

TP 6 : graphiques, tableaux

Informatique Fondamentale (IF1)

Semaine du 17 Novembre 2008

1 Graphiques

Comme vous l'avez vu en Cours/TD, la classe « `Deug` » contient des fonctions qui permettent de faire des graphiques. Dans ce TP, on pourra utiliser les fonctions suivantes :

- `static void startDrawings(int width, int height)`, qui crée une zone d'affichage de taille `width × height` pixels;
- `static void drawPoint(int x, int y)`, qui colore en noir le point de coordonnées `(x,y)`;
- `static void stopDrawings()`, qui ferme la zone d'affichage.

En particulier, vous n'utiliserez pas les fonctions telles que `drawLine` qui permettent de résoudre la plupart de ces exercices en une seule ligne.

Exercice 1. Écrivez un programme qui crée une zone d'affichage de largeur 400 et de hauteur 200, et qui dessine un segment de droite commençant en `(100, 100)` et se terminant en `(300, 100)`.

Exercice 2. Écrivez un programme qui affiche un segment de droite allant de `(100, 100)` à `(100, 300)`. (À vous de choisir la taille de la zone d'affichage).

Exercice 3. Écrivez un programme qui affiche un rectangle creux ayant des angles en `(100, 100)` et `(300, 300)`.

Exercice 4. Même question avec un rectangle plein.

Exercice 5. Écrivez un programme qui affiche un segment allant de `(100, 100)` à `(300, 300)`.

Exercice 6. Écrivez un programme qui affiche un triangle ayant des sommets en `(100, 100)`, `(100, 300)` et `(300, 300)`.

2 Tableaux

Exercice 7. Écrivez une fonction `affiche` d'entête

```
void affiche(int[] tableau)
```

qui affiche les éléments d'un tableau passé en paramètre. Écrivez une fonction `main` qui vous permette de tester cette fonction (par exemple en l'appliquant à un tableau *ad hoc* construit dans le `main`).

Exercice 8. Écrivez une fonction `entree` d'entête

```
int[] entree()
```

qui lit un entier `n`, puis qui lit `n` entiers, et retourne un tableau contenant ces `n` entiers. Écrivez une fonction `main` qui lit un tableau d'entiers puis l'affiche. (Vous pouvez bien-sûr vous servir de la fonction définie à l'exercice précédent.)

Exercice 9. Écrivez une fonction `afficheInverse` qui affiche les éléments d'un tableau passé en paramètre dans l'ordre inverse de celui dans lequel ils sont stockés. Écrivez une fonction `main` qui lit un entier `n`, puis lit `n` entiers, puis les affiche dans l'ordre inverse de celui dans lequel ils ont été rentrés. (Vous pouvez bien-sûr vous servir de fonctions définies aux exercices précédents.)

3 Recherche

Exercice 10. Écrivez une fonction `recherche`, d'entête

```
int recherche(int[] a, int valeur)
```

qui retourne le premier indice `i` tel que `a[i]` vaut `valeur`, ou `-1` si un tel indice n'existe pas. Écrivez une fonction `main` qui lit un entier `n`, puis qui lit `n` entiers, et indique si l'un de ces entiers vaut 42, et si oui, lequel. (Vous pouvez bien-sûr vous servir de fonctions définies aux exercices précédents.)

Exercice 11. Écrivez une fonction qui prend comme argument un tableau de nombres et retourne `true` si ce tableau est trié en ordre croissant. Écrivez une fonction `main` qui lit un tableau d'entiers et indique s'il est trié en ordre croissant.

Lorsque le tableau dans lequel on recherche une valeur est trié, il est plus efficace de procéder en utilisant un algorithme appelé *recherche par dichotomie* plutôt que la *recherche linéaire* utilisée ci-dessus.

La recherche par dichotomie manipule trois entiers $p \leq m \leq g$ (pour « petit », « moyen » et « grand »). À tout moment, `a[p] ≤ valeur ≤ a[g]`.

Initialement, `p` vaut 0 et `g` la taille du tableau moins un. À chaque étape du calcul, `m` vaut $\lfloor (p + g)/2 \rfloor$; si `a[m] < valeur`, alors `p` prend la valeur de `m`, sinon c'est `g` qui prend la valeur de `m`. Le calcul s'arrête lorsque `p = g`.

Exercice 12. Écrivez une fonction

```
int rechercheDichotomie(int[] a, int valeur)
```

qui utilise une recherche par dichotomie pour trouver un indice `i` tel que `a[i] = valeur`, et retourne `-1` si un tel indice n'existe pas.

Exercice 13. Écrivez un programme qui :

- lit un entier `valeur` ;
- lit un tableau d'entiers `a` ;
- vérifie si ce tableau est trié ;
- recherche un indice i tel que $a[i] = \text{valeur}$ en utilisant, selon le résultat du point précédent, soit une recherche linéaire soit une recherche par dichotomie ;
- affiche cet indice i .

Vous pourrez bien sûr vous servir des fonctions définies aux exercices précédents.

4 S'il vous reste du temps : tri

Le *tri par insertion* est un algorithme qui permet de trier un tableau *en place*, c'est-à-dire sans se servir d'un tableau additionnel.

Le tri par insertion d'un tableau `a` de taille n manipule une variable m , qui scinde le tableau en deux parties, la partie allant de 0 à $m - 1$, qui est déjà triée, et la partie de m à la taille n du tableau, qui reste à trier.

Initialement, m vaut 1. À chaque étape, on compare l'élément `a[m]` à chacun, successivement, des éléments `a[0]`, `a[1]`, etc. Lorsqu'on trouve le plus petit indice i tel que `a[i]` est supérieur à `a[m]`, on insère `a[m]` « avant » `a[i]` en effectuant une permutation circulaire des éléments `a[m]`, `a[i] .. a[m - 1]`. On incrémente ensuite m .

L'algorithme termine lorsque m vaut n .

5	3	1	2	4
3	5	1	2	4
1	3	5	2	4
1	2	3	5	4
1	2	3	4	5

FIG. 1 – Un tri par insertion. Dans la troisième ligne, par exemple, m vaut 3, et les trois premiers éléments du tableau sont triés. Comme $3 > 2$, i vaut 1, et après une permutation circulaire de 3, 5, 2, m vaut 4.

Exercice 14. Écrivez une fonction d'entête

```
void tri(int[] a)
```

qui trie en place le tableau `a`. Écrivez une fonction `main` qui lit un tableau, le trie, et affiche le tableau trié. Vérifiez que votre programme fonctionne correctement lorsque la même valeur apparaît plusieurs fois dans le tableau donné en entrée.