DATA STRUCTURE AND ALGORITHM

Lab 6

Recursion

Submitted By:

Ravi Pandey

075BCT065

INTRODUCTION

Recursion

The technique by which a function or module calls itself is known as recursion. Recursion is allowed in most programming languages. In recursion, a function **A** either calls itself directly or calls a function **B** that in turn calls the original function **A**. The function **A** is called recursive function.

Tail Recursion

A recursive function is tail recursive when recursive call is the last thing executed by the function. The tail recursive functions considered better than non tail recursive functions as tail-recursion can be optimized by compiler.

// An example of tail recursive function

void print(int n)

{

    if (n < 0)  return;

    std::cout << " " << n;

    // The last executed statement is recursive call

    print(n-1);

}

WRITE AN ALGORITHM AND PROGRAM FOR FOLLOWING USING RECURSION:

a. FIND THE FACTORIAL FOR GIVEN NUMBER ‘N’

Algorithm:

1. Get the input from user
2. Check if the number is positive or not, if not, display error message and terminate process
3. If number is valid, call a recursive function fact(n)
4. fact(n):

if(n=1):

return 1 //stopping condition

else:

return n\*fact(n-1) //recursive condition

Program Code:

#include <iostream>

long double fact(long double n)

{

    std::cout<<n<<"\*";

    if(n==1)

    {

        std::cout<<"\b \b = ";

        return 1;

    }

    return n\*fact(n-1);

}

int main()

{

    long double num;

    std::cout<<"Enter Number:\t";

    std::cin>>num;

    if(num<0)

    {

        std::cout<<"\nCan't find factorial of negative number";

        exit(0);

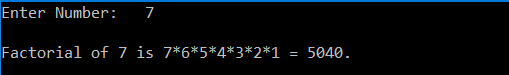
    }

    std::cout<<"\nFactorial of "<<num<<" is "<<fact(num)<<"."<<std::endl;

    return 0;

}

Output:



b. FIND THE FIBONACCI NUMBER FOR GIVEN TERM ‘N’.

Algorithm:

1. Get the input from user
2. Check if the number is positive or not, if not, display error message and terminate process
3. If number is valid, call fibb(n)
4. fibb(n):

if(n=1 or n=0):

return n //stopping condition

else:

return fibb(n-1) + fibb(n-2) //recursive condition

Program Code:

#include <iostream>

long double fibb(long double n)

{

    if(n==1 || n==0)

        return n;

    else

        return fibb(n-1)+fibb(n-2);

}

void displayFibbSeries(int n)

{

    std::cout<<"\nThe Fibonacci series upto "<<n<<"th term is:\n";

    for(int i =1;i<=n;i++)

        std::cout<<"  "<<fibb(static\_cast<long double>(i));

    std::cout<<'\n';

}

int main()

{

    int num;

    while(1)

    {

    std::cout<<"\nEnter Operation:\n1)Get nth Fibonacci term\n2)Get Fibonacci series upto nth term:";

    std::cin>>num;

    if(num<0)

        std::cout<<"\nInvalid Number!";

    else if(num==1)

    {

        std::cout<<"\nEnter n:";

        std::cin>>num;

        std::cout<<num<<"th Fibonacci term is: "<<fibb(static\_cast<long double>(num));

    }

    else if(num==2)

    {

        std::cout<<"\nEnter n:";

        std::cin>>num;

        displayFibbSeries(num);

    }

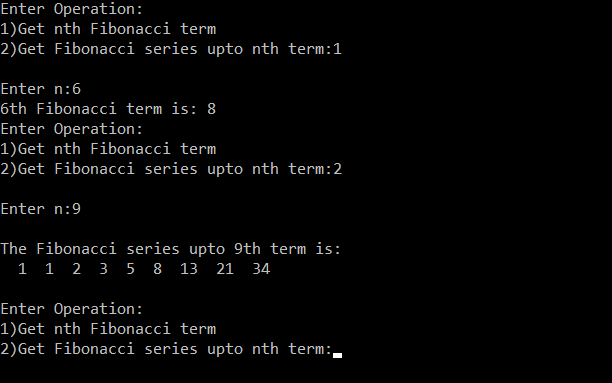
    else

        exit(0);

    }

}

Output:



c. SOLVE ABOVE QUESTIONS USING TAIL RECURSION.

Algorithm:

1. Fibonacci:
2. Create a wrapper function to call the recursive function: Fibb (n, a, b)
3. Pass n along with (a=1 and b=0) as parameters to recursive function
4. Add second and third parameters after each call and pass as second argument: fibb (n-1, a + b, b)
5. Decrease n by 1 after each recursive call
6. Stop recursion when n is 1 or 0.
7. Factorial:
8. Create a wrapper function to call recursive function: Fact(n,a=1)
9. Recursive function with two arguments
10. In each recursive call, decrease n by 1 and multiply n to second parameter
11. Stop recursion when n=1

Program Code:

#include <iostream>

template <class t = int >

class Factorial

{

    static t findFact(int n,int a)

    {

        std::cout<<n<<"\*";

        if(n==1)

        {

            std::cout<<"\b \b = ";

            return a;

        }

        return findFact(n-1,a\*n);

    }

public:

    static void checkValidity(t num)

    {

        if(num<0)

            throw INVALID\_NUMBER();

    }

    class INVALID\_NUMBER{};

    static t findFactFor(t num)

    {

        return findFact(num,1);

    }

};

template <class t = long>

class Fibonacci

{

    static t findFibbTerm(int n,int a,int b)

    {

        if(n==0)

            return b;

        return findFibbTerm(n-1,a+b,a);

    }

public:

    class INVALID\_NUMBER{};

    static t findFibb(t num)

    {

        if(num<0)

            throw INVALID\_NUMBER();

        findFibbTerm(num,1,0);

    }

    static void displayFibb(t num)

    {

        if(num<0)

            throw INVALID\_NUMBER();

        std::cout<<"\nThe Fibonacci series is:\n";

        for(t i=1;i<=num;i++)

            std::cout<<findFibbTerm(i,1,0)<<" ";

        std::cout<<'\n';

    }

};

int main()

{

    int num;

    std::cout<<"\nChoose Operation:\n1)Factorial\n2)Fibonacci\t";

    std::cin>>num;

    try

    {

        if(num == 1)

        {

            std::cout<<"\nEnter the number to evaluate factorial:\t";

            std::cin>>num;

            Factorial<int>::checkValidity(num);

            std::cout<<"\nThe factorial of "<<num<<" is "<<Factorial<int>::findFactFor(num);

        }

        else if(num == 2)

        {

            std::cout<<"\nChoose Operation:\n1)Find nth term\n2)Show series upto n\t";

            std::cin>>num;

            if(num == 1)

            {

                std::cout<<"\nEnter 'n' to evaluate Fibonacci:\t";

                std::cin>>num;

                int ans = Fibonacci<>::findFibb(num);

                std::cout<<"\nThe "<<num<<"th Fibonacci term is: "<<ans;

            }

            else if(num == 2)

            {

                std::cout<<"\nEnter 'n' to evaluate Fibonacci:\t";

                std::cin>>num;

                Fibonacci<>::displayFibb(num);

            }

        }

    }

    catch(Factorial<>::INVALID\_NUMBER)

    {

        std::cout<<"\nCannot find factorial of negative number.";

    }

    catch(Fibonacci<>::INVALID\_NUMBER)

    {

        std::cout<<"\nCannot find Fibonacci for negative 'n'.";

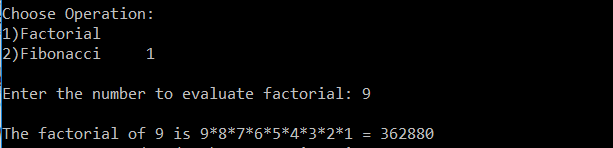
    }

    return 0;

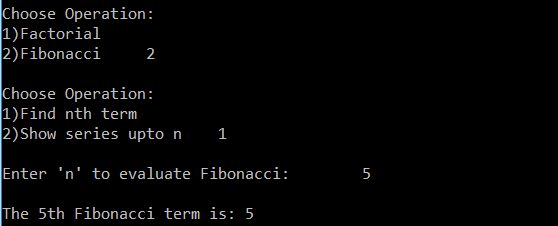
}

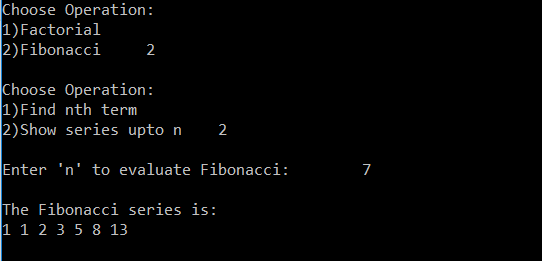
Output:

Factorial:



Fibonacci:





d. SOLVE TOWER OF HANOI PROBLEM FOR ‘N’ DISCS.

Algorithm:

1. Get the number of discs as input from user
2. In a method ‘Hanoi’ call itself to move ‘n-1’ discs from source to middle pole via destination pole
3. Move nth disc from source to destination
4. Call ‘Hanoi’ to move ‘n-1’ discs from the middle pole to destination via the source pole
5. Stop recursion when the number of discs passes is equal to zero.

Program Code:

#include <iostream>

// #include <ctime>

void printInstructions(const char\* source, const char\* destination)

{

    std::cout<<"Move disc from "<<source<<" to "<<destination<<"\n";

}

void hanoi(int num,const char\* source, const char\* via, const char\* destination)

{

    if(num==0)

        return;

    hanoi(num-1,source,destination,via);

    printInstructions(source,destination);

    hanoi(num-1,via,source,destination);

}

int main()

{

   int num;

   std::cout<<"Enter the number of discs:";

   std::cin>>num;

   std::cout<<"The steps to move "<<num<<" discs from A to C via B are:\n";

//    float a =clock();

   hanoi(num,"A","B","C");

//    std::cout<<(clock()-a)/CLOCKS\_PER\_SEC<<'\n';

    return 0;

}

Output:

