+ Write at least 3 paragraphs about recursion and its process.

A recursion is one that calls itself to solve a smaller version of its task until a final call which doesn’t require a self-call. Thus a function is known as recursive function if it calls to itself.

The recursion in programming is a technique for defining a problem in terms of one or more smaller versions of the same problem. The solution of the problem is built on the results from the smaller versions. The recursion is used for repetitive computation in which each action is stated in term of previous action. Many iterative or repetitive problems can be written in recursive form. But it is not necessary that all problems can be solved using recursion. To solve a problem using recursion the following two conditions must be satisfied:

1. Problem could be written or defined in term of its previous result.
2. Problem statement must include a stopping condition(i.e we must have an if statement the somewhere to force the function to return without the recursive call being executed, otherwise the function will never return).

+ WAP to find factorial of N number

#include<stdio.h>

#include<conio.h>

Long int factorial(int n)

{

If(n==1)

Return 1;

Else

Return (n\*factorial(n-1));

}

Int main()

{

Printf(“enter a number”);

Scanf(“%d”,&num);

Printf(“the factorial of %d is %d”,factorial(num));

Getch();

Return 0;

}

+ WAP to find multiplication of N numbers.

+ WAP to find fibonacci series in c using for loop.

#include <stdio.h>

#include <conio.h>

int fibonacci(int term);

int main()

{

int terms, count;

printf("Enter number of terms in Fibonacci series: ");

scanf("%d", &terms);

printf("Fibonacci series of %d terms is\n", terms);

for(count=0;count<terms;count++){

printf("%d ", fibonacci(count));

}

getch();

return 0;

}

int fibonacci(int term){

if(term < 2)

return term;

return fibonacci(term - 1) + fibonacci(term - 2);

}

• Pointer

+ Differentiate between Passing by Value and Passing by Reference, write programs for each.

Passing by Value

When we pass by value we have another variable that points at the other variable, so basically two copies of the same variable, and only the copy is modified leaving our original unchanged. You pass an argument by value by specifying the By Val (Visual Basic) keyword for the corresponding parameter in the procedure definition. When you use this passing mechanism, Visual Basic copies the value of the underlying programming element into a local variable in the procedure. The procedure code does not have any access to the underlying element in the calling code.

Passing by Reference

When we pass by reference we are handing in our only copy of the variable, and the function directly modifies that variable. You pass an argument by reference by specifying the By Ref (Visual Basic) keyword for the corresponding parameter in the procedure definition. When you use this passing mechanism, Visual Basic gives the procedure a direct reference to the underlying programming element in the calling code.

+ WAP to print the sum of two matrix using dynamic memory Allocation (STRICTLY USE DMA).

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,p,q,r,s,\*m1, \*m2, \*a;

clrscr();

printf("Enter the order of matrix A");

printf("\n");

printf("m=");

scanf("%d", &p);

printf("n=");

scanf("%d", &q);

printf("\nEnter the order of matrix B");

printf("\n");

printf("m=");

scanf("%d", &r);

printf("n=");

scanf("%d", &s);

if(q==r)

{

m1=(int\*)calloc(p\*q, sizeof(int));

m2=(int\*)calloc(r\*s, sizeof(int));

a=(int\*)calloc(p\*s, sizeof(int));

printf("\nEnter the elements of matrix A:");

for(i=0;i<p;i++)

{

for(j=0;j<q;j++)

{

scanf("%d",(m1+i\*q+j));

}

}

printf("\nEnter the elements of matrix B:");

for(i=0;i<r;i++)

{

for(j=0;j<s;j++)

{

scanf("%d",(m2+i\*r+j));

}

}

printf("\nAddition:");

for(i=0;i<p;i++)

{

for(j=0;j<q;j++)

{

\*(a+i\*q+j)=\*(m1+i\*q+j)+(\*(m2+i\*r+j));

}

}

printf("\nSum of the Matrices A and B:");

for(i=0;i<p;i++)

{

printf("\n");

for(j=0;j<q;j++)

{

printf("%d\t",\*(a+i\*q+j));

}

}

}

getch();

}

• Algorithm

+ Why an algorithm is necessary before trying to solve any problem?

An algorithm is the step-by-step description of the procedure written in human understandable language for solving given problem. Algorithm maintains sequences of computer instructions required to solve a problem in such a way that if the instruction are executed in the specified sequence, the desired result is obtained. So algorithm is necessary before trying to solve any problem to know about the flow of program step by step.

+ Explain one of your real life problem and design an algorithm to solve it.

To make a tea.

Step1:start

Step2:Turn on the gas.

Step3:Put the utensil in the gas.

Step4:Put some milk and water in the utensil.

Step5:Put some sugar and tea leaves in the mixture of milk and water.

Step6:Put the tea in the cup.

Step7:Stop

• MISSING GAP

+ Research about Tower of Hanoi using text books or internet. Write what you can understand; also suggest how can we solve a tower of hanoi problem (ALSO write an algorithm)

The Towers of Hanoi is a mathematical puzzle whose solution illustrates recursion. There are three pegs which can hold stacks of disks of different diameters. A larger disk may never be stacked on top of a smaller. Starting with n disks on one peg, they must be moved to another peg one at a time.