La récursivité

1. Définition :

- En programmation, la ***récursivité*** est une méthode qui permet à un sous programme (procédure ou fonction) de s'appeler elle-même.
- C'est Dans la partie corps (instructions) on retrouve un appel à la procédure (fonction) elle-même.

Exemple : une fonction C permettant de calculer la factorielle d'un entier **n** donné

Solution itérative	**Solution récursive**

int fact(int **n**)
{
int i, p=1 ;
 	
for(i=1 ;i< n ;i++)
p = p * i ;
 	
return p ;
}
	int fact(int **n**)
	{
	int p ;
	if (n = = 0)
	p = 1;
	else
	p = n * fact (n-1);
	
	return p ;
	}

Appel récursive

2. Types de récursivité :

- On distingue en général deux types de récursivité :
  - ✓ La récursivité simple
  - ✓ La récursivité croisée

2.1. La récursivité simple :

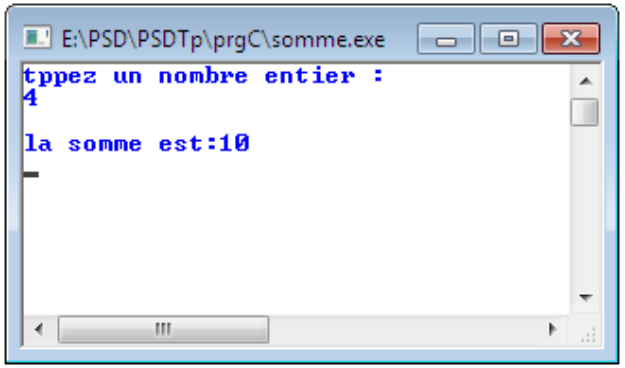
- C'est lorsqu'un sous programme (fonction ou procédure) s'appelle lui même. C'est en effet le cas général de récursivité comme on l'a déjà vu dans l'exemple précédent avec la fonction factorielle.

Exemple : calcul de la somme de **n** premiers nombres entiers positifs.

Somme(n) = 1 + 2 + 3 + … + (n-1) + n

```
#include <iostream>
using namespace std;
int nbr;
int Somme (int n)
 {
    int S;
    if (n == 1)
        S = 1 ;
    else
        S = Somme (n-1) + n ;

    return(S);
  }
int main()
 {
    cout <<"tppez un nombre entier :"<< endl;
    cin>> nbr;
    cout << "la somme est: "<< Somme(nbr) << endl;
    getchar();
  }
```



2.2. La récursivité croisée :

- On appelle récursivité croisé le fait que deux procédures P1 et P2 s'appellent mutuellement, c.-à-d : lorsque **P1** s'exécute, elle fait appel à **P2**, et lorsque **P2** s'exécute, elle fait appel à **P1**.

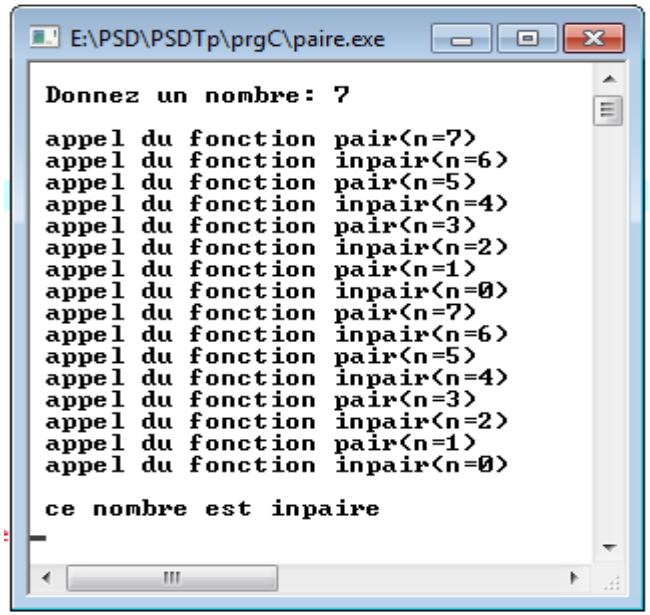
Exemple :

- Un nombre entier positif n peut être soit :
  Pair → n = 2*k
  Impair → n = 2*k+1
- Si on considère deux fonctions Pair (n) et Impair (n) à valeurs logiques (booléen) alors on aura :
  Si Pair (n) = vrai alors Impair (n) = faux
  Si Impair (n) = vrai alors Pair (n) = faux

```
#include <iostream>
using namespace std;
int nbr;
// Déclaration des entêtes des fonctions
int pair(int);
int inpair(int);
// L'implémentation des fonctions
int pair(int n)
  {
      int R;
      cout<<" appel du fonction pair (n=" << n << ")"<<endl;
      if (n==0)
          R=1;
      else
         R=inpair(n-1);

     return(R);
  }
int inpair(int n)
  {
     int R;
     cout<<" appel du fonction impair (n=" << n << ")"<<endl;
     if (n==0)
        R=0;
     else
        R=pair(n-1);

     return(R);
    }
//programme principal
int main()
  {
     cout<<"Donnez un nombre: "<< endl;
     cin>>n;
    // pair(nbr);
     if (pair(nbr)==0)
        cout<<" ce nombre est paire "<<endl;
     else
cout<<" ce nombre est impaire "<<endl;
     getchar();
    return 0;
 }
```



TP :

- la recherche dichotomie d'un élément dans un tableau ordonné s'effectue comme suit :

1) On divise le tableau en deux parties sensiblement égales,
2) On compare la valeur à chercher avec l'élément du milieu,
3) Si elles ne sont pas égales, on s'intéresse uniquement la partie contenant les éléments voulus et on délaisse l'autre partie.
4) On recommence ces 3 étapes jusqu'à avoir un seul élément à comparer.

En utilisant la récursivité, Ecrire une fonction récursive qui recherche par dichotomie un élément x ?

Solution : (en algorithmique)

La fonction retourne le rang de l'élément s'il existe sinon elle retourne -1.

```
Fonction Recherche (Val:entier; V[]:tableau d'entiers;
Iinf,Isup:entier): entier ;
Imil : entier ;
Début
Imil ← (Iinf+Isup) div 2 ;
si (Iinf>Isup ou V [Imil]= Val) alors// condition
d'arrêt
Si (Iinf>Isup) alors
Retourner(-1); // Val n'existe pas dans V
Sinon
Retourner(Imil);
finsi
Sinon
Si( Val< V [Imil] ) alors
Retourner (Recherche (Val, V, Iinf, Imil-1));// rec
Sinon
Retourner (Recherche (Val, V, Imil+1, Isup));//rec
finsi
finsi
Fin ;
```