TP – Variables et Fonctions
ING1 – Programmation C
Génie Mathématiques

EISTI

1 Saisie d'entier

(1) **Saisie d'un nombre**
Traduisez l'algorithme 1. Le but dans cet algorithme est de réaliser une fonction qui effectue la saisie d'un nombre entier. Si la saisie est immédiate en algorithmique, elle n'est pas triviale en C. En effet, nous devons pour chaque saisie, réaliser un contrôle de validité de saisie. C'est le rôle ici, de la fonction saisieEntier. Cette fonction doit donc retourner un entier valide. Si l'utilisateur n'entre pas un nombre correct, on utilisera la fonction `exit` pour quitter le programme. Sans ce cas, avant de quitter le programme, on indiquera à l'utilisateur pourquoi on quitte le programme. □

Attention en C, d'après les normes de programmation toutes les fonctions doivent être explicitement typé. Il faudra donc mettre en paramètre de la fonction `saisieEntier`, uniquement lors de la déclaration, `void` ⇔ `int saisieEntier(void)`.

Algorithme 1 Saisie sûre d'un nombre

```
fonction saisieEntier(): entier
    n: entier
    lire(n)
    retourner (n)
fin fonction
programme Saisie
    nb: entier
    nb ← saisieEntier()
    écrire (nb)
fin programme
```

2 Portée d'une variable

(2) Traduisez l'algorithme 2. Le but de cet exercice est de vous montrer que les variables ont une durée de vie, et que celles-ci ne s'étendent pas en dehors du bloc où elles sont déclarées. Vous allez devoir entrer deux entiers, entrez des valeurs différentes pour chacune des entrées, enfin observez ce qu'affiche le programme. Vous remarquerez alors que ce n'est pas parce que les variables portent le même nom, que le programme les reconnaît comme identiques. □

Algorithme 2 Portée d'une variable

```
procedure bidon()
    n: entier
    écrire (n)
    lire(n)
    écrire (n)
fin procedure
programme Saisie
    n: entier
    écrire (n)
    lire(n)
    écrire (n)
    bidon()
    écrire (n)
fin programme
```

3 Problème d'appel de fonction

(3) Traduisez l'algorithme 3. Le but de cet exercice est de vous montrer que par défaut les paramètres en C sont passés par valeurs, et non pas par référence. Ce pose évidement quelques problèmes dont le problème suivant. Nous verrons par la suite comment résoudre ce problème. □

Algorithme 3 Échange de valeurs

```
procedure echange(nb1, nb2: entier)
    tmp: entier
    écrire ("Avant échange")
    écrire ("Nb1 = ", nb1)
    écrire ("Nb2 = ", nb2)
    tmp ← nb1
    nb1 ← nb2
    nb2 ← tmp
    écrire ("Après échange")
    écrire ("Nb1 = ", nb1)
    écrire ("Nb2 = ", nb2)
fin procedure
programme Toto
    nb1, nb2: entier
    nb1 ← 5
    nb2 ← 7
    écrire ("Avant appel de la fonction")
    écrire ("Nb1 = ", nb1)
    écrire ("Nb2 = ", nb2)
    écrire ("Appel de la fonction echange")
    echange(nb1, nb2)
    écrire ("Après appel de la fonction")
    écrire ("Nb1 = ", nb1)
    écrire ("Nb2 = ", nb2)
fin programme
```

4 Programme un peu plus grand

(4) Écrivez un programme qui propose, via un menu, les actions suivantes (les prototypes de fonctions/procédures vous sont donnés):

1. Affichage d'un triangle : `void affichageTriangle(int n);`
2. Table de multiplication : `void tableMultiplication(int n);`
3. Nombre d'Amstrong : `void estAmstrong(int n);` □

Affichage d'un triangle

Cette procédure affiche un triangle isocèle sur le terminal en ne se servant que des caractères '*' et ' '(espace). Voici un exemple pour $n = 4$:

```
   *
  ***
 *****
*******
```

Table de multiplication

Cette procédure doit afficher la table de multiplication de l'entier donné en paramètre. Par exemple, pour 7, on obtiendrait :

```
Table de multiplication de 7 :
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

Nombre d'Amstrong

Cette procédure doit définir si un nombre (saisi par l'utilisateur) est un nombre d'Amstrong. C'est-à-dire que ce nombre est égal à la somme des cubes des chiffres qui le composent.
Exemple : 153 : $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$
Quelques nombres d'Amstrong : 1, 153, 370, 371, 407.