Algorithmique impérative

Objectifs

— Programmation impérative
— Structures de contrôle
— Méthodes des raffinages

Exercice 1 : Suite de Fibonacci

Les termes de la suite de Fibonacci sont définis par la relation de récurrence suivante :

$fib(0) = 0$
$fib(1) = 1$
$fib(n) = fib(n-1) + fib(n-2)$ si $n \geq 2$

1.1 Afficher le n-ième terme de la suite de fibonacci, n entier positif lu au clavier.
1.2 Afficher le 1er terme de la suite de Fibonnacci supérieur à M, un entier naturel lu au clavier tel que $M > 1$

Exercice 2 : Nombres parfaits

Écrire un programme qui affiche tous les nombres parfaits entre 2 et N, N étant lu au clavier.

Un nombre parfait est un entier égal à la somme de ses diviseurs, lui exclu. Par exemple, 28 est un nombre parfait ($28 = 1 + 2 + 4 + 7 + 14$).

Exercice 3 : Nombres amis

Deux nombres N et M sont amis si la somme des diviseurs de M (en excluant M lui-même) est égale à N et la somme des diviseurs de N (en excluant N lui-même) est égale à M.

Par exemple, 220 et 284 sont amis. En effet, la somme des diviseurs de 220 hors 220 est $1+2+4+5+10+11+20+22+44+55+110 = 284$ et la somme des diviseurs de 284 hors 284 est $1 + 2 + 4 + 71 + 142 = 220$.

Écrire un programme qui affiche tous les couples (N, M) de nombres amis tels que $0 < N < M \leq MAX$, MAX étant lu au clavier.

Indication : Les nombres amis compris entre 1 et 100000 sont (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368), (10744, 10856), (12285, 14595), (17296, 18416), (66928, 66992), (67095, 71145), (63020, 76084), (69615, 87633) et (79750, 88730).

Exercice 4 : Tri par insertion séquentielle

Soit $A[1..N]$ un vecteur de N entiers relatifs quelconques, l'objectif est de trier le vecteur A. Le vecteur A est trié si $A[1] \leq A[2] \leq \cdots \leq A[N]$.

Le tri utilisé est le tri par insertion séquentielle. C'est un tri en $(N-1)$ étapes. L'étape i consiste à placer le $(i+1)^{\text{e}}$ élément du vecteur à sa place dans le sous-vecteur $A(1..i+1)$ sachant que $A(1..i)$ a été trié par les étapes précédentes. Dans le cas du tri par insertion séquentielle,

la recherche de la position d'insertion, se fait séquentiellement en comparant successivement l'élément à insérer aux i premiers éléments du vecteur.
Exemple : Voici les différentes valeurs du vecteur 8 2 9 5 1 7 après chaque étape (la partie encadrée correspond à la partie du vecteur déjà traitée et donc triée) :

vecteur initial	:	$\boxed{8}\ \underline{2}\ 9\ 5\ 1\ 7$
après l'étape 1	:	$\boxed{2\ 8}\ \underline{9}\ 5\ 1\ 7$
après l'étape 2	:	$\boxed{2\ 8\ 9}\ \underline{5}\ 1\ 7$
après l'étape 3	:	$\boxed{2\ 5\ 8\ 9}\ \underline{1}\ 7$
après l'étape 4	:	$\boxed{1\ 2\ 5\ 8\ 9}\ \underline{7}$
après l'étape 5	:	$\boxed{1\ 2\ 5\ 7\ 8\ 9}$

4.1 Implanter l'algorithme du tri par insertion.
4.2 En conservant le principe du tri par insertion, expliquer comment améliorer l'efficacité de cet algorithme.

Exercice 5 : Tri par sélection
Soit $A[1..N]$ un vecteur de N entiers relatifs quelconques, l'objectif est de trier le vecteur A en utilisant le tri par sélection. Le vecteur A est trié si $A[1] \leq A[2] \leq \cdots \leq A[N]$.

Le tri par sélection est un tri en $(N-1)$ étapes. L'étape i consiste à ranger à sa place le i^{e} plus petit élément du vecteur.
Exemple : Voici les différentes valeurs du vecteur 8 2 9 5 1 7 après chaque étape (la partie encadrée correspond à la partie du vecteur déjà traitée et donc triée) :

vecteur initial	:	$8\ 2\ 9\ 5\ \underline{1}\ 7$
après l'étape 1	:	$\boxed{1}\ \underline{2}\ 9\ 5\ 8\ 7$
après l'étape 2	:	$\boxed{1\ 2}\ 9\ \underline{5}\ 8\ 7$
après l'étape 3	:	$\boxed{1\ 2\ 5}\ 9\ 8\ \underline{7}$
après l'étape 4	:	$\boxed{1\ 2\ 5\ 7}\ \underline{8}\ 9$
après l'étape 5	:	$\boxed{1\ 2\ 5\ 7\ 8\ 9}$

5.1 Implanter l'algorithme du tri par sélection.

On écrira une seule méthode principale dans la classe `TriSelectionMonolithique`.

Exercice 6 : Jeu du devin
Le jeu du devin se joue à deux joueurs. Le premier joueur choisit un nombre compris entre 1 et 999. Le second doit le trouver en un minimum d'essais. À chaque proposition, le premier joueur indique si le nombre proposé est plus grand ou plus petit que le nombre à trouver. En fin de partie, le nombre d'essais est donné.
Indication : On suppose qu'il existe une fonction qui permet d'obtenir un nombre aléatoire compris entre 1 et 999.
6.1 *La machine fait deviner le nombre.* Écrire un programme dans lequel la machine choisit un nombre et le fait deviner à l'utilisateur.

6.2 *La machine joue.* Écrire un programme dans lequel l'utilisateur choisit un nombre et la machine doit le trouver. Pour chaque nombre proposé, l'utilisateur indique s'il est trop petit (`'p'` ou `'P'`), trop grand (`'g'` ou `'G'`) ou trouvé (`'t'`, `'T'`).
Indication : On utilisera une recherche par dichotomie pour trouver le nombre.
6.3 *Le programme complet.* Écrire le programme de jeu qui donne le choix à l'utilisateur entre deviner ou faire deviner le nombre.
6.4 *On peut recommencer.* Compléter le programme précédent pour que l'ordinateur propose de faire une nouvelle partie lorsque la précédente est terminée.