# Department of Computing

**Name: Mahum Samar**

**CMS ID : 290647**

**CS250: Data Structure and Algorithms**

**Class: BSCS 9B**

# lab 05: stack

# Task 01

## Code:

# include <iostream>

using namespace std;

#define SIZESTACK 100

class Stack

{

public:

int top; //to keep track of the top of the stack

int array[SIZESTACK]; //array for implementing array based implementation.

Stack()

{

top = -1; //initializing the stack

}

bool isEmpty()

{

//method to check if the stack is empty

if(top == -1)

{

//if stack empty

return true;

}

else

{

//if stack is not empty

return false;

}

}

bool isFull()

{

//method to check if the stack is full

if(top == (SIZESTACK-1))

{

//stack full

return true;

}

else

{

return false;

}

}

void Push(int x)

{

//method to push the values in the stack

if(isFull())

{

cout << "push is full Stack Overflow.";

}

else

{

top++;

array[top] = x;

}

//PrintStack();

}

int Pop()

{

//method to pop the values out of the stack

if(isEmpty())

{

cout << "pop is empty. Stack Underflow.";

return -1;

}

else

{

int x = array[top];

top--;

return x;

}

}

int Top()

{

//method to get the top value stored in the stack

if(isEmpty())

{

cout << "top is empty. Stack Underflow.";

return -1;

}

else

{

return (array[top]);

}

}

void PrintStack()

{

//method to print the stack on the screen

while(top != -1)

{

cout << " " << Top();

Pop();

}

cout << endl;

}

};

int main()

{

Stack \*stack = new Stack();

cout << "Checking if the stack is empty." << endl;

if(stack -> isEmpty())

{

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

}

else

{

cout << "Stack is not empty." << endl;

}

//stack -> isFull();

cout << endl;

cout << "Pushing 6,5,4,3,2 in the stack..." << endl;

stack -> Push(6);

stack -> Push(5);

stack -> Push(4);

stack -> Push(3);

stack -> Push(2);

cout << endl;

cout << "Getting the top value of the stack:" << endl;

cout << stack -> Top() << endl;

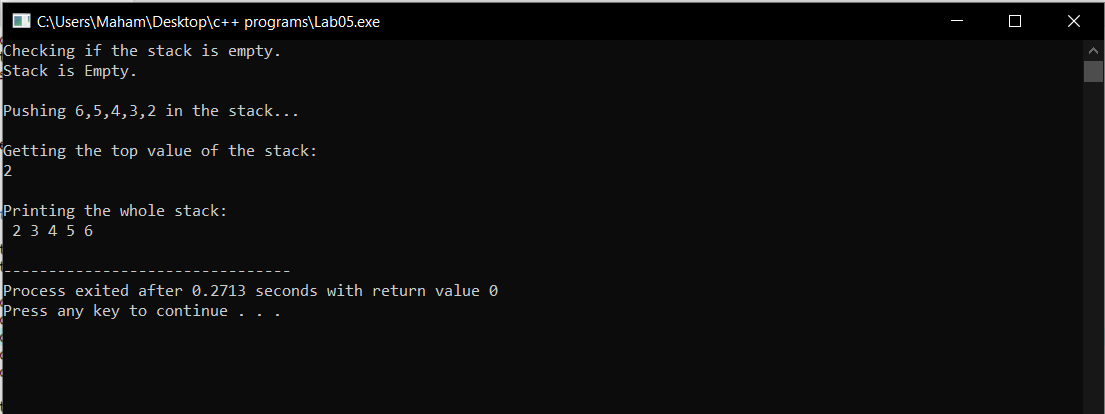
cout << endl;

cout << "Printing the whole stack:" << endl;

stack -> PrintStack();

}

## Code:



# Task 02

## Code:

# include <iostream>

using namespace std;

#define SIZESTACK 100

template <class StackType> class Stack

{

//generic stack class.

public:

int top; //to keep track of the top of the stack

StackType array[SIZESTACK]; //array for implementing array based implementation.

Stack()

{

top = -1; //initializing the stack

}

bool isEmpty()

{

//method to check if the stack is empty

if(top == -1)

{

//if stack empty

return true;

}

else

{

//if stack is not empty

return false;

}

}

bool isFull()

{

//method to check if the stack is full

if(top == (SIZESTACK-1))

{

//stack full

return true;

}

else

{

return false;

}

}

void Push(StackType x)

{

//method to push the values in the stack

if(isFull())

{

cout << "push is full Stack Overflow.";

}

else

{

top++;

array[top] = x;

}

//PrintStack();

}

StackType Pop()

{

//method to pop the values out of the stack

if(isEmpty())

{

cout << "pop is empty Stack Underflow.";

return -1;

}

else

{

StackType x = array[top];

top--;

return x;

}

}

StackType Top()

{

//method to get the top value stored in the stack

if(isEmpty())

{

cout << "top is empty Stack Underflow.";

return -1;

}

else

{

return (array[top]);

}

}

void PrintStack()

{

//method to print the stack on the screen

while(top != -1)

{

cout << " " << Top();

Pop();

}

cout << endl;

}

void CheckBalancedParenthesis()

{

int i = 0;

string s;

cout << "Enter the string containg the parenthesis : ";

cin >> s;

int flag = 1;

char space = ' ';

for(i =0; i < s.length() ; i++)

{

if(((s[i] >= 'a') && (s[i] <= 'z')) || ((s[i] >= 'A') && (s[i] <= 'Z') ) || (s[i] == '32') || (s[i] == '+') || (s[i] == '-') || (s[i] == '/') || (s[i] == '\*') || (s[i] == '.') || (s[i] == '^') || ((s[i] >= '0') && (s[i] <= '9')))

{

continue;

}

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

{

Push(s[i]);

continue;

}

//cout << Top() << endl;

if(!isEmpty())

{

if(s[i] == '}')

{

if(Top() == '{')

{

Pop();

continue;

}

else

{

flag = 0;

break;

}

}

if(s[i] == ']')

{

if(Top() == '[')

{

Pop();

continue;

}

else

{

flag = 0;

break;

}

}

if(s[i] == ')')

{

if(Top() == '(')

{

Pop();

continue;

}

else

{

flag = 0;

break;

}

}

}

else

{

flag = 0;

break;

}

}

if(isEmpty() && flag == 1)

{

cout << endl;

cout << "The string you entered has balanced paranthesis." << endl;

}

else

{

cout << endl;

cout << "The string you entered does not have balanced paranthesis. Error at character # " << i << " "<< Top() << " should be closed." << endl;

}

}

};

int main()

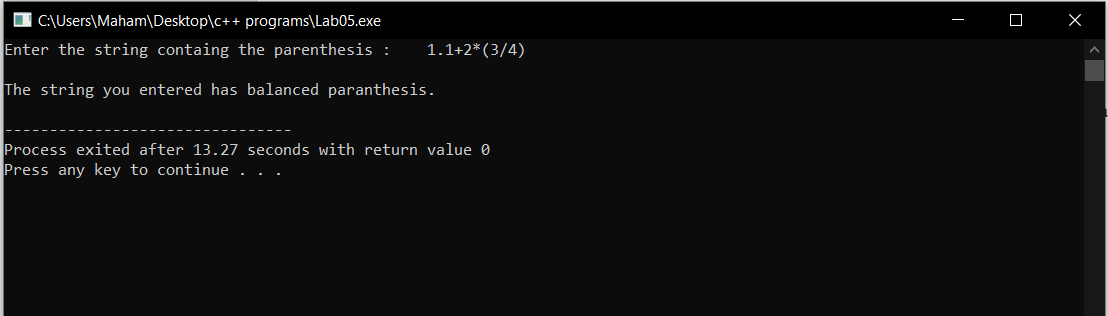
{

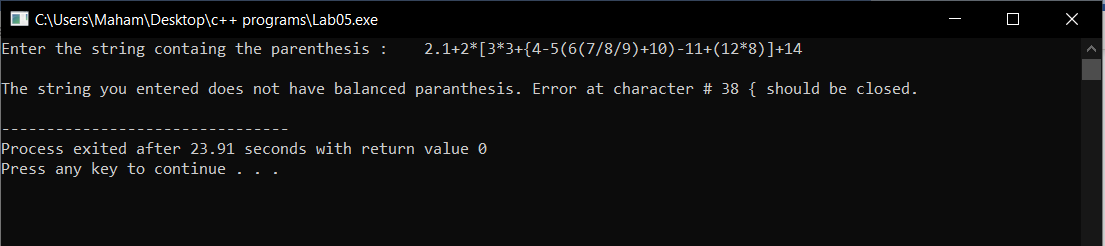
Stack <char> \*stackChar = new Stack<char>();

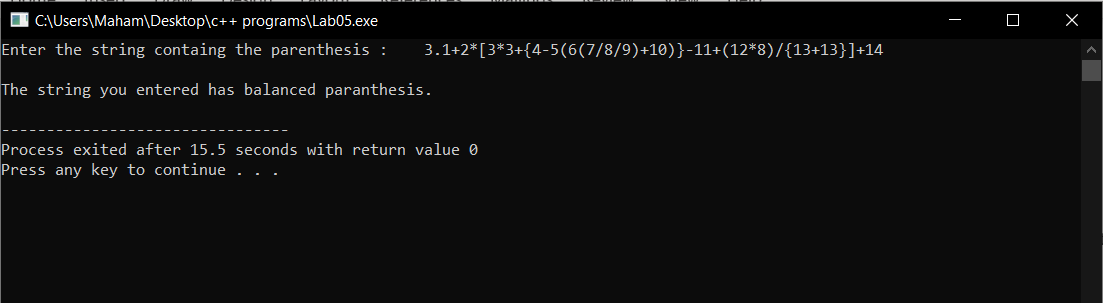
stackChar -> CheckBalancedParenthesis();

}

## output:







# Task 03

## Code:

# include <iostream>

#include<string.h>

#include<sstream>

#include<cmath>

using namespace std;

#define SIZESTACK 100

template <class StackType> class Stack

{

//generic stack class.

public:

int top; //to keep track of the top of the stack

StackType array[SIZESTACK]; //array for implementing array based implementation.

Stack()

{

top = -1; //initializing the stack

}

bool isEmpty()

{

//method to check if the stack is empty

if(top == -1)

{

//if stack empty

return true;

}

else

{

//if stack is not empty

return false;

}

}

bool isFull()

{

//method to check if the stack is full

if(top == (SIZESTACK-1))

{

//stack full

return true;

}

else

{

return false;

}

}

void Push(StackType x)

{

//method to push the values in the stack

if(isFull())

{

cout << "push is full Stack Overflow.";

}

else

{

top++;

array[top] = x;

}

//PrintStack();

}

StackType Pop()

{

//method to pop the values out of the stack

if(isEmpty())

{

cout << "pop is empty Stack Underflow.";

return -1;

}

else

{

StackType x = array[top];

array[top] = '\0';

top--;

return x;

}

}

StackType Top()

{

//method to get the top value stored in the stack

if(isEmpty())

{

cout << "top is empty Stack Underflow.";

return -1;

}

else

{

return (array[top]);

}

}

void PrintStack()

{

//method to print the stack on the screen

while(top != -1)

{

cout << " " << Top();

Pop();

}

cout << endl;

}

bool CheckBalancedParenthesis(string s)

{

int i = 0;

int flag = 1;

char space = ' ';

for(i =0; i < s.length() ; i++)

{

//checks if the character is letter special character or mathematical signs.

if(((s[i] >= 'a') && (s[i] <= 'z')) || ((s[i] >= 'A') && (s[i] <= 'Z') ) || (s[i] == '+') || (s[i] == '-') || (s[i] == '/') || (s[i] == '\*') || (s[i] == '.') || (s[i] == '^') || ((s[i] >= '0') && (s[i] <= '9')))

{

continue;

}

//if opening braces come

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

{

//push in the stack

Push(s[i]);

continue;

}

if(!isEmpty())

{

//if closing brace comes

if(s[i] == '}')

{

if(Top() == '{')

{

//removes the pair of braces.

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

if(s[i] == ']')

{

if(Top() == '[')

{

//removes the pair of braces

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

if(s[i] == ')')

{

if(Top() == '(')

{

//removes the pair of braces

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

}

else

{

flag = 0;

break;

}

}

if(isEmpty() && flag == 1)

{

return true;

}

else

{

cout << endl;

cout << "The string you entered does not have balanced paranthesis. Error at character # " << i << " "<< Top() << " should be closed." << endl;

return false;

}

}

int Precedence(char op)

{

//method to check the precedence of the operator.

if(op == '^')

{

return 3;

}

if((op == '\*') || (op == '/'))

{

return 2;

}

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

{

return 1;

}

return 0;

}

string InfixToPostfix(string infix)

{

//method to convert the infix form into the postfix form.

int i = 0; //loop counter

string postfix = ""; //postfix string

while(infix[i] != '\0')

{

//loop continues until end of expression is reached.

if((infix[i]>='a' && infix[i]<='z') || (infix[i]>='A'&& infix[i]<='Z') || (infix[i] <= '9' && infix[i] >= '0'))

{

//if letters or numbers appear

postfix = postfix + infix[i];

i++;

}

else

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

{

//if opening brace appear

Push(infix[i]);

i++;

}

else

if(infix[i]==')' || infix[i]=='}' || infix[i]==']')

{

//if closing brace appear.

if(infix[i] == ')')

{

while(array[top] != '(')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

if(infix[i] == ']')

{

while(array[top] != '[')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

if(infix[i] == '}')

{

while(array[top] != '{')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

}

else

{

//if the operator appears.

if(top == -1)

{

//if the stack is emptyyte

Push(infix[i]);

i++;

}

else

if(Precedence(infix[i]) <= Precedence(array[top]))

{

//if the precedence of current operator is less than the operator on the top

postfix = postfix +" ";

postfix = postfix + Pop();

while(Precedence(infix[i]) == Precedence(array[top]))

{

//operators from stack printed to postfix expression until the opeartor precedence in stack becomes equal to the precedence of the current oprator.

postfix = postfix + Pop();

if(top < 0)

{

break;

}

}

Push(infix[i]);

i++;

}

else

if(Precedence(infix[i]) > Precedence(array[top]))

{

//if the precedence of current operator is greater than operator in stack it is simply printed in the stack

postfix = postfix +" ";

Push(infix[i]);

i++;

}

}

}

while(top!= -1)

{

//after all the operators finish the stack operators are printed to the postfix expression.

postfix = postfix + Pop();

}

cout << "The converted infix expression is : " << postfix << endl;

return postfix; //returns postfix expression.

}

};

int main()

{

Stack <char> \*stackChar = new Stack<char>();

string infix,postfix;

//getting input from the user.

cout << "Enter the infix expresion : " ;

cin >> infix;

if(stackChar -> CheckBalancedParenthesis(infix))

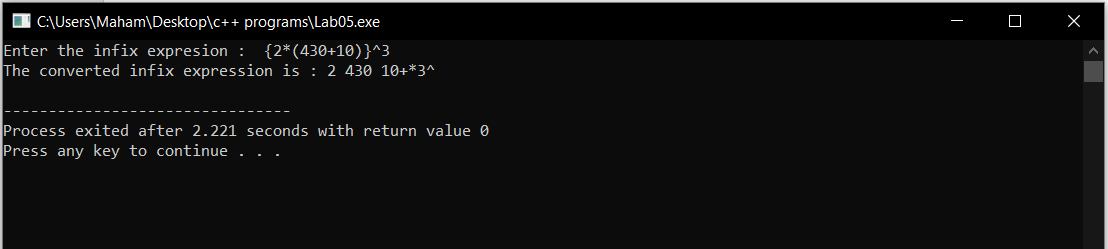
{

postfix = stackChar -> InfixToPostfix(infix);

}

}

## output:



# Task 04

## Code:

# include <iostream>

#include<string.h>

#include<sstream>

#include<cmath>

using namespace std;

#define SIZESTACK 100

template <class StackType> class Stack

{

//generic stack class.

public:

int top; //to keep track of the top of the stack

StackType array[SIZESTACK]; //array for implementing array based implementation.

Stack()

{

top = -1; //initializing the stack

}

bool isEmpty()

{

//method to check if the stack is empty

if(top == -1)

{

//if stack empty

return true;

}

else

{

//if stack is not empty

return false;

}

}

bool isFull()

{

//method to check if the stack is full

if(top == (SIZESTACK-1))

{

//stack full

return true;

}

else

{

return false;

}

}

void Push(StackType x)

{

//method to push the values in the stack

if(isFull())

{

cout << "push is full Stack Overflow.";

}

else

{

top++;

array[top] = x;

}

//PrintStack();

}

StackType Pop()

{

//method to pop the values out of the stack

if(isEmpty())

{

cout << "pop is empty Stack Underflow.";

return -1;

}

else

{

StackType x = array[top];

array[top] = '\0';

top--;

return x;

}

}

StackType Top()

{

//method to get the top value stored in the stack

if(isEmpty())

{

cout << "top is empty Stack Underflow.";

return -1;

}

else

{

return (array[top]);

}

}

void PrintStack()

{

//method to print the stack on the screen

while(top != -1)

{

cout << " " << Top();

Pop();

}

cout << endl;

}

bool CheckBalancedParenthesis(string s)

{

int i = 0;

int flag = 1;

char space = ' ';

for(i =0; i < s.length() ; i++)

{

//checks if the character is letter special character or mathematical signs.

if(((s[i] >= 'a') && (s[i] <= 'z')) || ((s[i] >= 'A') && (s[i] <= 'Z') ) || (s[i] == '+') || (s[i] == '-') || (s[i] == '/') || (s[i] == '\*') || (s[i] == '.') || (s[i] == '^') || ((s[i] >= '0') && (s[i] <= '9')))

{

continue;

}

//if opening braces come

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

{

//push in the stack

Push(s[i]);

continue;

}

if(!isEmpty())

{

//if closing brace comes

if(s[i] == '}')

{

if(Top() == '{')

{

//removes the pair of braces.

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

if(s[i] == ']')

{

if(Top() == '[')

{

//removes the pair of braces

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

if(s[i] == ')')

{

if(Top() == '(')

{

//removes the pair of braces

Pop();

continue;

}

else

{

//if not pair then loop breaks.

flag = 0;

break;

}

}

}

else

{

flag = 0;

break;

}

}

if(isEmpty() && flag == 1)

{

return true;

}

else

{

cout << endl;

cout << "The string you entered does not have balanced paranthesis. Error at character # " << i << " "<< Top() << " should be closed." << endl;

return false;

}

}

int Precedence(char op)

{

//method to check the precedence of the operator.

if(op == '^')

{

return 3;

}

if((op == '\*') || (op == '/'))

{

return 2;

}

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

{

return 1;

}

return 0;

}

string InfixToPostfix(string infix)

{

//method to convert the infix form into the postfix form.

int i = 0; //loop counter

string postfix = ""; //postfix string

while(infix[i] != '\0')

{

//loop continues until end of expression is reached.

if((infix[i]>='a' && infix[i]<='z') || (infix[i]>='A'&& infix[i]<='Z') || (infix[i] <= '9' && infix[i] >= '0'))

{

//if letters or numbers appear

postfix = postfix + infix[i];

i++;

}

else

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

{

//if opening brace appear

Push(infix[i]);

i++;

}

else

if(infix[i]==')' || infix[i]=='}' || infix[i]==']')

{

//if closing brace appear.

if(infix[i] == ')')

{

while(array[top] != '(')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

if(infix[i] == ']')

{

while(array[top] != '[')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

if(infix[i] == '}')

{

while(array[top] != '{')

{

postfix = postfix + Pop();

}

Pop();

i++;

}

}

else

{

//postfix = postfix +" ";

//if the operator appears.

if(top == -1)

{

//if the stack is emptyyte

Push(infix[i]);

i++;

}

else

if(Precedence(infix[i]) <= Precedence(array[top]))

{

//if the precedence of current operator is less than the operator on the top

postfix = postfix +" ";

postfix = postfix + Pop();

while(Precedence(infix[i]) == Precedence(array[top]))

{

//operators from stack printed to postfix expression until the opeartor precedence in stack becomes equal to the precedence of the current oprator.

postfix = postfix + Pop();

if(top < 0)

{

break;

}

}

Push(infix[i]);

i++;

}

else

if(Precedence(infix[i]) > Precedence(array[top]))

{

//if the precedence of current operator is greater than operator in stack it is simply printed in the stack

postfix = postfix +" ";

Push(infix[i]);

i++;

}

}

}

while(top!= -1)

{

//after all the operators finish the stack operators are printed to the postfix expression.

postfix = postfix + Pop();

}

cout << "The converted infix expression is : " << postfix << endl;

return postfix; //returns postfix expression.

}

int EvaluatePostfixExpression(string postfix)

{

string s="";

Stack <int> \*stackInt = new Stack<int>();

int result = 0;

int i = 0 ;

while(postfix[i] != '\0')

{

//if digit

if(isdigit(postfix[i]))

{

while(postfix[i] != ' ')

{

s+=postfix[i];

i++;

}

stringstream deg(s);

int x=0;

deg >> x;

cout<<x<<endl;

stackInt -> Push((x));

s="";

i--;

}

if(postfix[i] == ' ')

{

//if space occurs

i++;

continue;

}

else

{

int num1 = stackInt -> Pop();

cout<<"printing num1 " <<num1 << endl;

int num2 = stackInt -> Pop();

cout<<"printing num2 " <<num2 << endl;

switch(postfix[i])

{

case '^':

{

result = pow(num1,num2);

break;

}

case '\*':

{

result = num1 \* num2;

break;

}

case '/':

{

result = num1 / num2;

break;

}

case '+':

{

result = num1 + num2;

break;

}

case '-':

{

result = num1 - num2;

break;

}

default:

{

break;

}

stackInt -> Push(result);

i++;

}

}

}

return stackInt -> Pop();

}

};

int main()

{

Stack <char> \*stackChar = new Stack<char>();

string infix,postfix;

//getting input from the user.

cout << "Enter the infix expresion : " ;

cin >> infix;

if(stackChar -> CheckBalancedParenthesis(infix))

{

postfix = stackChar -> InfixToPostfix(infix);

}

Stack <int> \*stackInt = new Stack<int>();

int result = stackInt -> EvaluatePostfixExpression(postfix);

cout << result;

}