TD Informatique 07 : Fonctions et Procédures

Consignes générales : N'oubliez pas pour ce TD comme pour les suivants de vous créer un répertoire consacré au TD et d'enregister vos codes desssus.

On rappelle que les commandes à taper dans le terminal pour compiler puis éxécuter votre programme C :
— Pour compiler : *gcc -o nom_executable nom_programme.c*
— Pour exécuter : *./nom_executable*
Il est conseillé de toujours écrire l'algorithme en pseudo-code avant de passer sur machine.

Exercice 1

Que vont afficher les programmes suivants ?

1. Mystère 1

```
PROGRAMME mystere1
PROCEDURE display( x , y  : entier)
DEBUT
   SI ( x > y ) ALORS
      ÉcriRE( x )
   SINON
      SI ( x < y ) ALORS
         ÉcriRE( y )
      SINON
         ÉcriRE(« Egalité »)
      FIN DE SI
   FIN DE SI
FIN

PROCEDURE process( a , b , c , d : entier)
DEBUT
   ÉcriRE( "calcul" )
   display( a , b )
   display( c , d )
FIN

VARIABLE
   n1 , n2 , n3 , n4 : entier
DEBUT
   ÉcriRE( "Démarrage" )
   n1 ← 4
   n2 ← 9
   n3 ← 6
   n4 ← 5
   process( n1 , n2 , n3 , n4 )
   ÉcriRE( "fin" )
FIN
```

2. Mystère 2

```
PROGRAMME Mystere2
FONCTION fonction1 ( x ,y :entier) :entier
VARIABLE
   m : entier
DEBUT
   m ← ( x + y ) DIV 2
   RETOURNE m
FIN

FONCTION fonction2 (a,b,c,d : entier) : entier
VARIABLE
   m1 , m2 , res : entier
DEBUT
   m1 ← fonction1( a , b )
   m2 ← fonction1( c , d )
   res ← m1
   SI ( m2 > m1 ) ALORS
      res ← m2
   FIN SI
   RETOURNE res
FIN


VARIABLE
   n1 , n2 , n3 , n4 : entier
DEBUT
   n1 ← 9
   n2 ← 6
   n3 ← 3
   n4 ← 1
   ÉcriRE( fonction2( n1 , n2 , n3 , n4 ) )
FIN
```

Exercice 2 (*Nombres premiers ?*)

1. Écrire une fonction `prime` qui prend en paramètre un entier `nb`. Cette fonction retourne un booléen qui vaut VRAI si nb est un nombre premier, FAUX sinon.
2. Dans le programme principal, demander à l'utilisateur de saisir un nombre entier `N` strictement plus grand que 1.
3. Afficher tout les nombres premiers entre 2 et `N` à l'aide de la fonction `prime`
4. Traduire en C et tester l'algorithme sur machine.
5. Modifier ce programme pour que le programme affiche les `N` premiers nombres premiers (attention à ne pas prendre `N` trop grand !)

Exercice 3 (*Deviner le nombre*)

Nous allons coder un petit jeu où l'utilisateur doit deviner un nombre secret. L'utilisateur a 10 tentatives pour trouver le nombre secret : à chaque tentative il peut faire une nouvelle proposition. Le jeu s'arrete si le nombre secret a été trouvé (condition de victoire) ou si les 10 tentatives ont été utilisées (condition de défaite).

1. Dans le programme principal, générer un entier aléatoire entre 0 et 100 compris.
2. Écrire une procédure `CompareNumber` qui prends deux entiers a et b en paramètre. Cette procédure compare les deux paramètres et affiche :
   — "Plus grand" si $a < b$
   — "Plus petit" si $a > b$
   — "Trouvé !" si $a = b$
3. Écrire une fonction `guessSecret` qui prend en paramètre le nombre secret à deviner. Cette fonction va demander à l'utilisateur de saisir une valeur entière et, grace à la procédure écrite précédement va indiquer la comparaison entre les deux nombres. La fonction retourne 1 si l'utilisateur a trouvé le nombre secret et 0 sinon.
4. Écrire le reste du programme principal pour que l'utilisateur puisse jouer au jeu. Afficher à la fin s'il a gagné ou perdu.

Exercice 4 (*Somme de nombres successifs*)

1. Écrire une fonction qui prend deux valeurs entières a et b et qui retourne la somme de tous les nombres entiers se trouvant entre a et b inclus (exemple : $a = 7$ et $b = 10$ va retourner $7 + 8 + 9 + 10 = 34$).
2. Ecrire le programme principal permettant de tester cette fonction avec des valeurs saisies.

Exercice 5 (*Moyenne de N notes*)

1. Ecrire une fonction `getNote` qui ne prend aucun paramètre et qui demande à l'utilisateur de rentrer une valeur entière entre 0 et 20 inclus. Si la valeur saisie n'est pas dans l'intervalle, la fonction continue de demander en boucle de rentrer une valeur correcte. La fonction retournera la valeur saisie par l'utilisateur à partir du moment où cette dernière se situe dans l'intervalle 0-20.
2. Ecrire une fonction `average` qui prend 1 valeur entière N en paramètre. Cette fonction va demander à l'utilisateur de saisir N notes entières les unes après les autres. Pour ce faire, elle appellera en boucle la fonction `getNote` codée précédemment puis calculera la moyenne de toutes les notes et retournera le résultat du calcul.
3. Ecrire un programme principal dans lequel on va définir une valeur aléatoire comprise entre 5 et 10. Cette variable servira à indiquer le nombre de notes à saisir. Tester la fonction `average`. (on rappelle que la fonction aléa(N) peut être utilisée en pseudo-code pour générer un entier aléatoire entre 0 et N).

Exercice 6 (*maximum*)

1. Écrire une fonction `max2` qui prend en paramètre deux entiers a et b et qui retourne le plus grand des deux.
2. Écrire une fonction `max4` qui prend en paramètre quatre entiers a, b,c,d et qui retourne le plus grand des quatre. *Cette fonction ne doit pas utiliser de SI !*
3. Tester l'algorithme avec un programme principal.

Exercice 7 (*printf*) `printf` n'est pas une procédure mais une fonction (elle retourne donc quelque chose). Nous allons voir ce que représente cette valeur retournée et comment l'utiliser.

1. Voici un extrait de la documentation de la fonction printf
   `https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm` :

   ```
   Description
   The C library function printf() sends formatted output to stdout.

   Declaration
   int printf(const char *format, ...);

   Parameters
    format : This is the string that contains the text to be written to stdout. It can optionally contain
   embedded format tags that are replaced by the values specified in subsequent additional arguments and
   formatted as requested. Format tags prototype is %[flags][width][.precision][length]specifier

   Return Value
   If successful, the total number of characters written is returned. On failure, a negative number is
   returned.
   ```

   Que retourne la fonction `printf` ?

2. Soit le code suivant :

   ```c
   #include <stdio.h>

   int main(){
           int a=0;
           a=printf(??);
           printf("\na=%d\n",a);
           return 0;
   }
   ```

   Remplacer `??` pour que le programme affiche (entre autre) "a=3".

Exercice 8 (*scanf*) `scanf` n'est pas une procédure mais une fonction (elle retourne donc quelque chose). Nous allons voir ce que représente cette valeur retournée et comment l'utiliser.

1. Voici un extrait de la documentation de la fonction scanf
   `https://www.tutorialspoint.com/c_standard_library/c_function_scanf.htm` :

   ```
   Description
   The C library function scanf() reads formatted input from stdin.
   ```

```
Declaration
int scanf(const char *format, ...);

Parameters
 format : This is the C string that contains one or more of the following items.Whitespace character,
Non-whitespace character and Format specifiers. A format specifier will be like [=%[*][width][modifiers]

Return Value
On success, the function returns the number of items of the argument list successfully read.
```

Que retourne la fonction`scanf` ?

2. Soit le code suivant :

```
#include <stdio.h>

int main(){
        int a;
        int b;
        a=scanf("%d",&b);
        printf("\na=%d\n ",a);
        return 0;
}
```

Que fait-il faire que pour le programme affiche à l'execution "a=1" ? "a=0" ?

3. Écrire une fonction `askUserInt(int min, int max)` qui va servir à retourner une valeure entière saisie par l'utilisateur qui doit se trouver entre min et max compris et également à vérifier la validité de la saisie. On redemandera la saisie en cas d'erreur.

Conseil : vous pouvez (devez même) toujours tester le retour du scanf lors de la saisie. Vous pouvez donc garder cette dernière fonction écrite et la réutiliser à chaque TD !

Exercice 9 (*Jeu de dés*)

On souhaite écrire un algorithme pour simuler un jeu de lancer de dés à deux joueurs. Voici les règles :

— A chaque tour les deux joueurs peuvent choisir de lancer un ou deux dés. Le score est la somme cumulée des lancers de tous les tours de jeu.
— Avant de lancer le dé, le joueur choisit s'il veut lancer 1 ou 2 dés. S'il choisit de lancer deux dés, son score augmentera du résultat de la somme des deux dés. Cependant si les deux dés ont le même résultat son score sera **diminué** du résultat de la somme des dés.
— Si à l'issue d'un tour, un joueur a le score de 30 ou au-dessus, il remporte la partie.

1. Comment simuler un lancer de dés en informatique ?
2. Écrire une fonction `throwDice()` permettant de communiquer au reste du programme le résultat d'un lancer de dés.
3. Écrire une fonction `playerTurn()` qui correspond au tour d'un seul joueur. Cette fonction va renvoyer l'augmentation ou la diminution du score du joueur à l'issue de ce tour. Elle doit :
   — Afficher le message `"Voulez-vous lancer 1 ou 2 dés ?"` et récupérer la réponse de l'utilisateur. Il faudra s'assurer que l'utilisateur donne une valeur cohérente avant de passer à la suite.
   — En fonction du choix du joueur simuler le lancer de dés en utilisant la fonction `throwDice()`.
   — En fonction du résultat obtenu (cf les règles du jeu), renvoyer au reste du programme l'évolution du score du joueur.
4. Dans le programme principal, écrire l'algorithme permettant le déroulement du jeu :
   — Définir et initialiser les variables `score1` et `score2`, respectivement les scores du joueur 1 et 2.
   — Le jeu continue jusqu'à ce que le score d'un joueur atteigne 30 à la fin d'un tour. Lors d'un tour, le joueur 1 joue puis le joueur 2 et le score de chaque joueur est affiché. Écrire l'algorithme permettant le déroulement complet d'une partie.
   — Lorsque la partie se termine, afficher le joueur gagnant. Il est possible d'obtenir un match nul si les deux joueurs ont un score dépassant 30 à la fin du dernier tour.