TD 8 : Listes

Semaine du 19 mars 2007

1 Tables de hachage

▶ **Exercice 1**

```
table_de_hachage_t cree_table_de_hachage(int taille) {
    int i;
    table_de_hachage_t table = malloc(sizeof(struct table_de_hachage_s));
    table->taille = taille;
    table->tableau = malloc(table->taille * sizeof(liste_t));
    for (i = 0; i < table->taille; i++)
        table->tableau[i] = NULL;
    return table;
}
```

▶ **Exercice 2**

```
void detruit_table_de_hachage(table_de_hachage_t table) {
    int i;
    liste_t liste;
    for (i = 0; i < table->taille; i++)
        while (table->tableau[i] != NULL) {
            liste = table->tableau[i]->suivant;
            free(table->tableau[i]);
            table->tableau[i] = liste;
        }
    free(table->tableau);
    free(table);
}
```

▶ **Exercice 3**

```
int hachage(table_de_hachage_t table, char *cle) {
    int hache = 0;
    for (; *cle; cle++)
        hache += *cle;
    return (hache % table->taille);
}
```

▶ **Exercice 4**

```
void insere(table_de_hachage_t table, char *cle, int valeur) {
    int hache;
    liste_t liste = malloc(sizeof(struct liste_s));
    hache = hachage(table, cle);
    liste->suivant = table->tableau[hache];
    liste->cle = cle;
    liste->valeur = valeur;
    table->tableau[hache] = liste;
}
```

▶ **Exercice 5**

```
int recherche(table_de_hachage_t table, char *cle, int *valeur) {
    liste_t liste = table->tableau[hachage(table, cle)];
    for (; liste; liste = liste->suivant)
        if (strcmp(cle, liste->cle) == 0) {
            *valeur = liste->valeur;
            return 1;
        }
    return 0;
}
```

▶ **Exercice 6**

```
void supprime(table_de_hachage_t table, char *cle) {
    int hache;
    liste_t liste, *precedent;
    hache = hachage(table, cle);
    liste = table->tableau[hache];
    precedent = &table->tableau[hache];
    for (; liste; precedent = &liste->suivant, liste = liste->suivant)
        if (strcmp(cle, liste->cle) == 0) {
            *precedent = liste->suivant;
            free(liste);
            return;
        }
}
```

▶ **Exercice 7**

```
void affiche_numero(int numero) {
    int n = 100000000, m = 0;
    printf("0");
    while (n) {
        printf("%d", (numero / n) % 10);
        m++;
        if (m % 2 && m != 9)
            printf(".");
        n /= 10;
    }
    printf("\n");
}
```

2 Files

▶ **Exercice 8**

```
file_t cree_file(void) {
    file_t file = malloc(sizeof(struct file_s));
    file->debut = file->fin = NULL;
    return file;
}

int est_vide(file_t file) {
    return (file->debut == NULL); /* ou file->fin == NULL */
}

void detruit_file(file_t file) {
```

```
    liste_t liste;
    while (file->debut != NULL) {
        liste = file->debut->suivant;
        free(file->debut);
        file->debut = liste;
    }
    free(file);
}
```

► **Exercice 9**

```
void enfile(file_t file, char *cle, int valeur) {
    liste_t liste = malloc(sizeof(struct liste_s));
    liste->suivant = NULL;
    liste->cle = cle;
    liste->valeur = valeur;
    /* file vide : on met à jour debut */
    if (est_vide(file))
        file->debut = liste;
    /* au moins un élément dans la file : file->fin != NULL */
    else
        file->fin->suivant = liste;
    file->fin = liste;
}
```

► **Exercice 10**

```
void defile(file_t file, char **cle, int *valeur) {
    liste_t liste;
    assert(!est_vide(file));
    *cle = file->debut->cle;
    *valeur = file->debut->valeur;
    liste = file->debut->suivant;
    free(file->debut);
    /* un seul élément dans la file : on met à jour debut et fin */
    if (file->debut == file->fin)
        file->debut = file->fin = NULL;
    /* au moins deux éléments dans la file : on ne met à jour que début */
    else
        file->debut = liste;
}
```

► **Exercice 11**

```
void detruit_file2(file_t file) {
    char *cle;
    int valeur;
    while (!est_vide(file))
        defile(file, &cle, &valeur);
    free(file);
}
```