Банкузов Михаил 7 группа Лабораторная работа №7 Вариант 1



#include <iostream>

#include "stack.h"

using namespace std;

int main() {

Stack stack;

Stack stack1, stack2;

int choice;

do {

cout << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Peek" << endl;

cout << "4. Is empty?" << endl;

cout << "5. Clear" << endl;

cout << "6. Save to file" << endl;

cout << "7. Load from file" << endl;

cout << "8. Print from file" << endl;

cout << "9. Split stack" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

int data;

cout << "Enter data: ";

cin >> data;

stack.push(data);

break;

case 2:

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

break;

case 3:

cout << "Top of stack: " << stack.peek() << endl;

break;

case 4:

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

break;

case 5:

stack.clear();

cout << "Stack cleared" << endl;

break;

case 6:

stack.saveToFile();

cout << "Stack saved to file" << endl;

break;

case 7:

stack.loadFromFile();

cout << "Stack loaded from file" << endl;

break;

case 8:

stack.printFromFile();

break;

case 9:

stack.split(stack1, stack2);

cout << "Positive values:" << endl;

while (!stack1.isEmpty()) {

cout << stack1.pop() << endl;

}

cout << "Negative values:" << endl;

while (!stack2.isEmpty()) {

cout << stack2.pop() << endl;

}

break;

case 0:

cout << "Exiting program" << endl;

break;

default:

cout << "Invalid choice" << endl;

break;

}

} while (choice != 0);

}

// stack.cpp

#include <iostream>

#include <fstream>

#include "stack.h"

using namespace std;

Stack::Stack() {

head = nullptr; // Устанавливаем указатель на голову стека в значение нулевого указателя

}

Stack::~Stack() {

clear(); // Вызываем метод clear для освобождения памяти, выделенной под узлы стека

}

void Stack::push(int data) {

Node\* newNode = new Node; // Создаем новый узел стека

newNode->data = data; // Устанавливаем значение поля данных узла

newNode->next = head; // Устанавливаем указатель на следующий узел в цепочке на текущую голову стека

head = newNode; // Устанавливаем указатель на голову стека на только что созданный узел

}

int Stack::pop() {

if (isEmpty()) { // Проверяем, пуст ли стек

cerr << "Stack is empty" << endl; // Если стек пуст, выводим сообщение об ошибке

return 0; // Возвращаем нулевое значение

}

Node\* temp = head; // Создаем временный указатель на голову стека

int data = temp->data; // Сохраняем значение поля данных временного узла

head = temp->next; // Устанавливаем указатель на голову стека на следующий узел в цепочке

delete temp; // Освобождаем память, выделенную под временный узел

return data; // Возвращаем значение поля данных временного узла

}

int Stack::peek() {

if (isEmpty()) { // Проверяем, пуст ли стек

cerr << "Stack is empty" << endl; // Если стек пуст, выводим сообщение об ошибке

return 0; // Возвращаем нулевое значение

}

return head->data; // Возвращаем значение поля данных головного узла стека

}

bool Stack::isEmpty() {

return head == nullptr; // Проверяем, равен ли указатель на голову стека нулевому указателю

}

void Stack::clear() {

while (!isEmpty()) { // Пока стек не пуст

pop(); // Извлекаем узел из стека

}

}

void Stack::saveToFile() {

ofstream file("stack.txt"); // Открываем файл для записи

if (!file.is\_open()) { // Проверяем, удалось ли открыть файл

cerr << "Failed to open file for writing" << endl; // Если не удалось, выводим сообщение об ошибке

return; // Выходим из функции

}

Node\* current = head; // Создаем указатель на голову стека

while (current != nullptr) { // Пока указатель на текущий узел не является нулевым указателем

file << current->data << endl; // Записываем значение поля данных текущего узла в файл

current = current->next; // Устанавливает указатель

}

file.close();// Закрывает файл

}

void Stack::loadFromFile() {

ifstream file("stack.txt");// Открывает файл для чтения

if (!file.is\_open()) {// Проверка открытия

cerr << "Failed to open file for reading" << endl;

return;

}

clear();// Очищаем стек

int data;

while (file >> data) {// Считываются значения и добавляются в стек

push(data);

}

file.close();// Закрываем файл

}

void Stack::printFromFile() {

loadFromFile();// Функция загрузки

if (isEmpty()) {// Проверка

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

return;

}

cout << "Stack contents: ";

Node\* current = head;

while (current != nullptr) {// Перебираем все элементы стека

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void Stack::split(Stack& stack1, Stack& stack2) {

Node\* current = head;// Указатель на первый элемент

while (current != nullptr) {// Перебор всех элементов

if (current->data >= 0) {// Если больше 0, то в добавляет в первый стек

stack1.push(current->data);

}

else {// Если меньше 0, то добавляем во второй стек

stack2.push(current->data);

}

current = current->next;

}

}

// stack.h

struct Node {

int data;//данные, хранящиеся в узле

Node\* next;//указатель на следующий узел в списке

};

class Stack {

public:

Stack();//инициализирует пустой стек

~Stack();//удаляет все узлы в списке

void push(int data);//добавляет элемент на вершину стека

int pop();//удаляет и возвращает элемент с вершины стека

int peek();//возвращает элемент с вершины стека без удаления

bool isEmpty();//проверяет, пуст ли стек

void clear();//удаляет все элементы из стека

void saveToFile();//сохраняет содержимое стека в файле

void loadFromFile();//загружает содержимое стека из файла

void printFromFile();//выводит содержимое стека из файла в консоль

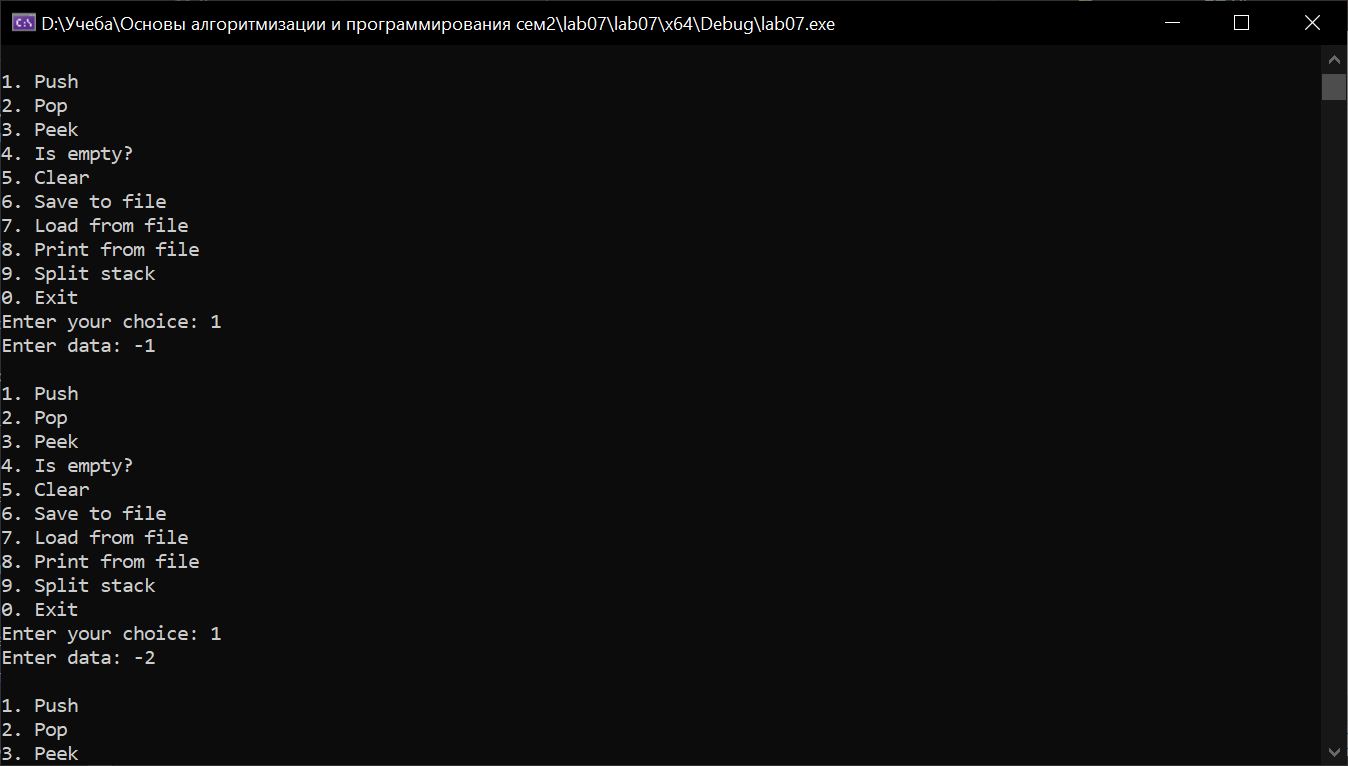
void split(Stack& stack1, Stack& stack2);//разделяет элементы стека на два стека в зависимости от их значения

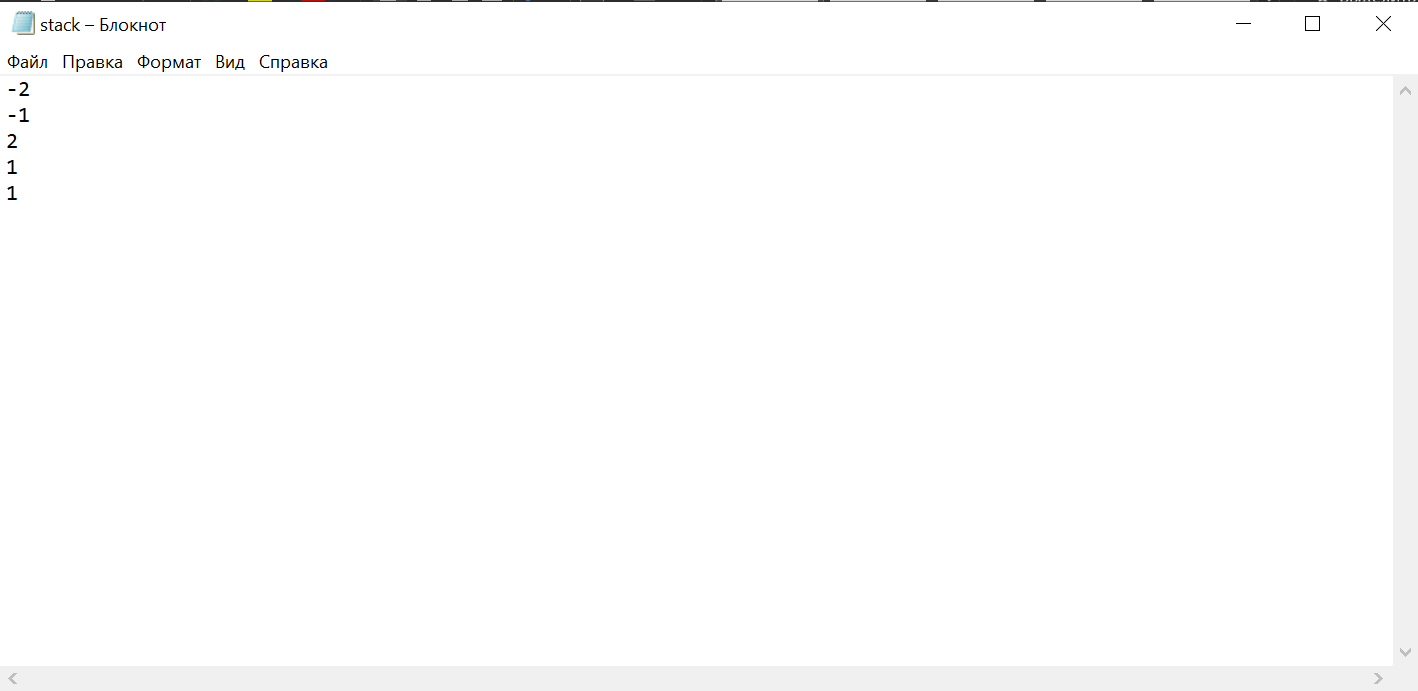
private:

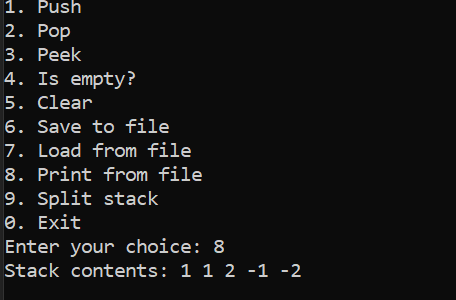
Node\* head;

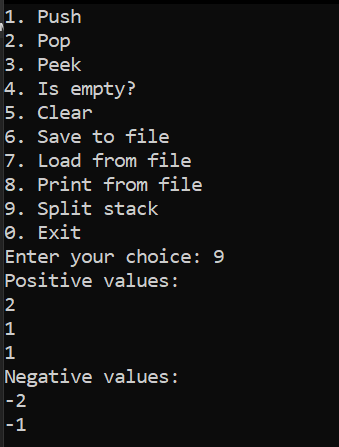
};

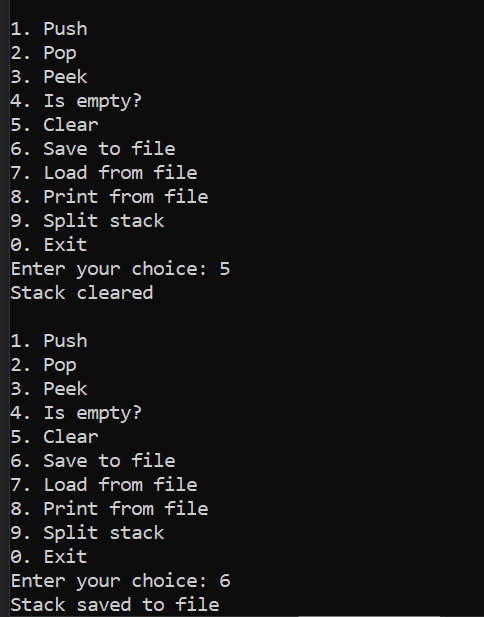
#pragma once

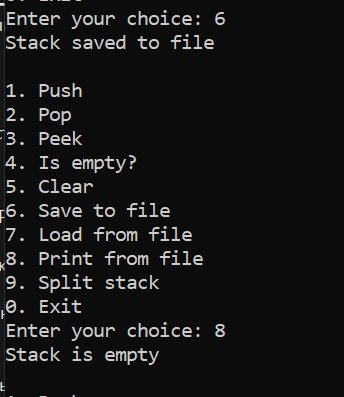




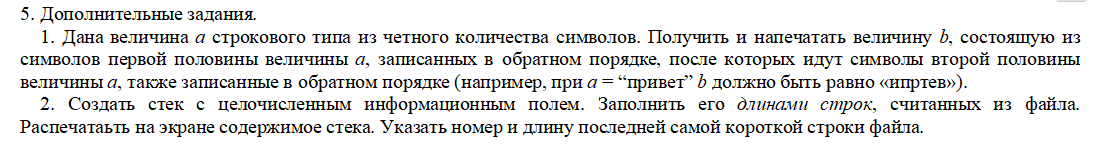








Дополнительные задания



Задание 1

#include <iostream>

#include <string>

using namespace std;

// определение структуры узла списка

struct Node {

char data; // данные узла (символ)

Node\* next; // указатель на следующий узел

};

// функция добавления элемента в стек

void push(Node\*& top, char c) {

Node\* newNode = new Node; // выделение памяти для нового узла

newNode->data = c; // сохранение переданного символа в новом узле

newNode->next = top; // указание на текущий верхний узел как следующий для нового

top = newNode; // изменение верхнего узла на новый

}

// функция извлечения элемента из стека

char pop(Node\*& top) {

if (top == nullptr) { // если стек пуст, возвращается нулевой символ

return '\0';

}

char c = top->data; // сохранение символа из верхнего узла

Node\* temp = top; // сохранение указателя на верхний узел

top = top->next; // изменение верхнего узла на следующий

delete temp; // освобождение памяти, занятой верхним узлом

return c; // возврат извлеченного символа

}

// главная функция

int main() {

setlocale(LC\_ALL, "Rus"); // установка локали для вывода русского текста

string a; // объявление переменной для ввода строки

cout << "Введите строку с четным числом символов: ";

cin >> a; // ввод строки пользователем

Node\* firstHalfTop = nullptr; // инициализация указателя на верхнюю часть стека первой половины строки

Node\* secondHalfTop = nullptr; // инициализация указателя на верхнюю часть стека второй половины строки

int n = a.length(); // вычисление длины введенной строки

// заполнение первой половины стека

for (int i = 0; i < n / 2; i++) {

push(firstHalfTop, a[i]);

}

// заполнение второй половины стека

for (int i = n / 2; i < n; i++) {

push(secondHalfTop, a[i]);

}

string b; // объявление переменной для результирующей строки

// извлечение символов из первой половины и добавление в результирующую строку

while (firstHalfTop != nullptr) {

b += pop(firstHalfTop);

}

// извлечение символов из второй половины и добавление в результирующую строку

while

(firstHalfTop != nullptr) {

b += pop(firstHalfTop);

}

while

(secondHalfTop != nullptr) {

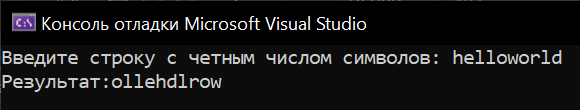
b += pop(secondHalfTop);

}

cout << "Результат:" << b << endl;

return 0;

}



Задание 2

#include <iostream>

#include <ctime>

#include <fstream>

#include <string>

#include <Windows.h>

#include <vector>

using namespace std;

struct Stack

{

int\* data;

int top;

friend void output(Stack& stack); // дружественная функция, которая имеет доступ к закрытым полям структуры Stack

};

void output(Stack& stack) // функция вывода элементов стека

{

for (int i = 0; i < stack.top; i++)

{

cout << stack.data[i] << '\t';

}

}

void find\_min(Stack& stack) // функция поиска минимального элемента стека и его позиции

{

int i = 0;

int min = stack.data[0];

for (; i < stack.top; i++)

{

if (min > stack.data[i])

{

min = stack.data[i];

}

}

cout << min << endl << "Находится под номером: " << i;

}

int main()

{

srand(time(NULL));

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Stack stack;

ifstream inputFile("fileA.txt"); // открытие файла для чтения

ofstream outputFile("fileB.txt"); // открытие файла для записи

string line;

vector<int> countiens; // вектор для хранения длин строк из файла

int count = NULL;

while (getline(inputFile, line, '\n')) // чтение строки из файла до символа переноса строки

{

countiens.push\_back(line.size()); // добавление длины строки в вектор

count++;

}

stack.top = count; // установка количества элементов в стеке равным количеству строк в файле

stack.data = new int[stack.top]; // выделение памяти под стек

for (int i = 0; i < count; i++) // заполнение стека длинами строк из вектора

{

stack.data[i] = countiens[i];

}

countiens.clear(); // очистка вектора

cout << endl << "Результат: ";

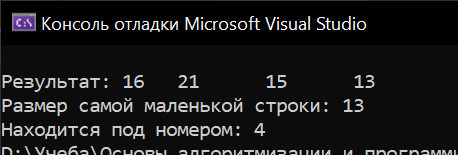
output(stack); // вывод элементов стека

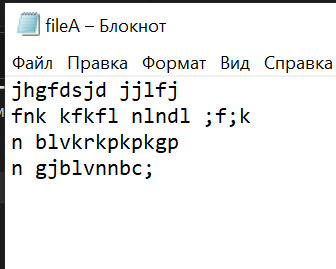
cout << endl << "Размер самой маленькой строки: ";

find\_min(stack); // поиск минимального элемента в стеке и его позиции

return 0;

}





Вариант 2



#include <iostream>

#include "stack.h"

using namespace std;

int main() {

Stack stack;

Stack stack1, stack2;

int choice;

do {

cout << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Peek" << endl;

cout << "4. Is empty?" << endl;

cout << "5. Clear" << endl;

cout << "6. Save to file" << endl;

cout << "7. Load from file" << endl;

cout << "8. Print from file" << endl;

cout << "9. Delete first negative" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

int data;

cout << "Enter data: ";

cin >> data;

stack.push(data);

break;

case 2:

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

break;

case 3:

cout << "Top of stack: " << stack.peek() << endl;

break;

case 4:

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

break;

case 5:

stack.clear();

cout << "Stack cleared" << endl;

break;

case 6:

stack.saveToFile();

cout << "Stack saved to file" << endl;

break;

case 7:

stack.loadFromFile();

cout << "Stack loaded from file" << endl;

break;

case 8:

stack.printFromFile();

break;

case 9:

cout << "Element has been delete!" << endl;

stack.deleteNegative();

stack.saveToFile();

stack.printFromFile();

break;

case 0:

cout << "Exiting program" << endl;

break;

default:

cout << "Invalid choice" << endl;

break;

}

} while (choice != 0);

}

// stack.cpp

#include <iostream>

#include <fstream>

#include "stack.h"

using namespace std;

Stack::Stack() {

head = nullptr; // Устанавливаем указатель на голову стека в значение нулевого указателя

}

Stack::~Stack() {

clear(); // Вызываем метод clear для освобождения памяти, выделенной под узлы стека

}

void Stack::push(int data) {

Node\* newNode = new Node; // Создаем новый узел стека

newNode->data = data; // Устанавливаем значение поля данных узла

newNode->next = head; // Устанавливаем указатель на следующий узел в цепочке на текущую голову стека

head = newNode; // Устанавливаем указатель на голову стека на только что созданный узел

}

int Stack::pop() {

if (isEmpty()) { // Проверяем, пуст ли стек

cerr << "Stack is empty" << endl; // Если стек пуст, выводим сообщение об ошибке

return 0; // Возвращаем нулевое значение

}

Node\* temp = head; // Создаем временный указатель на голову стека

int data = temp->data; // Сохраняем значение поля данных временного узла

head = temp->next; // Устанавливаем указатель на голову стека на следующий узел в цепочке

delete temp; // Освобождаем память, выделенную под временный узел

return data; // Возвращаем значение поля данных временного узла

}

int Stack::peek() {

if (isEmpty()) { // Проверяем, пуст ли стек

cerr << "Stack is empty" << endl; // Если стек пуст, выводим сообщение об ошибке

return 0; // Возвращаем нулевое значение

}

return head->data; // Возвращаем значение поля данных головного узла стека

}

bool Stack::isEmpty() {

return head == nullptr; // Проверяем, равен ли указатель на голову стека нулевому указателю

}

void Stack::clear() {

while (!isEmpty()) { // Пока стек не пуст

pop(); // Извлекаем узел из стека

}

}

void Stack::saveToFile() {

ofstream file("stack.txt"); // Открываем файл для записи

if (!file.is\_open()) { // Проверяем, удалось ли открыть файл

cerr << "Failed to open file for writing" << endl; // Если не удалось, выводим сообщение об ошибке

return; // Выходим из функции

}

Node\* current = head; // Создаем указатель на голову стека

while (current != nullptr) { // Пока указатель на текущий узел не является нулевым указателем

file << current->data << endl; // Записываем значение поля данных текущего узла в файл

current = current->next; // Устанавливает указатель

}

file.close();// Закрывает файл

}

void Stack::loadFromFile() {

ifstream file("stack.txt");// Открывает файл для чтения

if (!file.is\_open()) {// Проверка открытия

cerr << "Failed to open file for reading" << endl;

return;

}

clear();// Очищаем стек

int data;

while (file >> data) {// Считываются значения и добавляются в стек

push(data);

}

file.close();// Закрываем файл

}

void Stack::printFromFile() {

loadFromFile();// Функция загрузки

if (isEmpty()) {// Проверка

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

return;

}

cout << "Stack contents: ";

Node\* current = head;

while (current != nullptr) {// Перебираем все элементы стека

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void Stack::deleteNegative() {

Node\* current = head;//создаем указатель current на голову стека

Node\* prev = nullptr;//создаем указатель prev, который будет указывать на предыдущий узел в цепочке

while (current != nullptr) {//роверяем, является ли значение поля data текущего узла отрицательным

if (current->data < 0) {//проверяем, находится ли удаляемый элемент в начале стека

if (prev == nullptr) { // если удаляемый элемент находится в начале стека

head = current->next;//если удаляемый элемент находится в начале стека, то устанавливаем указатель на голову стека на следующий элемент после удаляемого

}

else {//если удаляемый элемент не находится в начале стека

prev->next = current->next;//устанавливаем указатель на следующий элемент после предыдущего узла так, чтобы он указывал на следующий после удаляемого

}

delete current;//удаляем текущий узел из памяти

return;

}

prev = current;

current = current->next;

}

}

// stack.h

struct Node {

int data;//данные, хранящиеся в узле

Node\* next;//указатель на следующий узел в списке

};

class Stack {

public:

Stack();//инициализирует пустой стек

~Stack();//удаляет все узлы в списке

void push(int data);//добавляет элемент на вершину стека

int pop();//удаляет и возвращает элемент с вершины стека

int peek();//возвращает элемент с вершины стека без удаления

bool isEmpty();//проверяет, пуст ли стек

void clear();//удаляет все элементы из стека

void saveToFile();//сохраняет содержимое стека в файле

void loadFromFile();//загружает содержимое стека из файла

void printFromFile();//выводит содержимое стека из файла в консоль

void deleteNegative();//удаляет первый отрицательный элемент

private:

Node\* head;

};

#pragma once

