TP 7 : Révisions, listes chaînées

Programmation en C (LC4)

Semaine du 12 mars 2007

▶ **Exercice 1**

```
char *lit_chaine(void) {
    int taille = 0, capacite = 8;
    int c;
    char *s = malloc(capacite);
    while ((c = getchar()) != '\n') {
        if (taille == capacite - 1) {
            capacite *= 2;
            s = realloc(s, capacite);
        }
        s[taille] = c;
        taille++;
    }
    s[taille] = '\0';
    return s;
}

char *inverse_chaine(char *s) {
    char *t;
    int i, n = strlen(s);
    t = malloc(n + 1);
    for (i = 0; i < n; i++)
        t[i] = s[n - 1 - i];
    t[n]= '\0';
    return t;
}

int est_palindrome(char *s) {
    char *t, *u = inverse_chaine(s);
    t = u;
    while (*s++ == *u++) {
    }
    free(t);
    return (*s == '\0');
}

int est_palindrome2(char *s) {
    int cmp;
    char *t = inverse_chaine(s);
    cmp = strcmp(s, t);
    free(t);
    return !cmp;
}
```

▶ Exercice 2

```
typedef struct personne {
    char *nom;
    char *prenom;
} personne;

void change_chaine(char *s) {
    if (*s == '\0')
        return;
    *s = toupper(*s);
    s++;
    while (*s) {
        *s = tolower(*s);
        s++;
    }
}

void change_personne(personne p) {
    change_chaine(p.nom);
    change_chaine(p.prenom);
}

int compare_personnes(personne p1, personne p2) {
    int cmp = strcmp(p1.nom, p2.nom);
    if (cmp == 0)
        cmp = strcmp(p1.prenom, p2.prenom);
    return cmp;
}

void echange_personnes(personne *p1, personne *p2) {
    personne t = *p1;
    *p1 = *p2;
    *p2 = t;
}

void tri_personnes(int taille, personne *personnes) { /* tri bulle */
    int i, j;
    for (i = taille - 2; i >= 0; i--)
        for (j = 0; j <= i; j++)
            if (compare_personnes(personnes[j], personnes[j + 1]) > 0)
                echange_personnes(&personnes[j], &personnes[j + 1]);
}
```

▶ Exercice 3

```
typedef struct monome {
    int degre;
    double coefficient;
    struct monome *suivant;
} *monome;

typedef monome polynome;

void affiche_polynome(polynome p) {
    while (p != NULL) {
```

```
        if (p->degre > 1) {
            printf("%gX^%d", p->coefficient, p->degre);
        } else {
            if (p->degre == 1)
                printf("%gX", p->coefficient);
            else
                printf("%g", p->coefficient);
        }
        p = p->suivant;
        if (p != NULL)
            printf("+");
    }
    printf("\n");
}

monome cree_monome(int degre, double coefficient) {
    monome m = malloc(sizeof(struct monome));
    m->degre = degre;
    m->coefficient = coefficient;
    m->suivant = NULL;
    return m;
}

void detruit_polynome(polynome p) {
    monome m;
    while (p != NULL) {
        m = p->suivant;
        free(p);
        p = m;
    }
}

polynome convertit_polynome(int degre, double *coefficients) {
    int i;
    polynome p = NULL, *q = &p;
    monome m;
    for (i = degre; i >= 0; i--)
        if (coefficients[i] != 0.0) {
            m = cree_monome(i, coefficients[i]);
            *q = m;
            q = &m->suivant;
        }
    return p;
}

polynome derive_polynome(polynome p)
{
    polynome q = NULL, *r = &q;
    monome m;
    while (p != NULL && p->degre != 0) {
        m = cree_monome(p->degre - 1, p->degre * p->coefficient);
        *r = m;
        r = &m->suivant;
        p = p->suivant;
    }
    return q;
```

```
}

polynome ajoute_polynome(polynome p, polynome q)
{
    double coefficient;
    polynome r = NULL, *s = &r, t;
    monome m;
    while (p != NULL && q != NULL) {
        if (p->degre < q->degre) {
            t = p; p = q; q = t;
        }
        coefficient = p->coefficient;
        if (p->degre == q->degre) {
            coefficient += q->coefficient;
            q = q->suivant;
        }
        m = cree_monome(p->degre, coefficient);
        *s = m;
        s = &m->suivant;
        p = p->suivant;
    }
    if (q != NULL) /* seul un des pointeurs p ou q peut être non NULL */
        p = q;
    while (p != NULL) {
        m = cree_monome(p->degre, p->coefficient);
        *s = m;
        s = &m->suivant;
        p = p->suivant;
    }
    return r;
}
```