**Discrete Structure Final Project Fall 2024**

**Group Members: 23F-0572 Muhammad Hamza Saleem**

**23F-0672 Waheed Zafar**

**23F-0706 Usman Ashfaq**

**Question#1:**

#include <iostream>

#include <string>

#include <stack>//This library is used to handle infix to postfix expression in this code

#include <iomanip>// for spacing

using namespace std;

const int mav\_variable = 10;

int prec(char c) {

if (c == '!') {

return 3;

} // NOT

if (c == '&') {

return 2;

}

// AND

if (c == '|') {

return 2;

}

// OR

if (c == '>') {

return 1;

}

// Implication

if (c == '=') {

return 1;

} // Equivalence

return 0; // Parentheses or invalid

}

bool evaluate(char op, bool a, bool b) {

switch (op) {// in this switch statement we donot use break statement because we use return in this code when the reurn statement is run then its automatically leave the function

case '&':

return a && b; // AND

case '|':

return a || b; // OR

case '>':

return (!a || b); // a -> b

case '=':

return (a == b); // a == b

default:

return false;

}

}

bool evaluateExpression(const string& str, bool arr[mav\_variable], const char variables[mav\_variable], int varCount) {

bool stack[mav\_variable];

int top = -1;

for (int i = 0; i < str.size(); i++) {

char c = str[i];

if (isalpha(c)) {//isalpha() is used to check the both lower and upper value of charact like a to z & A to Z

int temp = 0;

while (temp < varCount && variables[temp] != c) temp++;

stack[++top] = arr[temp];

}

else if (c == '!') {

stack[top] = !stack[top];// This condion used for Not operator

}

else {// This condition is used evaluate operators except not

bool b = stack[top--];

bool a = stack[top--];

stack[++top] = evaluate(c, a, b);

}

}

return stack[top];

}

string infixToPostfix(const string& str, char arr[mav\_variable], int& varCount) {//infixto postfix expression mean According the adjust of evaluation computer like higher to lower order

string res;

stack<char> st;

for (int i = 0; i < str.length(); i++) {

char c = str[i];

if (isalpha(c)) {

res += c;

bool found = false;

for (int j = 0; j < varCount; j++) {

if (arr[j] == c) {

found = true;

break;

}

}

if (!found) arr[varCount++] = c;

}

else if (c == '(') {

st.push(c);

}

else if (c == ')') {

while (!st.empty() && st.top() != '(') {

res += st.top();

st.pop();

}

if (!st.empty()) st.pop();

}

else {

while (!st.empty() && prec(st.top()) >= prec(c)) {

res += st.top();

st.pop();

}

st.push(c);

}

}

while (!st.empty()) {

res += st.top();

st.pop();

}

return res;

}

void generateTruthTable(bool\*\* table, int varCount, int rows) {

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < varCount; ++j) {

int temp = 1;

for (int k = 0; k < (varCount - j - 1); ++k) {

temp \*= 2;

}

table[j][i] = (i % (temp \* 2)) >= temp; // Check if i falls in the range for the current variable

}

}

}

void displayTruthTable(const char variables[mav\_variable], int varCount, bool\*\* truthTable,

const bool results1[], const bool results2[], int rows) {

cout << setw(8);

for (int i = 0; i < varCount; ++i) {

cout << variables[i] << setw(5);

}

cout << " Expr1" << setw(8) << "Expr2" << endl;

cout << string(8 + (varCount \* 5) + 15, '-') << endl;

for (int i = 0; i < rows; ++i) {

cout << setw(8);

for (int j = 0; j < varCount; ++j) {

cout << truthTable[j][i] << setw(5);

}

cout << results1[i] << setw(8) << results2[i] << endl;

}

}

int main() {

string expr1, expr2;

cout << "Enter the first logical expression: ";

cin >> expr1;

cout << "Enter the second logical expression: ";

cin >> expr2;

char arr[mav\_variable];

int count = 0;

string postfix1 = infixToPostfix(expr1, arr, count);

string postfix2 = infixToPostfix(expr2, arr, count);

int rows = 1;

for (int i = 0; i < count; ++i) {

rows \*= 2;

}

bool\*\* truthTable = new bool\* [mav\_variable];

for (int i = 0; i < mav\_variable; ++i) {

truthTable[i] = new bool[rows];

}

generateTruthTable(truthTable, count, rows);

bool\* results1 = new bool[rows];

bool\* results2 = new bool[rows];

for (int i = 0; i < rows; ++i) {

bool values[mav\_variable];

for (int j = 0; j < count; ++j) {

values[j] = truthTable[j][i];

}

results1[i] = evaluateExpression(postfix1, values, arr, count);

results2[i] = evaluateExpression(postfix2, values, arr, count);

}

displayTruthTable(arr, count, truthTable, results1, results2, rows);

bool equivalent = true;

for (int i = 0; i < rows; ++i) {//This condition check the true or false my expression

if (results1[i] != results2[i]) {

equivalent = false;

break;

}

}

if (equivalent) {

cout << "The two expressions are logically equivalent.\n";

}

else {

cout << "The two expressions are not logically equivalent.\n";

}

delete[] results1;

delete[] results2;

for (int i = 0; i < mav\_variable; ++i) {

delete[] truthTable[i];

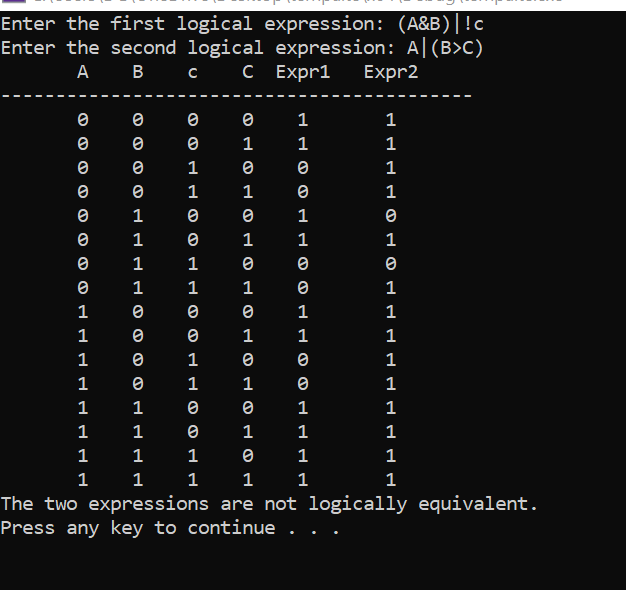
}

delete[] truthTable;

system("pause");

return 0;

}



**Question #2:**

#include <iostream>

#include <stack>

#include<iomanip>

#include<Windows.h>

using namespace std;

class Graph

{

private:

int vertices;

int Matrix[20][20];

int path[20];

bool visited[20];

//for Hamiltonian

bool Check\_Hamilton(int vertex, int count)

{

if (count == vertices)

{

int last\_Vertex = path[count - 1];

int first\_Vertex = path[0];

if (Matrix[last\_Vertex][first\_Vertex] == 1) //placed last vertex at start bcz

{ //to check edge between last and first

return true; //vertex

}

else

{

return false;

}

}

for (int i = 0; i < vertices; i++)

{

if (Matrix[vertex][i] == 1 && visited[i] != true)

{

visited[i] = true;

path[count] = i;

if (Check\_Hamilton(i, count + 1) == true)

{

return true;

}

visited[i] = false;

}

}

return false;

}

//For euler circuit

bool Euler\_Cicuit()

{

for (int i = 0; i < vertices; i++)

{

int degree = 0;

for (int j = 0; j < vertices; ++j)

{

degree += Matrix[i][j];

}

if (degree % 2 != 0) //return False if any vertex encounters odd degree

{

return false;

}

}

return true;

}

//for Euler Path

bool Check\_EulerPath()

{

int oddCount = 0;

for (int i = 0; i < vertices; ++i)

{

int degree = 0;

for (int j = 0; j < vertices; ++j)

{

degree += Matrix[i][j];

}

if (degree % 2 != 0)

{

++oddCount;

}

}

if ((oddCount == 2) || (oddCount == 0))

{

return true;

}

}

void Find\_Euler\_Circuit(int ver, int Copy\_Matrix[20][20], int Cirxuit\_Arr[], int& Pos)

{

for (int v = 0; v < vertices; v++)

{

if (Copy\_Matrix[ver][v] > 0) //to check whther the edge exists or not

{

Copy\_Matrix[ver][v]--;

Copy\_Matrix[v][ver]--; //here ver is the vertex passed to it

Find\_Euler\_Circuit(v, Copy\_Matrix, Cirxuit\_Arr, Pos);

// Recursively visit next vertex for onr vertx

}

}

Cirxuit\_Arr[Pos++] = ver; // Finally Add vertex to circuit

}

public:

Graph(int v) : vertices(v)

{

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < 20; j++)

{

Matrix[i][j] = 0;

}

}

}

void addEdge(int u, int v)

{

Matrix[u][v] = 1;

Matrix[v][u] = 1;

}

//checking for the Hamiltoninan Circuit if exists or not

void Check\_Hamilton\_Circuit()

{

for (int i = 0; i < vertices; ++i)

{

visited[i] = false;

}

path[0] = 0;

visited[0] = true;

if (Check\_Hamilton(0, 1) == true) //here passed 0 vertex and 1 is the visted value = 1

{

cout << "Hamiltonian Circuit found: { ";

for (int i = 0; i < vertices; i++)

{

cout << path[i] << " ";

}

cout << path[0] << " }\n";

}

else

{

cout << "No Hamiltonian Circuit exists.\n";

}

}

//checking for the Euler Circuit if exists or not

void Check\_euler\_Cicuit()

{

if (!Euler\_Cicuit())

{

cout << "No Euler Circuit exists.\n";

return;

}

int tempMatrix[20][20];

for (int i = 0; i < vertices; i++)

{

for (int j = 0; j < vertices; j++)

{

tempMatrix[i][j] = Matrix[i][j];

}

}

int circuit[40];

int circuitIndex = 0;

Find\_Euler\_Circuit(0, tempMatrix, circuit, circuitIndex);

cout << "Euler Circuit found: { ";

for (int i = 0; i < circuitIndex; i++)

{

cout << circuit[i] << " ";

}

cout << "}\n";

}

//checking for the Hamiltoninan Path if exists or not

void Check\_Hamiltonian\_Path()

{

for (int i = 0; i < vertices; ++i) {

visited[i] = false;

}

path[0] = 0;

visited[0] = true;

if (Check\_Hamilton(0, 1)) {

cout << "Hamiltonian Path found: { ";

for (int i = 0; i < vertices; ++i) cout << path[i] << " ";

cout << "}\n";

}

else {

cout << "No Hamiltonian Path exists.\n";

}

}

//checkig for the Euler Path exist or not

void Check\_Euler\_Path()

{

if (!Check\_EulerPath()) {

cout << "No Euler Path exists.\n";

return;

}

cout << "An Euler Path exists.\n";

}

};

int main()

{

system("color 60");

//taking vertices and edges from user

int vertices, edges, choice, u, v;

cout << "Enter the number of vertices: ";

cin >> vertices;

cout << "Enter the number of edges: ";

cin >> edges;

Graph graph(vertices);

cout << "Enter edges (vertex\_1 , vertex\_2):\n";

for (int i = 0; i < edges; i++)

{

cout << "Enter Edge " << i + 1 << " : \n";

cin >> u >> v;

graph.addEdge(u, v);

}

do

{

//showing the display menu to user

system("cls");

cout << setfill(' ') << setw(50) << "\n" << setfill(' ');

cout << "Press 0 to Exit\n";

cout << "Press 1 to Check Hamiltonian Circuit\n";

cout << "Press 2 to Check Hamiltonian Path\n";

cout << "Press 3 to Check Euler Circuit\n";

cout << "Press 4 to Check Euler Path\n";

cout << endl << setfill(' ') << setw(50) << "\n" << setfill(' ');

cout << "Select the option...! ";

cin >> choice;

//it will run the acc to user enterd choice

switch (choice)

{

case 1:

{

graph.Check\_Hamilton\_Circuit();

system("pause");

break;

}

case 2:

{

graph.Check\_Hamiltonian\_Path();

system("pause");

break;

}

case 3:

{

graph.Check\_euler\_Cicuit();

system("pause");

break;

}

case 4:

{

graph.Check\_Euler\_Path();

system("pause");

break;

}

case 5:

{

cout << "Exiting the program...!\n";

break;

}

default:

cout << "Wrong input. Please try again.\n";

}

} while (choice != 5);

system("pasue");

return 0;

}

A yellow screen with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Question# 03:**

#include<iostream>

using namespace std;

#include<Windows.h>

//finding factorial of number of user enterd

int Factorial(int n) {

if (n == 0 || n == 1) {

return 1;

}

else{

return n \* Factorial(n - 1);

}

}

//finding the fibonacci sequence upto the number of user desired

int Fibonacci(int n) {

if (n == 0 || n == 1) {

return n;

}

else {

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

}

// finding the Sum\_of\_Natural\_Numbers upto the user enterd number n

int Sum\_of\_Natural\_Numbers(int n) {

if (n == 0) {

return n;

}

else {

return n + Sum\_of\_Natural\_Numbers(n - 1);

}

}

//finding the exponent where x is number and n is the power

int power(int x, int n)

{

if (n == 0)

return 1;

if (x == 0)

return 0;

else {

return x \* power(x, n - 1);

}

}

double Sum\_of\_Arithmetic\_Series(int a, int d, int n) {

if (n == 0)

return 0;

//first to find out the term which we can say is current

int t1 = a + (n - 1) \* d;

return t1 + Sum\_of\_Arithmetic\_Series(a, d, n - 1);

}

int main()

{

system("color 30");

//shoeing the menu to user

char choice2;

do

{

cout << " ================================================\n";

cout << " == Welcome to the Recursive Function Menu : == \n";

cout << " == 1. Factorial of a number == \n";

cout << " == 2. Fibonacci Sequence(nth term) == \n";

cout << " == 3. Sum of First n Natural Numbers == \n";

cout << " == 4. Sum of an Arithmetic Series == \n";

cout << " == 5. Exponentiation(x ^ n) == \n";

cout << " == 6. Exit == \n";

cout << " ================================================\n";

int choice;

cout << "Please select an option: ";

cin >> choice;

switch (choice)

{

//factorial of a number

case 1: {

int n1;

cout << "Enter the number of which factorial to want to find: ";

cin >> n1;

int fact = Factorial(n1);

cout << "Factorial of a number = " << fact << endl;

cout << "1. Base Case: 0! = 1 (true).\n";

cout << "2.Inductive Step: Assume factorials are correct for n-1. Prove for n: n! = n \* (n-1)!\n";

break;

}

//fionacci sequence

case 2: {

int n2;

cout << "Enter the number upto which u want to find the fibonacci sequence: ";

cin >> n2;

int fib = Fibonacci(n2);

cout << "Factorial upto your enterd number = " << fib << endl;

cout << "1. Base Case: F(0) = 0, F(1) = 1 (true).\n";

cout << "2. Inductive Step: Assume F(k-1) and F(k-2) are correct. Prove for n: F(n) = F(n-1) + F(n-2).\n";

break;

}

//sum of natural numbers

case 3: {

int n3;

cout << "Enter the number upto which u want to find the sum of natural numbers: ";

cin >> n3;

int natural = Sum\_of\_Natural\_Numbers(n3);

cout << "The sum of Natural Numbers upto " << n3 << " = " << natural << endl;

cout << "1. Base Case: S(0) = 0 (true).\n";

cout << "2. Inductive Step: Assume S(k) = k \* (k + 1) / 2 for n = k. Prove for n = k+1: S(k+1) = S(k) + (k+1).\n";

break;

}

// Sum of an Arithmetic Series

case 4: {

int a, d, n;

cout << "Enter the first term a: ";

cin >> a;

cout << "Enter the common difference d: ";

cin >> d;

cout << "Enter the number of terms n: ";

cin >> n;

double sum = Sum\_of\_Arithmetic\_Series(a, d, n);

cout << "The sum of arithmetic sequence= "<<sum << endl;

cout << "1. Base Case: A(1) = a (true).\n";

cout << "2. Inductive Step: Assume the sum is correct for n = k. Prove for n = k+1: A(k+1) = A(k) + (a + k\*d).\n";

break;

}

//finding the exponent where x is number and n is the power

case 5: {

int num, powe;

cout << "Enter the number: ";

cin >> num;

cout << "Enter the power: ";

cin >> powe;

double expo = power(num, powe);

cout << "The Exponentiation (x^n)= " << expo << endl;

cout << "1. Base Case: x^0 = 1 (true).\n";

cout << "2. Inductive Step: Assume x^k = x \* x^(k-1). Prove for k+1: x^(k+1) = x \* x^k.\n";

break;

}

//exiting

case 6: {

cout << "Thank you for using the program!\n";

return 0;

}

default: {

cout << "Invalid chioce. Please try again.\n";

}

}

cout << "Would you like to perform another operation? (Y/N): ";

cin >> choice2;

} while (choice2 == 'Y' || choice2 == 'y'); //asking user if u want to continue or not

cout << endl;

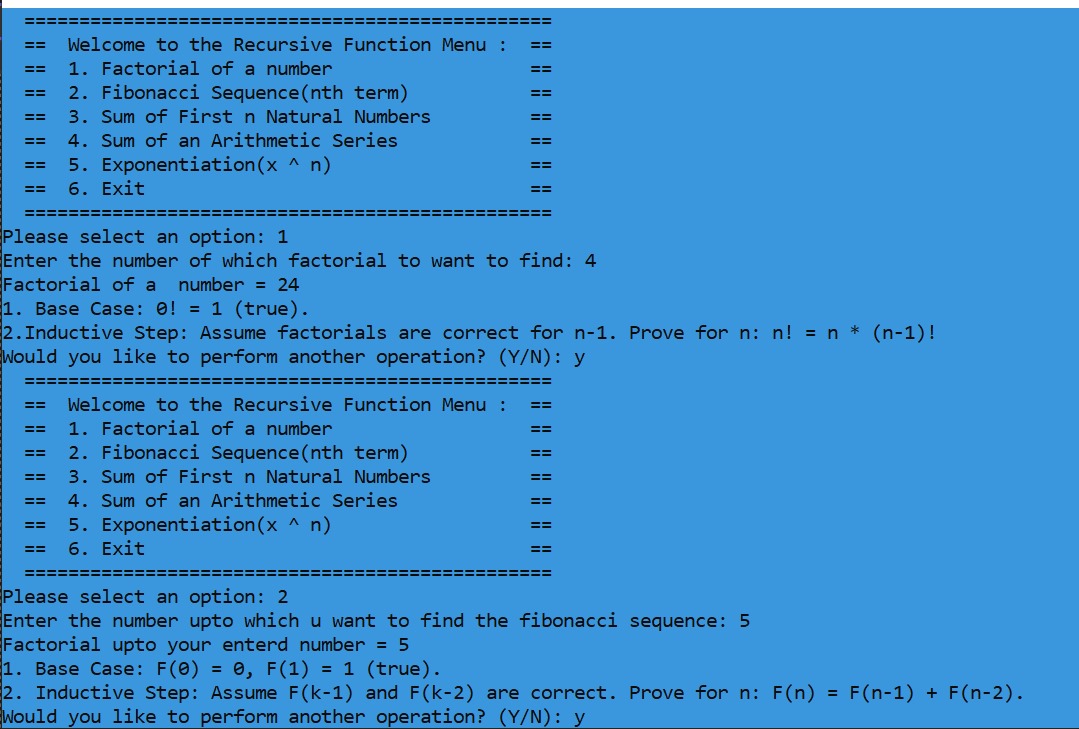
//finally showing thnaks staement

cout << "Thank you for using the program!\n";

system("pause");

return 0;

}



A blue screen with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A blue screen with black text

Description automatically generated