**Micro Project**

1. **Rationale:**

Recursion is a routine that calls itself again and again directly or indirectly. There are two types of recursion in the C language Direct calling and Indirect calling. The calling refers to the recursive call. The recursion is possible in C language by using method and function. The problems like the Tower of Hanoi, the Fibonacci series, and the nth derivative can be solved using recursion. The recursion uses a stack to store its calls in memory.

1. **Course action addressed:**
   1. Perform basic operations on arrays.

* Tower of Hanoi
* GCD(Greatest Common Divisor)
* Hailstone Sequence
* Fibonacci Series
* Factorial Of Number

1. **Action Methodology:**
2. **Algorithm:**

**Step 1:** Display a menu of options for the user to choose from:

1.Tower of Hanoi 2.GCD 3.Hailstone sequence 4.Fibonacci series 5.Factorial of number 6.Exit

**Step 2:** Prompt the user to select an option.

**Step 3**: if choice = 1 goto 9

**Step 4:** if choice = 2 goto 16

**Step 5:** if choice = 3 goto 26

**Step 6:** if choice = 4 goto 37

**Step 7:** if choice = 5 goto 43

**Step 8:** if choice = 6 goto 53

//Tower of Hanoi

**Step 9:** Input the number of disks

**Step 10:** Create a stack to hold the disks

**Step 11:** Push the disks onto the stack

**Step 12:** Create helper stacks to hold the disks

**Step 13:** Move the top disk from the first stack to one of the helper stacks, and then the top disk from the helper stack to the remaining stack

**Step 14:** Keep repeating step 6 until all the disks have been moved to the final stack

**Step 15:** goto 1

// GCD

**Step 16:** Input two numbers

**Step 17:** Create a stack and push the two numbers onto it

**Step 18:** Pop the two numbers off the stack

**Step 19:** Check if one of the numbers is 0

**Step 20:** If one of the numbers is 0, the other number is the divisor

**Step 21:** Otherwise, push both numbers onto the stack

**Step 22:** Subtract the smaller number from the larger number and push the result onto the stack

**Step 23:** Keep repeating steps 4-8 until one of the numbers is 0

**Step 24:** The other number is the divisor

**Step 25:** goto 1

//Hailstone Sequence

**Step 26:** Input a number

**Step 27:** Create a stack and push the number onto it

**Step 28:** Pop the number off the stack

**Step 29:** Check if the number is 1

**Step 30:** If the number is 1, the sequence is complete

**Step 31:** Otherwise, push the number onto the stack

**Step 32:** Check if the number is even

**Step 33:** If the number is even, divide it by 2 and push the result onto the stack

**Step 34:** Otherwise, multiply the number by 3 and add 1 and push the result onto the stack

**Step 35:** Keep repeating steps 4-10 until the number is 1

**Step 36:** goto 1

//Fibonacci Series

**Step 37:** Input the number of terms

**Step 38:** Create a stack and push two terms (0 and 1) onto it

**Step 39:** Pop the two terms off the stack

**Step 40:** Add the two terms and push the result onto the stack

**Step 41:** Keep repeating steps 4-5 until the desired number of terms has been reached

**Step 42:** goto 1

// Factorial of Number

**Step 43:** Input a number

**Step 44:** Create a stack and push the number onto it

**Step 45:** Pop the number off the stack

**Step 46:** Check if the number is 1

**Step 47:** If the number is 1, the result is 1

**Step 48:** Otherwise, push the number onto the stack

**Step 49:** Subtract 1 from the number and push the result onto the stack

**Step 50:** Keep repeating steps 4-8 until the number is 1

**Step 51:** Multiply all the numbers on the stack to get the result

**Step 52:** goto 1

**Step 53:** Stop

1. **Flowchart:**













1. **Source Code:**

#include<stdio.h>

// Main Menu

main()

{

while(1)

{

int choice;

printf("\n\n\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\n");

printf("1. Tower of Hanoi\n");

printf("2. Greatest Common Divisor\n");

printf("3. Hailstone Sequence\n");

printf("4. Fibonacci Series\n");

printf("5. Factorial of Number \n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

tower\_of\_hanoi();

break;

case 2:

gcd();

break;

case 3:

hailstone\_sequence();

break;

case 4:

fibonacci\_series();

break;

case 5:

factorial();

break;

case 6:

exit(0);

}

}

}

// Tower of Hanoi

void tower\_of\_hanoi()

{

int num,i;

printf("\n\nEnter the number of disk: ");

scanf("%d",&num);

tower\_of\_hanoi\_function(num,'A','B','C');

printf("\n\n");

}

void tower\_of\_hanoi\_function(int num,char frompeg,char topeg,char auxpeg)

{

if(num==1)

{

printf("\n\nMove disk 1 from %c to %c",frompeg,topeg);

return;

}

tower\_of\_hanoi\_function(num-1,frompeg,auxpeg,topeg);

printf("\n\nMove disk %d from %c to %c",num,frompeg,topeg);

tower\_of\_hanoi\_function(num-1,auxpeg,topeg,frompeg);

}

// Greatest Common Divisor

void gcd()

{

int num1, num2,hcf;

printf("\n\nEnter the two numbers: ");

scanf("%d %d",&num1,&num2);

hcf=gcd\_function(num1,num2);

printf("\n\nGCD of %d and %d is %d\n\n",num1,num2,hcf);

}

int gcd\_function(int num1,int num2)

{

if(num2!=0)

return gcd\_function(num2,num1%num2);

else

return num1;

}

// Hailstone Sequence

void hailstone\_sequence()

{

int num;

printf("\n\nEnter the number(>0): ");

scanf("%d",&num);

printf("\n\nHailstone Sequence of %d is: ",num);

while(num!=1)

{

printf("%d ",num);

if(num%2==0)

num=num/2;

else

num=(3\*num)+1;

}

printf("1\n\n");

}

// Fibonacci Series

void fibonacci\_series()

{

int i,n,a=0,b=1,temp;

printf("\n\nEnter the number of terms: ");

scanf("%d",&n);

printf("\n\nFibonacci Series is: ");

for(i=1;i<=n;i++)

{

printf("%d ",a);

temp=a+b;

a=b;

b=temp;

}

printf("\n\n");

}

// Factorial of Number Using Recursion in Stack

void factorial()

{

int num,f;

printf("\n\nEnter the number: ");

scanf("%d",&num);

//push(num);

f=factorial\_function(num);

printf("\n\nFactorial of %d is %d\n\n",num,f);

}

int factorial\_function(int num)

{

if(num==0)

return 1;

else

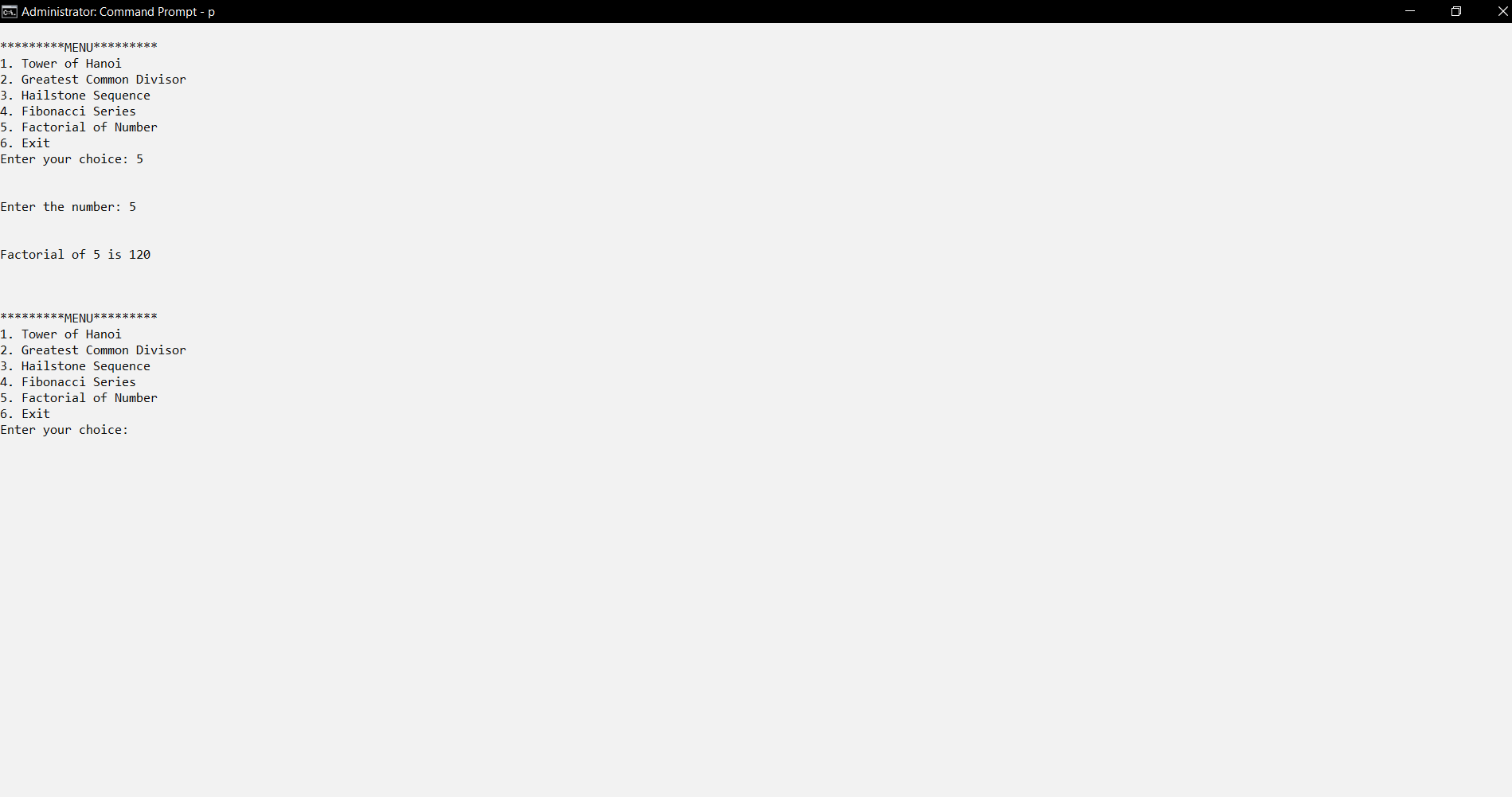
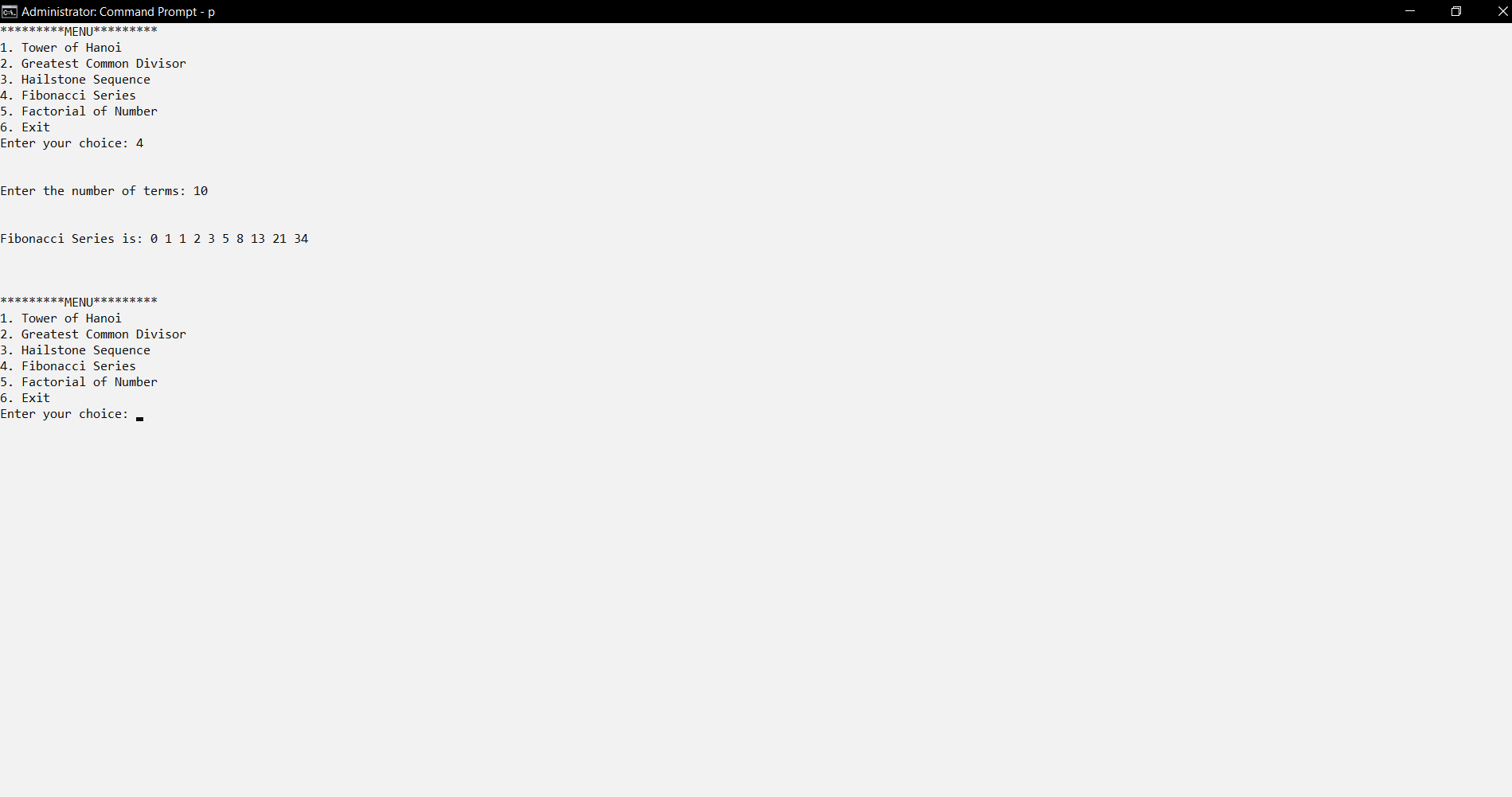
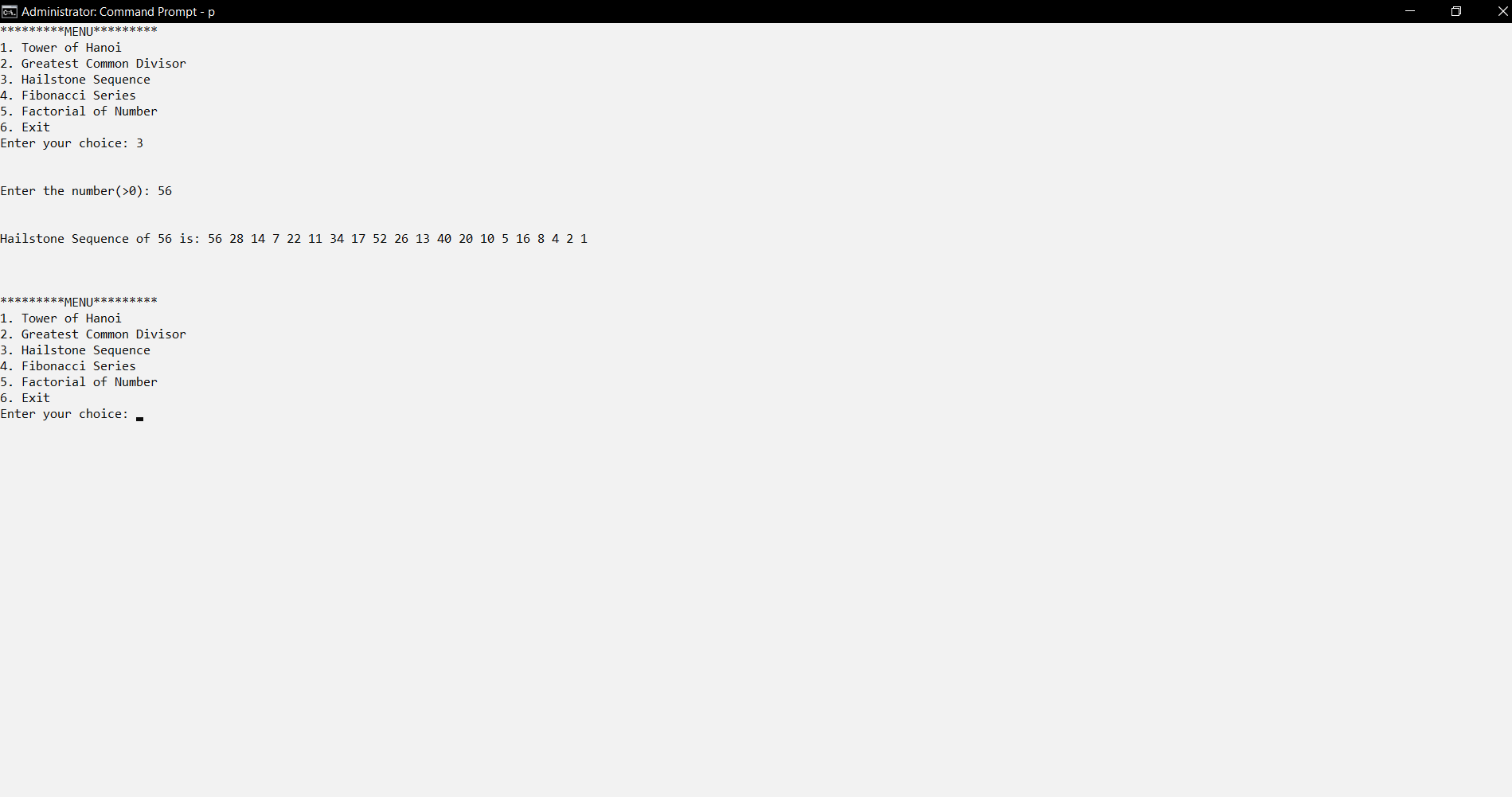
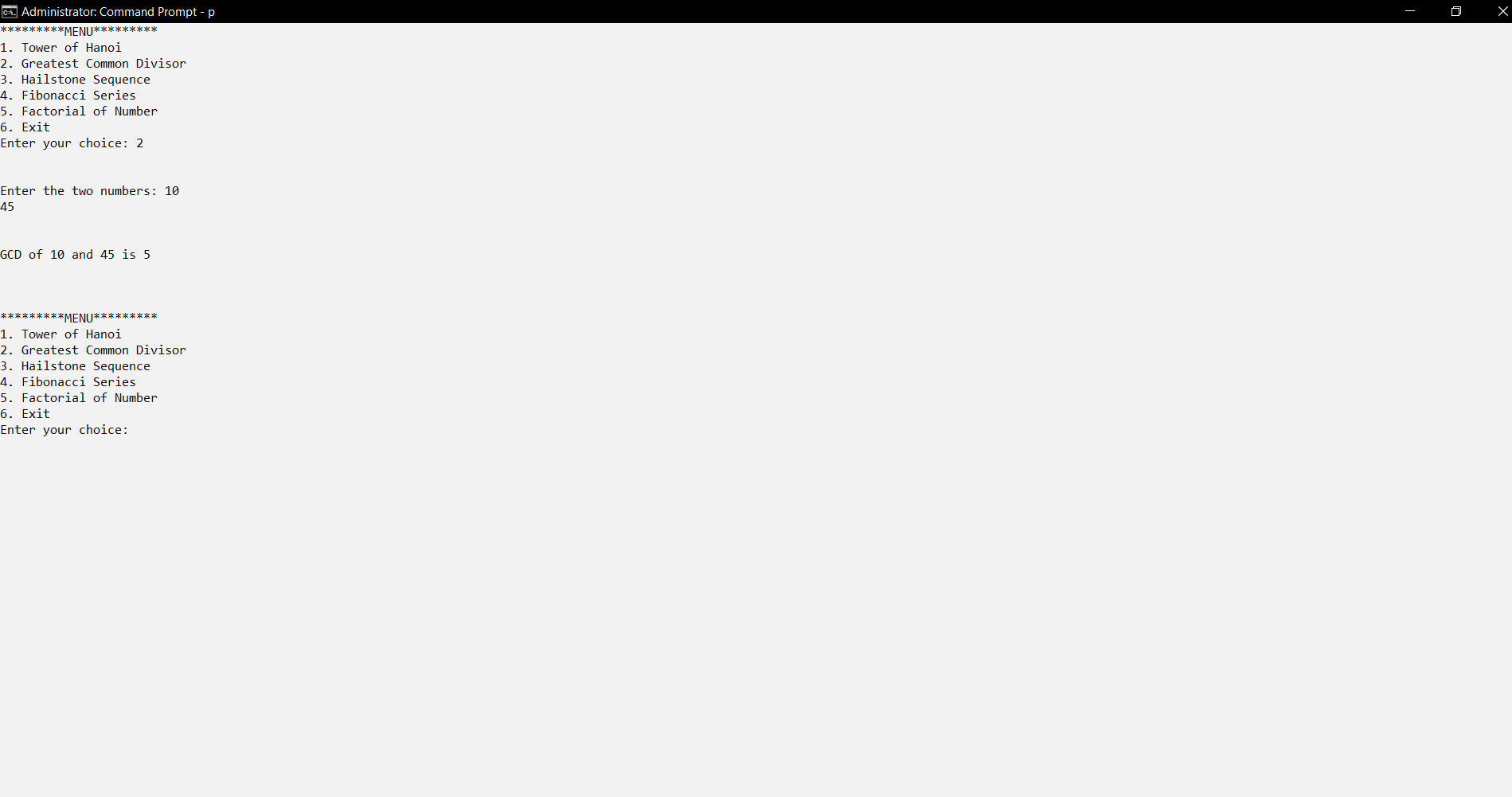
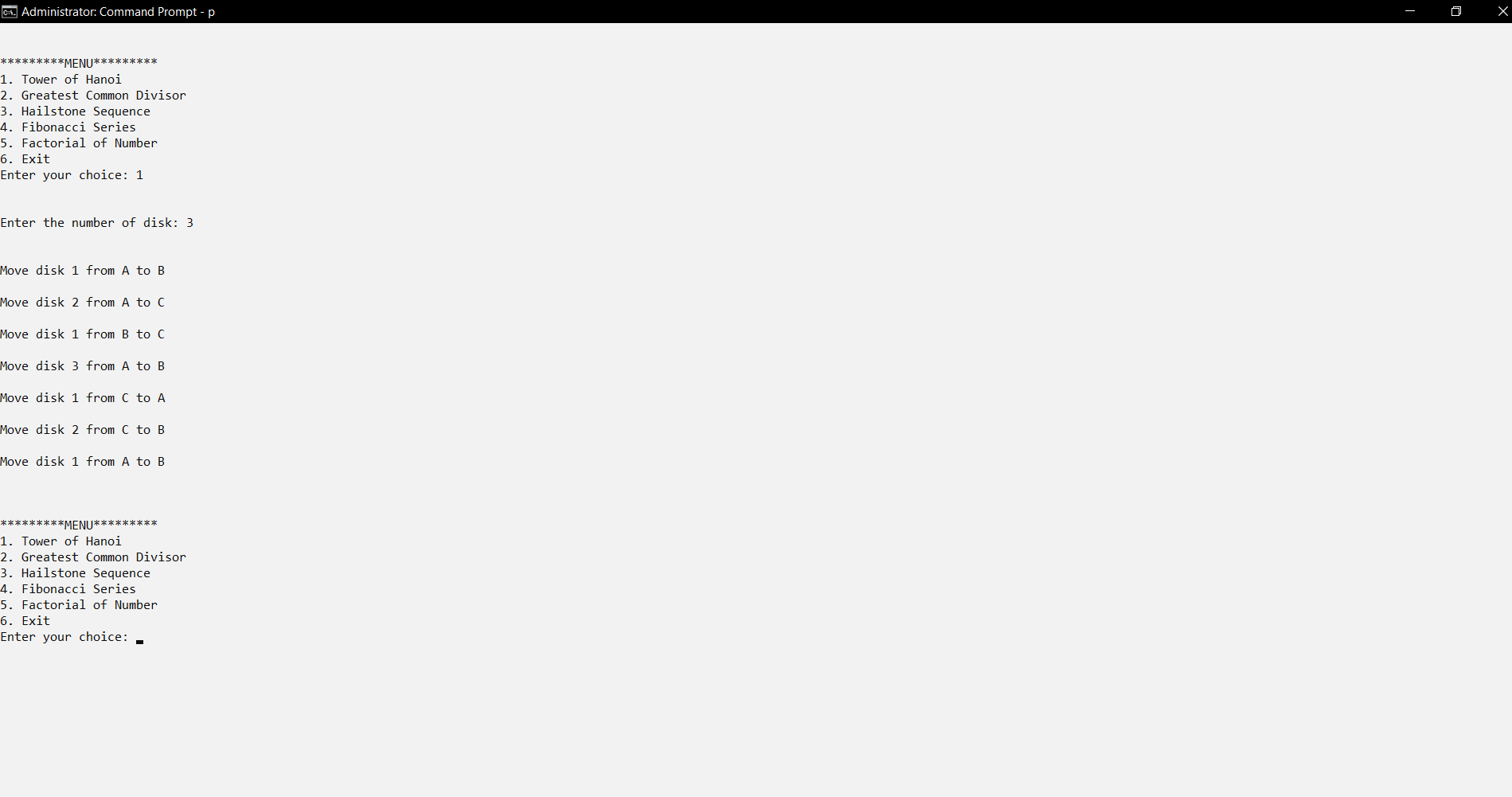
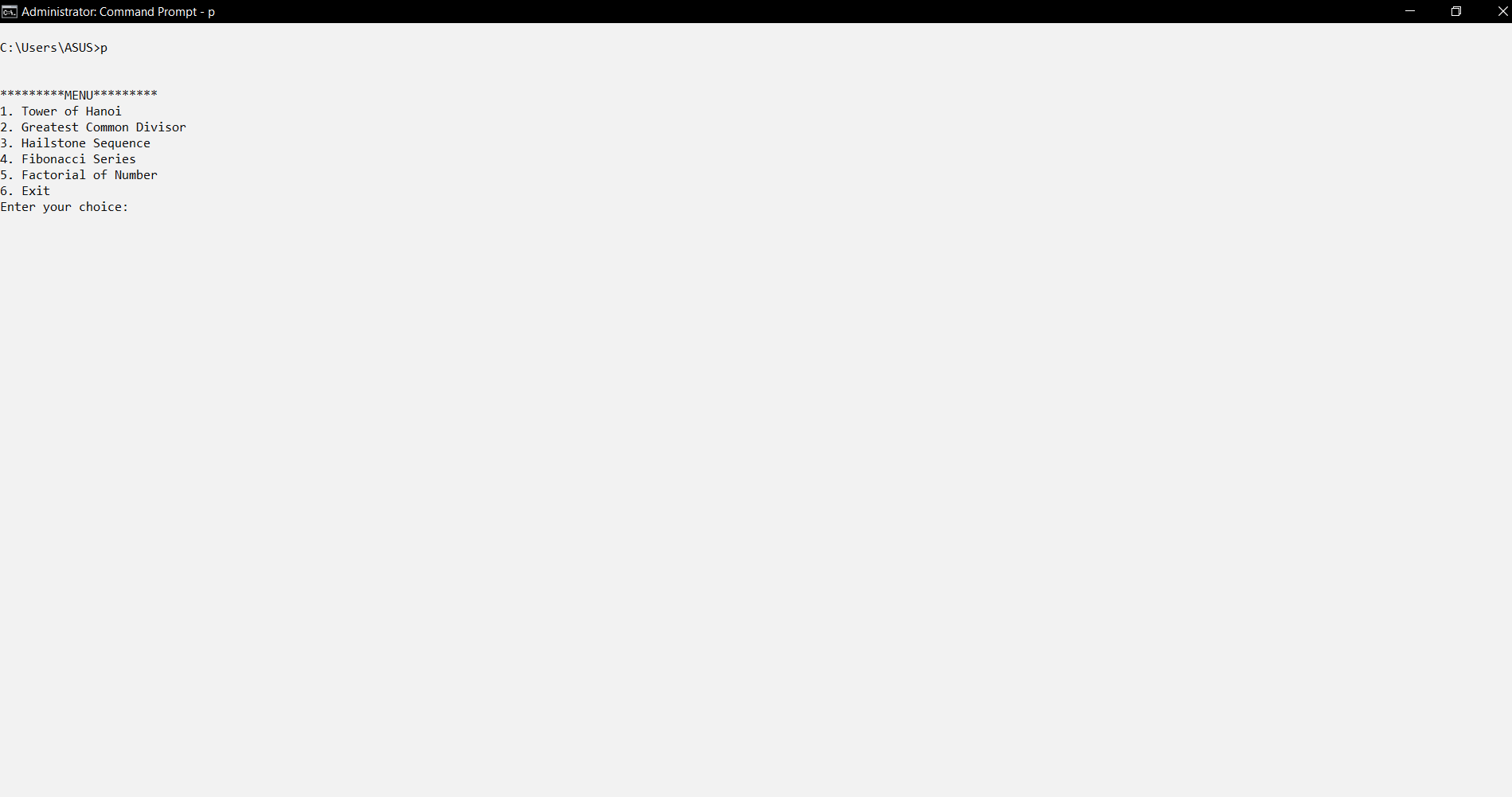
{

return num\*factorial\_function(num-1);

}

}

1. **Microproject Output:**

****

1. **Actual Resources Used:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No** | **Name of Resource** | **Specifications** | **Quantity** | **Remarks** |
| **1.** | Hardware: Computer System | Computer(i3-i5 preferable), RAM minimum 2 GB and onwards | 1 | **-** |
| **2.** | Operating System | Windows 7 or later Version/LINUX version 5.0 or Later Version | 1 | **-** |
| **3.** | Software | Turbo C/C++ Version 3.0 or Later Version | 1 | **-** |

1. **Skills Developed:**

* Problem Solving
* Debugging
* Memory Management
* Algorithmic Skills
* Logic Building

1. **Application of Microproject:**

Recursion is extremely useful and extensively used because many problems are elegantly specified or solved in a recursive way. The example of recursion as an application of stack is keeping books inside the drawer and the removing each book recursively.

1. **Area of Future improvement:**

Stack is used for storing the activation record of a function call which contains arguments, local variables, and the address of the calling function. Space complexity of the recursive function equals the number of calls made to the function as the same number of activation records will be stored in the call stack.