Initiation à la programmation en C

TP n°4

Antoine Miné

8 mars 2007

Site du cours: `http://www.di.ens.fr/~mine/enseignement/prog2006/`

Exercice 1. Échange.
Écrivez une fonction de prototype :

```
void echange(int* a, int* b);
```

qui échange le contenu de deux variables.
Que se passe-t-il si on appelle `echange(&x,&x)` ?

Exercice 2. Tri par insertion.
Voici une nouvelle méthode de tri de tableau. D'abord, on cherche le plus petit élément du tableau. On le positionne en première position en l'échangeant avec l'élément d'indice `0`. On cherche ensuite le plus petit élément du sous tableau commençant à l'indice `1`. On l'échange avec celui d'indice `1`. Et ainsi de suite...
Programmez le tri par insertion grâce à la fonction `echange` de l'exercice précédent.

Exercice 3. Lecture au clavier.
Écrivez une fonction

```
int lit_clavier(int* tab, int max);
```

qui lit au clavier au plus `max` entiers positifs et les stocke dans `tab`. La fonction retourne après avoir lu `max` valeurs ou bien quand l'utilisateur entre `-1`. Dans tous les cas, le nombre d'éléments lus est retourné.
Utilisez cette fonction et celle de l'exercice précédent pour faire un programme qui lit une liste d'entiers au clavier, la trie par ordre croissant et affiche le résultat trié à l'écran.

Exercice 4. Triangle de Pascal.
On souhaite afficher à l'écran un triangle de Pascal :

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5 10 10  5  1
1  6 15 20 15  6  1
1  7 21 35 35 21  7  1
1  8 28 56 70 56 28  8  1
```

celui-ci se calcule de la manière suivante :
– la $i-$ème ligne contient i colonnes,
– chaque ligne commence et se termine par un 1,
– l'élément à la ligne i, colonne j est la somme des éléments aux colonnes j et $j-1$ de la ligne $i-1$.

Faites un programme qui calcule dans un tableau bidimensionnel `int pascal[N][N]` les `N` premières lignes du triangle de Pascal. L'élément à la ligne i, colonne j sera stocké dans `pascal[i][j]` (si $j > i$, `pascal[i][j]` ne sera pas utilisé ; on "gâche" donc environ la moitié du tableau).
Affichez le triangle de Pascal à l'écran (pas toute la matrice, seulement les éléments utiles !).
Faites aussi un programme qui affiche le triangle de la manière suivante : si l'élément est impair on affichera le caractère `#`, sinon, le caractère espace.

Exercice 5. Le retour du triangle de Pascal.
Remarquons que chaque ligne du triangle de Pascal ne dépend que de la ligne précédente. Il n'est donc pas utile de stocker la matrice en entier en mémoire si on souhaite simplement l'afficher ligne par ligne. Il suffit de ne garder qu'une ligne "courante" qui sera mise à jour après affichage. On économise ainsi beaucoup de mémoire.
On propose la boucle suivante pour passer de la ligne `i` à la ligne `i+1` :

```
for ( j=1; j<i; j++ )
  ligne[j] = ligne[j] + ligne[j-1];
```

malheureusement, cela ne marche pas !
Déterminez pourquoi et proposez une version correcte.

Exercice 6. Tableaux et pointeurs.
Comparez les trois versions suivantes de `f`, de `g`, et de `h` :

```
void f1(int a[5]) { int i; for (i=0;i<5;i++) printf("%i ",a[i]); }
void f2(int a[])  { int i; for (i=0;i<5;i++) printf("%i ",a[i]); }
void f3(int *a)   { int i; for (i=0;i<5;i++) printf("%i ",a[i]); }

void g1(int a[5]) { printf("%i\n",sizeof(a)); }
void g2(int a[])  { printf("%i\n",sizeof(a)); }
void g3(int *a)   { printf("%i\n",sizeof(a)); }

void h1() { int  b[10]; f1(b); }
void h2() { int  b[];   f1(b); }
void h3() { int* b;     f1(b); }
```