Exercices supplémentaires sur la récursivité

Exercice 201:

- Ecrire une procédure récursive permettant d'afficher une chaine de caractères un nombre déterminé de fois.
- Tester cette procédure dans un algorithme/programme principal. Afficher par exemple la chaîne "Bonjour" 10 fois.

Solution :

Algorithm:

```
Algorithm affichageMessage;
Procedure affichage(a:String,n:integer);
Begin
If n≠0 then
     Begin
     Write(a);
     affichage(a,n-1);
     End;
Begin
affichage("Bonjour",10);
End.
```

Programme C:

```
#include <stdio.h>
void affichage(char a[],int n){
    if(n==0);
    else {
        printf("%s",a);
        affichage(a,n-1);
    }
}
int main(){
     affichage("bonjour",10);
}
```

Exercice 202:

- Ecrire une procédure récursive permettant d'afficher les nombres entiers compris entre 10 et 19, et puis les afficher dans le sens inverse (de 19 à 10).
- Tester cette procédure dans un algorithme/programme principal.

Solution :

Algorithm:

```
Algorithm affichageNombres;
Procedure affichage(n:integer);
Begin
If n<20 then
     Begin
     Write(n);
     affichage(n+1);
```

```
     Write(n);
     End;
Begin
affichage(10);
End.
```

Programme C:

```
#include <stdio.h>
void affichage(int n){
    if(n<20) {
       printf("%d ",n);
       affichage(n+1);
       printf("%d ",n);
    }
}
int main(){
    affichage(10);
}
```

Exercice 203:

- Modifier la procédure de l'exercice précédent pour qu'elle affiche les nombres entiers compris entre deux nombres ***a*** et ***b*** passés en paramètres, et puis les afficher dans le sens inverse.
- Ecrire l'algorithme/programme principal qui lit deux nombres entiers, et qui affiche les entiers compris entre eux, en utilisant la fonction précédente.

Solution :

Algorithm:

```
Algorithm affichageNombres;
Var x,y:integer;
Procedure affichage(a,b:integer);
Begin
If a<=b then
     Begin
     Write(a);
     affichage(a+1,b);
     Write(a);
     End;
Begin
Write("Donner 2 nombres integers: ");
Read(x,y);
If x<y then affichage(x,y)
Else affichage(y,x);
End.
```

Programme C:

```
#include <stdio.h>
void affichage(int a,int b){
    if(a<=b) {
        printf("%d ",a);
        affichage(a+1,b);
        printf("%d ",a);
    }
}
int main(){
    int x,y;
    printf("Donner 2 nombres entiers: ");scanf("%d%d",&x,&y);
    if(x<y)affichage(x,y);
```

```
    else affichage(y,x);
}
```

Exercice 204 :

- Écrire une fonction récursive qui, à partir d'un entier positif ***n*** et d'un flottant ***x***, renvoie x^n.
- Ecrire l'algorithme/programme principal qui lit deux nombres, et qui calcul et affiche leur puissance. Le calcul de la puissance doit être effectué en utilisant la fonction précédente.

Solution :

Algorithm:

```
Algorithm calculPuissance;
Var n:integer;x,p:real;
Function puissance(x:real,n:integer):real;
Begin
If n=1 then puissance ← x
Else puissance ← x*puissance(x,n-1);
End;
Begin
Read(x,n);
p ← puissance(x,n);
Write(x," ^ ",n," = ",p);
End.
```

Programme C:

```
#include <stdio.h>
float puissance(float x,int n){
    if(n==1)return x;
    else return x*puissance(x,n-1);
}
int main(){
    int n ;float x;
    puts("donner 2 nombres:");
    scanf("%f%d",&x,&n);
    printf("%.2f ^ %d = %.2f\n",x,n,puissance (x,n));
}
```

Exercice 205:

- Ecrire une procédure récursive qui calcule la puissance de deux nombres entiers passés en paramètres.
- Tester cette procédure dans un algorithme/programme principal.

Solution :

Algorithm:

```
Algorithm calculPuissance;
Var a:integer;b,p:real;
Procedure puiss(a:real,b:integer;Var c:real);
Begin
If b=0 then c ← 1
Else   Begin
       puiss(a,b-1,c);
       c ← c*a;
       End;
End;
Begin
```

```
Read(a,b);
puiss(a,b,p);
Write(a," ^ ",b," = ",p);
End.
```

Programme C:

1ère solution:

```
void puiss(int a,int b,int* c){
    if(b!=0){
        *c=*c*a;
        puiss(a,b-1,c);
    }
}
main(){
    int a,b,p;
    printf("Donner 2 nombres: ");
    scanf("%d%d",&a,&b);
    p=1;
    puiss(a,b,&p);
    printf("%d ^ %d = %d",a,b,p);
}
```

2ème solution:

```
void puiss(int a,int b,int* c){
    if(b==0)*c=1;
    else{
        puiss(a,b-1,c);
        *c=*c*a;
    }
}
main(){
    int a,b,p;
    printf("Donner 2 nombres: ");
    scanf("%d%d",&a,&b);
    puiss(a,b,&p);
    printf("%d ^ %d = %d",a,b,p);
}
```

Exercice 206 :

- Ecrire une fonction récursive qui calcule et retourne la somme des ***n*** premiers carrés. Par exemple, si ***n*** vaut 3, cette fonction calculera $1^2 + 2^2 + 3^2$.
- Tester cette fonction dans un algorithme/programme principal.

Solution :

Algorithm:

```
Algorithm calculSommesCarres;
Var n,s:integer;
Function sommeNPremiers(n:integer):integer;
Begin
If n=1 then sommeNPremiers ← 1
Else sommeNPremiers ← n*n+sommeNPremiers(n-1);
End;
Begin
Write("Donner un nombre: ");
```

```
Read(n);
s ← sommeNPremiers(n);
Write("la somme des ",n," premiers carées est ",s);
End.
```

Programme C:

```
#include <stdio.h>
int sommeNPremiers(int n){
    if(n==1)return 1;
    else return n*n+sommeNPremiers(n-1);
}
int main(){
    int n,s;
    printf("Donner un nombre: ");
    scanf("%d",&n);
    s=sommeNPremiers (n);
    printf("la somme des %d premiers carees est %d",n,s);
}
```

Exercice 207 :

- Ecrire une procédure récursive qui calcule et renvoie la somme des ***n*** premiers carrés. Par exemple, si ***n*** vaut 3, cette procédure calculera $1^2 + 2^2 + 3^2$.
- Tester cette procédure dans un algorithme/programme principal.

Solution :

Algorithm:

```
Algorithm calculSommesCarres;
Var n,s:integer;
Procedure sommeNPremiers(n:integer;Var s:integer);
Begin
If n=0 then s ← 0
Else Begin
      sommeNPremiers(n-1,s);
      s ← s + n*n;
      End;
End;
Begin
Write("Donner un nombre: ");
Read(n);
sommeNPremiers(n,s);
Write("la somme des ",n," premiers carées est ",s);
End.
```

Programme C:

1ère solution:

```
#include <stdio.h>
void somme(int n,int* s){
    if(n>0){
      *s=*s+n*n;
      somme(n-1,s);
    }
}
main(){
    int n,s;
    printf("Donner un nombre entier: ");
```

```
    scanf("%d",&n);
    s=0;
    somme(n,&s);
    printf("La somme est %d",s);
}
```

2ème solution:

```
#include <stdio.h>
void somme(int n,int* s){
    if(n==0)*s=0;
    else{
      somme(n-1,s);
      *s=*s+n*n;
    }
}
main(){
    int n,s;
    printf("Donner un nombre entier: ");
    scanf("%d",&n);
    somme(n,&s);
    printf("La somme est %d",s);
}
```

3ème solution:

```
#include <stdio.h>
void somme(int n,int i,int* s){
    if(i<=n){
      *s=*s+i*i;
      somme(n,i+1,s);
    }
}
main(){
    int n,s;
    printf("Donner un nombre entier: ");
    scanf("%d",&n);
    s=0;
    somme(n,1,&s);
    printf("La somme est %d",s);
}
```

Exercice 208 :

- Ecrire une fonction récursive qui prend en entrée un nombre entier positif n et qui teste et renvoie si ses chiffres sont ordonnés en ordre croissant.
- Tester cette procédure dabs un algorithme/program principal.

Solution :

Algorithm:

```
Algorithm verification;
Var n:integer;
Function testOrdonne(n :integer;dern:integer):Boolean;
Var dern,avDern : integer;
Begin
if n >= 0 and n < 10 Then
        testOrdonne ← True
```

```
Else Begin
     dern ← n mod 10;
     avDern ← (n div 10) mod 10;
     if dern <= avDern Then testOrdonne ← False
     Else return testOrdonne ← testOrdonne(n / 10);
     End;
End ;
Begin
Read(n);
If testOrdonne(n) = True then
     write("Le chiffre d'unité est suivi de ses multiples")
Else write("Le chiffre d'unité est suivi de ses multiples");
End.
```

Programme C:

```
int testOrdonne(int n) {
    int dern,avDern;
    // Cas de base : si n est un chiffre unique, il est ordonné en ordre
croissant
    if (n >= 0 && n < 10)
        return 1;
    else{
        //Récupérer le dernier chiffre
        dern = n % 10;
        //Récupérer le chiffre précédent
        avDern = (n / 10) % 10;
        // Si le dernier chiffre est inférieur ou égal au chiffre
précédent, les chiffres ne sont pas ordonnés
        if (dern <= avDern)
            return 0;
        // Sinon, récursion pour vérifier les chiffres restants
        else return testOrdonne(n / 10);
    }
}
int main() {
    int nombre;
    printf("Entrez un nombre entier positif : ");
    scanf("%d", &nombre);
    // Appel de la fonction récursive pour tester si les chiffres sont
ordonnés en ordre croissant
    if (testOrdonne(nombre))
        printf("Les chiffres sont ordonnes en ordre croissant");
    else
        printf("Les chiffres ne sont pas ordonnes en ordre croissant");

    return 0;
}
```

Exercice 209 :

- Ecrire une fonction récursive qui prend en entrée un nombre entier positif *n* et qui teste et renvoie si le chiffre d'unité de n est suivi de son multiple.
- Tester cette procédure dabs un algorithme/program principal.

Solution :

Algorithm:

```
Algorithm verification;
Var n,dern:integer;
Function test(n :integer;dern:integer):Boolean;
Begin
If n<10 Then
     If n mod dern = 0 Then test ← True
     Else test ← False
Else test ← test(n div 10,dern);
End ;
Begin
Read(n);
dern ← n mod 10;
If test(n/10,dern) = True then
     write("Le chiffre d'unité est suivi de ses multiples")
Else write("Le chiffre d'unité est suivi de ses multiples");
End.
```

Programme C:

```
int test(int n, int dern) {
    // Si le nombre a une seule chiffre
    if (n < 10) {
        // Vérifier si le chiffre est suivi de ses multiples
        if (n % dern == 0)
            return 1;
        else
            return 0;
    }
    //Sinon,récursion pour tester le chiffre des unités du nombre réduit
    return test(n / 10, dern);
}
int main() {
    int n;
    printf("Entrez un nombre entier : ");
    scanf("%d", &n);
    // Obtenir le chiffre des unités
    int dern = n % 10;
    // Appel de la fonction récursive
    if (test(n / 10, dern))
        printf("Le chiffre d'unite est suivi de ses multiples");
    else
        printf("Le chiffre d'unite n'est pas suivi de ses multiples");
    return 0;
}
```

Exercice 210 :

- Ecrire la fonction récursive **`puiss(n)`** qui reçoit en entrée un nombre entier positif *n* et fournit en sortie le nombre 10^k, avec *k* le nombre de chiffres de *n*.

  Exemple: puiss(346) renvoie 1000

- Écrire, en utilisant la fonction **`puiss`**, la fonction récursive **`miroir(n)`** qui, à partir d'un nombre entier positif *n*, renvoie son miroir.

  Exemple: Le miroir de 7346 est 6437.

Solution :

Algorithm:

```
Algorithm calculMiroir;
Var n,m:integer;
Function puiss(n:integer);
If n = 0 then puiss ← 1
Else puiss ← 10*puiss(n div 10);
End;
Function miroir(n:integer):integer;
Begin
If n=0 then miroir  ← 0
Else miroir ← puiss(n div 10)*(n mod 10)+miroir(n div 10);
End;
Begin
Write("Donner un nombre n:");
Read(n);
m ← miroir(n);
Write("Le miroir de ",n," est ",m);
End.
```

Programme C:

```
#include <stdio.h>
int puiss(int n){
    if(n== 0)return 1;
    else return 10*puiss(n/10);
}
int miroir(int n){
    if(n==0) return 0;
    else return puiss(n/10)*(n%10)+miroir(n/10);
}
main(){
    int n,m;
    printf("Donner un nombre: ");
    scanf("%d",&n);
    m=miroir(n);
    printf("Le miroir de %d est %d",n,m);
}
```

Exercice 214:

On veut calculer le PGCD entre deux nombres ***a*** et ***b*** par la méthode de soustractions successives. Cette méthode opère comme suit:

- Si ***a = b***, le PGCD est ***a***.
- Sinon, on calcule le PGCD du couple formé par la différence entre ***a*** et ***b***, et le plus petit des deux.

  PGCD(a,b) = PGCD(a−b,b) si **a > b**

  PGCD(a,b) = PGCD(a,b−a) si **b > a**

  PGCD(a,b) = a si **a = b**

Pour ce faire, il vous ai demandé de:

- Donner une fonction récursive permettant de calculer le PGCD de 2 nombres entiers passés en paramètres par la méthode de soustractions successives.
- Ecrire l'algorithme/programme principal qui lit deux nombres, et qui calcul et affiche le PGCD entre eux. Le calcul du PGCD doit être effectué en utilisant la fonction précédente.

Solution :

Algorithm:

```
Algorithm calcule_PGCD;
Var x,y,p:integer;
```

```
Function pgcd(a,b:integer):integer;
Begin
If a=b then pgcd ← a
Else If a>b then pgcd ← pgcd(a-b,b)
Else pgcd ← pgcd(a,b-a);
End.
Begin
Write("Donner 2 nombres: ");
Read(x,y);
If x≤0 or y≤0 then write("Erreur de saisi")
Else Begin
      p ← pgcd(x,y);
      Write("Le PGCD est ",p);
      End;
End.
```

Programme C:

```
#include<stdio.h>
int pgcd(int a,int b){
  if(a==b)return a;
  else if(a>b)return pgcd(a-b,b);
       else return pgcd(a,b-a);
}
main(){
    int x,y,p;
    printf("Donner 2 nombres: ");
    scanf("%d%d",&x,&y);
    if(x<=0 || y<=0)printf("Erreur de saisi");
    else{
        p=pgcd(x,y);
        printf("Le PGCD de %d et %d est %d",x,y,p);
    }
}
```

Exercice 215 :

L'algorithme d'Euclide permet de calculer le PGCD entre 2 nombres entiers ***a*** et ***b***, en laissant un reste nul. Ainsi l'algorithme d'Euclide procède comme suit :

- Si ***b*** = 0, l'algorithme termine et rend la valeur ***a*** ;
- Sinon, l'algorithme calcule le reste ***r*** de la division euclidienne de ***a*** par ***b***, puis recommence avec ***a = b*** et ***b = r***.

Pour ce faire, il vous ai demandé de:

- Ecrire une fonction itérative et une récursive permettant de calculer le PGCD entre 2 nombres entiers passés en paramètres en utilisant l'algorithme d'Euclide.
- Ecrire l'algorithme/programme principal qui lit deux nombres, et qui calcul et affiche le PGCD entre eux. Le calcul du PGCD doit être effectué en utilisant les deux fonctions précédentes.

Solution :

Algorithm:

```
Algorithm calcule_PGCD;
Var x,y,p1,p2:integer;
Function pgcdIteratif(a,b:integer):integer;
Var r:integer;
Begin
While b≠0 Do
```

```
     Begin
     r ← a mod b;
     a ← b;
     b ← r;
     End;
pgcdIteratif ← a;
End;
Function PGCDRecursif(a,b:integer):integer;
Begin
If b=0 then PGCDRecursif ← a
Else PGCDRecursif ← PGCDRecursif(b,a mod b);
End;
Begin
Write("Donner 2 nombres: ");
Read(x,y);
If x≤0 or y≤0 then write("Erreur de saisi")
Else Begin
      p1 ← pgcdIteratif(x,y);
      p2 ← PGCDRecursif(x,y);
      Write("Le PGCD est ",p1,p2);
      End;
End.
```

Programme C:

```
#include<stdio.h>
int pgcdIteratif(int a,int b){
    int r;
    while(b!=0){
        r=a%b;
        a=b;
        b=r;
    }
    return a;
}
int PGCDRecursif(int a,int b){
     if(b==0)return a;
     else return PGCDRecursif(b,a%b);
}
main(){
    int x,y,p1,p2;
    printf("Donner 2 nombres: ");
    scanf("%d%d",&x,&y);
    if(x<=0 || y<=0)printf("Erreur de saisi");
    else{
        p1=pgcdIteratif(x,y);
        p2=PGCDRecursif(x,y);
        printf("Le PGCD de %d et %d est %d, %d",x,y,p1,p2);
    }
}
```

Exercice 216 :

- Ecrire la fonction récursive ***puissance*** qui prend en entrée deux entiers ***x*** et ***y*** et qui retourne x^y.
- Ecrire la fonction récursive ***conversion*** qui prend en entrée un entier ***n*** composé uniquement des 0 et des 1 considéré comme un nombre binaire et qui retourne son équivalent en décimal.

  Exemple : $(10110)_2 = 0*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 = 2+4+16 = (22)_{10}$

- Ecrire finalement l'algorithme/programme principal qui lit un nombre binaire ***n***, et qui calcule et affiche son équivalent décimal. Ce dernier est obtenu en appelant la fonction ***conversion*** précédente.

Solution :

Algorithm:

```
Algorithm conversion_binaire_decimal;
Var n,r:integer;
Function puissance(x,y:integer):integer;
Begin
If y=0 then puissance ← 1
Else puissance ← x*puissance(x,y-1);
End;
Function conversion(n,p:integer):integer;
Var a:integer;
Begin
If n≠0 then
     Begin
     a ← n mod 10;
     conversion ← a*puissance(2,p)+conversion(n div 10,p+1);
     End;
End;
Begin
Write("Donner un nombre binaire: ");
Read(n);
r ← conversion(n,0);
Write("Son équivalent décimal est: ",r);
End.
```

Programme C:

```
#include<stdio.h>
int puissance(int x,int y){
    if(y==0)return 1;
    else return x*puissance(x,y-1);
}
int conversion(int n,int p){
    int a;
    if(n!=0){
        a=n%10;
        return a*puissance(2,p)+conversion(n/10,p+1);
    }
}
int main(){
    int n,r;
    printf("Donner un nombre binaire: ");
    scanf("%d",&n);
    r=conversion(n,0);
    printf("Son equivalent decimal est: %d",r);
    return 0;
}
```

Exercice 217:

La suite de Fibonacci (F_n) est définie par récurrence par :

$$F_0 = 0$$
$$F_1 = 1$$

$F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$;

Il vous ai demandé dans cet exercice de:

- Donner une fonction récursive permettant de calcule F_n pour un nombre n passé en paramètres.
- Ecrire l'algorithme/programme principal qui permet de lire un nombre entier positif et qui calcule et affiche la suite de Fibonacci pour ce nombre en utilisant la fonction précédente.

Solution :

Algorithm:

```
Algorithm Fibonacci;
Var n,f:integer;
Function fibo(n:integer):integer;
Begin
If n=0 then fibo ← 0
Else If n=1 then fibo ← 1
Else fibo ← fibo(n-1)+fibo(n-2);
End;
Begin
Write("Donner un nombre integer positif: ");
Read(n);
If n<0 then write("Erreur de saisi")
Else  Begin
      f ← fibo(n);
      Write("F",n," = ",f);
      End;
End.
```

Programme C:

```
#include<stdio.h>
int fibo(int n){
  if(n==0)return 0;
  else if(n==1)return 1;
  else return fibo(n-1)+fibo(n-2);
}
int main(){
    int n,f;
    printf("Donner un nombre entier positif: ");
    scanf("%d",&n);
    if(n<0)printf("Erreur de saisi");
    else{
        f=fibo(n);
        printf("F%d = %d",n,f);
    }
}
```

Exercice 218:

- Ecrire une fonction récursive permettant de déterminer si un nombre passé en paramètres est pair ou impair. Elle retourne Vrai si le nombre est pair et Faux si le nombre est impair.

  On peut formuler les notions récursives suivantes :

  - Le 0 est pair et non impair.
  - Un nombre n est paire si $n-1$ n'est pas pair.
  - Un nombre n est impair si $n-1$ n'est pas impair.
- Ecrire l'algorithme/programme principal qui lit un nombre entier, et qui détermine s'il est pair ou impair en utilisant la fonction précédente.

Solution :

Algorithm:

```
Algorithm test_Parité;
Var n:integer;
Function estPair(n:integer):Boolean;
Begin
If n=0 then estPair ← True
Else If estPair(n-1)=True then
              estPair ← False
Else estPair ← True;
End;
Begin
Write("donner un nombre:");
Read(n);
If n<0 then write("Erreur de saisi")
Else If estPair(n)= True then
              write(n," est pair");
Else write(n," est impair");
End.
```

Programme C:

```
#include<stdio.h>
int pairImpair(int n){
    if(n==0) return 1;
    else if(pairImpair(n-1)==1)
              return 0;
         else return 1;
}
int main(){
    int n;
    printf("donner un nombre:");
    scanf("%d",&n);
    if(n<0)printf("Erreur de saisi");
    else if(pairImpair(n)==1)
              printf("%d est pair",n);
         else printf("%d est impair",n);
}
```

Exercice 219:

- Ecrire une fonction récursive permettant de calculer le nombre de combinaisons de ***p*** objets pris dans un ensemble de ***n*** objets différents, noté C_n^p. Une définition récursive de C_n^p est la suivante :

$$C_n^p = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ C_{n-1}^p + C_{n-1}^{p-1} & \text{sinon} \end{cases}$$

- En utilisant la fonction précédente, écrire l'algorithme/programme principal qui lit deux nombres entiers positifs non nuls ***n*** et ***p***, et qui calcule et affiche C_n^p.

Solution :

Algorithm:

```
Algorithm calcul_combinaisons;
```

```
Var n,p,c:integer;
Function combinaison(n,p:integer):integer;
Begin
If p=0 or p=n combinaison ← 1
Else combinaison ← combinaison(n-1,p) +combinaison(n-1,p-1);
End;
Begin
Write("donner n and p: ");
Read("%d%d",&n,&p);
If n≤0 or p≤0 or p>n then write("Erreur de saisi")
Else   Begin
       c ← combinaison(n,p);
       Write("La combinaison de ",p," parmi ",n," est: ",c);
       End;
End.
```

Programme C:

```
#include<stdio.h>
int combinaison(int n,int p){
 if(p==0 || p==n)return 1 ;
 else return combinaison(n-1,p) +combinaison(n-1,p-1);
}
int main(){
    int n,p,c;
    printf("donner n et p: ");
    scanf("%d%d",&n,&p);
    if(n<=0 || p<=0 || p>n)printf("Erreur de saisi");
    else{
        c=combinaison(n,p);
        printf("La combinaison de %d parmi %d est: %d",p,n,c);
    }
}
```

Exercice 220:

- Ecrire une fonction récursive qui calcul et retourne la somme des éléments d'un tableau de 6 entiers passé en paramètre.
- Ecrire l'algorithme/programme principal qui permet de saisir un tableau de 6 entiers, et qui calcule et affiche la somme de ses éléments. Cette dernière est obtenue en appelant la fonction précédente.

Solution :

Algorithm:

```
Algorithm somme_tableau;
Const N=6;
Type Tab=Tableau[N] d'integers;
Var t:tab;i,s:integer;
Function calculSomme(t:tab,i:integer):integer;
Begin
If i<N then  calculSomme ← t[i]+calculSomme(t,i+1)
Else calculSomme ← 0;
End;
Begin
For i ← 0 To N-1 Do
```

```
     Begin
     Write("Donner l'élément : ",i);
     Read(t[i]);
     End;
i ← 0;
s ← calculSomme(t,i);
Write("La somme est ",s);
End.
```

Programme C:

```
#include <stdio.h>
#define N 6
typedef int tab[N];
int calculSomme(tab t,int i){
    if(i<N)
        return t[i]+calculSomme(t,i+1);
    else return 0;
}
main(){
   tab t;int i,s;
   for(i=0;i<N;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&t[i]);
    }
   i=0;
   s=calculSomme(t,i);
   printf("La somme est %d\n",s);
}
```

Exercice 221:

- Ecrire une procédure récursive qui calcul la somme des éléments d'un tableau de 6 entiers passé en paramètre.
- Ecrire l'algorithme/programme principal qui permet de saisir un tableau de 6 entiers, et qui calcule et affiche la somme de ses éléments. Cette dernière est obtenue en appelant la procédure précédente.

Solution :

Algorithm:

```
Algorithm somme_tableau;
Const N=6;
Type tab=Tableau[N] d'integers;
Var t:tab;i,s:integer;
Procedure calculSomme(t:tab,i:integer;Var s:integer);
Begin
If i<N then
     Begin
     s ← s+t[i];
     calculSomme(t,i+1,s);
     End;
End;
Begin
For i ← 0 To N-1 Do
     Begin
     Write("Donner l'élément : ",i);
```

```
     Read(t[i]);
     End;
i ← 1;
s ← t[0];
calculSomme(t,i,s);
Write("La somme est ",s);

End.
```

Programme C:

```
#include <stdio.h>
#define N 6
typedef int tab[N];
void calculSomme(tab t,int i,int* s){
    if(i<N){
        *s=*s+t[i];
        calculSomme(t,i+1,s);
    }
}
main(){
   tab t;int i,s;
   for(i=0;i<N;i++){
        printf("Donner l'element %d: ",i);
        scanf("%d",&t[i]);
    }
    i=1;
    s=t[0];
    calculSomme(t,i,&s);
    printf("La somme est %d\n",s);
}
```

Exercice 222 :

- Ecrire la procédure récursive ***saisirTableau*** permettant de saisir les éléments d'un tableau de 5 entiers.
- Ecrire la procédure récursive ***inverser*** qui réarrange les éléments d'un tableau de 5 entiers en ordre inverse.
- Ecrire la procédure récursive ***afficherTableau*** qui affiche les éléments d'un tableau de 5 entiers.
- En utilisant les procédures précédentes, écrire l'algorithme/programme principal qui permet de saisir un tableau de 5 entiers, de l'inverser, et d'afficher le tableau résultant.

Solution :

Algorithm:

```
Algorithm inverser_tableau;
Const n=5;
Type tab=Tableau[n] d'integers;
Var t:Tab;i:integer;
Procedure inverser(t:tab,i:integer,j:integer);
Var a:integer;
Begin
If i<j then
     Begin
     a ← t[i];
     t[i] ← t[j];
```

```
     t[j] ← a;
     inverser(t,i+1,j-1);
     End;
End;
Procedure saisirTableau(t:tab,i:integer);
Begin
If i<n then
     Begin
     Write("Donner l'element ",i,": ");
     Read(t[i]);
     saisirTableau(t,i+1);
     End;
End;
Procedure afficherTableau(t:tab,i:integer);
Begin
If i<n then
     Begin
     Write(t[i]);
     afficherTableau(t,i+1);
     End;
End;
Begin
saisirTableau(t,0);
inverser(t,0,n-1);
Write("Le tableau après inversion est: ");
afficherTableau(t,0);

End.
```

Programme C:

```
#include<stdio.h>
#define n 5
typedef int tab[n];
void inverser(tab t,int i,int j){
    int a;
    if(i<j){
       a=t[i];
       t[i]=t[j];
       t[j]=a;
       inverser(t,i+1,j-1);
    }
}
void saisirTableau(int t[n],int i){
    if(i<n){
        printf("Donner l'element %d: ",i);
        scanf("%d",&t[i]);
        saisirTableau(t,i+1);
    }
}
void afficherTableau(int t[n],int i){
    if(i<n){
        printf("%d ",t[i]);
        afficherTableau(t,i+1);
    }
}
```

```
int main(){
    int t[n],i;
    saisirTableau(t,0);
    inverser(t,0,n-1);
    printf("Le tableau apres inversion est:\n");
    afficherTableau(t,0);
}
```

Exercice 224: récursivité imbriquée

La suite d'Ackerman est définie par :

$$Ack(m,n) = \begin{cases} n+1 & si\ m = 0 \\ Ack(m-1,1) & si\ n = 0\ et\ m > 0 \\ Ack\big(m-1, Ack(m,n-1)\big) & sinon \end{cases}$$

- Ecrire ma fonction ***ackerman*** permettant de calculer la suite d'Ackerman pour deux nombres entiers passés en paramètre.
- Ecrire l'algorithme/programme principal qui permet de saisir deux nombres entiers positifs et qui calcule et affiche la valeur de la suite d'Ackerman pour ces nombres. Utiliser la fonction ***ackerman()*** précédente.

Solution :

Algorithm:

```
Algorithm calcul;
Var n,m,a:integer;
Function ackerman(m,n:integer):integer;
Begin
If m=0 then ackerman ← n+1
Else If m>0 and n=0 then ackerman ← ackerman(m-1,1)
Else ackerman ←  ackerman(m-1,ackerman(m,n-1));
End;
Begin
Write("donner deux nombres integers positifs m and n: ");
Read(m,n);
If n<0 or m<0 then write("Erreur de saisi")
Else   Begin
       a ← ackerman(m,n);
       printf("La suite d'Ackerman est: ",a);
       End;
End.
```

Programme C:

```
#include<stdio.h>
int ackerman(int m, int n){
    if(m==0) return n+1;
    else if(m>0 && n==0)return ackerman(m-1,1);
         else return ackerman(m-1,ackerman(m,n-1));
}
int main(){
    int n,m,a;
    printf("donner deux nombres entiers positifs m et n: ");
    scanf("%d%d",&m,&n);
    if(n<0 || m<0)printf("Erreur de saisi");
    else{
        a=ackerman(m,n);
```

```
        printf("La suite d'Ackerman est: %d",a);
    }
}
```

Exercice 225: Récursivité croisée

- Ecrire deux fonctions récursives croisées ***pair*** et ***impair*** qui permettent de tester la parité d'un nombre entier ***n*** de la façon suivante:
  - Le 0 est pair et non impair.
  - Un nombre ***n*** est pair si ***n-1*** est impair.
  - Un nombre ***n*** est impair si ***n-1*** est pair.
- Ecrire l'algorithme/programme principal qui permet de saisir un nombre entier positif et qui teste sa parité. Utiliser les fonctions précédentes.

Solution :

Algorithm:

```
Algorithm pair_impair;
Var n:integer;p:Boolean;
Function pair(n:integer):Boolean;
Begin
If n=0 then pair ← True
Else pair ← impair(n-1);
End;
Function impair(n:integer):Boolean;
If n=0 then impair ← False
Else impair ← pair(n-1);
End;
Begin
Write("donner un nombre integer positif: ");
Read(n);
If n<0 then write("Erreur de saisi")
Else  Begin
      p ← pair(n);
      If p=True then write(n," est pair")
      Else write(n," est impair");
      End;
End.
```

Programme:

```
#include<stdio.h>
int pair(int n){
    if(n==0) return 1;
    else return impair(n-1);
}
int impair(int n){
    if(n==0) return 0;
    else return pair(n-1);
}
int main(){
    int n,p;
    printf("donner un nombre entier positif: ");
    scanf("%d",&n);
    if(n<0)printf("Erreur de saisi");
    else{
```

```
        p=pair(n);
        if(p==1)printf("%d est pair",n);
        else printf("%d est impair",n);
    }
}
```

Exercice 226 :

Voici les 2 suites récurrentes suivantes :

$$U_n = U_{n-1} + V_{n-1}$$
$$V_n = U_{n-1} \times V_{n-1}$$

Avec $U_0 = 1$, $V_0 = 2$.

- Ecrire les deux fonctions récursives mutuellement croisées U et V permettant de calculer U_n et V_n.
- Tester ces fonctions dans un l'algorithme/programme principal.

Solution :

Algorithm:

```
Algorithm suites;
Var n,u,v:integer;
Function U(n:integer):integer;
Begin
If n=0 then U ← 1
Else U ← U(n-1)+V(n-1);
End;
Function V(n:integer):integer;
Begin
If n=0 then V ← 2
Else V ← U(n-1)*V(n-1);
End;
Begin
Write("Donner un nombre:");
Read(n);
u ← U(n);
v ← V(n);
Write("U(",n,")=",u);
Write("V(",n,")=",v);
End.
```

Programme C:

```
#include<stdio.h>
int U(int n){
      if(n==0)return 1;
      else return U(n-1)+V(n-1);
}
int V(int n){
      if(n==0)return 2;
      else return U(n-1)*V(n-1);
}
int main(){
    int n;
    printf("Donner un nombre:");
    scanf("%d",&n);
```

```
    int u=U(n);
    int v=V(n);
    printf("U(%d)=%d\n",n,u);
    printf("V(%d)=%d",n,v);
}
```

Exercice 227:

Un palindrome est une chaîne de caractères (mot ou phrase) que l'on peut lire dans les deux sens (de droite à gauche ou de gauche à droite). Par exemple: Laval, radar, elle, non, ...etc.

- Ecrire la fonction récursive ***estPalindrome*** permettant de vérifier si un mot passé en paramètre est un palindrome ou pas.
- Ecrire l'algorithme/programme principal qui lit un mot d'au maximum 20 lettres, et qui affiche un message indiquant si le mot saisi est un palindrome ou non. La décision qu'un mot est un palindrome ou non doit se faire à l'aide de la fonction ***estPalindrome*** précédente.

Solution :

Algorithm:

```
Algorithm test_palyndrome;
Var mot:String[20];
Function palyndrome(mot:String[20],i:integer):Boolean;
Begin
If i=(Longueur(mot)-1) div 2 then palyndrome ← True
Else If mot[i]≠mot[Longueur(mot)-1-i] then palyndrome ← False
Else palyndrome ← palyndrome(mot,i+1);
End;
Begin
Write("Donner un mot: ");
Read(mot);
If palyndrome(mot,0)=True then write(mot," est un palyndrome")
Else write(mot," n'est pas un palyndrome");
End.
```

Programme C:

```
#include <stdio.h>
#include <string.h>
int palyndrome(char mot[],int i){
    if(i==(strlen(mot)-1)/2)return 1;
    else if(mot[i]!=mot[strlen(mot)-1-i])return 0;
    else return palyndrome(mot,i+1);
}
main(){
   char mot[20];
   printf("Donner un mot: ");
   gets(mot);
   if(palyndrome(mot,0)==1)printf("'%s' est un palyndrome",mot);
   else printf("'%s' n'est pas un palyndrome",mot);
}
```

Exercice 228:

La recherche dichotomique, ou recherche par dichotomie, est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié (on considère dans cet algorithme que le tableau est trié dans l'ordre croissant).

Le principe de cet algorithme est le suivant :

Il compare la valeur recherchée à la valeur du milieu du tableau.

- Si c'est la valeur recherchée, on s'arrête et on retourne sa position.
- Si la valeur recherchée est inférieure à la valeur du milieu de tableau, alors, la valeur recherchée (si elle existe) est située dans la partie gauche du tableau, sinon elle est dans la partie droite.

On répète le procédé de comparaison jusqu'à ce que l'on obtienne la valeur recherchée, ou jusqu'à ce que l'on ait réduit l'intervalle de recherche à un intervalle vide : cela signifie que la valeur recherchée n'est pas présente dans le tableau. Ainsi, à chaque étape, la zone de recherche de la valeur est divisée par deux.

On voudrais dans cet exercice implémenter et expérimenter cet algorithme. Il vous ai ainsi demandé de:

- Ecrire la procédure récursive ***saisirTableau*** permettant de saisir les éléments d'un tableau d'entiers.
- Ecrire la fonction récursive ***estTrie*** qui vérifie si un tableau d'entiers passé en paramètre est trié dans l'ordre croissant.
- Ecrire la fonction récursive ***dichotomie*** qui prend en entrée une valeur ***v*** est un tableau ***T*** et qui effectue la recherche dichotomique de ***v*** dans ***T*** selon le principe illustré ci-dessus.
- En s'aidant des sous-programmes précédents, écrire l'algorithme/programme principal qui permet de remplir au clavier un tableau de 6 entiers, et de tester s'il est trié dans l'ordre croissant. Si c'est le cas, on demande de l'utilisateur de saisir une valeur à rechercher et on vérifie l'existence de cette valeur dans le tableau et dans quelle case.

Solution :

Algorithm:

```
Algorithm recherche;
Const n=5;
Type tab=Tableau[n] d'integers;
Var t:Tab;x,pos:integer;
Procedure remplirTableau(t:tab,i,taille:integer);
Begin
If i<taille then
     Begin
     Write("Donner l'element ",i,": ");
     Read(t[i]);
     remplirTableau(t,i+1,taille);
     End;
End;
Function estTrié(t:tab,i,taille:integer):Boolean;
Begin
If i=taille then estTrié ← True
Else If t[i]<t[i-1] then estTrié ← False
Else estTrié ← estTrié(t,i+1,taille);
End;

Function dichotomie(t:tab,v,a,b:integer):integer;
Var m:integer;
Begin
m ← (a+b)/2;
If a>b then dichotomie ← -1
Else If t[m]=v then dichotomie ← m
```

```
Else If v<t[m] then dichotomie ← dichotomie(t,v,a,m-1)
Else dichotomie ← dichotomie(t,v,m+1,b);
End;
Begin
remplirTableau(t,0,n);
If estTrié(t,1,n)=True then
     Begin
     Write("Le tableau est trié");
     Write("Quelle est la valeur cherchée? ");
     Read(x);
     pos ← dichotomie(t,x,0,n-1);
     If pos>-1 then write(x," existe dans la case ",pos)
     Else write(x," n'existe pas dans le tableau");
     End
Else write("Le tableau n'est pas trié");
End.
```

Programme C:

```
#include <stdio.h>
#include <string.h>
void remplirTableau(int t[],int i,int taille){
    if(i<taille){
        printf("Donner l'element %d: ",i);
        scanf("%d",&t[i]);
        remplirTableau(t,i+1,taille);
    }
}
int estTrie(int t[],int i,int taille){
    if(i==taille)return 1;
    else if(t[i]<t[i-1])return 0;
    else return estTrie(t,i+1,taille);
}
int dichotomie(int t[],int v,int a,int b){
    int m=(a+b)/2;
    if(a>b)return -1;
    if(t[m]==v)return m;
    else if(v<t[m])return dichotomie(t,v,a,m-1);
    else return dichotomie(t,v,m+1,b);
}
main(){
    #define n 6
    int t[n];
    int x,pos;
    remplirTableau(t,0,n);
    if(estTrie(t,1,n)==1){
        printf("Le tableau est trie\n");
        printf("Quelle est la valeur cherchee? ");
        scanf("%d",&x);
        pos=dichotomie(t,x,0,n-1);
        if(pos>-1)printf("%d existe dans la case %d\n",x,pos);
        else printf("%d n'existe pas dans le tableau\n",x);
    }
    else printf("Le tableau n'est pas trie");
}
```

Exercice 229:

On considère les repas que l'on peut composer partir:

- D'une liste des entrées,
- D'une liste des plats principaux,
- D'une liste des desserts.

Voici donc la « carte »:

Entrée	Plat	Dessert
Salade	Cuisse de poulet farcie	Crêpe
Soupe	Saumon fumée	Mousse
	Steak grillé	Glace

- Ecrire la procédure récursive **Menu** qui permet d'afficher tous les repas possibles à partir d'une liste d'entrées, une liste de plats principaux, et une liste de desserts passées en paramètres.
  Ainsi, un repas est constitué d'une entrée, un plat et un dessert. Par exemple [Salade, Poisson, Glace].
- Ecrire l'algorithme/programme principal qui initialise les trois listes, et qui affiche tous les combinaisons possibles (les repas) en appelant la procédure précédente.

Solution :

Algorithm:

```
Algorithm choix;
Const nbEntrees=2;nbPlats=3;nbDesserts=3;
Type tab=Tableau[nbPlats] de String[30];
Var entree,plat,dessert:tab;
Procedure menu(entree,plat,dessert:tab,i,j,k:integer);
Begin
If i=nbEntrees-1 and j=nbPlats-1 and k=nbDesserts-1 then
     write(entree[i],plat[j],dessert[k]);
Else Begin
      Write(entree[i],plat[j],dessert[k]);
      If k<nbDesserts-1 then menu(entree,plat,dessert,i,j,k+1)
      Else If k=nbDesserts-1 and j<nbPlats-1 then
          menu(entree,plat,dessert,i,j+1,0)
      Else menu(entree,plat,dessert,i+1,0,0);
      End;
End;
Begin
entree[0] ← "Salade"; entree[1] ← "Soupe";
plat[0] ← "Cuisse de poulet farcie";
plat[1] ← "Saumon fumée";
plat[2] ← "Steak grillé";
dessert[0] ← "Crêpe";
dessert[1] ← "Mousse";
dessert[2] ← "Glâce";
menu(entree,plat,dessert,0,0,0);
End.
```

Programme C:

```
#include<stdio.h>
#define nbEntrees 2 //le nombre d'elements dans le tableau "entree"
#define nbPlats 3 //le nombre d'elements dans le tableau "entree"
#define nbDesserts 3 //le nombre d'elements dans le tableau "entree"
```

```
#define longChaine 30 //la longueur d'une chaine de caractères
void menu(char entree[][longChaine],char plat[][longChaine],char
dessert[][longChaine],int i,int j,int k){
    if(i==nbEntrees-1 && j==nbPlats-1 && k==nbDesserts-1)
        printf("[%s, %s, %s]\n",entree[i],plat[j],dessert[k]);
    else{
        printf("[%s, %s, %s]\n",entree[i],plat[j],dessert[k]);
        if(k<nbDesserts-1)menu(entree,plat,dessert,i,j,k+1);
        else if(k==nbDesserts-1 && j<nbPlats-1)
            menu(entree,plat,dessert,i,j+1,0);
        else menu(entree,plat,dessert,i+1,0,0);
    }
}
main(){
    char entree[nbEntrees][longChaine]={"Salade","Soupe"};
    char plat[nbPlats][longChaine]={"Cuisse de poulet farcie","Saumon
fumee","Steak grille"};
    char dessert[nbDesserts][longChaine]={"Crepe","Mousse","Glace"};
    menu(entree,plat,dessert,0,0,0);
    return 0;
}
```

Exercice 230:

- Ecrire une procédure récursive permettant de saisir une matrice au clavier.
- Ecrire une fonction récursive qui calcule la somme des éléments d'une matrice passé en paramètres.
- En utilisant les sous-programmes précédents, écrire l'algorithme/programme principal qui permet de saisir une matrice de 3 lignes et de 2 colonnes, et qui calcule et affiche la somme de ses éléments.

Solution :

Algorithm:

```
Algorithm manipulation_matrice;
Const n=3;n=2;
Type tab=Tableau[n,m] d'integer;
Var A:tab;i,j,s:integer;
Procedure saisirMatrice(A:tab,i,j:integer);
Begin
Write("Donner l'élément [",i,",",j,"]: ");
Read(A[i,j]);
If i<n and j<m-1 then saisirMatrice(A,i,j+1)
Else If i<n-1 and j=m-1 then saisirMatrice(A,i+1,0);
End;
Function somme(A:tab,i,j:integer):integer;
Begin
If i=n then return 0;
Else If i<n and j<m-1 then somme ← A[i,j]+somme(A,i,j+1)
Else If i<n and j==m-1 then somme ← A[i,j]+somme(A,i+1,0);
End;
Begin
saisirMatrice(A,0,0);
```

```
s ← somme(A,0,0);
Write("La somme est ",s);
End.
```

Programme C:

```
#include <stdio.h>
#define n 3
#define m 2
void saisirMatrice(int A[n][m],int i,int j){
    printf("Donner l'element [%d,%d]: ",i,j);
    scanf("%d",&A[i][j]);
    if(i<n && j<m-1)
            saisirMatrice(A,i,j+1);
    else if(i<n-1 && j==m-1)
            saisirMatrice(A,i+1,0);
}
int somme(int A[n][m],int i,int j){
        if(i==n)return 0;
        else if(i<n && j<m-1)return A[i][j]+somme(A,i,j+1);
             else if(i<n && j==m-1)return A[i][j]+somme(A,i+1,0);
}
main(){
    int A[n][m];int i,j;
    saisirMatrice(A,0,0);
    int s=somme(A,0,0);
    printf("La somme est %d",s);
}
```