TP1: Manipulation des tableaux en C

Programmation en C (LC4)

Semaine du 1er février 2010

1 Rappels pratiques

On rappelle que pour compiler un programme C, on fait appel au compilateur gcc de la manière suivante dans un terminal : `gcc fichier.c -o nom_prog`. Les options de compilation plus avancées feront l'objet d'un prochain TP.

Pour le TP, on utilisera la fonction `printf` qui se trouve dans la bibliothèque `stdio`. Cette bibliothèque contient les fonctions utiles pour gérer les entrées/sorties du programmes, donc entre autre la récupération d'une ligne de texte saisie par l'utilisateur, l'affichage de texte dans la console, l'ouverture, la lecture et l'écriture dans un fichier. Pour l'utiliser, il faut rajouter la ligne `#include<stdio.h>` en tête de son fichier.

La fonction `printf` est celle servant à afficher du texte dans le terminal (et donc est très utile pour tester les fonctions écrites). Elle possède un nombre arbitraire d'arguments, dont le premier est forcément une chaîne de caractère. Cette chaine peut contenir des jokers, permettant d'afficher le contenu de variables, prises dans l'ordre d'apparition des arguments de `printf`. Ces jokers peuvent être `%d` pour un entier non-signé, `%u` pour un entier signé, `%f` pour un double, `%c` pour un caractère ou `%s` pour une chaine de caractères. Entre le symbole `%` et la lettre, on peut rajouter un chiffre n, qui forcera l'affichage à se faire sur au moins n caractères. On n'oubliera pas non plus que le caractère de retour à la ligne est `\n`

Par exemple, si `a` et `b` sont des variables de type `int` valant respectivement 3 et 4, la ligne
`printf("a vaut %d \n", a);` affichera à l'exécution sur le terminal
```
a vaut 3
```
et la ligne
`printf("a vaut %3d et b vaut %d.\n", a, b);` affichera à l'exécution sur le terminal
```
a vaut   3 et b vaut 4.
```

Des informations plus précises sont évidemment disponibles dans le manuel.

2 Permutations

Les permutations sont fréquemment croisées en mathématiques, mais aussi en informatique où elles sont utilisées par exemple en cryptographie ou pour analyser l'efficacité d'algorithmes. Nous modélisons des permutations de $[0 \dots n-1]$ dans $[0 \dots n-1]$ à l'aide d'un tableau. La valeur de la case i du tableau correspond à l'image de i par la permutation. Ainsi la permutation :

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 0 & 2 \end{pmatrix}$$

est dénotée par le tableau C :

```
int f[4] = { 1, 3, 0, 2 };
```

Exercice 1 Ecrire une fonction `void affiche_tableau(int n, int f[])` qui affiche le contenu du tableau f de taille n de manière lisible.

Exercice 2 Ecrire une fonction `int est_identite(int n, int f[])` qui teste si une permutation f de $[0 \dots n-1]$ est l'identité. Cette fonction renverra un entier valant 1 en cas de réponse positive et 0 autrement.

Exercice 3 Une inversion d'une permutation σ est un couple (i, j) d'entiers tels que $i < j$ et $\sigma(i) > \sigma(j)$. Ecrire une fonction `int nombre_inversions(int n, int f[])` qui calcule le nombre d'inversions d'une permutation.

Exercice 4 Ecrire la fonction
`void compose_permutations(int n, int perm1[], int perm2[], int perm_composee[])`
qui calcule la composée σ, de σ_1 et σ_2, telle que $\sigma = \sigma_1 \circ \sigma_2$ (et donc $\sigma(i) = \sigma_1(\sigma_2(i))$).

Exercice 5 Ecrire un fonction `void inverse_directe(int n, int perm[], int perm_inverse[])` qui calcule l'inverse σ^{-1} de la permutation σ, tel que $\sigma^{-1} \circ \sigma = Id$.

Exercice 6 Ecrire une fonction `int rang(int n, int perm[])` qui calcule le nombre de fois qu'il faut composer la permutation `perm` avec elle-même pour obtenir l'identité.

Exercice 7 Ecrire une fonction `void inverse_ordre(int n, int perm[], int inverse[])` qui calcule l'inverse σ^{-1} de la permutation σ, tel que $\sigma^{-1} \circ \sigma = Id$. en utilisant la fonction précédente.

3 Tri

Le tri par insertion est un tri relativement simple et utilisé couramment. L'idée est qu'il est facile d'insérer un élément dans un tableau déjà trié, et qu'avec ça trier revient juste à insérer les éléments un à un dans un tableau.

Exercice 8 Ecrire une fonction `void insere(int n, int dst[], int val)` qui prend un tableau `dst` de taille n, trié dans l'ordre croissant et dont la dernière case est indéterminée, et y insère l'entier `val` de manière à ce que le tableau reste trié.

Exercice 9 Ecrivez une fonction `void tri(int n, int src[], int dst[])` qui prend un tableau `src` de taille n et le recopie trié dans le tableau `dst` en utilisant la fonction précédente.

Exercice 10 Ecrivez une fonction `void tri_en_place(int n, int tab[])` qui prend un tableau `tab` de taille n et le trie sans utiliser de nouveau tableau.