Université de Lorraine, site de Nancy, licence | **Informatique**
Mathématiques

algorithmique et programmation 2 : TP numéro 1 (fonctions et procédures)

responsable de l'UE : Jean Lieber, année scolaire 2013-2014

Cet énoncé est commun aux étudiants des deux licences informatique et mathématiques. Le langage de programmation utilisé est le C pour les premiers et Python 3 pour les seconds. On rappelle qu'un aide-mémoire sur C et Python 3 est disponible à l'adresse `homepages.loria.fr/JLieber/cours/ap2/`.

Ex. 1 : Traduire un algorithme sur les entiers naturels en un programme. On considère l'algorithme suivant :

Fonction `dpn` (n : `entier naturel`) : `entier naturel`
Variables
  d : `entier naturel`
Début
  Si `est_nul`(n) **Alors**
    retourner `succ`(`zéro`)
  Finsi
  $d \leftarrow$ `dpn`(`préc`(n))
  retourner `plus`(d, d)
Fin

Écrivez un programme qui implante cette fonction et qui la teste, à l'aide de deux procédures (de noms respectifs `test_dpn_param` et `test_dpn`).

Que fait ce programme ? Répondez à cette question dans un commentaire du programme.

Ex. 2 : Traduire un algorithme sur les entiers naturels en un programme. On considère l'algorithme suivant :

Procédure `afficher_entiers_jusqu_à` (n : `entier naturel`)
Début
  `afficher_entiers_intervalle`(`zéro`, n)
Fin

Procédure `afficher_entiers_intervalle` (a : `entier naturel`, b : `entier naturel`)
Début
  Si $a > b$ **Alors**
    retourner
  Finsi
  `afficher`(a)
  `afficher_entiers_intervalle`(`succ`(a), b)
Fin

Écrivez un programme qui implante ces procédures et qui les teste, à l'aide de deux nouvelles procédures (de noms respectifs `test_afficher_entiers_jusqu_a_param` et `test_afficher_entiers_jusqu_a`).

Ex. 3 : Traduire un algorithme sur les tableaux en un programme. On considère l'algorithme suivant :

Fonction `est_trié` (T : `tableau de réel`[n]) : `booléen`
Variables
  i : `entier naturel`
Début
  $i \leftarrow 0$
  Tant que $i < n - 1$ **et** $T[i] \leqslant T[i+1]$ **Faire**
    $i \leftarrow i + 1$
  Fintantque
  retourner $i \geqslant n - 1$
Fin

Écrivez un programme qui implante cette fonction et qui la teste, à l'aide de deux procédures (de noms respectifs `test_est_trie_param` et `test_est_trie`).

Ex. 4 : Traduire un jeu d'axiomes en programme récursif. La fonction `somme_cubes` est la fonction de profil `somme_cubes : entier naturel` $\longrightarrow$ `entier naturel` définie par le jeu d'axiomes suivant :

[1] $\texttt{somme\_cubes}(\texttt{zéro}) = \texttt{zéro}$ (ou $\texttt{somme\_cubes}(0) = 0$)

[2] $\texttt{somme\_cubes}(\texttt{succ}(x)) = \texttt{plus}(\texttt{puissance}(\texttt{succ}(x), \texttt{succ}(\texttt{succ}(\texttt{succ}(\texttt{zéro})))), \texttt{somme\_cubes}(x))$
(ou $\texttt{somme\_cubes}(x+1) = (x+1)^3 + \texttt{somme\_cubes}(x)$)

Q1 Traduisez ce jeu d'axiomes en programme récursif (vous pouvez, dans un premier temps, le traduire en algorithme récursif sur le papier).
Q2 Testez ce programme (par deux procédures).
Q3 Modifiez le programme pour que l'exécution de `somme_cubes` soit « tracée », ce qui signifie que l'appel à $\texttt{somme\_cubes}(n)$ doit être affiché ainsi que son résultat. Ainsi, $\texttt{somme\_cubes}(3)$ doit donner :

```
appel à somme_cubes(3)
 appel à somme_cubes(2)
  appel à somme_cubes(1)
   appel à somme_cubes(0)
   somme_cubes(0) retourne 0
  somme_cubes(1) retourne 1
 somme_cubes(2) retourne 9
somme_cubes(3) retourne 36
```

L'indentation de l'affichage (espaces en début de lignes) est optionnelle.
Q4 On veut vérifier l'égalité suivante (pour tout entier naturel n) :

$$\texttt{somme\_cubes}(n) = \left(\frac{n(n+1)}{2}\right)^2$$

Pour cela, on doit faire une preuve, mais pour se convaincre que cela vaut le coup de faire cette preuve, on peut tester cette égalité pour un nombre fini de valeurs.

Écrivez un programme qui teste cette égalité pour les valeurs de n entre 0 et 100.

Ex. 5 : Traduire un jeu d'axiomes en programme récursif et itératif. La suite de Fibonacci est la fonction de profil `fibo : entier naturel` $\longrightarrow$ `entier naturel` définie par le jeu d'axiomes suivant :

[1] $\texttt{fibo}(\texttt{zéro}) = \texttt{succ}(\texttt{zéro})$ (ou $\texttt{fibo}(0) = 1$)

[2] $\texttt{fibo}(\texttt{succ}(\texttt{zéro})) = \texttt{succ}(\texttt{zéro})$ (ou $\texttt{fibo}(1) = 1$)

[3] $\texttt{fibo}(\texttt{succ}(\texttt{succ}(x))) = \texttt{fibo}(x) + \texttt{fibo}(\texttt{succ}(x))$ (ou $\texttt{fibo}(x+2) = \texttt{fibo}(x) + \texttt{fibo}(x+1)$)

Q1 Traduisez ce jeu d'axiomes en programme récursif (vous pouvez, dans un premier temps, le traduire en algorithme récursif sur le papier).
Q2 Testez ce programme (par deux procédures).
Q3 Modifiez le programme précédent pour qu'à chaque appel à la fonction `fibo` soit affichée la valeur des paramètres. Ainsi, $\texttt{fibo}(2)$ doit donner l'affichage :

```
fibo(2)
fibo(0)
fibo(1)
```

Testez alors le programme pour le paramètre 5.
Que constatez-vous ?
Q4 Écrivez un nouveau programme pour `fibo` qui remédie au problème mis en évidence à la question précédente.
On demande que ce programme soit itératif.
Q5 (faculative) Écrivez un programme récursif pour `fibo` tel que l'exécution de $\texttt{fibo}(n)$ fasse appel à moins de n appels récursifs.