RECURSION

It is a modular programming technique.

A function calling itself is known as recursive function i.e, function call statement for calling the same function appears inside the function.

In recursion, the same function acts as calling function and called function.

Ex:

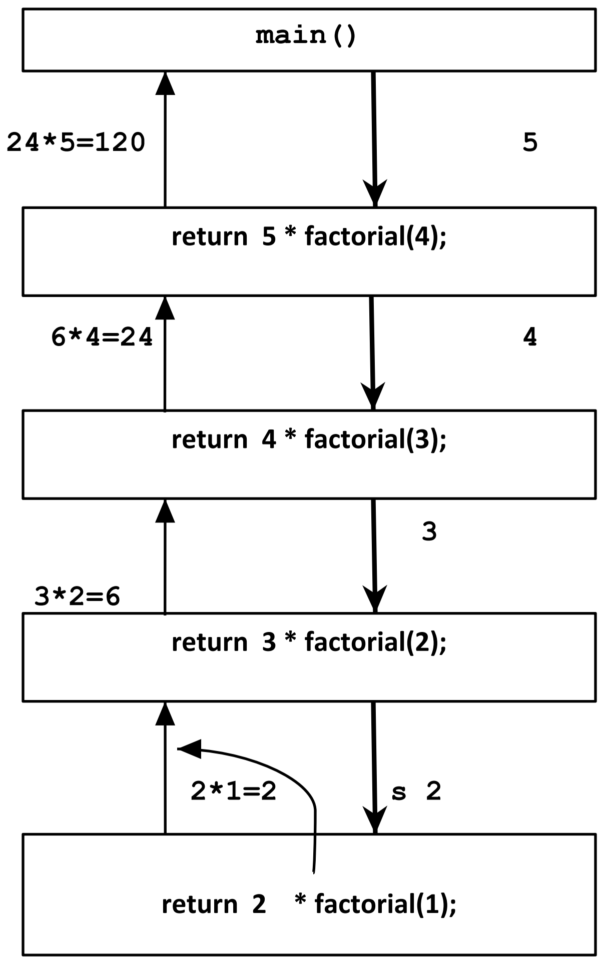
```
int main()
{
    Printf("here main() is a recursion function");
    main();
}
```

There must be an exclusive terminating condition or else the execution will continue indefinitely.

Program to find factorial of number using recursion

```
#include<stdio.h>
int factorial( int k);
int main()
{
  int s,n;
  printf("enter a no for printing factorials from 1 to entered no. ");
  scanf("%d",&n);
  s = factorial(n);
  printf(" factorial = %d", s);
}
 int  factorial (int k )
{
  if ( k==1)
   return  1;
  else
 return  k * factorial(k-1);
}
```

Consider   k = 5



In the above program :-

- main() is calling function for factorial (5)
- factorial (5) is calling function for factorial(4)
- factorial(4) is calling function for factorial(3)
- factorial(3) is calling function for factorial(2)
- factorial(2) is calling function for factorial(1)

so, called function will return the value to its calling function.

Difference between recursion and iteration:

Both recursion and iteration are based on control structures.

- Iteration uses repetition structures to achieve looping and recursion uses selection structure to make repeated function calls.
- Both iteration and recursion use a condition to terminate. Iteration terminates when condition in loop fails. Recursion terminates when condition in 'if' statement becomes TRUE.
- Recursion helps us to solve a complex problem by breaking the program into smaller problems, that are similar to original problem. But each recursive call creates another copy of function in memory which consumes more and more memory. Thus, recursion should be applied only to larger programs.

Program to find " x power y " using recursion

```
#include<stdio.h>
int power(int,int);
int  main()
{
        int b,e,res;
        printf("Enter the value of base and exponent");
        scanf("%d%d",&b,&e);
        res = power(b,e);
        printf("value of %d power %d is %d",b,e,res);
        }

int power(int b, int e)
{
        if(e==0)
                return 1;
        else if(e==1)
                return b;
        else
                return  (b * power(b,e-1));
}
```

Program to find fibonacii series using recursion

```
#include<stdio.h>
int  main()
{
int i,n;
printf("Enter the n value:");
scanf("%d",&n);
printf("the fibinacci series is\n");
for(i=0;i<n;i++)
{
printf("%d\n",fib(i));
}
}


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

Program for binary search using recursion

```
#include  <stdio.h>
main()
{
            int a[50];
             int n,no,x,result;

             printf("Enter the number of terms : ");
             scanf("%d",&no);

             printf("Enter the elements :\n");
             for(x=0;x<no; x++)
              scanf("%d",&a[x]);

             printf("Enter the number to be searched : ");
             scanf("%d",&n);

             result = binarysearch(a, n, 0, no-1);  // function call

             if(result == -1)
              printf("Element not found");
             return 0;
}

binarysearch(int a[] ,int n, int low ,int high)
{
            int mid;
             if (low > high)
              return -1;

             mid = (low + high)/2;

             if(n == a[mid])
               {
                 printf("The element is at position %d\n",mid+1);
                 return 0;
               }
             if(n < a[mid])
               {
                 high = mid - 1;
                 binarysearch(a, n, low ,high);  // recursive call
               }
             if(n > a[mid])
               {
                   low = mid + 1;
                 binarysearch(a, n, low ,high);  // recursive call

               }
 }
```