SERIE D'EXERCICES SUR LES PROCEDURES ET LES FONCTIONS
Niveau 1ère année tranc commun MI

Exercice 1 (TD et TP)

Un nombre parfait est un entier positif supérieur à 1, égal à la somme de ses diviseurs ; on ne compte pas comme diviseur le nombre lui-même.
Exemple : 6 est un nombre parfait puisque : 6 = 3 + 2 + 1.

- Ecrire une procédure qui confirme si un entier est un nombre parfait ou nom.
- Utiliser cette procédure pour chercher les nombres parfaits entre 1 et 10000.

Exercice 2 (TD et TP)

Ecrire une procédure qui prend pour arguments un entier n (la taille du tableau), le tableau de réels T et affiche le plus grand élément du tableau T ainsi que sa position dans le tableau.

Exercice 3 (TD et TP)

Deux nombres sont opposés si leur somme est égale à 0. Deux nombres sont inverses si leur produit est égal à 1. Ecrire deux procédures qui testent si deux nombres sont opposés ou non et si sont inverses ou non.

Exercice 4 (A DOMICILE)

Ecrire une procédure qui permet d'entrer deux valeurs M et N et d'afficher toutes les valeurs pairs entre M et N.

Exercice 5 (TD et TP)

Ecrire un programme qui calcule la partie entière d'un nombre positif à l'aide d'une fonction.

Exercice 6 (TD et TP)

Ecrire un programme (Algo + C) qui permet de synthétiser un tableau de N valeurs réels (N est déterminé par l'utilisateur). Cette synthèse se repose sur l'utilisation des fonctions suivantes :

- La fonction SommeTab (N,Tab) qui retourne la somme des N valeurs réelles stockées dans le tableau Tab.
- La fonction MoyenneTab (N,Tab) qui retourne la moyenne des N valeurs réelles stockées dans le tableau Tab. Cette fonction devra impérativement utiliser la fonction SommeTab écrite précédemment.
- Les fonctions MaxTab(N,Tab) et MinTab(N,Tab) qui retournent respectivement la valeur maximale et la valeur minimale des N valeurs réelles stockées dans le tableau Tab.

Exercice 7 (A DOMICILE)

Ecrire une fonction (Algo + C) qui calcule le factoriel d'un entier afin de l'utiliser pour calculer la somme de la série : 1/1 ! + 2/2 !+……+ n/n ! (n est déterminé par l'utilisateur)

Exercice 8 (TD et TP)

Soit le programme suivant :

```
#include <stdio.h>
void somcar(int a,int b, int z)
{
z=a*a+b*b;
return ;
}
```

```
int main()
{
int x,y,R;
printf ("donner x:");
scanf("%d",&x);
printf ("donner y:");
scanf("%d",&y);
r=0;
somcar(x,y,R);
printf ("la somme de %d^2 + %d^2 = %d \n",x,y,R);
return 0;
}
```

1- Ecrire son algorithme
2- Donner la valeur de **R** après l'exécution de ce programme pour les valeurs suivantes : x=3, y=4
- Qu'est-ce que vous avez remarquez ? pourquoi ?
- Comment peut-on remédier ce problème
3- Substituer cette procédure par une fonction.

Exercice 9 (TD et TP)
On demande d'écrire un programme avec une fonction qui transforme un nombre fourni par L'utilisateur dans une zone de saisie en sa valeur absolue (transformation réelle).

Exercice 10 (TD et TP)

La fonction d'Ackermann est la fonction récursive définie par :

- A(0,n) = n+1;
- $m \geq 1$, A(m,0) = A(m-1,1);
- $m \geq 1$, $n \geq 1$, A(m,n) = A(m-1,A(m,n-1)).

Ecrire un algorithme pour cette fonction et un programme C qui utilise cette fonction pour 1 <= m <= 2 et 1 <= n <= 10

Exercice 11 (TD et TP)

La suite de Fibonacci est définie par les formules suivantes :

- Fib (0) = 0
- Fib (1) = 1
- Fib (n) = Fib (n−2) + Fib (n−1) pour $n \geq 2$.

Ecrire un algorithme pour cette fonction et un programme C qui utilise cette fonction pour calculer n'importe quel terme.

Exo1

```
Algorithme Nombre_parfait
Variables
  x : entier
Procédure Parfait (a : entier)
Variables
  i, somme : entier ;
Début
    Somme ← 0
    Pour i Allant de 1 A a-1 Faire
        Si (a mod i)=0 Alors
        somme ←somme+i
    Fin si
    Fin Pour
    Si (somme = a) Alors
        Ecrire ( a,″ est un nombre parfait″)
    FinSi
Fin Procédure
Début
Pour x Allant de 1 A 10000 Faire
   Parfait(x)
Fin Pour
Fin
```

```
#include <stdio.h>
void parfait(int a)
{
int i,somme;
somme=0;
for (i=1;i<a;i++)
if (a%i==0)
somme=somme+i;
if (somme==a)
printf ("le nombre %d est parfait \n",a);
return ;
}
int main()
{
int x;
for (x=1;x<=10000;x++)
parfait(x);
return 0;
}
```

Exo2

```
Algorithme maximum
Variables
    i,a : entier
    X : tableau [] de réel
Procédure Max_tab (N : entier, TAB :tableau [ ] de réel)
Variables
i,indice : entier ;
max : réel
Début
  max ← TAB[0]
  indice ← 0
Pour i=0 À N FAIRE
    Si (TAB[i] >max) Alors
      max ← TAB[i]
      indice ← i
    Fin si
Fin pour
   Ecrire (‘Le max =’, max , ‘Avec l’indice =’,i)
Fin Procédure
Début
   Ecrire (’Donner la dimension de votre tableau : ‘)
   Lire (a)
Pour i Allant de 0 A a-1 Faire
  Lire (X[i])
Fin Pour
Max_tab(a, X)
Fin
```

```
#include <stdio.h>
void Max_tableau (int n, float tab[])
{
int i, indice;
float max;
max=tab[0]; indice=0;
for (i=0;i<n;i++)
printf("%.2f \t",tab[i]);
printf("\n");
for (i=1;i<n;i++)
if (tab[i]>max)
{max=tab[i]; indice=i;}
printf("le Max = %.2f et sa position = %d
\n",max,indice+1);
return ;
}
int main()
{
int i,a;
printf(" Donner la dimension du tableau: ");
scanf("%d",&a);
float X[a];
printf("Introduiser les elements du tableau: \n ");
for (i=0;i<a;i++)
{ printf("tab[%d]:",i+1);
scanf("%f",&X[i]);}
printf("\n");
Max_tableau(a,X);
return 0;
}
```

Exo 3

```
Algorithme test_nombre
Variables
  x,y : entier
Procédure Oppose (a : entier,  b:entier)
Début
    Si (a + b = 0) Alors
       Ecrire (a,' et', b,' sont opposés)
    Fin si
Fin Procédure
Procédure Inverse (a : entier, b : entier)
Début
    Si (a * b = 1) Alors
       Ecrire (a,' et', b,' sont inverses)
    Fin si
Fin Procédure
Début
      Ecrire ('Donner x :')
      Lire (x)
      Ecrire ('Donner y :')
      Lire (y)
      Oppose (x, y)
      Inverse (x, y)
Fin
```

Exo 5

Algorithme Partie_Entiere
Variables
a : réel
Fonction entiere (x : reel) : entier
Variable
y : entier
Debut
y← 0 ;
Tantque (y < x) **faire**
y←y + 1
FinTanque
y ← y-1
return y
Fin fonction
Début
Lire (a)
Ecrire (entiere (a))
Fin
	#include <stdio.h>
	int entiere (float x)
	{
	int y;
	y=0;
	while (y<x)
	y=y+1;
	y=y-1;
	return y;
	}
	int main()
	{
	float a;
	printf ("donner a:");
	scanf("%f",&a);
	printf ("la partie entiere de %f = %d \n",a ,
	entiere (a));
	return 0;
	}

Exo 6

```
Algorithme Synthese_TAB
Variables
Dim, i : entier
X : tableau [] de réel
Fonction SommeTab (N : entier, TAB :tableau [ ] de réel) : entier
Variables
    i : entier ;
    somme : réel
Début
  Somme ← 0
  Pour i=0 À N-1 FAIRE
     somme ← somme + TAB[i]
  Fin pour
  return somme
Fin Fonction
Fonction MoyenneTab (N : entier, TAB :tableau [ ] de réel) : entier
Début
  return (SommeTab(N,TAB)/N)
Fin Fonction
Fonction MaxTab (N : entier, TAB :tableau [ ] de réel) : entier
Variables
    i : entier ;
    Max : réel
Début
  Max ← TAB[0]
  Pour i=1 À N-1 FAIRE
     Si (TAB [ i ]> max ) Alors
       Max ← TAB[i]
     Fin Si
  Fin pour
return Max
Fin Fonction
Fonction MinTab (N : entier, TAB :tableau [ ] de réel) : entier
Variables
i : entier ;
Min : réel
Début
    Min ← TAB[0]
    Pour i=1 À N-1 FAIRE
       Si (TAB [ i ] < Min ) Alors
         Min ← TAB[i]
       Fin Si
    Fin pour
    return Min
Fin Fonction

Début
    Lire (Dim)
```

```
#include <stdio.h>
float SommeTab( int N, float Tab [ ] )
{
int i ;
float somme =0;
for ( i =0; i<N; i++)
somme += Tab [ i ] ;
return somme ;
}
float MoyenneTab( int N, float Tab [ ] )
{
return (SommeTab(N,Tab)/N);
}
float MaxTab( int N, float Tab [ ] )
{
int i ;
float max = Tab [ 0 ] ;
for ( i =1; i<N; i++)
{
if (Tab [ i ]> max )
max = Tab [ i ] ;
}
return max ;
}
float MinTab( int N, float Tab [ ] )
{
int i ;
float min = Tab [ 0 ] ;
for ( i =1; i<N; i++)
{
if (Tab [ i ]< min )
min = Tab [ i ] ;
}
return min ;
}
int main()
{
int i, Dim;
printf("Donner la dimension de votre tableau: ");
scanf("%d",&Dim);
float X[Dim];
for (i=0;i<Dim;i++) {
printf("X[%d] =", i+1);
scanf("%f",&X[i]); }
printf("La somme des elements de ce tableau =
%f\n",SommeTab (Dim,X));
printf("La moyenne des elements de ce tableau =
%f\n",MoyenneTab(Dim,X));
printf("Le Max dans ce tableau =
%f\n",MaxTab(Dim,X));
printf("Le Min dans ce tableau =
%f\n",MinTab(Dim,X));
return 0;
}
```

Pour i=0 **À** Dim-1 **FAIRE** Lire (X[i]) **Fin pour** Ecrire (‘La somme = ‘, SommeTab(Dim,X)) Ecrire (‘La Moyenne = ‘, MoyenneTab(Dim,X)) Ecrire (‘Le Max = ‘, MaxTab(Dim,X)) Ecrire (‘Le Min = ‘, MinTab(Dim,X)) **Fin**	

Exo 8

Le passage du programme vers son algorithme peut être comme suit :

Algorithme Somme_carre
Variables
x,y,R :entier
Procédure somcar (a : entier, b : entier, z : entier)
Début
z=a*a+b*b
Fin Procédure
Début
Ecrire (″Donner x : ″)
Lier (x)
Ecrire (″Donner y : ″)
Lire (y)
r←0
somcar(x,y,R)
Ecrire (R)
Fin

- Pour x=3 et y=4 on trouve que R =0 ? Parce que le mode de passage utilisé dans ce programme est un mode de passage par valeur.
- Nous pouvons remédier ce Pb par l’utilisation du mode de passage par adresse (par variable). Dans ce cas la procédure doit être déclarée dans l’algorithme comme suit :

Procédure somcar (a : entier, b : entier, **var** z : entier)
Et le programme doit être changé comme suit :

```
#include <stdio.h>
void somcar (int a,int b, int *z) /* z doit être précédé par * pour désigner une variable */
/* adresse (Pointeur) */
{
*z=a*a+b*b;
return ;
}
int main()
{
int x,y,R;
printf ("donner x:");
scanf("%d",&x);
printf ("donner y:");
scanf("%d",&y);
R=0;
```

somcar(x,y, &R); /* R doit être précédé par & pour désigner qu'on utilise l'adresse de R */
printf ("la somme de %d^2 + %d^2 = %d \n",x,y,R);
return 0;
}

- Nous pouvons substituer cette procédure par une fonction comme suit :

Fonction somcar (a : entier, b : entier) : entier
Variables z :entier
Début
z=a*a+b*b
Fin Fonction
int somcar (int a , intb)
{int z
z=a*a+b*b;
return z;
}

Exo 9
Algorithme val_absolue
Variables
x:entier
Fonction absolue (**Var** a : entier) : entier /* **Var** pour désigner le passage par variable (adresse)
Début
Si (a<0) **alors**
a← -a
Fin si
Fin fonction
Début
Ecrire (″Donner x : ″)
Lier (x)
x=absolue(x)
ecrire (‘ la valeur absolut de x=’,x)
Fin
Programme C

```
#include <stdio.h>
int absolut(int *a)
{
if (*a<0)
*a=-*a;
return *a;
}
int main()
{
int x;
printf ("donner x:");
scanf("%d",&x);
x=absolut(&x);
printf ("la valeur absolut de x = %d\n",x);
return 0;
}
```

Exo 10

```
Fonction Ackermann (n, m: entier): entier
Var
résultat: entier;
Début
Si (m = 0) Alors
résultat := n + 1;
Sinon
Si (n = 0) Alors
résultat := Ackermann(m - 1, 1)
Sinon
résultat := Ackermann(m - 1,
Ackermann(m, n - 1));
FinSi
Return résultat;
Fin
```

```
#include <stdio.h>
#include <stdlib.h>
int Ackermann (int M,int N)
{
if(M == 0) return N+1;
else
{
if(N == 0) return Ackermann(M-1,1);
else return Ackermann(M-1,(Ackermann(M,N-1)));
}
}
int main()
{
int I,J;
for(I=1;I<=2;I++) for(J=1;J<=10;J++) {
printf("Ackermann(%d,%d)=%d\n",I,J,Ackermann(I,J));
}
return 0;
}
```

Exo 11

```
Fonction Fibonacci(n: entier): entier;

Var
   résultat: entier;

Debut

   Si(n = 0) Alors
      résultat := 0;
   Sinon
     Si(n = 1) Alors
      résultat := 1;
    Sinon
      résultat := Fibonacci(n - 1) + Fibonacci(n - 2)
    FinSi

   return résultat;

Fin
```

```
#include <stdio.h>

int fib(int n)
{
   if (n == 0)

            return 0;
      else
            if (n == 1)
              return 1;
            else
        return (fib (n - 1) + fib (n - 2));
}

int main()
{
   int I;
   printf("Donnez un nombre: ");
         scanf("%d",&I);
   printf("Fibonachi(%d)=%d\n",I,fib(I));

   return 0;
}
```