Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**Тема:** Лабораторная работа №11

Семестр: 2

Выполнил студент ИВТ-23-2б:

Синицын Владислав Сергеевич

(дата, подпись)

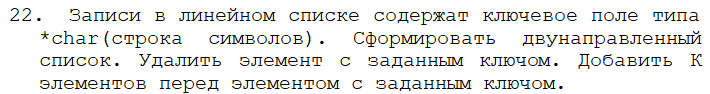
Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2024

Задание:



***Односвязный список***

Программное решение:

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* pointer\_to\_next\_node = nullptr;

};

struct List {

Node\* head\_node = nullptr;

Node\* tail\_node = nullptr;

};

void pushBack(List& list, const char& data) {

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == nullptr) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

} else {

list.tail\_node->pointer\_to\_next\_node = new\_node;

list.tail\_node = new\_node;

}

}

void del(List& list, int n, int size) // n - позиция удаляемого элемента

{

if ((list.head\_node != NULL) && (n < size) && (n >= 0)) // если по этому номеру что-то лежит и этот элемент внутри списка

{

Node\* temp = list.head\_node, \* helping = list.head\_node;

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

{

helping = temp; // предыдущее значение temp

temp = temp->pointer\_to\_next\_node;

}

if (temp == list.head\_node) // если элемент который надо удалить первый

{

list.head\_node = temp->pointer\_to\_next\_node;

}

else

{

helping->pointer\_to\_next\_node = temp->pointer\_to\_next\_node;

}

delete temp;

size--; // уменьшаем размер списка

return;

}

}

int main()

{

system("chcp 1251 > NULL");

int size = 12;

List list;

for (int i = 0; i != size; i++) { //заполнение массива

pushBack(list, i \* i);

}

Node\* current\_node = list.head\_node; //вывод

while (current\_node != nullptr) {

cout << current\_node->data << endl;

current\_node = current\_node->pointer\_to\_next\_node;

}

cout << endl << "Введите позицию элемента с которого хотите начать удаление "; //с какой позиции удаление

int delete\_ellement;

cin >> delete\_ellement;

cout << endl;

delete\_ellement--;

cout << "Введите кол-во элементов на удаление "; //сколько удалится

int quantity\_ellement;

cin >> quantity\_ellement;

cout << endl;

for (int i = 0; i < quantity\_ellement; i++) { //удаление элемента

del(list, delete\_ellement, size);

}

current\_node = list.head\_node; //вывод

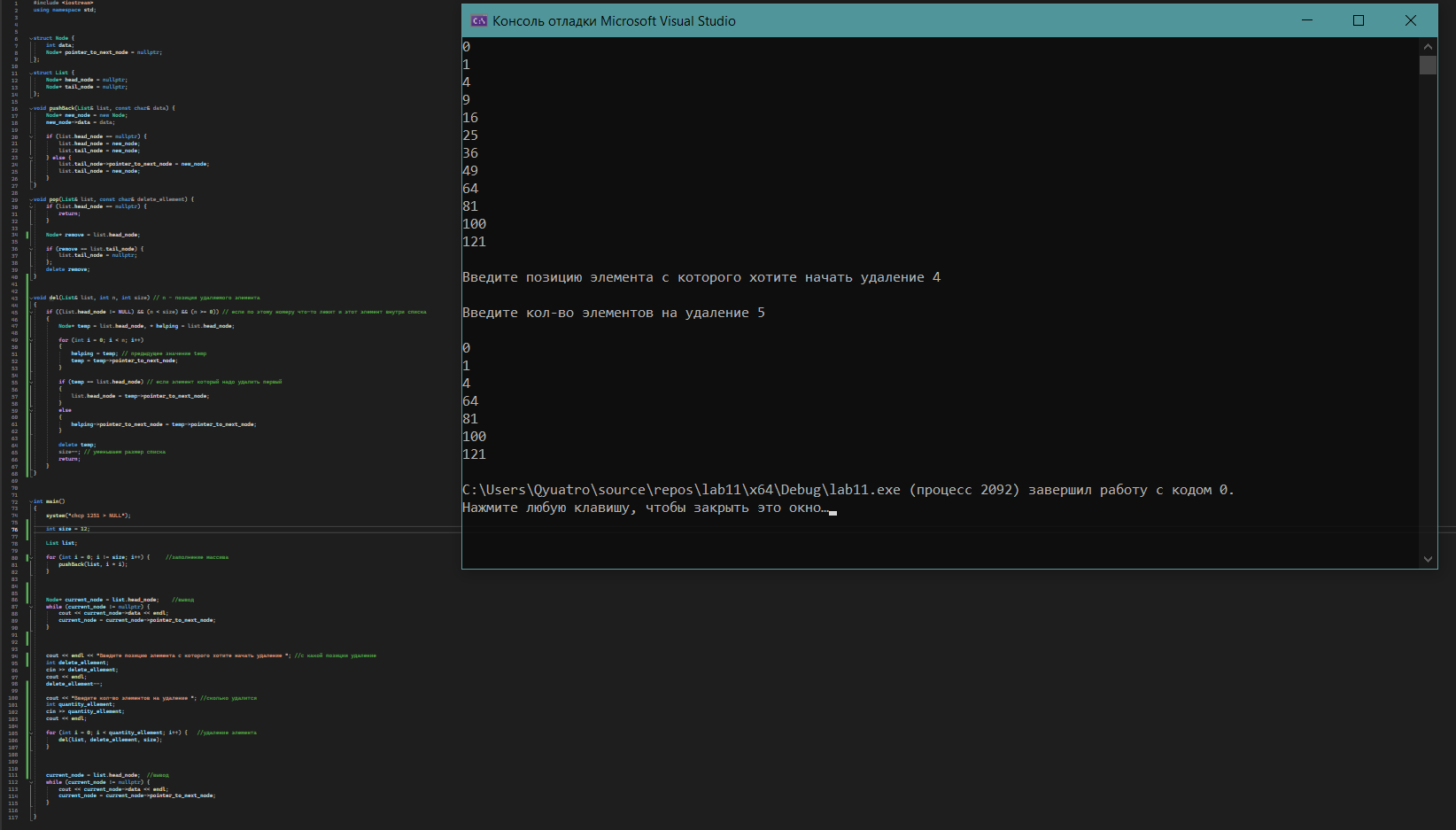
while (current\_node != nullptr) {

cout << current\_node->data << endl;

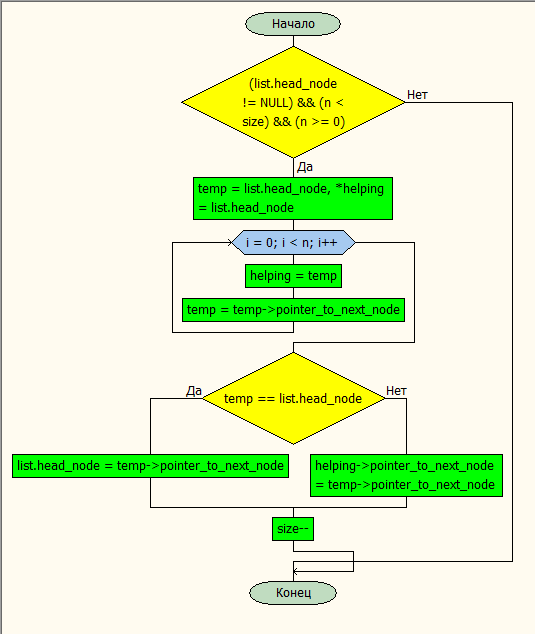
current\_node = current\_node->pointer\_to\_next\_node;

}

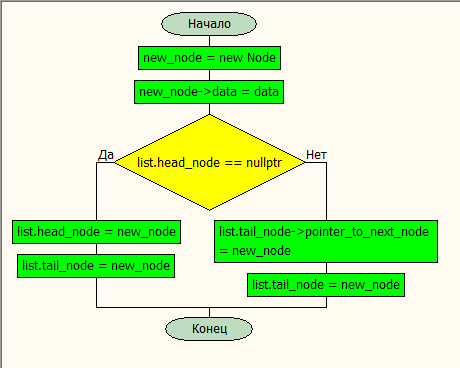
}



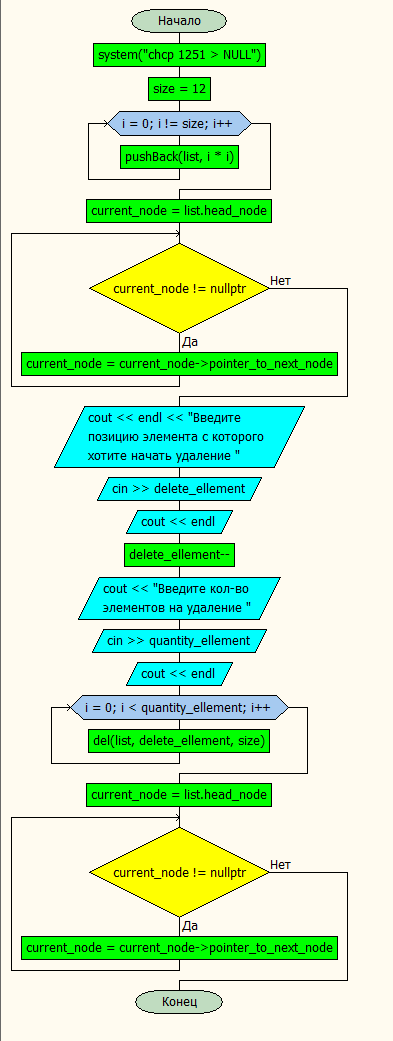
del ()



Push back



Main ()



***Двусвязный список***

Программное решение:

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* pointer\_to\_next\_node = nullptr;

Node\* pointer\_to\_prev\_node = nullptr;

};

struct List {

Node\* head\_node = nullptr;

Node\* tail\_node = nullptr;

};

void insert(List &list, const int& data, const int& index = 0) {

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == nullptr) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

return;

}

int counter = 0;

Node\* current\_node = list.head\_node;

while (counter != index) {

current\_node = current\_node->pointer\_to\_next\_node;

++counter;

}

new\_node->pointer\_to\_prev\_node = current\_node;

if (current\_node->pointer\_to\_next\_node != nullptr) {

new\_node->pointer\_to\_next\_node = current\_node->pointer\_to\_next\_node;

current\_node->pointer\_to\_next\_node->pointer\_to\_prev\_node = new\_node;

}

current\_node->pointer\_to\_next\_node = new\_node;

list.tail\_node = new\_node;

}

void Remove(List& list, int idx, int size)

{

Node\* current\_node = list.head\_node;

Node\* temp = nullptr;

int i = 0;

while (current\_node->pointer\_to\_next\_node != NULL)

{

if (i != idx)

{

current\_node = current\_node->pointer\_to\_next\_node;

i++;

}

else

{

if (idx == 1) {

temp = list.head\_node;

list.head\_node = current\_node;

}

else {

temp = current\_node->pointer\_to\_prev\_node; // указывает на узель, который должен удаляться

current\_node->pointer\_to\_prev\_node = temp->pointer\_to\_prev\_node;

current\_node->pointer\_to\_prev\_node->pointer\_to\_next\_node = current\_node;

}

delete temp;

size--;

return;

}

}

}

int main()

{

system("chcp 1251 > NULL");

int size = 5;

List list;

Node\* current\_node = list.head\_node;

for (int i = size; i != 0; i--) { //заполнение массива

insert(list, i);

}

current\_node = list.head\_node; //вывод

while (current\_node != nullptr) {

cout << current\_node->data << endl;

current\_node = current\_node->pointer\_to\_next\_node;

}

cout << endl << "Введите позицию элемента с которого хотите начать удаление "; //с какой позиции удаление

int delete\_ellement;

cin >> delete\_ellement;

cout << endl;

cout << "Введите кол-во элементов на удаление "; //сколько удалится

int quantity\_ellement;

cin >> quantity\_ellement;

cout << endl;

for (int i = 0; i < quantity\_ellement; i++) { //удаление элемента

Remove(list, delete\_ellement, size);

}

current\_node = list.head\_node; //вывод

while (current\_node != nullptr) {

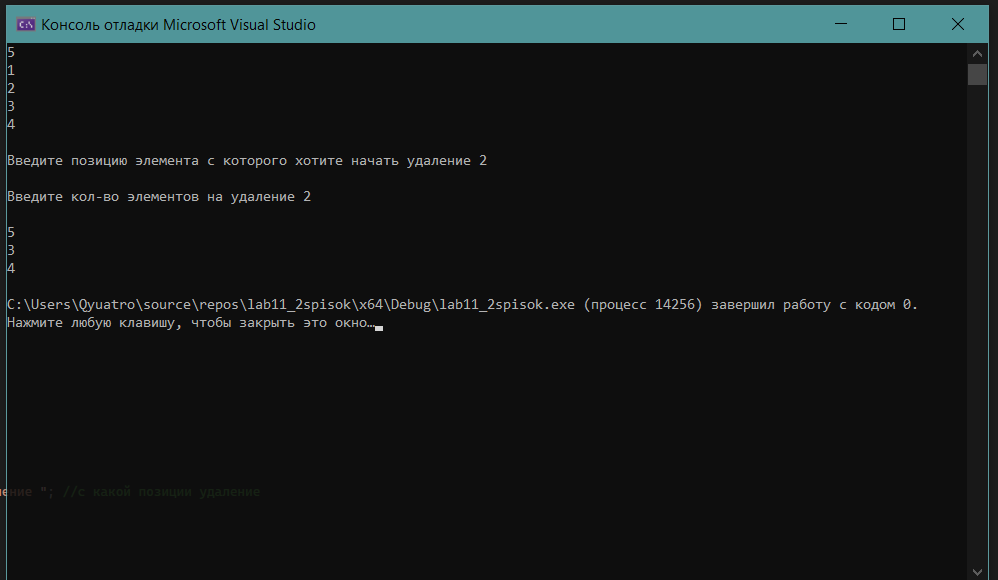
cout << current\_node->data << endl;

current\_node = current\_node->pointer\_to\_next\_node;

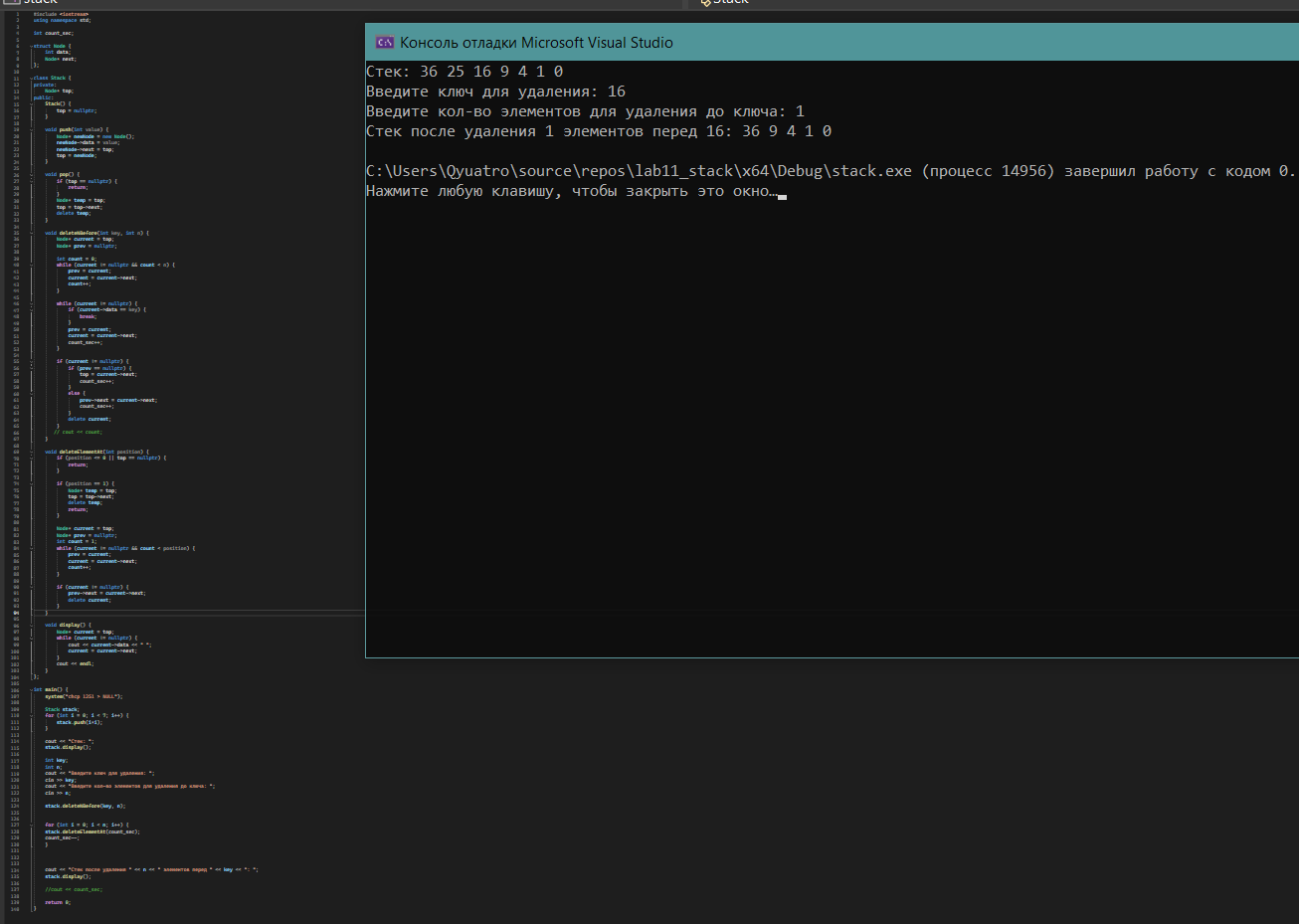
}

return 0;

}



Стек:



#include <iostream>

using namespace std;

int count\_sec;

struct Node {

int data;

Node\* next;

};

class Stack {

private:

Node\* top;

public:

Stack() {

top = nullptr;

}

void push(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = top;

top = newNode;

}

void pop() {

if (top == nullptr) {

return;

}

Node\* temp = top;

top = top->next;

delete temp;

}

void deleteNBefore(int key, int n) {

Node\* current = top;

Node\* prev = nullptr;

int count = 0;

while (current != nullptr && count < n) {

prev = current;

current = current->next;

count++;

}

while (current != nullptr) {

if (current->data == key) {

break;

}

prev = current;

current = current->next;

count\_sec++;

}

if (current != nullptr) {

if (prev == nullptr) {

top = current->next;

count\_sec++;

}

else {

prev->next = current->next;

count\_sec++;

}

delete current;

}

// cout << count;

}

void deleteElementAt(int position) {

if (position <= 0 || top == nullptr) {

return;

}

if (position == 1) {

Node\* temp = top;

top = top->next;

delete temp;

return;

}

Node\* current = top;

Node\* prev = nullptr;

int count = 1;

while (current != nullptr && count < position) {

prev = current;

current = current->next;

count++;

}

if (current != nullptr) {

prev->next = current->next;

delete current;

}

}

void display() {

Node\* current = top;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

};

int main() {

system("chcp 1251 > NULL");

Stack stack;

for (int i = 0; i < 7; i++) {

stack.push(i\*i);

}

cout << "Стек: ";

stack.display();

int key;

int n;

cout << "Введите ключ для удаления: ";

cin >> key;

cout << "Введите кол-во элементов для удаления до ключа: ";

cin >> n;

stack.deleteNBefore(key, n);

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

stack.deleteElementAt(count\_sec);

count\_sec--;

}

cout << "Стек после удаления " << n << " элементов перед " << key << ": ";

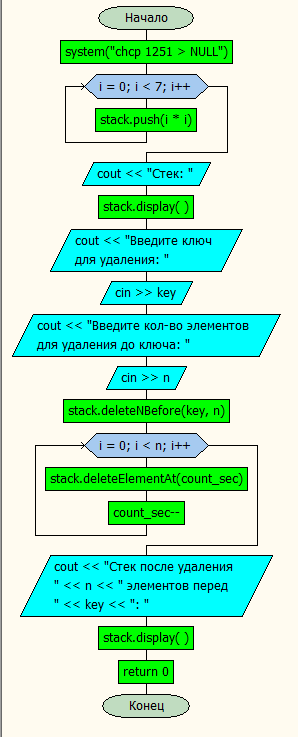
stack.display();

//cout << count\_sec;

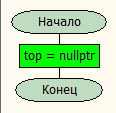
return 0;

}

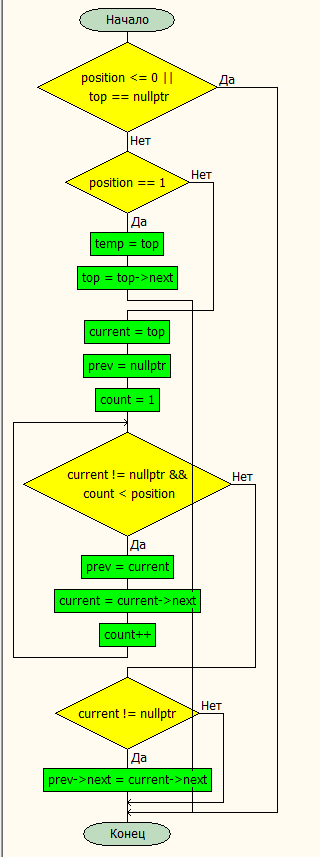
Main



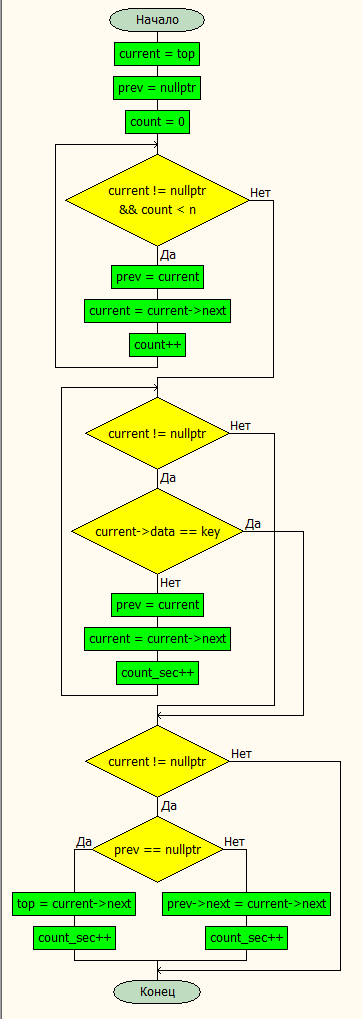
stack



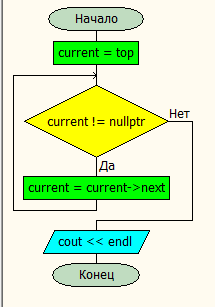
Delete ellement



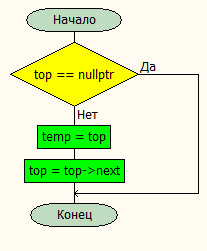
Delete before



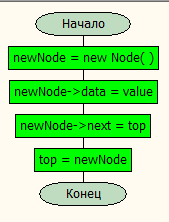
Display



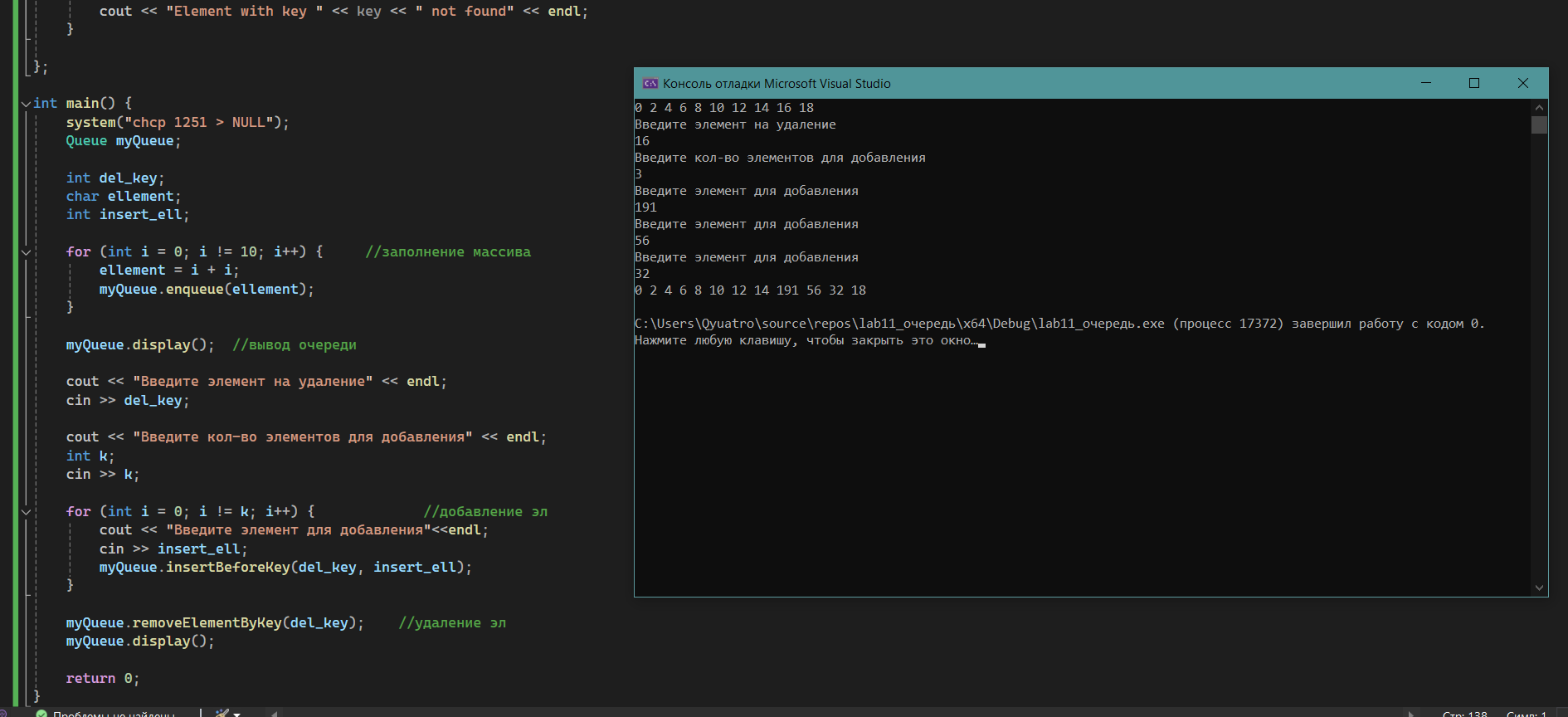
Pop



Push



***Очередь***



#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = nullptr;

rear = nullptr;

}

void enqueue(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (rear) {

rear->next = newNode;

}

rear = newNode;

if (front == nullptr) {

front = rear;

}

}

void insertBeforeKey(int key, int value) {

Node\* newNode = new Node;

newNode->data = value;

if (front == nullptr) {

cout << "Queue is empty" << endl;

return;

}

Node\* current = front;

Node\* prev = nullptr;

while (current != nullptr) {

if (current->data == key) {

if (prev == nullptr) {

newNode->next = front;

front = newNode;

}

else {

newNode->next = current;

prev->next = newNode;

}

return;

}

prev = current;

current = current->next;

}

cout << "Element with key " << key << " not found" << endl;

}

void display() {

Node\* current = front;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

void removeElementByKey(int key) {

Node\* current = front;

Node\* prev = nullptr;

while (current != nullptr) {

if (current->data == key) {

if (prev == nullptr) {

front = current->next;

}

else {

prev->next = current->next;

}

delete current;

return;

}

prev = current;

current = current->next;

}

cout << "Element with key " << key << " not found" << endl;

}

};

int main() {

system("chcp 1251 > NULL");

Queue myQueue;

int del\_key;

char ellement;

int insert\_ell;

for (int i = 0; i != 10; i++) { //заполнение массива

ellement = i + i;

myQueue.enqueue(ellement);

}

myQueue.display(); //вывод очереди

cout << "Введите элемент на удаление" << endl;

cin >> del\_key;

cout << "Введите кол-во элементов для добавления" << endl;

int k;

cin >> k;

for (int i = 0; i != k; i++) { //добавление эл

cout << "Введите элемент для добавления"<<endl;

cin >> insert\_ell;

myQueue.insertBeforeKey(del\_key, insert\_ell);

}

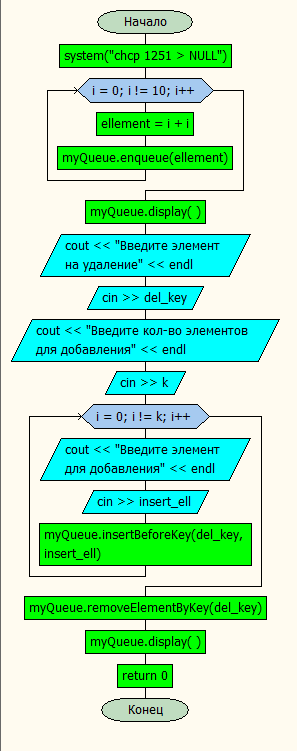
myQueue.removeElementByKey(del\_key); //удаление эл

myQueue.display();

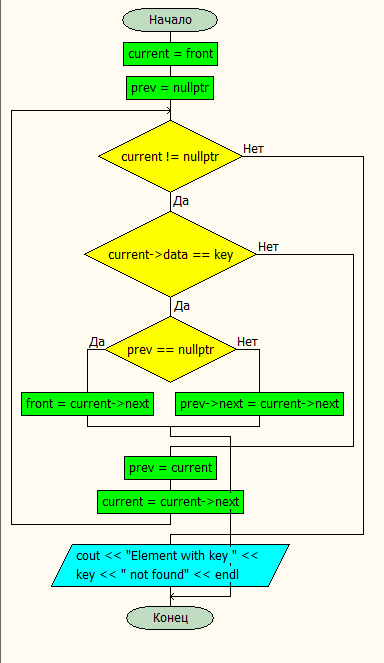
return 0;

}

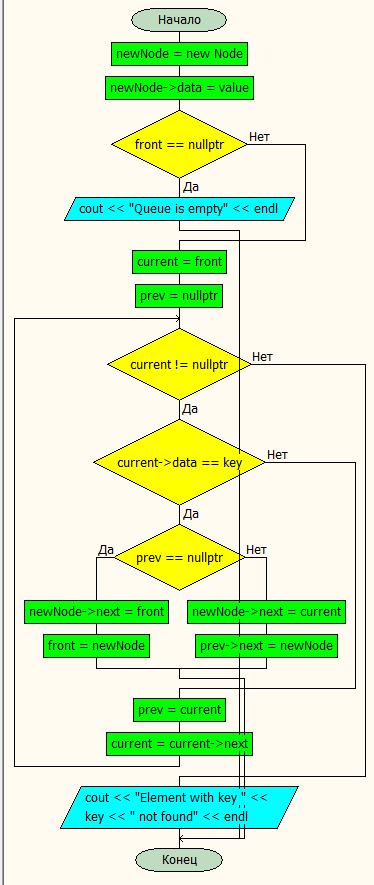
Main



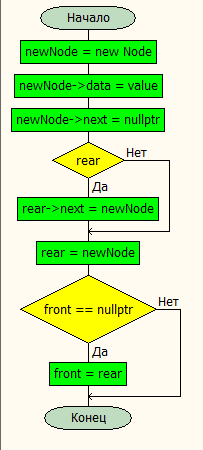
Remove ellement by key



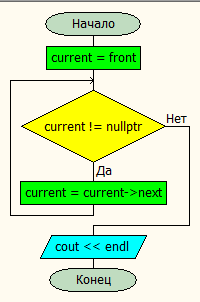
Insert



Enqueue



Display



queue

