Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Тема: «Реализация контейнеров через структуры.»

Выполнил

Студент группы РИС-22-2б

Чиртулов М.В.

Проверил доц. кафедры ИТАС

Полякова О.А.

Пермь 2023

## Постановка задачи

1)Реализовать односвязный список через структуры.

2)Реализовать двусвязный список через структуры.

3)Реализовать стек через структуры.

4)Реализовать очередь через структуры.

## 1 Односвязный список

**Программный код:**

*#include <iostream>*

*using namespace std;*

*struct Node {*

*string val;*

*Node\* next;*

*Node(string \_val) : val(\_val), next(nullptr) {}*

*};*

*struct list {*

*Node\* first;*

*Node\* last;*

*list() : first(nullptr), last(nullptr) {}*

*bool is\_empty() {*

*return first == nullptr;*

*}*

*void push\_back(string \_val) {*

*Node\* p = new Node(\_val);*

*if (is\_empty()) {*

*first = p;*

*last = p;*

*return;*

*}*

*last->next = p;*

*last = p;*

*}*

*void print() {*

*if (is\_empty()) return;*

*Node\* p = first;*

*while (p) {*

*cout << p->val << " ";*

*p = p->next;*

*}*

*cout << endl;*

*}*

*Node\* find(string \_val) {*

*Node\* p = first;*

*while (p && p->val != \_val) p = p->next;*

*return (p && p->val == \_val) ? p : nullptr;*

*}*

*void remove\_first() {*

*if (is\_empty()) return;*

*Node\* p = first;*

*first = p->next;*

*delete p;*

*}*

*void remove\_last() {*

*if (is\_empty()) return;*

*if (first == last) {*

*remove\_first();*

*return;*

*}*

*Node\* p = first;*

*while (p->next != last) p = p->next;*

*p->next = nullptr;*

*delete last;*

*last = p;*

*}*

*void remove(string \_val) {*

*if (is\_empty()) return;*

*if (first->val == \_val) {*

*remove\_first();*

*return;*

*}*

*else if (last->val == \_val) {*

*remove\_last();*

*return;*

*}*

*Node\* slow = first;*

*Node\* fast = first->next;*

*while (fast && fast->val != \_val) {*

*fast = fast->next;*

*slow = slow->next;*

*}*

*if (!fast) {*

*cout << "This element does not exist" << endl;*

*return;*

*}*

*slow->next = fast->next;*

*delete fast;*

*}*

*Node\* operator[] (const int index) {*

*if (is\_empty()) return nullptr;*

*Node\* p = first;*

*for (int i = 0; i < index; i++) {*

*p = p->next;*

*if (!p) return nullptr;*

*}*

*return p;*

*}*

*};*

*int main()*

*{*

*//Работа с односвязным списком*

*list l;*

*cout << l.is\_empty() << endl;*

*l.push\_back("3");*

*l.push\_back("123");*

*l.push\_back("8");*

*l.print();*

*cout << l.is\_empty() << endl;*

*l.remove("123");*

*l.print();*

*l.push\_back("1234");*

*l.remove\_first();*

*l.print();*

*l.remove\_last();*

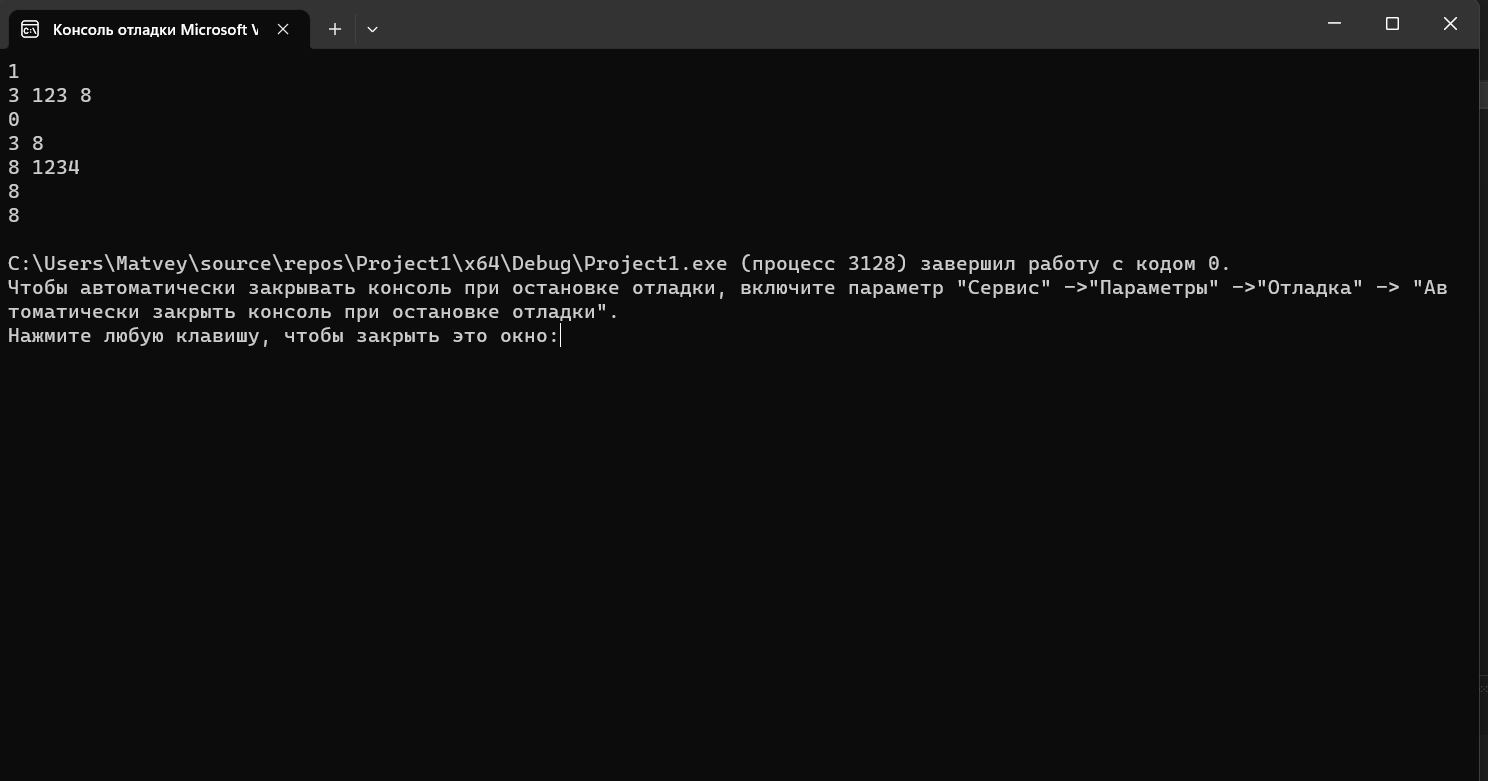
*l.print();*

*cout << l[0]->val << endl;*

*return 0;*

*}*

## Результаты работы программы



## Блок-схема

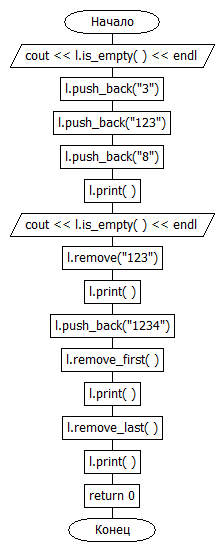


Рисунок 1 – структура list

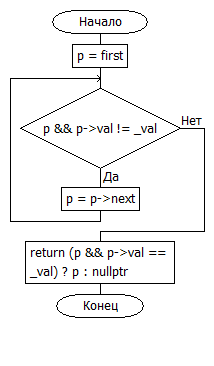


Рисунок 2 – функция find

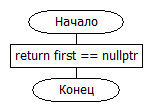


Рисунок 3 – функция is\_empty

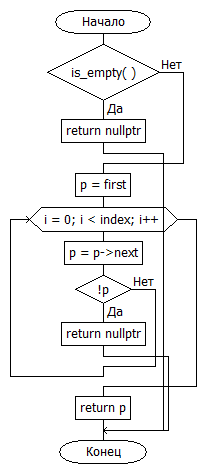


Рисунок 4 – функция operator

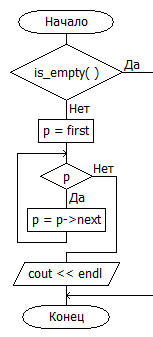


Рисунок 5 – функция print

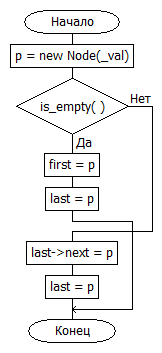


Рисунок 6 – функция push\_back

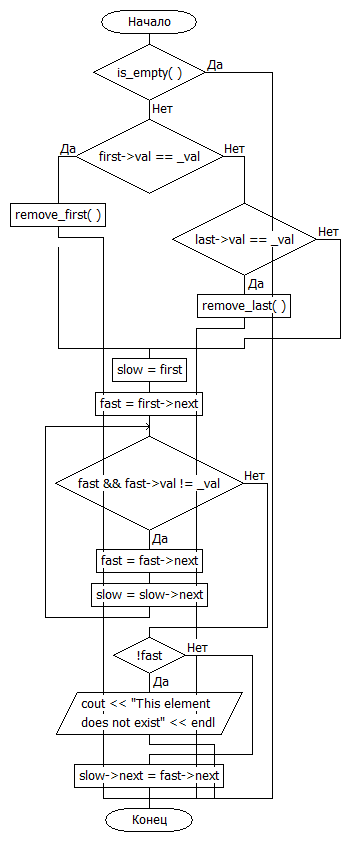


Рисунок 7 – функция remove

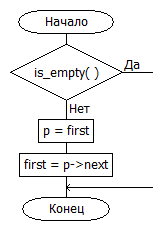


Рисунок 8 – функция remove\_last

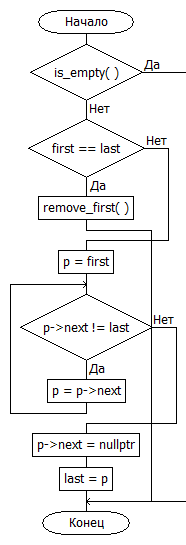


Рисунок 9 – функция remove\_first

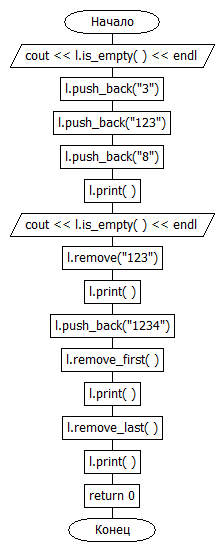


Рисунок 10 – функция main

## 2. Двусвязный список

**Программный код:**

*#include <iostream>*

*#include <string>*

*using namespace std;*

*struct list*

*{*

*int field; // поле данных*

*struct list\* next; // указатель на следующий элемент*

*struct list\* prev; // указатель на предыдущий элемент*

*};*

*struct list\* init(int a) // а- значение первого узла*

*{*

*struct list\* lst;*

*// выделение памяти под корень списка*

*lst = (struct list\*)malloc(sizeof(struct list));*

*lst->field = a;*

*lst->next = NULL; // указатель на следующий узел*

*lst->prev = NULL; // указатель на предыдущий узел*

*return(lst);*

*}*

*struct list\* addelem(list\* lst, int number)*

*{*

*struct list\* temp, \* p;*

*temp = (struct list\*)malloc(sizeof(list));*

*p = lst->next; // сохранение указателя на следующий узел*

*lst->next = temp; // предыдущий узел указывает на создаваемый*

*temp->field = number; // сохранение поля данных добавляемого узла*

*temp->next = p; // созданный узел указывает на следующий узел*

*temp->prev = lst; // созданный узел указывает на предыдущий узел*

*if (p != NULL)*

*p->prev = temp;*

*return(temp);*

*}*

*struct list\* deletelem(list\* lst)*

*{*

*struct list\* prev, \* next;*

*prev = lst->prev; // узел, предшествующий lst*

*next = lst->next; // узел, следующий за lst*

*if (prev != NULL)*

*prev->next = lst->next; // переставляем указатель*

*if (next != NULL)*

*next->prev = lst->prev; // переставляем указатель*

*free(lst); // освобождаем память удаляемого элемента*

*return(prev);*

*}*

*void listprint(list\* lst)*

*{*

*struct list\* p;*

*p = lst;*

*do {*

*cout << p->field << " "; // вывод значения элемента p*

*p = p->next; // переход к следующему узлу*

*} while (p != NULL); // условие окончания обхода*

*}*

*struct list\* swap(struct list\* lst1, struct list\* lst2, struct list\* head)*

*{*

*// Возвращает новый корень списка*

*struct list\* prev1, \* prev2, \* next1, \* next2;*

*prev1 = lst1->prev; // узел предшествующий lst1*

*prev2 = lst2->prev; // узел предшествующий lst2*

*next1 = lst1->next; // узел следующий за lst1*

*next2 = lst2->next; // узел следующий за lst2*

*if (lst2 == next1) // обмениваются соседние узлы*

*{*

*lst2->next = lst1;*

*lst2->prev = prev1;*

*lst1->next = next2;*

*lst1->prev = lst2;*

*if (next2 != NULL)*

*next2->prev = lst1;*

*if (lst1 != head)*

*prev1->next = lst2;*

*}*

*else if (lst1 == next2) // обмениваются соседние узлы*

*{*

*lst1->next = lst2;*

*lst1->prev = prev2;*

*lst2->next = next1;*

*lst2->prev = lst1;*

*if (next1 != NULL)*

*next1->prev = lst2;*

*if (lst2 != head)*

*prev2->next = lst1;*

*}*

*else // обмениваются отстоящие узлы*

*{*

*if (lst1 != head) // указатель prev можно установить только для элемента,*

*prev1->next = lst2; // не являющегося корневым*

*lst2->next = next1;*

*if (lst2 != head)*

*prev2->next = lst1;*

*lst1->next = next2;*

*lst2->prev = prev1;*

*if (next2 != NULL) // указатель next можно установить только для элемента,*

*next2->prev = lst1; // не являющегося последним*

*lst1->prev = prev2;*

*if (next1 != NULL)*

*next1->prev = lst2;*

*}*

*if (lst1 == head)*

*return(lst2);*

*if (lst2 == head)*

*return(lst1);*

*return(head);*

*}*

*struct list\* deletehead(list\* root)*

*{*

*struct list\* temp;*

*temp = root->next;*

*if (temp != NULL)*

*temp->prev = NULL;*

*free(root); // освобождение памяти текущего корня*

*return temp; // новый корень списка*

*}*

*int main() {*

*list a;*

*list\* p = &a;*

*p = init(9);*

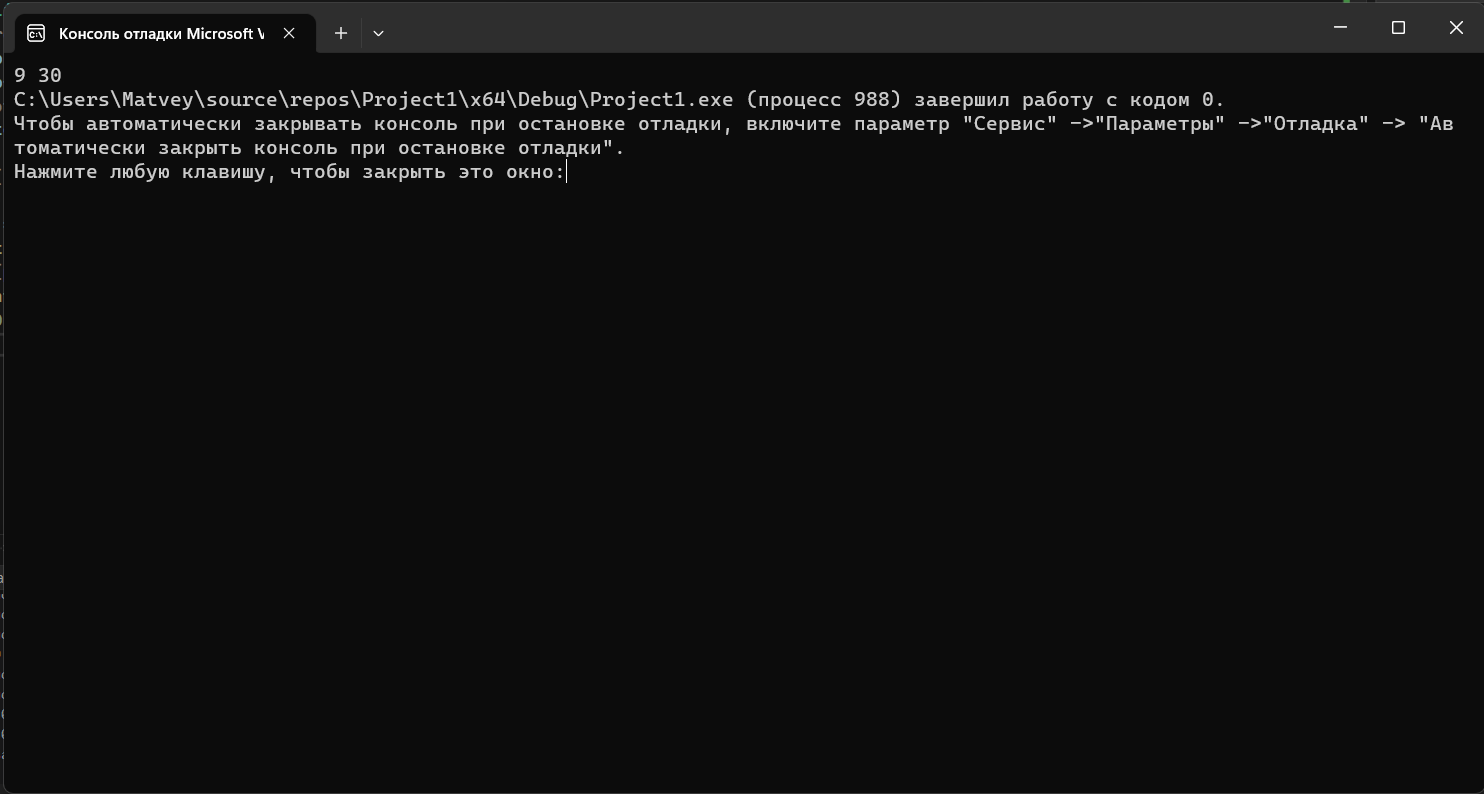
*addelem(p, 30);*

*listprint(p);*

*return 0;*

*}*

## Результаты работы программы



## Блок-схема

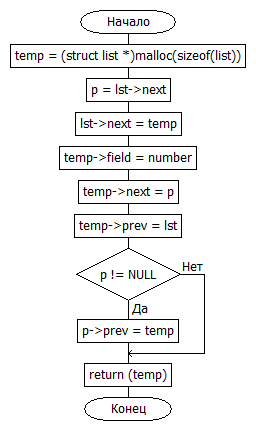


Рисунок 1 – функция addelem

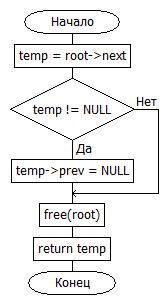


Рисунок 2 – функция deletehead

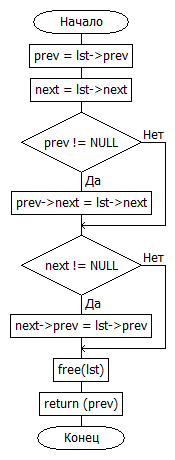


Рисунок 3 – функция deletelem

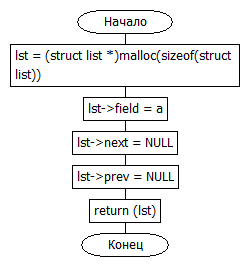


Рисунок 4 – функция init

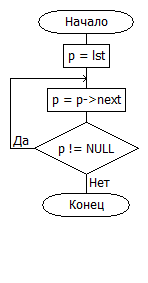


Рисунок 5 – функция print

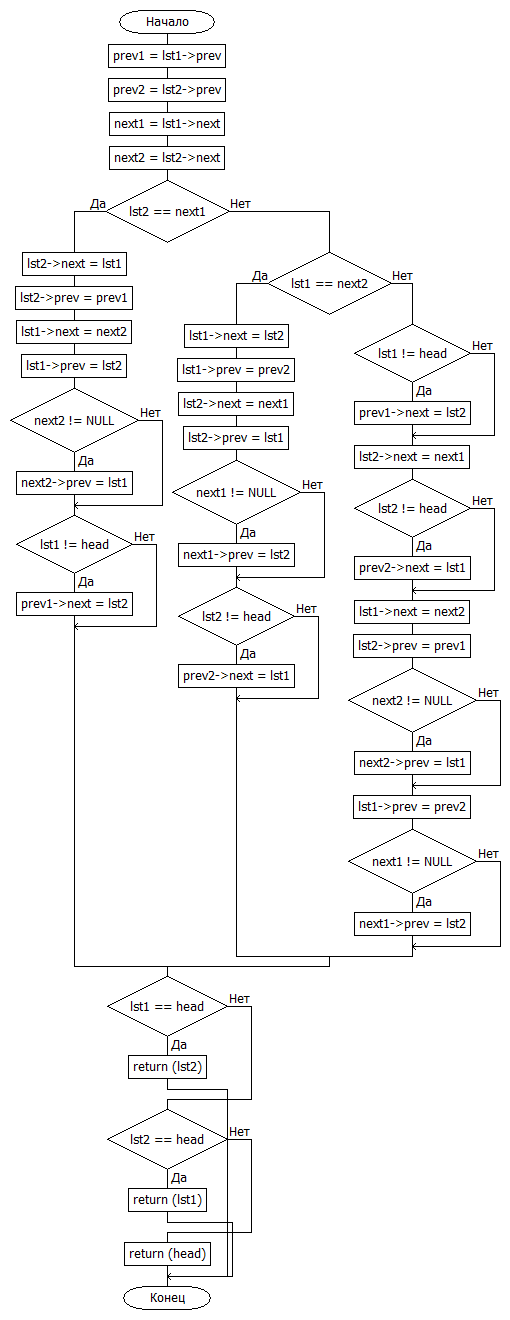


Рисунок 6 – функция swap

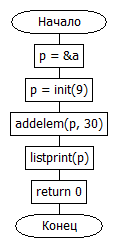


Рисунок 7 – функция main

## 3. Стек

***Программный код:***

*#include <iostream>*

*#include <stack> // подключаем библиотеку для*

*// использования стека*

*using namespace std;*

*int main() {*

*setlocale(LC\_ALL, "rus");*

*stack <int> steck; // создаем стек*

*int i = 0;*

*cout << "Введите шесть любых целых чисел: " << endl; // предлагаем пользователю*

*// ввести 6 чисел*

*while (i != 6) {*

*int a;*

*cin >> a;*

*steck.push(a); // добавляем введенные числа*

*i++;*

*}*

*if (steck.empty()) cout << "Стек не пуст"; // проверяем пуст ли стек (нет)*

*cout << "Верхний элемент стека: " << steck.top() << endl; // выводим верхний элемент*

*cout << "Удаляем верхний элемент " << endl;*

*steck.pop(); // удаляем верхний элемент*

*cout << "Новый верхний элемент: " << steck.top() << endl; // выводим уже новый*

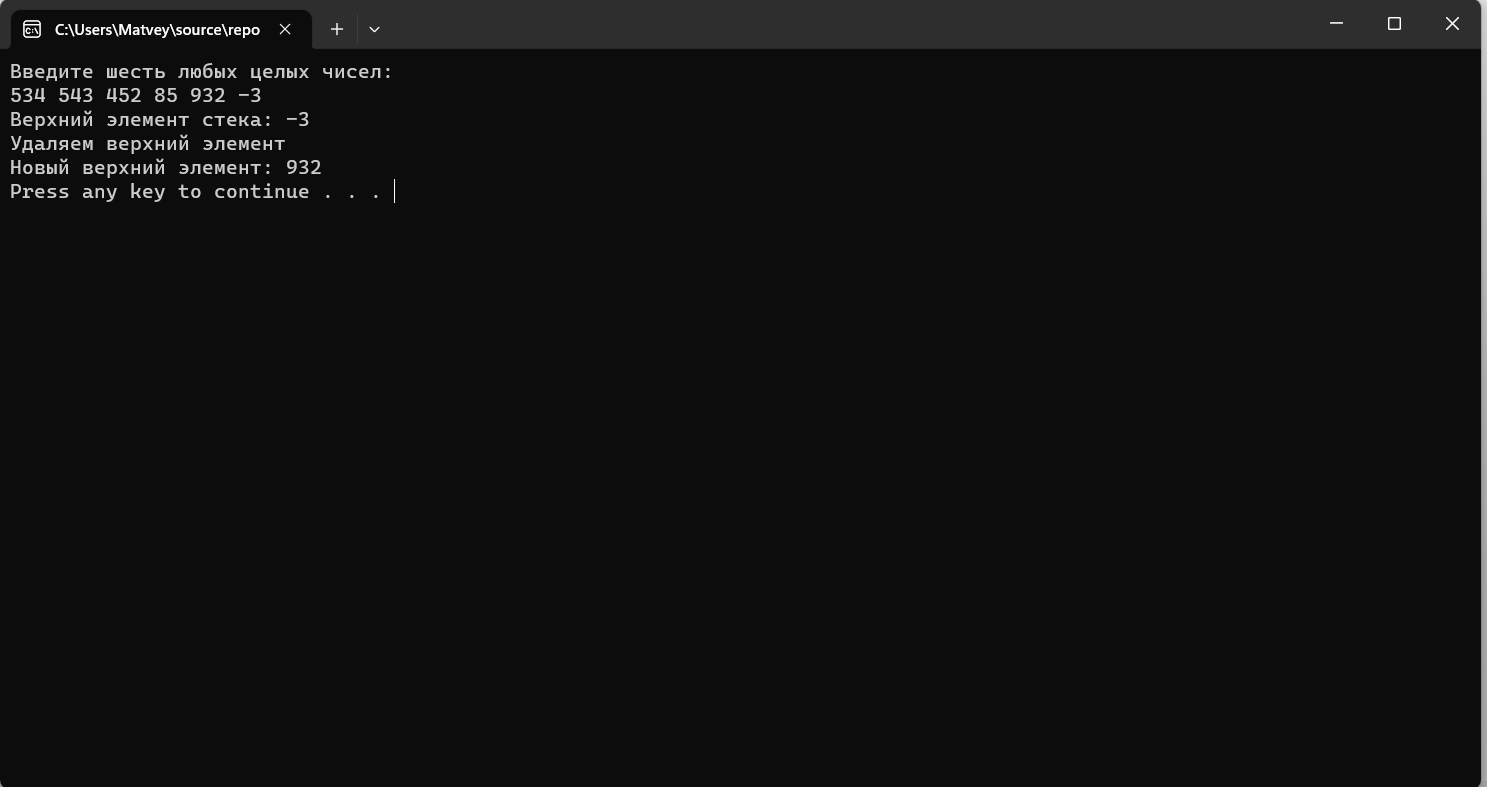
*// верхний элемент*

*system("pause");*

*return 0;*

*}*

## Результаты работы программы

****

## Блок – схема

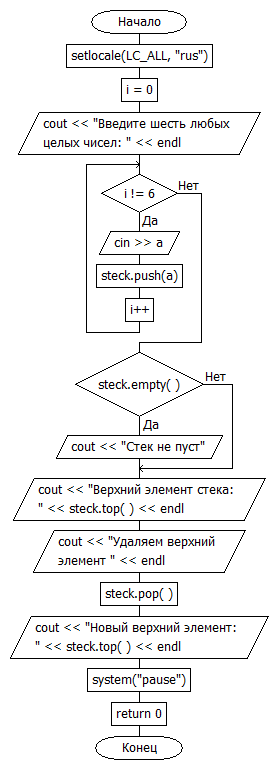


Рисунок 1 - функция main

## 4. Очередь

**Программный код:**

*#include <iostream>*

*using namespace std;*

*struct node {*

*node\* next;*

*int info;*

*};*

*void push(node\*& next, node\*& End) {*

*node\* el = new node;*

*cout << "Введите элемент очереди:";*

*cin >> el->info;*

*el->next = NULL;*

*End->next = el;*

*End = el;*

*}*

*void pop(node\*& el, node\*& Begin) {*

*el = Begin;*

*Begin = Begin->next;*

*delete el;*

*}*

*void peek(node\*& el) {*

*cout << "Значение в начале очереди: ";*

*cout << el->info << endl;*

*}*

*void queue\_List(node\*& Begin) {*

*node\* current\_element = Begin;*

*cout << "Элементы очереди: ";*

*while (current\_element) {*

*cout << current\_element->info << " ";*

*current\_element = current\_element->next;*

*}*

*cout << endl;*

*}*

*void isEmpty(node\*& queue) {*

*if (queue == NULL)*

*cout << "Очередь пуста!" << endl;*

*else*

*cout << "Очередь имеет элементы" << endl;*

*}*

*int main()*

*{*

*setlocale(LC\_ALL, "Russian");*

*int k = 1;*

*cout << "Создание очереди. Пожалуйста, введите первый элемент очереди:" << endl;*

*node\* Begin, \* End;*

*node\* el = new node;*

*cin >> el->info;*

*el->next = NULL;*

*Begin = End = el;*

*while (k) {*

*cout << "1 - добавление элемента в конец очереди\n2 - удаление элемента из начала очереди\n3 - функция определения значения в начале очереди\n4 - функция вывода элементов очереди на экран\n5 - функция определения пуста очередь или нет\n" << "0 - выход" << endl;*

*cin >> k;*

*if (k == 0)*

*return 0;*

*if (k == 1)*

*push(el, End);*

*if (k == 2)*

*pop(el, Begin);*

*if (k == 3)*

*peek(Begin);*

*if (k == 4)*

*queue\_List(Begin);*

*if (k == 5)*

*isEmpty(el);*

*}*

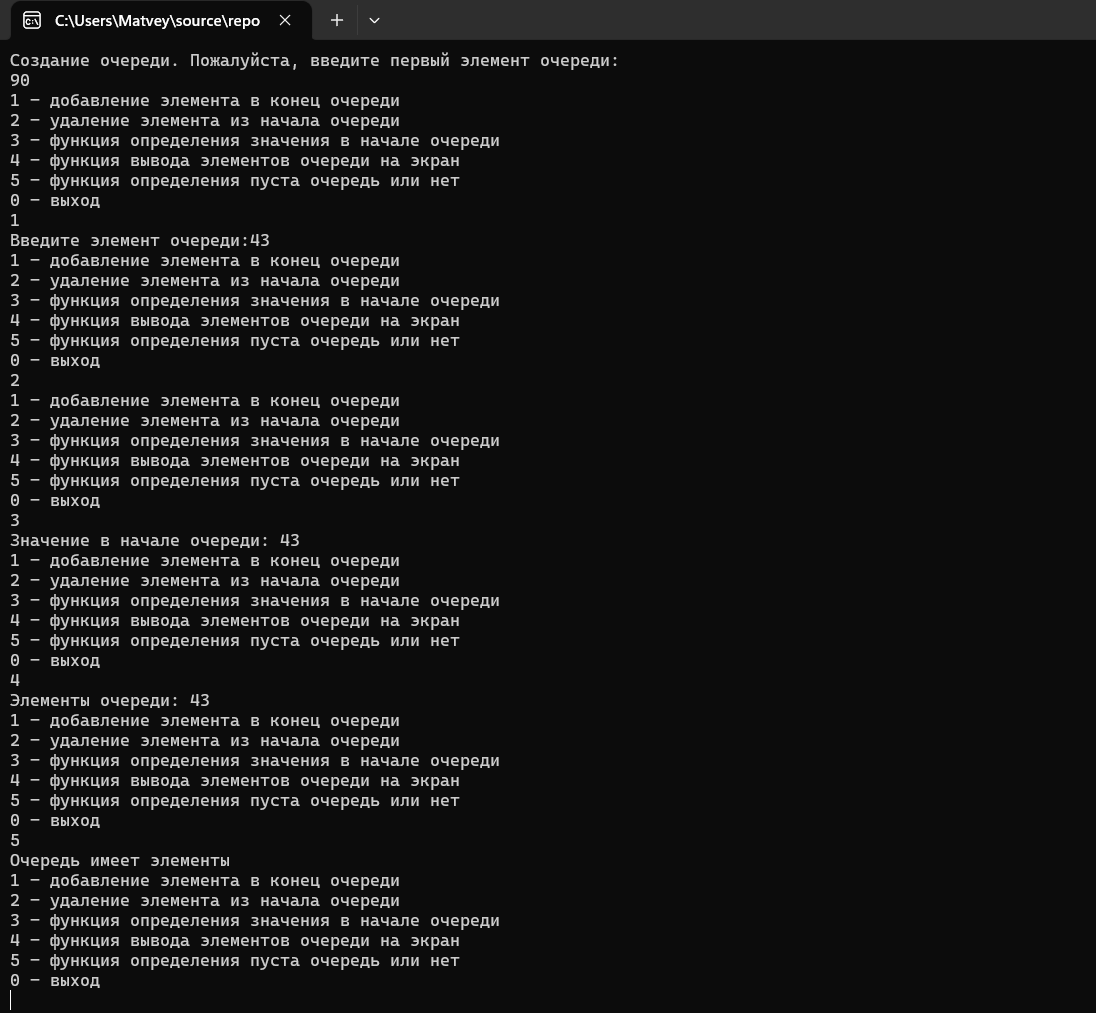
*cout << endl;*

*system("pause");*

*return 0;*

*}*

## Результаты работы программы



## Блок – схема

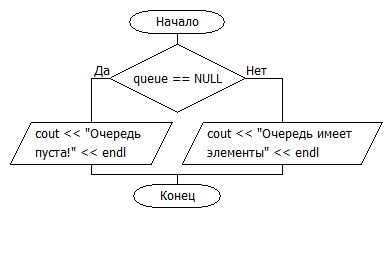


Рисунок 1 – функция isEmpty

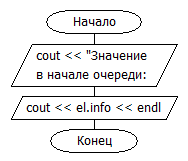


Рисунок 2 - функция peek

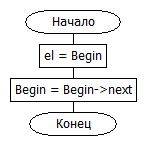


Рисунок 3 - функция pop

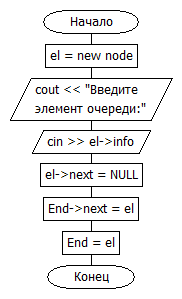


Рисунок 4 - функция push

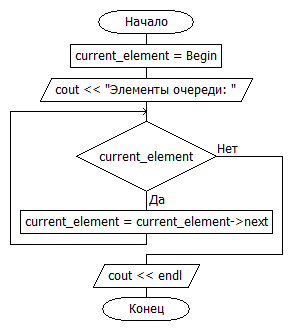


Рисунок 5 - функция queue\_List

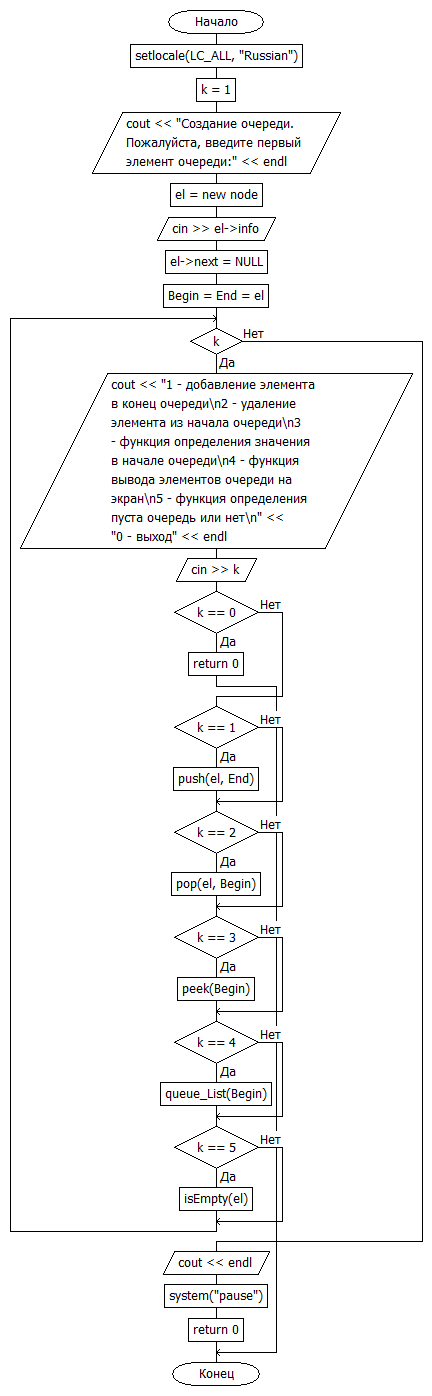


Рисунок 6 - функция main