Navlakhi™

Chapter 8

Recursion

Recursion means a function that calls itself.
Advantage: Smaller code size
Disadvantage: Slower & high amount of stack memory required.

Example 1:

Factorial using recursion
As we all know

$$5! = 5 \text{ X } 4 \text{ X } 3 \text{ X } 2 \text{ X } 1 = 5 \text{ X } 4!$$

Similarly

$$4! = 4 \text{ X } 3!$$
$$3! = 3 \text{ X } 2!$$
$$2! = 2 \text{ X } 1!$$
$$1! = 1 \text{ X } 0!$$

And

$$0! = 1$$

Thus factorial(n)= n X factorial(n – 1)
This completes the first step (i.e. designing your iterative formula). Now we have to design the stopping criteria is when will the above formula not be applicable. The answer is when n=0. This completes the second step.

Now we have to design the function. The function will have n as the pre-requirement & the stopping criteria i.e. n=0 will have to be tested first & then the recursive condition. Always the criteria which does not satisfy the recursive formula is put first…. Note in factorial the answer will be a long int & not a int (since int is only upto 32767).

```
#include<stdio.h>
long factorial(int n)
{       if(n==0)
        {
                return 1;
        }
        else
        {
                return n*factorial(n-1);
        }
}
void main( )
{       int x;
        long p;

        printf( "Enter the number ");
        scanf("%d",&x);

        p=factorial(x);
        printf( "Factorial of &d=%ld\n", x , p);
}
```

NOTE: In the factorial function we have put the non-recursive criteria first & then the recursive criteria.

Example 2:

Let's try GCD using recursion….

If you recollect we had done GCD/HCF by taking a reverse loop & the moment we found the first number dividing both we stopped & termed it as the GCD.

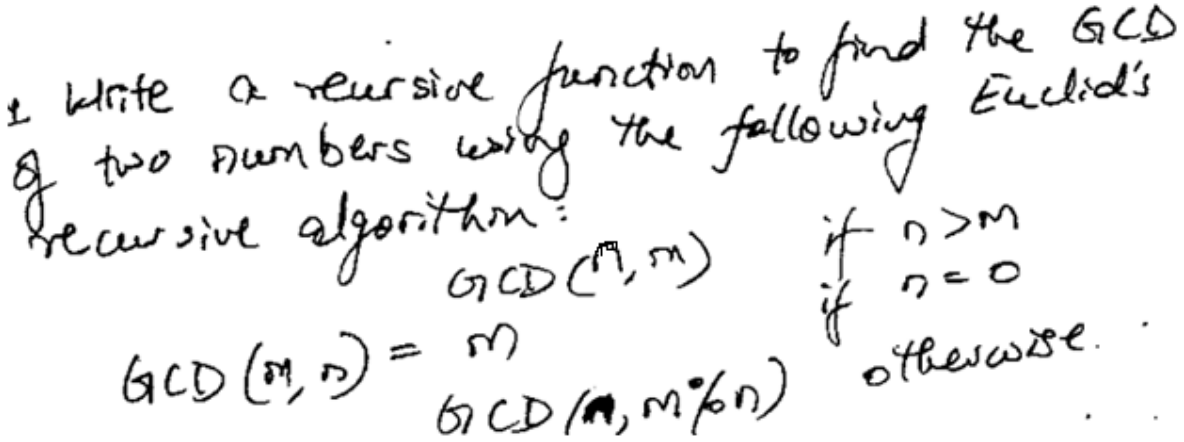
Hence the repetitive criteria is dividing by 'i' (from m to 1) in steps of -1 till we find an 'i' dividing both.

```
#include<stdio.h>
int GCD(int m, int n, int i)
{
        if(m%i==0 && n%i==0)
        {
                return i;
        }
        else
        {
                return GCD(m,n,i-1);
        }
}
void main( )
{
        int m,n,ans;

        printf( "Enter the 2 numbers ");
        scanf("%d%d",&m,&n);

        ans=GCD(m,n,m);

        printf( "GCD=%d",ans);
}
```

NOTE: the function needs m,n & intial value of 'i' which can be m or n.

Example 3:

In some question we don't have to think of the login…. As can be seen in the question given below, there is on non-recursive condition, & that has to be first & the remaining below it. Don't care to think if the logic works or not….




```
#include<stdio.h>

int GCD(int m, int n)
{
        if(n==0)
        {
                return m;
        }
        else if(n>m)
        {
                return GCD(n,m);
        }
        else
        {
                return GCD(n,m%n);
        }
}
void main( )
{
        int m,n,ans;

        printf( "enter 2 numbers ");
        scanf("%d%d",&m,&n);

        ans=GCD(m,n);

        printf( "GCD=%d",ans);
}
```

Example 4:

Recurive fibonacci…. (it is a series 1, 1, 2, 3, 5, 8, 13,……., basically next term is sum of previous 2 terms).

Here we have to repeatedly do c=a+b (initial value of a=1 & b=1), then we shift by 1 i.e. a becomes current b & b becomes current c, so we get a new c using the same formula c=a+b. We stop when we have printed the necessary number of terms (say n).

```
#include<stdio.h>
void fibo(int a, int b, int done, int n)
{
        if(done==n)
        {
                return;
        }
        else
        {
                int c=a+b;
                printf("%d\t",c);
                a=b;
                b=c;
                fibo(a,b,done+1,n);
        }
}
void main( )
{
        int n;

        printf( "Feed in the number of terms ");
        scanf("%d", &n);

        if (n==1)
                printf( "1");
        else if(n==2)
                printf("1    1");
        else
        {
                printf( "1    1     ");
                fibo(1,1,2,n);
        }
}
```

NOTE: The logic of c=a+b can only begin if n >=3. Hence special cases of n=1 & n=2 have been handled separately in main.

Example 5:

(b) Write a program to find value of y using recursive function, where $y = x^n$, x & y are real number & 'n' is an integer. 12

NOTE:

$$y = x^n$$
$$y = x [x^{n-1}]$$
$$y = x[x[x^{n-2}]]$$
$$\text{.....}$$
$$\text{.....}$$
$$y = x[x[x[x\text{.....}[x^0]\text{....}]]]]]]$$

Recursion will stop when the power of x becomes 0.

Program:

```
#include <stdio.h>
float power(float x, int n)
{
        if (n==0)
        {
                return 1;
        }
        else
        {
                return (x * power(x,n – 1));    //power reduces by 1, hence n – 1
        }
}

void main( )
{
        float x,y;
        int n;

        printf( "Enter x,n");
        scanf("%d%d",&x,&n);

        y = power(x,n);

        printf( "Answer=%d",y);
}
```