TD 4 : Pointeurs et structures

Semaine du 18 février 2008

1 Chaînes de caractères

▶ **Exercice 1**

```
char *recherche(char *s, char c) {
    if (s != NULL) {
      while (*s != '\0') {
          if (*s == c)
              return s;
          s++;
      }
    }
    return NULL;
}
```

▶ **Exercice 2**

```
int compte(char *s, char c) {
    int n = 0;
    s = recherche(s, c);
    while (s != NULL) {
        n++;
        s++;
        s = recherche(s, c);
    }
    return n;
}
```

2 Polynômes

▶ **Exercice 3**

```
struct polynome *somme_polynome(struct polynome *P, struct polynome *Q) {
    struct polynome *resultat = malloc(sizeof(struct polynome));
    int i;
    if (P->degre < Q->degre) {
        struct polynome *T = P; P = Q; Q = T;
    } /* Q->degre <= P->degre */
    resultat->degre = P->degre;
    resultat->coefficients = malloc((P->degre + 1) * sizeof(double));
    for (i = 0; i<= Q->degre; i++)
        resultat->coefficients[i] = P->coefficients[i] + Q->coefficients[i];
    for (; i <= P->degre; i++) /* i = Q->degre + 1 */
        resultat->coefficients[i] = P->coefficients[i];
    return resultat;
}
```

```c
struct polynome *produit_polynome(struct polynome *P,struct polynome *Q) {
    struct polynome *resultat = malloc(sizeof(struct polynome));
    int i, j;
    resultat->degre = P->degre + Q->degre;
    resultat->coefficients = malloc((resultat->degre + 1) * sizeof(double));
    for (i = 0; i <= resultat->degre; i++)
        resultat->coefficients[i] = 0.0;
    for (i = 0; i <= P->degre; i++)
        for (j = 0; j <= Q->degre; j++)
            resultat->coefficients[i + j] += P->coefficients[i]
                                             * Q->coefficients[j];
    return resultat;
}
```

3 Matrices

▶ **Exercice 4**

```c
struct matrice *alloue_matrice(int lignes, int colonnes) {
    struct matrice *A = malloc(sizeof(struct matrice));
    int i;
    A->lignes = lignes;
    A->colonnes = colonnes;
    A->coefficients = malloc(A->lignes * sizeof(double *));
    for (i = 0; i < A->lignes; i++)
        A->coefficients[i] = malloc(A->colonnes * sizeof(double));
    return A;
}
```

▶ **Exercice 5**

```c
struct matrice *produit_matrice(struct matrice *A, struct matrice *B) {
    struct matrice *resultat;
    int i, j, k;
    if (A->colonnes != B->lignes)
        return NULL;
    resultat = alloue_matrice(A->lignes, B->colonnes);
    for (i = 0; i < A->lignes; i++)
        for (j = 0; j < B->colonnes; j++) {
            resultat->coefficients[i][j] = 0.0;
            for (k = 0; k < A->colonnes; k++)
                resultat->coefficients[i][j] += A->coefficients[i][k]
                                               * B->coefficients[k][j];
        }
    return resultat;
}
```

▶ **Exercice 6**

```c
void libere_matrice(struct matrice *A) {
    int i;
    for (i = 0; i < A->lignes; i++)
        free(A->coefficients[i]);
    free(A->coefficients);
    free(A);
}
```

4 Pile

▶ **Exercice 7**

```
void empile_pile_simple(struct pile_simple *pile, int n) {
    int *ptr, i;
    ptr = malloc((pile->taille + 1) * sizeof(int));
    for (i = 0; i < pile->taille; i++)
        ptr[i] = pile->elements[i];
    ptr[i] = n; /* ptr[pile->taille] = n */
    if (pile->taille != 0)
        free(pile->elements);
    pile->elements = ptr;
    pile->taille++;
}

void empile_pile_simple2(struct pile_simple *pile, int n) {
    if (pile->taille == 0)
        pile->elements = NULL; /* realloc(NULL, .) équivaut à malloc(.) */
    pile->elements = realloc(pile->elements,
                             (pile->taille + 1) * sizeof(int));
    pile->elements[pile->taille] = n;
    pile->taille++;
}

int depile_pile_simple(struct pile_simple *pile) {
    int *ptr, i, n = pile->elements[pile->taille - 1];
    pile->taille--;
    if (pile->taille != 0)
        ptr = malloc(pile->taille * sizeof(int));
    else /* pile->taille == 0 */
        ptr = NULL;
    for (i = 0; i < pile->taille; i++)
        ptr[i] = pile->elements[i];
    free(pile->elements);
    pile->elements = ptr;
    return n;
}

int depile_pile_simple2(struct pile_simple *pile) {
    int n = pile->elements[pile->taille - 1];
    pile->taille--;
    if (pile->taille != 0)
        pile->elements = realloc(pile->elements, pile->taille * sizeof(int));
    else { /* pile->taille == 0 */
        free(pile->elements);
        pile->elements = NULL;
    }
    return n;
}
```

▶ **Exercice 8**

```
void empile_pile_amortie(struct pile_amortie *pile, int n) {
    if (pile->taille == pile->capacite) {
        int *ptr, i;
        if (pile->capacite != 0)
```

```
            pile->capacite *= 2;
        else
            pile->capacite = 1;
        ptr = malloc(pile->capacite * (sizeof(int)));
        for (i = 0; i < pile->taille; i++)
            ptr[i] = pile->elements[i];
        if (pile->taille != 0)
            free(pile->elements);
        pile->elements = ptr;
    }
    pile->elements[pile->taille] = n;
    pile->taille++;
}

void empile_pile_amortie2(struct pile_amortie *pile, int n) {
    if (pile->taille == pile->capacite) {
        if (pile->capacite == 0) {
            pile->capacite = 1;
            pile->elements = NULL; /* realloc(NULL, .) équivaut à malloc(.) */
        } else
            pile->capacite *= 2;
        pile->elements = realloc(pile->elements,
                                 pile->capacite * (sizeof(int)));
    }
    pile->elements[pile->taille] = n;
    pile->taille++;
}

int depile_pile_amortie(struct pile_amortie *pile) {
    int n = pile->elements[pile->taille - 1];;
    pile->taille--;
    if (pile->taille <= pile->capacite / 4) {
        int *ptr, i;
        pile->capacite /= 2;
        if (pile->capacite != 0)
            ptr = malloc(pile->capacite * sizeof(int));
        else /* pile->capacite == 0 */
            ptr = NULL;
        for (i = 0; i < pile->taille; i++)
            ptr[i] = pile->elements[i];
        free(pile->elements);
        pile->elements = ptr;
    }
    return n;
}

int depile_pile_amortie2(struct pile_amortie *pile) {
    int n = pile->elements[pile->taille - 1];
    pile->taille--;
    if (pile->taille <= pile->capacite / 4) {
        pile->capacite /= 2;
        if (pile->capacite != 0)
            pile->elements = realloc(pile->elements,
                                     pile->capacite * sizeof(int));
        else { /* pile->capacite == 0 */
            free(pile->elements);
            pile->elements = NULL;
        }
    }
    return n;
}
```