Recursion

- A function calling itself is called recursion
- There must be a base condition that will terminate the recursion otherwise it will go into infinite loop

Syntax :

Type fun(param)

{

if(< base condition >)
{

1…….
2. fun(param)
3…….
}

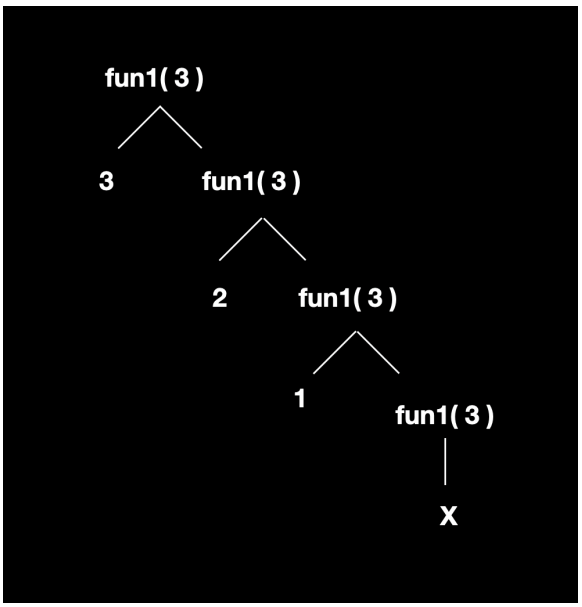
}

An example of Recursive Function :

```c
void fun1(int n)
{
    if(n>0)
    {
        printf("%d" , n );
        fun1(n-1);
    }
}

void main()
{
    int x = 3;
    fun1(x);
}
```

Working Of Recursive Function :



- Once the functions are executed and the end result is obtained then it trace back to the previous functions and terminates it until all the calls are closed and terminated .
- Printing of function can be done on calling time or returning time .

Head Recursion

```c
#include <stdio.h>

void fun(int n)
{
    if(n>0)
    {
        fun(n-1);
        printf("%d ",n);

    }
}

int main() {
    int x=3;

    fun(x);
    return 0;
}
```

Tail Recursion

```c
#include <stdio.h>

void fun(int n)
{
    if(n>0)
    {
       printf("%d ",n);
       fun(n-1);

    }
}

int main() {
    int x=3;
```

```
    fun(x);
    return 0;
}
```

Ds

Static and Global Variable in Recursion

- A stack is created as , for every call each time a new variable will be created inside the memory and it'll have its value this is also know as local variable of a function

Example:

```
int func (int n)
{

  Static int x =0;           // static variables in recursion

    if(n>0)
    {
        x++

        return fun(n-1)+ x ;
        }
    return 0;

}
main()
{
    int a = 5;
    printf("%d", fun(a));

}
```

- If static variables are inside recursive function don’t show them in each tracing tree write them in global or outside variable and maintain a single copy of it

```
int x = 0;              // global variable in recursion

int func (int n)
{
    if(n>0)
    {
        x++

        return fun(n-1)+ x ;
        }
    return 0;
```

```
}
main()
{
    int a = 5;
    printf("%d", fun(a));

}
```

Static Variables in Recursion

```c
#include <stdio.h>

int fun(int n)
{
    static int x=0;
    if(n>0)
    {
        x++;
        return fun(n-1)+x;
    }
    return 0;
}

int main() {

    int r;
    r=fun(5);
    printf("%d\n",r);

    r=fun(5);
    printf("%d\n",r);

    return 0;
}
```

Global Variabels in Recursion

```c
#include <stdio.h>


int x=0;

int fun(int n)
{
   if(n>0)
    {
```

```
        x++;
        return fun(n-1)+x;
    }
    return 0;
}

int main() {

    int r;
    r=fun(5);
    printf("%d\n",r);

    r=fun(5);
    printf("%d\n",r);

    return 0;
}
```

Tail Recursion

- If a recursive function is calling itself and that recursive call is the last statement in a function then it is called as tail recursion.
- After that call it will nor perform any thing.
- All the function will be performing on the calling time itself
- If there is some function that need to be performed after its returning time then it is not a tail function

Example :

```
void fun(int n)
{
    if(n>0)
    {
        printf("%d", n);

        fun(n-1);
    }
}
fun(3);
```

Tail Recursion v/s loops:

- Tail recursion can easily converted into loops as its structure and syntax is almost same
- In term of time taken by both is same 0(n)
- Space taken by tail is 0(n) where as the space for loops is 0(1)
- To conclude , if you are using tail recursions its better to convert it into loop as the space used is less

Head Recursion

- Structure for head recursion

```
Void fun( int n )
{
        if( n > 0)
   {
        fun(n-1);                  // head recursion
                ……….
                ……….
                ……….
    }
}
```

- Here the first statement inside the function is recursive call and all the processing of this function will be done after that

- In head recursion the function doesn't have to process any operation at the time of function calling it has to do everything at the time of returning such functions are head recursion

Head Recursion to Loop

- Although it is possible but It is difficult to convert head recursion into a loop function

Tree Recursion

- If a recursive function calling itself more than one time then it is tree recursion

syntax :

```
fun(n)
{
        if( n > 0 )
    {

        ……..
        ……..
        ……..
        ……..

        fun(n -1);
        ……..
        ……..

        fun(n-1);
        ……..
        ……..
    }
}
```

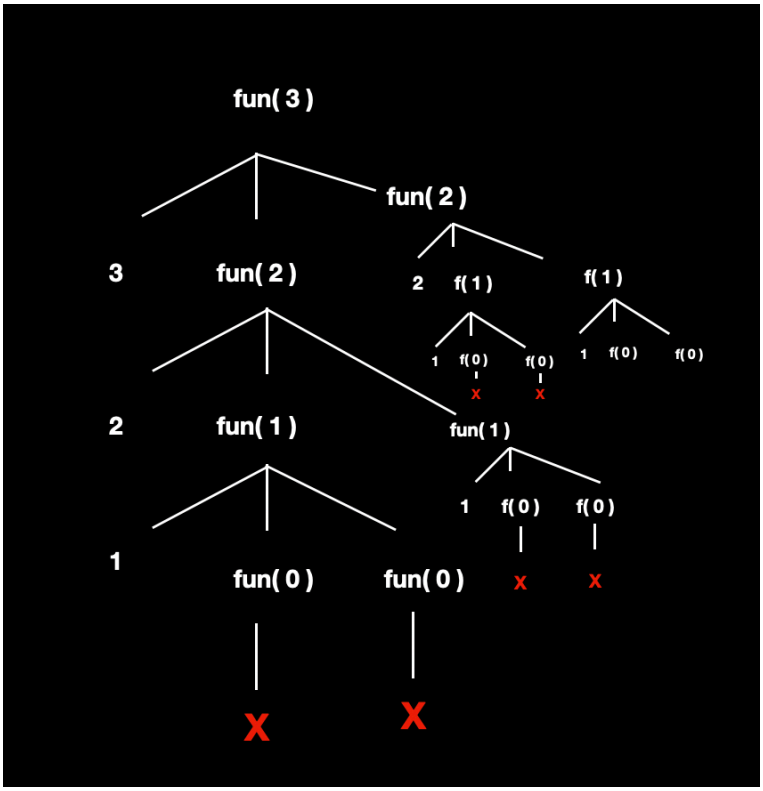
Example :

```c
void fun(int n)
{
     if(n>0)
     {
          printf("%d", n);

          fun(n-1);
          fun(n-1);

     }
}
fun(3);
```

- Activation record idea



Tree Recursion

```c
#include <stdio.h>
void fun(int n)
{
    if(n>0)
    {
        printf("%d ",n);
        fun(n-1);
        fun(n-1);
    }
}
int main() {
    fun(3);
    return 0;
}
```

Indirect Recursion

```c
#include <stdio.h>

void funB(int n);

void funA(int n)
{
    if(n>0)
    {
        printf("%d ",n);
        funB(n-1);
    }
}
void funB(int n)
{
    if(n>1)
    {
        printf("%d ",n);
        funA(n/2);
    }
}

int main()
{
    funA(20);
    return 0;
}
```

Nested Recursion

- A recursive function will pass parameter as a recursive call is called nested recursion
- A recursive function taking recursive call as its parameter is called nested recursion

Example :

```
int fun(int n)
{
    if(n>100)
        return n –10;
    else
        return fun(fun (fun(n+11)));
}
fun(95)
```

Nested Recursion

```c
#include <stdio.h>

int fun(int n)
{
    if(n>100)
        return n-10;
    return fun(fun(n+11));
}

int main()
{
    int r;
    r=fun(95);
    printf("%d\n",r);
    return 0;
}
```

Sum of N natural numbers

```c
int sum(int n)
{
    if(n==0)
        return 0;
    return sum(n-1)+n;
}

int Isum(int n)
{
    int s=0,i;
        for(i=1;i<=n;i++)
                s=s+i;

    return s;

}
int main()
{
    int r=sum(5);
    printf("%d ",r);

    return 0;
}
```

Factorial of N

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

int Ifact(int n)
{
    int f=1,i;
        for(i=1;i<=n;i++)
                f=f*i;

    return f;
}
int main()
{
    int r=Ifact(5);
    printf("%d ",r);

    return 0;
}
```

Power Function

```
int power(int m,int n)
{
    if(n==0)
        return 1;
    return power(m,n-1)*m;
}

int power1(int m,int n)
{
    if(n==0)
        return 1;
    if(n%2==0)
        return                                  power1(m*m,n/
2);
    return m * power1(m*m,(n-1)/2);
}
int main()
{
    int r=power1(9,3);
    printf("%d ",r);

    return 0;
}
```

Taylor Series using Static variables

```c
double e(int x, int n)
{
    static double p=1,f=1;
    double r;

    if(n==0)
        return 1;
    r=e(x,n-1);
    p=p*x;
    f=f*n;
    return r+p/f;
}
int main()
{
    printf("%lf \n",e(4,15));
    return 0;
}
```

Taylor Series

```c
double e(int x, int n)
{
    static double p=1,f=1;
    double r;

    if(n==0)
        return 1;
    r=e(x,n-1);
    p=p*x;
    f=f*n;
    return r+p/f;
}
int main()
{
    printf("%lf \n",e(4,15));
    return 0;
}
```

Taylor Series Horner's Rule

```c
double e(int x, int n)
{
    static double s;
    if(n==0)
        return s;
    s=1+x*s/n;
    return e(x,n-1);

}
int main()
{
    printf("%lf \n",e(2,10));
    return 0;
}
```

Taylor Serie Iterative

```c
#include <stdio.h>
```

```c
double e(int x, int n)
{
    double s=1;
    int i;
    double num=1;
    double den=1;

    for(i=1;i<=n;i++)
    {
        num*=x;
        den*=i;
        s+=num/den;
    }
    return s;
}
int main()
{
    printf("%lf \n",e(1,10));
    return 0;
}
```

Taylor Serie Iterative

```c
#include <stdio.h>

double e(int x, int n)
{
    double s=1;
    int i;
    double num=1;
    double den=1;

    for(i=1;i<=n;i++)
    {
        num*=x;
        den*=i;
        s+=num/den;
    }
    return s;
}
int main()
{
    printf("%lf \n",e(1,10));
    return 0;
}
```

Fibonacci

```c
#include <stdio.h>

int fib(int n)
{
    int t0=0,t1=1,s=0,i;

    if(n<=1) return n;

    for(i=2;i<=n;i++)
    {
        s=t0+t1;
        t0=t1;
        t1=s;
    }

    return s;
}

int rfib(int n)
{
    if(n<=1)return n;
    return rfib(n-2)+rfib(n-1);
}

int F[10];

int mfib(int n)
{
    if(n<=1)
    {
        F[n]=n;
        return n;
    }
    else
    {
        if(F[n-2]==-1)
            F[n-2]=mfib(n-2);
        if(F[n-1]==-1)
```

```
            F[n-1]=mfib(n-1);
        F[n]=F[n-2]+F[n-1];
        return F[n-2]+F[n-1];
    }
}

int main()
{
    int i;
    for(i=0;i<10;i++)
        F[i]=-1;

    printf("%d \n",mfib(5));
    return 0;
}
```

Combination Formula

```c
#include <stdio.h>

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

int nCr(int n,int r)
{
    int num,den;

    num=fact(n);
    den=fact(r)*fact(n-r);

    return num/den;
}

int NCR(int n,int r)
{
    if(n==r || r==0)
        return 1;
    return NCR(n-1,r-1)+NCR(n-1,r);

}

int main()
{
    printf("%d \n",NCR(5,3));
    return 0;
}
```

Tower of Hanoi

```c
#include <stdio.h>

void TOH(int n,int A,int B,int C)
{
    if(n>0)
    {
        TOH(n-1,A,C,B);
        printf("(%d,%d)\n",A,C);
        TOH(n-1,B,A,C);
    }
}
int main()
{
    TOH(4,1,2,3);
    return 0;
}
```