

TD6 – Manipulations de pointeurs

Langage C (LC4)

Semaine du 12 mars 2012

1 Un peu d'arithmétique des pointeurs

Question 1. On considère les déclarations suivantes.

```
struct st3 { int a; int b; int c; };
int t[30];
int *p = t;
char *s = (char *) t;
struct st3 *pst3 = (struct st3 *) t;
```

Indiquez les valeurs des expressions ci-dessous après l'exécution de la boucle suivante.

```
int i;
for (i = 0; i < 30; i++)
t[i] = 10 * i;
```

On supposera qu'on travaille sur une machine où la taille des `int` et des `char` valent respectivement 4 octets et 1 octet.

1. `*p + 2`
2. `*(p + 2)`
3. `&p + 1`
4. `&t[4] - 3`
5. `t + 3`
6. `&t[7] - p`
7. `p + (*p - 10)`
8. `*(p + *(p + 8) - t[7])`
9. `s[4]`
10. `&(s[4]) - &(s[2])`
11. `pst3[3].b`
12. `((struct st3 *)&t[6]) - pst3`

► Exercice 1

```
*p + 2 == 2
*(p + 2) == 20
&t[4] - 3 == &t[1]
t + 3 == &t[3]
&t[7] - p == 7
p + (*p - 10) == &t[-10]
*(p + *(p + 8) - t[7]) == 100
s[4] == 10
&(s[4]) - &(s[2]) == 2
pst3[3].b == 100
((struct st3 *) &t[6]) - pst3 == 2
```

2 Listes doublement chaînées

On veut implémenter des listes doublement chaînées, où chaque élément pointe vers son prédécesseur et son successeur. On peut utiliser le type suivant :

```
typedef struct cellule {
    struct cellule *precedent; /* un pointeur vers la cellule precedente */
    struct cellule *suivant;   /* un pointeur vers la cellule suivante */
    int contenu;
} *liste_circulaire;
```

Question 2. Écrire une fonction `int est_bien_circulaire(liste_circulaire l)` qui teste si une telle liste est bien doublement chaînée. **Exercice 2**

```
int est_bien_chainee(liste_circulaire l) {
    struct cellule *ptr = l;
    if (l) {
        while(ptr->suivant != l) {
            if (!ptr->suivant)
                return 0;
            if (ptr->suivant->precedent != ptr)
                return 0;
            ptr = ptr->suivant;
        }
        return(l->precedent == ptr);
    }
    else
        return 1;
}
```

Question 3. Écrire une fonction `void affiche_liste(liste_circulaire l)` qui affiche le contenu d'une liste doublement chaînée sur la sortie standard.

► **Exercice 3**

```
void affiche_liste(liste_circulaire l) {
    struct cellule *ptr = l;
    if (l) {
        printf("[%d", l->contenu);
        ptr = l->suivant;
        while (ptr != l) {
            printf(", %d", ptr->contenu);
            ptr = ptr->suivant;
        }
        printf("]\n");
    }
    else
        printf("[]");
}
```

Question 4. Écrire une fonction `liste_circulaire insere(liste_circulaire l, int x)` qui insère (une cellule contenant) l'entier `x` entre les cellules pointées par `l` et `l->suivant` si `l` n'est pas `NULL`, et crée une liste contenant juste `x` dans le cas contraire. Elle doit renvoyer en résultat un pointeur vers le nœud de la liste créé pour contenir `x`.

► **Exercice 4**

```
liste_circulaire insere(liste_circulaire l, int x) {
    struct cellule *c = malloc(sizeof(struct cellule));
    c->contenu = x;
    if (l) {
        c->suivant = l->suivant;
```

```

        l->suivant->precedent = c;
        c->precedent = l;
        l->suivant = c;
    } else {
        c->precedent = NULL;
        c->suivant = NULL;
    }
    return c;
}

```

Question 5. Écrire une fonction `liste_circulaire supprime(liste_circulaire l)` qui efface l'élément pointé par `l` de la liste à laquelle il appartient. Elle doit renvoyer `NULL` si la liste est vide après l'effacement de `l`, et `l->precedent` sinon.

► **Exercice 5**

```

liste_circulaire supprime(liste_circulaire l) {
    if (l->suivant == l) {
        free(l);
        return NULL;
    } else {
        struct liste_circulaire *c = l->precedent;
        c->suivant = l->suivant;
        l->suivant->precedent = c;
        free(l);
        return c;
    }
}

```

Question 6. Écrire une fonction `int compte(liste_circulaire l)` qui compte le nombre d'éléments dans la liste pointée par `l`.

► **Exercice 6**

```

int compte(liste_circulaire l) {
    struct cellule c = l;
    int res = 0;
    if (!l) return res;
    do {
        res++;
        c = c->suivant;
    } while (c != l);
    return res;
}

```

Question 7. Écrire une fonction `void inverse(liste_circulaire l)` qui inverse l'ordre des éléments de la liste `l` (sans toucher aux champs `contenu`).

► **Exercice 7**

```

void inverse(struct liste_circulaire l) {
    liste_circulaire *p = l;
    liste_circulaire *q = l;
    do {
        q = p->suivant;
        p->suivant = p->precedent;
        p->precedent = q;
        p = q;
    } while (p != l);
}

```