Recursion

Recursion is a process in which a function calls itself with reduced input and has a base condition to stop the process.

Note :→ A recursion must have

(i) Recursive Function

(ii) Base Condition or Termination or stopping condition.

Principles of Recursion

Recursion is implemented through use of functions. A function that contains a function call to itself or a function call to a second function which eventually calls the first function is known as a recursive function.

Types of Recursion

(1) Direct Recursion

(2) Indirect Recursion

(3) Tail Recursion

(4) Non-tail Recursion

Direct Recursion

A function is said to be directly recursive if it calls itself.

Example

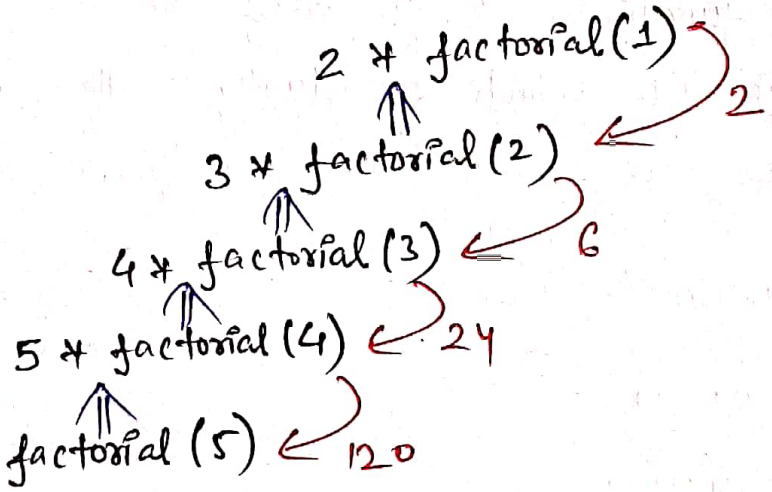
```
A()
{
// statement
A()
}
```

Example: Factorial of a Number

```
int factorial (int n)
{
  if (n==1)                    } Base condition
     return (1);
  else
     return (n * factorial(n-1)); // Recursive Function
}
```

Analysis: factorial (5) = 120

To implement Recursion, Stack data structure is used.



Non-Recursive Function to find Factorial of a Number

```
int factorial (int n)
{ int i, prod = 1;
  for (i=1 ; i<=n ; i++)
  {
   prod = prod * i;
  }
  return (prod);
}
```

(OR)

```
int factorial (int n)
{ int i, prod = 1;
for (i=0 ; i<n ; i++)
{
 prod = prod * (n-i);
}
return (prod);
}
```

Indirect Recursion

A function is said to be indirectly recursive if it contains a call to another function which ultimately calls it.

Example

```
A()
{
// statement
B();
}
```

```
B()
{
// statement
A();
}
```

Example

```
int fun1 (int n)
{
if (n==0)
    return n;
else
    return fun2(n);
}
```

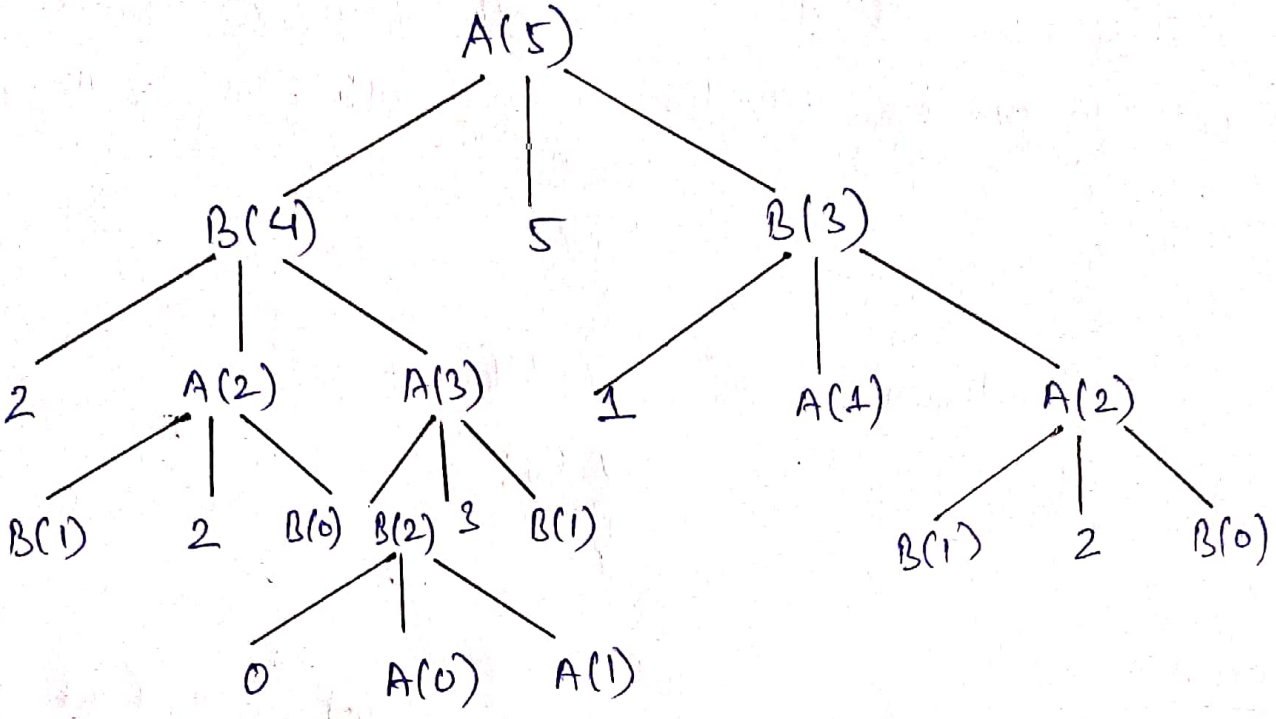
```
int fun2(int n)
{
return fun1(n-1);
}
```

Example

```
A(n)
{
if (n<=1) return;
else
  {
   B(n-1);
   printf("%d", n);
   B(n-2);
  }
}
```

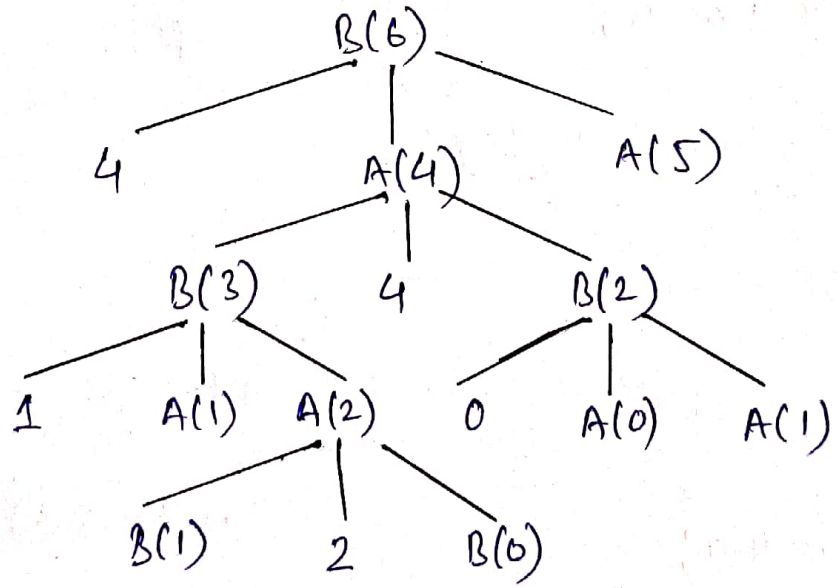
```
B(n)
{
if (n<=1) return;
else
{
 printf("%d", n-2);
 A(n-2);
 A(n-1);
}
}
```

Q Find A(5)



∴ output is 2 2 0 3 5 1 2

Q Find B(6)



∴ Output is 4 1 2 4 0 2 2 0 3 5 1 2

A(5)

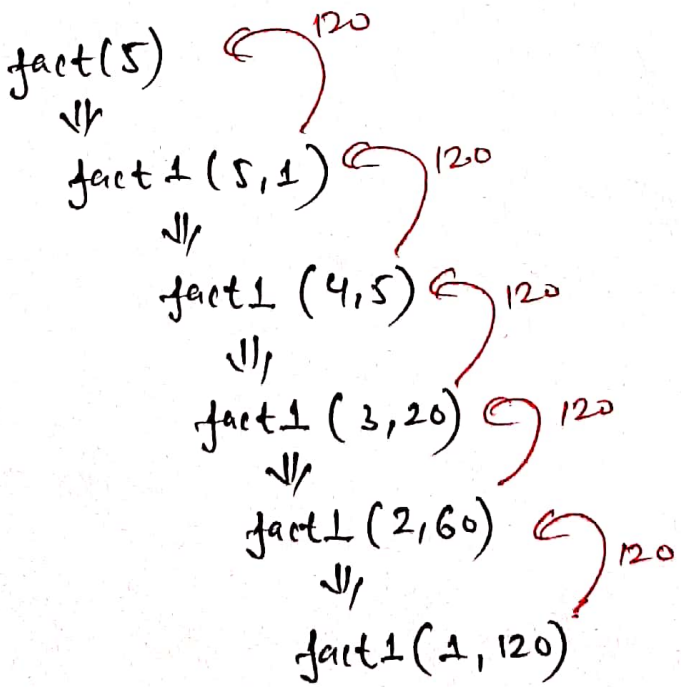
Tail Recursion

In tail Recursion, no operations are pending to be performed when the recursive function returns to its caller.

Example :->

```
int fact(n)
{
  return fact1(n, 1);
}
int fact1(int n, int res)
{
   if (n==1)
      return res;
   else
      return fact1(n-1, n*res);
}
```

Suppose fact(5)



∴ fact(5) = 120

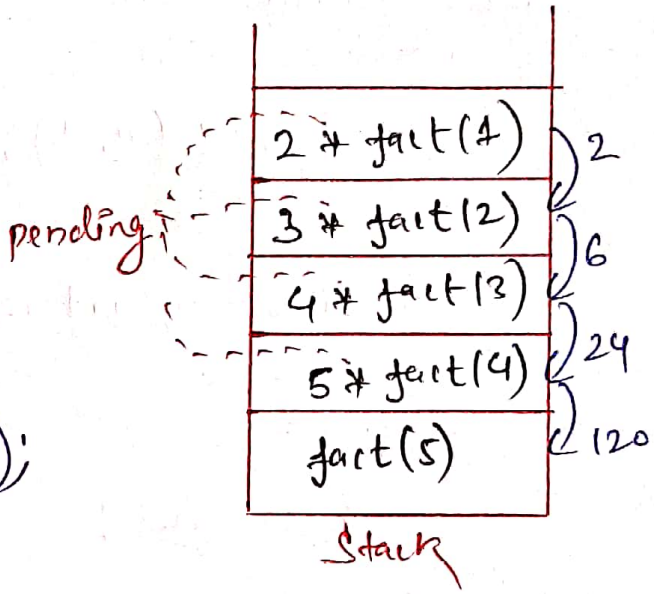
Non-tail Recursion

In non-tail Recursion, there is a pending operation to be performed when the recursive function return to its caller.

Example: Recursive code of Factorial of a number, Fibonacci Series, Towers of Hanoi.

Factorial of a Number

```
int fact(int n)
{
  if (n==1)
      return 1;
  else
      return (n*fact(n-1));
}
```



Here, pending operation is multiplication to be performed on return from each recursive call.

Fibonacci Series

0, 1, 1, 2, 3, 5, 8, ...

```
int Fibonacci(int n)
{
  if (n==0)
      return 0;
  else if (n==1)
      return 1;
  else
      return (Fibonacci(n-1) + Fibonacci(n-2));
}
```

Some important Recursive Functions

① Sum of numbers from 1 to n

```
int Sum(int n)
{
    if (n==0)
        return 0;
    else
        return (n + Sum(n-1));
}
```

② Display numbers from 1 to n

```
void display1(int n)
{
    if (n==0)
        return;
    printf("%d", n);
    display1(n-1);
}
```

this function display numbers from n to 1.

```
void display2(int n)
{
    if (n==0)
        return;
    display2(n-1);
    printf("%d", n);
}
```

this function display numbers from 1 to n.

③ Display and Sum of Series

1+2+3+4+5 = 15

```
#include<stdio.h>
int rseries(int n);
void main()
{
 printf("Enter number of terms : ");
 scanf("%d",&n);
 printf("%d", rseries(n));
}

int rseries(int n)
{
    int sum;
    if(n==0)
        return 0;
    return (n + rseries(n-1));
    printf("%d + ",n);
}
```

(OR)

```
int rseries(int n)
{
  int sum;
  if(n==0)
     return 0;
sum = n + rseries(n-1);
printf("%d + ",n);
return sum;
}
```

④ Sum of digits of an integer

Example:> Sum of digits of 45329

Sumdigits(45329) = 9 + Sumdigits(4532)

Sumdigits(4532) = 2 + Sumdigits(453)

Sumdigits(453) = 3 + Sumdigits(45)

Sumdigits(45) = 5 + Sumdigits(4)

Sumdigits(4) = 4

∴ Sumdigits(n) = least significant digit of n + sumdigits (n with least significant digit removed)

```
int sumdigits(long int n)
{
    if (n/10 == 0) /* if n is a single digit number */
        return n;
    else
        return n%10 + sumdigits(n/10);
}
```

⑤ Display the digits of an integer

```
void display(long int n)
{
  if (n/10 == 0)
    {
      printf("%d", n);
      return;
    }
  display(n/10);
  printf("%d", n%10);
}
```

⑥ Display the digits of an integer in reverse order

```
void Rdisplay(long int n)
  { if (n/10 == 0)
      { printf("%d", n);
        return;
      }
    printf("%d", n%10);
    display(n/10);
  }
```