Université d'Aix-Marseille III - Faculté des Sciences et Techniques de Saint-Jérôme

Licence Sciences et Technologies - Mention Maths-Info

I3 - Algorithmique et Programmation

TD n°1

1. Le jeu de Nim

Le jeu de Nim est un jeu à deux joueurs qui comporte 4 rangées d'allumettes. La première rangée contient 1 allumette, la deuxième 3 allumettes, la troisième 5 et la quatrième 7. Alternativement, chaque joueur doit retirer une ou plusieurs allumettes (autant qu'il veut mais au moins une) dans une *seule* rangée. Le gagnant est celui qui retire la ou les dernières allumettes.

On mémorisera l'état du jeu par un tableau de quatre entiers dont chaque case indique le nombre d'allumettes présentes dans une rangée.

1.1. Analyser le problème et le décomposer en sous-problèmes.

1.2. Ecrire les lignes de code C qui résolvent chacun des sous-problèmes. Vous préciserez dans chaque cas quelles sont les entrées et les sorties.

1.3. En déduire les fonctions et procédures correspondantes. Vous préciserez la spécification de chaque fonction ou procédure.

1.4. Ecrire le programme qui permet à deux utilisateurs de jouer au jeu de Nim.

2. Tri de tableau

On considère un tableau `t` de `MAX` entiers. Le type C utilisé est le suivant :

```
#define MAX 100
typedef int tabent[MAX];
```

On supposera que tous les éléments du tableau sont différents.

2.1. Donner les lignes de code C qui, étant donnés un tableau `t` et un entier `k` avec $1 \leq k \leq MAX$, détermine l'indice dans le tableau de la k^{ème} valeur du tableau, c'est-à-dire de l'élément qui possède $k-1$ valeurs inférieures et $n-k$ valeurs supérieures exactement.
En déduire la fonction correspondante.

2.2. En utilisant la fonction précédente, écrire les lignes de code C qui, étant donné un tableau, réalise le tri de ce tableau. En déduire la fonction correspondante. Le tri est réalisé selon la méthode suivante : étant donné un tableau à trier `t`, on place dans la case d'indice $k-1$ d'un tableau auxiliaire la $k^{ème}$ valeur du tableau `t`. Ainsi, à l'issue du traitement, le tableau auxiliaire contient le tableau `t` trié.

2.3. Même question sans employer de tableau auxiliaire.

2.4. Ecrire un programme principal qui saisit au clavier un tableau `t`, puis le trie et enfin l'affiche.

3. Tri de tableau (suite)

On considère un tableau t de `MAX` entiers. Pour ceci, on utilisera le type défini dans l'exercice précédent. Cependant, dans ce problème, nous supposerons que toutes les valeurs stockées dans le tableau à trier sont des entiers compris entre 0 et MAX − 1, et que certaines valeurs peuvent être présentes plusieurs fois dans le tableau à trier.

3.1. Ecrire la fonction dont l'en-tête et les spécifications sont donnés ci-dessous :

```
int nb_oc(tabent t, int i)
/* specifications : a pour resultat le nombre d'occurrences */
/* de la valeur t[i] dans le tableau t                      */
```

3.2. En utilisant la fonction **nb_oc**, écrire la fonction C dont l'en-tête et les spécifications sont donnés ci-dessous :

```
void t_nb_oc(tabent t, tabent c)
/* specifications : en sortie, pour tout i, 0 <= i < MAX, c[t[i]]  */
/* a pour valeur le nombre d'occurrences de la valeur t[i] dans    */
/* le tableau t.                                                   */
/* Les elements c[k] ou k ne figure pas dans t seront affectes a 0 */
```

Par exemple, si t[3] = 5, et que la valeur 5 est présente 7 fois dans t, on aura c[t[3]] = c[5] = 7.

3.3. En utilisant la fonction **t_nb_oc**, écrire la fonction C dont l'en-tête et les spécifications sont donnés ci-dessous :

```
void tri(tabent t)
/* specifications : en sortie, t est trie dans l'ordre croissant */
```

La méthode employée consistera à recopier dans **t** et en bonne position, autant de fois que nécessaire, les occurrences des éléments figurant initialement dans **t**, à l'aide de la structure calculée par la procédure **t_nb_oc**. Par exemple, si la valeur 0 est présente 4 fois dans **t** en entrée et si la valeur 1 est présente 2 fois dans t en entrée, on aura c[0] = 4 et c[1] = 2 ; on affectera alors t[0], t[1], t[2] et t[3] à la valeur 0, puis t[4] et t[5] à la valeur 1, et ainsi de suite jusqu'à la réorganisation totale du tableau **t**.

3.4. L'inefficacité relative de la fonction **tri** est essentiellement due à l'emploi de la fonction **nb_oc** dans la procédure **t_nb_oc**. Si nous n'utilisons pas celle-ci, il est possible de définir une fonction de tri s'appuyant sur le même principe, mais qui optimise significativement la fonction précédente. Réécrire la procédure **t_nb_oc** afin d'améliorer la fonction de tri précédente.

4. Problème du drapeau hollandais

On considère un tableau `d` de `N` caractères. Le type C utilisé est le suivant :

```
#define N ...
typedef char drapeau[N];
```

Ce tableau ne peut contenir que les caractères 'B', 'b' et 'R', qui représentent respectivement les 3 couleurs 'Bleu', 'blanc' et 'Rouge' du drapeau hollandais. On suppose qu'initialement, le tableau `d` contient les différentes couleurs qui sont présentes en nombres différents (il peut n'y avoir qu'une couleur présente) et réparties aléatoirement.

Ecrire une fonction C qui réorganise le tableau de telle sorte que l'ordre des couleurs soit respecté. Pour cela, on pourrait utiliser une version adaptée du tri précédent. Cependant, on préférera une solution plus élégante et plus efficace. Cette solution consiste à découper le tableau en quatre parties :

- la première est constituée des caractères 'B' déjà examinés,
- la seconde est constituée des caractères 'b' déjà examinés,
- la troisième contient les éléments qui restent à étudier,
- la quatrième est constituée des caractères 'R' déjà examinés.

A chaque étape, on examine un élément de la troisième partie et on le place alors dans l'une des trois autres parties.