La récursivité

La récursivité

La récursivité est souvent utilisée en mathématiques et en informatique.

Un programme récursif est un programme qui s'appelle lui-même.

Exemple de la fonction factorielle :

- en mathématiques :

$$n! = n.(n-1)! \text{ , pour } n \geq 1 \text{ avec } 0!=1$$

- en informatique :

```
int factorielle( int n ){
    if( n == 0 ){
        return 1 ;
    }
    return n * factorielle( n-1 ) ;
}
```

```
void main(){
   int n ;
   n = factorielle( 3 ) ;
}
```

Fonction factorielle

```
int factorielle( int n ){
    if( n == 0 ){
        return 1 ;
    }
    return n * factorielle( n-1 ) ;
}
```

```
int factorielle( n = 3 ){
    if( 3 == 0 ){
        return 1 ;
    }
    return 3 * factorielle( 2 ) ;
}
```

```
int factorielle( n = 2 ){
    if( 2 == 0 ){
        return 1 ;
    }
    return 2 * factorielle( 1 ) ;
}
```

```
int factorielle( n = 0 ){
    if( 0 == 0 ){
        return 1 ;
    ...
    }
    return 0 * factorielle( -1 ) ;
}
```

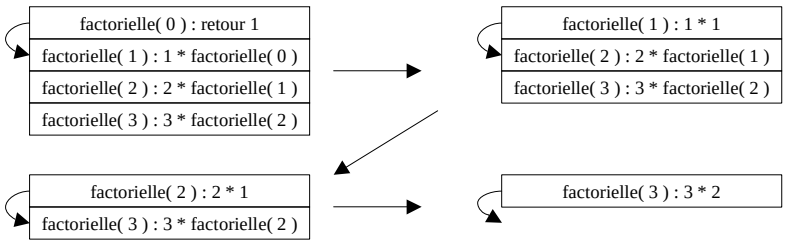


- Mémoriser les résultats intermédiaires ?

Fonction factorielle : utilisation d'une pile

```
void main(){
   int n ;
   n = factorielle( 3 ) ;
}
```

```
int factorielle( int n ){
    if( n == 0 ){
        return 1 ;
    }
    return n * factorielle( n-1 ) ;
}
```



Fonction factorielle

Dans tout programme récursif, il faut une condition de terminaison pour que la fonction cesse de s'appeler.

```
int factorielle( int n ){
    if( n == 0 ){
        return 1 ;
    }
    return n * factorielle( n-1 ) ;
}
```

- la condition de terminaison ne suffit pas toujours :



```
void main(){
   int n ;
   n = factorielle( -1 ) ;
}
```


Suppression de la récursion

- Toute fonction récursive peut s'écrire de façon itérative.

```
int factorielle( int n ){
    if( n == 0 ){
        return 1 ;
    }
    return n * factorielle( n-1 ) ;
}
```

```
int factorielle( int n ){
   int fact ;
   fact = n ;
   while( n > 1 ){
            fact = fact * (n-1) ;
            n = n-1 ;
   }
   return fact ;
}
```

Suppression de la récursion terminale : cas général

```
int recursive( int n ){
  if( cond(n) ){
        return f(n) ;
  }
  return recursive( g(n) ) ;
}
```



```
int iterative( int n0 ){
  n = n0 ;
  while( !cond( n ) ){
        n = g( n ) ;
  }
  return f( n ) ;
}
```


Suite de Fibonacci

Exemple de la suite de Fibonacci :

- en mathématiques :

$$F_n = F_{n-1} + F_{n-2} \text{ , pour } n \geq 2 \text{ avec } F_0 = F_1 = 1$$

- en informatique :

```
int fibonacci( int n ){
  if( n ≤ 1 ){
        return 1 ;
  }
  return fibonacci( n-1 ) + fibonacci( n-2 ) ;
}
```

Les limites de la récursivité

```
int fibonacci( int n ){
  if( n ≤ 1 ){
        return 1 ;
  }
  return fibonacci( n-1 ) + fibonacci( n-2 ) ;
}
```



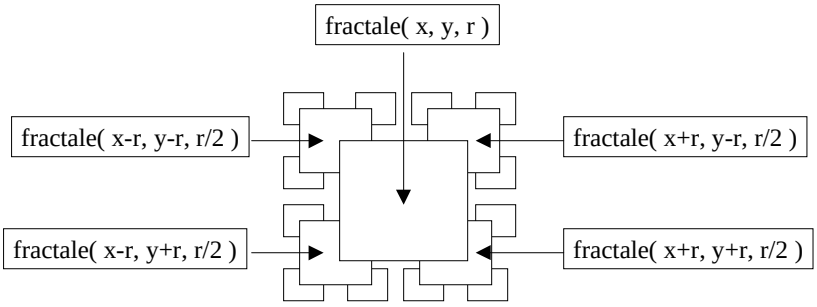
F_{n-1} et F_{n-2} sont évaluées séparément : on ne se sert pas de F_{n-2} pour calculer F_{n-1}.



Fonction itérative ⇒ calcule F_n en temps linéaire.

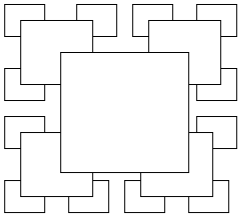
```
int fibonacci( int f[], int n ){
  f[ 0 ] = f[ 1 ] = 1 ;
  for( int i=2; i<=n; i++ ){
        f[ i ] = f[ i-1 ] + f[ i-2 ] ;
  }
  return f[ n ] ;
}
```


La récursivité pour créer des objets fractals



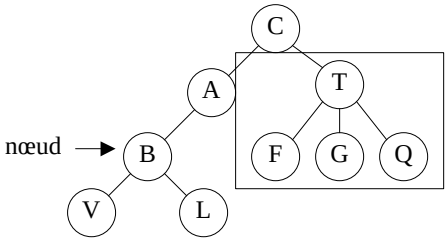

```
void fractale( int x, int y, int r ) {
  ...
  dessinerCarre( x, y, r ) ;    /* trace un carré de coté 2r centré sur x, y */
}
```

La récursivité pour créer des objets fractals



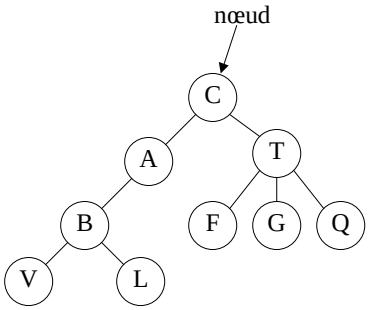
```
void fractale( int x, int y, int r ){
  if( r > 0 ){
        fractale( x-r, y+r, r/2 ) ;
        fractale( x+r, y+r, r/2 ) ;
        fractale( x-r, y-r, r/2 ) ;
        fractale( x+r, y-r, r/2 ) ;
        dessinerCarr( x, y, r ) ;        /* trace un carré de coté 2r centré sur x, y */
  }
}
```


La récursivité pour explorer un arbre (1/2)




```
void explorer( NŒUD nœud ) {
    NŒUD suivant ;
    ENSEMBLE_NŒUDS fils ;
    fils <- enumererFils( nœud )              /* dresse la liste des fils */
    while( fils != NULL ){                    /* pour tous les fils */
        suivant = extraireNoeud( fils ) ;     /* extrait chaque fils … */
        …
    }
}
```

La récursivité pour explorer un arbre (2/2)




```
#define PROFONDEUR_MAX 3

void explorer( NŒUD nœud, int profondeur ){
    NŒUD suivant ;
    ENSEMBLE_NŒUDS fils ;
    if( profondeur == PROFONDEUR_MAX ){
        return ;
    }
    profondeur = profondeur+1 ;
    fils  enumererFils( nœud ) ;            /* dresse la liste des fils */
    while( fils != NULL ){                  /* pour tous les fils */
        suivant = extraireNoeud( fils ) ;   /* extrait chaque fils … */
        explorer( suivant, profondeur ) ;   /* pour l'explorer */
    }
}
```