Министерство науки и высшего образования Российской Федерации федеральное государственное бюджетное образовательное учреждение высшего образования  
Московский авиационный институт  
(Национальный исследовательский университет)

Институт №3  
«Системы управления, информатика и электроэнергетика»  
Кафедра 304

**Отчёт по лабораторной работе**  
  
по учебной дисциплине «Программирование»   
на тему «Поиск кратчайших путей в графе. Построение остовного дерева графа»

Исполнитель:  
обучающиеся группы М3О-207Б-23  
Ильин А.А.  
Романов Д.И.

Проверили:  
Дмитриева Е.А.

Ивашенцев И. В.

Москва 2024

Оглавление

[Задание 3](#_Toc182143329)

[Блок схема 4](#_Toc182143330)

[Код программы 15](#_Toc182143331)

[Результаты 25](#_Toc182143332)

[Вывод 27](#_Toc182143333)

## Задание

**Задание**

Для взвешенного ориентированного графа, состоящего как минимум из 10 вершин, реализовать по вариантам:

1. алгоритм поиска кратчайшего пути;
2. сделав тот же самый граф неориентированным, построить его остовное дерево минимальной стоимости.

Граф для первого пункта задания придумывается самостоятельно (не нужно связывать каждую вершину со всеми остальными, не делайте граф полным).

Матрицу смежности ( значения весов каждого ребра) лучше определить в начале программы, не вводить значения с клавиатуры или из файла.

**Должны быть представлены промежуточные результаты.**

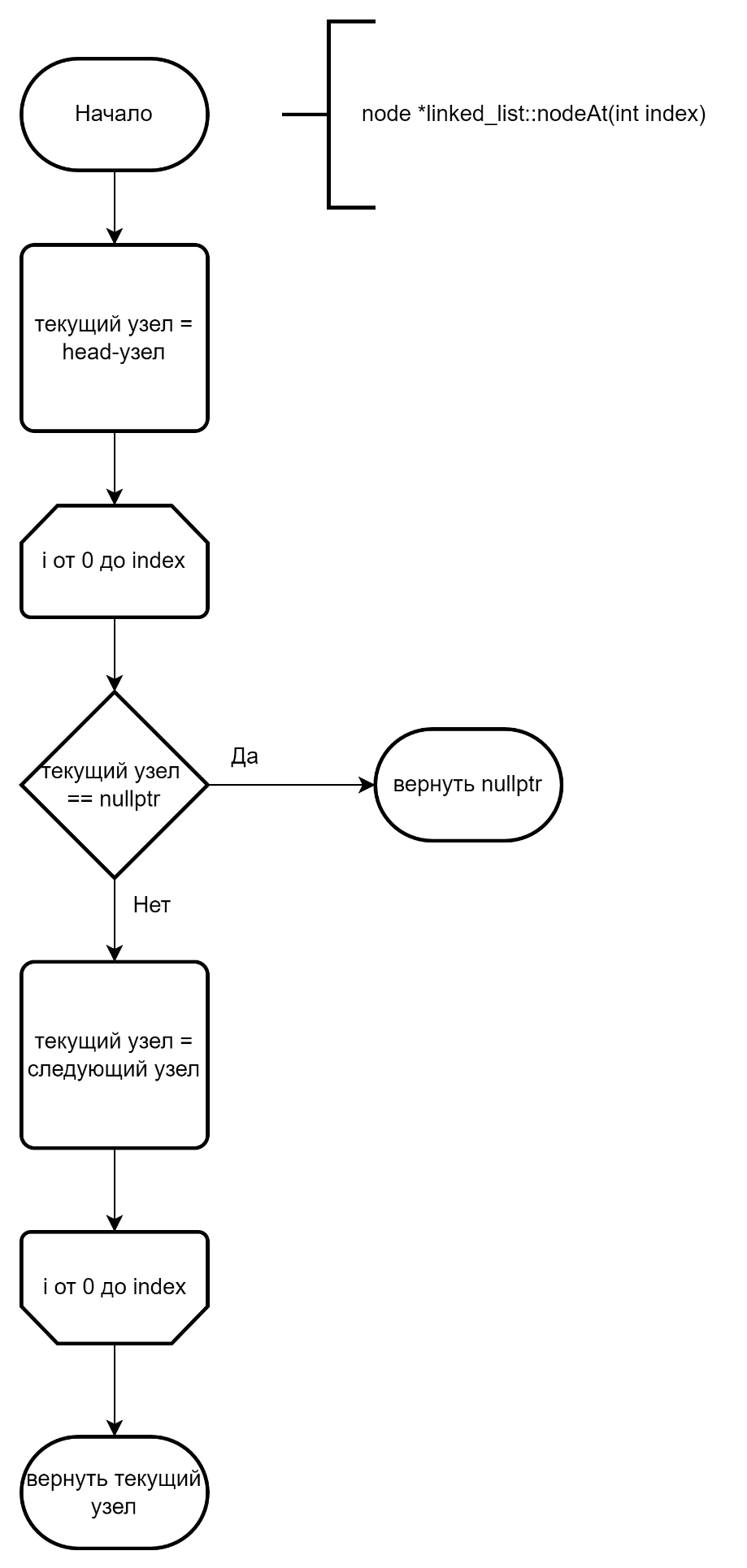
**По каждому кратчайшему пути указать предшествующие вершины.**

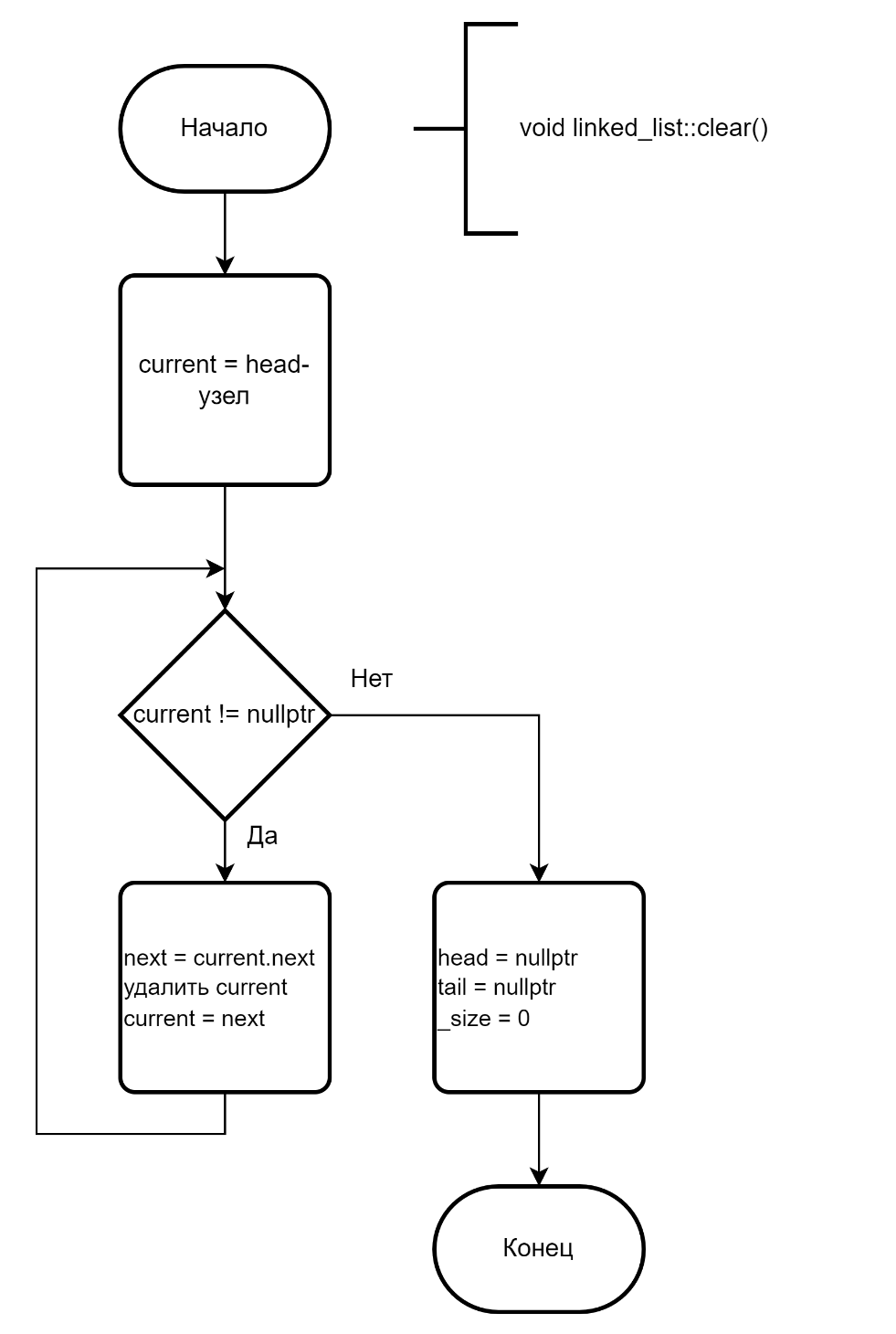
Граф и полученное остовное дерево должны быть изображены на рисунках в отчете.

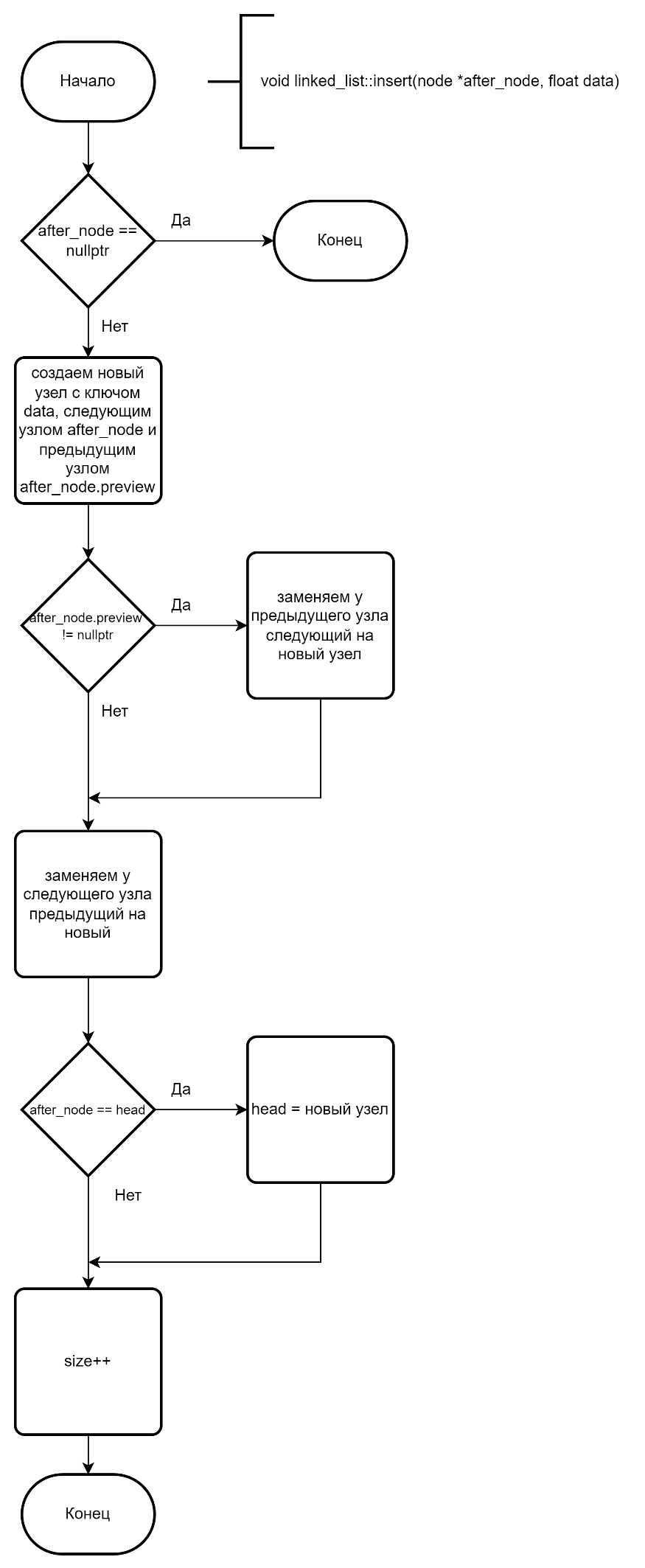
**Варианты заданий**

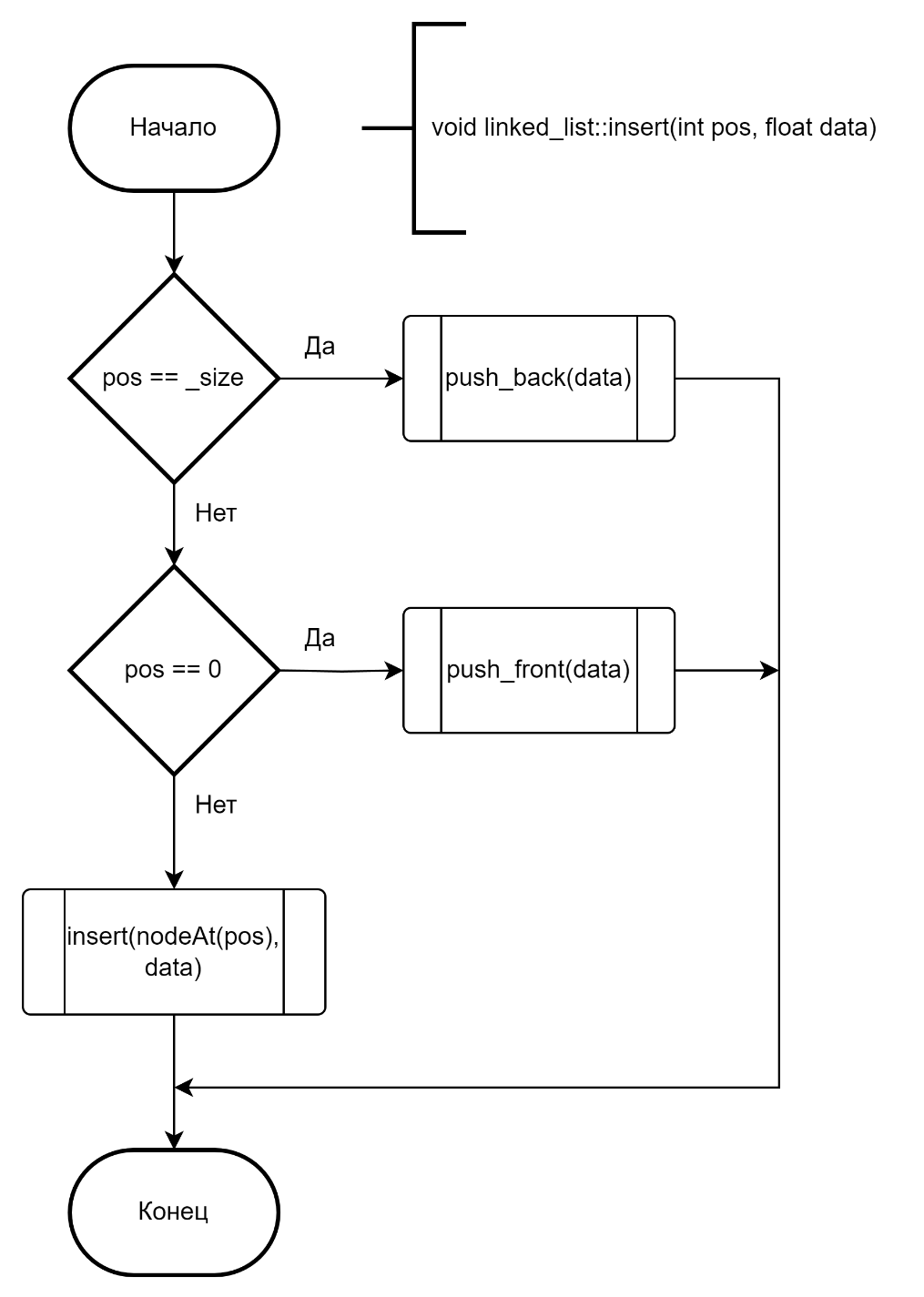
|  |  |  |
| --- | --- | --- |
| № вар. | Алгоритм, реализуемый в п.1 задания | Алгоритм, п.2 задания |
| 4 | Дейкстры | Прима |

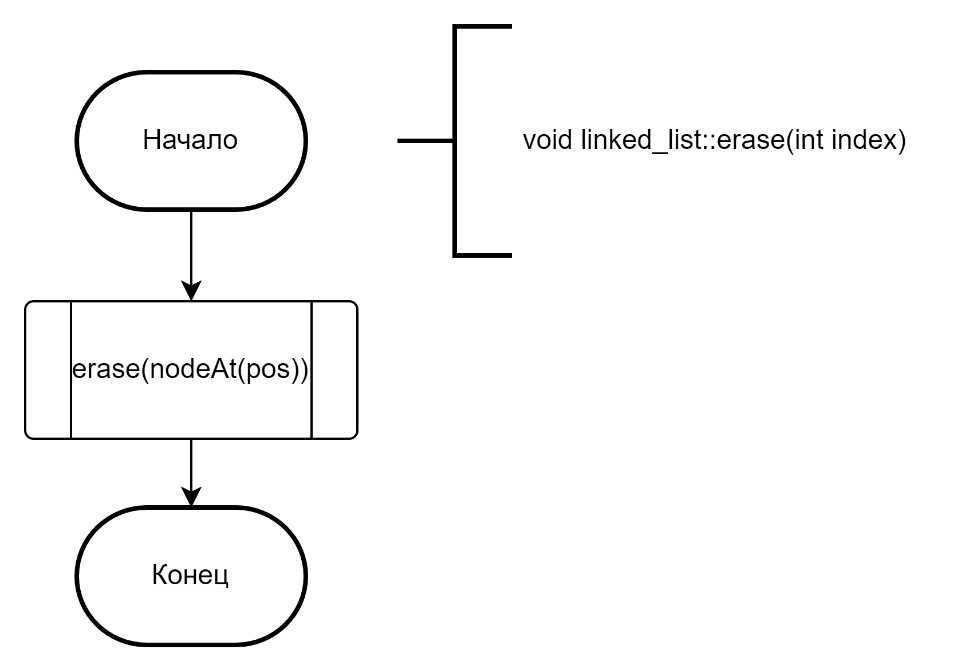
## Блок схема

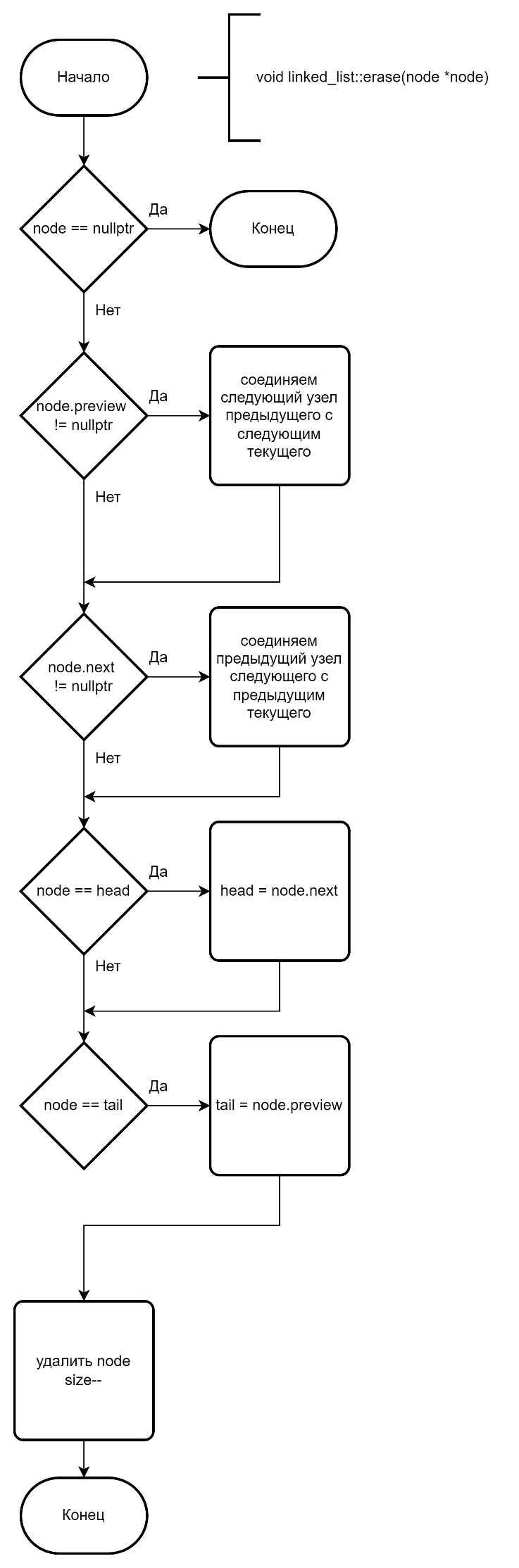


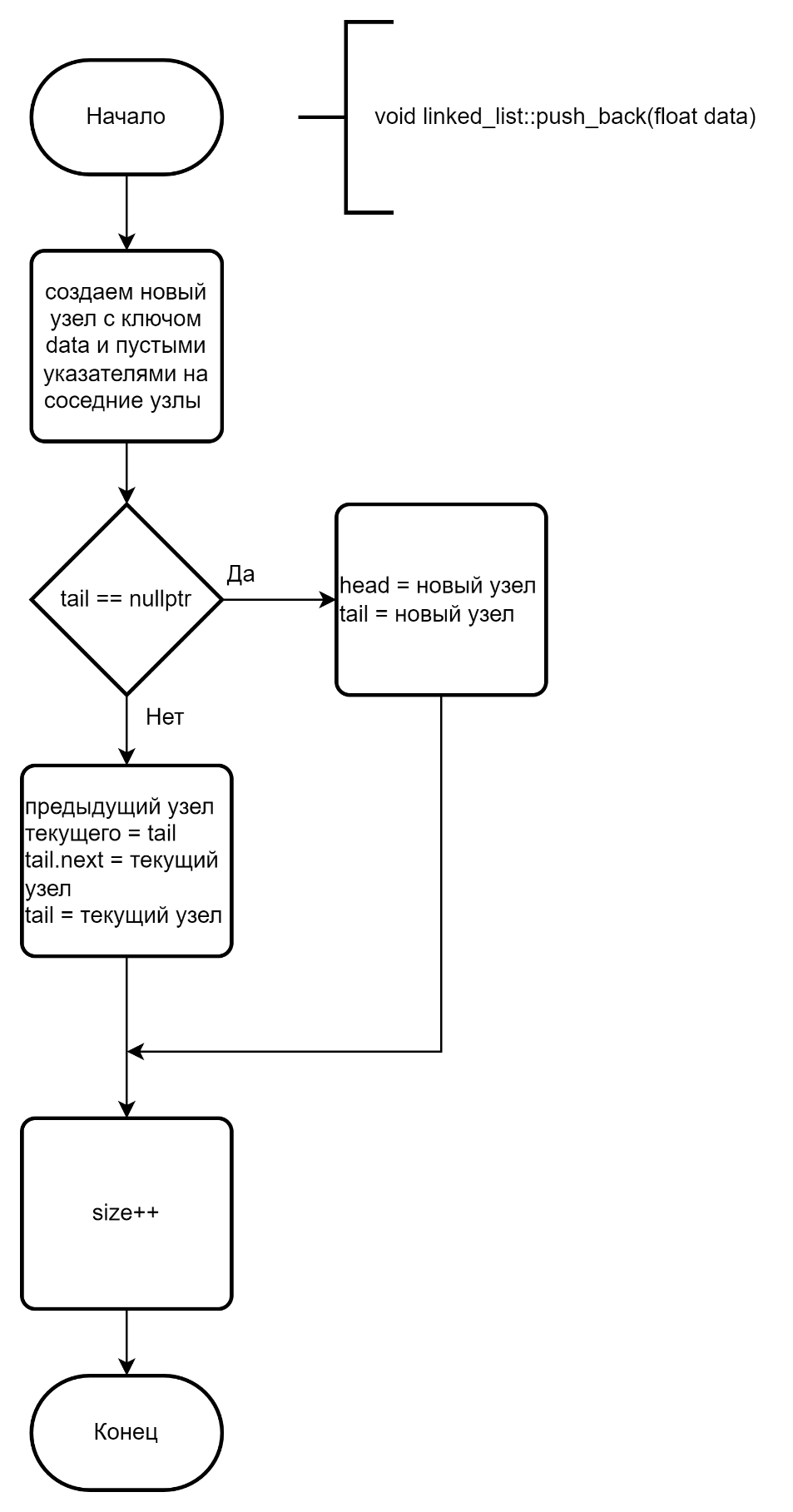


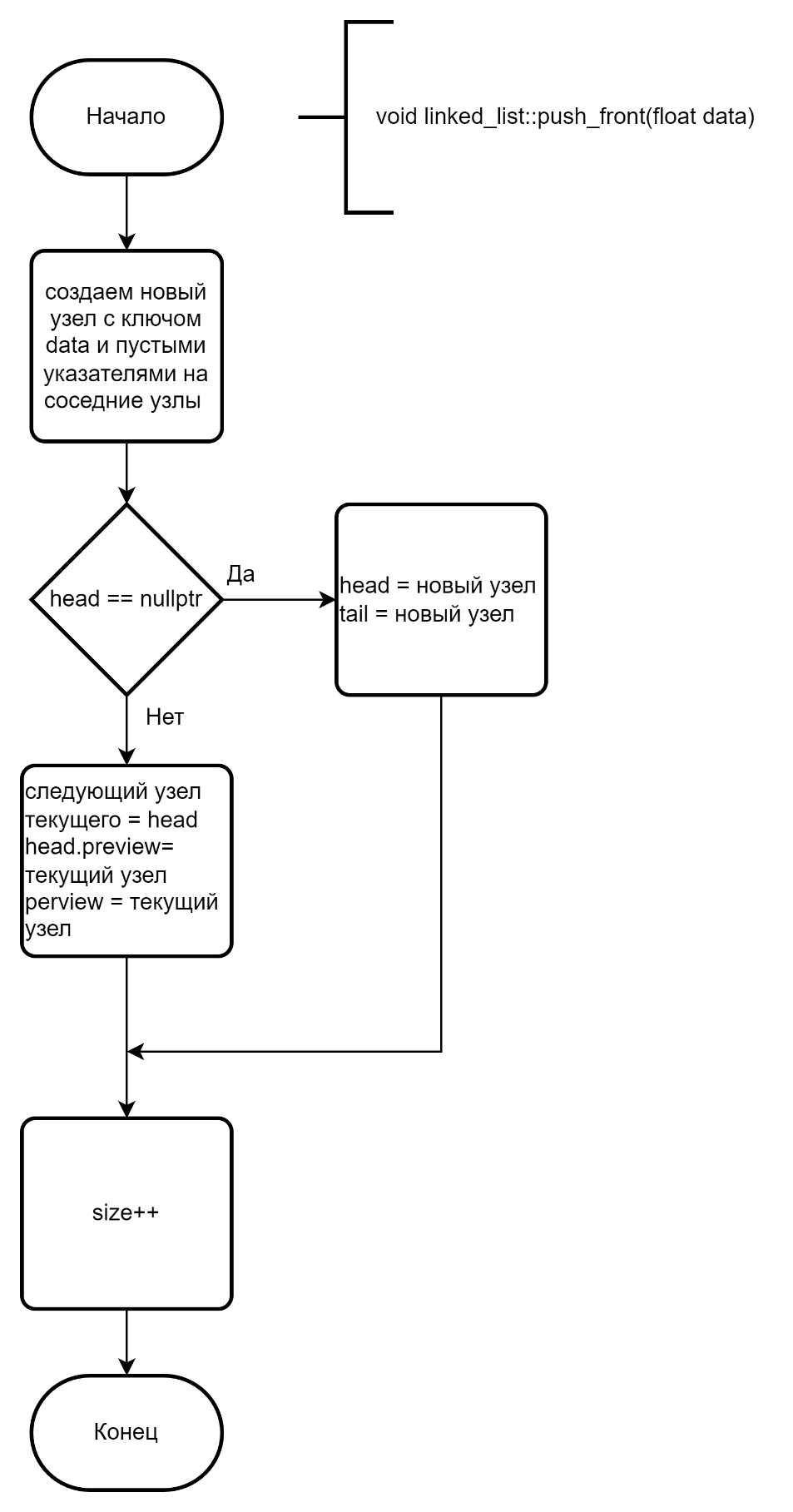


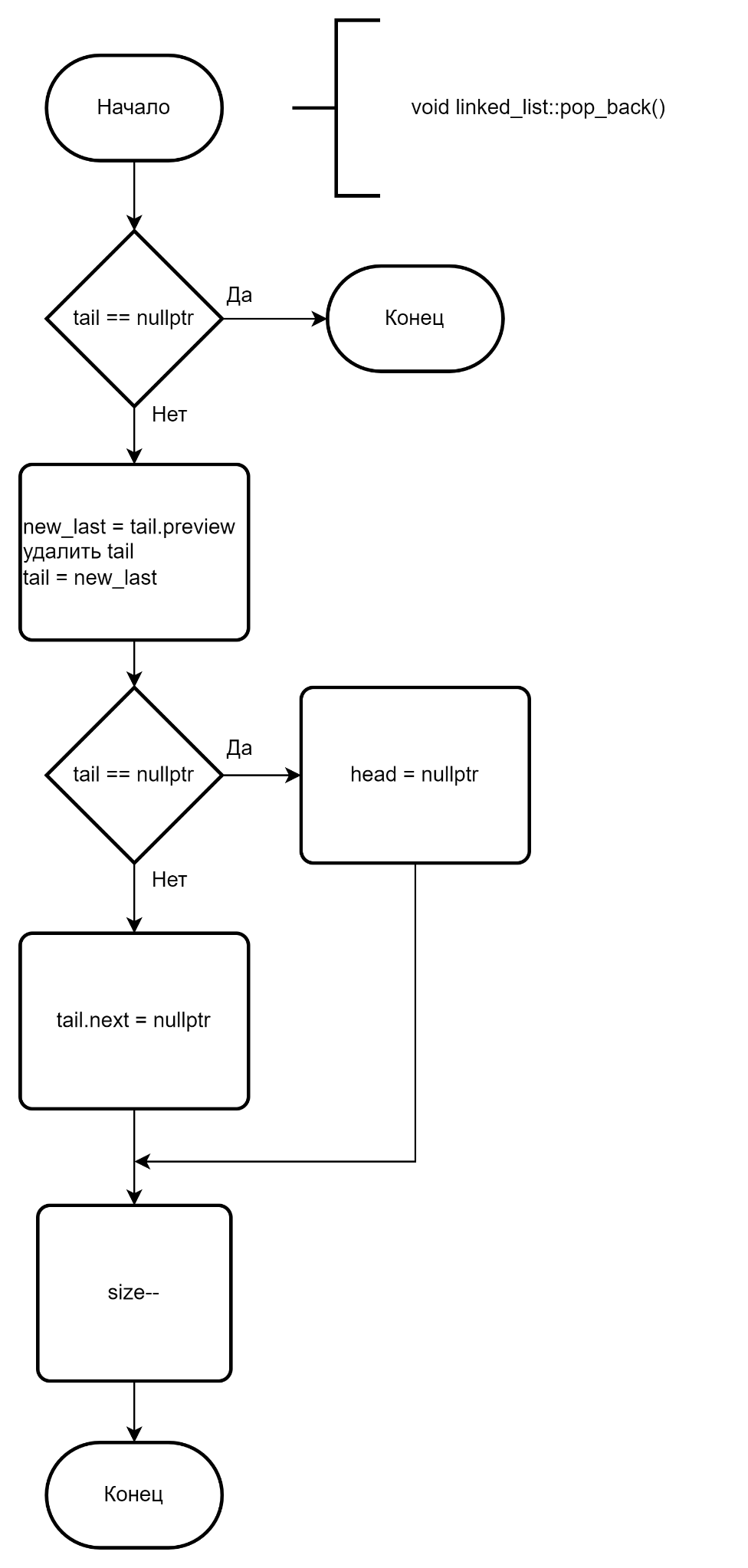


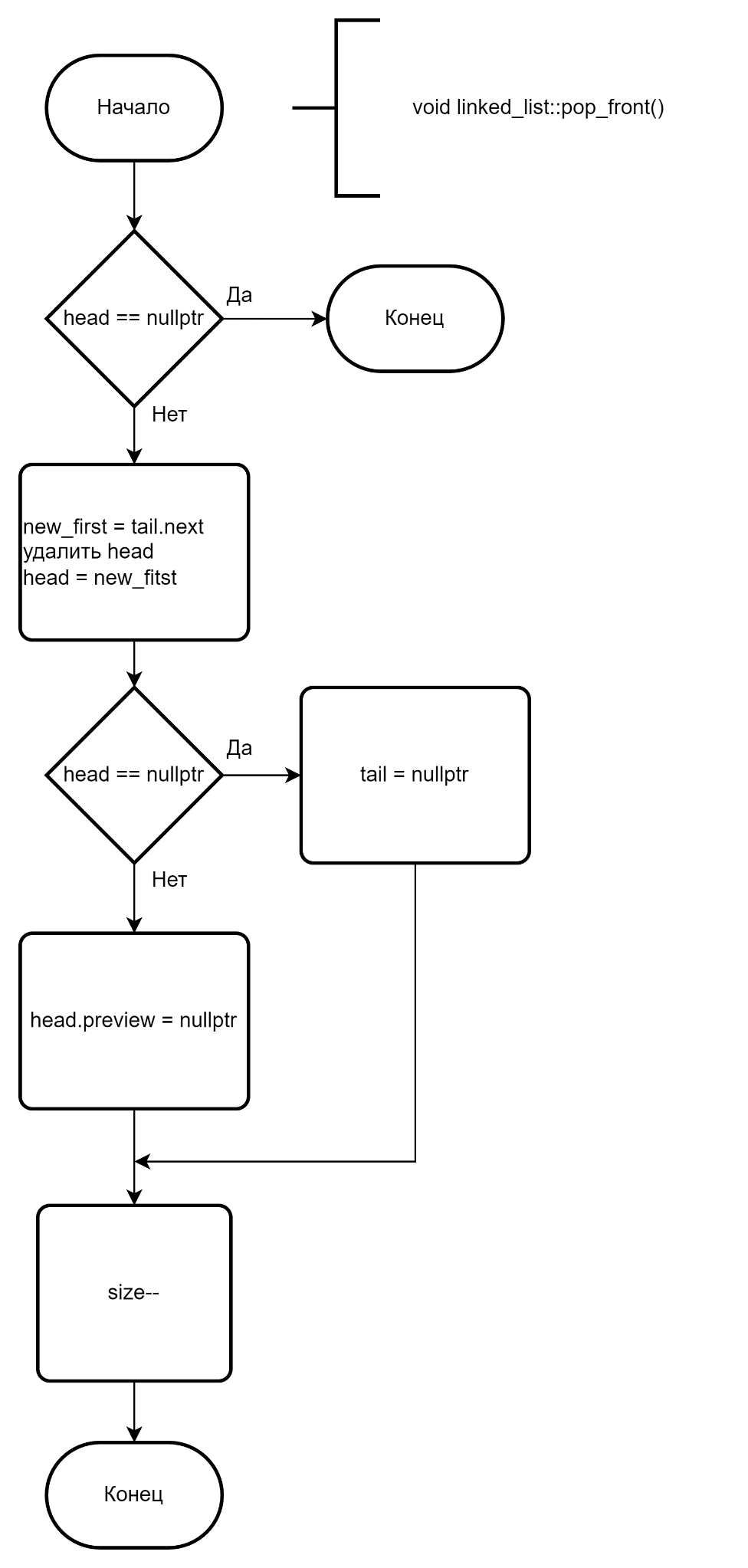


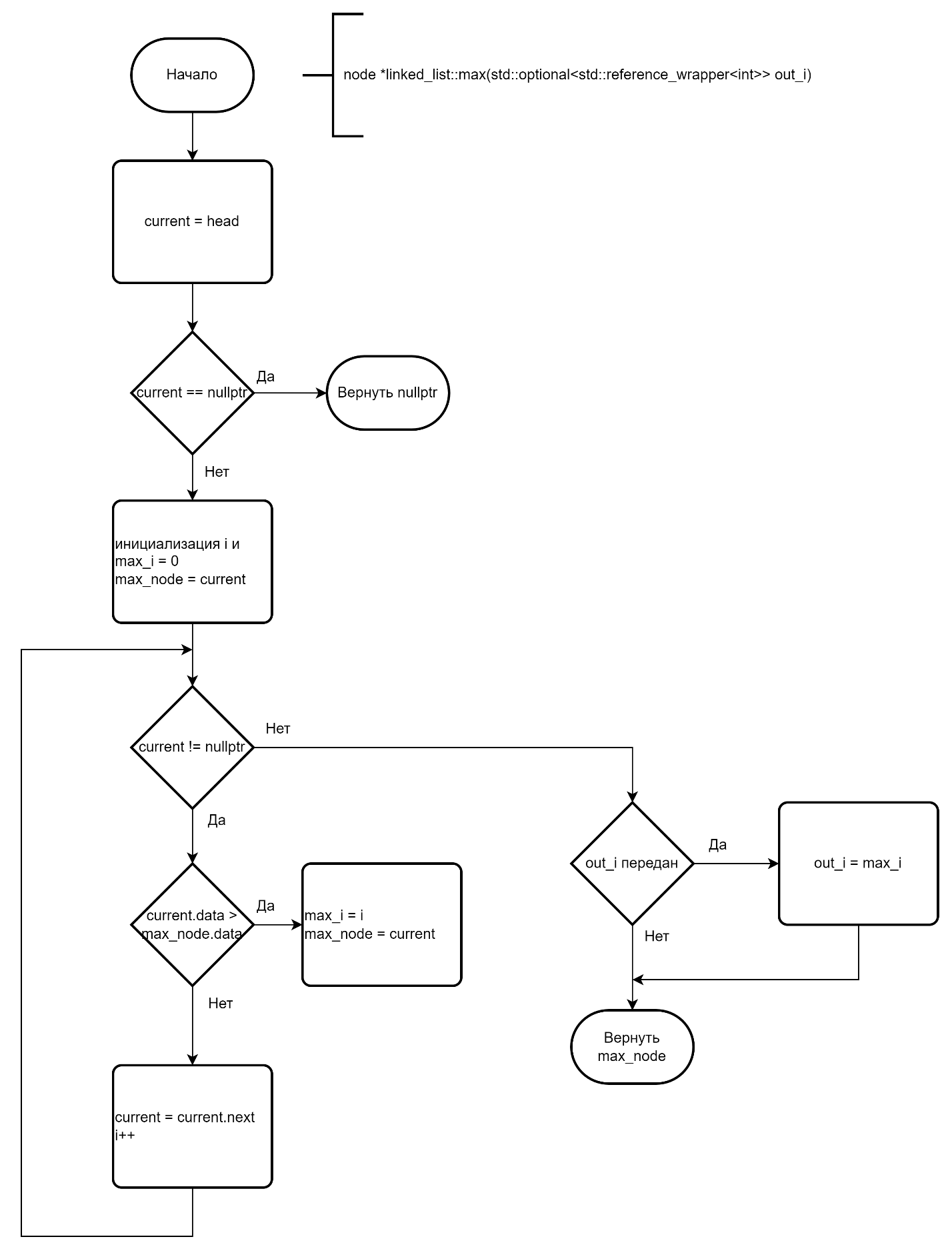


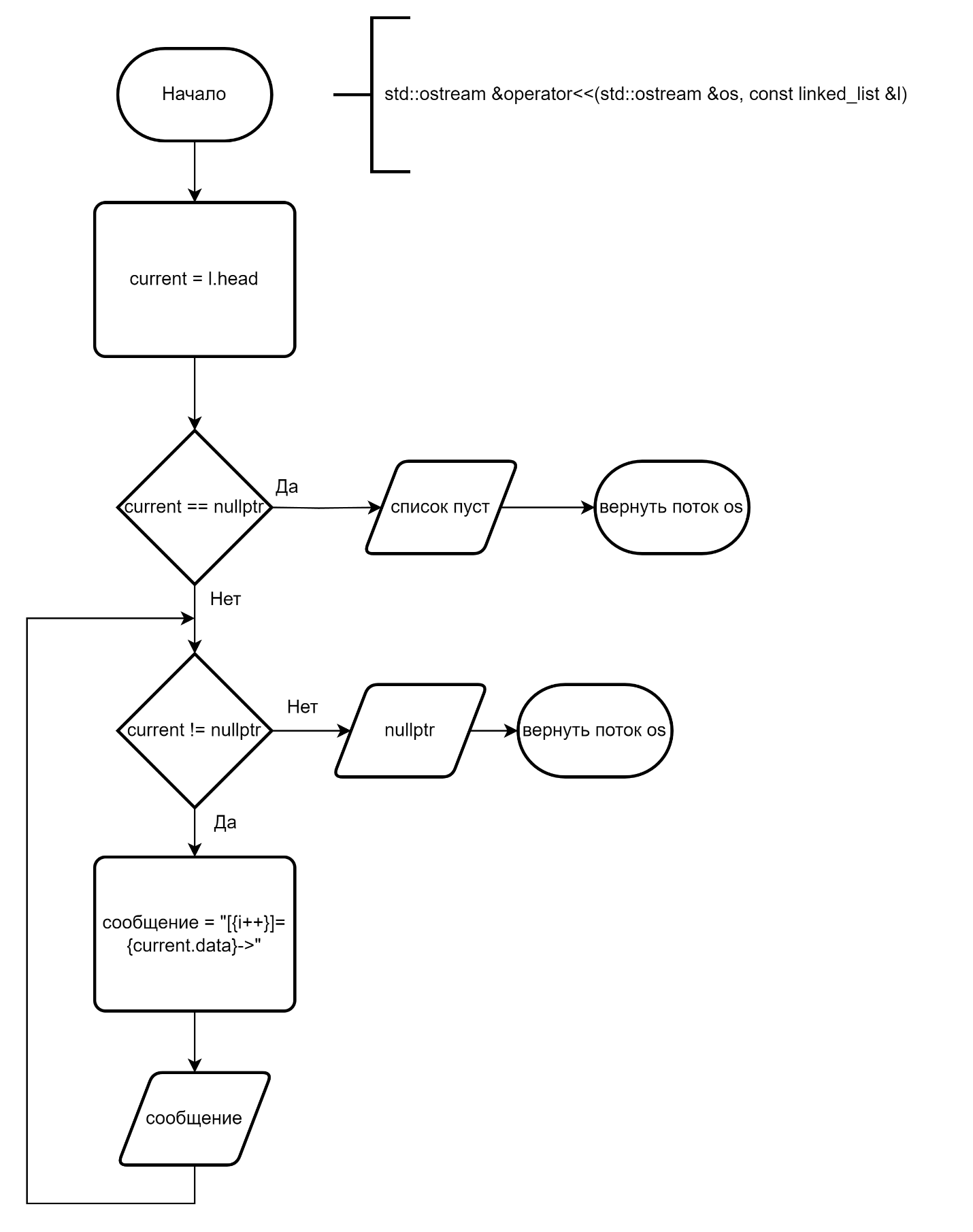












## Код программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* КАФЕДРА № 304 2 КУРС \*

\*---------------------------------------------------------------------------------------------------------\*

\* Project Type : Win32 Console Application \*

\* Project Name : LinkedList \*

\* File Name : main.cpp \*

\* Language : C/C++ \*

\* Programmer(s) : Романов Д.И., Ильин А.А \*

\* Modified By : \*

\* Created : 20/09/2024 \*

\* Last Revision : 15/11/2024 \*

\* Comment(s) : Линейные списки \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <algorithm>

#include <iostream>

#include <random>

#include <fstream>

#include <filesystem>

#include "linked\_list.h"

void fill\_linked\_list(llist::linked\_list& list, int size = 20, float min = -50, float max = 50);

int main() {

std::cout << "Лабораторная работа №4 - Линейные списки\n";

std::cout << " Вариант №8\n";

llist::linked\_list list{};

std::cout << "Создан пустой список\n";

bool exit = false;

while (!exit) {

if(std::cin.fail()) {

std::cout << "Ошибка ввода. Ввод должен быть целым числом\n";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

continue;

}

std::cout << "\nВыберите операцию:\n";

std::cout << "-1) выйти из программы\n";

std::cout << "1) вставить элемент в начало списка\n";

std::cout << "2) вставить элемент в конец списка\n";

std::cout << "3) вставить элемент в заданную позицию списка\n";

std::cout << "4) удалить элемент из начала списка\n";

std::cout << "5) удалить элемент из конца списка\n";

std::cout << "6) удалить элемент из заданной позиции списка\n";

std::cout << "7) очистить список\n";

std::cout << "8) найти максимальный элемент списка\n";

std::cout << "9) вывести список\n";

std::cout << "10) случайно заполнить список\n";

std::cout << "Из них по заданию варианта:\n";

std::cout << "8) найти максимальный элемент списка\n";

std::cout << "3) вставить элемент в заданную позицию списка\n";

std::cout << "5) удалить элемент из конца списка\n";

std::cout << "> ";

int option = 0;

std::cin >> option;

switch (option) {

case -1: {

std::cout << "Выход...\n";

exit = true;

}

break;

case 1: {

std::cout << "Введите значение нового элемента\n> ";

float new\_el;

std::cin >> new\_el;

std::cout << "Вставляем " << new\_el << " в начало\n> ";

list.push\_front(new\_el);

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 2: {

std::cout << "Введите значение нового элемента\n> ";

float new\_el;

std::cin >> new\_el;

std::cout << "Вставляем " << new\_el << " в конец\n> ";

list.push\_back(new\_el);

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 3: {

std::cout << "Введите значение нового элемента\n> ";

float new\_el;

std::cin >> new\_el;

std::cout << "Введите индекс нового элемента. "

"Он будет вставлен перед элементом с соответствующим индексом\n> ";

int new\_index;

std::cin >> new\_index;

std::cout << "Вставляем " << new\_el << " перед " << new\_index << "-м \n> ";

list.insert(new\_index, new\_el);

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 4: {

std::cout << "Удаляем первый элемент " << list.begin()->data << '\n';

list.pop\_front();

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 5: {

std::cout << "Удаляем последний элемент " << list.end()->data << '\n';

list.pop\_back();

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 6: {

std::cout << "Введите индекс удаляемого элемента\n> ";

int index;

std::cin >> index;

std::cout << "Удаляем " << index << "-й элемент \n";

list.pop\_front();

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 7: {

std::cout << "Очищаем список\n";

list.clear();

std::cout << "Содержание списка: " << list << '\n';

}

break;

case 8: {

std::cout << "Находим максимальный элемент списка\n";

int max\_i;

llist::node \*max\_node = list.max(max\_i);

std::cout << "Это " << max\_node->data << " на месте " << max\_i << '\n';

}

break;

case 9: {

std::cout << "Введите название файла для сохранения или "

"оставьте строку пустой, что бы вывести в консоль\n> ";

std::string filename;

std::cin.ignore();

std::getline(std::cin , filename);

if(filename.empty()) {

std::cout << "Содержание списка: " << list << '\n';

} else {

auto output\_path = std::filesystem::current\_path().replace\_filename(filename);

std::cout << "Сохраняем содержание списка в файл " << output\_path << '\n';

std::ofstream fout(output\_path);

fout << list << '\n';

}

}

break;

case 10: {

std::cout << "Добавляем в список 20 случайных элементов в интервале -50 - 50\n";

fill\_linked\_list(list);

std::cout << "Содержание списка: " << list << '\n';

}

break;

default: {

std::cout << "Неизвестная операция\n";

}

break;

}

}

return 0;

}

void fill\_linked\_list(llist::linked\_list& list, int size, float min, float max) {

std::random\_device rd;

auto random\_number = rd();

// unsigned random\_number = 150444277;

std::mt19937 gen(random\_number);

std::uniform\_real\_distribution<float> dist(min, max);

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

list.push\_back(dist(gen));

}

}

linked\_list.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef LW\_LINKED\_LIST\_H

#define LW\_LINKED\_LIST\_H

#include <optional>

#include <iostream>

namespace llist {

class node {

public:

float data;

node \*next;

node \*prev;

explicit node(float newData);

node(float newData, node \*newNext, node \*newPrev);

~node() = default;

friend std::ostream &operator<<(std::ostream &os, const node &n);

};

class linked\_list {

private:

node \*head;

node \*tail;

int \_size;

public:

linked\_list();

~linked\_list();

// node accessors

node \*begin() const; // O(1)

node \*end() const; // O(1)

node \*nodeAt(int index) const; // O(n)

//capacity

bool empty() const; // O(1)

int size() const; // O(1)

//modifiers

void clear(); // O(n)

void insert(node \*node, float data); // before node O(n)

void insert(int pos, float data); // before pos O(n)

void erase(node \*node); // O(1)

void erase(int index); // O(n)

void push\_back(float data); // O(1)

void pop\_back(); //O(1)

void push\_front(float data); // O(1)

void pop\_front(); // O(1)

// operators

node \*max(std::optional<std::reference\_wrapper<int>> = {});

// operator overloads

friend std::ostream &operator<<(std::ostream &os, const linked\_list &l);

};

} // llist

#endif //LW\_LINKED\_LIST\_H

#pragma clang diagnostic pop

linked\_list.cpp

#include "linked\_list.h"

namespace llist {

#pragma region node

node::node(float newData) {

data = newData;

next = nullptr;

prev = nullptr;

}

node::node(float newData, node \*newNext, node \*newPrev) {

data = newData;

next = newNext;

prev = newPrev;

}

std::ostream &operator<<(std::ostream &os, const node &n) {

os << "data: " << n.data << " next: ";

if (n.next == nullptr) {

os << "nullptr";

} else {

os << n.next->data << " (" << n.next << ")";

}

os << " prev: ";

if (n.prev == nullptr) {

os << "nullptr";

} else {

os << n.prev->data << " (" << n.prev << ")";

}

return os;

}

#pragma endregion

#pragma region linked\_list

linked\_list::linked\_list() : head{}, tail{}, \_size{} {}

linked\_list::~linked\_list() {

node \*current = head;

while (current != nullptr) {

node \*next = current->next;

delete current;

current = next;

}

}

node \*linked\_list::begin() const { return head; }

node \*linked\_list::end() const { return tail; }

node \*linked\_list::nodeAt(int index) const {

node \*current = head;

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

if (current == nullptr)

return nullptr;

current = current->next;

}

return current;

}

bool linked\_list::empty() const { return head == nullptr; }

int linked\_list::size() const { return \_size; }

void linked\_list::clear() {

node \*current = head;

// delete nodes until there's none left

while (current != nullptr) {

node \*next = current->next;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

\_size = 0;

}

void linked\_list::insert(node \*after\_node, float data) {

if (after\_node == nullptr) {

return;

}

// link the new node to its neighbors

node \*new\_node = new node(data, after\_node, after\_node->prev);

// link the prev node to the new node

if (after\_node->prev != nullptr) {

after\_node->prev->next = new\_node;

}

// link the after node to the new node

after\_node->prev = new\_node;

if (after\_node == head) {

head = new\_node;

}

\_size++;

}

void linked\_list::insert(int pos, float data) {

if (pos == \_size) {

push\_back(data);

} else if (pos == 0) {

push\_front(data);

} else {

insert(nodeAt(pos), data);

}

}

void linked\_list::erase(node \*node) {

if (node == nullptr) {

return;

}

// link the prev node to the next one

if (node->prev != nullptr) {

node->prev->next = node->next;

}

// link the next node to the prev one

if (node->next != nullptr) {

node->next->prev = node->prev;

}

// fix head

if (node == head) {

head = node->next;

}

// fix tail

if (node == tail) {

tail = node->prev;

}

delete node;

\_size--;

}

void linked\_list::erase(int index) {

erase(nodeAt(index));

}

void linked\_list::push\_back(float data) {

node \*new\_node = new node(data);

if (tail == nullptr) {

// empty

head = new\_node;

tail = new\_node;

} else {

// link tail node to ours

new\_node->prev = tail;

// link our node to tail

tail->next = new\_node;

// fix tail

tail = new\_node;

}

\_size++;

}

void linked\_list::pop\_back() {

if (tail == nullptr) { return; }

node \*new\_last = tail->prev;

delete tail;

tail = new\_last;

if (tail == nullptr) {

// list is empty now, fix head

head = nullptr;

} else {

// unlink new tail

tail->next = nullptr;

}

\_size--;

}

void linked\_list::push\_front(float data) {

node \*new\_node = new node(data);

if (head == nullptr) {

// empty

head = new\_node;

tail = new\_node;

} else {

// link head node to ours

new\_node->next = head;

// link our node to head

head->prev = new\_node;

// fix head

head = new\_node;

}

\_size++;

}

void linked\_list::pop\_front() {

if (head == nullptr) { return; }

node \*new\_first = head->next;

delete head;

head = new\_first;

if (head == nullptr) {

// list is empty now, fix tail

tail = nullptr;

} else {

// unlink new head

head->prev = nullptr;

}

}

node \*linked\_list::max(std::optional<std::reference\_wrapper<int>> out\_i) {

node \*current = head;

if (current == nullptr) {

return nullptr;

}

int i = 0, max\_i = 0;

node \*max = current;

while (current != nullptr) {

if (current->data > max->data) {

max\_i = i;

max = current;

}

current = current->next;

i++;

}

if (out\_i) {

// callee needs max index too

out\_i->get() = max\_i;

}

return max;

}

std::ostream &operator<<(std::ostream &os, const linked\_list &l) {

node \*current = l.head;

if (current == nullptr) {

os << "empty list";

return os;

}

int i = 0;

while (current != nullptr) {

os << '[' << i++ << "]=" << current->data << "->";

current = current->next;

}

os << "nullptr";

return os;

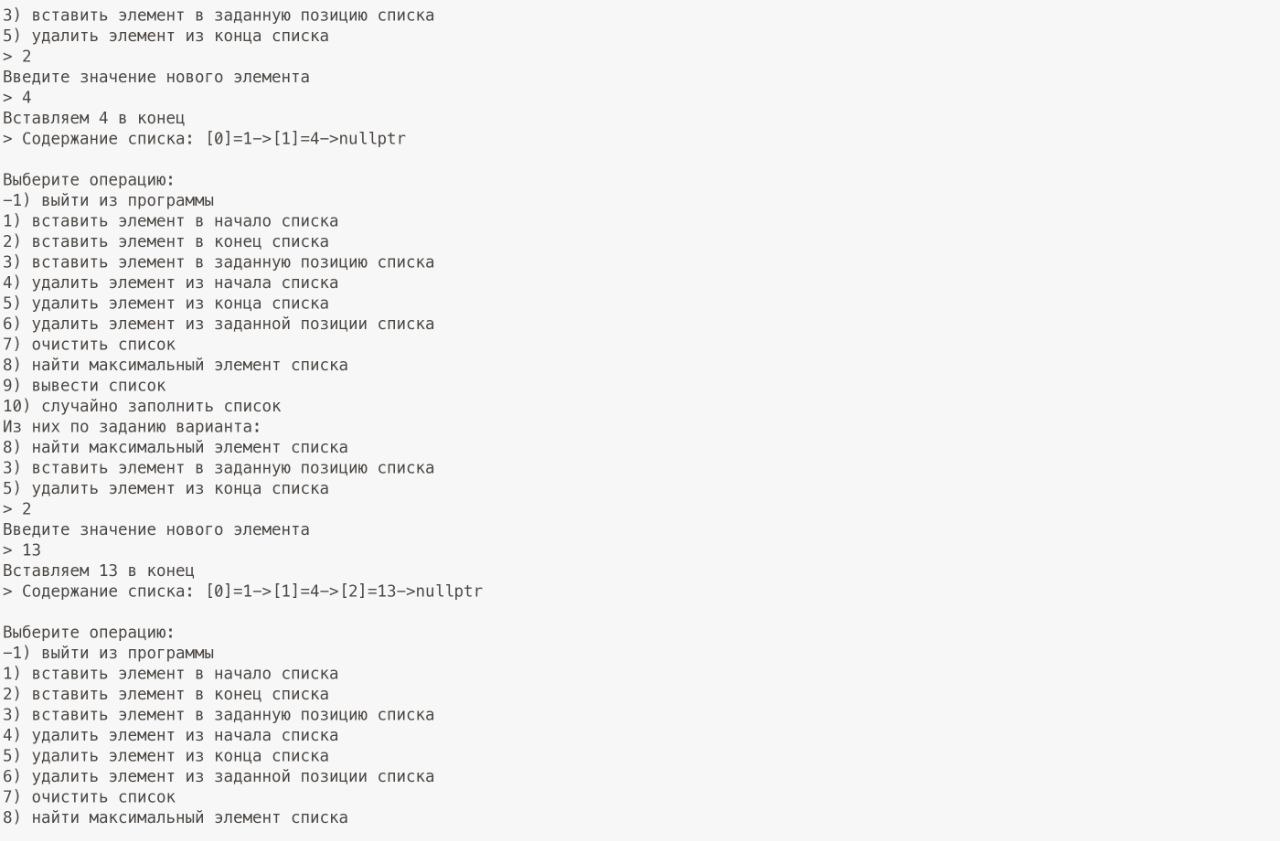
}

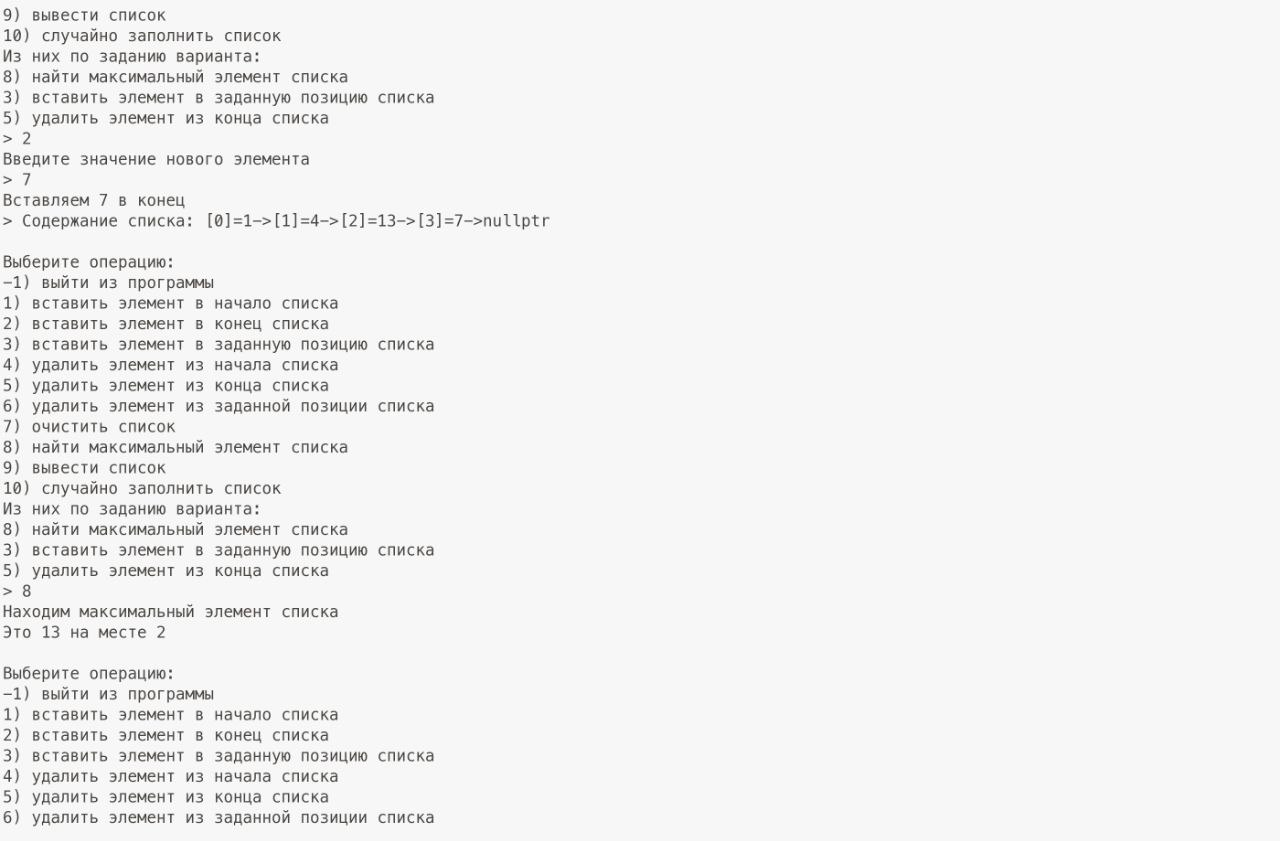
#pragma endregion

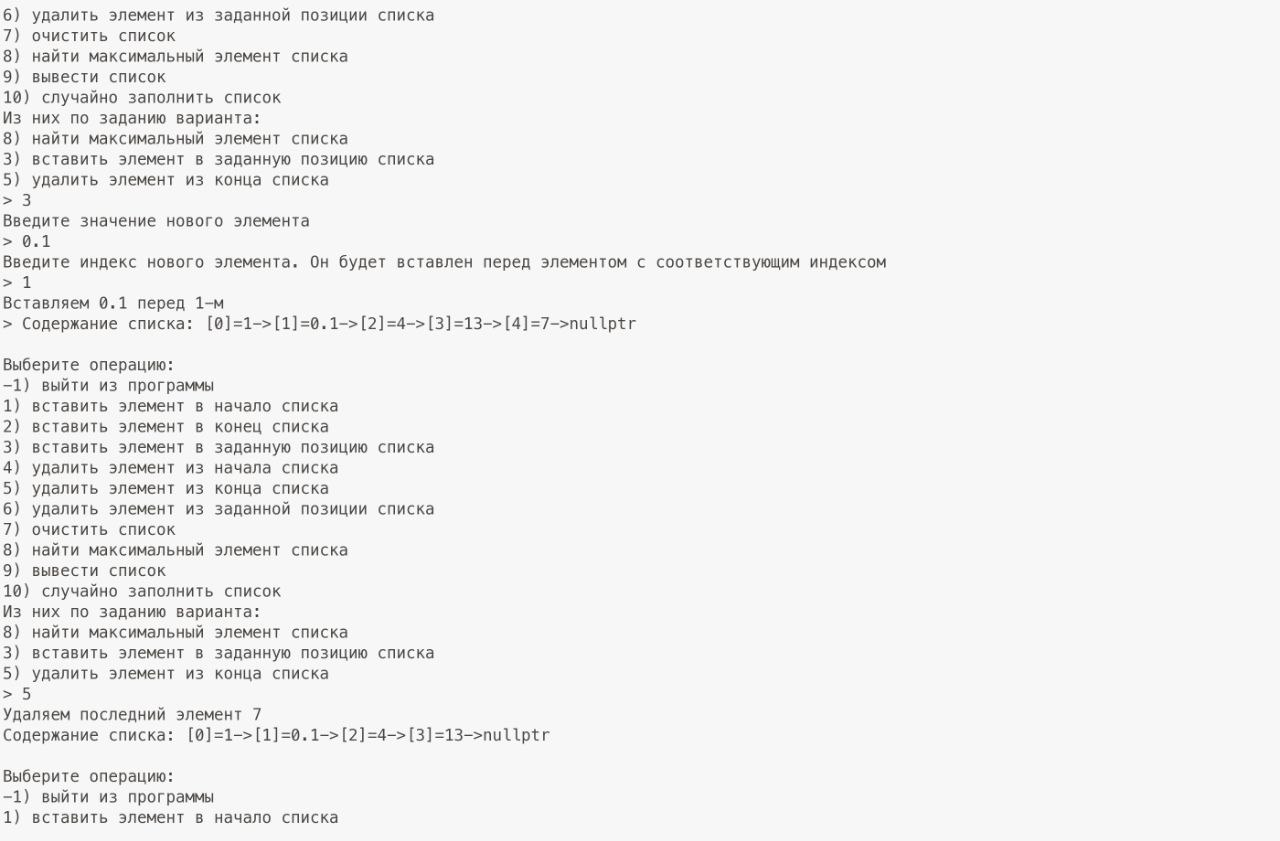
} // llist

## Результаты











## Вывод

В данной работе мы на практике реализовали двусвязный список и разобрались в принципе его работы.