Programmation C (E. Lazard)
Examen du 1er février 2008

CORRECTION

(durée 2h)

I. Expressions

Donnez les résultats affichés par le programme suivant :

```
int f1(int i)  { return i+1;                    }
int f2(int i)  { return i++;                    }
int f3(int i)  { printf("%d\n",i==0); return i; }
int f4(int i)  { printf("%d\n",i=0);  return i; }
int f5(int *i) { return ++(*i);                 }
int f6(int *i) { (*i)++;          return (*i)++; }
int main() {
        int a, b;
        a=f1(0);  b=f2(1);  printf("a=%d, b=%d\n",a,b);
        a=f3(a);  b=f4(a);  printf("a=%d, b=%d\n",a,b);
        a=f5(&a); b=f6(&a); printf("a=%d, b=%d\n",a,b);
}
```

Corrigé :

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

Ce programme affiche donc :

```
a=1, b=1
0
0
a=1, b=0
a=4, b=3
```

II. Compilation

Le programme suivant (qui recopie et affiche une chaîne) compile sans erreur. Cependant, il ne s'exécute pas correctement. Quel(s) est(sont) le(s) problème(s) ? Corrigez le programme en ajoutant ce qui est nécessaire et en modifiant **au minimum** ce qui est écrit.

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

void main() {
    short index ;
```

```
    char texte[] = "Programmation\nC\n";
    char *Ptr ;
    while (texte[index++]) {
        *(Ptr+index) = texte[index] ;
    }
    printf("%s", Ptr) ;
}
```

Corrigé :

- le pointeur `Ptr` n'est pas initialisé à une adresse mémoire ;
- il faut donc ajouter "`#include <stdlib.h>`" pour le `malloc()` ;
- l'index n'est pas initialisé à 0 ;
- l'incrémentation de l'index ne doit pas se faire dans le `while` mais après l'affectation ;
- il faut ajouter un `'\0'` final à la chaîne copiée.

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

void main() {
    short index = 0;
    char texte[] = "Programmation\nC\n";
    char *Ptr ;
    Ptr = malloc( strlen(texte)+1 ) ;
    while (texte[index]) {
        *(Ptr+index) = texte[index] ;
        index++ ;
    }
    Ptr[index] = '\0' ;
    printf("%s", Ptr) ;
}
```

III. Chaînes de caractères et tri

On désire implémenter un algorithme de tri par sélection sur un tableau de 10 chaînes de caractères rentrées par l'utilisateur.

L'algorithme général de tri est le suivant :

- on parcourt le tableau des éléments 0 à $N-1$ (N vaut donc ici 10) en mémorisant le plus petit élément. Cet élément est ensuite échangé avec l'élément 0 ;
- on recommence en parcourant le tableau des éléments 1 à $N-1$. Le plus petit de ces éléments est échangé avec élément 1 ;
- on procède de la même façon pour tous les indices restants. L'algorithme est donc construit comme deux boucles imbriquées l'une dans l'autre : boucle externe `i` de 0 à $N-1$ et boucle interne `j` de `i` à $N-1$.

1. Écrivez une fonction `char *lire(void);` de saisie d'une chaîne de caractères tapée par l'utilisateur. Cette fonction doit lire au clavier une chaîne qu'on supposera de taille maximum 100 caractères et renvoie son adresse.

2. Écrivez une fonction `void echanger(...);` qui échange deux chaînes de caractères en paramètre (sans les recopier, c'est-à-dire en permutant la valeur des pointeurs correspondants).
3. Complétez le squelette du programme suivant :

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

main() {
    char *tab[10] /* tableau de 10 pointeurs sur une chaîne */
```

   - Lire 10 chaînes à l'aide de `lire()` et stocker leur adresse dans `tab`
   - Écrire l'algorithme de tri qui classe les chaînes dans le tableau de la plus petite à la plus grande
     - les permutation se feront avec `echanger()`
     - 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 $s1 < s2$.
   - Afficher les 10 chaînes dans l'ordre du tableau.

```
}
```

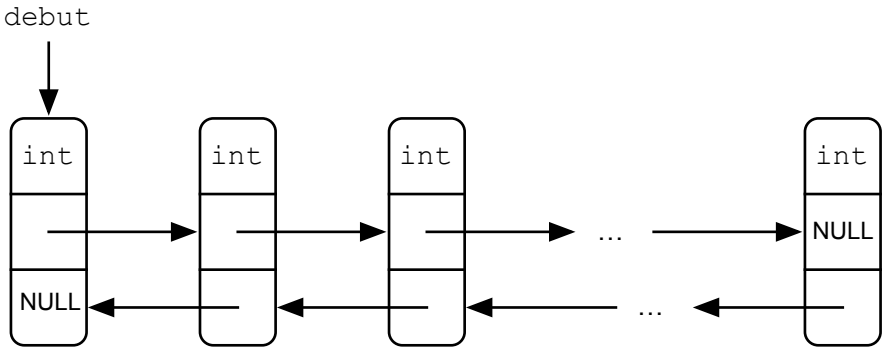
Corrigé :

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

char *lire(void) {
    char *p = malloc(100) ;
    gets(p) ;
    return p ;
}

void echanger(char **p1, char **p2) {
    char *tmp ;
    tmp = *p1 ;
    *p1 = *p2 ;
    *p2 = tmp ;
}

main() {
    char *tab[10] ;
    int i, j, min ;
    for (i=0 ; i<10 ; i++)
        tab[i] = lire() ;
    for (i=0 ; i<10 ; i++) {
        min = i ;
        for (j=i+1 ; j<10 ; j++)
            if ( strcmp(tab[min], tab[j]) > 0 )
                min = j ;
        echanger(&tab[i], &tab[min]) ;
    }
```

```
    for (i=0 ; i<10 ; i++)
        puts(tab[i]) ;
}
```

IV. Liste chaînée

On souhaite gérer une liste doublement chaînée où chaque cellule a un `int` comme valeur (`debut` pointe vers la tête de la liste).



Dans toutes les fonctions suivantes, la liste peut être vide (et dans ce cas, le pointeur de tête `debut` vaut `NULL`).

1. Définissez la structure correspondante `struct cellule`.
2. Écrivez une fonction qui insère une valeur entière dans une nouvelle cellule en tête de liste. Cette fonction prend comme argument un pointeur sur la tête de la liste ainsi que la valeur entière et renvoie un pointeur sur la tête de la liste.
3. Écrivez une fonction qui efface proprement la liste en supprimant tous ses éléments de la mémoire. Cette fonction prend comme argument un pointeur sur la tête de la liste et renvoie un pointeur sur la tête de la liste.
4. Écrivez une fonction qui échange deux éléments successifs de la liste en permutant la place des deux cellules dans la liste (et pas uniquement en échangeant la valeur de l'entier). Cette fonction prend comme argument un pointeur sur la tête de la liste, un pointeur sur chacun des deux éléments à échanger et renvoie un pointeur sur la tête de la liste. Pour cette question, la liste ne peut pas être vide, au moins les deux éléments à permuter existent.
5. Écrivez une fonction qui supprime le plus grand élément (valeur entière la plus grande) de la liste. Cette fonction prend comme argument un pointeur sur la tête de la liste et renvoie un pointeur sur la tête de la liste. La liste n'est pas ordonnée mais toutes les valeurs entières sont distinctes.

Corrigé :

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

struct cellule {
    int val ;
    struct cellule *next ;
    struct cellule *pred ;
} ;

struct cellule *InsereTete(struct cellule *Debut, int n) {
    struct cellule *p = malloc(sizeof(struct cellule)) ;
```

```
    p->val = n ;
    p->next = Debut ;
    p->pred = NULL ;
    if (Debut)
        Debut->pred = p ;
    return p ;
}

/* Il faut penser a : */
/* tester pour la liste vide, bien effacer la derniere cellule, renvoyer NULL */
struct cellule *SupprimeTout(struct cellule *Debut) {
    struct cellule *p ;
    while (p=Debut) {
        Debut = Debut->next ;
        free(p) ;
    }
    return NULL ;
}

/* Il faut penser a : */
/* verifier s'il y a une cellule avant p1 ou apres p2, changer Debut si c'etait p1 */
struct cellule *EchangeSucc(
            struct cellule *Debut, struct cellule *p1, struct cellule *p2) {
    struct cellule *avant = p1->pred ;
    struct cellule *apres = p2->next ;
    p1->pred = p2 ;
    p2->next = p1 ;
    if (avant)
        avant->next = p2 ;
    p2->pred = avant ;
    if (apres)
        apres->pred = p1 ;
    p1->next = apres ;
    if (Debut == p1) /* p1 etait la tete ? */
        return p2 ;  /* a changer */
    else return Debut ;
}

/* Il faut penser a : */
/* tester pour la liste vide, bien parcours aussi la derniere cellule, */
/* verifier s'il y a une cellule avant ou apres le max, changer Debut si egal max */
/* Attention a l'initialisation de l'entier qui n'est pas forcement positif */
/* struct cellule *SuppMax(struct cellule *Debut) {
    struct cellule *PtrMax, *p ;
    /* PtrMax va pointer sur le plus grand, */
    /* p parcourt la liste */
    if (!Debut)
        return NULL;
    PtrMax = p = Debut ;
    do {
```

```
        if (p->val > PtrMax->val)
            PtrMax = p ;
        p = p->next ;
    } while (p) ;
    if (PtrMax->next)
        PtrMax->next->pred = PtrMax->pred ;
    if (PtrMax->pred)
        PtrMax->pred->next = PtrMax->next ;
    if (Debut == PtrMax)
        Debut = PtrMax->next ;
    free(PtrMax) ;
    return Debut ;
}
```

On peut généraliser l'échange de deux cellules à deux éléments quelconques de la liste :

```
struct cellule *EchangeGeneral(
            struct cellule *Debut, struct cellule *p1, struct cellule *p2) {
    struct cellule *avant_p1 = p1->pred ;
    struct cellule *avant_p2 = p2->pred ;
    struct cellule *apres_p1 = p1->next ;
    struct cellule *apres_p2 = p2->next ;
    p1->pred = avant_p2 ;
    p1->next = apres_p2 ;
    p2->pred = avant_p1 ;
    p2->next = apres_p1 ;
    if (avant_p1)
        avant_p1->next = p2 ;
    if (apres_p1)
        apres_p1->pred = p2 ;
    if (avant_p2)
        avant_p2->next = p1 ;
    if (apres_p2)
        apres_p2->pred = p1 ;
    if (Debut == p1) /* p1 etait la tete ? */
        return p2 ;  /* a changer */
    else if (Debut == p2) /* p2 etait la tete ? */
        return p1 ;  /* a changer */
    else return Debut ;
}
```