TP 5

Programmation en C (LC4)

Semaine du 26 février 2007

1 Jouons avec les pointeurs

2 Plusieurs fichiers !

```
main: main.o traduction.o dico.o
	gcc -o main main.o traduction.o dico.o

dico.o: dico.c dico.h traduction.h
	gcc -o dico.o -c dico.c

traduction.o: traduction.c traduction.h
	gcc -o traduction.o -c traduction.c

main.o: main.c dico.h
	gcc -o main.o -c main.c

clean:
	rm -f main.o traduction.o dico.o main
```

3 Pile (Rappel TD4)

```
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;
}

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;
}
```