DSA Assessment – 1

Q. 1) Implement the getMin() function in the stack which can retain the minimum value.

#include <bits/stdc++.h>  
  
using namespace std;  
  
class Stack {  
public:  
 int SIZE;  
 int\* stk;  
 int ptr = -1;  
  
 Stack(int n) {  
 SIZE = n;  
 stk = new int[SIZE];  
 }  
  
 ~Stack() {  
 delete[] stk;  
 }  
  
 void push(int x) {  
 if (isFull()) {  
 cout << "Stack Overflow" << endl;  
 } else {  
 stk[++ptr] = x;  
 }  
 }  
  
 void pop() {  
 if (isEmpty()) cout << "Underflow" << endl;  
 else ptr--;  
 }  
  
 int peek() {  
 if (isEmpty()) return INT\_MIN;  
 return stk[ptr];  
 }  
  
 bool isEmpty() {  
 return ptr == -1;  
 }  
  
 bool isFull() {  
 return ptr == SIZE;  
 }  
};

class MinStack : public Stack {  
public:  
 MinStack(int n) : Stack(n) {}  
  
 int getMin() {  
 if (this->isEmpty()) {  
 cout << "Stack is empty, no minimum." << endl;  
 return INT\_MAX;  
 }  
  
 Stack tempStack(SIZE);  
 int minElement = this->peek();  
  
 while (!this->isEmpty()) {  
 int element = this->peek();  
 minElement = min(minElement, element);  
 tempStack.push(element);  
 this->pop();  
 }  
  
 // build original stack  
 while (!tempStack.isEmpty()) {  
 this->push(tempStack.peek());  
 tempStack.pop();  
 }  
  
 return minElement;  
 }  
};

Test Case 1:

// --- Test Case 1: Empty Stack ---  
// This test checks the behavior of getMin() and peek() on a stack with no elements.  
int main() {  
 cout << "--- Test Case 1: Empty Stack ---" << endl;  
 MinStack\* stk = new MinStack(5);  
  
 cout << "Minimum Element: " << stk->getMin() << endl;  
 cout << "Stack top: " << stk->peek() << endl;  
  
 delete stk;  
 return 0;  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 2:

// --- Test Case 2: Stack with Negative Numbers and Duplicates ---  
// This test ensures getMin() works correctly with negative values and  
// when the minimum value appears more than once.  
int main() {  
 cout << "--- Test Case 2: Negative Numbers and Duplicates ---" << endl;  
 MinStack\* stk = new MinStack(5);  
 stk->push(-10);  
 stk->push(-5);  
 stk->push(-20);  
 stk->push(0);  
 stk->push(-20);  
  
 cout << "Stack top: " << stk->peek() << endl; // Should be -20  
 cout << "Minimum Element: " << stk->getMin() << endl; // Should be -20  
  
 stk->pop(); // Removes the top -20  
 cout << "Popped one element." << endl;  
 cout << "New stack top: " << stk->peek() << endl; // Should be 0  
 cout << "Minimum Element after pop: " << stk->getMin() << endl; // Should still be -20  
  
 delete stk;  
 return 0;  
}

Output:

A computer screen shot of a black screen

AI-generated content may be incorrect.

Test case 3:

// --- Test Case 3: Minimum Element at the Bottom ---  
// This test verifies that getMin() can find the minimum element even if it was the  
// first one pushed (i.e., it's at the bottom of the stack).  
int main() {  
 cout << "--- Test Case 3: Minimum at the Bottom ---" << endl;  
 MinStack\* stk = new MinStack(5);  
 stk->push(1); // Minimum element  
 stk->push(100);  
 stk->push(50);  
 stk->push(200);  
 stk->push(25);  
  
 cout << "Stack top: " << stk->peek() << endl; // Should be 25  
 cout << "Minimum Element: " << stk->getMin() << endl; // Should be 1  
  
 // Verify the stack is unchanged after getMin()  
 cout << "Stack top after getMin(): " << stk->peek() << endl; // Should still be 25  
  
 delete stk;  
 return 0;  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 4:

// --- Test Case 4: Stack Overflow and Underflow ---  
// This test checks the boundary conditions: attempting to push to a full stack  
// and pop from an empty stack.  
int main() {  
 cout << "--- Test Case 4: Overflow and Underflow ---" << endl;  
 MinStack\* stk = new MinStack(3); // A small stack  
 cout << "Stack size is 3." << endl;  
  
 stk->push(10);  
 stk->push(20);  
 stk->push(5); // Stack is now full  
  
 cout << "Current Minimum: " << stk->getMin() << endl; // Should be 5  
 cout << "Attempting to push to a full stack..." << endl;  
 stk->push(99); // Should print "Stack Overflow"  
  
 cout << "Popping all elements..." << endl;  
 stk->pop();  
 stk->pop();  
 stk->pop(); // Stack is now empty  
  
 cout << "Attempting to pop from an empty stack..." << endl;  
 stk->pop(); // Should print "Stack Underflow"  
  
 delete stk;  
 return 0;  
}

Output:

A screenshot of a computer program

AI-generated content may be incorrect.

Test Case 5:

// --- Test Case 5: Interleaved Operations ---  
// This test simulates more complex usage by mixing push, pop, and getMin calls  
// to ensure the stack state remains consistent.  
int main() {  
 cout << "--- Test Case 5: Interleaved Operations ---" << endl;  
 MinStack\* stk = new MinStack(8);  
 stk->push(2);  
 stk->push(0);  
 stk->push(3);  
 stk->push(0);  
 cout << "Minimum is: " << stk->getMin() << endl; // Should be 0  
 stk->pop(); // Removes a 0  
 cout << "Popped. Top is now: " << stk->peek() << endl; // Should be 3  
 cout << "Minimum is: " << stk->getMin() << endl; // Should still be 0  
 stk->pop(); // Removes 3  
 cout << "Popped. Top is now: " << stk->peek() << endl; // Should be 0  
 cout << "Minimum is: " << stk->getMin() << endl; // Should still be 0  
 stk->pop(); // Removes the last 0  
 cout << "Popped. Top is now: " << stk->peek() << endl; // Should be 2  
 cout << "Minimum is: " << stk->getMin() << endl; // Should now be 2  
  
 delete stk;  
 return 0;  
}

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Q. 2) Implement the traversal between webpages in a browser using stack

Code:

#include <bits/stdc++.h>  
  
using namespace std;  
  
  
template <typename A>  
class Stack {  
public:  
 int SIZE;  
 A\* stk;  
 int ptr = -1;  
  
 Stack(int n) {  
 SIZE = n;  
 stk = new A[SIZE];  
 }  
  
 Stack () {  
 SIZE = 1000;  
 stk = new A[SIZE];  
 ptr = -1;  
 }  
  
 ~Stack() {  
 delete[] stk;  
 }  
  
 void push(A x) {  
 if (isFull()) {  
 cout << "Stack Overflow" << endl;  
 } else {  
 stk[++ptr] = x;  
 }  
 }  
  
 void pop() {  
 if (isEmpty()) cout << "Underflow" << endl;  
 else ptr--;  
 }  
  
 A peek() {  
 if (isEmpty()) return A();  
 return stk[ptr];  
 }  
  
 bool isEmpty() {  
 return ptr == -1;  
 }  
  
 bool isFull() {  
 return ptr == SIZE - 1;  
 }  
};  
  
class Browser {  
 Stack<string> backStack;  
 Stack<string> forwardStack;  
 string currPage;  
public:  
 Browser(string firstPage) : currPage(firstPage) {}  
  
 string getCurrentPage() {  
 return ! currPage.empty() ? currPage : "No current page available";  
 }  
  
 void goToPage(const string& page) {  
 backStack.push(currPage);  
 currPage = page;  
 while (! forwardStack.isEmpty()) forwardStack.pop();  
 }  
  
 void moveForwardToPage() {  
 if (forwardStack.isEmpty()) {  
 cout << "Forward Stack is empty. Can't move forward." << endl;  
 return;  
 }  
 backStack.push(currPage);  
 currPage = forwardStack.peek(); forwardStack.pop();  
 }  
  
 void moveBackToPage() {  
 if (backStack.isEmpty()) {  
 cout << "Back Stack is empty. Can't move back." << endl;  
 return;  
 }  
 forwardStack.push(currPage);  
 currPage = backStack.peek(); backStack.pop();  
 }  
};

Test Case 1:

int main() {  
 cout << "--- Test Case 1: Clearing the Forward Stack ---" << endl;  
 Browser browser("google.com");  
 browser.goToPage("github.com");  
 browser.goToPage("linkedin.com");  
  
 cout << "Current Page: " << browser.getCurrentPage() << endl; // linkedin.com  
 browser.moveBackToPage();  
 cout << "Moved Back. Current Page: " << browser.getCurrentPage() << endl; // github.com  
  
 // At this point, "linkedin.com" is in the forward stack.  
 // Going to a new page should clear it.  
 cout << "Going to a new page..." << endl;  
 browser.goToPage("stackoverflow.com");  
 cout << "Current Page: " << browser.getCurrentPage() << endl; // stackoverflow.com  
  
 // Now, trying to move forward should fail.  
 browser.moveForwardToPage(); // Should print "Forward Stack is empty."  
 cout << "Current Page after trying to go forward: " << browser.getCurrentPage() << endl; // Should still be stackoverflow.com  
 return 0;  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 2:

// --- Test Case 2: Simple Navigation ---  
// This test checks the basic flow: visiting a few pages, going back,  
// and then going forward.  
int main() {  
 cout << "--- Test Case 2: Simple Navigation ---" << endl;  
 Browser browser("homepage.com");  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 browser.goToPage("about.com");  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 browser.goToPage("contact.com");  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 browser.moveBackToPage();  
 cout << "Moved Back. Current Page: " << browser.getCurrentPage() << endl; // Should be about.com  
  
 browser.moveForwardToPage();  
 cout << "Moved Forward. Current Page: " << browser.getCurrentPage() << endl; // Should be contact.com  
 return 0;  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 3:

// --- Test Case 3: Boundary Conditions ---  
// This test attempts to move back and forward at the very beginning  
// of a session, when the stacks are empty.  
int main() {  
 cout << "--- Test Case 3: Boundary Conditions ---" << endl;  
 Browser browser("initial-page.org");  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 // Try to move back when the back stack is empty  
 browser.moveBackToPage(); // Should print "Back Stack is empty."  
 cout << "Current Page after trying to go back: " << browser.getCurrentPage() << endl;  
  
 // Try to move forward when the forward stack is empty  
 browser.moveForwardToPage(); // Should print "Forward Stack is empty."  
 cout << "Current Page after trying to go forward: " << browser.getCurrentPage() << endl;  
   
 return 0;  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Test Case 4:

// --- Test Case 4: Multiple Back and Forth ---  
// This test simulates a user navigating back and forth multiple times  
// to ensure the stacks are managed correctly in a sequence.  
int main() {  
 cout << "--- Test Case 4: Multiple Back and Forth ---" << endl;  
 Browser browser("PageA");  
 browser.goToPage("PageB");  
 browser.goToPage("PageC");  
 browser.goToPage("PageD");  
  
 cout << "Current Page: " << browser.getCurrentPage() << endl; // PageD  
  
 browser.moveBackToPage(); // -> PageC  
 browser.moveBackToPage(); // -> PageB  
 cout << "Current Page after two backs: " << browser.getCurrentPage() << endl; // PageB  
  
 browser.moveForwardToPage(); // -> PageC  
 cout << "Current Page after one forward: " << browser.getCurrentPage() << endl; // PageC  
  
 browser.moveBackToPage(); // -> PageB  
 browser.moveBackToPage(); // -> PageA  
 cout << "Current Page after two more backs: " << browser.getCurrentPage() << endl; // PageA  
  
 // Try to go back one more time (should fail)  
 browser.moveBackToPage(); // Should print "Back Stack is empty."  
 cout << "Current Page after final back attempt: " << browser.getCurrentPage() << endl; // PageA  
 return 0;  
}

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Test Case 5:

// --- Test Case 5: Emptying Stacks Completely ---  
// This test involves navigating and then using the back button until the  
// original page is reached and the back stack is empty.  
int main() {  
 cout << "--- Test Case 5: Emptying Stacks Completely ---" << endl;  
 Browser browser("Start.com");  
 browser.goToPage("Middle.com");  
 browser.goToPage("End.com");  
  
 cout << "Current Page: " << browser.getCurrentPage() << endl; // End.com  
  
 cout << "Moving all the way back..." << endl;  
 browser.moveBackToPage(); // -> Middle.com  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
 browser.moveBackToPage(); // -> Start.com  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 // Back stack should now be empty.  
 browser.moveBackToPage(); // Should print "Back Stack is empty."  
  
 cout << "Moving all the way forward..." << endl;  
 browser.moveForwardToPage(); // -> Middle.com  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
 browser.moveForwardToPage(); // -> End.com  
 cout << "Current Page: " << browser.getCurrentPage() << endl;  
  
 // Forward stack should now be empty.  
 browser.moveForwardToPage(); // Should print "Forward Stack is empty."  
 return 0;  
}

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Q. 3) Write a program to convert the given infix expression to Polish notation & reverse Polish notation. Also, evaluate them using stack.

Code:

#include <iostream>  
#include <string>  
#include <stack>  
#include <unordered\_map>  
#include <cmath> // For pow()  
#include <cctype> // For isdigit()  
#include <algorithm>  
  
using namespace std;  
  
template <typename A>  
class Stack {  
public:  
 int SIZE;  
 A\* stk;  
 int ptr = -1;  
  
 Stack(int n) {  
 SIZE = n;  
 stk = new A[SIZE];  
 }  
  
 Stack () {  
 SIZE = 1000;  
 stk = new A[SIZE];  
 ptr = -1;  
 }  
  
 ~Stack() {  
 delete[] stk;  
 }  
  
 void push(A x) {  
 if (isFull()) {  
 cout << "Stack Overflow" << endl;  
 } else {  
 stk[++ptr] = x;  
 }  
 }  
  
 void pop() {  
 if (isEmpty()) cout << "Underflow" << endl;  
 else ptr--;  
 }  
  
 A peek() {  
 if (isEmpty()) return A();  
 return stk[ptr];  
 }  
  
 bool isEmpty() {  
 return ptr == -1;  
 }  
  
 bool isFull() {  
 return ptr == SIZE - 1;  
 }  
  
 int size() {  
 if (ptr == -1) return 0;  
 return ptr + 1;  
 }  
};  
  
  
int operatorPrecedence(char op) {  
 unordered\_map<char, int> opMap;  
 opMap['^'] = 3;  
 opMap['\*'] = 2;  
 opMap['/'] = 2;  
 opMap['+'] = 1;  
 opMap['-'] = 1;  
 // Returns 0 if op is not found (e.g., for parentheses), which works for the logic.  
 if (opMap.count(op)) {  
 return opMap[op];  
 }  
 return 0;  
}  
  
string getReversePolish(const string& infix) {  
 string postFix;  
 Stack<char> stk;  
  
 for (int i = 0; i < infix.length(); i++) {  
 char token = infix[i];  
  
 // skip spaces  
 if (isspace(token)) continue;  
  
 if (isdigit(token)) {  
 string number;  
 while (i < infix.length() && isdigit(infix[i])) {  
 number.push\_back(infix[i]);  
 i++;  
 }  
  
 // We currently don't know how infix[i] should be handled. If we don't decrement it, we will skip infix[i]  
 i--; // Decrement i to offset the for-loop's increment  
  
 postFix.append(number);  
 postFix.push\_back(' '); // Add space delimiter after the number  
 }  
 else if (token == '(') {  
 stk.push(token);  
 }  
 else if (token == ')') {  
 while (!stk.isEmpty() && stk.peek() != '(') {  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' ');  
 stk.pop();  
 }  
 if (!stk.isEmpty()) {  
 stk.pop(); // Pop the opening parenthesis '('  
 }  
 else {  
 cerr << "Mismatched parenthesis found" << endl;  
 return ""; // we will handle this case in main function  
 }  
 }  
 else if (string("+-\*/^").find(token) != string::*npos*) {  
 while (  
 ! stk.isEmpty() &&  
 stk.peek() != '(' &&  
 (  
 // Pop if operator on stack has higher precedence  
 operatorPrecedence(stk.peek()) > operatorPrecedence(token) ||  
 // OR if they have equal precedence AND the token is left-associative  
 (operatorPrecedence(stk.peek()) == operatorPrecedence(token) && token != '^')  
 )  
 ){  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' '); // add delimiter  
 stk.pop();  
 }  
 stk.push(token);  
 }  
 }  
  
 // Pop any remaining operators from the stack to the output string  
 while (!stk.isEmpty()) {  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' ');  
 stk.pop();  
 }  
  
 return postFix;  
}  
  
double getOperationResult(char op, double first, double second) {  
 if (op == '+')  
 return first + second;  
 if (op == '-')  
 return first - second;  
 if (op == '\*')  
 return first \* second;  
 if (op == '/') {  
 // Handle division by zero  
 if (second == 0) {  
 cerr << "Error: Division by zero." << endl;  
 return 0;  
 }  
 return first / second;  
 }  
 if (op == '^')  
 return pow(first, second);  
 return 0.0;  
}  
  
double evaluateReversePolish(string postfix) {  
 Stack<double> stk;  
  
 for (int i = 0; i < postfix.length(); ++i) {  
 char token = postfix[i];  
  
 // If the token is a digit, parse the full number  
 if (isdigit(token)) {  
 string number;  
 while (i < postfix.length() && isdigit(postfix[i])) {  
 number.push\_back(postfix[i]);  
 i++;  
 }  
 // i is now at the space after the number, loop will increment past it  
 stk.push(stod(number)); // stod converts string to double  
 }  
 else if (string("+-\*/^").find(token) != string::*npos*) {  
 // Pop the two operands from the stack  
 double firstPop = stk.peek(); stk.pop();  
 double secondPop = stk.peek(); stk.pop();  
 // Perform operation and push result back onto stack  
 double res = getOperationResult(token, secondPop, firstPop);  
 stk.push(res);  
 }  
 // we ignore spaces  
 }  
  
 if (stk.size() > 1) {  
 cout << "Invalid format found. Try again" << endl;  
 return INT\_MIN;  
 }  
  
 // The final result is the last item on the stack  
 return stk.peek();  
}  
  
string getReversePolishForPolish(const string& infix) {  
 string postFix;  
 Stack<char> stk;  
  
 for (int i = 0; i < infix.length(); i++) {  
 char token = infix[i];  
  
 if (isspace(token)) continue;  
  
 if (isdigit(token)) {  
 string number;  
 while (i < infix.length() && isdigit(infix[i])) {  
 number.push\_back(infix[i]);  
 i++;  
 }  
 i--;  
 postFix.append(number);  
 postFix.push\_back(' ');  
 }  
 else if (token == ')') { // Note: in the reversed string, this was originally ')'  
 stk.push(token);  
 }  
 else if (token == '(') { // Note: in the reversed string, this was originally '('  
 while (!stk.isEmpty() && stk.peek() != ')') {  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' ');  
 stk.pop();  
 }  
 if (!stk.isEmpty()) {  
 stk.pop();  
 }  
 else {  
 cerr << "Mismatched parenthesis found" << endl;  
 return "";  
 }  
 }  
 else if (string("+-\*/^").find(token) != string::*npos*) {  
 // This while loop condition is the critical change.  
 while (  
 !stk.isEmpty() &&  
 stk.peek() != ')' &&  
 (  
 // Pop if operator on stack has strictly higher precedence  
 operatorPrecedence(stk.peek()) > operatorPrecedence(token) ||  
 // OR if they have equal precedence AND the token is now LEFT-associative (i.e. '^')  
 (operatorPrecedence(stk.peek()) == operatorPrecedence(token) && token == '^')  
 )  
 ){  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' ');  
 stk.pop();  
 }  
 stk.push(token);  
 }  
 }  
  
 while (!stk.isEmpty()) {  
 postFix.push\_back(stk.peek());  
 postFix.push\_back(' ');  
 stk.pop();  
 }  
  
 return postFix;  
}  
  
string getPolishExpression(const string& infix) {  
 string prefix = infix;  
  
 reverse(prefix.begin(), prefix.end());  
 prefix = getReversePolishForPolish(prefix); // the only difference from the original function is that we remove  
 // the right associativity of ^ for reversing.  
 reverse(prefix.begin(), prefix.end());  
 return prefix;  
}  
  
double evaluatePolishExpression(const string& prefix) {  
 Stack<double> stk;  
  
 for (int i = prefix.length() - 1; i >= 0; i--) {  
 char token = prefix[i];  
 if (isdigit(token)) {  
 string number;  
 while (i >= 0 && isdigit(prefix[i])) {  
 number.push\_back(prefix[i]);  
 i--;  
 }  
 // we have definitely encountered a space here - since we added spaces explicitly in our prefix expression  
 // therefore, it's safe to not decrement i  
 reverse(number.begin(), number.end()); // single we are parsing right to left  
 stk.push(stod(number));  
 }  
 else if (string("+\*-/^").find(token) != string::*npos*) {  
 double firstPop = stk.peek(); stk.pop();  
 double secondPop = stk.peek(); stk.pop();  
 stk.push(getOperationResult(token, firstPop, secondPop));  
 }  
 // ignore the spaces  
 }  
  
 if (stk.size() > 1) {  
 cout << "Invalid format found. Try again" << endl;  
 return INT\_MIN;  
 }  
  
 return stk.peek();  
}  
  
  
int main() {  
 string infix;  
 cout << "Enter an infix expression: " << endl;  
 getline(cin, infix); // getline reads the entire line including spaces  
  
 cout << "Enter one of the following options (1/2)" << endl;  
 cout << "1. Convert to Polish Notation and evaluate" << endl;  
 cout << "2. Convert to Reverse Polish Notation and evaluate" << endl;  
 char option;  
 cin >> option;  
 double result;  
  
 if (option == '1') {  
 string prefix = getPolishExpression(infix);  
 if (prefix.empty()) {  
 cout << "Invalid format found. Try again" << endl;  
 return 0;  
 }  
 cout << "Polish Notation expression: " << prefix << endl;  
 result = evaluatePolishExpression(prefix);  
 cout << "Result from Polish Notation expression: " << result << endl;  
 }  
 else if (option == '2') {  
 string postfix = getReversePolish(infix);  
 if (postfix.empty()) {  
 cout << "Invalid format found. Try again" << endl;  
 return 0;  
 }  
 cout << "Reverse Polish Notation expression: " << postfix << endl;  
  
 result = evaluateReversePolish(postfix);  
 cout << "Result from Reverse Polish Notation: " << result << endl;  
 }  
 else {  
 cout << "Invalid option. Try again" << endl;  
 }  
 return 0;  
}

Terminal Window Output for three test cases:

1.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

2.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

3.

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

Q. 4) Solve the Tower of Hanoi problem.

Code:

#include <bits/stdc++.h>  
  
using namespace std;  
  
  
void hanoi(int n, char src, char dest, char aux) {  
 if (n == 1) {  
 cout << "Move disk 1 from " << src << " to " << dest << endl;  
 return;  
 }  
 hanoi(n - 1, src, aux, dest);  
 cout << "Move disk " << n << " from " << src << " to " << dest << endl;  
 hanoi(n - 1, aux, dest, src);  
}  
  
  
int main() {  
 int n;  
 cout << "Enter the number of disks: " << endl;  
 cin >> n;  
 hanoi(n, 'A', 'C', 'B');  
 return 0;  
}

Test cases:

1.

A screen shot of a computer

AI-generated content may be incorrect.

2.

A screenshot of a computer program

AI-generated content may be incorrect.

Inference: It takes 2^n – 1 moves to solve the problem where ‘n’ is the number of disks.

The entire source code can be found on Github: https://github.com/qburi/DSA-LAB