National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 05

# Problem 01

#include <iostream>

using namespace std;

class IntStack

{

private:

int top;

int \*stackArray;

int stackSize;

void initializeStack(size\_t size);

public:

IntStack(size\_t size);

void push(int data);

int pop(int &value);

bool isFull();

bool isEmpty();

int topElement();

};

int main()

{

IntStack stack(10);

cout << boolalpha;

int popValue;

cout << "Stack empty: " << stack.isEmpty() << endl;

stack.pop(popValue);

stack.push(5);

stack.push(10);

stack.push(15);

stack.push(20);

cout << "Stack empty: " << stack.isEmpty() << endl;

cout << "Top element on the stack: " << stack.topElement() << endl;

stack.pop(popValue);

stack.pop(popValue);

cout << "Stack full: " << stack.isFull() << endl;

stack.push(25);

stack.push(30);

stack.push(35);

stack.push(40);

stack.push(45);

stack.push(50);

stack.push(55);

stack.push(60);

stack.push(65);

cout << "Stack full: " << stack.isFull() << endl;

stack.pop(popValue);

stack.pop(popValue);

cout << "Top element on the stack: " << stack.topElement() << endl;

system("pause");

return 0;

}

void IntStack::initializeStack(size\_t size)

{

stackArray = new int[size];

top = -1;

stackSize = size;

}

IntStack::IntStack(size\_t size)

{

if (size > 0)

{

initializeStack(size);

}

else

{

cout << "Size of stack cannot be -ve" << endl;

}

}

bool IntStack::isFull()

{

if (top == stackSize - 1)

{

return true;

}

else

{

return false;

}

}

bool IntStack::isEmpty()

{

if (top == -1)

{

return true;

}

else

{

return false;

}

}

void IntStack::push(int data)

{

if (isFull())

{

cout << "Stack overflow -> Value " << data << " has NOT been pushed to stack" << endl;

}

else

{

top++;

stackArray[top] = data;

cout << "Value " << data << " has been pushed to stack" << endl;

}

}

int IntStack::pop(int &value)

{

if (isEmpty())

{

cout << "Stack underflow -> cannot pop from empty stack" << endl;

value = INT\_MIN;

return value;

}

else

{

cout << "Value " << stackArray[top] << " has been popped" << endl;

value = stackArray[top];

top--;

return value;

}

}

int IntStack::topElement()

{

if (!isEmpty())

{

return stackArray[top];

}

else

{

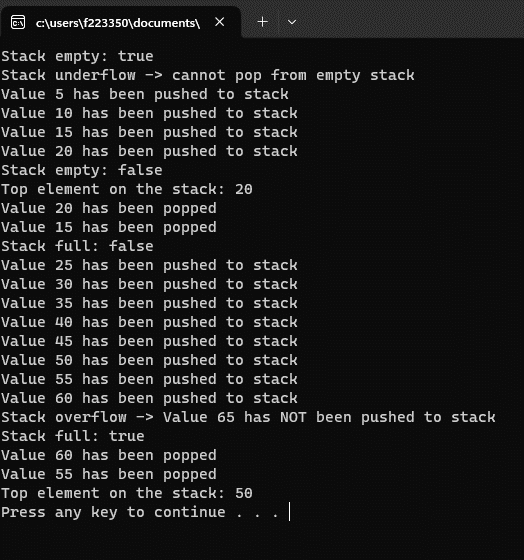
cout << "Stack underflow (Empty)" << endl;

return INT\_MIN;

}

}

## Output



# Problem 02

#include <iostream>

#include <stack>

#include <string>

using namespace std;

struct Node

{

int data;

Node \*next;

Node()

: data(0), next(0) {}

Node(int data)

: data(data), next(nullptr) {}

};

class StackLL

{

private:

bool isOperator(char c);

bool highPrec(char op1, char op2);

Node \*top;

void freeStack()

{

while (!isEmpty())

{

pop();

}

}

public:

StackLL()

: top(nullptr) {}

~StackLL() { freeStack(); }

bool isEmpty();

void push(int data);

int pop();

int topElement();

string infixToPostfix(string expression);

string postfixToInfix(string postfixExpr);

};

int main()

{

StackLL stack;

string infixExpr1 = "a+b\*(c^d-e)^(f+g\*h)-I";

string postfixExpr1 = stack.infixToPostfix(infixExpr1);

cout << "Infix Expression: " << infixExpr1 << endl;

cout << "Postfix Expression: " << postfixExpr1 << endl;

string postfixExpr2 = postfixExpr1;

string infixExpr2 = stack.postfixToInfix(postfixExpr1);

cout << "Postfix Expression: " << postfixExpr2 << endl;

cout << "Infix Expression: " << infixExpr2 << endl;

system("pause");

return 0;

}

bool StackLL::isEmpty()

{

if (top == nullptr)

{

return true;

}

else

{

return false;

}

}

void StackLL::push(int data)

{

Node \*newNode = new Node(data);

newNode->next = top;

cout << "Pushed " << data << endl;

top = newNode;

}

int StackLL::pop()

{

if (!isEmpty())

{

Node \*temp = top;

top = top->next;

temp->next = nullptr;

int poppedVal = temp->data;

cout << "Popped " << temp->data << endl;

delete temp;

return poppedVal;

}

else

{

cout << "Stack underflow (Empty)" << endl;

return INT\_MIN;

}

}

int StackLL::topElement()

{

if (!isEmpty())

{

return top->data;

}

else

{

return INT\_MIN;

}

}

bool StackLL::isOperator(char c)

{

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

}

bool StackLL::highPrec(char op1, char op2)

{

int op1Weight, op2Weight;

switch (op1)

{

case '+':

case '-':

op1Weight = 1;

break;

case '\*':

case '/':

op1Weight = 2;

break;

case '^':

op1Weight = 3;

break;

default:

op1Weight = 0;

break;

}

switch (op2)

{

case '+':

case '-':

op2Weight = 1;

break;

case '\*':

case '/':

op2Weight = 2;

break;

case '^':

op2Weight = 3;

break;

default:

op2Weight = 0;

break;

}

return op1Weight >= op2Weight;

}

string StackLL::infixToPostfix(string expression)

{

stack<char> operatorStack;

string postfixStr = "";

for (char c : expression)

{

if (isalnum(c))

{

postfixStr += c;

}

else if (c == '(')

{

operatorStack.push(c);

}

else if (c == ')')

{

while (!operatorStack.empty() && operatorStack.top() != '(')

{

postfixStr += operatorStack.top();

operatorStack.pop();

}

if (!operatorStack.empty() && operatorStack.top() == '(')

{

operatorStack.pop();

}

}

else if (isOperator(c))

{

while (!operatorStack.empty() && operatorStack.top() != '(' && highPrec(operatorStack.top(), c))

{

postfixStr += operatorStack.top();

operatorStack.pop();

}

operatorStack.push(c);

}

}

while (!operatorStack.empty())

{

postfixStr += operatorStack.top();

operatorStack.pop();

}

return postfixStr;

}

string StackLL::postfixToInfix(string postfixExpr)

{

stack<string> operandStack;

for (char c : postfixExpr)

{

if (isalpha(c))

{

string operand(1, c);

operandStack.push(operand);

}

else if (isOperator(c))

{

string operand2 = operandStack.top();

operandStack.pop();

string operand1 = operandStack.top();

operandStack.pop();

string infix = "(" + operand1 + c + operand2 + ")";

operandStack.push(infix);

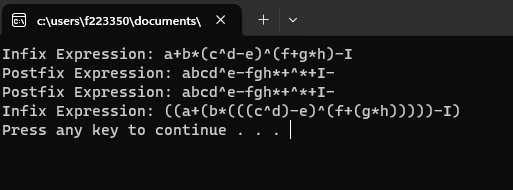
}

}

return operandStack.top();

}

## Output



# Problem 03

#include <iostream>

using namespace std;

class IntStack

{

private:

int top;

int \*stackArray;

int stackSize;

void initializeStack(size\_t size);

public:

IntStack(size\_t size);

~IntStack();

void push(int data);

int pop(int &value);

bool isFull();

bool isEmpty();

int topElement();

void reverseStack();

};

int main()

{

IntStack stack(10);

int popValue;

for (int i = 1; i <= 5; i++)

{

stack.push(i \* 10);

cout << i \* 10 << " pushed" << endl;

}

stack.reverseStack();

for (int i = 0; i < 5; i++)

{

cout << "Popped: " << stack.pop(popValue) << endl;

}

system("pause");

return 0;

}

void IntStack::initializeStack(size\_t size)

{

stackArray = new int[size];

top = -1;

stackSize = size;

}

IntStack::~IntStack()

{

delete[] stackArray;

stackArray = nullptr;

}

IntStack::IntStack(size\_t size)

{

if (size > 0)

{

initializeStack(size);

}

else

{

cout << "Size of stack cannot be -ve" << endl;

}

}

bool IntStack::isFull()

{

if (top == stackSize - 1)

{

return true;

}

else

{

return false;

}

}

bool IntStack::isEmpty()

{

if (top == -1)

{

return true;

}

else

{

return false;

}

}

void IntStack::push(int data)

{

if (isFull())

{

cout << "Stack overflow -> Value " << data << " has NOT been pushed to stack" << endl;

}

else

{

top++;

stackArray[top] = data;

}

}

int IntStack::pop(int &value)

{

if (isEmpty())

{

cout << "Stack underflow -> cannot pop from empty stack" << endl;

value = INT\_MIN;

return value;

}

else

{

value = stackArray[top];

top--;

return value;

}

}

int IntStack::topElement()

{

if (!isEmpty())

{

return stackArray[top];

}

else

{

cout << "Stack underflow (Empty)" << endl;

return INT\_MIN;

}

}

void IntStack::reverseStack()

{

int \*temp = new int[stackSize];

int index = 0;

int val;

while (!isEmpty())

{

temp[index] = pop(val);

index++;

}

int val1 = index;

index = 0;

while (!isFull() && index < val1)

{

push(temp[index]);

index++;

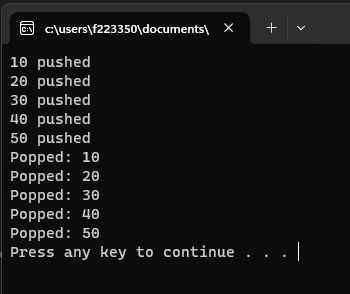
}

delete temp;

temp = nullptr;

}

## Output



# Problem 04

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*next;

Node()

: data(0), next(0) {}

Node(int data)

: data(data), next(nullptr) {}

};

class StackLL

{

private:

Node \*top;

void freeStack()

{

while (!isEmpty())

{

pop();

}

cout << "Stack empty: " << isEmpty() << endl;

}

public:

StackLL()

: top(nullptr) {}

~StackLL() { freeStack(); }

bool isEmpty();

void push(int data);

int pop();

int topElement();

};

int main()

{

StackLL \*stack = new StackLL;

cout << boolalpha;

cout << "Stack empty: " << stack->isEmpty() << endl;

stack->push(10);

stack->push(20);

stack->push(30);

stack->push(40);

stack->push(50);

cout << "Stack empty: " << stack->isEmpty() << endl;

cout << "Top Element: " << stack->topElement() << endl;

delete stack;

system("pause");

return 0;

}

bool StackLL::isEmpty()

{

if (top == nullptr)

{

return true;

}

else

{

return false;

}

}

void StackLL::push(int data)

{

Node \*newNode = new Node(data);

newNode->next = top;

cout << "Pushed " << data << endl;

top = newNode;

}

int StackLL::pop()

{

if (!isEmpty())

{

Node \*temp = top;

top = top->next;

temp->next = nullptr;

int poppedVal = temp->data;

cout << "Popped " << temp->data << endl;

delete temp;

return poppedVal;

}

else

{

cout << "Stack underflow (Empty)" << endl;

return INT\_MIN;

}

}

int StackLL::topElement()

{

if (!isEmpty())

{

return top->data;

}

else

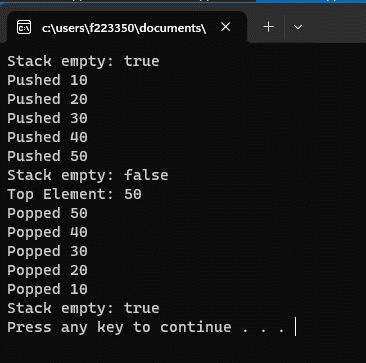
{

return INT\_MIN;

}

}

## Output



# Problem 05

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int data;

Node \*next;

Node()

: data(0), next(0) {}

Node(int data)

: data(data), next(nullptr) {}

};

class StackLL

{

private:

Node \*top;

void freeStack()

{

while (!isEmpty())

{

pop();

}

}

public:

StackLL()

: top(nullptr) {}

~StackLL() { freeStack(); }

bool isEmpty();

void push(int data);

int pop();

int topElement();

bool isBalanced(string exp);

};

int main()

{

StackLL stack;

cout << boolalpha;

string exp1 = "{[{}{}]}[()]";

cout << exp1 << " is balanced: " << stack.isBalanced(exp1) << endl;

string exp2 = "{{}{}}";

cout << exp2 << " is balanced: " << stack.isBalanced(exp2) << endl;

string exp3 = "[]{}()";

cout << exp3 << " is balanced: " << stack.isBalanced(exp3) << endl;

string exp4 = "{()}[)";

cout << exp4 << " is balanced: " << stack.isBalanced(exp4) << endl;

string exp5 = "{(})";

cout << exp5 << " is balanced: " << stack.isBalanced(exp5) << endl;

system("pause");

return 0;

}

bool StackLL::isEmpty()

{

if (top == nullptr)

{

return true;

}

else

{

return false;

}

}

void StackLL::push(int data)

{

Node \*newNode = new Node(data);

newNode->next = top;

top = newNode;

}

int StackLL::pop()

{

if (!isEmpty())

{

Node \*temp = top;

top = top->next;

temp->next = nullptr;

int poppedVal = temp->data;

delete temp;

return poppedVal;

}

else

{

cout << "Stack underflow (Empty)" << endl;

return INT\_MIN;

}

}

int StackLL::topElement()

{

if (!isEmpty())

{

return top->data;

}

else

{

return INT\_MIN;

}

}

bool StackLL::isBalanced(string exp)

{

StackLL stack;

for (char c : exp)

{

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

{

stack.push(c);

}

else if (c == ')' || c == ']' || c == '}')

{

if (stack.isEmpty())

{

return false;

}

char top = stack.pop();

if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{'))

{

return false;

}

}

}

return stack.isEmpty();

}

## Output

