**Muhammad Abdullah**

**19F-0916 | SE(3A)**

Data Structures

LAb 5 stacks Formation

**Question # 1:**

**PROGRAM:**

#include<iostream>

using namespace std;

int Size;

int top = -1;

int \*Array = new int[Size];

class Two\_Stacks

{

public:

Two\_Stacks()

{

Stack1 = NULL, Stack2 = NULL;

Stack1 = Array, Stack2 = Array;

}

int \*Stack1, \*Stack2;

void push1(int Value) //push function according to choice for 1st stack

{

if (top != Size - 1)

{

top++;

Stack1[top] = Value;

}

else

cout << endl << "1st Stack is FULL !!" << endl;

}

void push2(int Value) //push function according to choice for 2nd stack

{

if (top != Size - 1)

{

top++;

Stack2[top] = Value;

}

else

cout << endl << "2nd Stack is FULL !!" << endl;

}

int pop1() //pop function according to choice for 1st stack

{

if (top != -1)

{

int temp = 0;

temp = Stack1[top];

top--;

return temp;

}

else

cout << endl << "1st Stack is Empty !!";

}

int pop2() //pop function according to choice for 2nd stack

{

if (top != -1)

{

int temp;

temp = Stack2[top];

top--;

return temp;

}

else

cout << endl << "1st Stack is Empty !!";

}

void Display() // This is just to confirm output

{

int temp = top;

while (top != -1)

{

cout << "\nStack 1 = " << Stack1[top] << " \nStack 2 = " << Stack2[top];

top--;

}

}

};

int main()

{

int Opt = 0, Choice = 0, Value = 0;

cout << "Enter the Size for the Stack to be formed : ";

cin >> Size;

Two\_Stacks Object;

while (Choice != -1)

{

system("cls");

cout << "Press 1 to Push Value in 1st Stack !!" << endl;

cout << "Press 2 to Push Value in 2nd Stack !!" << endl;

cout << "Press 3 to Pop Value from 1st Stack !!" << endl;

cout << "Press 4 to Pop Value from 2nd Stack !!" << endl;

cout << "Press 5 to Check Array (for checking purpose) !!" << endl;

cout << "Press 0 to Exit from the Program !!" << endl;

cout << endl << "Enter Choice : ";

cin >> Opt;

switch (Opt)

{

case 1:

{

cout << endl << "Enter Value to Enter in 1st Stack : ";

cin >> Value;

Object.push1(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Value to Enter in 2nd Stack : ";

cin >> Value;

Object.push2(Value);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << Object.pop1() << " is Poped from 1st Stack !" << endl;

system("pause");

break;

}

case 4:

{

cout << Object.pop2() << " is Poped from 2nd Stack !" << endl;

system("pause");

break;

}

case 5:

{

Object.Display();

system("pause");

break;

}

case 0:

{

Choice = -1;

break;

}

default:

cout << endl << "Invalid Entry " << endl;

system("pause");

}

}

cout << endl << endl;

system("pause");

}

A screenshot of a computer screen

Description automatically generated

**Question # 2:**

**PROGRAM:**

#include<iostream>

using namespace std;

struct Arrayy //Reversing Stack

{

int Size;

int top = -1;

int \*Array = NULL;

};

class Stack

{

public:

Stack(int size)

{

ARRAY.Size = size;

ARRAY.Array = new int[size];

}

Arrayy ARRAY;

void push(int Value) //push function

{

if (ARRAY.top != ARRAY.Size - 1)

{

ARRAY.top++;

ARRAY.Array[ARRAY.top] = Value;

}

else

cout << endl << "Stack is FULL !!" << endl;

}

int pop() //pop function

{

if (ARRAY.top != -1)

{

int temp = 0;

temp = ARRAY.Array[ARRAY.top];

ARRAY.top--;

return temp;

}

else

cout << endl << "Stack is Empty !!";

}

void Reverse() //Reverse Function by making another stack

{

Arrayy temp;

temp.Size = ARRAY.Size;

temp.Array = new int[temp.Size];

int tempARRAY = ARRAY.top;

if (tempARRAY != -1)

{

while (tempARRAY != -1)

{

temp.Array[++temp.top] = pop();

tempARRAY--;

}

cout << endl << "Reversed Stack = ";

while (temp.top != -1)

{

cout << temp.Array[temp.top] << " ";

temp.top--;

}

}

else

cout << endl << "Stack is Empty !!" << endl;

}

void Display() // This is just to confirm output

{

int temp = ARRAY.top;

if (temp != -1)

{

cout << "Values = ";

while (temp != -1)

{

cout << ARRAY.Array[temp] << " ";

temp--;

}

}

else

cout << endl << "Stack is Empty !!" << endl;

}

};

int main()

{

int Opt = 0, Choice = 0, Value = 0, Size = 0;

cout << "Enter the Size for the Stack to be formed : ";

cin >> Size;

Stack Object(Size);

while (Choice != -1)

{

system("cls");

cout << "Press 1 to Push Value in Stack !!" << endl;

cout << "Press 2 to POP Value from Stack !!" << endl;

cout << "Press 3 to Reverse Values in Stack !!" << endl;

cout << "Press 4 to Check Array (for checking purpose) !!" << endl;

cout << "Press 0 to Exit from the Program !!" << endl;

cout << endl << "Enter Choice : ";

cin >> Opt;

switch (Opt)

{

case 1:

{

cout << endl << "Enter Value in Stack : ";

cin >> Value;

Object.push(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl << Object.pop() << " is Poped Value !" << endl;

system("pause");

break;

}

case 3:

{

cout << endl << "Original ";

Object.Display();

cout << endl;

Object.Reverse();

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl;

Object.Display();

system("pause");

break;

}

case 0:

{

Choice = -1;

break;

}

default:

cout << endl << "Invalid Entry " << endl;

system("pause");

}

}

cout << endl << endl;

system("pause");

}

A screenshot of a computer screen

Description automatically generated

**Question # 3:**

**PROGRAM:**

#include<iostream>

using namespace std;

struct Node //Link List based Implementation of Stack

{

int value;

Node \*Next\_Node = NULL;

};

class Stack

{

public:

Stack()

{

Head = NULL;

}

Node \*Head;

void push(int Value) //push function

{

if (Head == NULL)

{

Node \*temp = new Node;

temp->value = Value;

temp->Next\_Node = NULL;

Head = temp;

}

else

{

Node \*temp = new Node;

temp->value = Value;

temp->Next\_Node = Head;

Head = temp;

}

}

int pop() //pop function

{

if (Head != NULL)

{

Node \*temp = new Node;

int tem;

temp = Head;

Head = Head->Next\_Node;

//cout << endl << temp->value << " is Poped !" << endl;

tem = temp->value;

free(temp);

return tem;

}

else

cout << endl << "Stack is Empty !!";

}

void Reverse() //Reverse Function by making another stack

{

Node \*temp = NULL;

Node \*temporary = Head;

Node \*TT = NULL;

if (temporary != NULL)

{

while (temporary != NULL)

{

Node \*TT = new Node;

TT->value = pop();

if (temp == NULL)

{

TT->Next\_Node = NULL;

temp = TT;

}

else

{

TT->Next\_Node = temp;

temp = TT;

}

if (temporary == NULL)

break;

temporary = Head;

}

cout << endl << "Reversed Stack = ";

while (temp != NULL)

{

cout << temp->value << " ";

temp = temp->Next\_Node;

}

}

else

cout << endl << "Stack is Empty !!" << endl;

}

void Display() // This is just to confirm output

{

Node \*temp = Head;

if (temp != NULL)

{

cout << "Values = ";

while (temp != NULL)

{

cout << temp->value << " ";

temp = temp->Next\_Node;

}

}

else

cout << endl << "Stack is Empty !!" << endl;

}

};

int main()

{

int Opt = 0, Choice = 0, Value = 0;

Stack Object;

while (Choice != -1)

{

system("cls");

cout << "Press 1 to Push Value in Stack !!" << endl;

cout << "Press 2 to POP Value from Stack !!" << endl;

cout << "Press 3 to Reverse Values in Stack !!" << endl;

cout << "Press 4 to Check Array (for checking purpose) !!" << endl;

cout << "Press 0 to Exit from the Program !!" << endl;

cout << endl << "Enter Choice : ";

cin >> Opt;

switch (Opt)

{

case 1:

{

cout << endl << "Enter Value in Stack : ";

cin >> Value;

Object.push(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

Object.pop();

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl << "Original ";

Object.Display();

cout << endl;

Object.Reverse();

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl;

Object.Display();

system("pause");

break;

}

case 0:

{

Choice = -1;

break;

}

default:

cout << endl << "Invalid Entry " << endl;

system("pause");

}

}

cout << endl << endl;

system("pause");

}

A screenshot of a computer screen

Description automatically generated

**Question # 4:**

**NOTE: (**THIS CODE CONTAINS ALL REQUIRED FUNCTIONALITIES LIKE PUSH, POP, ISEMPTY, ISOPERATOR, OPERATIONS, PRECEDENCE etc. FUNCTIONS**)**

**PROGRAM:**

#include <iostream>

#include<string> // By M.Abdullah

using namespace std; //Computer Stack Implementation

int \*Arrayy = NULL;

char \*Array = NULL; // Operator Stack for POSTFIX

int top = -1;

void push(char C); // Data for In to Post

int pop();

void Vpush(int); // For Evaluation

int Vpop();

int Precedence(char Operator); // Required Function

char IsOperator(char C);

char StackTop();

bool isEmpty();

int Operations(int X, int Y, char Operator);

string In\_To\_Post(string Expression); //Conversion

int Evaluation(string Expressions); //Evaluation

int main() // Main Code

{

string expression,evaluation;

expression = "(2+3)\*(3+5)";

cout << endl << "InFix Expression is : " << expression << endl; // expression

cout << endl << "PostFix Expression is : " << In\_To\_Post(expression) << endl; // conversion

evaluation = In\_To\_Post(expression);

cout << endl << "PostFix Evaluation is : " << Evaluation(evaluation); //evaluation

cout << endl << endl;

system("pause");

}

void push(char C) // Push Function For Conversion

{

top++;

Array[top] = C;

}

int pop() // Pop Function For Conversion

{

char x;

x = Array[top];

top--;

return x;

}

void Vpush(int C) // Push Function For Evaluation

{

top++;

Array[top] = C;

}

int Vpop() // Pop Function For Evaluation

{

int x;

x = Array[top];

top--;

return x;

}

int Precedence(char Operator) //Precedence Checking Function

{

if (Operator == '+' || Operator == '-')

return 1;

else if (Operator == '\*' || Operator == '/')

return 2;

else if (Operator == '^')

return 3;

else

return 0;

}

char IsOperator(char c) //Tells Whether Recieved Character is Operator or Not

{

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')

return true;

else

return false;

}

char StackTop() // Returns Top value or Character from Stack

{

if (top != -1)

{

return Array[top];

}

else

cout << endl << "Stack is Empty !!" << endl;

}

bool isEmpty() // Checks Whether Stack is Empty or Not

{

if (top == -1)

{

cout << endl << "Stack is Empty !!" << endl;

return true;

}

else

{

cout << endl << "Stack is Not Empty !!" << endl;

return false;

}

}

int Operations(int X, int Y, char Operator) //Used to perform operations to evaluate PostFix

{

switch (Operator)

{

case '+': return X + Y;

case '-': return X - Y;

case '\*': return X \* Y;

case '/': return X / Y;

case '^': return X ^ Y;

}

}

string In\_To\_Post(string Expression) // Conver InFix Expression to PostFix Expression

{

Array = new char[Expression.length()];

string Final;

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

{

if ((Expression[i] >= 'a' && Expression[i] <= 'z') || (Expression[i] >= 'A' && Expression[i] <= 'Z') ||

(Expression[i] >= '1' && Expression[i] <= '9'))

{

Final = Final + Expression[i];

}

else if (Expression[i] == ' ')

{

continue;

}

else if (Expression[i] == '(')

{

push(Expression[i]);

}

else if (Expression[i] == ')')

{

while (top != -1 && Array[top] != '(')

{

char opt = pop();

Final = Final + opt;

}

pop();

}

else

{

while (top != -1 && Precedence(Array[top]) >= Precedence(Expression[i]))

{

char opt = pop();

Final = Final + opt;

}

push(Expression[i]);

}

}

while (top != -1)

{

char opt = pop();

Final = Final + opt;

}

return Final;

}

int Evaluation(string Expressions)

{

top = -1;

Arrayy = new int[Expressions.length()]; //Stack Creation

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

{

if (Expressions[i] == ' ')

continue; //Ignoring Spaces in Expression

else if (isdigit(Expressions[i]))

{

Vpush(Expressions[i] - '0'); // ASCII of 0 Because Expression is in character so

// it needs to be change into int for solving

}

else if (Expressions[i] == '+' || Expressions[i] == '-' || Expressions[i] == '\*' || Expressions[i] == '/' || Expressions[i] == '^')

{

int X = Vpop(); //When Operator found, instanly perform action

int Y = Vpop(); // on it then push value into stack of operands

char Z = Expressions[i];

Vpush(Operations(Y, X, Z));

}

else if (Expressions[i] == '(' || Expressions[i] == ')')

{

cout << "ERROR, BRACKETS ARE NOT ENCOUNTERED IN POSTFIX EVALUATION !!";

break;

}

}

return Vpop();

}

A screenshot of a computer screen

Description automatically generated

**Question # 5 (A): INFIX TO POSTFIX**

**PROGRAM:**

#include <iostream>

#include<string> // By M.Abdullah

using namespace std; // Infix to Post fix

char \*OArray = NULL; // Operator Stack

int Otop = -1;

void Opush(char C);

int Opop();

int Precedence(char Operator); // Required Function

string In\_To\_Post(string Expression);

int main()

{

string Expression;

Expression = "a + b\*(c^d - e) ^ (f + g\*h) - I";

cout << "Given Expression = " << Expression << endl;

cout << endl << "Post-Fix Expression is : " << In\_To\_Post(Expression);

cout << endl << endl;

system("pause");

}

void Opush(char C)

{

Otop++;

OArray[Otop] = C;

}

int Opop()

{

char x;

x = OArray[Otop];

Otop--;

return x;

}

int Precedence(char Operator)

{

if (Operator == '+' || Operator == '-')

return 1;

else if (Operator == '\*' || Operator == '/')

return 2;

else if (Operator == '^')

return 3;

else

return 0;

}

string In\_To\_Post(string Expression)

{

OArray = new char[Expression.length()];

string Final;

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

{

if ((Expression[i] >= 'a' && Expression[i] <= 'z') || (Expression[i] >= 'A' && Expression[i] <= 'Z') ||

(Expression[i] >= '1' && Expression[i] <= '9'))

{

Final = Final + Expression[i];

}

else if (Expression[i] == ' ')

{

continue;

}

else if (Expression[i] == '(')

{

Opush(Expression[i]);

}

else if (Expression[i] == ')')

{

while (Otop != -1 && OArray[Otop] != '(')

{

char opt = Opop();

Final = Final + opt;

}

Opop();

}

else

{

while (Otop != -1 && Precedence(OArray[Otop]) >= Precedence(Expression[i]))

{

char opt = Opop();

Final = Final + opt;

}

Opush(Expression[i]);

}

}

while (Otop != -1)

{

char opt = Opop();

Final = Final + opt;

}

return Final;

}

A screenshot of a computer screen

Description automatically generated

**Question # 5 (B): POSTFIX TO INFIX**

**PROGRAM:**

#include <iostream>

#include<string> // By M.Abdullah

using namespace std; // Postfix to Infix

// this will only work if infix have brackets otherswise i think it will flactuate

string \*OArray = NULL; // Operator Stack

int Otop = -1;

void Opush(string C);

string Opop();

int Precedence(char Operator); // Required Function

string Post\_To\_In(string Expression);

int main()

{

string Expression;

Expression = "abc++";

cout << "Given Expression = " << Expression << endl;

cout << endl << "InFix Expression is : " << Post\_To\_In(Expression);

cout << endl << endl;

system("pause");

}

void Opush(string C)

{

Otop++;

OArray[Otop] = C;

}

string Opop()

{

string x;

x = OArray[Otop];

Otop--;

return x;

}

int Precedence(char Operator)

{

if (Operator == '+' || Operator == '-')

return 1;

else if (Operator == '\*' || Operator == '/')

return 2;

else if (Operator == '^')

return 3;

else

return 0;

}

string Post\_To\_In(string Expression)

{

OArray = new string[Expression.length()];

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

{

if ((Expression[i] >= 'a' && Expression[i] <= 'z') || (Expression[i] >= 'A' && Expression[i] <= 'Z') ||

(Expression[i] >= '1' && Expression[i] <= '9'))

{

string A(1, Expression[i]);

Opush(A);

}

else

{

string A = OArray[Otop];

Opop();

string B = OArray[Otop];

Opop();

Opush("(" + B + Expression[i] + A + ")"); // first in last out concept

}

}

return OArray[Otop];

}

**A screenshot of a computer screen

Description automatically generated**

**Question # 5 (C): PREFIX TO POSTFIX**

**PROGRAM:**

#include <iostream>

#include<string> // By M.Abdullah

using namespace std; // I am using to fucntions one for Pre to In and other for In to Post

string \*OArray = NULL; // Operator Stack for INFIX

int Otop = -1; // Data for Pre to In

void Opush(string C);

string Opop(); // Required Function

string Pre\_To\_In(string Expression);

char \*Array = NULL; // Operator Stack for POSTFIX

int top = -1;

void push(char C); // Data for In to Post

int pop();

int Precedence(char Operator); // Required Function

string In\_To\_Post(string Expression);

int main() // Main Code

{

string Expression, Infix;

Expression = "\*+AB-CD";

cout << "Given Expression = " << Expression << endl;

cout << endl << "InFix Expression is : " << Pre\_To\_In(Expression) << endl; // pre to in

Infix= In\_To\_Post(Expression);

cout << endl << "PostFix Expression is : " << In\_To\_Post(Infix); // in to post

cout << endl << endl;

system("pause");

}

void Opush(string C)

{

Otop++;

OArray[Otop] = C;

}

string Opop() // pop push for pre to in

{

string x;

x = OArray[Otop];

Otop--;

return x;

}

void push(char C)

{

top++;

Array[top] = C;

} // pop push for in to post

int pop()

{

char x;

x = Array[top];

top--;

return x;

}

int Precedence(char Operator)

{

if (Operator == '+' || Operator == '-')

return 1;

else if (Operator == '\*' || Operator == '/')

return 2;

else if (Operator == '^')

return 3;

else

return 0;

}

string Pre\_To\_In(string Expression) //Pre to In conversion if Infix contains brackets

{

OArray = new string[Expression.length()];

for (int i = Expression.length() - 1; i >= 0; i--)

{

if (Expression[i]=='+' || Expression[i] == '-' || Expression[i] == '\*' || Expression[i] == '/' || Expression[i] == '^')

{

string A = OArray[Otop];

Opop();

string B = OArray[Otop];

Opop();

string C = "(" + A + Expression[i] + B + ")";

Opush(C);

}

else

{

Opush(string(1, Expression[i]));

}

}

return OArray[Otop];

}

string In\_To\_Post(string Expression)

{

Array = new char[Expression.length()];

string Final;

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

{

if ((Expression[i] >= 'a' && Expression[i] <= 'z') || (Expression[i] >= 'A' && Expression[i] <= 'Z') ||

(Expression[i] >= '1' && Expression[i] <= '9'))

{

Final = Final + Expression[i];

}

else if (Expression[i] == ' ')

{

continue;

}

else if (Expression[i] == '(')

{

push(Expression[i]);

}

else if (Expression[i] == ')')

{

while (top != -1 && Array[top] != '(')

{

char opt = pop();

Final = Final + opt;

}

pop();

}

else

{

while (top != -1 && Precedence(Array[top]) >= Precedence(Expression[i]))

{

char opt = pop();

Final = Final + opt;

}

push(Expression[i]);

}

}

while (top != -1)

{

char opt = pop();

Final = Final + opt;

}

return Final;

}

A screenshot of a computer screen

Description automatically generated