**JAVA RECURSIVE**

import java.util.Scanner;

public class javarecursive

{

public void solve(int n, String start, String aux, String end)

{

if(n==1)

System.out.println("Move disc " + n + " " + start + " -> " + end);

else

{

solve(n-1,start,end,aux);

System.out.println("Move disc " + n + " " + start + " -> " + end);

solve(n-1, aux,start,end);

}

}

public static void main(String[] args)

{

javarecursive ob = new javarecursive();

System.out.print("Enter number of discs: ");

Scanner scanner = new Scanner(System.in);

int discs = scanner.nextInt();

ob.solve(discs, "A", "B", "C");

}

}

**JAVA NONRECURSIVE**

import java.io.\*;

import java.util.Stack;

import java.util.EmptyStackException;

public class NON {

public static int legalMove(Stack A, Stack B)

{

int a,b;

try {

a = Integer.parseInt(A.peek().toString());

}

catch(EmptyStackException e){

a = 0;

}

try {

b = Integer.parseInt(B.peek().toString());

}

catch(EmptyStackException e){

b = 0;

}

if(a==b) return 0;

if(a == 0) // If peg A is empty, then pop from B and push the disk onto A

{

A.push(B.pop());

return 2; // Return 2 as move occurred from B to A

}

else if(b == 0) // If peg B is empty, then pop from A and push the disk onto B

{

B.push(A.pop());

return 1; // Return 1 since move occurred from A to B

}

if(a<b)

{

B.push(A.pop());

return 1; // Return 1 since move occurred from A to B

}

else if(a > b) // value of top disk of peg A is greater than the value of topmost disk of peg B

{

A.push(B.pop());

return 2; // Return 2 since move occurred from B to A

}

return -1;

}

public static void main(String[] args) {

// TODO code application logic here

int stepNumber = 0;

int status = 0;

try {

Stack Source = new Stack();

Stack Auxilary = new Stack();

Stack Destination = new Stack();

System.out.println("Enter the number of disks : ");

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(input.readLine());

if(n<=0)

{

System.out.println("Sorry wrong input, negative numbers not allowed.");

System.exit(1);

}

for(int i=n; i>0; i--)

Source.push(i);

int m = n%2;

do {

if(m==1)

{

if((status = legalMove(Source,Destination)) == 1)

System.out.println((++stepNumber) + ". Source --> Destination");

else if (status == 2)

System.out.println((++stepNumber) + ". Destination --> Source");

if((status = legalMove(Source,Auxilary)) == 1)

System.out.println((++stepNumber) + ". Source --> Auxilary");

else if(status == 2)

System.out.println((++stepNumber) + ". Auxilary --> Source");

else

break;

}

else

{

if((status = legalMove(Source,Auxilary)) == 1)

System.out.println((++stepNumber) + ". Source --> Auxilary");

else if (status == 2)

System.out.println((++stepNumber) + ". Auxilary --> Source");

if((status = legalMove(Source,Destination)) == 1)

System.out.println((++stepNumber) + ". Source --> Destination");

else if(status == 2)

System.out.println((++stepNumber) + ". Destination --> Source");

}

if((status = legalMove(Auxilary,Destination)) == 1)

System.out.println((++stepNumber) + ". Auxilary --> Destination");

else if(status == 2)

System.out.println((++stepNumber) + ". Destination --> Auxilary");

}while(Destination.size()!=n);

System.out.println("X-----------------------X");

}

catch (Exception e){

}

}

}

**C# RECURSIVE**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace csharprecursive

{

class TOH

{

int num;

public TOH()

{

num = 0;

}

public TOH(int val)

{

num = val;

}

public int discs

{

get

{

return num;

}

set

{

if (value > 0)

num = value;

}

}

public void move(int n, int from, int to, int aux)

{

if (n > 0)

{

move(n - 1, from, aux, to);

Console.WriteLine("Move disk {0} from {1} to {2}", n, from, to);

move(n - 1, aux, to, from);

}

}

static void Main(string[] args)

{

TOH ob = new TOH();

string discs;

Console.Write("enter number of discs: ");

discs = Console.ReadLine();

ob.num = Convert.ToInt32(discs);

ob.move(ob.num, 1, 2, 3);

Console.ReadLine();

return;

}

}

}

**C#NONRECURSIVE**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace csharprecursive

{

class csharpnonrecursive

{

static int legalMove(Stack<int> A, Stack<int> B)

{

int a, b;

try {

a = A.Peek();

}

catch(Exception e)

{

a = 0;

}

try

{

b = B.Peek();

}

catch (Exception e)

{

b = 0;

}

if (a == b) return 0;

if (a == 0)

{

A.Push(B.Pop());

return 2;

}

else if (b == 0)

{

B.Push(A.Pop());

return 1;

}

if (a < b)

{

B.Push(A.Pop());

return 1;

}

else if (a > b)

{

A.Push(B.Pop());

return 2;

}

return -1;

}

static void Main()

{

int stepNumber = 0;

int status = 0;

try

{

Stack<int> source = new Stack<int>();

Stack<int> aux = new Stack<int>();

Stack<int> dest = new Stack<int>();

Console.WriteLine("enter number of discs: ");

int n = Convert.ToInt32(Console.ReadLine());

if (n <= 0)

{

Console.WriteLine("wrong input");

}

for (int i = n; i > 0; i--)

source.Push(i);

int m = n % 2;

do

{

if (m == 1)

{

if ((status = legalMove(source, dest)) == 1)

Console.WriteLine("'{0}' source --> dest ", (++stepNumber));

else if (status == 2)

Console.WriteLine("'{0}' dest--> source", (++stepNumber));

if ((status = legalMove(source, aux)) == 1)

Console.WriteLine("'{0}' source --> aux", (++stepNumber));

else if (status == 2)

Console.WriteLine("'{0}' aux--> source", (++stepNumber));

else break;

}

else

{

if ((status = legalMove(source, aux)) == 1)

Console.WriteLine("'{0}' source --> aux", (++stepNumber));

else if (status == 2)

Console.WriteLine("'{0}' aux--> source", (++stepNumber));

if ((status = legalMove(source, dest)) == 1)

Console.WriteLine("'{0}' source --> dest", (++stepNumber));

else if (status == 2)

Console.WriteLine("'{0}' dest--> source", (++stepNumber));

}

if ((status = legalMove(aux, dest)) == 1)

Console.WriteLine("'{0}' aux--> dest", (++stepNumber));

else if (status == 2)

Console.WriteLine("'{0}' dest--> aux", (++stepNumber));

}

while (dest.Count != n);

}

catch (Exception e) { Console.WriteLine("'{0}'", e); }

Console.ReadLine();

}

}

}

**C++ RECURSIVE**

#include<iostream>

#include<stdlib.h>

using namespace std;

void tower(int num, char beg, char aux, char end)

{

if (num >= 1)

{

tower(num - 1, beg, end, aux);

cout << "Move disc " << num << " from " << beg << " to " << end << endl;

tower(num - 1, aux, beg, end);

}

}

int main()

{

int N;

cout << "enter number of discs: ";

cin >> N;

tower(N, 'A', 'B', 'C');

system("pause");

return 0;

}

**C++ NONRECURSIVE**

#include <iostream>

#include <stack>

#include <deque>

using namespace std;

int legalMove(stack<int>& A, stack<int>& B)

{

int a, b;

if (!A.empty())

a = A.top();

else

a = 0;

if (!B.empty())

b = B.top();

else

b = 0;

if (a == b)return 0;

if (a == 0) {

if (!B.empty()) {

A.push(B.top());

B.pop();

return 2;

}

}

else if (b == 0) {

if (!A.empty()) {

B.push(A.top());

A.pop();

return 1;

}

}

if (a < b) {

if (!A.empty()) {

B.push(A.top());

A.pop();

return 1;

}

}

else if (a>b) {

if (!B.empty()) {

A.push(B.top());

B.pop();

return 2;

}

}

return -1;

}

void main()

{

int steps = 0;

int status = 0;

int n;

try {

stack<int> source;

stack<int> aux;

stack<int> dest;

cout << "enter number of discs: ";

cin >> n;

if (n <= 0) {

cout << "wrong input" << endl;

return;

}

for (int x = n; x > 0; x--)

{

source.push(x);

}

int m = n % 2;

do {

if (m == 1) {

if ((status = legalMove(source, dest)) == 1)

cout << ++steps << "source --> dest" << endl;

else if (status == 2)

cout << ++steps << "dest --> source" << endl;

if ((status = legalMove(source, aux)) == 1)

cout << ++steps << "source --> aux" << endl;

else if (status == 2)

cout << ++steps << "aux --> source" << endl;

else

break;

}

else {

if ((status = legalMove(source, aux)) == 1)

cout << ++steps << "source --> aux" << endl;

else if (status == 2)

cout << ++steps << "aux --> source" << endl;

if ((status = legalMove(source, dest)) == 1)

cout << ++steps << "source --> dest" << endl;

else if (status == 2)

cout << ++steps << "dest --> source" << endl;

}

if ((status = legalMove(aux, dest)) == 1)

cout << ++steps << "aux --> dest" << endl;

else if (status == 2)

cout << ++steps << "dest --> aux" << endl;

} while (dest.size() != n);

}

catch (exception e) {

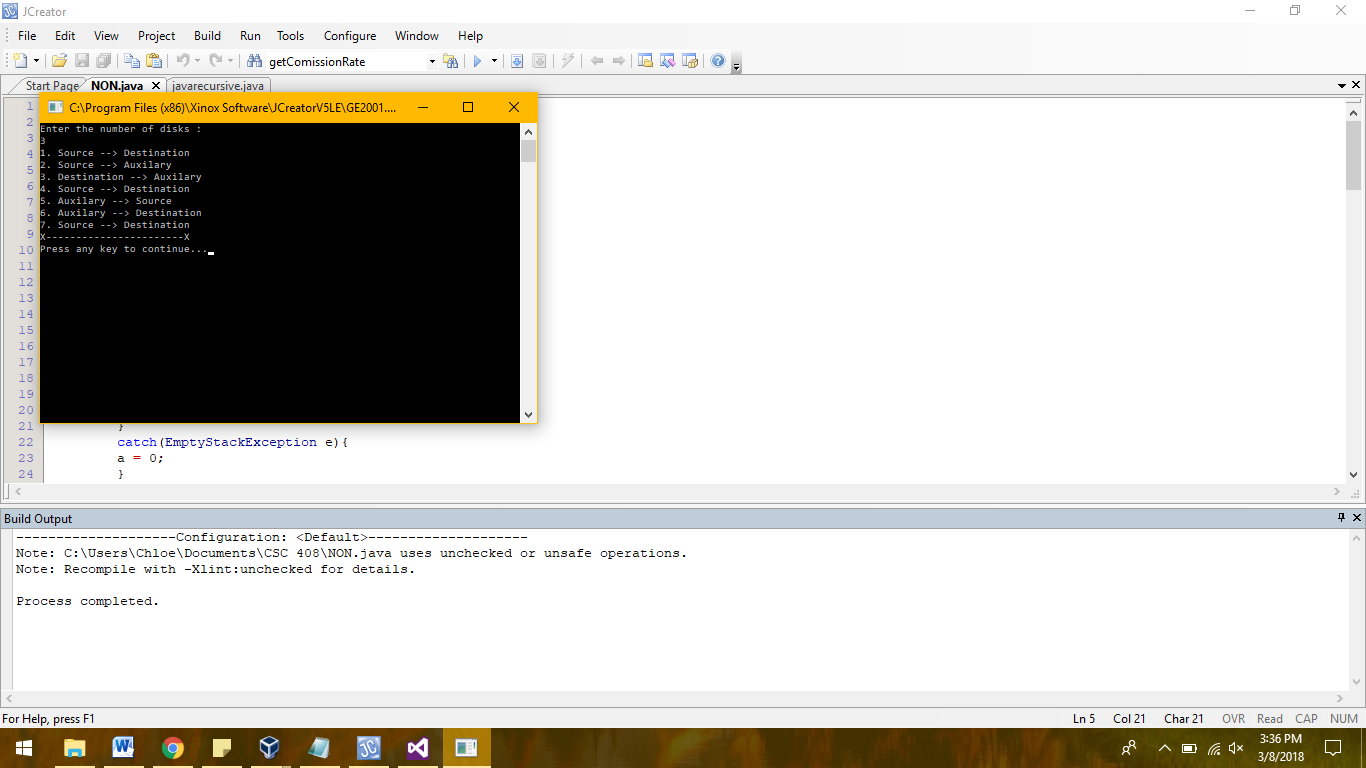
cout << "exception" << endl;

}

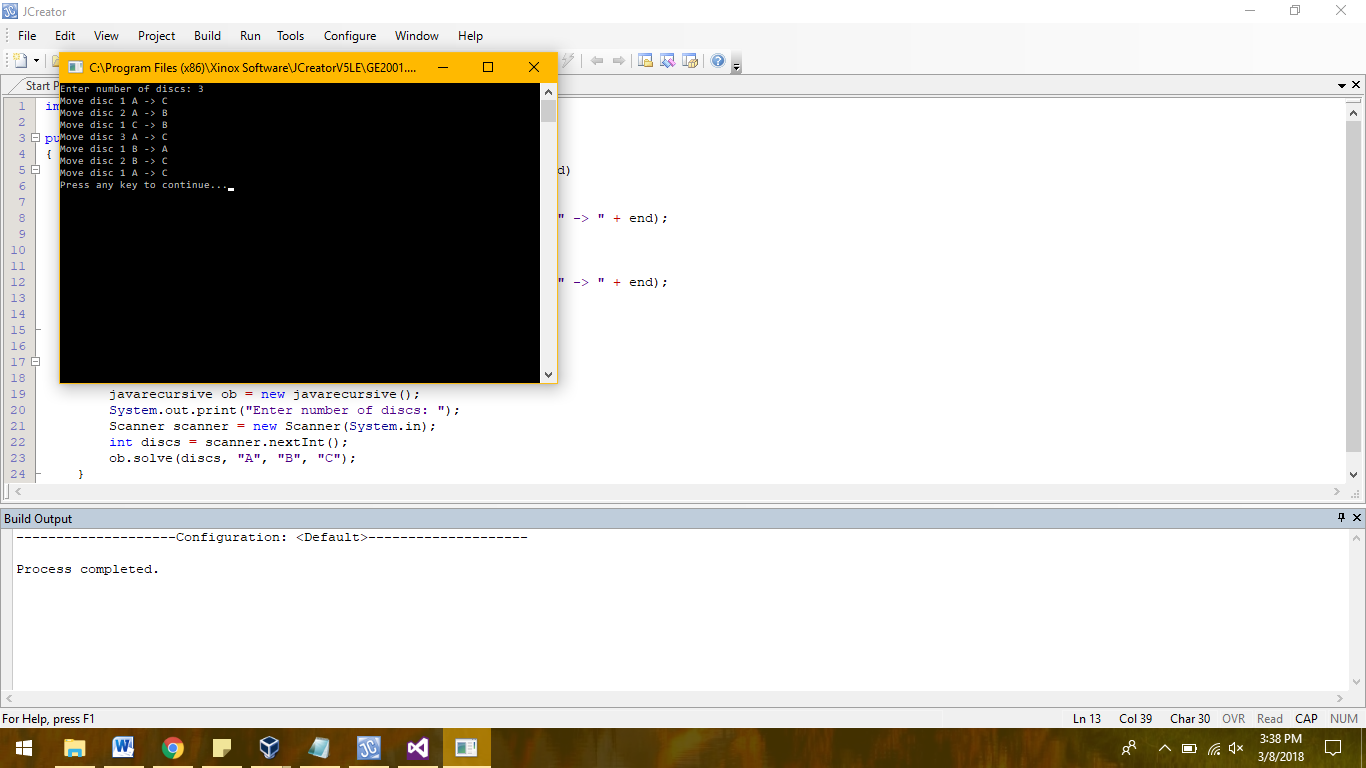
system("pause");

}

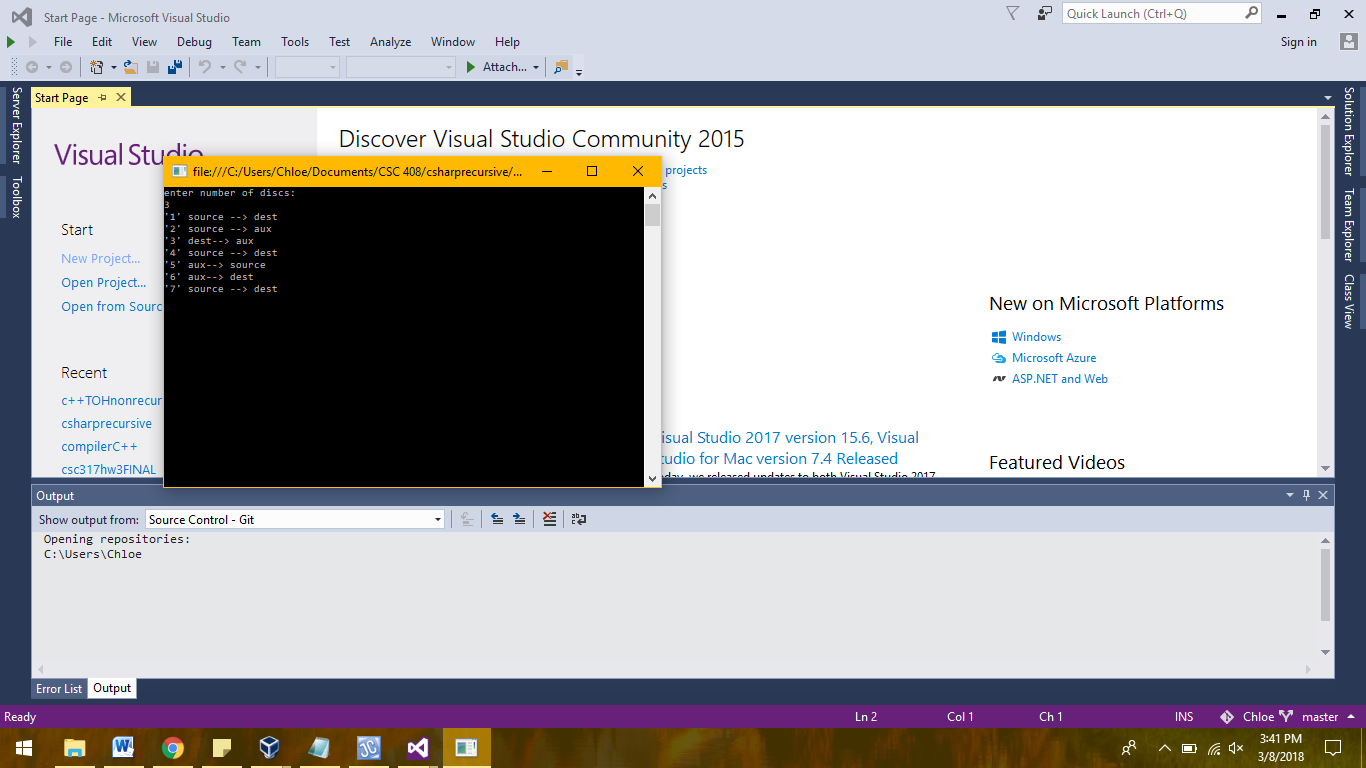
**JAVA NONRECURSIVE**



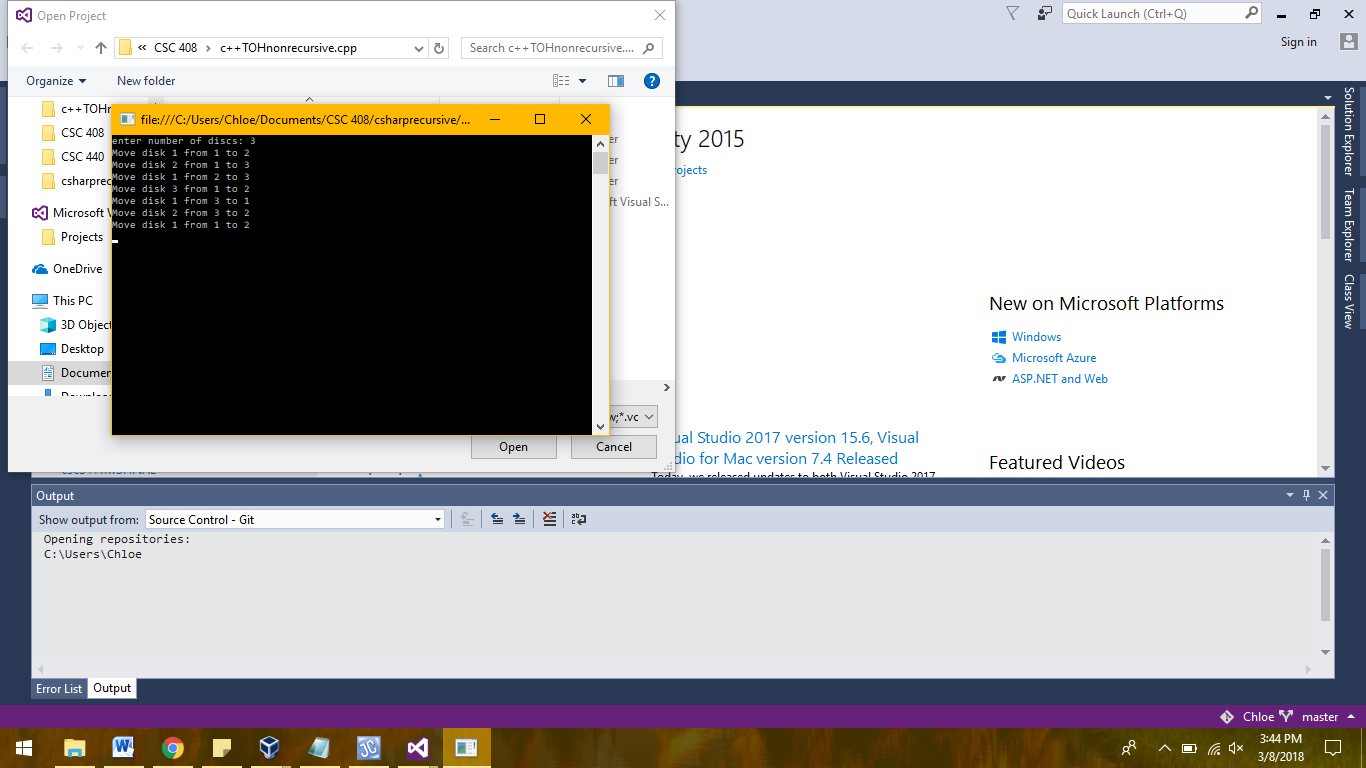
**JAVA RECURSIVE**



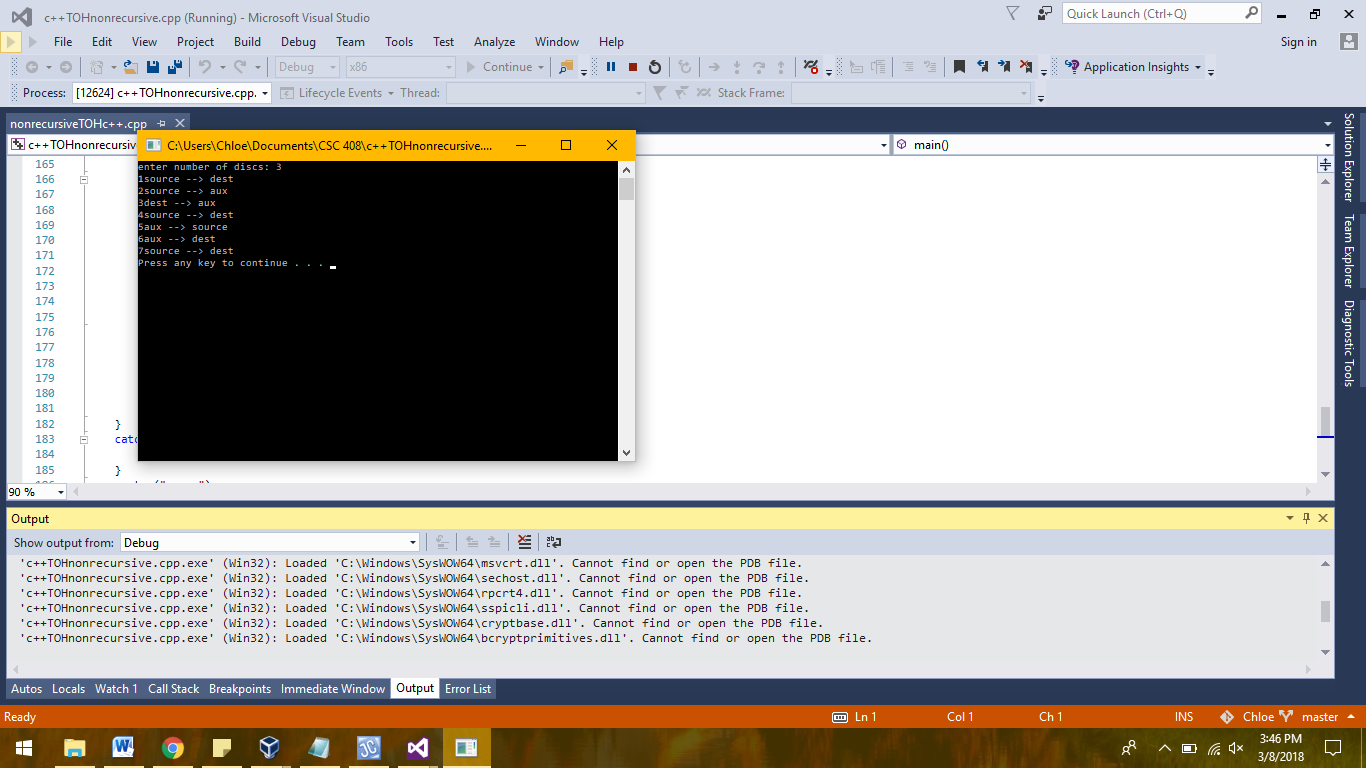
**C# NONRECURSIVE**



**C# RECURSIVE**



**C++ NONRECURSIVE**



**C++ RECURSIVE**

