**Lab 5**

**Task 1**

**Stack as Link List based implementation**

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node\* next;

};

class Implement {

public:

Node\* top = NULL;

bool IsEmpty()

{

return(top == NULL);

}

void push(int val) {

struct Node\* newnode = new Node();

//(struct Node\*)malloc(sizeof(struct Node));

newnode->data = val;

newnode->next = top;

top = newnode;

}

void pop() {

if (top == NULL)

cout << "Stack Underflow" << endl;

else {

cout << "The popped element is " << top->data << endl;

top = top->next;

}

}

void display() {

struct Node\* ptr;

if (top == NULL)

cout << "stack is empty";

else {

ptr = top;

cout << "Stack elements are: ";

while (ptr != NULL) {

cout << ptr->data << " ";

ptr = ptr->next;

}

}

}

};

int main()

{

Implement\* I = new Implement();

I->push(5);

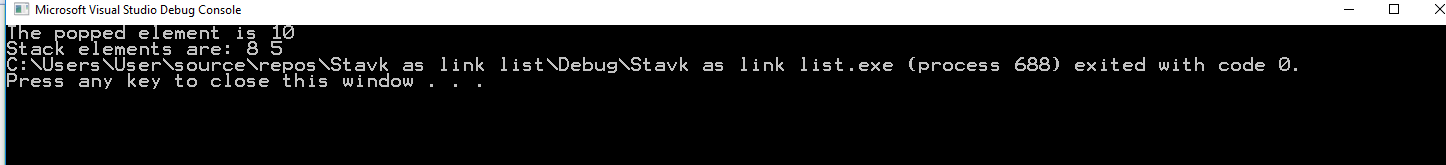
I->push(8);

I->push(10);

I->pop();

I->display();

I->IsEmpty();

 }

**Stack as Array Based Implementation**

#include <iostream>

#include <string>

using namespace std;

const int array\_size = 10;

struct stack

{

public:

int top = -1;

char stack[array\_size];

};

class Implement

{

public:

string str;

bool IsEmpty(struct stack\* pointer)

{

if (pointer->top == -1)

{

cout << "stack underflow";

return true;

}

else { return false; }

}

bool IsFull(struct stack\* pointer)

{

if (pointer->top == array\_size)

{

cout << "stack overflow";

return true;

}

else

{

return false;

}

}

void Push(struct stack\* pointer, char value)

{

pointer->top++;

pointer->stack[pointer->top] = value;

}

char Top(struct stack\* pointer)

{

if (IsEmpty(pointer))

{

return NULL;

}

else

return (pointer->stack[pointer->top]);

}

char pop(struct stack\* pointer)

{

return (pointer->stack[pointer->top--]);

}

int main()

{

stack\* p = new stack();

Implement\* I = new Implement();

/\*string expression;

cin >> expression;\*/

//I->infixToPostfix(p,expression);\*/

//I->BalanceBrackets(p, expression);

//I->calculatePostfix(p, expression);

I->Push(p,'p');

cout << I->Top(p);

I->Push(p, 'a');

cout << I->Top(p); I->Push(p, 'k');

cout << I->Top(p); I->Push(p, 'i');

cout << I->Top(p); I->Push(p, 's');

cout << I->Top(p); I->Push(p, 't');

cout << I->Top(p); I->Push(p, 'a'); cout << I->Top(p);

I->Push(p, 'n');

cout << I->Top(p);

cout << I->IsEmpty(p);

cout << I->IsFull(p);

**Task 2**

void BalanceBrackets(struct stack\* p,string exp)

{

char ch;

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

{

if (exp[i] == '(' || exp[i] == '{' || exp[i]== '[')

{ ch=exp[i];

Push(p,ch);}

switch (exp[i])

{ ch = Top(p);

case ')':

{

if (ch == '{' || ch == '[')

{

cout << "do not matching brackets";

break;

}

if (ch=='(')

{

pop(p);

continue;

}

}

case '}':

{

if (ch == '[' || ch == '(')

{

cout << "do not matching brackets";

break;

}

if (ch == '{')

{

pop(p);

continue;

}

}

case']':

{

pop(p);

if (ch == '{' || ch == '(')

{

cout << "do not matching brackets";

break;

}

if (ch == '[')

{

continue;

}

}

}

}

 }

**Task4**

**Calculate Prefix**

void calculatePostfix(struct stack\* pointer, string exp)

{

char ch;

int op1;

int op2;

string temp;

int result;

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

{

ch = exp[i];

cout << ch << endl;

if (ch >= '0' && ch <= '9')

{ while (ch != ' ')

{

temp = ch + temp;

}

stoi(temp);

cout << stoi(temp);

Push(pointer, stoi(temp));

}

else

{

if (!IsEmpty(pointer))

{

if (ch == '+')

{

op1 = pop(pointer);

cout << op1 << endl;

op2 = pop(pointer);

cout << op2 << endl;

result =(op1) + (op2);

cout << result << endl;

Push(pointer, result);

}

if (ch == '-')

{

op1 = pop(pointer);

cout << op1 << endl;

op2 = pop(pointer);

cout << op2 << endl;

result = (op1)-(op2);

cout << result << endl;

Push(pointer, result);

}

if (ch == '\*')

{

op1 = pop(pointer);

cout << op1 << endl;

op2 = pop(pointer);

cout << op2 << endl;

result = (op1)\*(op2);

cout << result << endl;

Push(pointer, result);

}

if (ch == '/')

{

op1 = pop(pointer);

cout << op1 << endl;

op2 = pop(pointer);

cout << op2 << endl;

result = (op1)/(op2);

cout << result << endl;

Push(pointer, result);

}

if (ch == '^')

{

op1 = pop(pointer);

cout << op1 << endl;

op2 = pop(pointer);

cout << op2 << endl;

result = (op1)^(op2);

cout << result << endl;

Push(pointer, result);

}

}

}

}