/// Andrew Souza

/// Comp 210 -- Spring 2024

/// Exam 3

#include <iostream>

using namespace std;

// I will create a linked-list-based stack for more ease of dynamic sizing

struct Node {

int value = -1;

Node\* next = nullptr;

Node\* prev = nullptr;

};

class LinkedListStack {

private:

Node\* head = nullptr; // The head will be considered the "top" of the stack

// The tail is redundant in this case

unsigned int size = 0;

public:

void Push(int x) {

Node\* temp = new Node;

temp->value = x;

if (head == nullptr) {

head = temp;

} else {

temp->next = head;

head = temp;

}

size++;

}

int Pop() {

int poppedVal = head->value;

Node\* temp = head;

head = head->next;

delete temp;

size--;

return poppedVal;

}

int Peek() {

return head->value;

}

unsigned int GetSize() {

return size;

}

Node\* GetHeadData() {

return head;

}

};

class MathStack : public LinkedListStack {

public:

// +add - This member function will pop off the top two values off the stack

// , add them together and pushes their sum back on to the stack.

void add() {

if (GetSize() >= 2) {

int A = Pop();

int B = A + Pop();

Push(B);

} else {

cout << "Insufficent values for add\n";

}

}

// +sub - This member function pops the top two values off the stack,

// subtracts the first from the second and pushes the difference onto the stack.

void sub() {

if (GetSize() >= 2) {

int A = Pop();

int B = Pop() - A;

Push(B);

} else {

cout << "Insufficent values for sub\n";

}

}

// +mult - This member function pops the top two values off the stack ,

// multiples them and pushes their product onto the stack.

void mult() {

if (GetSize() >= 2) {

int A = Pop();

int B = Pop() \* A;

Push(B);

} else {

cout << "Insufficent values for mult\n";

}

}

// +div - This member function pops the top two values off the stack,

// divides the second value by the first, and pushes the quotient onto the stack.

void div() {

if (GetSize() >= 2) {

double A = Pop();

double B = Pop() / A;

Push(B);

} else {

cout << "Insufficent values for div\n";

}

}

// +print - This member function prints all the values held within

// the stack in order from top to bottom.

void print() {

Node\* temp = GetHeadData();

int i = 0;

while (temp != nullptr) {

cout << i++ << ": " << temp->value << "\n";

temp = temp->next;

}

}

};

int main() {

MathStack myStack;

myStack.Push(1);

myStack.Push(2);

myStack.Push(3);

myStack.Push(4);

myStack.Push(5);

myStack.Push(6);

myStack.Push(7);

myStack.Push(8);

myStack.add(); // 8 + 7 = 15

myStack.sub(); // 6 - 15 = -9

myStack.mult(); // 5 \* -9 = -45

myStack.div(); // 4 / -45 = 0 (integer division)

myStack.print();

cout << endl;

return 0;

}

