МИНОБРНАУКИ РОССИИ

Федеральное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

(ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова»)

Факультет: «Программное обеспечение»

Кафедра: «Программное обеспечение»

Лабораторная работа №1

Выполнил:

Студент группы Б04-191-1з

Байков Даниил Олегович

Проверил:

Старший преподаватель

Еланцев М.О.

Ижевск, 2020

# Оглавление

[Оглавление 2](#_Toc39239316)

[1. ЗАДАНИЕ 3](#_Toc39239317)

[Реализовать структуру данных однонаправленный цикличный список: 3](#_Toc39239318)

[Реализовать структуру данных стек на базе массива: 3](#_Toc39239319)

[2. КОД ПРОГРАММЫ 4](#_Toc39239320)

[LabOne.cpp: 4](#_Toc39239321)

[3. ПРИМЕР РАБОТЫ 9](#_Toc39239322)

# ЗАДАНИЕ

## Реализовать структуру данных однонаправленный цикличный список:

Список должен содержать следующие операции:

* Вставка элемента в конец(целочисленное значение)

Вставка должна корректно обрабатывать случай, когда список пуст(указатель на первый узел пуст).

* Поиск элемента в списке

Поиск должен возвращать найденный узел списка, содержащий искомое значение

* Вставка значения после заданного узла списка
* Удаления элемента по значению

В случае, если элемент с ключом не найден, функция должна вернуть false, иначе - true

* Печать всех элементов списка
* Освобождение памяти от структуры данных

## Реализовать структуру данных стек на базе массива:

Структура данных должна содержать следующие операции:

* Инициализация (с указанием размер буфера под стек или очередь)
* Вставка элемента (целочисленное значение, вставка на вершину стека)
* Извлечение элемента (с вершины стека)

Операция должна возвращать код ошибки, в случае если в структуре данных нет элементов (например, функция возвращает false в случае, если нет элементов, и true - если элемент был извлечен, сам элемент в таком случае возвращается через параметр по ссылке)

* Клонирование (создание копии структуры данных со всеми заполненным на текущий момент элементами)
* Освобождение памяти от структуры данных

# КОД ПРОГРАММЫ

## LabOne.cpp:

// unidirectional circular linked list

// stack

#include <iostream>

using namespace std;

struct Data {

int value = 0;

Data\* next = NULL;

Data() {

}

Data(int value) {

this->value = value;

this->next = this;

}

Data(int value, Data\* next) {

this->value = value;

this->next = next;

}

};

struct LinkedList {

Data\* head = NULL;

void add(int value) { // Add to the end

if (head == NULL) {

Data\* newData = new Data(value);

head = newData;

return;

}

else {

Data\* point = head;

while (point->next != head) {

point = point->next;

}

Data\* newData = new Data(value, head);

point->next = newData;

return;

}

}

Data\* get(int value) { //returns pointer to Data with same value

Data\* point = head;

if (point == NULL) {

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

return NULL;

}

do {

if (point->value == value) return point;

point = point->next;

} while (point != head);

return NULL;

}

void add(int value, int afterValue) { //adds a new value after a certain

Data\* after = get(afterValue);

if (after == NULL) {

cout << "No value = " << afterValue << " in list. Can't add element after nothing." << endl;

return;

}

Data\* newData = new Data(value, after->next);

after->next = newData;

}

bool remove(int value) { // removes Data with same value

Data\* point = get(value);

if (point == NULL) return false;

Data\* first = head;

Data\* last = head;

while (last->next != head) {

last = last->next;

}

if (first->value == value) { // if try to delete first

if (first->next == head) {

head = NULL;

delete first;

return true;

}

else {

last->next = first->next;

head = first->next;

delete first;

return true;

}

}

else {

Data\* prev = head;

while (prev->next != point) {

prev = prev->next;

}

prev->next = point->next;

delete point;

return true;

}

}

void show() { // prints all elements of Linked List

Data\* point = head;

if (point == NULL) {

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

return;

}

do {

cout << point->value << " >> ";

point = point->next;

} while (point != head);

cout << "HEAD=" << head->value << endl;

}

void erase() { // removes all Data from mem

Data\* point = head->next;

Data\* toRemove;

head->next = NULL;

while (point) {

toRemove = point;

point = point->next;

delete toRemove;

}

head = NULL;

}

};

struct Stack {

int\* stack = NULL;

int head = 0;

int tail = 0;

int size = 0;

void init(int size) {

if (size < 1) {

if (stack) erase();

return;

}

stack = new int[size+1];

this->size = size+1;

this->head = 0;

this->tail = 0;

}

void push(int value) {

if (stack == NULL) {

cout << "Stack is not initialized." << endl;

return;

}

if ((tail + 1) % size == head) {

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

return;

}

stack[tail] = value;

tail++;

tail = tail % size;

}

bool pop(int\* buffer) {

if (stack == NULL) {

cout << "Stack is not initialized." << endl;

return false;

}

if (head == tail) {

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

return false;

}

\*buffer = stack[head];

head++;

head = head % size;

return true;

}

Stack\* clone() {

if (stack == NULL) {

cout << "Stack is not initialized." << endl;

return NULL;

}

Stack\* result = new Stack();

result->stack = new int[size];

result->size = size;

result->head = head;

result->tail = tail;

for (int i = 0; i < size; i++) {

result->stack[i] = stack[i];

}

return result;

}

void erase() {

if (stack == NULL) {

cout << "Stack is not initialized." << endl;

return;

}

head = 0;

tail = 0;

size = 0;

delete[] stack;

stack = NULL;

}

void show() {

if (stack == NULL) {

cout << "Stack is not initialized." << endl;

return;

}

if (head == tail) {

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

return;

}

for (int i = 0; i < size; i++) {

cout << stack[i];

if (i == head) cout << "(head)";

if (i == tail) cout << "(tail)";

cout << " ";

}

cout << endl;

}

};

void testStack() {

Stack\* s = new Stack();

int\* buffer = new int;

s->show();

s->init(5);

s->push(1);

s->push(2);

s->push(3);

s->push(4);

s->push(5);

s->show();

s->push(6);

s->push(7);

s->show();

s->pop(buffer);

s->pop(buffer);

s->pop(buffer);

s->show();

s->push(9);

s->push(10);

s->push(11);

s->push(12);

s->show();

s->erase();

s->init(2);

s->push(5);

s->show();

s->pop(buffer);

cout << \*buffer << endl;

s->pop(buffer);

cout << "------------ END OF TEST STACK -------------" << endl;

}

void testList() {

LinkedList l;

l.add(3);

l.add(13);

l.add(23);

l.add(33);

cout << "add 3,13,23,33" << endl;

l.show();

l.remove(3);

cout << "remove 3" << endl;

l.show();

cout << "add 5,6" << endl;

l.add(5);

l.add(6);

l.show();

cout << "remove 6" << endl;

l.remove(6);

l.show();

cout << "remove 33" << endl;

l.remove(33);

l.show();

cout << "add after 123123123" << endl;

l.add(50, 123123123);

cout << "add after 23" << endl;

l.add(99, 23);

l.show();

l.erase();

cout << "erase" << endl;

l.show();

l.add(10);

l.show();

cout << "---------------END OF TEST LIST---------------" << endl;

}

int main() {

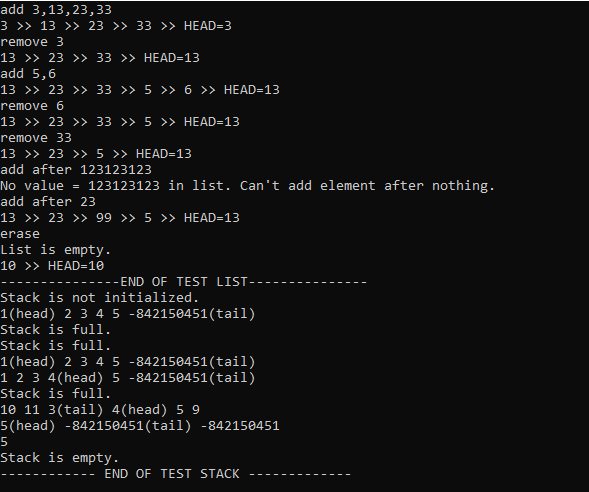
testList();

testStack();

return 0;

}

# ПРИМЕР РАБОТЫ



Рисунок