National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 06

# Problem 01

#include <iostream>

using namespace std;

class Stack

{

private:

int top;

int \*stackArray;

int stackSize;

public:

Stack(int size);

bool push(int data);

int pop();

bool isEmpty();

int peek();

};

int main()

{

cout << boolalpha;

Stack stack(10);

cout << "Before pushing" << endl;

cout << "Stack empty or not: " << stack.isEmpty() << endl;

// pushing stack

for (int i = 1; i <= 11; i++)

{

stack.push(i \* 10);

}

cout << "After pushing" << endl;

cout << "Stack empty or not: " << stack.isEmpty() << endl;

cout << "Peek element: " << stack.peek() << endl;

// popping stack

for (int i = 1; i <= 11; i++)

{

stack.pop();

}

system("pause");

return 0;

}

Stack::Stack(int size)

{

if (size > 0)

{

stackArray = new int[size];

top = -1;

stackSize = size;

}

else

{

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

}

}

bool Stack::push(int data)

{

if (top == stackSize - 1)

{

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

return false;

}

else

{

top++;

stackArray[top] = data;

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

return true;

}

}

int Stack::pop()

{

if (isEmpty())

{

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

return INT\_MIN;

}

else

{

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

return stackArray[top--];

}

}

int Stack::peek()

{

if (!isEmpty())

{

return stackArray[top];

}

else

{

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

return INT\_MIN;

}

}

bool Stack::isEmpty()

{

if (top == -1)

{

return true;

}

else

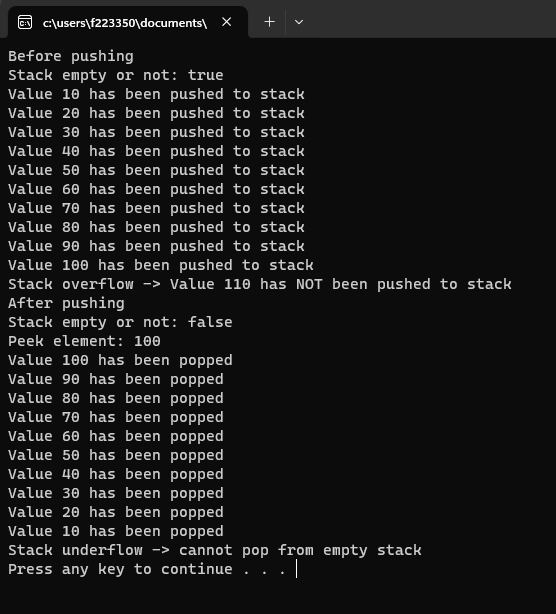
{

return false;

}

}

## Output



# Problem 02

#include <iostream>

#include <string>

using namespace std;

class Stack

{

private:

int top;

char \*stackArray;

int stackSize;

public:

Stack(int size);

bool push(char data);

char pop();

bool isEmpty();

char peek();

int precedence(char op);

string convertToPostfix(string infix);

};

int main()

{

Stack stack(10);

string infixExpression;

cout << "Enter an infix expression: ";

getline(cin, infixExpression);

string postfixExpression = stack.convertToPostfix(infixExpression);

cout << "Postfix expression: " << postfixExpression << endl;

system("pause");

return 0;

}

Stack::Stack(int size)

{

if (size > 0)

{

stackArray = new char[size];

top = -1;

stackSize = size;

}

else

{

cout << "Size of stack cannot be negative" << endl;

}

}

bool Stack::push(char data)

{

if (top == stackSize - 1)

{

return false;

}

else

{

top++;

stackArray[top] = data;

return true;

}

}

char Stack::pop()

{

if (isEmpty())

{

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

return '\0';

}

else

{

return stackArray[top--];

}

}

char Stack::peek()

{

if (!isEmpty())

{

return stackArray[top];

}

else

{

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

return '\0';

}

}

bool Stack::isEmpty()

{

return top == -1;

}

int Stack::precedence(char op)

{

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

{

return 1;

}

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

{

return 2;

}

else if (op == '^')

{

return 3;

}

else

{

return -1;

}

}

string Stack::convertToPostfix(string infix)

{

string postfix;

int infixLength = infix.length();

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

{

if (infix[i] == ' ' || infix[i] == '\t')

{

continue;

}

else if (isdigit(infix[i]))

{

while (i < infixLength && isdigit(infix[i]))

{

postfix += infix[i++];

}

postfix += ' ';

i--;

}

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

{

push(infix[i]);

}

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

{

while (!isEmpty() && peek() != '(')

{

postfix += pop();

postfix += ' ';

}

if (!isEmpty() && peek() == '(')

{

pop();

}

}

else

{

while (!isEmpty() && peek() != '(' && (precedence(infix[i]) <= precedence(peek())))

{

postfix += pop();

postfix += ' ';

}

push(infix[i]);

}

}

while (!isEmpty())

{

postfix += pop();

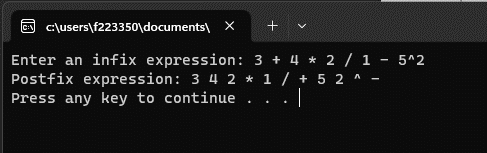
postfix += ' ';

}

return postfix;

}

## Output



# Problem 03

#include <iostream>

#include <string>

using namespace std;

class Stack

{

private:

int top;

char \*stackArray;

int stackSize;

public:

Stack(int size);

bool push(char data);

char pop();

bool isEmpty();

char peek();

int precedence(char op);

string convertToPostfix(string infix);

int evaluatePostfix(string postfix);

};

int main()

{

Stack stack(10);

string postfixExp;

cout << "Enter an infix expression: ";

getline(cin, postfixExp);

int result = stack.evaluatePostfix(postfixExp);

cout << "Result: " << result << endl;

system("pause");

return 0;

}

Stack::Stack(int size)

{

if (size > 0)

{

stackArray = new char[size];

top = -1;

stackSize = size;

}

else

{

cout << "Size of stack cannot be negative" << endl;

}

}

bool Stack::push(char data)

{

if (top == stackSize - 1)

{

return false;

}

else

{

top++;

stackArray[top] = data;

return true;

}

}

char Stack::pop()

{

if (isEmpty())

{

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

return '\0';

}

else

{

return stackArray[top--];

}

}

char Stack::peek()

{

if (!isEmpty())

{

return stackArray[top];

}

else

{

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

return '\0';

}

}

bool Stack::isEmpty()

{

if (top == -1)

{

return true;

}

else

{

return false;

}

}

int Stack::precedence(char op)

{

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

{

return 1;

}

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

{

return 2;

}

else if (op == '^')

{

return 3;

}

else

{

return -1;

}

}

string Stack::convertToPostfix(string infix)

{

string postfix;

int infixLength = infix.length();

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

{

if (infix[i] == ' ' || infix[i] == '\t')

{

continue;

}

else if (isdigit(infix[i]))

{

while (i < infixLength && isdigit(infix[i]))

{

postfix += infix[i++];

}

postfix += ' ';

i--;

}

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

{

push(infix[i]);

}

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

{

while (!isEmpty() && peek() != '(')

{

postfix += pop();

postfix += ' ';

}

if (!isEmpty() && peek() == '(')

{

pop();

}

}

else

{

while (!isEmpty() && peek() != '(' && (precedence(infix[i]) <= precedence(peek())))

{

postfix += pop();

postfix += ' ';

}

push(infix[i]);

}

}

while (!isEmpty())

{

postfix += pop();

postfix += ' ';

}

return postfix;

}

int Stack::evaluatePostfix(string postfix)

{

int length = postfix.length();

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

{

if (postfix[i] == ' ' || postfix[i] == '\t')

{

continue;

}

else if (isdigit(postfix[i]))

{

int operand = 0;

while (i < length && isdigit(postfix[i]))

{

operand = (operand \* 10) + (postfix[i] - '0');

i++;

}

i--;

push(operand);

}

else

{

int operand2 = pop();

int operand1 = pop();

switch (postfix[i])

{

case '+':

push(operand1 + operand2);

break;

case '-':

push(operand1 - operand2);

break;

case '\*':

push(operand1 \* operand2);

break;

case '/':

push(operand1 / operand2);

break;

case '^':

push(pow(operand1, operand2));

break;

default:

cout << "Invalid operator" << endl;

return 0;

}

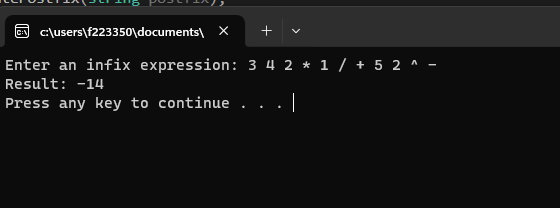
}

}

return pop();

}

## Output



# Problem 04

#include <iostream>

#include <string>

using namespace std;

class Book

{

public:

Book(int bookNum, string bookName, float pr, int edit, string cat)

{

book\_no = bookNum;

book\_name = bookName;

price = pr;

edition = edit;

category = cat;

}

int book\_no;

string book\_name;

float price;

int edition;

string category;

};

class Node

{

public:

Book \*book;

Node \*next;

};

class Stack

{

public:

Node \*top;

Stack() : top(nullptr) {}

~Stack()

{

while (!isEmpty())

{

pop();

}

}

bool isEmpty()

{

if (top == nullptr)

{

return true;

}

else

{

return false;

}

}

void push(Book \*book)

{

Node \*newNode = new Node;

newNode->book = book;

newNode->next = top;

top = newNode;

}

void pop()

{

if (isEmpty())

{

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

return;

}

Node \*temp = top;

top = top->next;

delete temp;

}

Book \*peek()

{

if (isEmpty())

{

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

return nullptr;

}

return top->book;

}

};

class LibraryMain

{

private:

Stack bookStack;

public:

void addBook(Book \*book)

{

bookStack.push(book);

}

Book \*issueBook()

{

if (bookStack.isEmpty())

{

cout << "No books available in the library." << endl;

return nullptr;

}

Book \*book = bookStack.peek();

bookStack.pop();

return book;

}

int getTotalBooks()

{

int count = 0;

Node \*currentNode = bookStack.top;

while (currentNode != nullptr)

{

count++;

currentNode = currentNode->next;

}

return count;

}

};

class LibraryDepartment

{

private:

string department;

int totalBooksCategoryWise;

Stack issuedBooks;

public:

LibraryDepartment(string dept) : department(dept), totalBooksCategoryWise(0) {}

void issueBook(Book \*book)

{

issuedBooks.push(book);

totalBooksCategoryWise++;

}

void showIssuedBooks()

{

cout << "Books issued by " << department << " department:" << endl;

Node \*currentNode = issuedBooks.top;

while (currentNode != nullptr)

{

Book \*book = currentNode->book;

cout << "Book Name: " << book->book\_name << endl;

cout << "Book No: " << book->book\_no << endl;

cout << "Price: " << book->price << endl;

cout << "Edition: " << book->edition << endl;

cout << "Category: " << book->category << endl;

currentNode = currentNode->next;

}

}

};

int main()

{

LibraryMain lib;

Book \*book1 = new Book(1, "Book 1", 10.99, 1, "Category 1");

Book \*book2 = new Book(2, "Book 2", 15.99, 2, "Category 2");

Book \*book3 = new Book(3, "Book 3", 20.99, 1, "Category 1");

Book \*book4 = new Book(4, "Book 4", 12.99, 3, "Category 3");

Book \*book5 = new Book(5, "Book 5", 18.99, 2, "Category 2");

lib.addBook(book1);

lib.addBook(book2);

lib.addBook(book3);

lib.addBook(book4);

lib.addBook(book5);

Book \*issuedBook = lib.issueBook();

if (issuedBook != nullptr)

{

cout << "Book issued:" << endl;

cout << "Book Name: " << issuedBook->book\_name << endl;

cout << "Book No: " << issuedBook->book\_no << endl;

cout << "Price: " << issuedBook->price << endl;

cout << "Edition: " << issuedBook->edition << endl;

cout << "Category: " << issuedBook->category << endl;

delete issuedBook;

}

LibraryDepartment dep1("Department 1");

LibraryDepartment dep2("Department 2");

dep1.issueBook(book1);

dep1.issueBook(book2);

dep2.issueBook(book3);

dep2.issueBook(book4);

dep1.showIssuedBooks();

dep2.showIssuedBooks();

return 0;

}

## Output

