Programmation avancée en langage C

L3 Bio-informatique
examen du 13 mai 2014 - durée 3 heures
téléphones, ordinateurs, documents interdits - 2 pages

Exercice 1

Écrire un programme qui prend en paramètre (sur la ligne de commande) un nom de fichier (supposé être un fichier texte) et renvoie le nombre de lignes de ce fichier, ou -1 en cas d'erreur.

Exercice 2

Discuter de la légalité du code suivant :

```
int n;
scanf("%i",&n);
int t3[n];
```

Exercice 3

On considère des variables définies ainsi :

```
int x; int t[10]; int t2[10]; int m[10][10]; int *p; int *p2; int **q;
```

On suppose que toutes ces variables sont correctement initialisée, en particulier `p` est un tableau dynamique de 10 cases et `q` de 10x10. Puis on effectue l'une des affectations de la liste suivante. Dire si elle est valable ou non (provoquera une erreur de compilation). On ne fait pas ces affectations à la suite les unes des autres, mais on considère 20 programmes différents, chacun contenant une seule affectation. Donner simplement les numéros des affectations qui ne compilent pas.

1	`t[3] = x;`	8	`m[3] = t;`	15	`q[3] = t;`
2	`t + 3 = x;`	9	`m = q;`	16	`q[3] = p;`
3	`t2 = t;`	10	`q = m;`	17	`x = q[3] - q[5];`
4	`t = p;`	11	`x = t - t2;`	18	`p2 = q + p;`
5	`t = p + 3;`	12	`x = p - p2;`	19	`p2 = q - 3;`
6	`p = t;`	13	`t = t2 + x;`	20	`t = m[3];`
7	`p = t + 3;`	14	`p2 = p + x;`		

Exercice 4

Le but de cet exercice est de stocker une liste chaînée de réels (chaque réel au format `double`) sur disque. On veut un stockage **binaire** et non texte : l'usage de `fprintf` est interdit, il faut utiliser `fwrite`. Rappel, la syntaxe est
`size_t fwrite(const void *ptr, size_t taillelem, size_t nbelem, FILE *fichier);`

Commencer par définir le type d'une cellule de liste chaînée de réels.

Puis faire une fonction qui prend en paramètre
- un nom de fichier, et
- la tête d'une liste chaînée de réels

qui ouvre le fichier en écriture, et écrit tous les réels de la liste. Elle renverra -1 en cas d'erreur ; et 0 sinon.

Problème : arbres d'entiers

Un **arbre d'entiers** est un arbre binaire où chaque nœud contient (c'est-à-dire, qu'il stocke) un nombre entier [1]. Un nœud, en plus de la valeur qu'il stocke, peut avoir un fils gauche et un fils droit.

Question 1

Proposer un type pour stocker un arbre d'entiers (en utilisant `typedef` de préférence)

Question 2

Faire une fonction `nbNoeuds` qui, étant donné le nœud racine N d'un arbre d'entiers, renvoie le nombre de nœuds de cet arbre.

Question 3

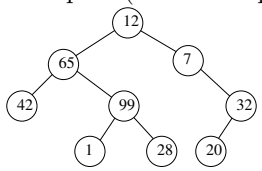
Faire une fonction `rechercher` qui, étant donné le nœud racine N d'un arbre d'entiers et une valeur entière x, renvoie 1 si (au moins) un des nœuds de l'arbre contient x, et 0 sinon

Question 4

Un **arbre d'entiers unique** (A.E.U.) est un arbre d'entiers qui ne contient pas deux nœuds stockant la même valeur (tout nombre est présent 0 ou 1 fois dans l'arbre). Faire une fonction `estAEU` qui, étant donné le nœud racine N d'un arbre d'entiers, renvoie 1 s'il est racine d'un A.E.U. et 0 sinon

Question 5

Un A.E.U peut être codé en tableau de la façon suivante. Un arbre à n nœuds donne un tableau de deux lignes et n colonnes. Tout nœud N a un *numéro de colonne* i (qui peut être différent de la *valeur* stockée dans N !) de sorte que le numéro de colonne d'un fils soit inférieur à celui de son père (la racine a donc numéro de colonne 0). Ensuite T[0][i] est la valeur du nœud N et T[1][i] est la valeur de son père (si N n'est pas racine. Sinon, on pose T[1][0]=0).

Exemple : 

peut se coder en

12	65	99	42	7	32	1	20	28
0	12	65	65	12	7	99	32	99

Faire une fonction `coderAEU` qui prend en paramètre le nœud racine N d'un arbre d'entiers et, si cet arbre est un A.E.U, renvoie un tableau codant cet arbre (et `NULL` sinon).

1. c'est la même structure que pour des arbres binaires de recherche, par exemple ; mais sans la contrainte entre la valeur d'un nœud et celle de ses fils