```
/*
File name: postfix.cpp
Created by: Tan Qi Hao
Created on: 10/17/2019
Synopsis: This program used stack to evaluate expression that is in
          postfix notations. eg. 3 4 - 5 + = 4

*/

#include <iostream>
#include <cstdlib>

using namespace std;

const char SENTINEL = ';';

struct Node {
    int data;
    Node *link;
};

// precondition: c is initialized
// postcondition: returns true if c is '+', '-', '*' or '/'
bool isOperator(char c);

// precondition: o1 and o2 are initialized and op is an operator
// postcondition: returns op(o1, o2), e.g. if op is '-' then returns o1-o2
int calculate(int o1, int o2, char op);

// precondition: c is a digit
// postcondition: returns the integer value of c
int charToInt(char c);

class Stack {

public:
    // default constructor
    // initializes the stack to an empty stack
    Stack();

    // this is a const function, meaning it cannot change any of the member
variables
    // returns true if the stack is empty, false otherwise
    bool isEmpty() const;

    // this is a const function, meaning it cannot change any of the member
variables
    // returns the value at the top of stack, does not modify the stack, does not
check if the stack is empty or not
    int top() const;

    // adds data to the top of stack
    void push(int data);

    // removes the top value of stack, does not return it, does nothing if the
stack is empty
    void pop();

private:
    Node *listHead; // pointer to the head of a linked list

};

int main() {
    char in;
    Stack operandStack;

    cout << "Enter a postfix expression (ending with " << SENTINEL << " and press
Enter):\n";
    cin >> in ;
    while ( in != SENTINEL) {
        if (isOperator( in )) {
            // pop two numbers from stack
            int n1, n2;
            if (operandStack.isEmpty()) {
                // print error message

            cout << "Not enough operands entered for operator: "
               << in << endl;

                exit(1);
            }
            n2 = operandStack.top();
            operandStack.pop();

            if (operandStack.isEmpty()) {
                // pring error message

            cout << "Not enough operands entered for operator: "
               << in << endl;
                exit(1);
            }
            n1 = operandStack.top();
            operandStack.pop();

            // push the result of calculation to the top of operandStack
          int calculation = calculate(n1, n2, in);
            operandStack.push(calculation);

        } else {
            // push the number to the top of opernadStack
        operandStack.push(charToInt(in));

        }
        cin >> in ;
    }

    // pop a number from the top of stack
    int result;
    result = operandStack.top();
    operandStack.pop();

    if (operandStack.isEmpty()) // nothing left in the stack
        cout << "\nThe result is: " << result << endl;
    else // there are still numbers in the stack
    {
        // print an error message
```

```
      cout << "Not enough operators entered! " << endl;
    }

    return 0;
}

//This default constructor initialize the stack to an empty stack
Stack::Stack() {
  listHead = NULL;
}

//This function determine whether the stack is empty or not
bool Stack::isEmpty() const {

  if(listHead == NULL){

    return true;

  }else{

    return false;
  }

}

//This function return the top of the stack
int Stack::top() const {

  return listHead -> data;
}

//This function removes the top of stack
void Stack::pop() {

  if(listHead != NULL){ //Do nothing when the stack is empty
    Node *temp_ptr = listHead; // Temporary pointer

    listHead = listHead -> link;
    delete [] temp_ptr; //Delete the pointer
  }

}

//Push a number to the top of stack
void Stack::push(int data) {

  Node *AddNode; //Temporary pointer to add node
  AddNode = new Node;

  AddNode -> data = data;

  AddNode -> link = listHead;

  listHead = AddNode;
}

//Calculate the result based on the operators
int calculate(int o1, int o2, char op) {
  if(op == '+'){
    return o1 + o2;

  }else if(op == '-'){
    return o1 - o2;

  }else if(op == '*'){
    return o1 * o2;

  }else if(op == '/'){
    return o1 / o2;

  }
}

bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

int charToInt(char c) {
        return (static_cast<int>(c) - static_cast<int>('0'));}
```