**DEVELOP A PROGRAM TO DEMONSTRATE THE CONCEPT OF RECURSION (FACTORIAL, BINARY SEARCH, TOWER OF HANOI)**

**FACTORIAL**

**THEORY**

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

void recursion() {

recursion(); /\* function calls itself \*/

}

int main() {

recursion();

}

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

**CODE WITH COMMENTS**

#include<stdio.h>

int fact(int n);//function prototype

int main()//main function

{

int n,f;//variable declaration

printf("enter the number whose factorial is to be found\n");

scanf("%d",&n);//scanning the entered number

f=fact(n);//function call

printf("factorial=%d",f);//print the factorial

}

int fact(int n)//function defination

{

if(n==0) return 0;//if entered number is 0 returns 0

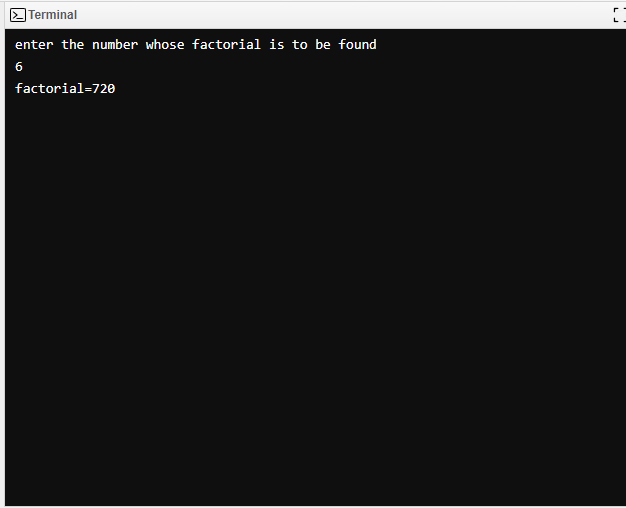
else if(n==1) return 1;//if entered number is 1 returns 1

else

return n\*fact(n-1);//reccursive function

}

**OUTPUT**

****

**ALGORITHM**

**1**. Start

**2.**Read number num

**3.**Call factorial(num)

**4.** If no<0 then return -1

**5.**Else if no=0 then return 1

**6.**Else

**7**. return(no\*factorial(no-1))

**8.** Print factorial fact

**BINARY SEARCH**

**THEORY**

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

void recursion() {

recursion(); /\* function calls itself \*/

}

int main() {

recursion();

}

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Binary Search is a searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array.

**CODE WITH COMMENTS**

#include<stdio.h>//header file

int binarySearch(int array[], int low, int high)//function defination

{

int x;//variable declaration

if(high>=low)//high should always be greater than low

{

int mid=low+(high-low)/2;//caculation of mid

if(array[mid]==x)//element found in the mid of the array

return mid;//returns mid

if(array[mid]>x)//if element found in the left half

return binarySearch(array, x, low, mid-1);//here high is replaced by mid-1

else

return binarySearch(array, x, mid+1, high);//here low is replaced by mid+1

}

else return -1;

}

int main()

{

int array[], n, search, i;

n=sizeof(array)/sizeof(array[0]);//size of array is calculated

printf("enter the array elements\n");

for(i=0;i<n;i++)//loop to scan the array

scanf("%d",array[i]);

printf("enter the element to be searched\n");//serach element entered

scanf("%d",&search);//serach element scanned

int result=binarySearch(array[], search, 0,n-1);//function call

if(result==-1) //if function returns -1

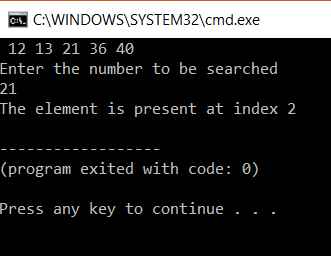
printf("not found\n");//element is not found

else

printf("search element found at %d",result);//element found at particular position

}

**OUTPUT**



**ALGORITHM**

binarySearch(arr, x, low, high)

if low > high

return False

else

mid = (low + high) / 2

if x == arr[mid]

return mid

else if x > arr[mid] // x is on the right side

return binarySearch(arr, x, mid + 1, high)

else // x is on the right side

return binarySearch(arr, x, low, mid - 1)

// Binary Search in C

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

// If found at mid, then return it

if (array[mid] == x)

return mid;

**TOWER OF HANOI**

**THEORY**

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules:

Only one disk can be moved at a time.

Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

No disk may be placed on top of a smaller disk.

Examples:

**Tower of Hanoi using Recursion:**

 The idea is to use the helper node to reach the destination using recursion. Below is the pattern foris problem:

Shift ‘N-1’ disks from ‘A’ to ‘B’, using C.

Shift last disk from ‘A’ to ‘C’.

Follow the steps below to solve the problem:

Create a function towerOfHanoi where pass the N (current number of disk), from\_rod, to\_rod, aux\_rod.

Make a function call for N – 1 th disk.

Then print the current the disk along with from\_rod and to\_rod

Again make a function call for N – 1 th disk.

**CODE WITH COMMENTS**

#include<stdio.h>

void towerofHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)//function defination

{

if(n==1)//if only one rod exixts

{

printf("move disk 1 from rod %c to rod %c",from\_rod,to\_rod);

return;//the above instruction to follow

}

towerofHanoi(n-1,from\_rod, aux\_rod, to\_rod);//function call

printf("move disk %d from rod %c to rod %c",n,from\_rod,to\_rod);//instruction

towerofHanoi(n-1,from\_rod, to\_rod, from\_rod);//reccursive function call

}

int main()//main function

{

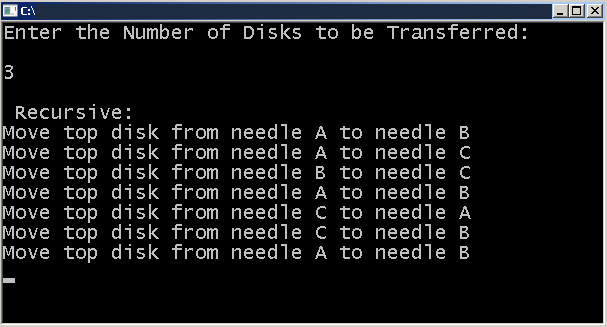
int n=4;//vari8able declaration

towerofHanoi(n,\'A\',\'B\',\'C\');//function call

return 0;

}

**OUTPUT**



**ALGORITHM**

Create a function towerOfHanoi where pass the N (current number of disk), from\_rod, to\_rod, aux\_rod.

Make a function call for N – 1 th disk.

Then print the current the disk along with from\_rod and to\_rod

Again make a function call for N – 1 th disk.