell *nouv = (struct cell *) malloc(sizeof(struct cell)) ;
    nouv->str = NULL ;
    nouv->next = *p ;
    *p = nouv ;
}

char *pop(struct cell **p) {
    char *chaine ;
    struct cell* last ;
    if ( (p==NULL) || (*p==NULL))
        return ;
    last = (*p)->next ;
```

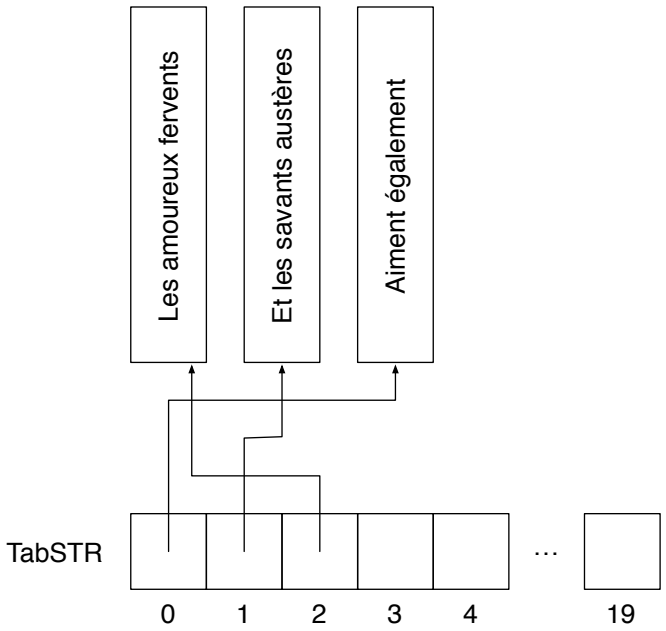
```
    if (last == NULL)
        return NULL ;
    chaine = last->str ;
    free(*p) ;
    *p = last ;
    return chaine ;
}
```

V. Tableau de chaînes

On souhaite saisir à partir du clavier un ensemble de chaînes de caractères dans un tableau de pointeurs. L'ensemble se termine avec la lecture de la chaîne « `The End` » (qui n'est pas insérée dans le tableau). Le tableau doit rester constamment trié en ordre croissant (ordre alphabétique).

Ainsi avec le tableau de la figure ci-dessous, la lecture de la chaîne « `Aiment également` » entraîne le décalage des pointeurs du tableau à partir du début puisque la chaîne est alphabétiquement inférieure à « `Et les savants austères` ». Notez que les chaînes elles-mêmes ne se déplacent pas.

La taille du tableau de pointeurs est fixe et est égale à 20.



Le programme principal sera le suivant :

```
void main() {
    char * TabSTR[20] ;  /* tableau de pointeurs */
    int nbr_str ;        /* recupère le nombre de chaînes */

    /* lecture lit au clavier les chaînes et les met dans le tableau à leur place */
    nbr_str = lecture(TabSTR, 20) ;

    /* affiche visualise à l'écran le tableau */
    affiche(TabSTR, nbr_str) ;
}
```

Écrivez les fonctions `int lecture(char *[], int)` (qui renvoie le nombre de chaînes vraiment lues avec un maximum donné par le second paramètre) et `void affiche(char *[], int)` (qui affiche toutes les chaînes – le nombre est donné par le second paramètre – dans l'ordre du tableau). On

supposera qu'une chaîne est de taille maximum 99 caractères. On pourra comparer les chaînes avec la fonction `int strcmp(char *s1, char *s2)` qui renvoie 1 si $s1 > s2$, 0 si $s1 = s2$ et -1 si $s1 < s2$.

CORRIGÉ :

```
/* Fonction de lecture d'un tableau de chaînes de caractères          */
/* Entrée : char *tab[] : tableau de chaînes de caractères            */
/*          int nbMax : nombre de chaînes maximum du tableau          */
/* Sortie : int : nombre de chaînes lues et insérées dans le tableau  */
int lecture(char* tab[],int nbMax) {
    int nbStr=0, pos, index ;

   /* Déclaration d'une chaîne à saisir */
   char buffer[100] ;

   /*
    * Tant qu'il y a moins de nbMax chaînes lues
    * et que la chaîne "The End" n'a pas été saisie
    */
   while(nbStr<nbMax && strcmp(gets(buffer),"The End")) {
       /* On recherche la position de la chaîne saisie dans le tableau */
       /* en comparant la chaîne saisie et celles du tableau           */
       pos = 0 ;
       while ( (pos<nbStr) && (strcmp(tab[pos], buffer)<0) )
           pos++ ;

       /* On décale les chaînes du tableau (en partant de la fin)    */
       /* pour y insérer la nouvelle chaîne à la position trouvée    */
       for(index=nbStr; index>pos; index--)
           tab[index] = tab[index-1] ; /* déplacer l'élément d'un cran */

       /* Allocation mémoire pour insérer la nouvelle chaîne */
       tab[pos] = (char *) malloc(strlen(buffer)+1) ;
       strcpy(tab[pos],buffer) ; /* copie de la chaîne dans le tableau */
       nbStr++ ;
   }
   return nbStr ;
}

/* Fonction d'affichage d'un nombre fixe de chaînes d'un tableau */
/* Entrée : char *tab[] : tableau de chaînes de caractères       */
/*                        int nbr : nombre de chaînes à afficher  */
void affiche(char *tab[], int nbr) {
    int i ;

    for(i=0; i<nbr; i++)
        puts(tab[i]);
}
```