МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий

Кафедра МОСИТ

ИТОГОВЫЙ ОТЧЕТ

ПО ПРАКТИКУМУ

по дисциплине

“СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ”

Практические работы № 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Учебная группа: **ИКБО-02-19**

Преподаватель: **к.т.н., доцент Сыромятников В.П.**

Студент: **Миронов А.Д.**

2020/2021 учебный год

Москва

Содержание

Введение

Структуры и алгоритмы обработки данных крайне важный для любого программиста - будь тот начниающим или уже специалистом с многолетним стажем, ведь любая программа работает с данными. С каждым годом становится все важнее и необходимее более эффективно работать с ними, которые увеличиваются многократно, также кажен выбор структуры или алгоритма, ведь способов применения масса, и они сильно различаются по использованию памяти и быстродействию.

Это дисциплина знакомит нас с фундаментальными понятиями в программипровании, без которых невозможно создать качественный продукт, устроится на работу и называть себя «програмистом» вообще. Поэтому было крайне важно самостоятельно освоить этот базис, но не без помощи преподавательского состава, который направлял нас.

В этом отчете предоставлен итог обучения, где уже можно увидеть практическую ценность наших новообретенных знаний, а также зачатки настоящего профессионала.

Практическая работа № 1

ЛИНЕЙНЫЕ СВЯЗНЫЕ СПИСКИ

Постановка задачи

Составить программу создания линейного односвязного списка (ЛОС) и реализовать основные алгоритмы работы с ЛОС, обеспечивающие следующие действия:

1 - Добавить элемент

1.3.2 – Перед заданным элементом

2 - Удалить элемент

2.3.2 – Все вхождения

6 - Вычислить длину ЛОС

7 - Вывести (распечатать) ЛОС на экран

Перечисленные действия оформить в виде самостоятельных режимов работы созданного ЛОС . Выбор режимов производить с помощью пользовательского меню.

Провести тестирование программы. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Оформить отчет с подробным описанием созданного ЛОС, принципов программной реализации алгоритмов работы с ЛОС, описанием текста исходного кода и проведенного тестирования программы.

Сделать выводы о проделанной работе, основанные на полученных результатах.

1. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

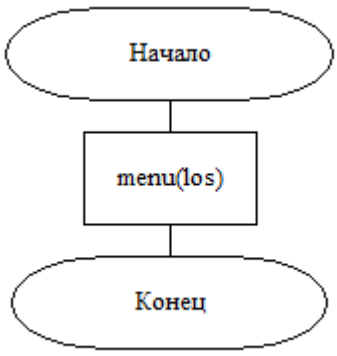


Рис.1 Схема алгоритма функции main

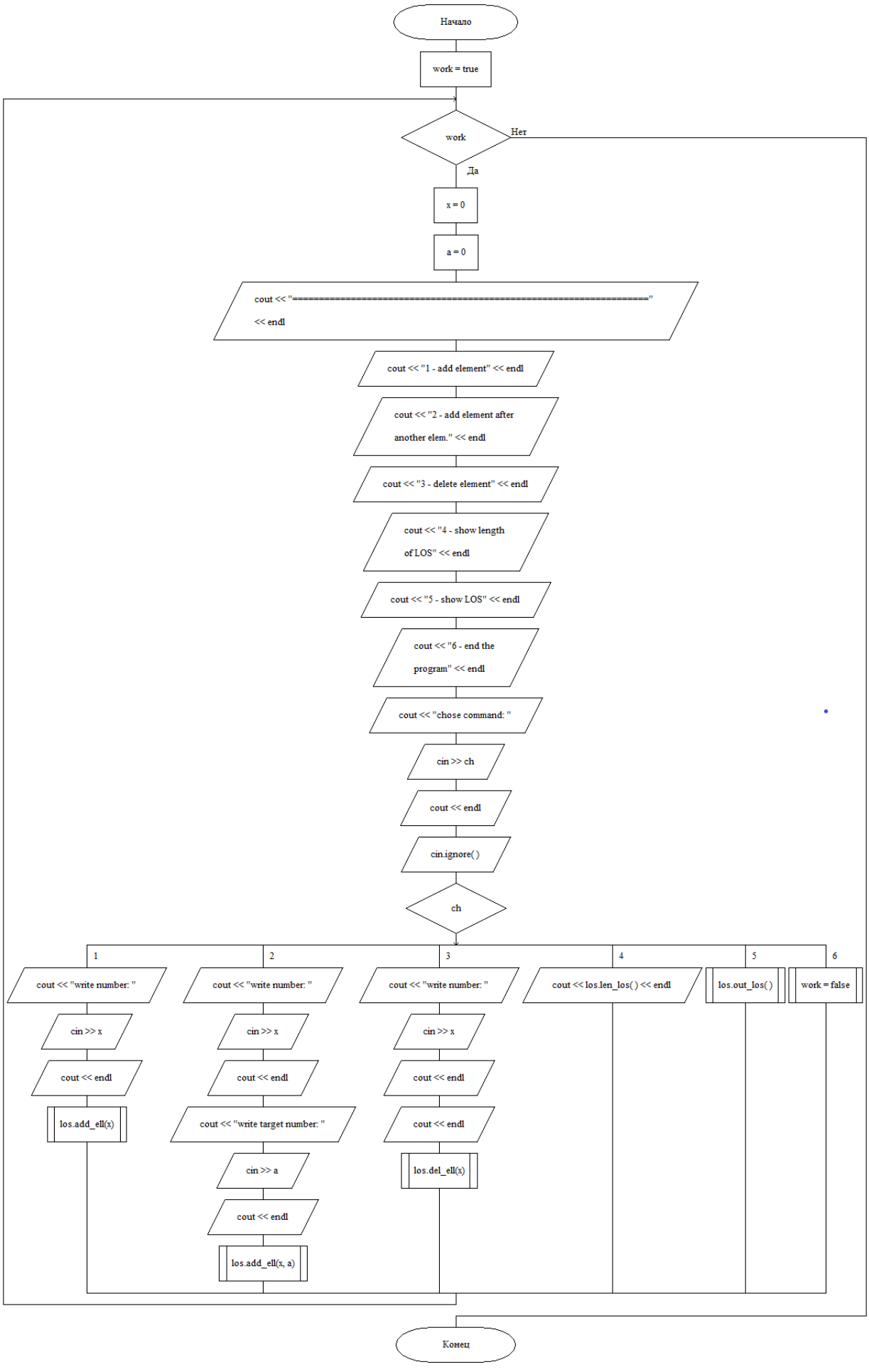

Начало
work = true
work
Нет
Да
x = 0
a = 0
cout << "=="
<< endl
cout << "1 - add element" << endl
cout << "2 - add element after
another elem." << endl
cout << "3 - delete element" << endl
cout << "4 - show length
of LOS" << endl
cout << "5 - show LOS" << endl
cout << "6 - end the
program" << endl
cout << "chose command: "
cin >> ch
cout << endl
cin.ignore()
ch
1
2
3
4
5
6
cout << "write number: "
cin >> x
cout << endl
los.add_ell(x)
cout << "write number: "
cin >> x
cout << endl
cout << "write target number: "
cin >> a
cout << endl
los.add_ell(x, a)
cout << "write number: "
cin >> x
cout << endl
cout << endl
los.del_ell(x)
cout << los.len_los() << endl
los.out_los()
work = false
Конец

Рис.2 Схема алгоритма функции menu

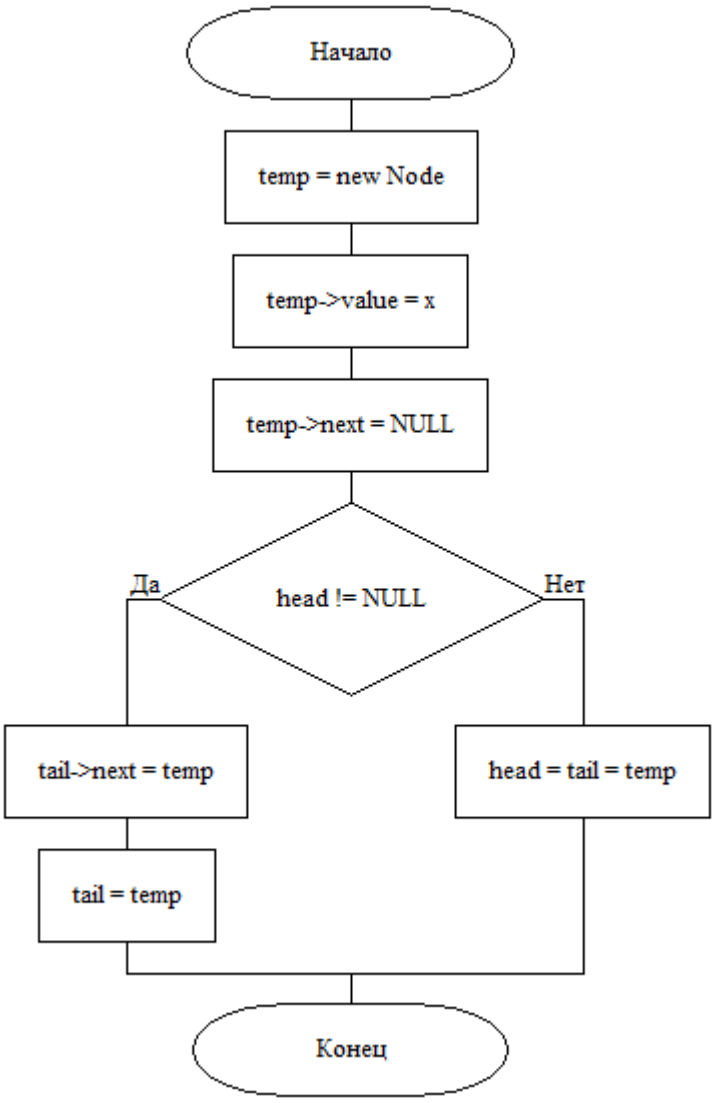


Рис.3 Схема алгоритма функции add_ell

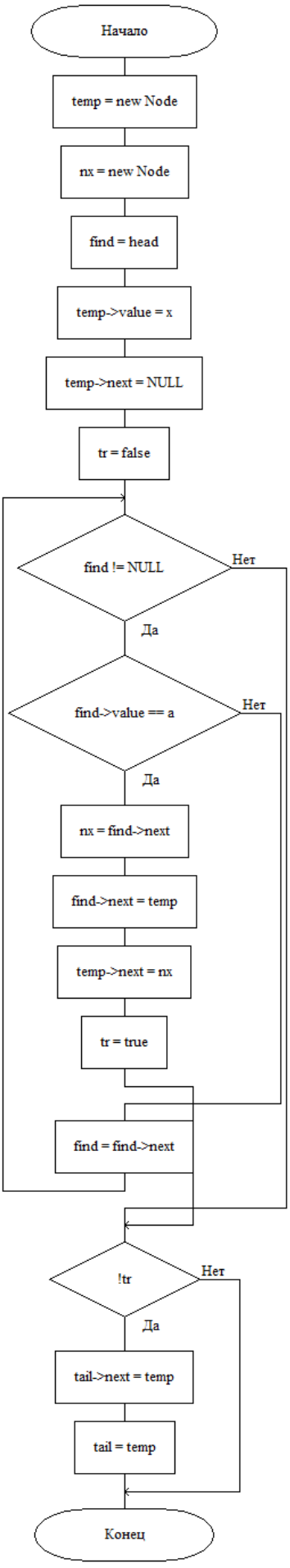

Начало
temp = new Node
nx = new Node
find = head
temp->value = x
temp->next = NULL
tr = false
find != NULL
Нет
Да
find->value == a
Нет
Да
nx = find->next
find->next = temp
temp->next = nx
tr = true
find = find->next
!tr
Нет
Да
tail->next = temp
tail = temp
Конец

Рис.4 Схема алгоритма функции add_ell

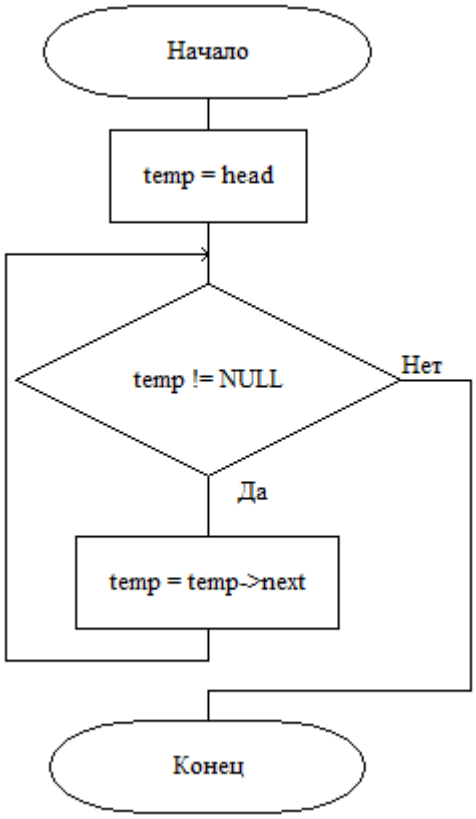


Рис.5 Схема алгоритма функции out_los

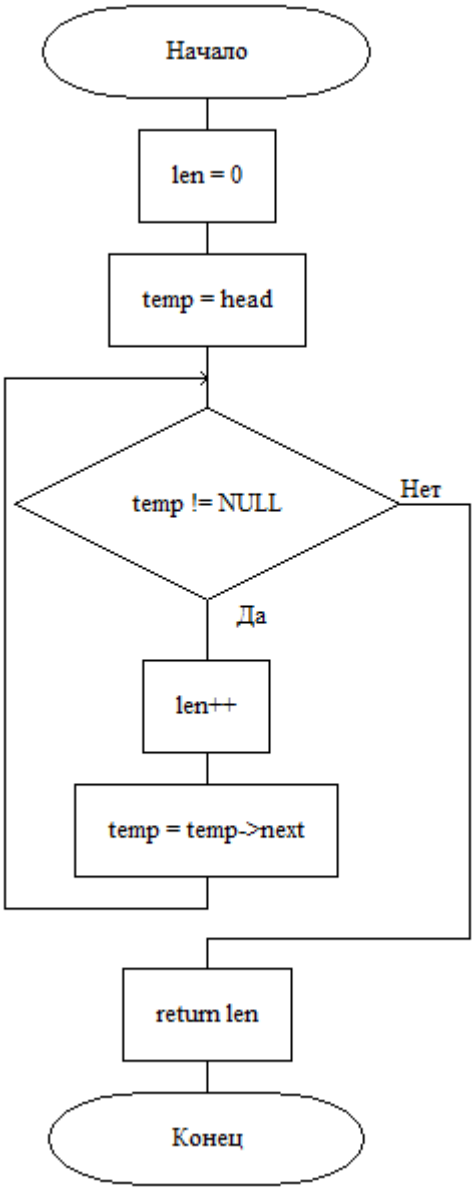


Рис.6 Схема алгоритма функции len_los

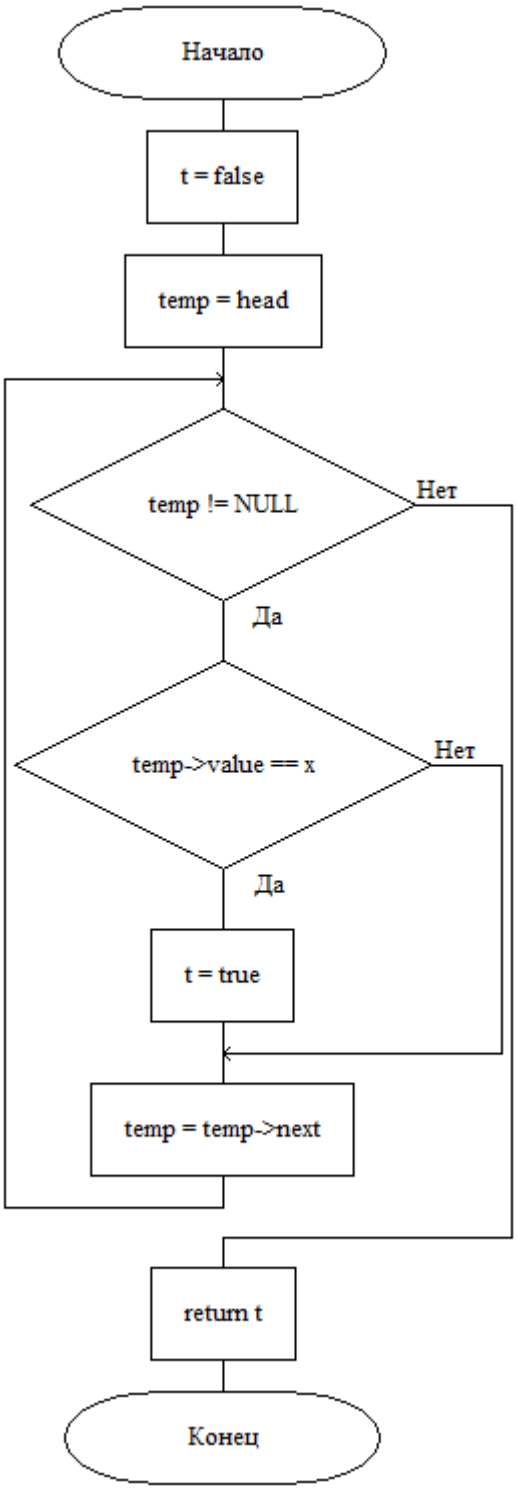


Рис.7 Схема алгоритма функции in_los

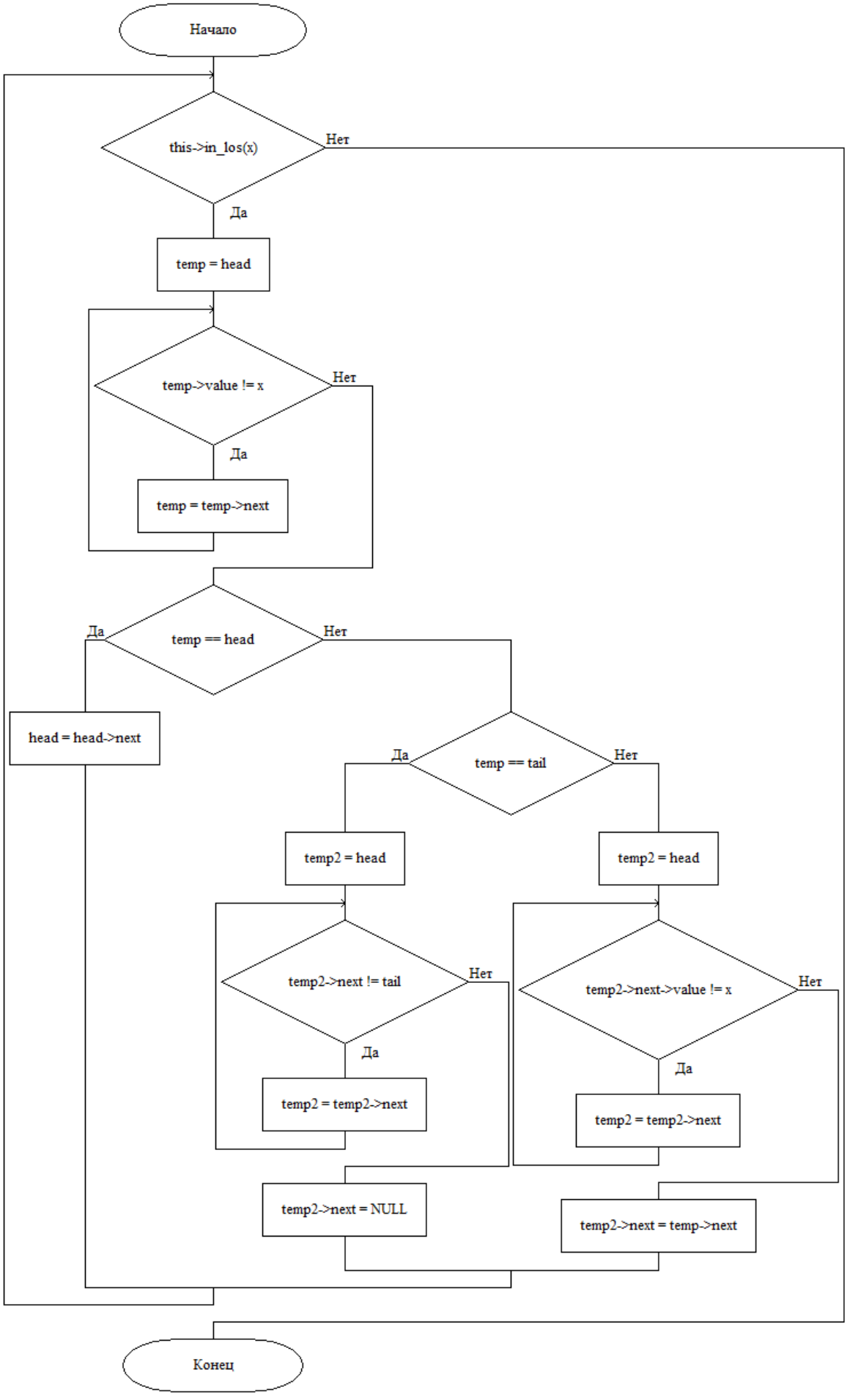

Начало
this->in_los(x)
Нет
Да
temp = head
temp->value != x
Нет
Да
temp = temp->next
temp == head
Да
Нет
head = head->next
temp == tail
Да
Нет
temp2 = head
temp2 = head
temp2->next != tail
Нет
Да
temp2 = temp2->next
temp2->next->value != x
Нет
Да
temp2 = temp2->next
temp2->next = NULL
temp2->next = temp->next
Конец

Рис.7 Схема алгоритма функции dell_ell

2. Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Los.h"

void menu(Los los) {
    bool work = true;
    while (work) {
        int x = 0;
        int a = 0;
        cout << "=====================================================================" << endl;
        cout << "1 - add element" << endl;
        cout << "2 - add element after another elem." << endl;
        cout << "3 - delete element" << endl;
        cout << "4 - show length of LOS" << endl;
        cout << "5 - show LOS" << endl;
        cout << "6 - end the program" << endl;
        cout << "chose command: ";
        int ch;
        cin >> ch;
        cout << endl;
        cin.ignore();
        switch (ch) {
            case 1:
                cout << "write number: ";
                cin >> x;
                cout << endl;

                los.add_ell(x);
                break;
            case 2:
                cout << "write number: ";
                cin >> x;
                cout << endl;
```

```
                cout << "write target number: ";
                cin >> a;
                cout << endl;

                los.add_ell(x, a);
                break;
            case 3:
                cout << "write number: ";
                cin >> x;
                cout << endl;

                cout << endl;
                los.del_ell(x);
                break;
            case 4:
                cout << los.len_los() << endl;
                break;
            case 5:
                los.out_los();
                break;
            case 6:
                work = false;
                break;
        }
    }
}

int main() {
    Los los;
    menu(los);


}
```

Los.h

```
#ifndef PR_1_LOS_H
#define PR_1_LOS_H

#include <iostream>

using namespace std;

struct Node {
    Node *next = NULL;
    int value = 0;
};

class Los {
private:
    Node *head, *tail;
public:
    Los() : head(NULL), tail(NULL) {};

    void add_ell(int x);

    void add_ell(int x, int a);

    void del_ell(int x);

    int len_los();

    void out_los();
```

```
    bool in_los(int x);

};


#endif
```

Los.cpp

```
#include "Los.h"
void Los::add_ell(int x) {
    Node *temp = new Node;
    temp->value = x;
    temp->next = NULL;
    if (head != NULL) {
        tail->next = temp;
        tail = temp;
    } else head = tail = temp;
};

void Los::add_ell(int x, int a) {
    Node *temp = new Node;
    Node *nx = new Node;
    Node *find = head;
    temp->value = x;
    temp->next = NULL;
    bool tr = false;

    while(find != NULL){
        if (find->value==a){
            nx = find->next;
            find->next = temp;
            temp->next = nx;



            tr = true;
            break;
        }
        find = find->next;
    }
    if(!tr){
        tail->next = temp;
        tail = temp;
    }


};

void Los::out los() {
    Node *temp = head;
    while(temp!=NULL){
        cout<<temp->value<<endl;
        temp = temp->next;
    }
}

int Los::len_los() {
    int len = 0;
    Node *temp = head;
    while(temp!=NULL){
```

```
        len++;
        temp = temp->next;
    }
    return len;
}

bool Los::in_los(int x) {
    bool t = false;
    Node *temp = head;
    while(temp!=NULL){
        if(temp->value == x)    t = true;
        temp = temp->next;
    }
    return t;
}

void Los::del_ell(int x) {
    while (this->in_los(x)) {
        Node *temp = head;
        while (temp->value != x) {
            temp = temp->next;
        }
        if (temp == head) {
            head = head->next;
        } else if (temp == tail) {
            Node *temp2 = head;
            while (temp2->next != tail) {
                temp2 = temp2->next;
            }
            temp2->next = NULL;
        } else {
            Node *temp2 = head;
            while (temp2->next->value != x) {
                temp2 = temp2->next;
            }
            temp2->next = temp->next;

        }


    }
}
```

1. Тестирование программы

Ниже представлен результат работы программы с введённым ЛОС

```
1 - add element
2 - add element after another elem.
3 - delete element
4 - show length of LOS
5 - show LOS
6 - end the program
chose command:1

write number:23

========================================================================
```

Рис.8 Скриншот добавления в список

```
========================================================================
1 - add element
2 - add element after another elem.
3 - delete element
4 - show length of LOS
5 - show LOS
6 - end the program
chose command:2

write number:45

write target number:23

========================================================================
```

Рис.9 Скриншот добавления в список перед числом 23

```
==========================================================================
1 - add element
2 - add element after another elem.
3 - delete element
4 - show length of LOS
5 - show LOS
6 - end the program
chose command:3

write number:23



==========================================================================
```

Рис.10 Скриншот удаления из списка числа 23

```
==========================================================================
1 - add element
2 - add element after another elem.
3 - delete element
4 - show length of LOS
5 - show LOS
6 - end the program
chose command:5

45
==========================================================================
```

Рис.11 Скриншот вывода списка на экран

```
==========================================================================
1 - add element
2 - add element after another elem.
3 - delete element
4 - show length of LOS
5 - show LOS
6 - end the program
chose command:4

1
==========================================================================
```

Рис.12 Скриншот вывода длинны списка на экран

1. Выводы

1. В ходе работы был создан односвязный список, ссылки между элементами которого осуществляются при помощи указателей на объекты класса Node.
2. Также были реализованы функции работы с линейным связным списком: добавление элементов, удаление, вывод всех элементов списка, вывод длины.
3. Были изучены положительные и негативные стороны ЛОС:
4. Преимущества: хранение неограниченного количеством и типом данных переменных, неограниченное количество элементов списка, структура ЛОС.
5. Недостатки: не очень удобный поиск конкретного элемента, так как приходится проходить весь ЛОС с крайнего элемента, так как нет способа выбрать конкретный n-ный элемент.
6. Таким образом, была изучена работа линейного связного списка и функций работы над ними и их реализация.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).

Практическая работа № 2

Линейный список. Очередь

Цель работы

Научиться создавать очереди и реализовывать на ней различные алгоритмы

1. **Постановка задачи**

1.

Реализовать очередь (строка).

2.

Задача № 4

Преобразовать арифметическое выражение из префиксной формы в инфиксную. Выражение может содержать скобки. Операнды выражения – целые числа, операции: + - * /.

2. **Решение**

Перевод из префиксной в инфиксную форму осуществляется с помощью очереди и нескольких рекурсивных функций.

```
1 - add elem
2 - remove elem
3 - show first elem
4 - is queue empty
:1
 write string:
qwerty
Elem qwerty has been added
1 - add elem
2 - remove elem
3 - show first elem
4 - is queue empty
:1
 write string:
asdf
Elem asdf has been added
1 - add elem
2 - remove elem
3 - show first elem
4 - is queue empty
:2
 Elem qwerty out
1 - add elem
2 - remove elem
3 - show first elem
4 - is queue empty
:3
 Elem asdf - first in queue
1 - add elem
2 - remove elem
3 - show first elem
4 - is queue empty
:
```

Рис. 1 Интерфейс программы 1

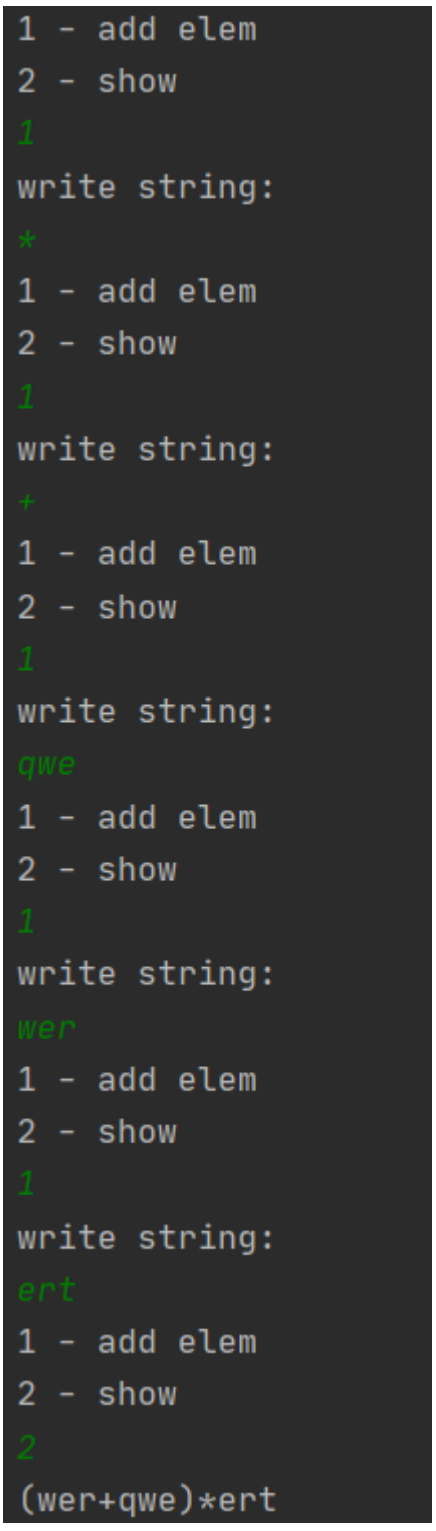


Рис. 2 Интерфейс программы 2

3. **Вывод**

В результате выполнения работы я:

1. Освоил алгоритм работы очереди и его реализацию на языке программирования C++
2. Освоил перевод одной формы записи выражения в другую при помощи очередей

4. **Исходный код программы к заданию 1**

```
#include <iostream>
using namespace std;
struct Node
{
   string x;
   Node* Next;
};
class Queue
{
   Node* Head, * Tail;
public:
   Queue() :Head(NULL), Tail(NULL) {};
   ~Queue();
   void Add_elem(string x);
   void Show();
   void del();
   void isempty();
```

```
};
Queue::~Queue()
{
   Node* temp = Head;
   while (temp != NULL)

   {
      temp = Head->Next;
      delete Head;
      Head = temp;

   }
}
void Queue::Add_elem(string x)
{
   Node* temp = new Node;
   temp->x = x;
   temp->Next = NULL;
   cout << "Elem ";
   cout << x;
   cout << " has been added \n";

   if (Head != NULL)
   {
      Tail->Next = temp;
      Tail = temp;
```

```
    }
    else Head = Tail = temp;
}

void Queue::del()
{
    if (Head != NULL)
    {
        Node* temp = Head;
        cout << "Elem " << Head->x << " out " << endl;
        Head = Head->Next;
        delete temp;
    }
}
void Queue::isempty() {
    Node* temp = Head;
    if (temp == NULL)      cout << "queue is empty \n";
    else cout << "queue is not empty \n";
}
void Queue::Show() {
    Node* temp = Head;
    if (temp == NULL)      cout << "queue is empty \n";
    else cout << "Elem " << temp->x << " - first in queue " << "\n";
}

void menu(Queue lst) {
```

```
//    system("chcp 65001");
    cout << "1 - add elem " << endl;
    cout << "2 - remove elem " << endl;
    cout << "3 - show first elem " << endl;
    cout << "4 - is queue empty " << endl;
    cout<<": ";
    //cout << "5" << endl;
    int inp;
    cin >> inp;

    if (inp == 1) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        lst.Add_elem(inp2);
        menu(lst);
    } else if (inp == 2) {
        lst.del();
        menu(lst);

    } else if (inp == 3) {
        lst.Show();
        menu(lst);
    } else if (inp == 4) {
        lst.isempty();
        menu(lst);
```

```
    }
}

int main()
{
    Queue lst;
    menu(lst);

}
```

5. **Исходный код программы к заданию 2**

Main.cpp

```
#include <iostream>

using namespace std;

#include "Queue.h"
#include <windows.h>
#include <string>
```

```
void act(Queue lst);

bool mux(string x) {
    if (x == "*" || x == "/") return true;
    else return false;
}

bool sign(string x) {
    if (x == "+" || x == "-" || x == "*" || x == "/") return true;
    else return false;
}

void out_str(string x[3]) {

    if (!mux(x[2])) {
        cout << ("(" + x[0] + x[2] + x[1] + ")");
    } else {
        cout << (x[0] + x[2] + x[1]);
    }
}


void show_2_3_4(Queue lst) {
    if (lst.len() == 4) {
        string *s = new string[4];
        s[0] = lst.Show();
```

```
    lst.del();

    s[1] = lst.Show();
    lst.del();

    s[2] = lst.Show();
    lst.del();

    s[3] = lst.Show();
    lst.del();
    if (mux(s[0]) && !mux(s[1])) cout << (s[0] + "(" + s[2] + s[1] + s[3] + ")");
    else cout << (s[0] + s[2] + s[1] + s[3]);
} else if (lst.len() == 3) {
    string *s = new string[3];
    s[0] = lst.Show();
    lst.del();

    s[1] = lst.Show();
    lst.del();

    s[2] = lst.Show();
    lst.del();
    if (mux(s[0])) cout << (s[1] + s[0] + s[2]);
    else cout << ("(" + s[1] + s[0] + s[2] + ")");
} else if (lst.len() == 2) {
    string *s = new string[2];
```

```
        s[0] = lst.Show();
        lst.del();

        s[1] = lst.Show();
        lst.del();
        cout << (s[0] + s[1]);

    }
}

void act2(Queue lst, string *buf, Queue lst_2) {

    if (!sign(buf[0]) && !sign(buf[1]) && sign(buf[2])) {
        out_str(buf);
        if (lst.len() >= 3) {
            for (int i = 0; i < 3; i++) {
                buf[2 - i] = lst.Show();
                lst.del();
            }
            act2(lst, buf, lst_2);

        } else {
            int ln = lst.len();
            for (int i = 0; i < ln; i++) {

                string t = lst.Show();
```

```
            lst.del();

            lst_2.Add_elem(t);

        }
        act(lst_2);
    }

} else {
    if (lst.len() >= 1) {
        lst_2.Add_elem(buf[2]);
        buf[2] = buf[1];
        buf[1] = buf[0];
        buf[0] = lst.Show();
        lst.del();
        act2(lst, buf, lst_2);
    } else {
        lst_2.Add_elem(buf[2]);
        lst_2.Add_elem(buf[1]);
        lst_2.Add_elem(buf[0]);
        act(lst_2);
    }
}

for (int i = 0; i < 3; i++) {
```

```
    }

}


void act(Queue lst) {
    Queue lst_2;
    string buf[3] = {"", "", ""};
    if (lst.len() <= 4) show_2_3_4(lst);
    else {
        for (int i = 0; i < 3; i++) {
            buf[2 - i] = lst.Show();
            lst.del();
        }
        act2(lst, buf, lst_2);

    }

}

void menu(Queue lst) {

    cout << "1 - add elem " << endl;
    cout << "2 - show " << endl;
```

```
    int inp;
    cin >> inp;

    if (inp == 1) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        lst.Add_elem(inp2);
        menu(lst);
    } else {
        act(lst);
    }
}

int main() {
    SetConsoleOutputCP(CP_UTF8);
    Queue lst;
    menu(lst);
}
```

Queue.h

```
#include <iostream>

#ifndef PR_1_2_QUEUE_H
#define PR_1_2_QUEUE_H

struct Node {
```

```
    std::string x;
    Node *Next = NULL;
};

class Queue {

    Node *Head, *Tail;
public:
    Queue() : Head(NULL), Tail(NULL) {};

    ~Queue();

    void Add_elem(std::string x);

    string Show();

    void del();

    int len();


};


#endif //PR_1_2_QUEUE_H
```

Queue.cpp

```
#include <iostream>

using namespace std;

#include "Queue.h"

Queue::~Queue() {
    Node *temp = Head;
    while (temp != NULL) {
        temp = Head->Next;
        delete Head;
        Head = temp;

    }
}

void Queue::Add_elem(string x) {
    Node *temp = new Node;
    temp->x = x;
    temp->Next = NULL;
//    cout << "Elem ";
//    cout << x;
//    cout << " was added\n";

    if (Head != NULL) {
```

```
        Tail->Next = temp;
        Tail = temp;
    } else Head = Tail = temp;
}

void Queue::del() {
    if (Head != NULL) {
        Node *temp = Head;
//        cout << "Elem " << Head->x << " out " << endl;
        Head = Head->Next;
//        return temp->x;
        delete temp;

    } else cout << "queue is empty \n";
}

string Queue::Show() {
    Node *temp = Head;
    return temp->x;
//    if (temp == NULL)    cout << "queue is empty \n";
//    else cout << "Elem " << temp->x << " - first in queue " << "\n";
}

int Queue::len() {
    Node *temp = Head;
    int c = 0;
```

```
    if (temp == NULL) return 0;
    else {
        while (temp != NULL) {
            temp = temp->Next;
            c++;
        }
        return c;
    }
}
```

Практическая работа № 3

СТЕКИ

Вариант 22

Постановка задачи

Составить программу создания стека на базе линейного односвязного списка и реализовать основные алгоритмы работы со стеком, обеспечивающие следующие действия:

1 - Добавить элемент
2 – Вывести верхний элемент
2.1 – С удалением
2.2 – Без удаления
3 – Проверить наличие элементов
4 - Вычислить длину стека
5 - Вывести (распечатать) стек на экран

Перечисленные действия оформить в виде самостоятельных режимов работы созданного стека . Выбор режимов производить с помощью пользовательского меню.
Провести полное тестирование (всех режимов работы) программы на стеке, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.
Оформить отчет с подробным описанием созданного стека, принципов программной реализации алгоритмов работы со

стеком, описанием текста исходного кода и проведенного тестирования программы.
Сделать выводы о проделанной работе, основанные на полученных результатах.

3. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add**– функция добавления элемента в стек
- **void pop** – функция извлечения элемента из стека (удаление)
- **void peek** – функция возврата верхнего элемента (без удаления)
- **void show –** функция вывода стека на экран
- **void not_empty** – функция проверки наличия элементов
- **void len –** функция вычисления длинны стека

Стек (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю. В цифровом вычислительном комплексе стек называется магазином — по аналогии с магазином в огнестрельном оружии (стрельба начнётся с патрона, заряженного последним). В 1946 Алан Тьюринг ввёл понятие стека. А в 1957 году немцы Клаус Самельсон и Фридрих Л. Бауэр запатентовали идею Тьюринга.

Рис.1 Принцип работы стека

Функция main создает класс Stack и вызывает функцию menu.

Функция peek возвращает значение верхнего элемента.

Функция not_empty проверяет не является ли головной элемент NULL.

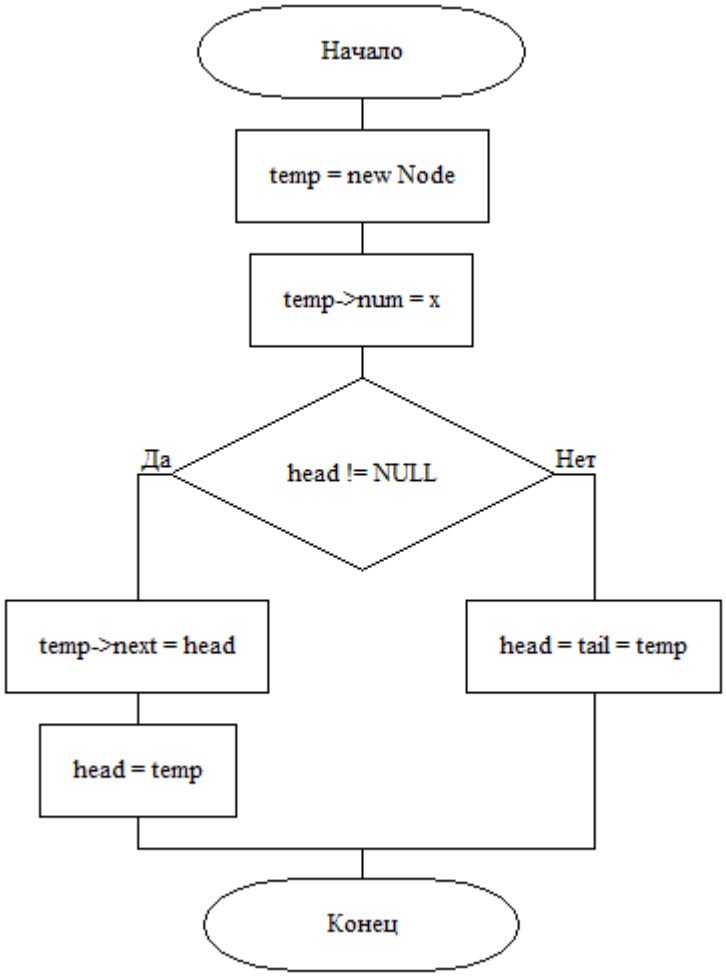


Рис.2 Схема алгоритма функции add

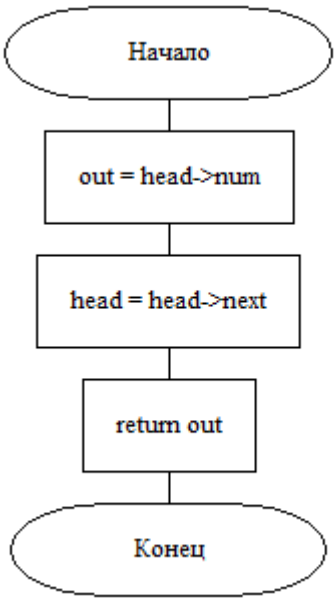


Рис.3 Схема алгоритма функции pop

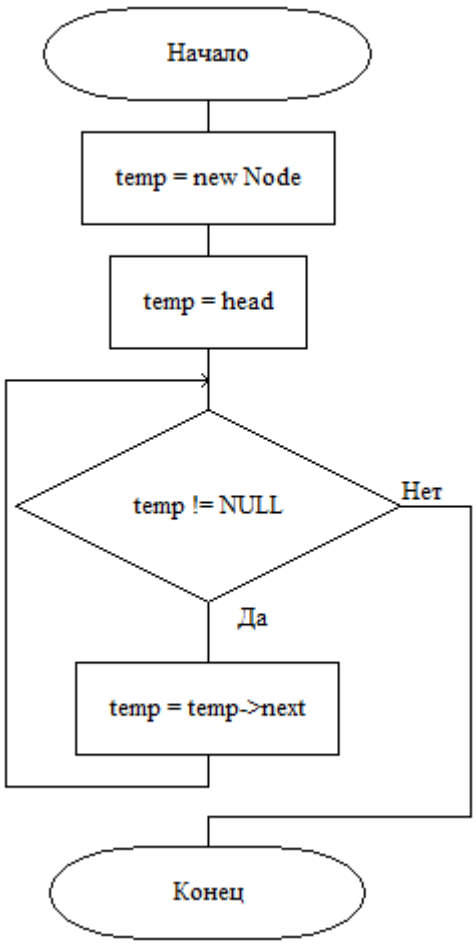


Рис.4 Схема алгоритма функции show

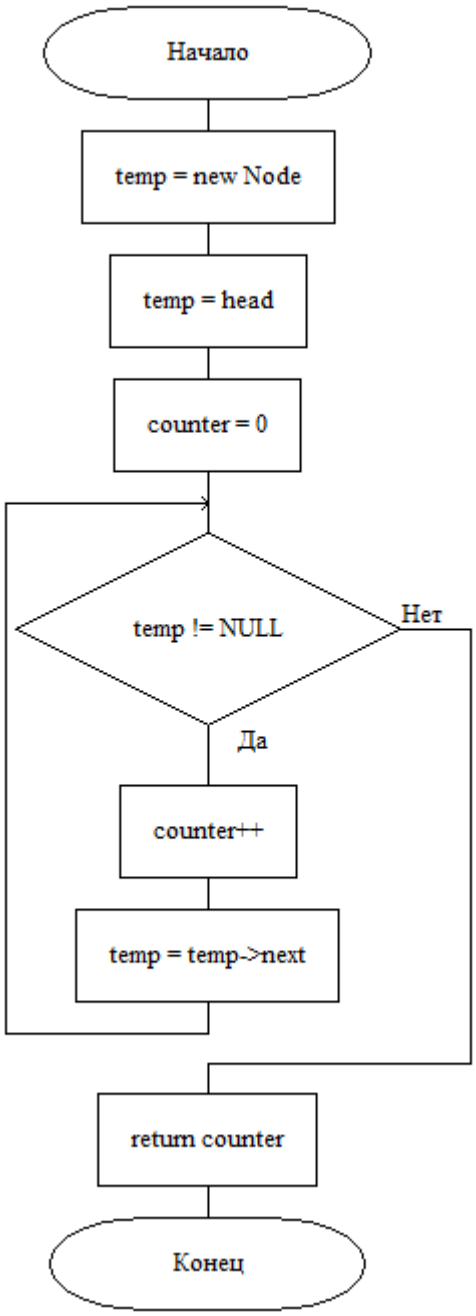


Рис.5 Схема алгоритма функции len

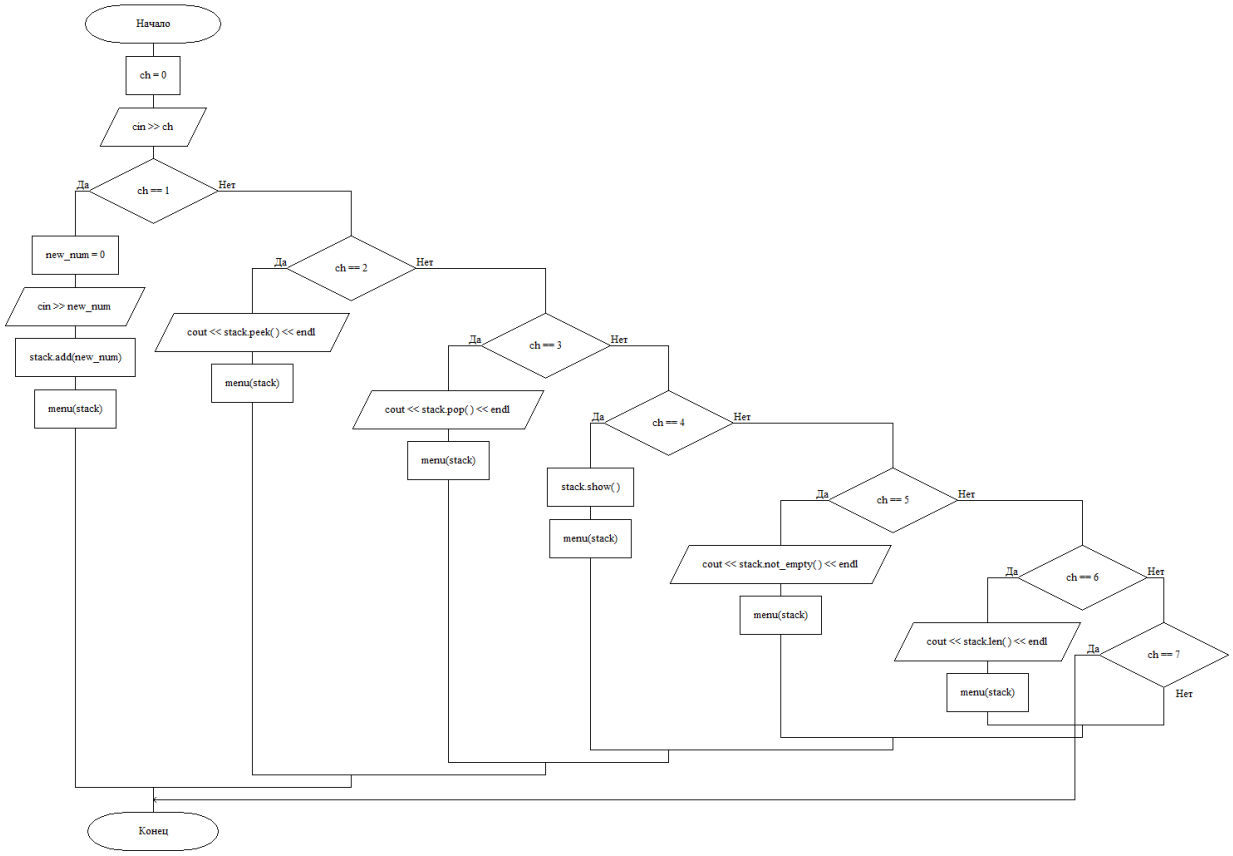


Рис.6 Схема алгоритма функции menu

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Stack.h"
#include <iostream>

void menu(Stack stack) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить элемент." << endl;
	cout << "[2] - Вывести верхний элемент (без удаления)." << endl;
	cout << "[3] - Вывести верхний элемент (с удалением)." << endl;
	cout << "[4] - Вывести стек на экран." << endl;
	cout << "[5] - Проверить пустой ли стек." << endl;
	cout << "[6] - Вывести длинну стека." << endl;
	cout << "[7] - Завершить программу." << endl;
	cout << "----> ";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
		int new_num = 0;
		cout << "Введите число: ";
		cin >> new_num;
		stack.add(new_num);
		cout << "Элемент "<<new_num<<" добавлен." << endl;
		menu(stack);
	}
	else if(ch == 2)
	{
		cout<<stack.peek()<<endl;
		menu(stack);
	}
	else if (ch == 3)
	{
		cout << stack.pop() << endl;
		menu(stack);
	}
	else if (ch == 4)
	{
		stack.show();
		menu(stack);
	}
	else if (ch == 5)
	{
		cout << stack.not_empty()<<endl;
		menu(stack);
	}
	else if (ch == 6)
	{
		cout<<stack.len()<<endl;
		menu(stack);
	}
	else if (ch == 7)
	{
		cout << "Программа завершена.";
		return;
	}
```

```
}


int main()
{
	setlocale(LC_ALL, "Russian");
	Stack *st = new Stack;
	menu(*st);

}
```

Stack.h

```
#include <iostream>
using namespace std;
#pragma once

struct Node
{
	int num;
	Node* next = NULL;
};

class Stack
{
private:
	Node* head = NULL, * tail = NULL;
public:
	void add(int x);
	int pop();
	void show();
	int peek();
	bool not_empty();
	int len();

};
```

Stack.cpp

```
#include "Stack.h"
void Stack::add(int x) {
	Node* temp = new Node;
	temp->num = x;
	if (head != NULL) {
		temp->next = head;
		head = temp;
	}
	else head = tail = temp;
}

int Stack::pop() {
```

```
	int out = head->num;
	head = head->next;
	return out;
}

void Stack::show() {
	Node* temp = new Node;
	temp = head;
	while (temp != NULL) {
		cout << temp->num << endl;
		temp = temp->next;
	}
}

int Stack::peek() {
	return head->num;
}

bool Stack::not_empty() {
	return head != NULL;
}

int Stack::len() {
	Node* temp = new Node;
	temp = head;
	int counter = 0;
	while (temp != NULL) {
		counter++;
		temp = temp->next;
	}
	return counter;
}
```

4. Тестирование программы

Ниже представлен результат работы программы с введённым стеком

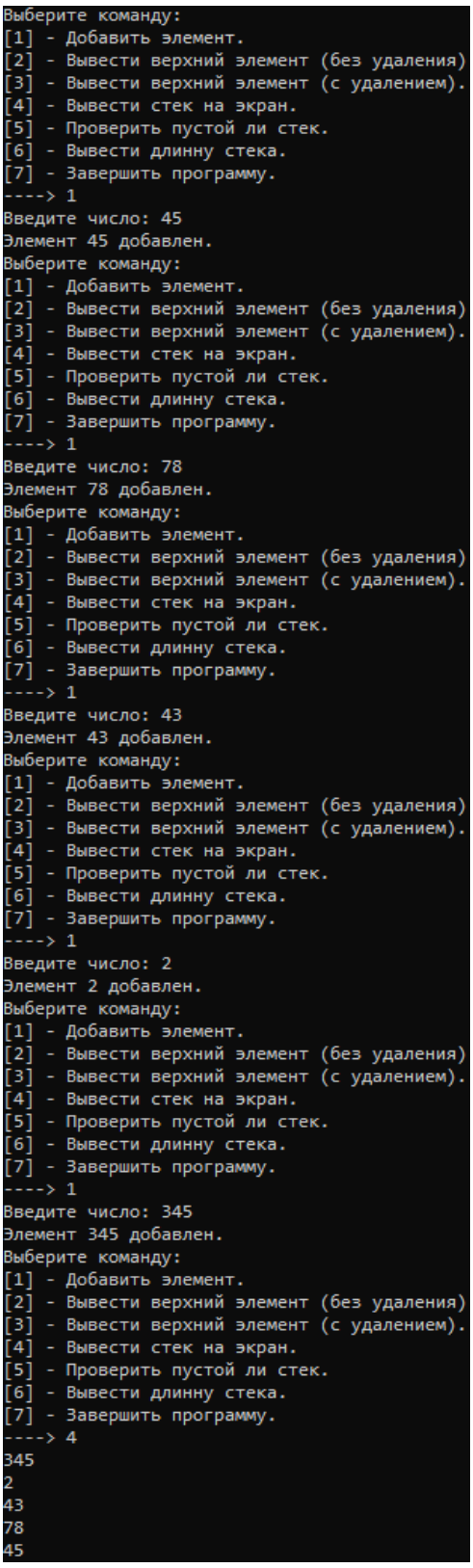


Рис.7 Скриншот добавления элементов в стек и вывод стека

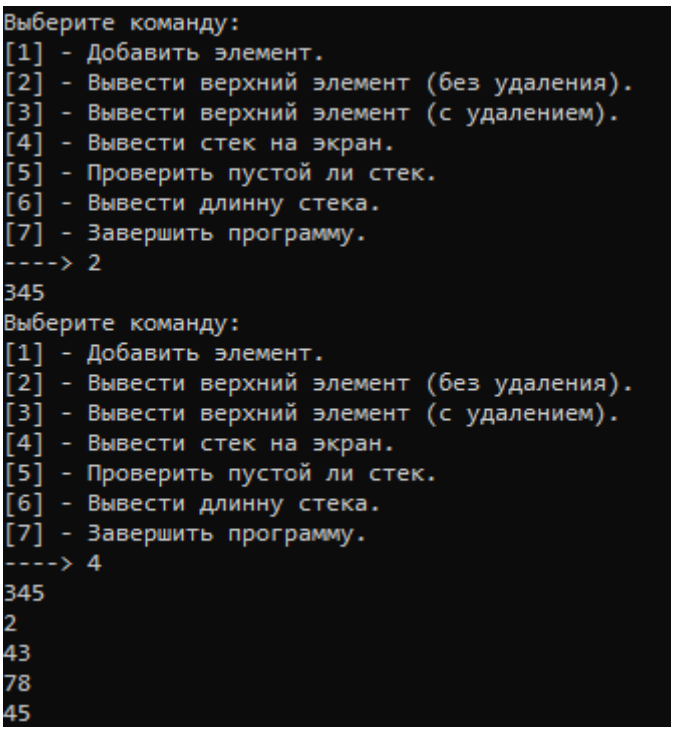


Рис.8 Скриншот вывода верхнего элемента (без удаления) и всего стека

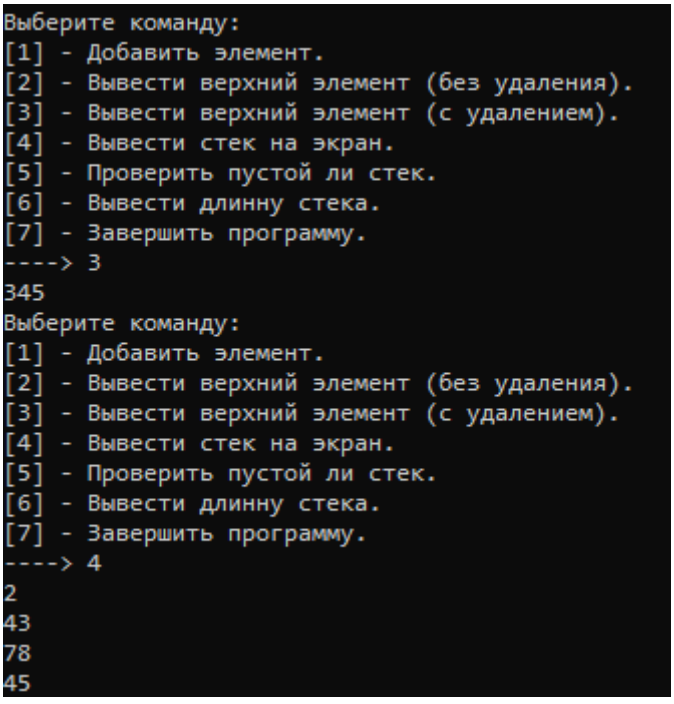


Рис.9 Скриншот вывода верхнего элемента (с удалением) и вывод стека

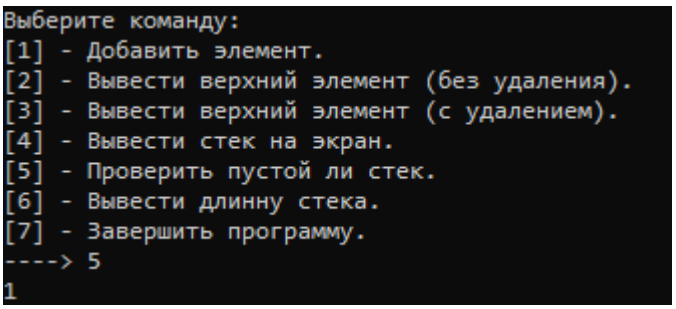


Рис.10 Скриншот проверки стека на пустоту

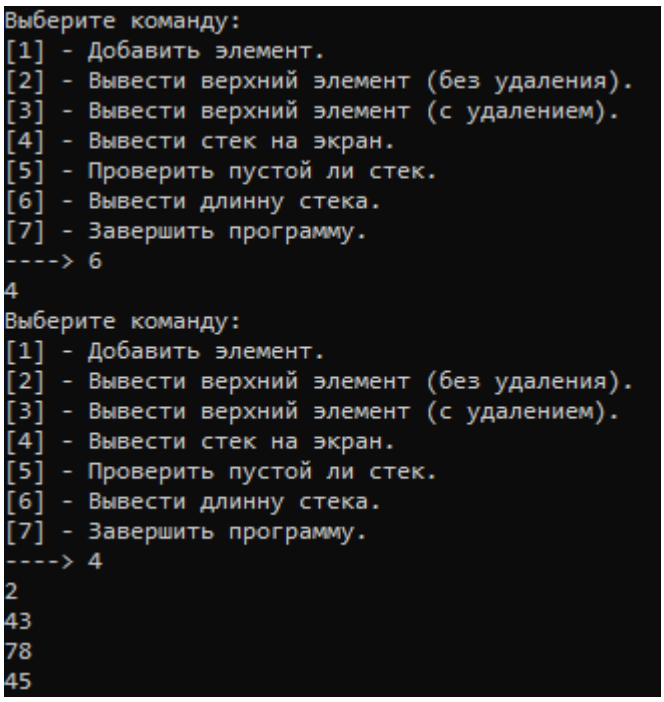


Рис.11 Скриншот вывода длинны стека и всего стека

5. Выводы

1. В ходе работы был создан стек на основе односвязного списка, ссылки между элементами которого осуществляются при помощи указателей на объекты класса Node.
2. Также были реализованы функции работы со стеком: добавление элементов, удаление, вывод всех элементов списка, вывод длины.
3. Были изучены положительные и негативные стороны стека:
4. Преимущества: стек может хранить неограниченное количество данных любого типа.
5. Недостатки: элементы можно извлекать только сверху, что усложняет взаимодействие с элементами.
6. Таким образом, была изучена работа стека на базе линейного связного списка и функций работы над ними и их реализация.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание Стека. интернет-ресурс: https://ru.wikipedia.org/wiki/Стек (Дата обращения – 02.11.2020).

Практическая работа № 4

ЛИНЕЙНЫЕ ДВУСВЯЗНЫЕ СПИСКИ

Постановка задачи

Составить программу создания линейного двусвязного списка и реализовать основные алгоритмы работы с ЛДС, обеспечивающие следующие действия:

1 - Добавить элемент
2 - Удалить элемент
2.1 – Последний элемент
2.3 – Заданный элемент
3 – Проверить наличие элемента
6 - Вычислить длину ЛОС
7 - Вывести (распечатать) ЛОС на экран

Перечисленные действия оформить в виде самостоятельных режимов работы созданного ЛДС. Выбор режимов производить с помощью пользовательского меню.

Провести полное тестирование (всех режимов работы) программы на стеке, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Оформить отчет с подробным описанием созданного ЛДС, принципов программной реализации алгоритмов работы с ЛДС, описанием текста исходного кода и проведенного тестирования программы.

Сделать выводы о проделанной работе, основанные на полученных результатах.

6. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add**– функция добавления элемента в ЛДС
- **void del_last** – функция удаления последнего элемента
- **bool in_lds** – функция проверки наличия элемента в ЛДС
- **void show –** функция вывода стека на экран
- **void del** – функция удаления выбранного элемента
- **int len –** функция вычисления длинны стека
- **Node* find –** функция возвращающая элемент по его значению

Двусвязный список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и две ссылки («связки») на следующий и предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями. Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. Как и односвязный список, двусвязный допускает только последовательный доступ к элементам, но при этом дает возможность перемещения в обе стороны. В этом списке проще производить удаление и перестановку элементов, так как легко доступны адреса тех элементов списка, указатели которых направлены на изменяемый элемент.

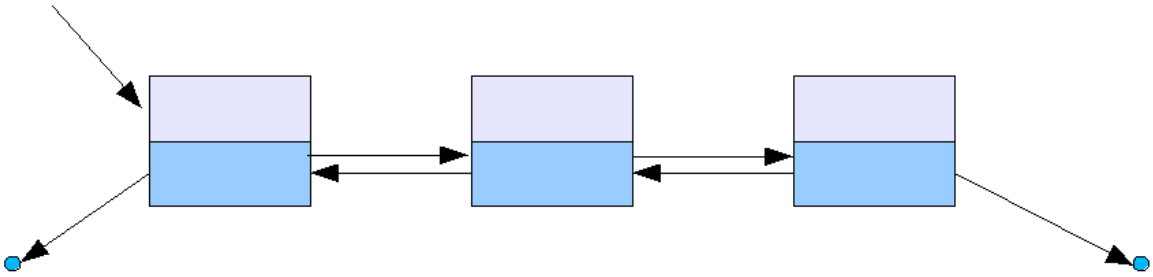


Рис.1 Принцип работы ЛДС

Функция main создает класс Lds и вызывает функцию menu.

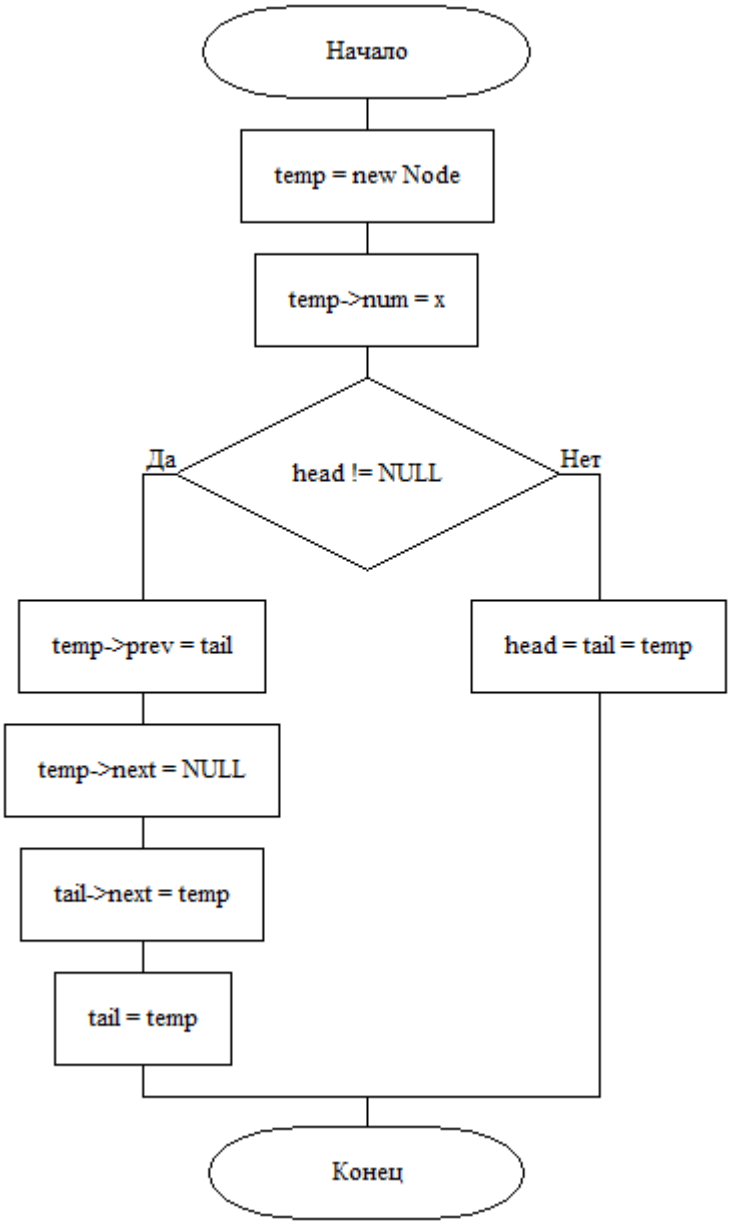


Рис.2 Схема алгоритма функции add

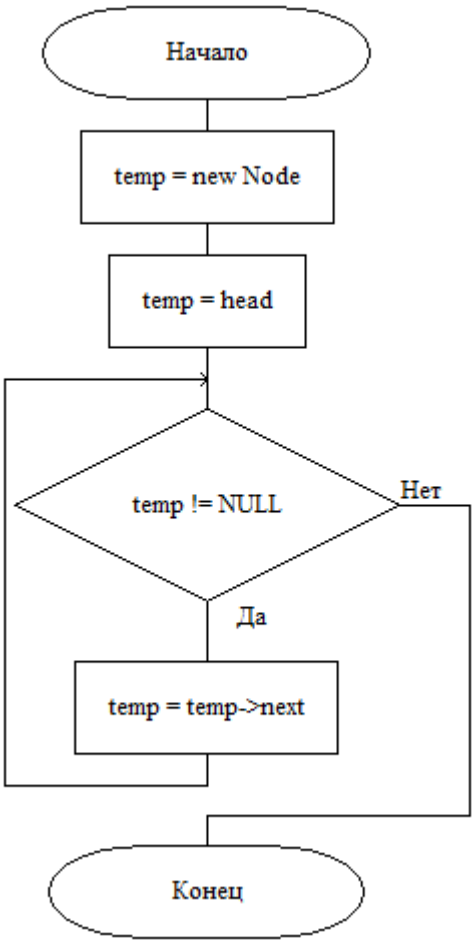


Рис.3 Схема алгоритма функции show

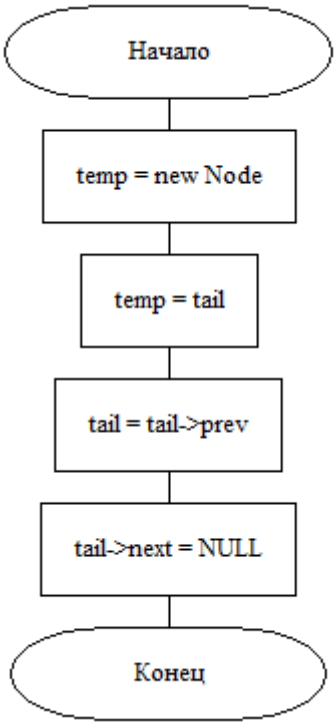


Рис.4 Схема алгоритма функции del_last

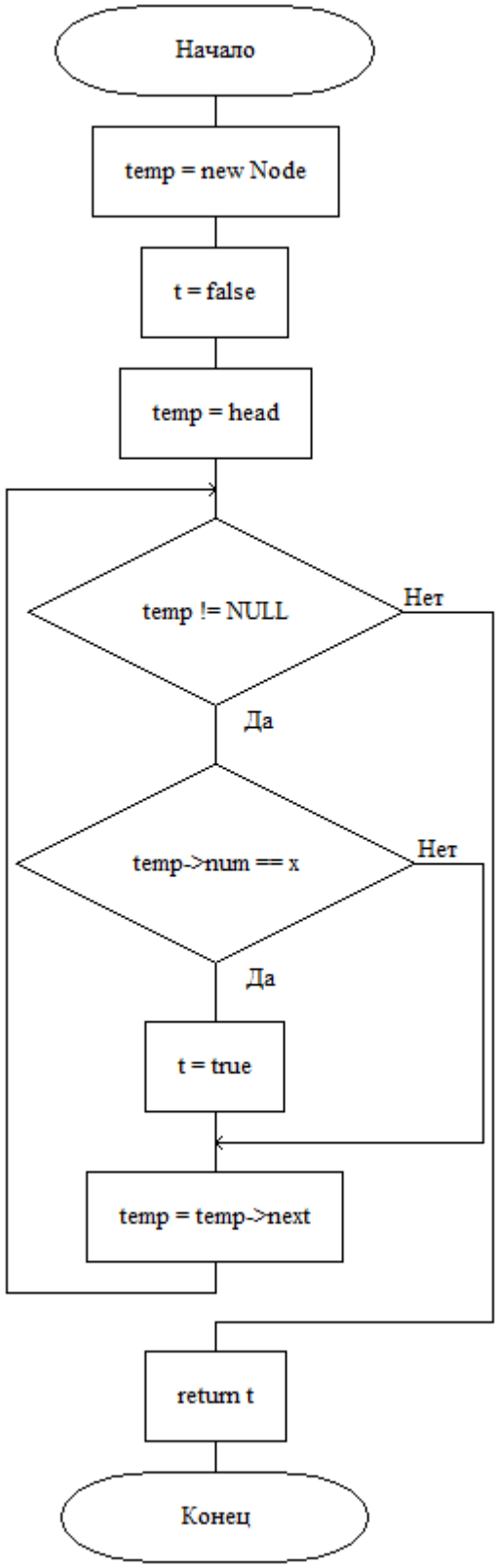


Рис.5 Схема алгоритма функции in_lds

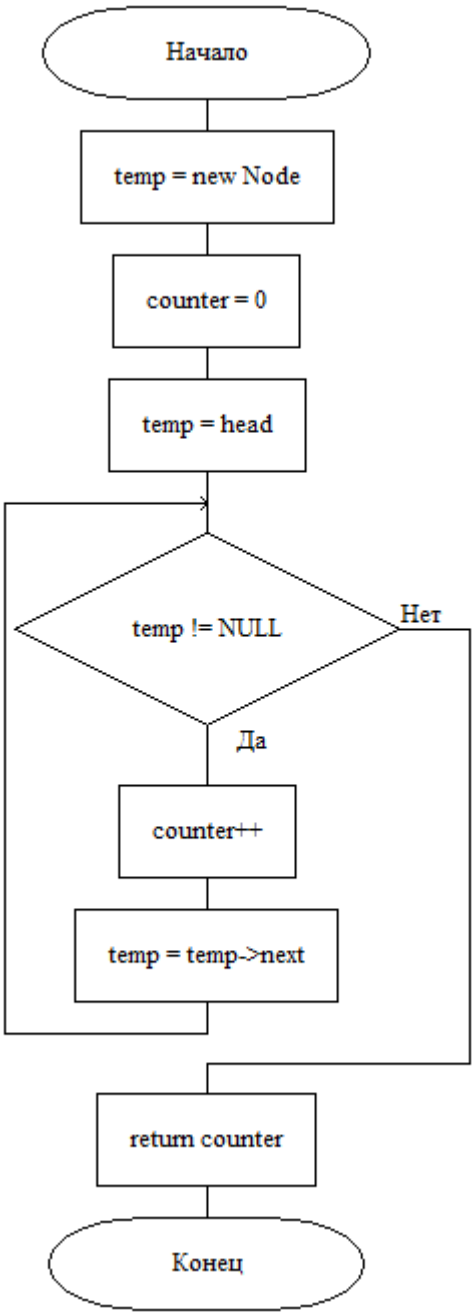


Рис.6 Схема алгоритма функции len

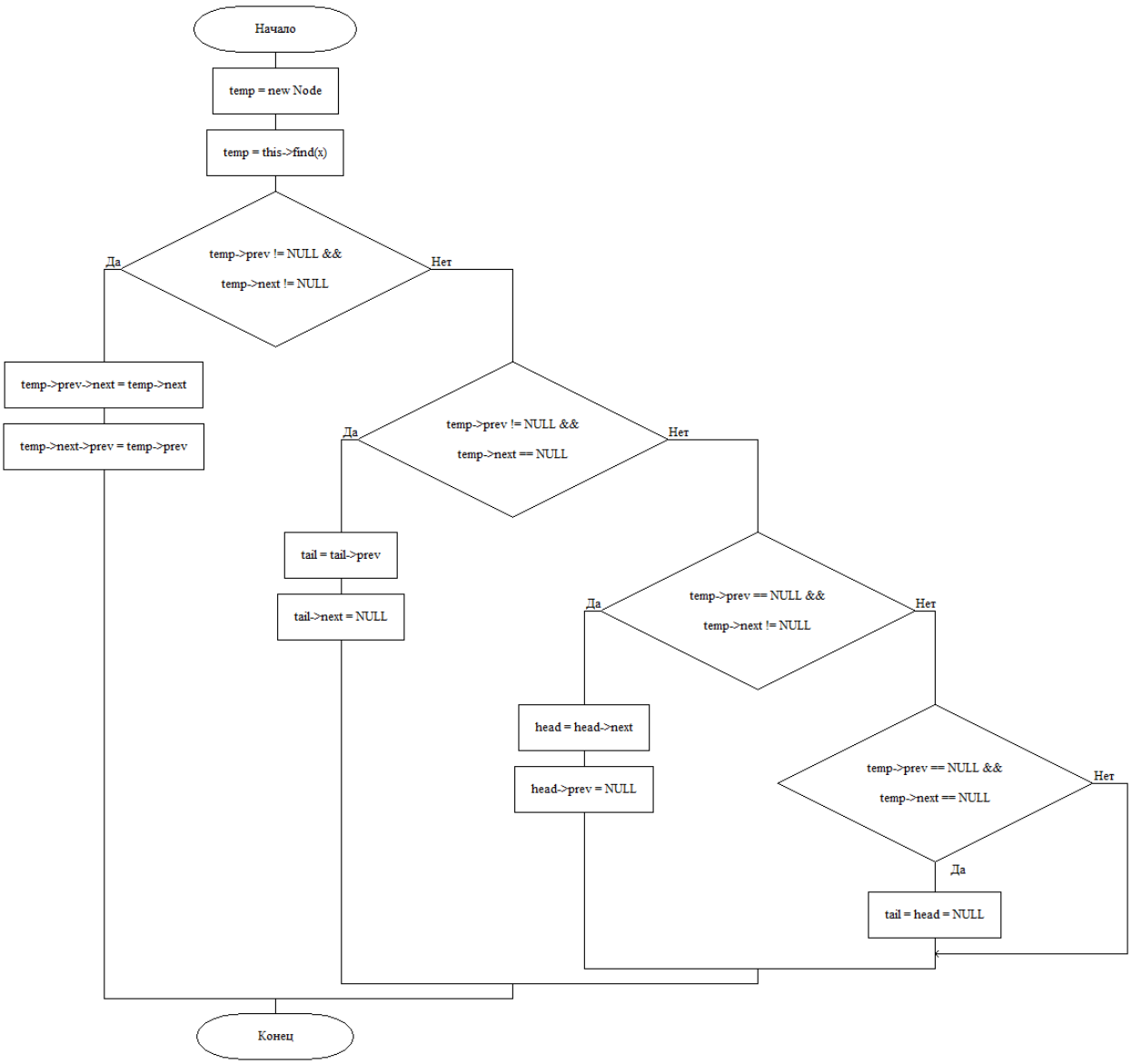


Рис.7 Схема алгоритма функции del

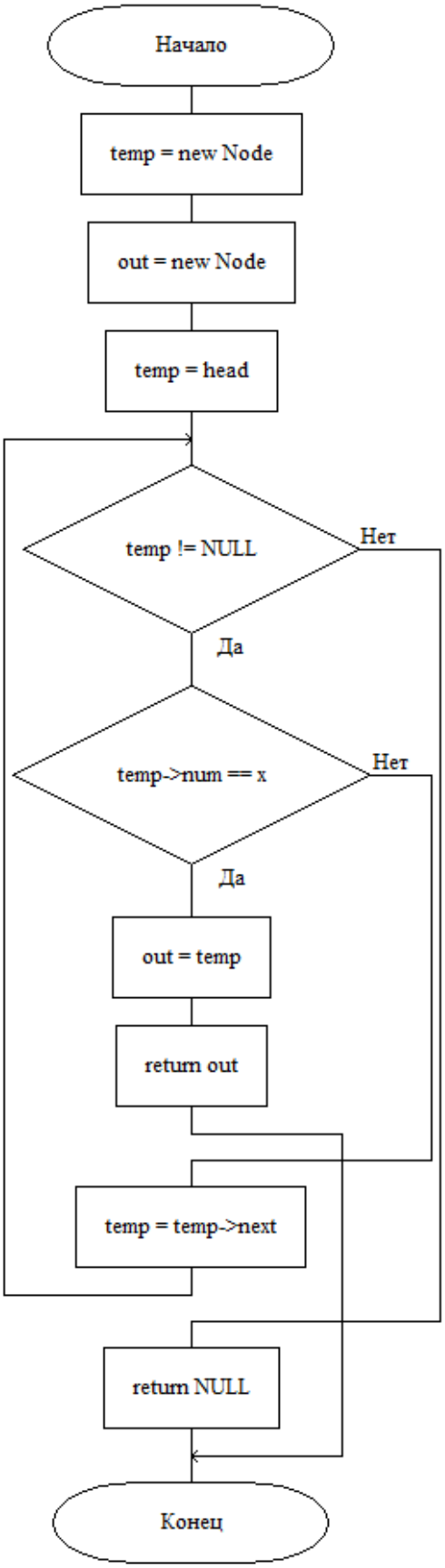

Начало
temp = new Node
out = new Node
temp = head
temp != NULL
Нет
Да
temp->num == x
Нет
Да
out = temp
return out
temp = temp->next
return NULL
Конец

Рис.8 Схема алгоритма функции find

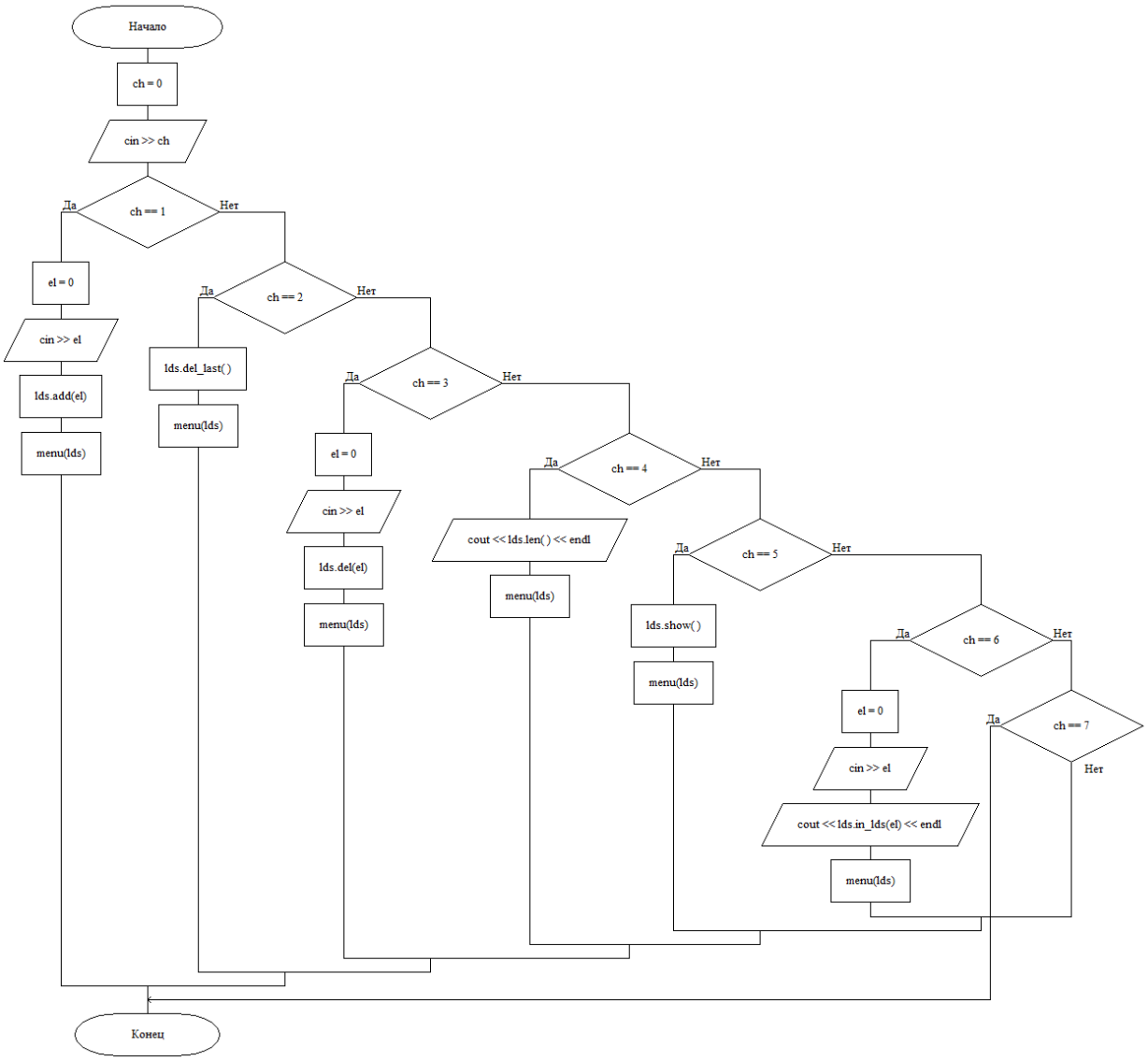


Рис.9 Схема алгоритма функции menu

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Lds.h"

void menu(Lds lds) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить элемент." << endl;
	cout << "[2] - Удалить последний элемент." << endl;
	cout << "[3] - Удалить элемент по значению." << endl;
	cout << "[4] - Длинна списка." << endl;
	cout << "[5] - Вывести список." << endl;
	cout << "[6] - Проверить наличие элемента в списке." << endl;
	cout << "[7] - Завершить программу." << endl;
	cout << "----> ";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
	 int el = 0;
	 cout << "Введите элемент:";
	 cin >> el;
	 lds.add(el);
	 cout << "Элемент "<<el<<" был добавлен в список." << endl;
		menu(lds);
	}
	else if (ch == 2)
	{
	 lds.del_last();
	 cout << "Последний элемент был удален" << endl;
		menu(lds);
	}
    else if (ch == 3)
    {

	 int el = 0;
	 cout << "Введите элемент:";
	 cin >> el;
	 lds.del(el);
	 cout << "Элемент " << el << " был удален из списка." << endl;
	 menu(lds);
    }
    else if (ch == 4)
    {
	 cout << "Длинна списка = "<<lds.len()<<endl;
	 menu(lds);
    }
    else if (ch == 5)
    {
	 lds.show();
	 menu(lds);
    }
    else if (ch == 6)
    {
	 int el = 0;
	 cout << "Введите элемент:";
	 cin >> el;
```

```
        cout << lds.in_lds(el) << endl;
        menu(lds);
    }
    else if (ch == 7)
    {

        return;
    }

}

int main()
{
    setlocale(LC_ALL, "Russian");
    Lds * lds = new Lds;
    menu(* lds);
}
```

Lds.h

```
#include <iostream>
#pragma once
using namespace std;

struct Node {
	int num;
	Node* prev = NULL;
	Node* next = NULL;
};

class Lds
{
private:
	Node *head = NULL, *tail = NULL;
public:
	void add(int x);
	void show();
	void del_last();
	bool in_lds(int x);
	int len();
	void del(int x);
	Node* find(int x);

};
```

Lds.cpp

```
#include "Lds.h"

void Lds::add(int x) {
	Node* temp = new Node;
	temp->num = x;
	if (head != NULL) {
		temp->prev = tail;
```

```
                temp->next = NULL;
                tail->next = temp;
                tail = temp;

        }
        else head = tail = temp;
}

void Lds::show() {
        Node* temp = new Node;
        temp = head;
        while (temp != NULL) {
                cout << temp->num<<endl;
                temp = temp->next;
        }

}

void Lds::del_last() {
        Node* temp = new Node;
        temp = tail;
        tail = tail->prev;
        tail->next = NULL;
        delete temp;

}

bool Lds::in_lds(int x) {
        Node* temp = new Node;
        bool t = false;
        temp = head;
        while (temp != NULL) {
                if (temp->num == x) {
                        t = true;
                }
                temp = temp->next;
        }
        return t;
}

int Lds::len() {
        Node* temp = new Node;
        int counter = 0;
        temp = head;
        while (temp != NULL) {
                counter++;
                temp = temp->next;
        }
        return counter;
}

void Lds::del(int x) {
        Node* temp = new Node;
        temp = this->find(x);
        if (temp->prev != NULL && temp->next != NULL) {
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
        }
        else if (temp->prev != NULL && temp->next == NULL) {
                tail = tail->prev;
                tail->next = NULL;
        }
        else if (temp->prev == NULL && temp->next != NULL) {
```

```
                head = head->next;
                head->prev = NULL;
        }
        else if (temp->prev == NULL && temp->next == NULL) {
                tail = head = NULL;
        }
        delete temp;
}

Node* Lds::find(int x) {
        Node* temp = new Node;
        Node* out = new Node;
        temp = head;
        while (temp != NULL) {
                if (temp->num == x) {
                        out = temp;
                        return out;
                }
                temp = temp->next;
        }
        return NULL;
        }
```

7. Тестирование программы

Ниже представлен результат работы программы с введённым списком

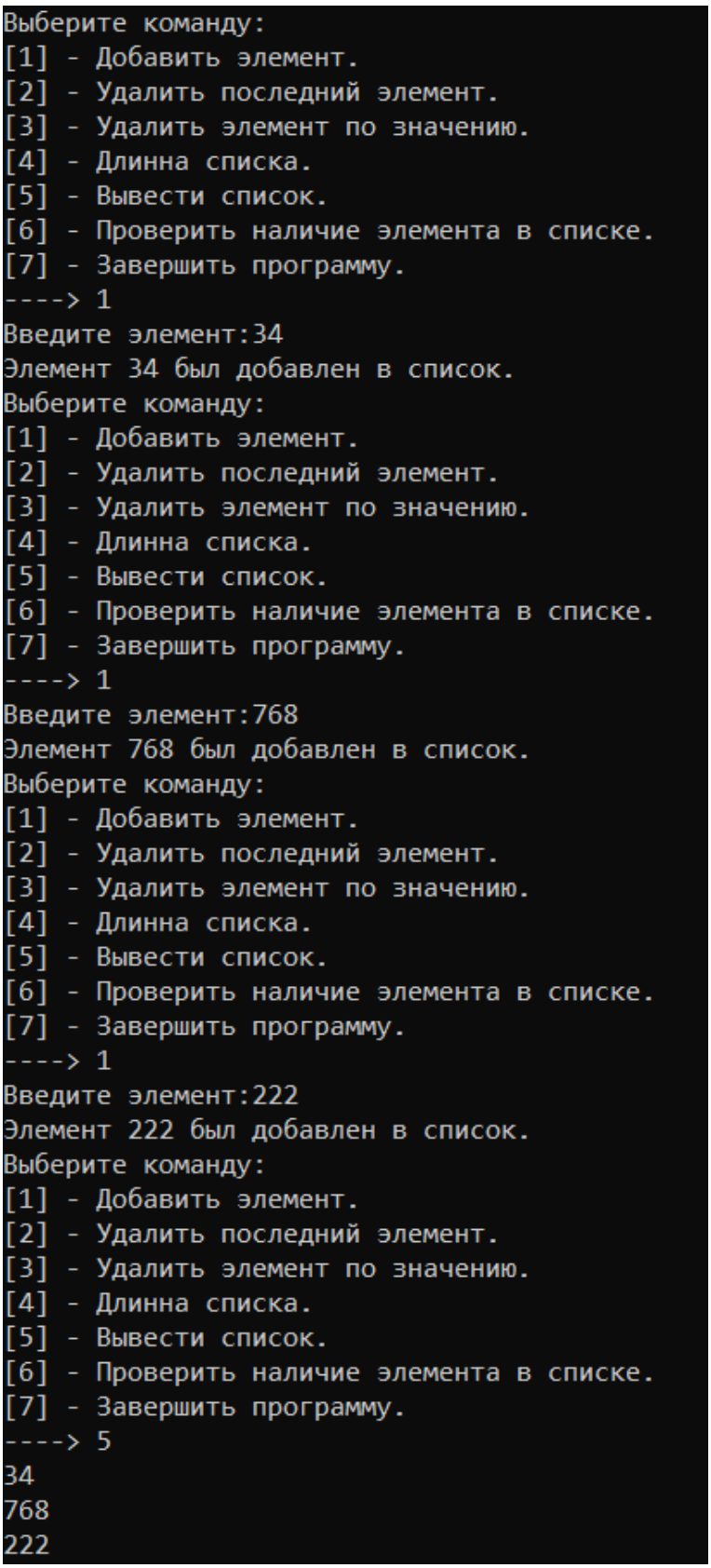


Рис.10 Скриншот добавления элементов в список и вывод списка

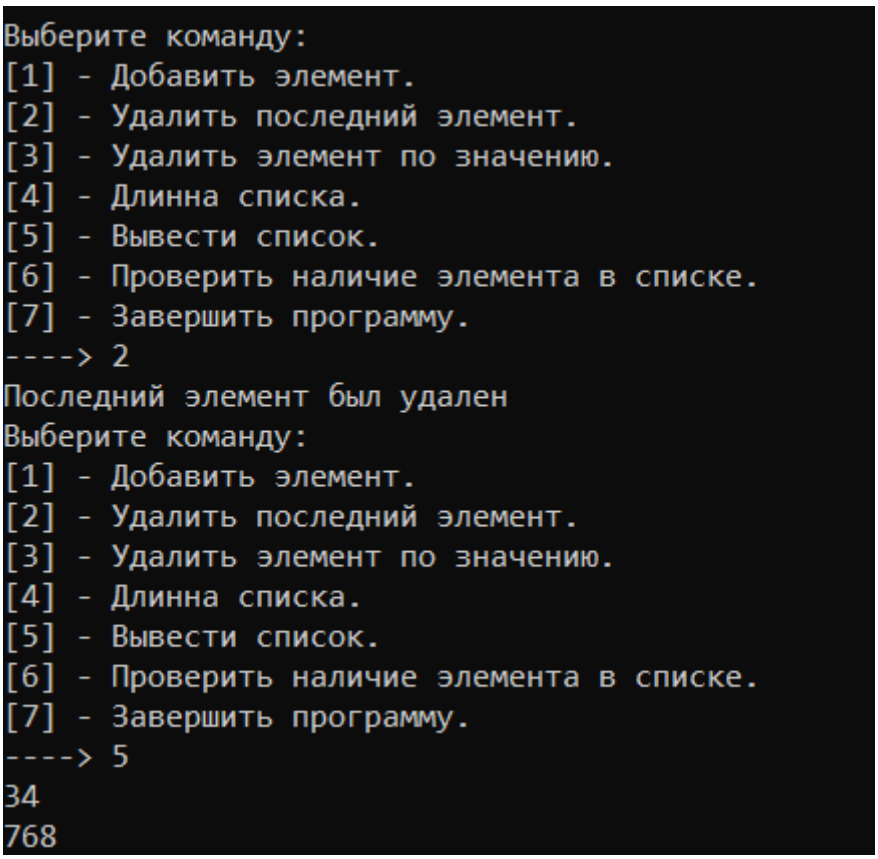


Рис.11 Скриншот удаления последнего элемента и вывод списка

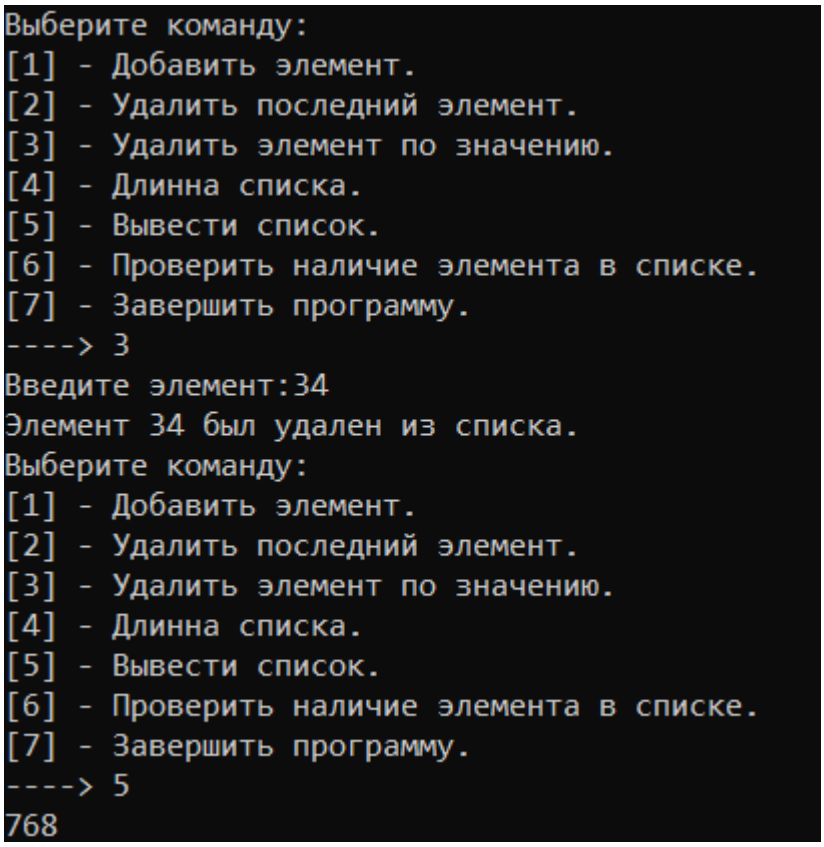


Рис.12 Скриншот удаления элемента по значению и вывод списка

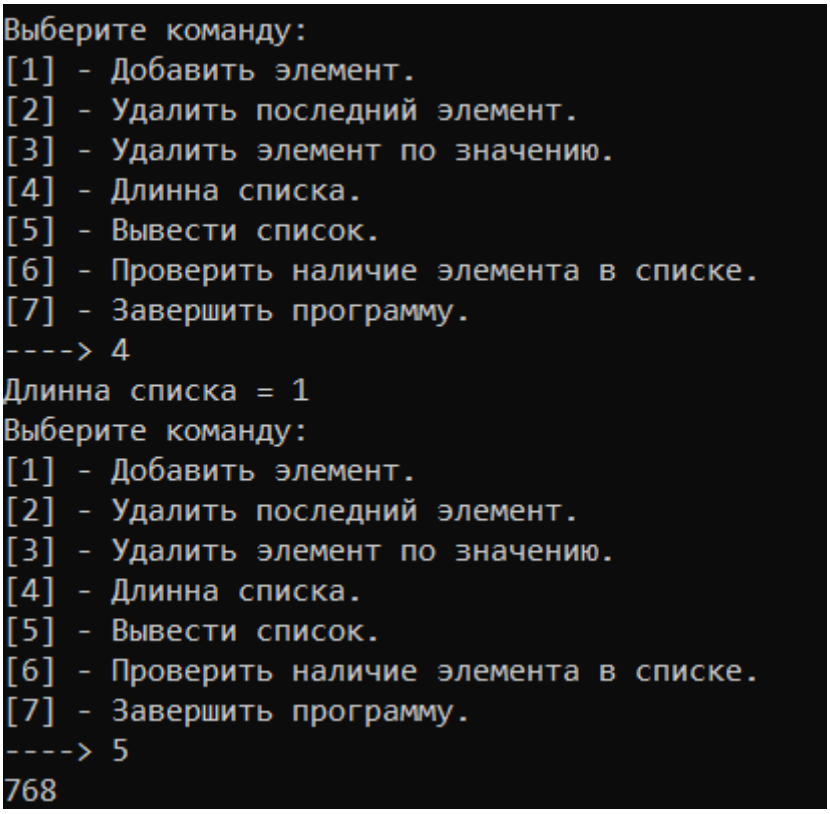


Рис.13 Скриншот вывода длинны и вывод списка

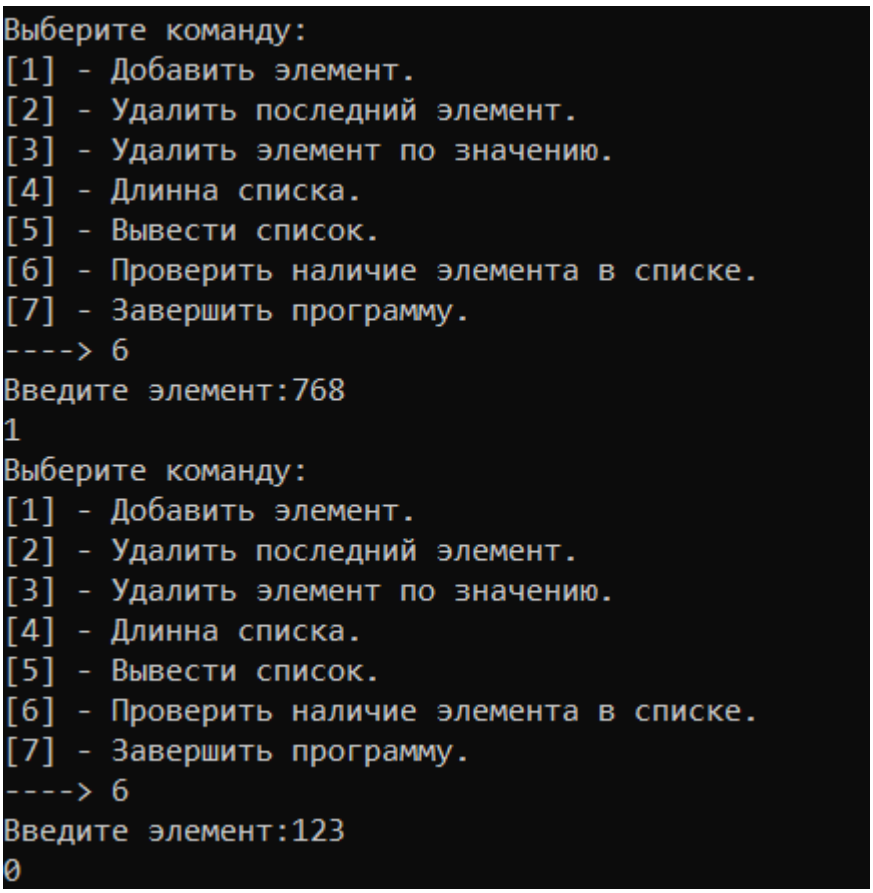


Рис.14 Скриншот поиска элемента по значению

8. Выводы

1. В ходе работы был создан линейный двусвязный список, ссылки между элементами которого осуществляются при помощи указателей на структуры Node.
2. Также были реализованы функции работы с ЛДС: добавление элементов, удаление, вывод всех элементов списка, вывод длины, поиск элементов.
3. Были изучены положительные и негативные стороны ЛДС:
4. Преимущества: ЛДС может хранить неограниченное количество данных любого типа.
5. Недостатки: Занимает больше памяти, чем обычный массив.
6. Таким образом, была изучена работа линейного двусвязного списка и функций работы над ними и их реализация.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание связных списков. интернет-ресурс: https://ru.wikipedia.org/wikiСвязный_список (Дата обращения 02.11.2020).

Практическая работа № 5

ДЕКИ

Постановка задачи

Составить программу создания дека на базе линейного двусвязного списка и реализовать основные алгоритмы работы с деком, обеспечивающие следующие действия:

1 - Добавить элемент
- 1.1 – Слева
- 1.2 – Справа

2 – Вывести элемент (с удалением)
- 2.1 – Слева
- 2.2 – Справа

3 – Вывести элемент (без удаления)
- 2.1 – Слева
- 2.2 – Справа

4 - Вычислить длину дека

5 - Вывести (распечатать) дек на экран

Перечисленные действия оформить в виде самостоятельных режимов работы созданного дека. Выбор режимов производить с помощью пользовательского меню.

Провести полное тестирование (всех режимов работы) программы на стеке, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Оформить отчет с подробным описанием созданного стека, принципов программной реализации алгоритмов работы со стеком, описанием текста исходного кода и проведенного тестирования программы.

Сделать выводы о проделанной работе, основанные на полученных результатах.

9. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add_left** – функция добавления элемента в дек слева
- **void add_right** – функция добавления элемента в дек справа
- **int pop_left** – функция извлечения элемента слева (удаление)
- **int pop_right** – функция извлечения элемента справа (удаление)
- **int peek_left** – функция возврата элемента слева (без удаления)
- **int peek_right** – функция возврата элемента справа (без удаления)
- **void show –** функция вывода дека на экран
- **int len –** функция вычисления длинны дека

Двусвязная очередь (жарг. дэк, дек от англ. deque — double ended queue; двусторонняя очередь, очередь с двумя концами) — абстрактный тип данных, в котором элементы можно добавлять и удалять как в начало, так и в конец. Может быть реализована при помощи двусвязного списка.

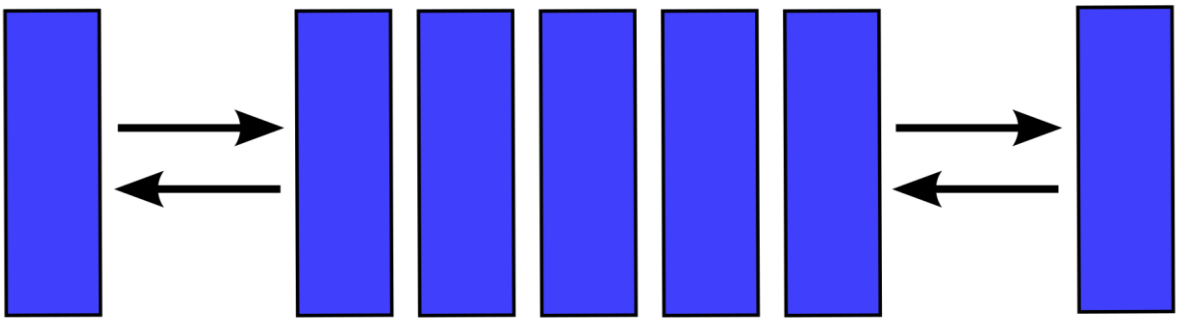


Рис.1 Принцип работы дека

Функция main создает класс Stack и вызывает функцию menu.

Функция peek_left возвращает значение левого элемента.

Функция peek_right возвращает значение правого элемента.

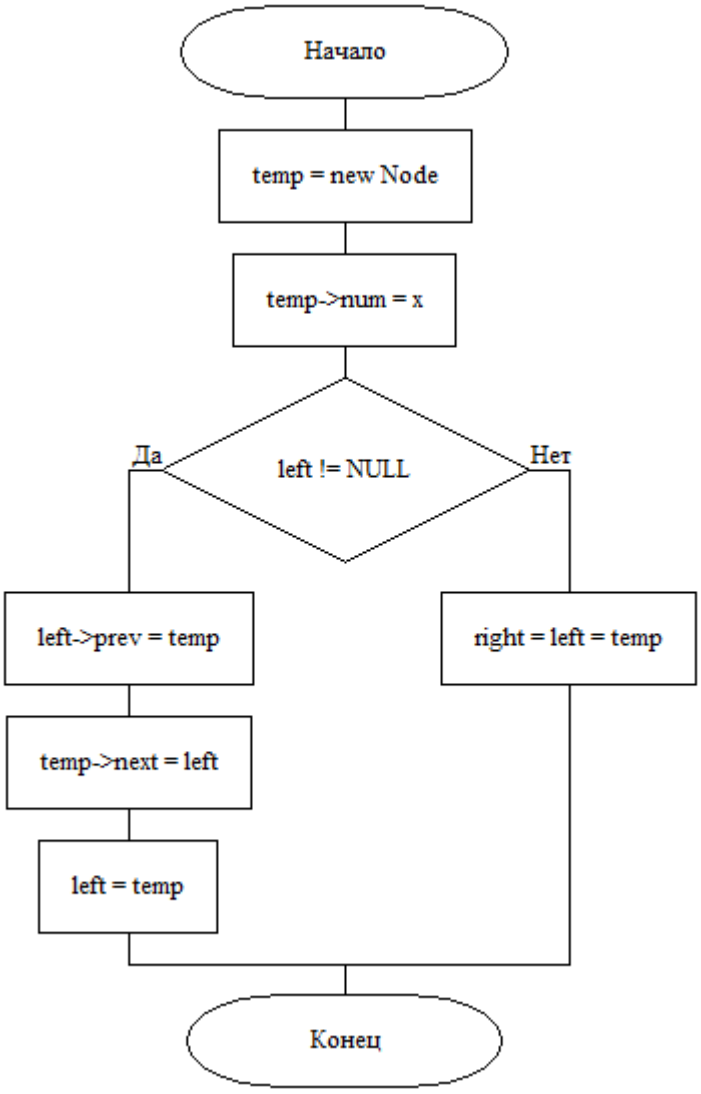


Рис.2 Схема алгоритма функции add_left

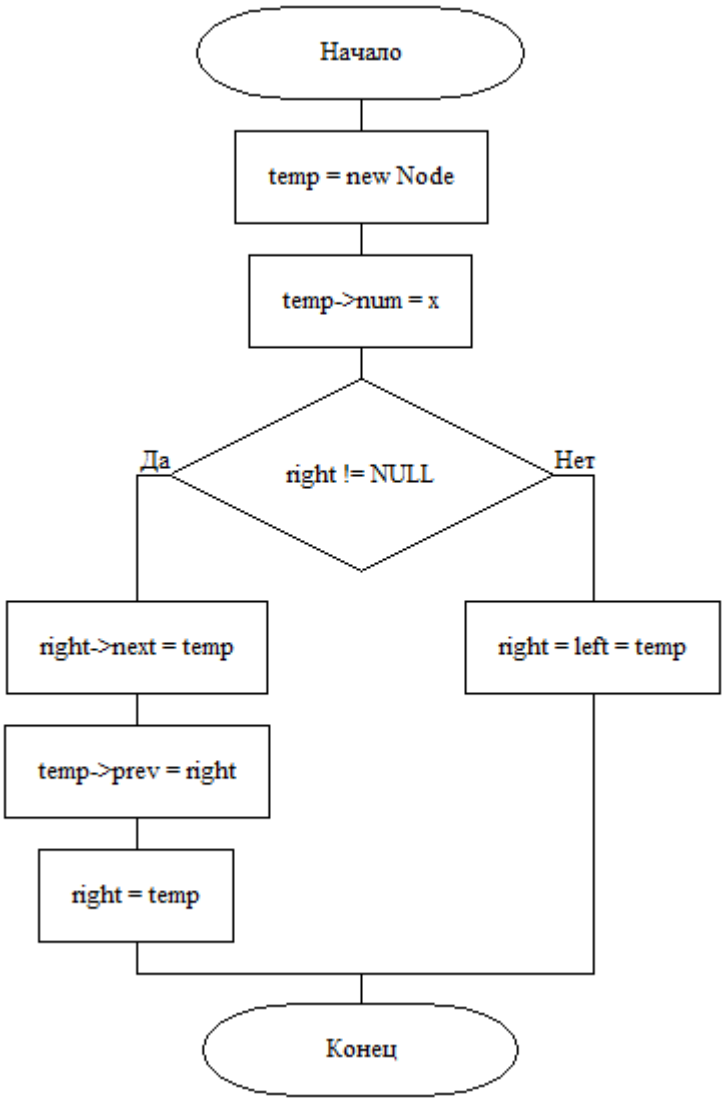


Рис.3 Схема алгоритма функции add_right

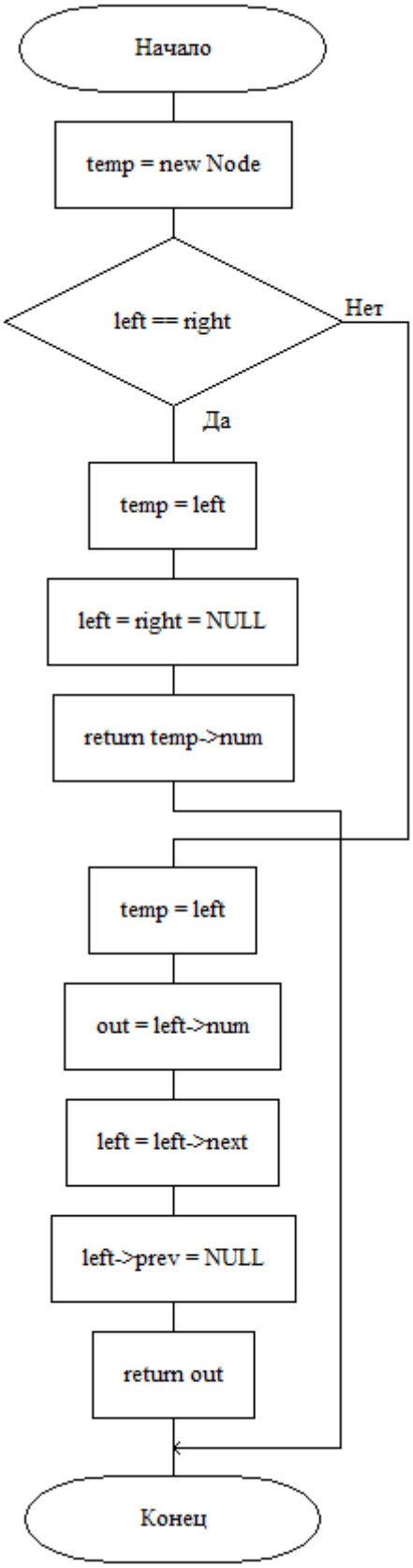


Рис.4 Схема алгоритма функции pop_left

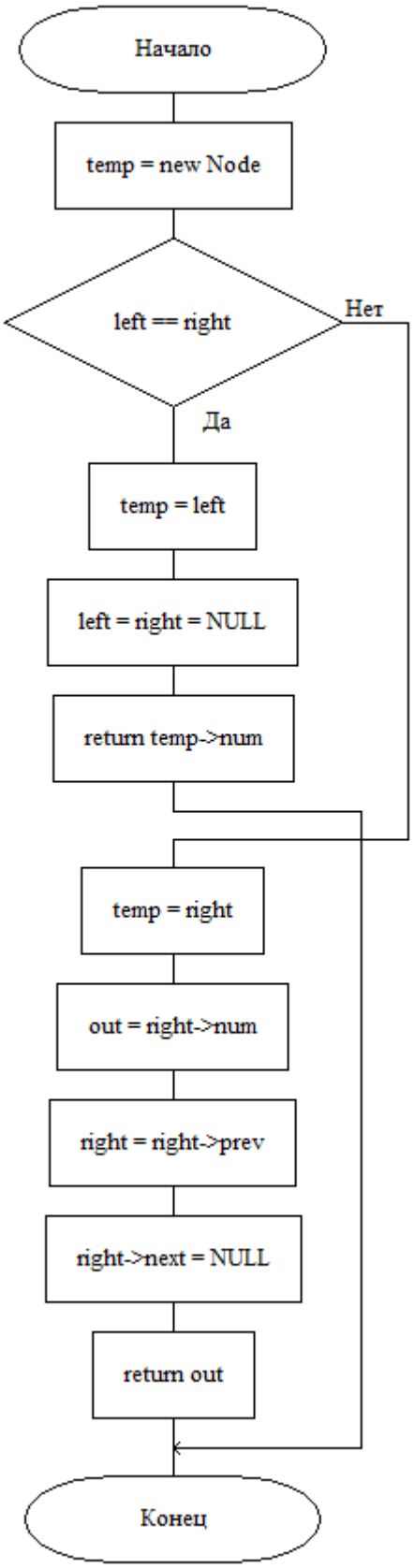


Рис.5 Схема алгоритма функции pop_right

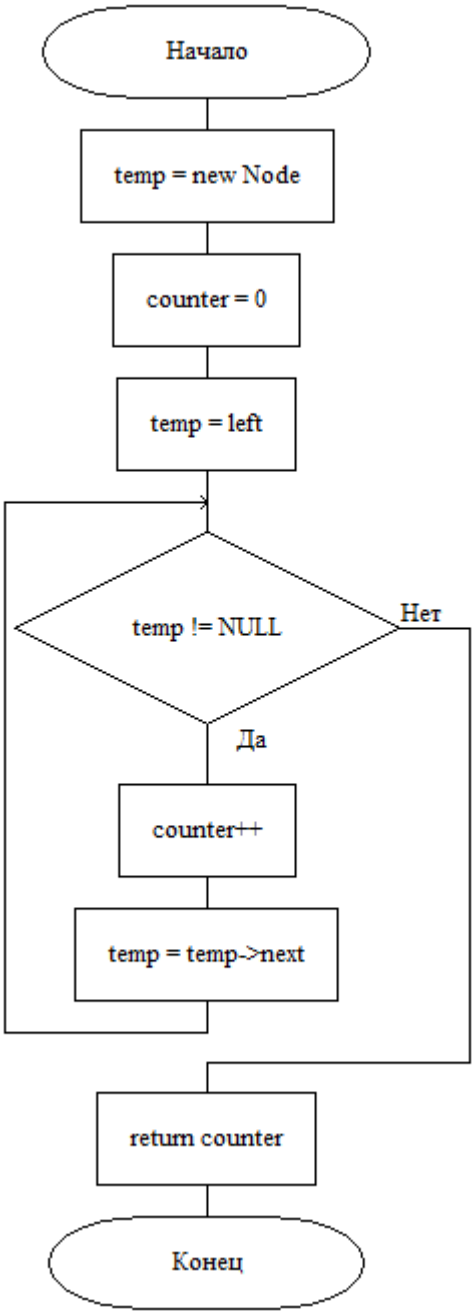


Рис.6 Схема алгоритма функции len

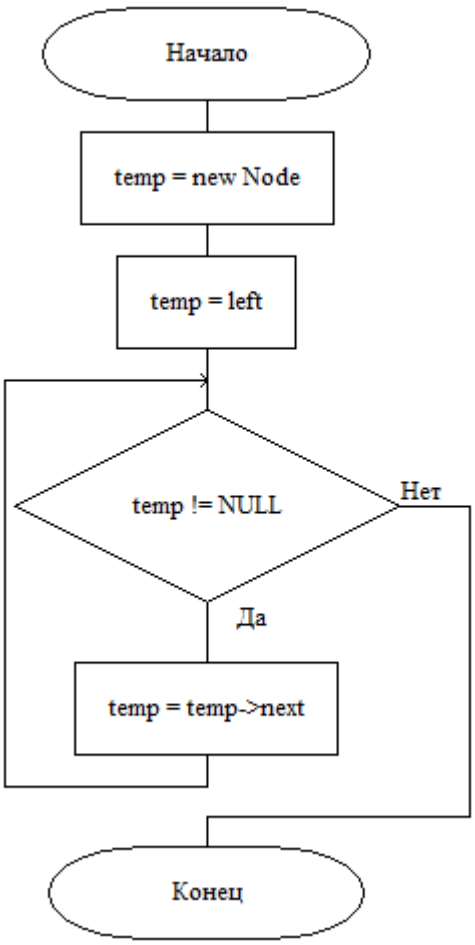


Рис.7 Схема алгоритма функции show

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Deque.h"
void menu(Deque dec) {
	cin.ignore();
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить элемент." << endl;
	cout << "[2] - Вывести элемент. (с удалением)" << endl;
```

```
cout << "[3] - Вывести элемент. (без удаления)" << endl;
cout << "[4] - Вывести длинну дека." << endl;
cout << "[5] - Вывести дек." << endl;
cout << "[6] - Завершить программу." << endl;
cout << "---->";
int ch = 0;
cin >> ch;
if (ch == 1) {
	cout << "____[1] - Добавить слева." << endl;
	cout << "____[2] - Добавить справа." << endl;
	cout << "____[3] - Назад." << endl;
	cout << "-------->";
	int ch2 = 0;
	cin >> ch2;
	if (ch2 == 1) {
		int x = 0;
		cout << "Введите число." << endl;
		cout << "-------->";
		cin >> x;
		dec.add_left(x);
		menu(dec);
	}
	else if (ch2 == 2) {
		int x = 0;
		cout << "Введите число." << endl;
		cout << "-------->";
		cin >> x;
		dec.add_right(x);
		menu(dec);
	}
	else if (ch2 == 3) {
		menu(dec);
	}

}
else if (ch == 2)
{
	cout << "____[1] - Вывести слева." << endl;
	cout << "____[2] - Вывести справа." << endl;
	cout << "____[3] - Назад." << endl;
	cout << "-------->";
	int ch2 = 0;
	cin >> ch2;
	if (ch2 == 1) {
		cout <<	dec.pop_left()<<endl;
		menu(dec);
	}
	else if (ch2 == 2) {
		cout << dec.pop_right() << endl;
		menu(dec);
	}
	else if (ch2 == 3) {
		menu(dec);
	}
}
else if (ch == 3)
{
	cout << "____[1] - Вывести слева." << endl;
	cout << "____[2] - Вывести справа." << endl;
	cout << "____[3] - Назад." << endl;
	cout << "-------->";
	int ch2 = 0;
	cin >> ch2;
```

```
			if (ch2 == 1) {
				cout << dec.peek_left() << endl;
				menu(dec);
			}
			else if (ch2 == 2) {
				cout << dec.peek_right() << endl;
				menu(dec);
			}
			else if (ch2 == 3) {
				menu(dec);
			}
		}
		else if (ch == 4)
		{
			cout << dec.len() << endl;
			menu(dec);
		}
		else if (ch == 5)
		{
			dec.show();
			menu(dec);
		}
		else if (ch == 6)
		{
			return;
		}
}

int main()
{
		setlocale(LC_ALL, "Russian");
		Deque deque;
		menu(deque);
}
```

Deque.h

```
#include <iostream>
#pragma once
using namespace std;

struct Node {
		int num = 0;
		Node* prev = NULL;
		Node* next = NULL;
};

class Deque
{
private:
		Node * left, * right;
public:
		Deque() : left(NULL), right(NULL) {}
		void add_left(int x);
		void add_right(int x);
		int pop_left();
		int pop_right();
		int peek_left();
```

```
	int peek_right();
	int len();
	void show();
	};
```

Deque.cpp

```
#include "Deque.h"

void Deque::add_left(int x) {
	Node* temp = new Node;
	temp->num = x;
	if (left != NULL) {
		left->prev = temp;
		temp->next = left;
		left = temp;
	}
	else {
		right = left = temp;
	}
}

void Deque::add_right(int x) {
	Node* temp = new Node;
	temp->num = x;
	if (right != NULL) {
		right->next = temp;
		temp->prev = right;
		right = temp;
	}
	else {
		right = left = temp;
	}
}

int Deque::pop_left() {
	Node* temp = new Node;
	if (left == right) {
		temp = left;
		left = right = NULL;
		return temp->num;
	}

	temp = left;
	int out = left->num;
	left = left->next;
	left->prev = NULL;
	delete temp;
	return out;
}

int Deque::pop_right() {
	Node* temp = new Node;
	if (left == right) {
		temp = left;
		left = right = NULL;
		return temp->num;
	}
	temp = right;
	int out = right->num;
```

```
        right = right->prev;
        right->next = NULL;
        delete temp;
        return out;
}

int Deque::peek_left() {
        return left->num;
}

int Deque::peek_right() {
        return right->num;
}

int Deque::len() {
        Node* temp = new Node;
        int counter = 0;
        temp = left;
        while (temp != NULL) {
                counter++;
                temp = temp->next;
        }
        return counter;
}

void Deque::show() {
        Node* temp = new Node;
        temp = left;
        while (temp != NULL) {
                cout << temp->num<<endl;
                temp = temp->next;
        }
        }
```

10.Тестирование программы

Ниже представлен результат работы программы с введённым деком

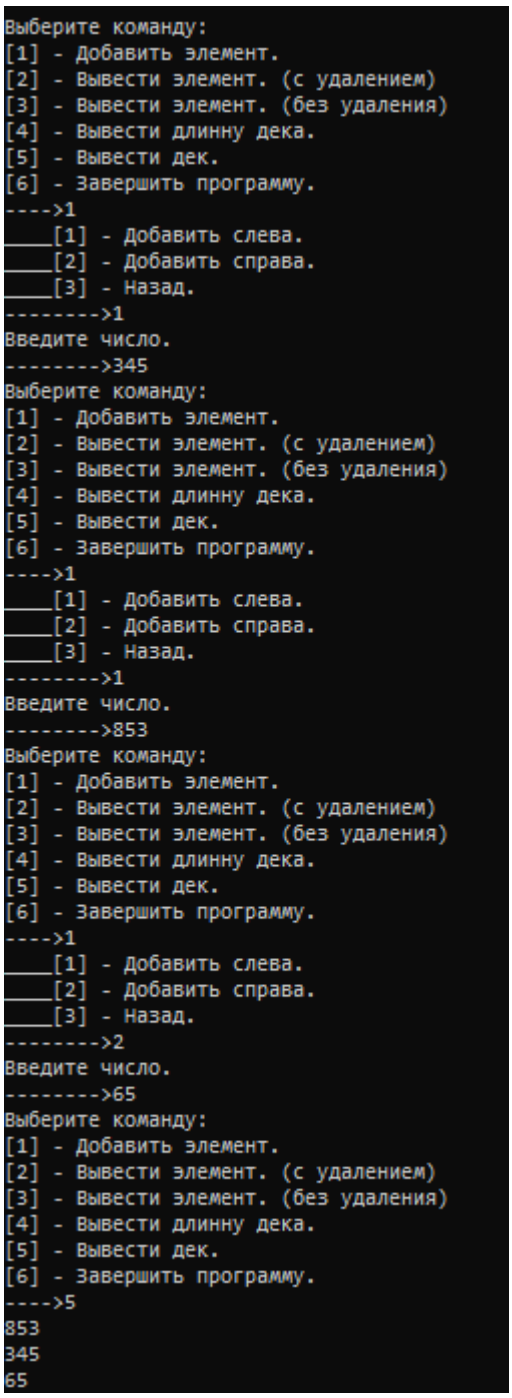


Рис.8 Скриншот добавления элементов в дек и вывод дека

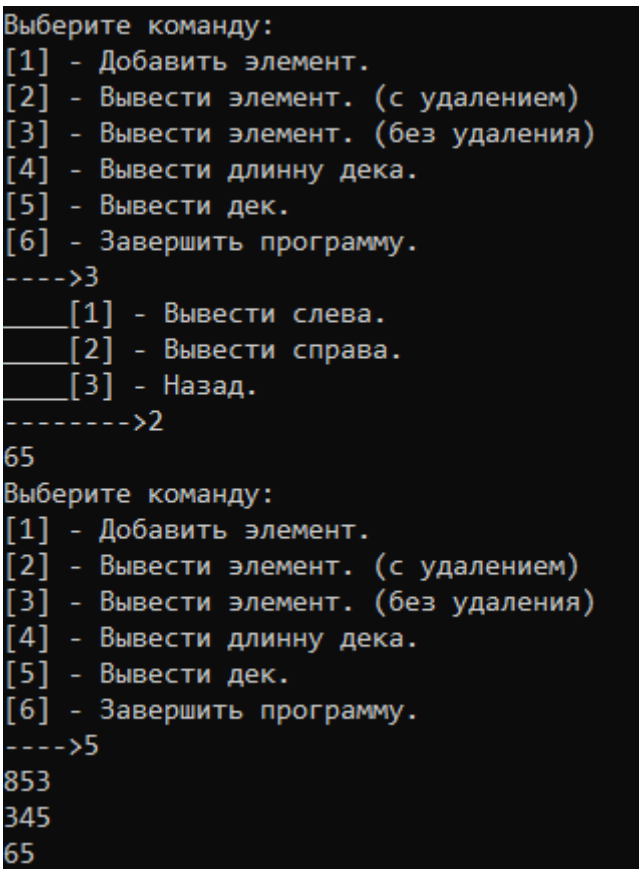


Рис.9 Скриншот вывода элемента справа без удаления и вывода дека

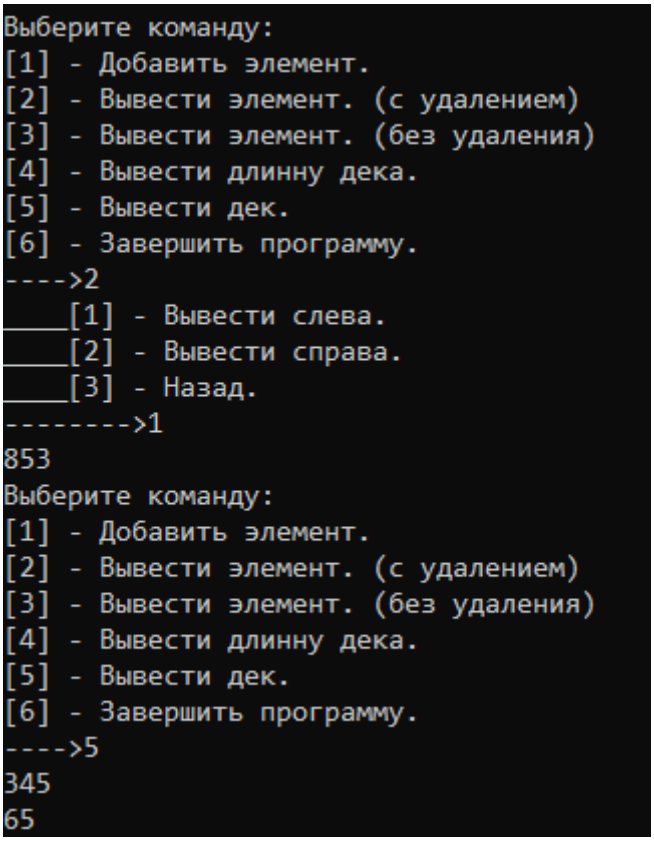


Рис.10 Скриншот вывода элемента справа с удалением и вывода дека

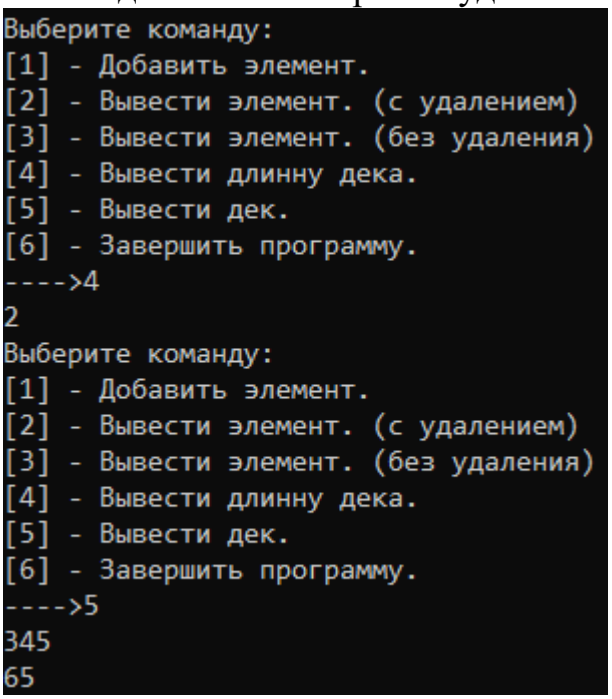


Рис.11 Скриншот вывода длинны дека и вывода дека

11.Выводы

1. В ходе работы был создан дек на основе двусвязного списка, ссылки между элементами которого осуществляются при помощи указателей на объекты класса Node.
2. Также были реализованы функции работы с деком: добавление элементов, удаление, вывод всех элементов списка, вывод длины.
3. Были изучены положительные и негативные стороны стека:
4. Преимущества: дек может хранить неограниченное количество данных любого типа.
5. Недостатки: элементы можно извлекать и вставлять только с концов дека, что усложняет взаимодействие с элементами.
6. Таким образом, была изучена работа дека на базе линейного двусвязного списка и функций работы над ними и их реализация.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание Дека. интернет-ресурс: https://ru.wikipedia.org/wiki/Двухсторонняя_очередь (Дата обращения – 02.11.2020).

Практическая работа № 6

БИНАРНОЕ ДЕРЕВО ПОИСКА

Вариант 22

Постановка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту. Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню. Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе. Сделать выводы о проделанной работе, основанные на полученных результатах. Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

12. Описание алгоритма

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

В данной практической реализован алгоритм двоичного дерева на языке C++. Распределение строк в дереве осуществляется путем их сравнения. Реализация прямого и обратного обхода выполнена рекурсивно.

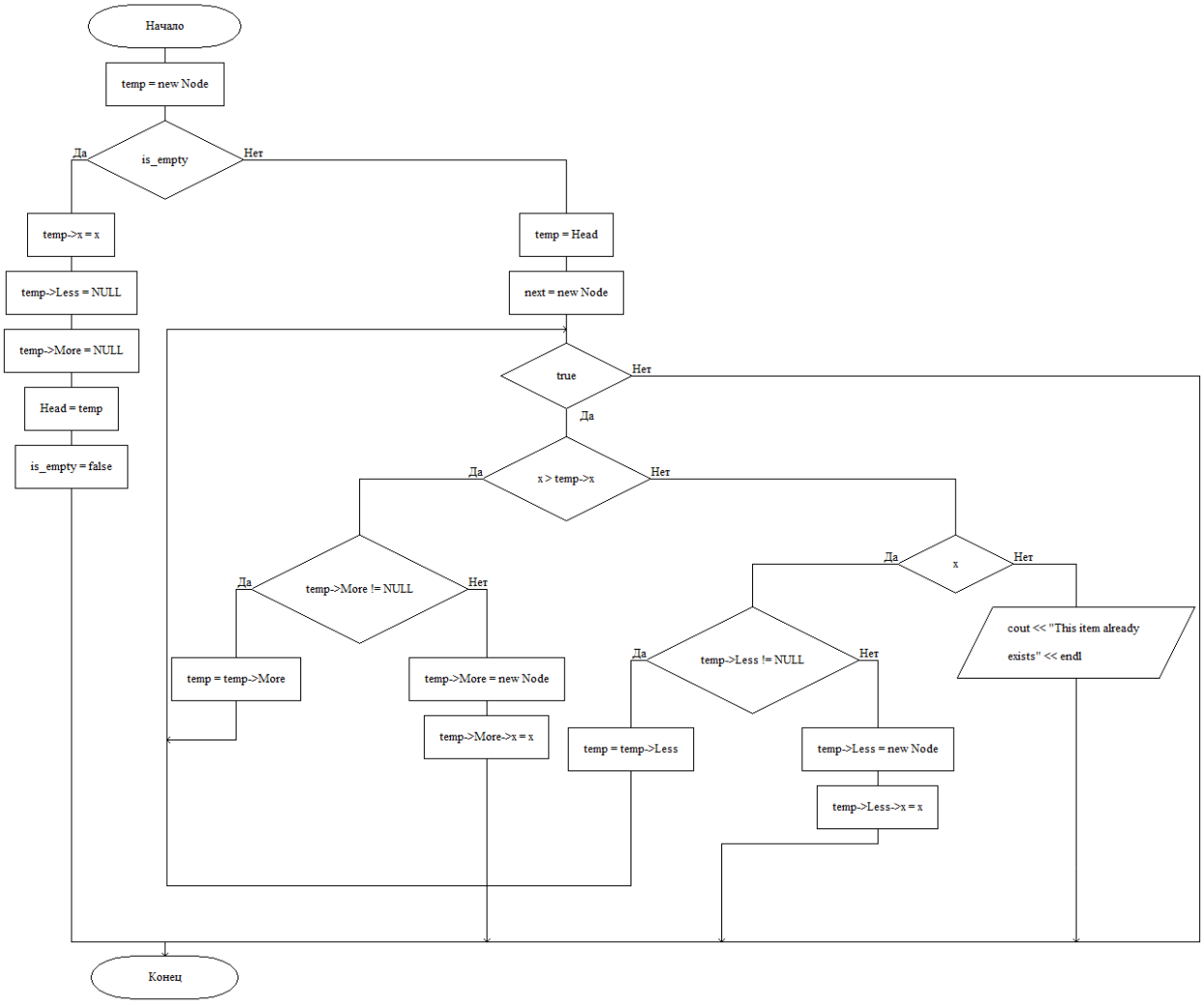


Рис.1 Схема алгоритма функции add

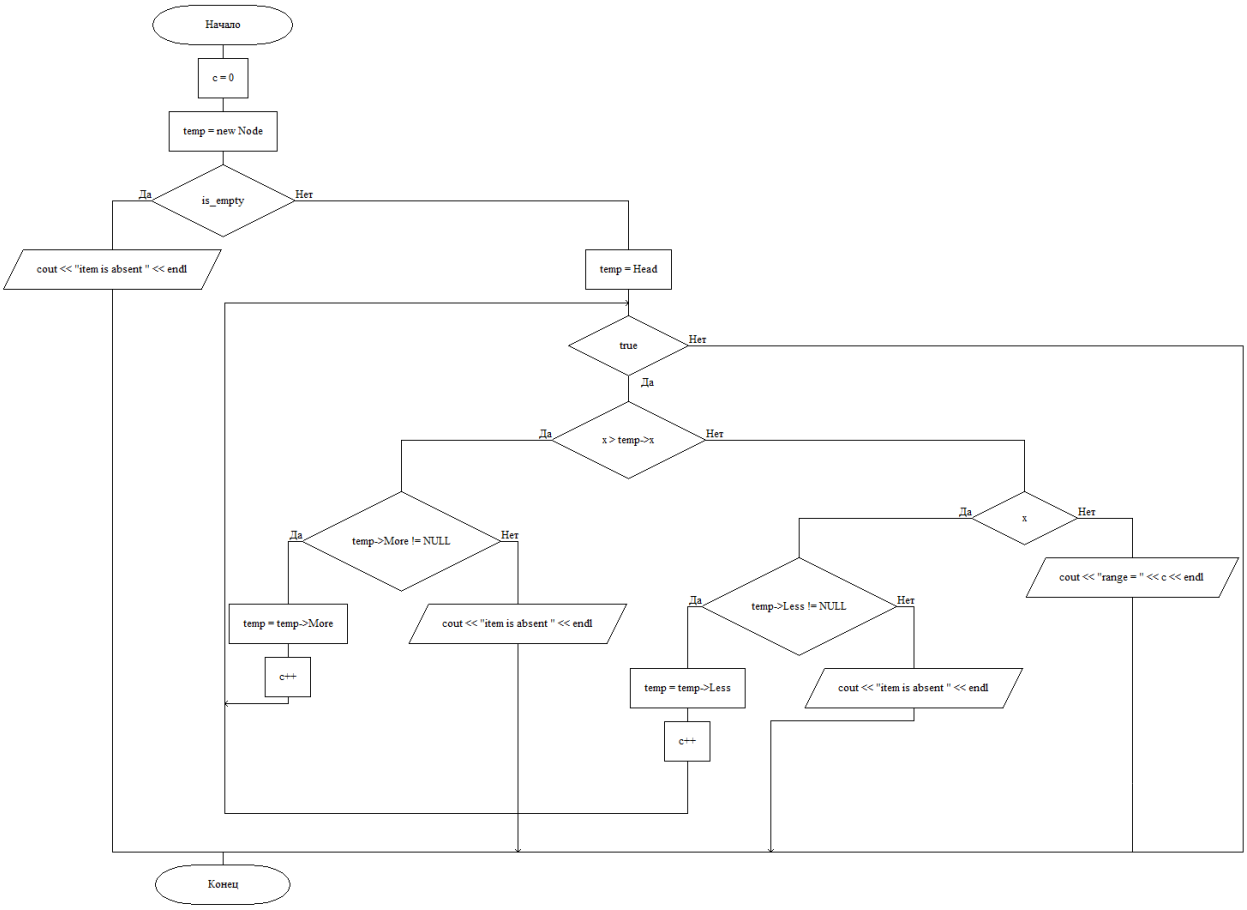


Рис.2 Схема алгоритма функции chek

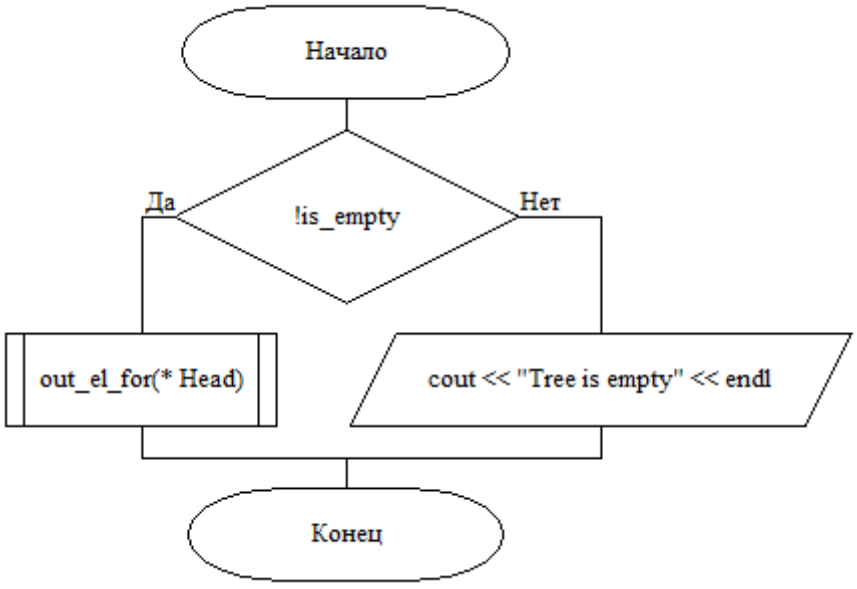


Рис.3 Схема алгоритма функции forward

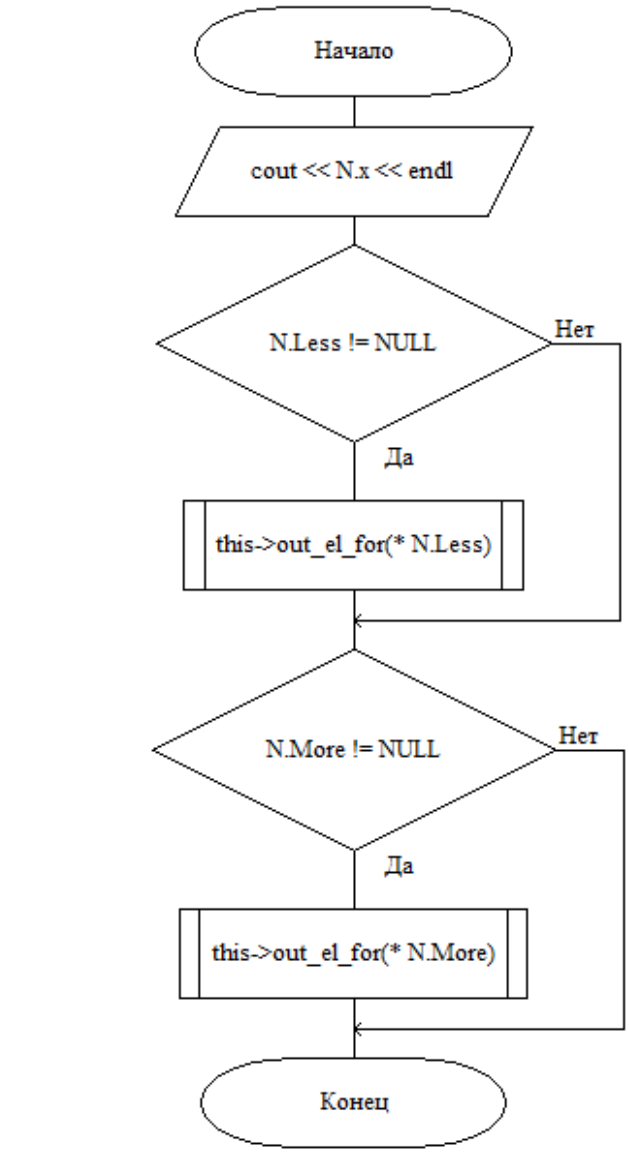


Рис.4 Схема алгоритма функции out_el_for

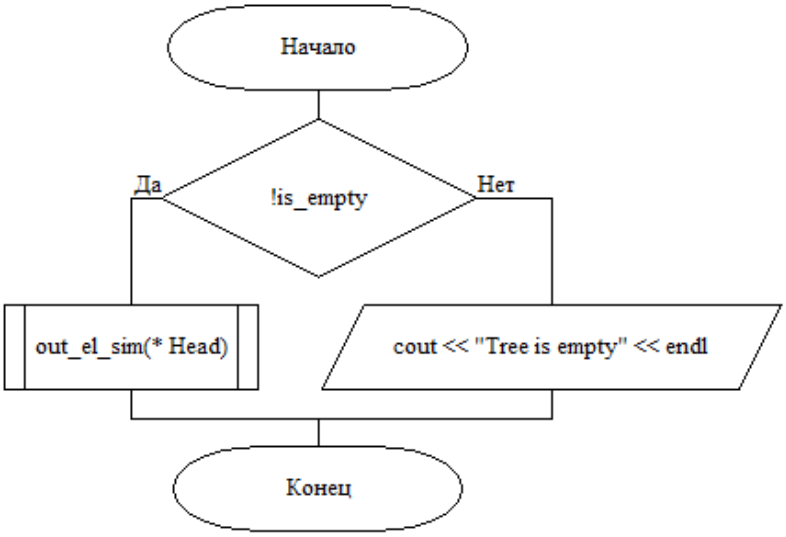


Рис.5 Схема алгоритма функции simetr

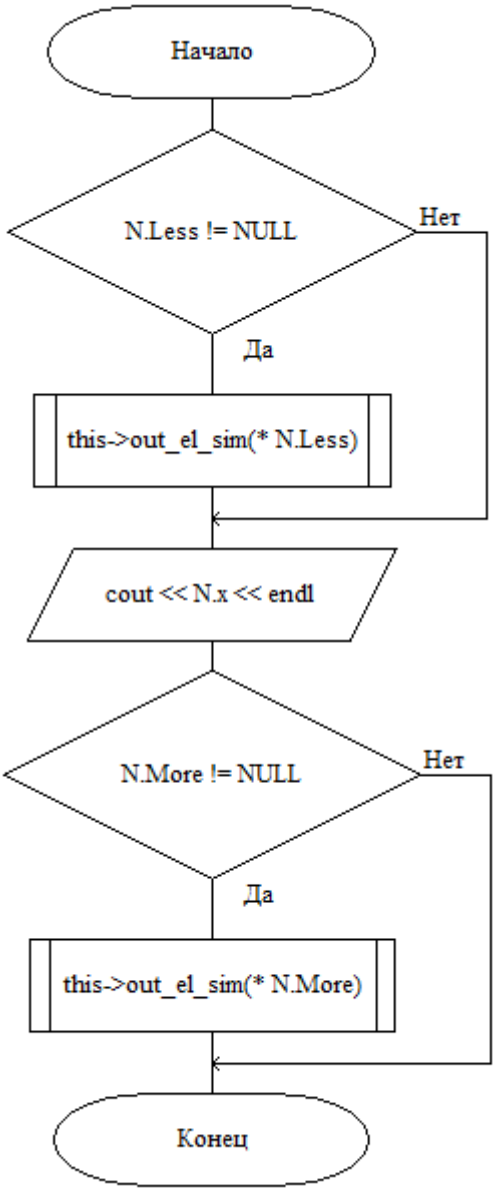


Рис.6 Схема алгоритма функции out_el_sim

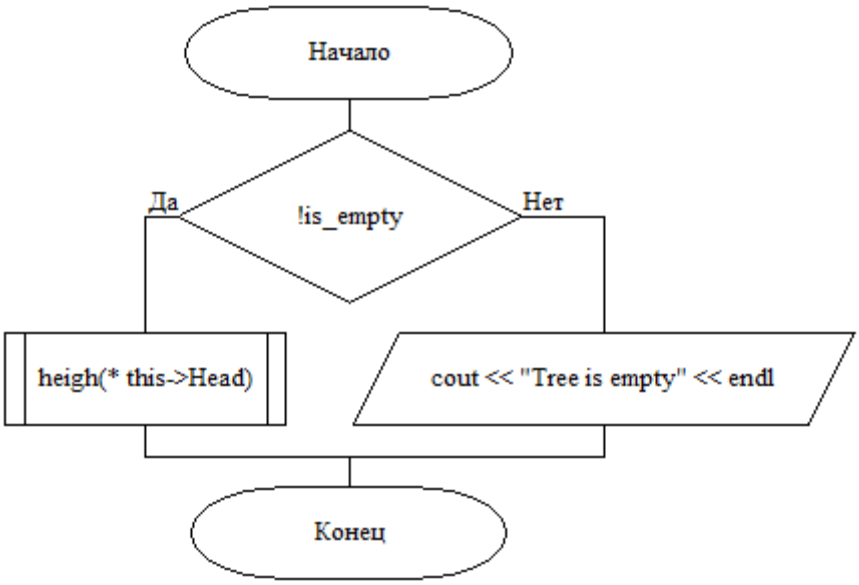


Рис.7 Схема алгоритма функции out_heigh

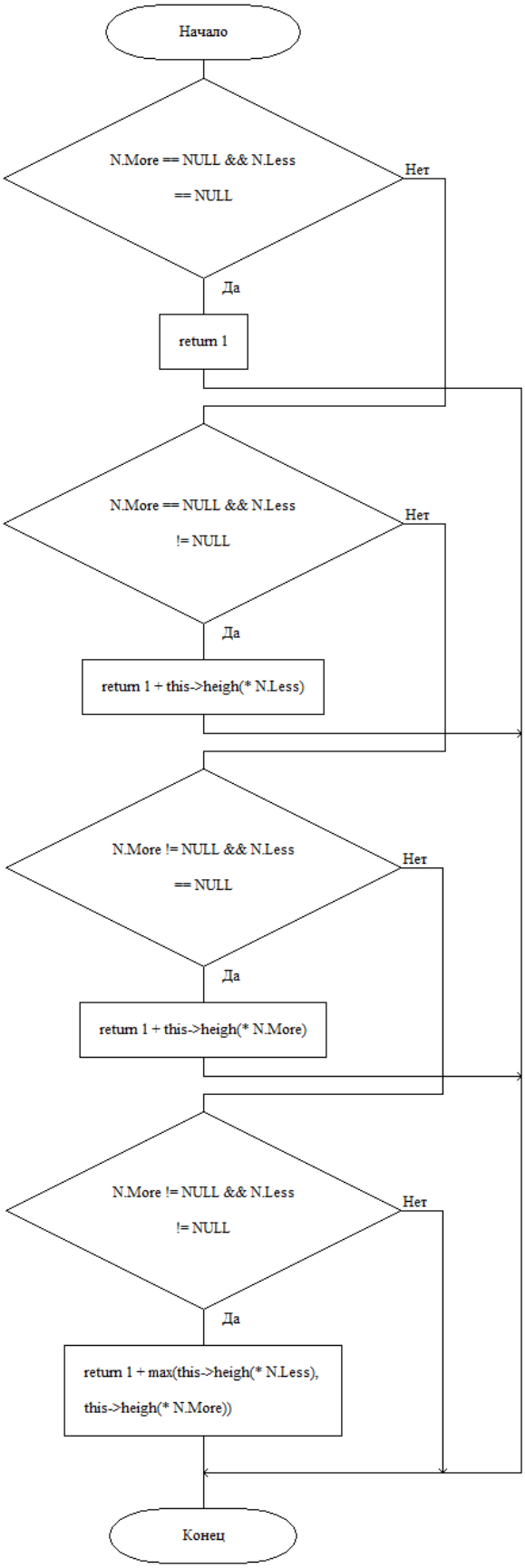

Начало
N.More == NULL && N.Less == NULL
Нет
Да
return 1
N.More == NULL && N.Less != NULL
Нет
Да
return 1 + this->heigh(* N.Less)
N.More != NULL && N.Less == NULL
Нет
Да
return 1 + this->heigh(* N.More)
N.More != NULL && N.Less != NULL
Нет
Да
return 1 + max(this->heigh(* N.Less), this->heigh(* N.More))
Конец

Рис.8 Схема алгоритма функции heigh

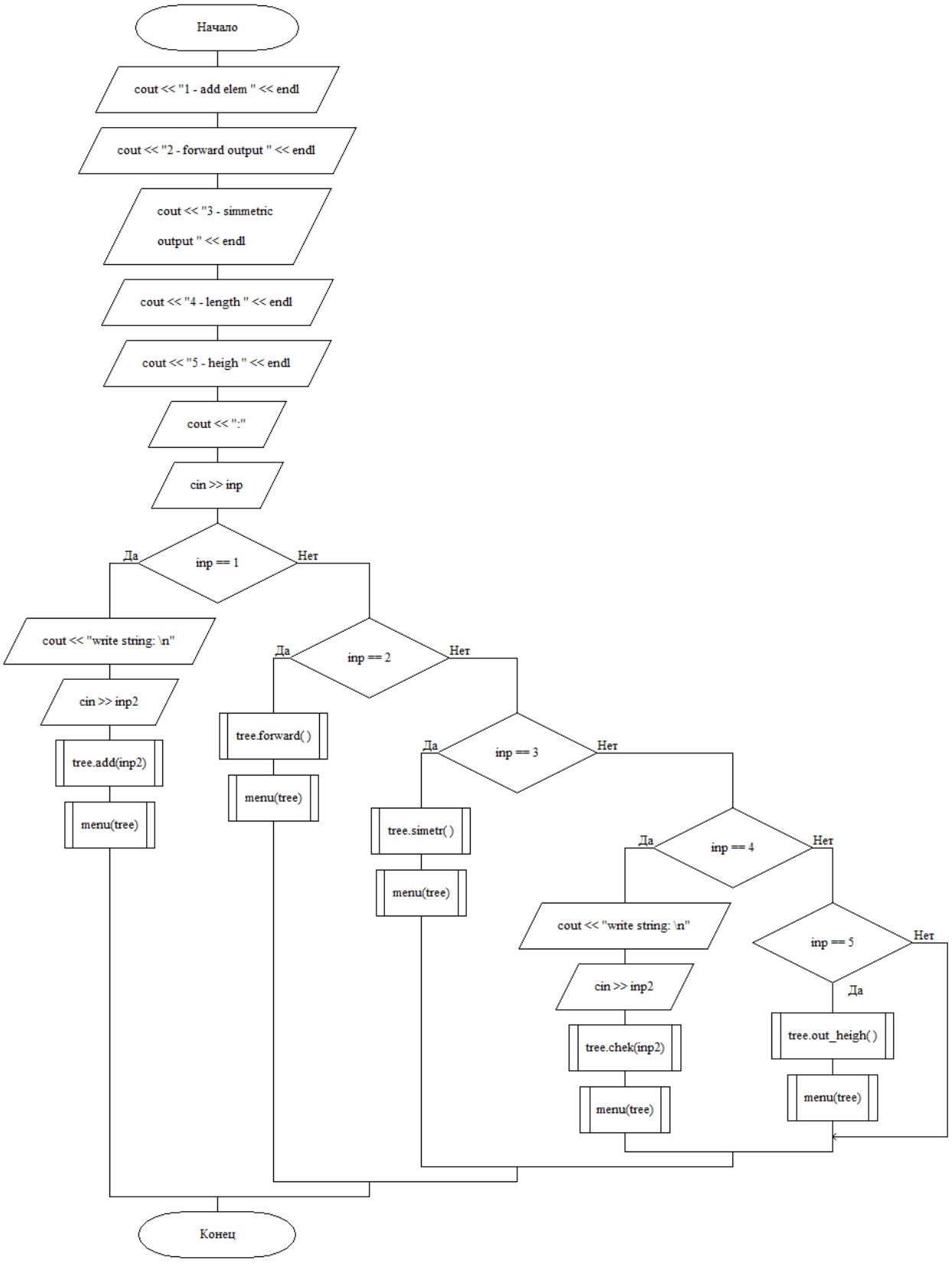


Рис.9 Схема алгоритма функции menu

13. Реализация алгоритма

Текст исходного кода программы

```
#include <iostream>

using namespace std;

struct Node {
    string x;
    Node *More = NULL;
    Node *Less = NULL;
};

class Tree {
    Node *More, *Less;

public:
    bool is_empty;
    Node *Head = NULL;

    Tree() : More(NULL), Less(NULL) { is_empty = true; };

    void add(string x);

    void forward();

    void simetr();

    void out_el_for(Node N);

    void out_el_sim(Node N);

    void chek(string x);

    int heigh(Node N);

    void out_heigh();

};

void Tree::add(string x) {
    Node *temp = new Node;
    if (is_empty) {
        temp->x = x;
```

```
        temp->Less                              =                               NULL;
        temp->More                              =                               NULL;
        Head                                =                                   temp;
        is_empty                              =                                 false;

    }                                       else                                    {
        temp                                =                                   Head;
        Node              *next              =              new                  Node;
        while                               (true)                                  {
            if              (x              >              temp->x)                 {
                if            (temp->More           !=           NULL)              {
                    temp                        =                       temp->More;

                }                               else                                {
                    temp->More              =              new                  Node;
                    temp->More->x                       =                       x;
                    break;
                }
            }         else         if         (x         <         temp->x)         {
                if            (temp->Less           !=           NULL)              {
                    temp                        =                       temp->Less;

                }                               else                                {
                    temp->Less              =              new                  Node;
                    temp->Less->x                       =                       x;
                    break;
                }
            }                                   else                                {
                cout    <<    "This    item    already    exists"    <<    endl;
                break;
            }
        }
    }
}

void                    Tree::chek(string                   x)                      {
    int                         c                       =                       0;
    Node                *temp               =               new                 Node;
    if                                  (is_empty)                              {
        cout       <<       "item       is       absent       "       <<       endl;

    }                                   else                                    {
        temp                                =                                   Head;

        while                               (true)                                  {
            if              (x              >              temp->x)                 {
                if            (temp->More           !=           NULL)              {
                    temp                        =                       temp->More;
                    c++;

                }                               else                                {


                    cout     <<     "item     is     absent     "     <<     endl;
                    break;
                }
            }         else         if         (x         <         temp->x)         {
                if            (temp->Less           !=           NULL)              {
                    temp                        =                       temp->Less;
                    c++;
```

```
                }                                   else                                    {

                    cout     <<     "item     is     absent     "     <<     endl;
                    break;
                }
            }                                       else                                    {
                cout     <<     "range     =     "     <<     c     <<     endl;
                break;
            }
        }

    }
}


void                                Tree::forward()                                         {
    if                                                  (!is_empty)out_el_for(*Head);
    else       cout       <<       "Tree       is       empty"       <<       endl;
}

void                  Tree::out_el_for(Node                  N)                  {
    cout                <<                N.x                <<                endl;
    if                (N.Less                !=                NULL)                {
        this->out_el_for(*N.Less);
    }
    if                (N.More                !=                NULL)                {
        this->out_el_for(*N.More);
    }

}

void                                Tree::simetr()                                          {

    if                                                  (!is_empty)out_el_sim(*Head);
    else       cout       <<       "Tree       is       empty"       <<       endl;
}

void                  Tree::out_el_sim(Node                  N)                  {

    if                (N.Less                !=                NULL)                {
        this->out_el_sim(*N.Less);
    }
    cout                <<                N.x                <<                endl;
    if                (N.More                !=                NULL)                {
        this->out_el_sim(*N.More);
    }

}

int             max(int             a,             int             b)             {
    if             (a             >=             b)             return             a;
    else                                    return                                  b;
}

void                                Tree::out_heigh()                                       {
    if  (!is empty)  cout  <<  "heigh  =  "  <<  this->heigh(*this->Head)  <<  endl;
    else       cout       <<       "Tree       is       empty"       <<       endl;

}
```

```
int Tree::heigh(Node N) {

    if (N.More == NULL && N.Less == NULL) return 1;
    if (N.More == NULL && N.Less != NULL) return 1 + this->heigh(*N.Less);
    if (N.More != NULL && N.Less == NULL) return 1 + this->heigh(*N.More);
    if (N.More != NULL && N.Less != NULL) return 1 + max(this->heigh(*N.Less),
this->heigh(*N.More));


}

void menu(Tree tree) {
    cout << "1 - add elem " << endl;
    cout << "2 - forward output " << endl;
    cout << "3 - simmetric output " << endl;
    cout << "4 - length " << endl;
    cout << "5 - heigh " << endl;
    cout << ":";

    int inp;
    cin >> inp;

    if (inp == 1) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        tree.add(inp2);
        menu(tree);
    } else if (inp == 2) {
        tree.forward();
        menu(tree);

    } else if (inp == 3) {
        tree.simetr();
        menu(tree);
    } else if (inp == 4) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        tree.chek(inp2);
        menu(tree);
    } else if (inp == 5) {
        tree.out_heigh();
        menu(tree);
    }
}

int main() {

    Tree a;
    menu(a);

}
```

14.Тестирование программы

Рис.10 Скриншот интерфейса программы

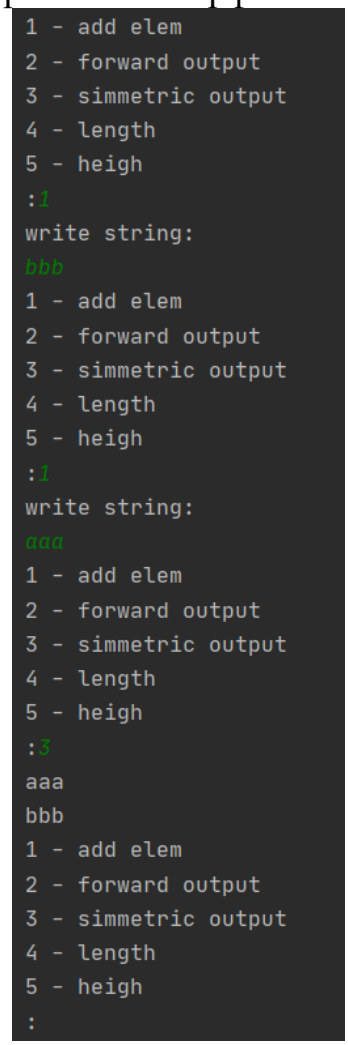


Рис.11 Скриншот добавления 2-х элементов и симметричного вывода

Выводы

1. В ходе работы было создано бинарное дерево поиска, ссылки между элементами которого осуществляются при помощи указателей на объекты класса Node.
2. Также были реализованы функции работы с деревьями: добавление элементов, прямой и симметричный выводы, вывод длинны и высоты дерева.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 03.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 03.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 03.11.2020).
5. Теоретические сведения о бинарных деревьях поиска, интернет-ресурс: https://habr.com/ru/post/267855 (Дата обращения – 03.11.2020).

Практическая работа № 7

АВЛ - деревья

Вариант-22Ф

Постановка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.

Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

15. Описание алгоритма

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:

Малое левое вращение

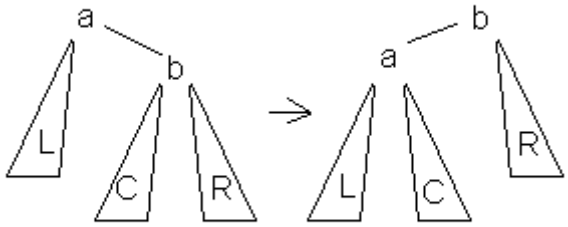


Рис.1 Малое левое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева <= высота R.

Большое левое вращение

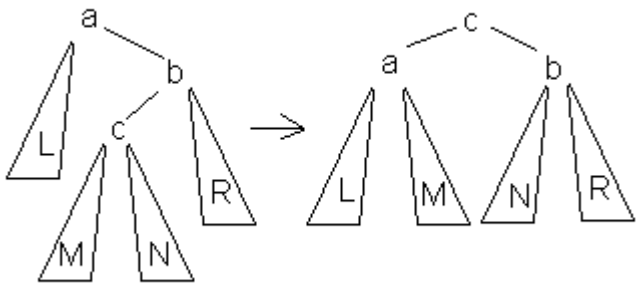


Рис.2 Большое левое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева > высота R.

Малое правое вращение

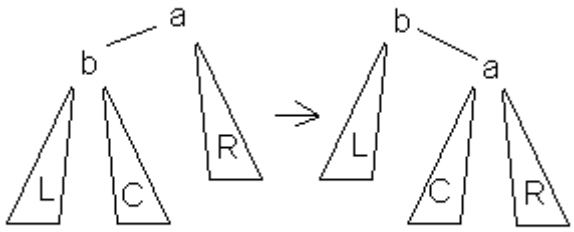


Рис.3 Малое правое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота C <= высота L.

Большое правое вращение

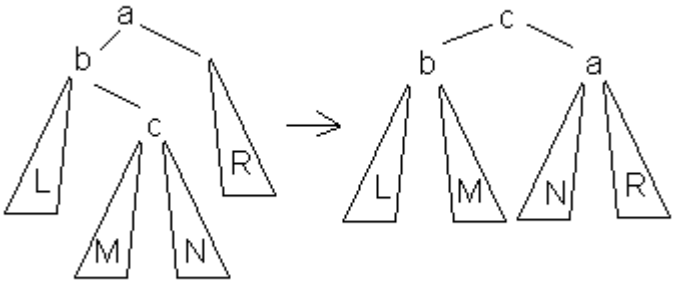


Рис.4 Большое правое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота c-поддерева > высота L.

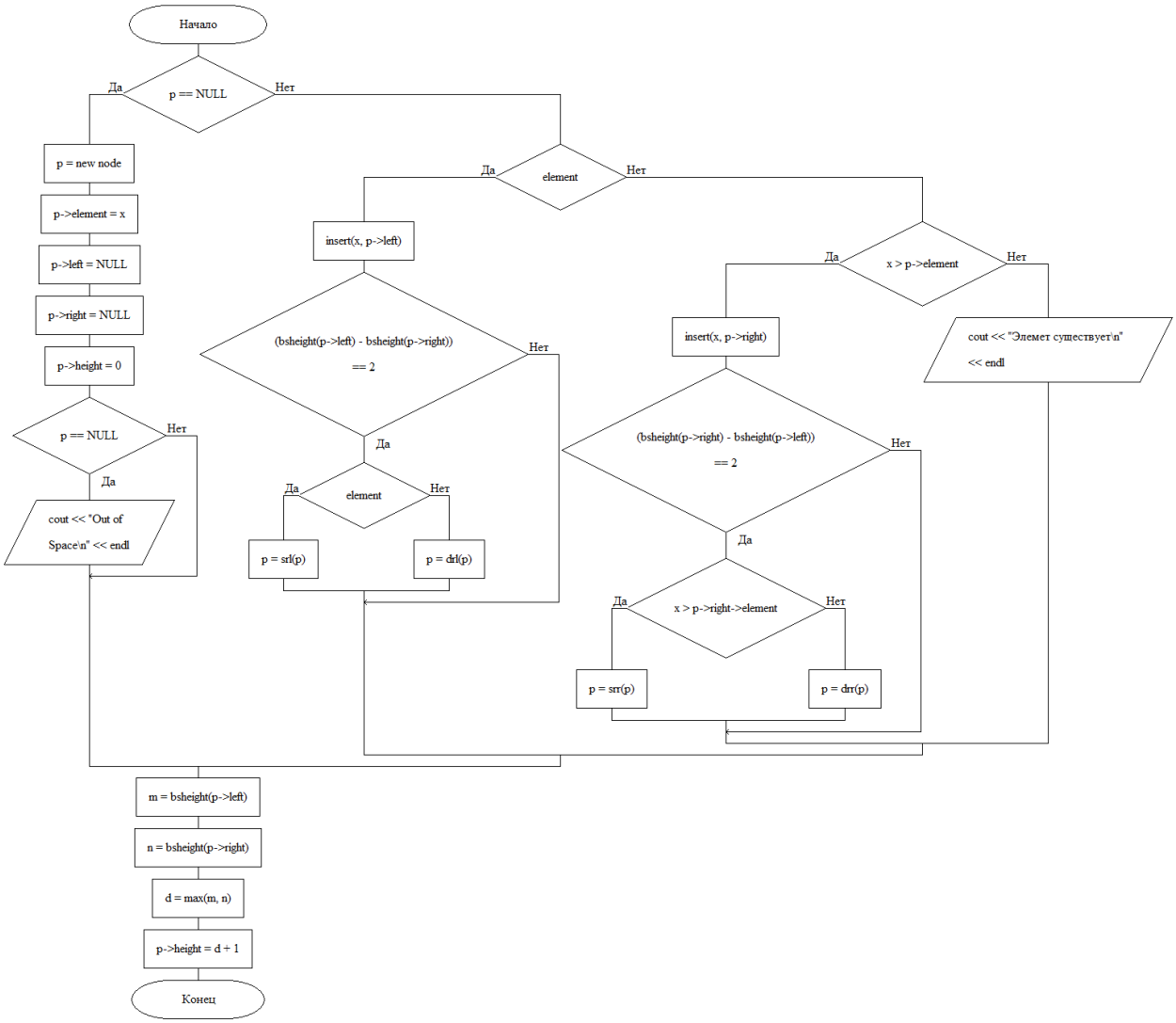


Рис.5 Схема алгоритма функции insert

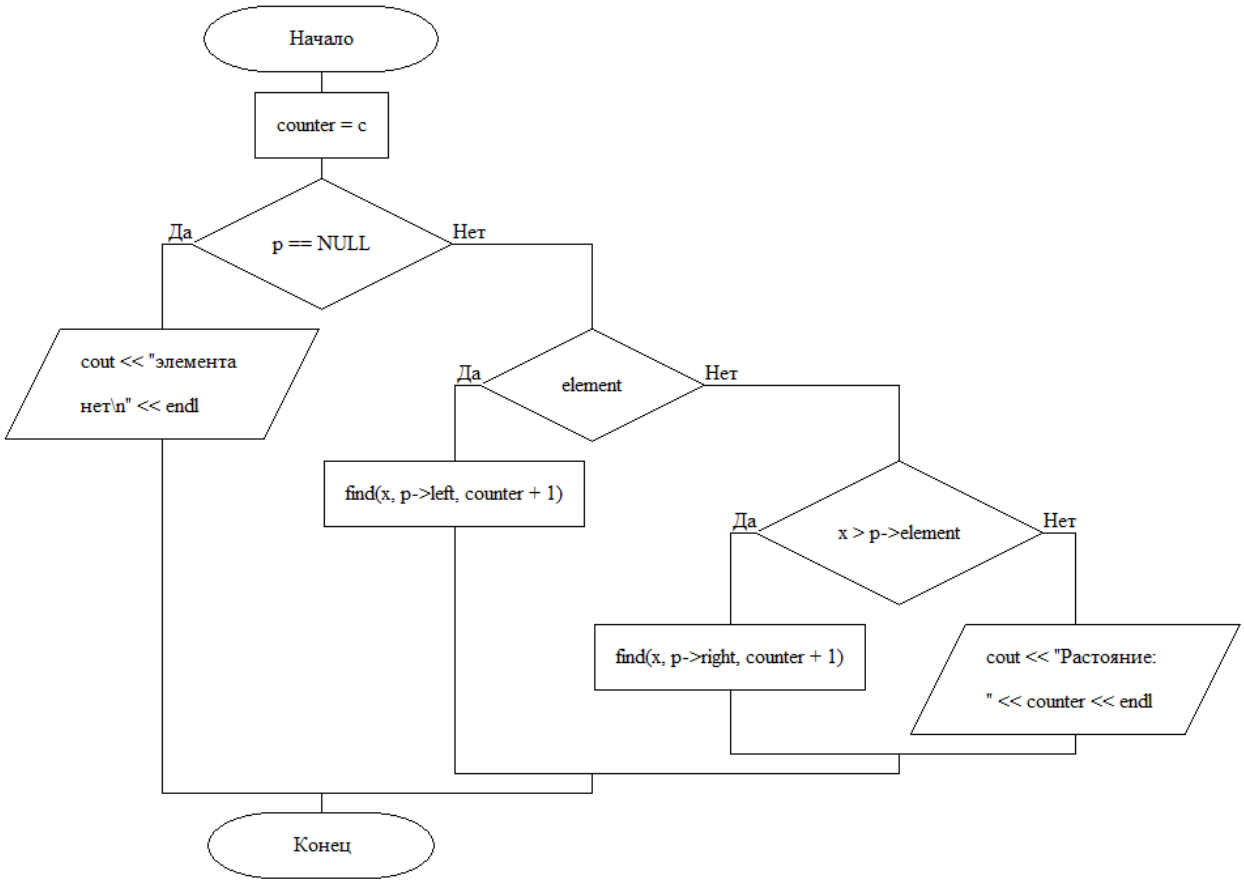


Рис.6 Схема алгоритма функции find

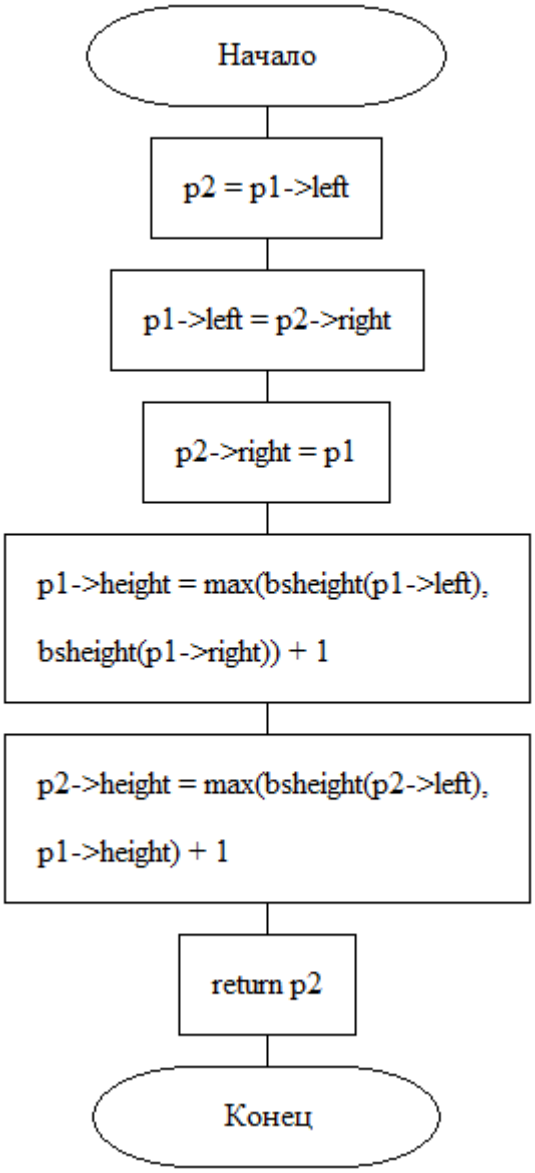


Рис.7 Схема алгоритма функции srl

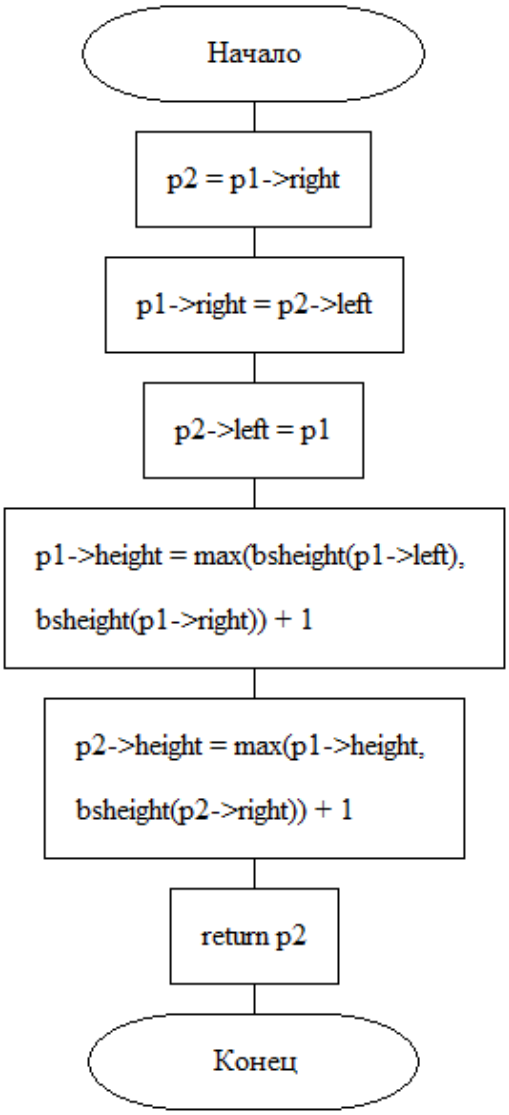


Рис.8 Схема алгоритма функции srr

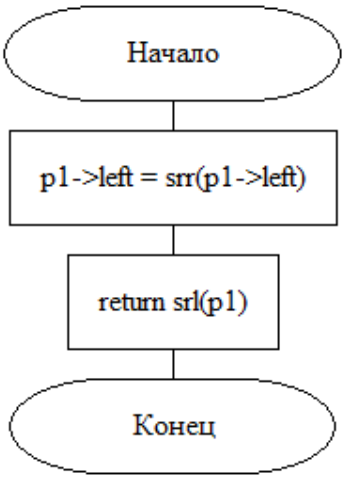


Рис.9 Схема алгоритма функции drl

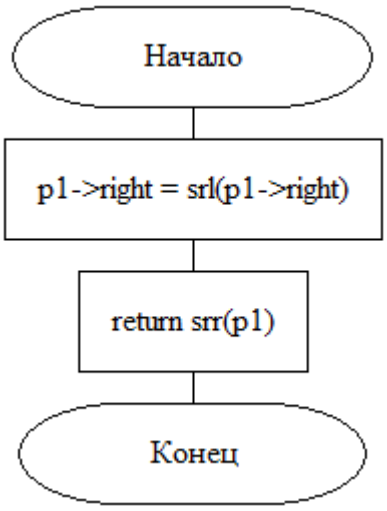


Рис.10 Схема алгоритма функции drr

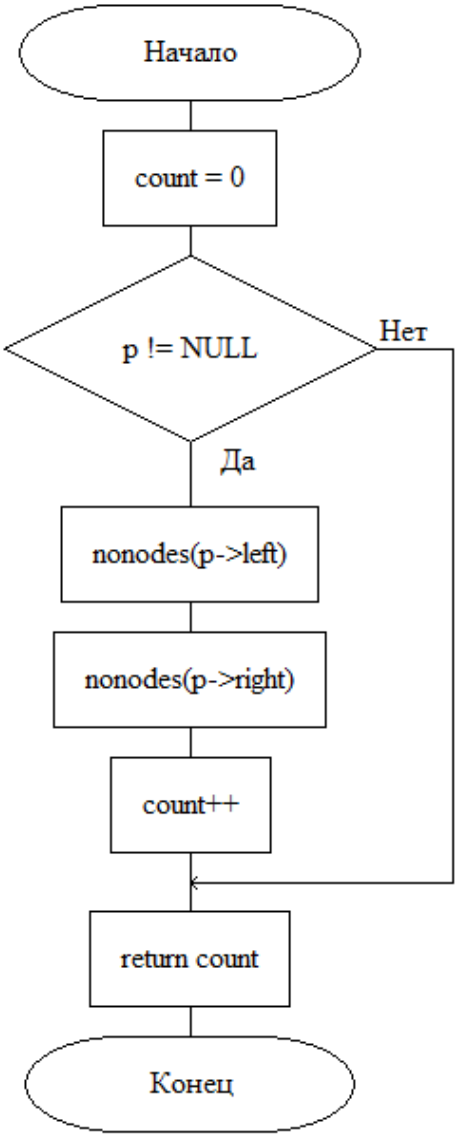


Рис.11 Схема алгоритма функции nonodes

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include <iostream>
#include<ctype.h>
#include <stdlib.h>
#include <conio.h>

using namespace std;

struct node
{
	int element;
	node* left;
	node* right;
	int height;
};

typedef struct node* nodeptr;

class bstree
{
public:
	void insert(int, nodeptr&);
	void find(int, nodeptr&m, int);
	void makeempty(nodeptr&);
	void copy(nodeptr&, nodeptr&);
	nodeptr nodecopy(nodeptr&);
	void preorder(nodeptr);
	int bsheight(nodeptr);
	nodeptr srl(nodeptr&);
	nodeptr drl(nodeptr&);
	nodeptr srr(nodeptr&);
	nodeptr drr(nodeptr&);
	int max(int, int);
	int nonodes(nodeptr);
```

```
};

void bstree::insert(int x, nodeptr& p)
{
	if (p == NULL)
	{
		p = new node;
		p->element = x;
		p->left = NULL;
		p->right = NULL;
		p->height = 0;
		if (p == NULL)
		{
			cout << "Out of Space\n" << endl;
		}
	}
	else
	{
		if (x < p->element)
		{
			insert(x, p->left);
			if ((bsheight(p->left) - bsheight(p->right)) == 2)
			{
				if (x < p->left->element)
				{
					p = srl(p);
				}
				else
				{
					p = drl(p);
				}
			}
		}
		else if (x > p->element)
		{
			insert(x, p->right);
			if ((bsheight(p->right) - bsheight(p->left)) == 2)
			{
				if (x > p->right->element)
				{
					p = srr(p);
				}
				else
				{
					p = drr(p);
				}
			}
		}
		else
		{
			cout << "Элемет существует\n" << endl;
		}
	}
	int m, n, d;
	m = bsheight(p->left);
	n = bsheight(p->right);
	d = max(m, n);
	p->height = d + 1;
}

void bstree::find(int x, nodeptr& p, int c)
{
	int counter = c;
```

```
	if (p == NULL)
	{
		cout << "элемента нет\n" << endl;
	}
	else
	{
		if (x < p->element)
		{
			find(x, p->left, counter+1);
		}
		else
		{
			if (x > p->element)
			{
				find(x, p->right, counter + 1);
			}
			else
			{
				cout << "Растояние: "<<counter<< endl;
			}
		}
	}
}

void bstree::copy(nodeptr& p, nodeptr& p1)
{
	makeempty(p1);
	p1 = nodecopy(p);
}

void bstree::makeempty(nodeptr& p)
{
	nodeptr d;
	if (p != NULL)
	{
		makeempty(p->left);
		makeempty(p->right);
		d = p;
		free(d);
		p = NULL;
	}
}
nodeptr bstree::nodecopy(nodeptr& p)
{
	nodeptr temp;
	if (p == NULL)
	{
		return p;
	}
	else
	{
		temp = new node;
		temp->element = p->element;
		temp->left = nodecopy(p->left);
		temp->right = nodecopy(p->right);
		return temp;
	}
}


void bstree::preorder(nodeptr p)
{
	if (p != NULL)
```

```
	{
		cout << p->element << "\t";
		preorder(p->left);
		preorder(p->right);
	}
}


int bstree::max(int value1, int value2)
{
	return ((value1 > value2) ? value1 : value2);
}
int bstree::bsheight(nodeptr p)
{
	int t;
	if (p == NULL)
	{
		return -1;
	}
	else
	{
		t = p->height;
		return t;
	}
}

nodeptr bstree::srl(nodeptr& p1)
{
	nodeptr p2;
	p2 = p1->left;
	p1->left = p2->right;
	p2->right = p1;
	p1->height = max(bsheight(p1->left), bsheight(p1->right)) + 1;
	p2->height = max(bsheight(p2->left), p1->height) + 1;
	return p2;
}
nodeptr bstree::srr(nodeptr& p1)
{
	nodeptr p2;
	p2 = p1->right;
	p1->right = p2->left;
	p2->left = p1;
	p1->height = max(bsheight(p1->left), bsheight(p1->right)) + 1;
	p2->height = max(p1->height, bsheight(p2->right)) + 1;
	return p2;
}
nodeptr bstree::drl(nodeptr& p1)
{
	p1->left = srr(p1->left);
	return srl(p1);
}
nodeptr bstree::drr(nodeptr& p1)
{
	p1->right = srl(p1->right);
	return srr(p1);
}

int bstree::nonodes(nodeptr p)
{
	int count = 0;
	if (p != NULL)
	{
		nonodes(p->left);
```

```
		nonodes(p->right);
		count++;
	}
	return count;
}

int main()
{
	setlocale(LC_ALL, "Russian");
	nodeptr root, root1, min, max;
	int a, choice, findele, delele;
	char ch = 'y';
	bstree bst;

	root = NULL;
	root1 = NULL;

	do
	{
		cout << "[1] - Вставить новый узел" << endl;
		cout << "[2] - Растояние от головного объекта до искомого" << endl;
		cout << "[3] - Показать высоту дерева" << endl;
		cout << "[4] - Прямой обход дерева" << endl;
		cout << "[5] - Выход" << endl;
		cin >> choice;

		switch (choice)
		{
		case 1:
			cout << "Добавление нового узла" << endl;
			cout << "Введите элемент: ";
			cin >> a;
			bst.insert(a, root);
			cout << "Новый элемент добавлен успешно" << endl;
			break;
		case 2:
			cout << "Введите искомый элемент: ";
			cin >> findele;
			if (root != NULL)
			{
				bst.find(findele, root, 0);
			}
			break;
		case 4:
			cout << "Вывод дерева:" << endl;
			bst.preorder(root);
			cout << endl;
			break;
		case 3:
			cout << "Дерево имеет высоту: " << bst.bsheight(root)+1 << endl;

			break;
		case 5:
			cout << "Программа завершена" << endl;
			break;
		default:

			break;
		}
	} while (choice != 0);
	return 0;
}
```

16.Тестирование программы

Ниже представлен результат работы программы

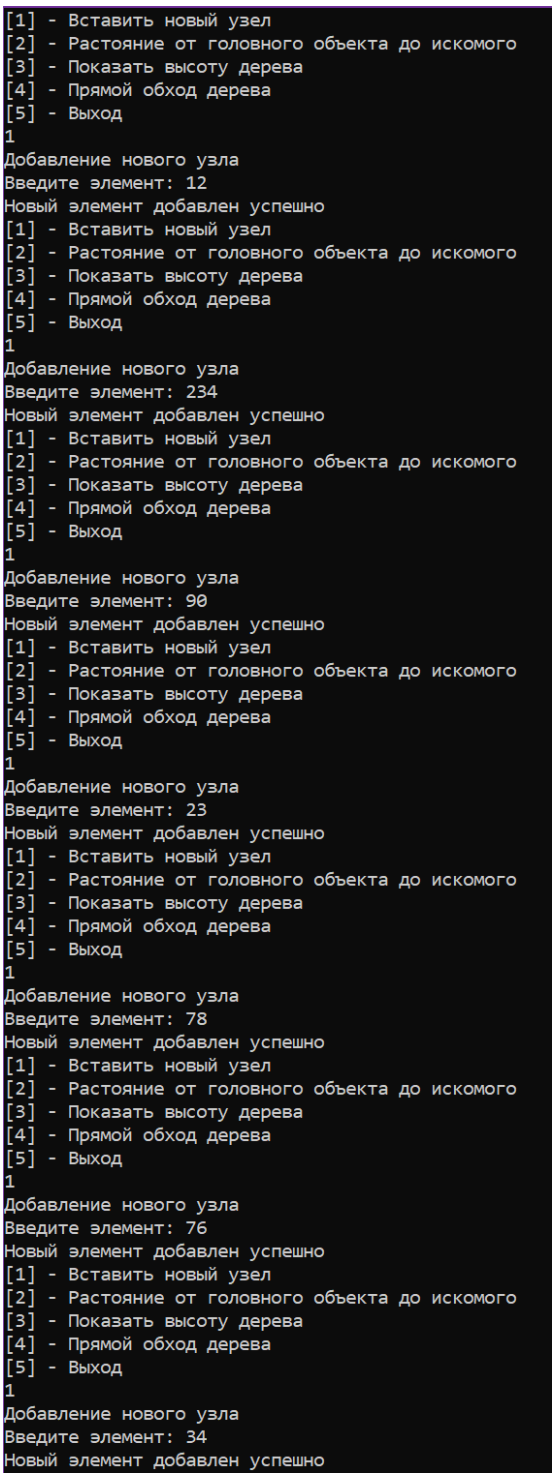


Рис.12 Скриншот добавления элементов в дерево

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
3
Дерево имеет высоту: 4
```

Рис.13 Скриншот вывода высоты дерева

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
2
Введите искомый элемент: 90
Растояние: 1
```

Рис.14 Скриншот вывода расстояния от головного объекта

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
4
Вывод дерева:
78      23      12      76      34      90      234
```

Рис.15 Скриншот вывода дерева

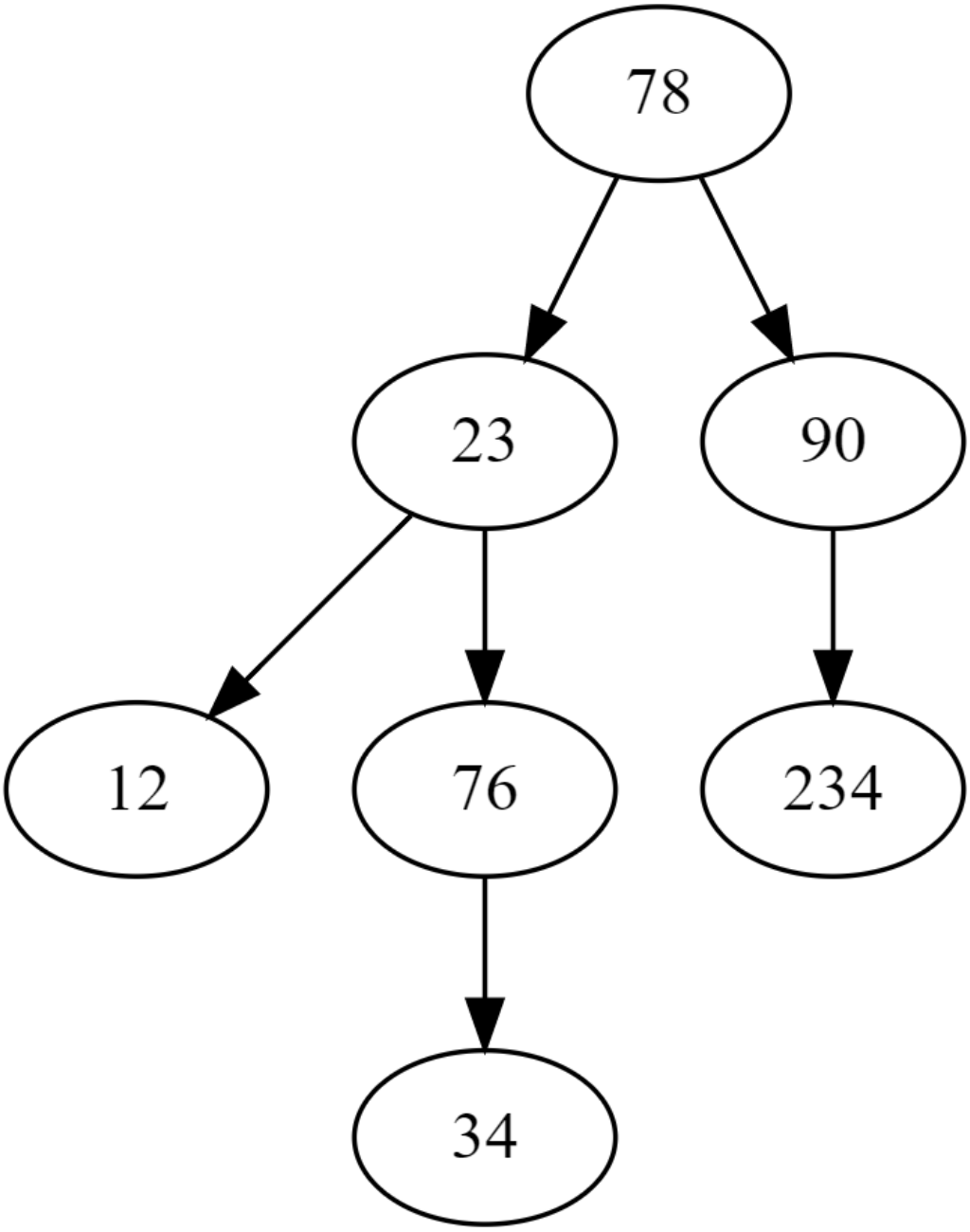


Рис.15 Скриншот визуального вывода дерева после балансировки

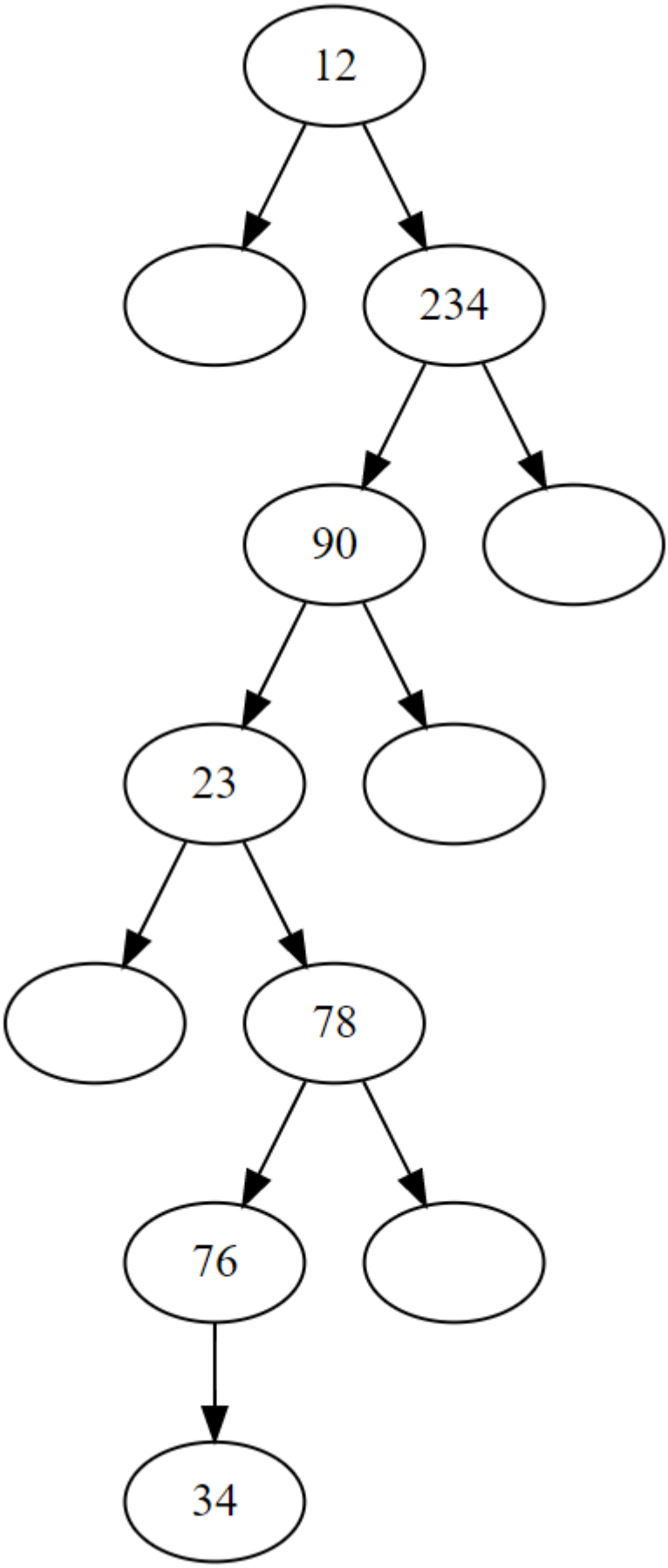

12
234
90
23
78
76
34

Рис.15 Скриншот визуального вывода дерева до балансировки

Выводы

1. В ходе работы была создана программа для работы с AVL-деревьями.
2. Также были реализованы функции добавления, поиска элементов, а также вычисления высоты.
3. Были изучены преимущества и недостатки хранения данных в AVL - деревьях:
4. Преимущества: поиск в сбалансированном AVL – дереве имеет временную сложность log(n), что является довольно эфективным по сравнению с аналогами.
5. Недостатки: не смотря на довольно низкую временную сложность поиска главным недостатком AVL-дерева является необходимость постоянной автобалансировки, что требует некоторых затрат по времени. Также поиск в AVL – дереве сильно уступает ХЭШ – таблице по скорости.
6. Таким образом, была изучена работа AVL - деревьев

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 30.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 30.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 30.11.2020).
5. Описание AVL - деревьев. интернет-ресурс: https://ru.wikipedia.org/wiki/АВЛ-дерево (Дата обращения – 30.11.2020).
6. Построение графов по DOT-нотации. интернет-ресурс: https://dreampuf.github.io/GraphvizOnline (Дата обращения – 30.11.2020).

Практическая работа № 8

ФАЙЛЫ

Вариант-22Ф

Постановка задачи

Создать программные модули с операциями над двоичными и текстовыми файлами для выполнения задания варианта. Двоичный файл состоит из записей определенной структуры. Записи имеют ключ, уникальный в пределах файла. Для выполнения варианта задания использовать потоковые файлы C++.

1 - Преобразовать данные в двоичный файл
2 - Отобразить все записи двоичного файла
3 - Удаление записей по начальному фрагменту ключа
4 - Заменить не ключевой параметр у нескольких записей по ключу

Перечисленные действия оформить в виде самостоятельных режимов работы созданного дека. Выбор режимов производить с помощью пользовательского меню.

Провести полное тестирование (всех режимов работы) программы на стеке, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Оформить отчет с подробным описанием созданного алгоритма, принципов программной реализации алгоритмов работы с файлами, описанием текста исходного кода и проведенного тестирования программы.

Сделать выводы о проделанной работе, основанные на полученных результатах.

17. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void read_file** – функция считывания товаров файла.
- **void add_in_file** – функция добавления товара в файл.
- **int search_in_list_id** – функция поиска товара в списке.
- **int search_in_list_part_id** – функция поиска по началу ключа.
- **void del_elems_part_id** – функция удаления товаров по началу ключа.
- **void change_elem_id** – функция изменения товара по ключу.
- **void change_elems –** функция именения нескольких товаров.
- **void file_update() –** функция обновления файла.
- **void out_list() –** функция вывода всех товаров из файла.

Для работы с файлами в языке с++ используется класс fstream.

ifstream для считывания файла, ofstream для его записи.

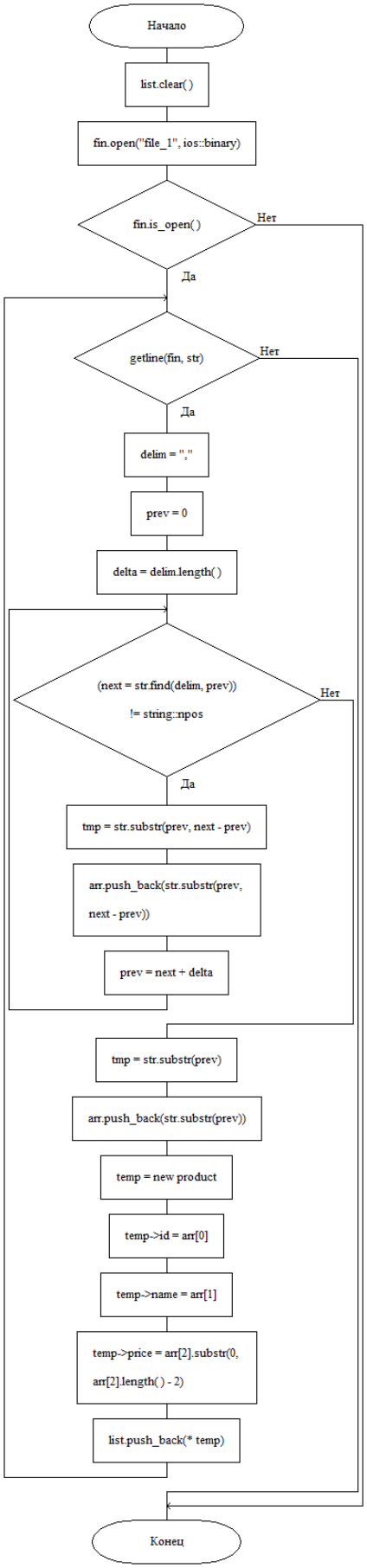

Начало
list.clear( )
fin.open("file_1", ios::binary)
fin.is_open( )
Нет
Да
getline(fin, str)
Нет
Да
delim = ";"
prev = 0
delta = delim.length( )
(next = str.find(delim, prev))
!= string::npos
Нет
Да
tmp = str.substr(prev, next - prev)
arr.push_back(str.substr(prev,
next - prev))
prev = next + delta
tmp = str.substr(prev)
arr.push_back(str.substr(prev))
temp = new product
temp->id = arr[0]
temp->name = arr[1]
temp->price = arr[2].substr(0,
arr[2].length( ) - 2)
list.push_back(* temp)
Конец

Рис.1 Схема алгоритма функции read_file

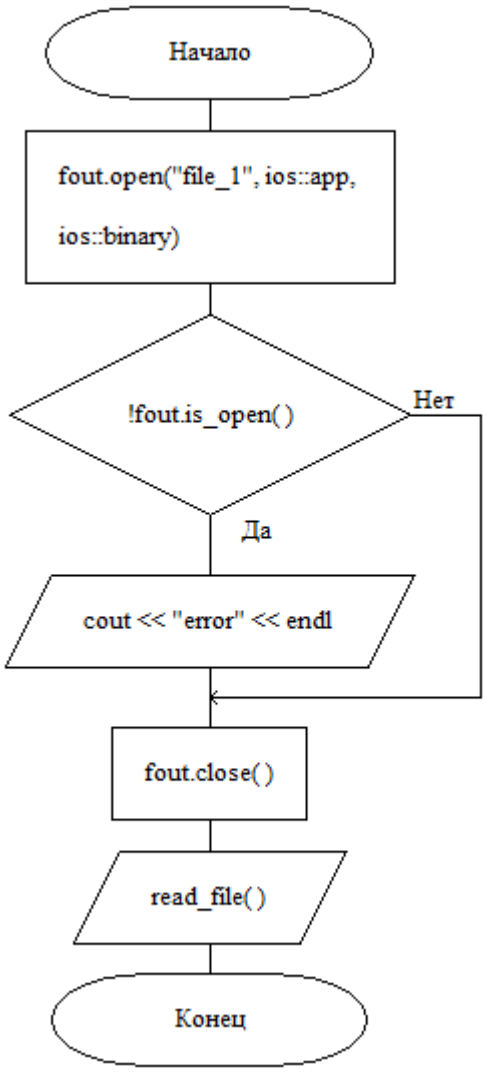


Рис.2 Схема алгоритма функции add_in_file

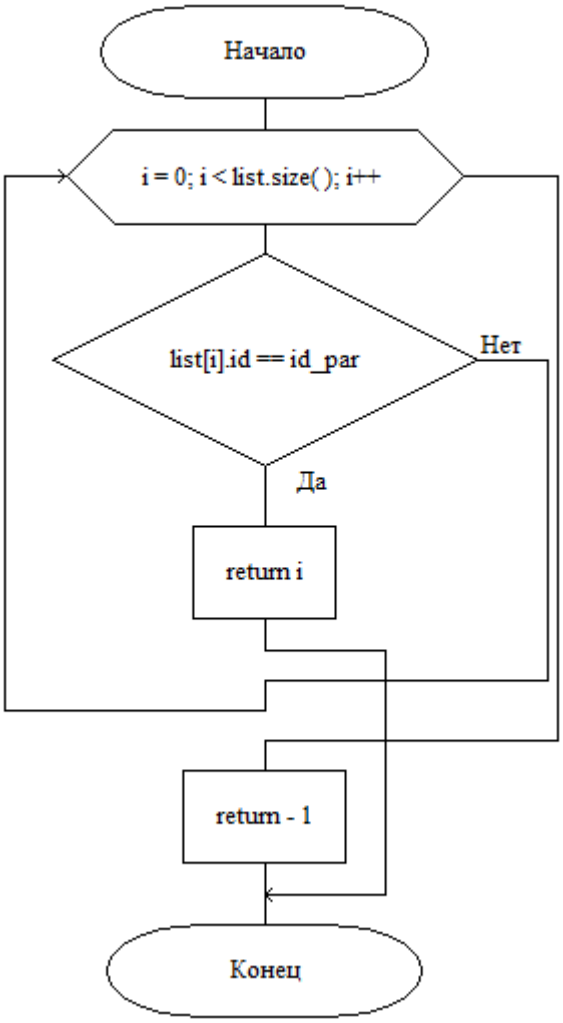


Рис.3 Схема алгоритма функции search_in_list_id

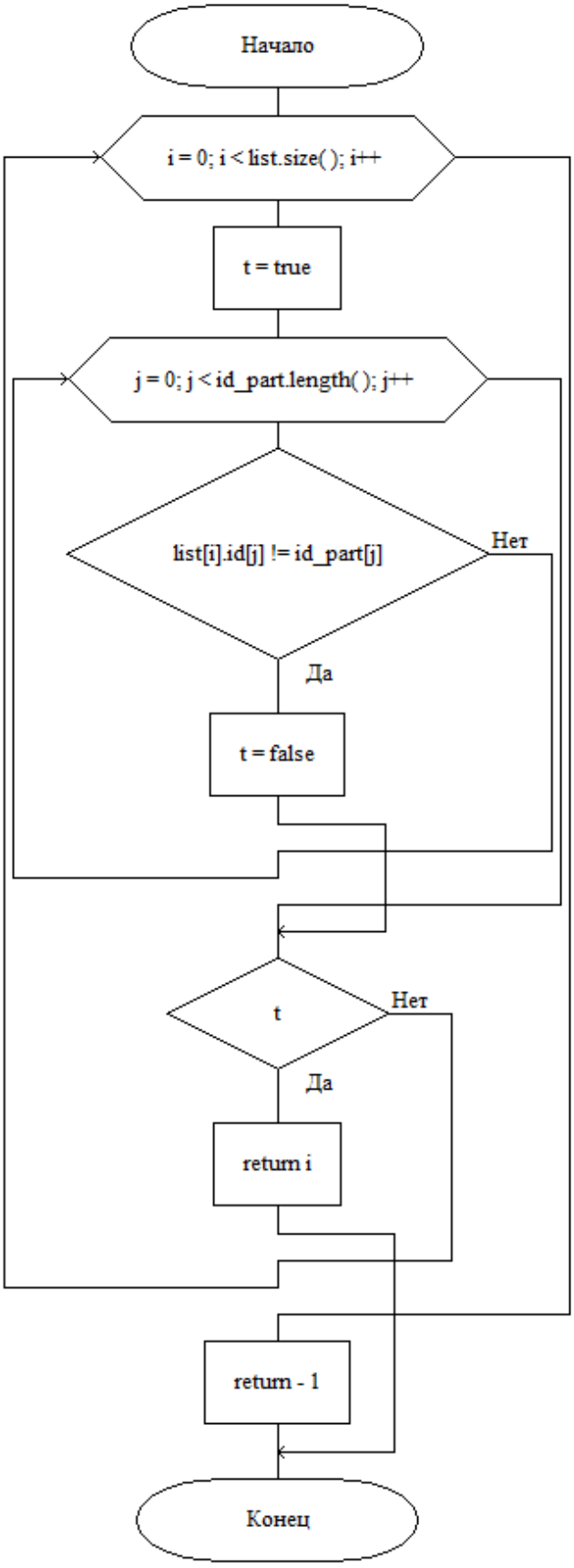

Начало
i = 0; i < list.size( ); i++
t = true
j = 0; j < id_part.length( ); j++
list[i].id[j] != id_part[j]
Нет
Да
t = false
t
Нет
Да
return i
return - 1
Конец

Рис.4 Схема алгоритма функции search_in_list_part_id

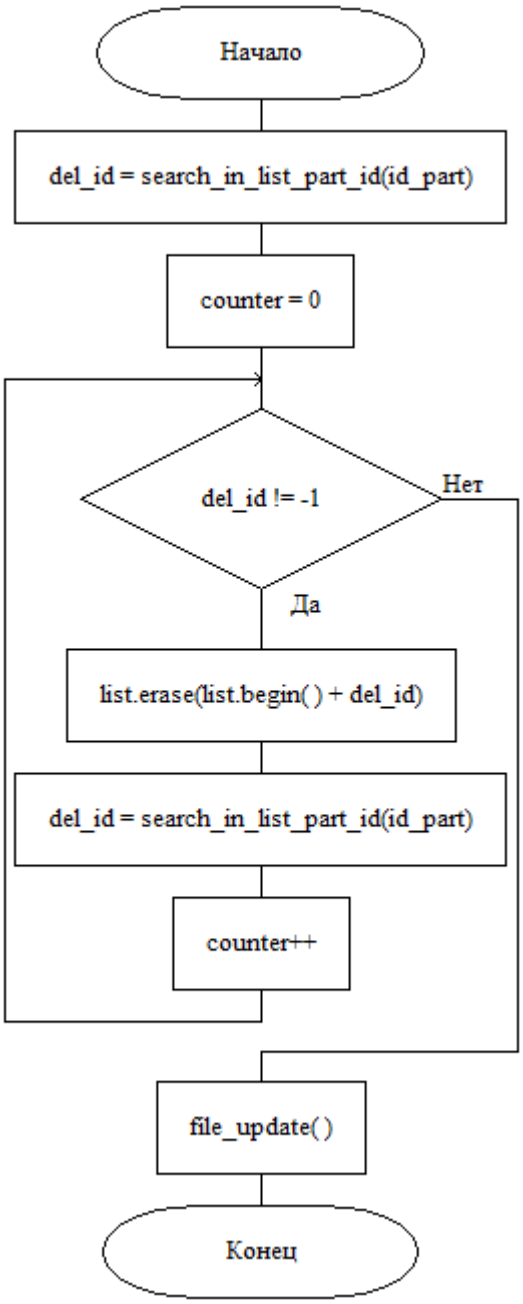


Рис.5 Схема алгоритма функции del_elems_part_id

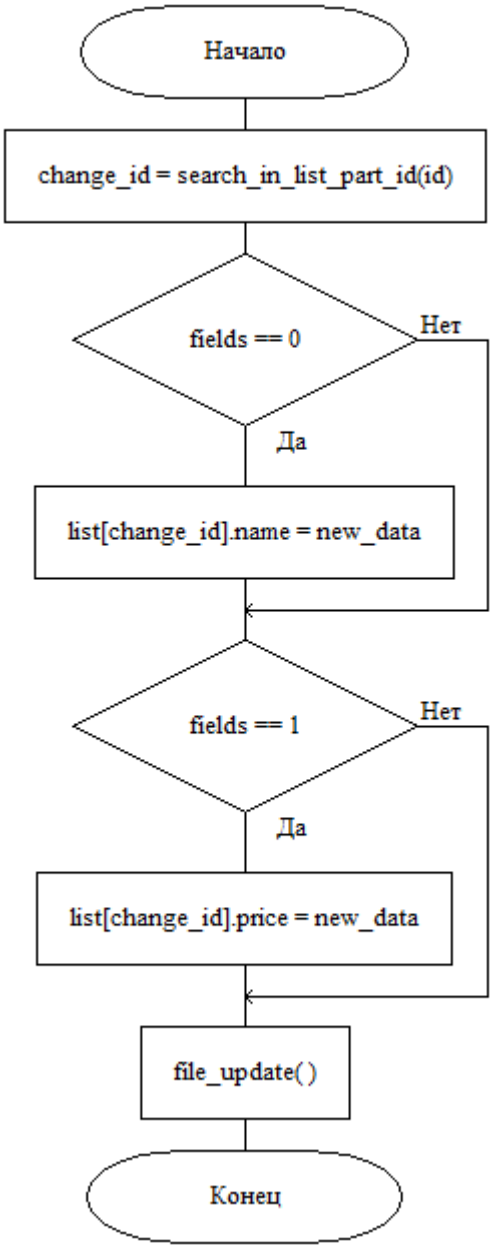


Рис.6 Схема алгоритма функции change_elem_id

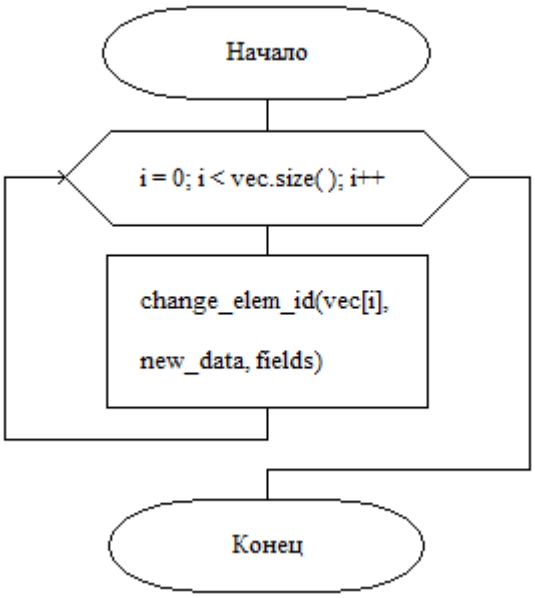


Рис.7 Схема алгоритма функции change_elems

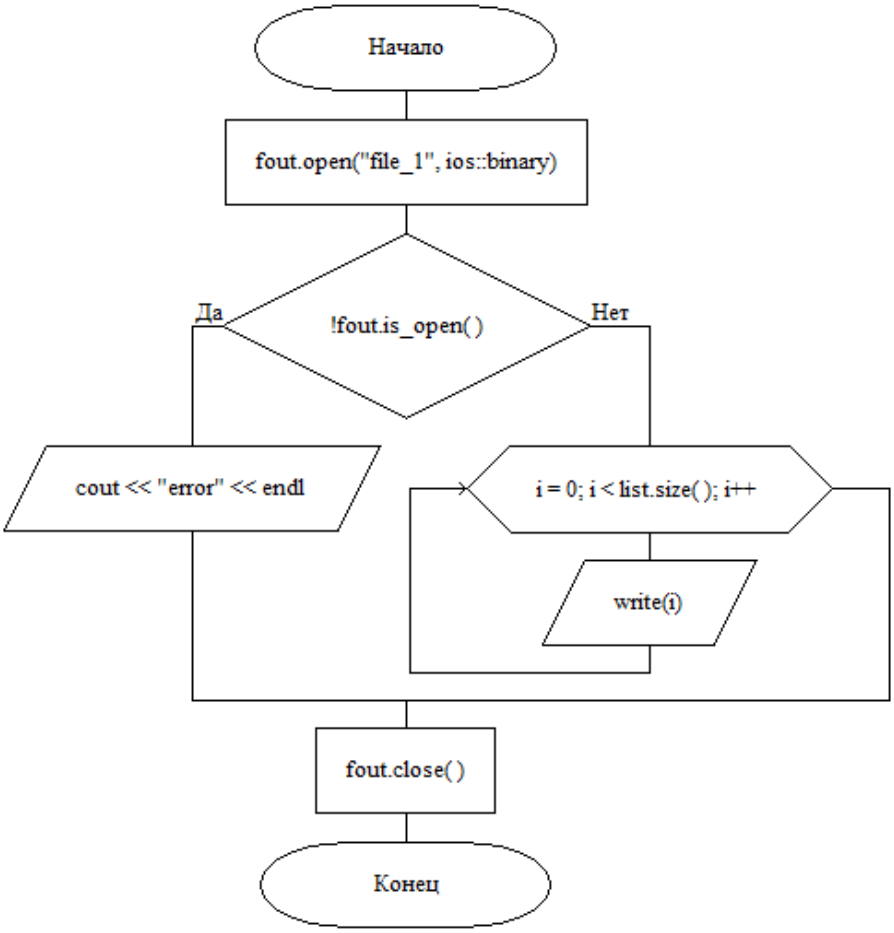


Рис.8 Схема алгоритма функции file_update

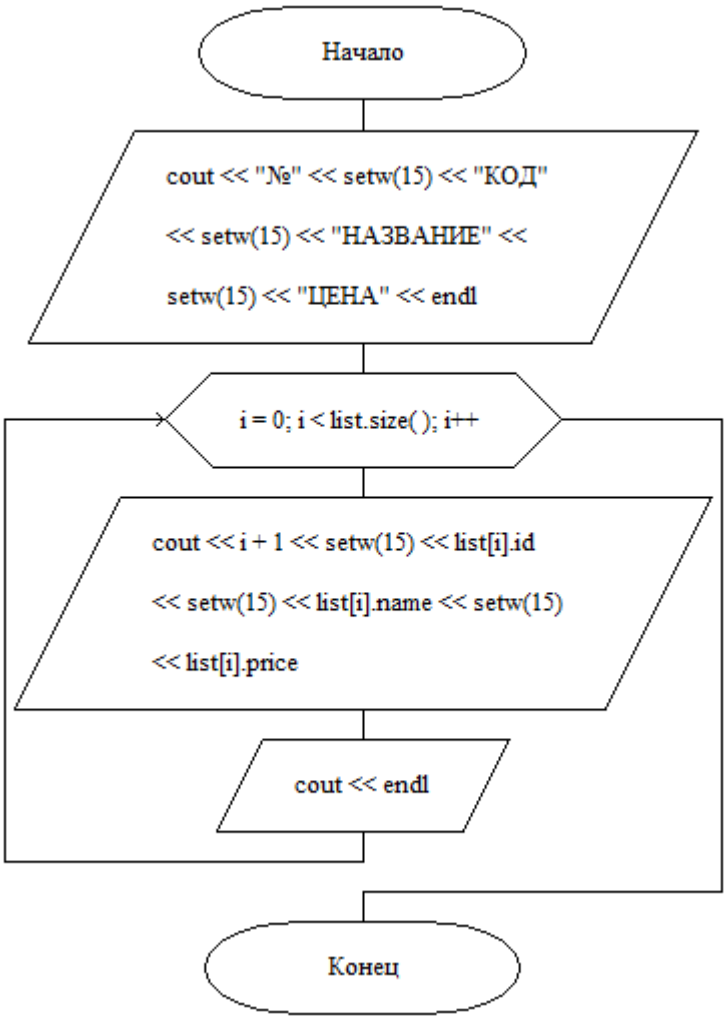


Рис.9 Схема алгоритма функции out_list()

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;
void file_update();
struct product{
    string id;
    string name;
    string price;
};
vector<product> list;

void read_file() {
    list.clear();
    string str;
    ifstream fin;
    fin.open("file_1", ios::binary);
    if (fin.is_open()) {
        while (getline(fin, str))
        {
            vector<string> arr;
            size_t next;
            string delim = ",";
            size_t prev = 0;
            size_t delta = delim.length();
            while ((next = str.find(delim, prev)) != string::npos) {
                string tmp = str.substr(prev, next - prev);
                arr.push_back(str.substr(prev, next - prev));
                prev = next + delta;
            }
            string tmp = str.substr(prev);
            arr.push_back(str.substr(prev));
            product* temp = new product;
            temp->id = arr[0];
            temp->name = arr[1];
            temp->price = arr[2].substr(0, arr[2].length()-1);
            list.push_back(*temp);
        }
    }
}

void add_in_file(string s) {
    ofstream fout;
    fout.open("file_1", ios::app, ios::binary);
    if (!fout.is_open()) cout << "error" << endl;
    else {
        fout << s << endl;
    }
```

```
    fout.close();
    read_file();
}

int search_in_list_id(string id_par) {
    for (int i = 0; i < list.size(); i++) {
        if (list[i].id == id_par) {
            return i;
        }
    }
    return -1;
}

int search_in_list_part_id(string id_part) {
    for (int i = 0; i < list.size(); i++) {
        bool t = true;
        for (int j = 0; j < id_part.length(); j++) {
            if (list[i].id[j] != id_part[j]) {
                t = false;
                break;
            }
        }
        if (t)   return i;
    }
    return -1;
}

void del_elems_part_id(string id_part) {
    int del_id = search_in_list_part_id(id_part);
    int counter = 0;
    while (del_id != -1) {
        list.erase(list.begin() + del_id);
        del_id = search_in_list_part_id(id_part);
        counter++;
    }
    file_update();
    cout << "Удалено товаров: " << counter <<  endl;
}

void change_elem_id(string id, string new_data, int fields) {
    int change_id = search_in_list_part_id(id);


    if(fields == 0){
        list[change_id].name = new_data;
    }
    if (fields == 1) {
        list[change_id].price = new_data;
    }
    file_update();
}

void change_elems(vector<string> vec, string new_data, int fields) {
    for (int i = 0; i < vec.size(); i++) {
        change_elem_id(vec[i], new_data, fields);
    }
    cout << "Элементов изменено: " << vec.size() << endl;
}

void file_update() {
    ofstream fout;
    fout.open("file_1", ios::binary);
    if (!fout.is_open()) cout << "error" << endl;
```

```
    else {
        for (int i = 0; i < list.size(); i++) {
            fout << list[i].id+","+ list[i].name + "," + list[i].price<<endl;
        }

    }
    fout.close();

}

void out_list() {
    cout << "№" << setw(15) << "КОД" << setw(15) << "НАЗВАНИЕ" << setw(15) << "ЦЕНА" << endl;
    for (int i = 0; i < list.size(); i++) {
        cout  <<i+1<<setw(15)<<  list[i].id  <<  setw(15)  <<  list[i].name  <<  setw(15)  <<
list[i].price;
        cout << endl;
    }
}

int main()
{
    setlocale(LC_ALL, "Russian");
    ofstream fout;
    fout.open("file_1", ios::binary);
    if (!fout.is_open()) cout << "error" << endl;
    else {
        fout << "";
    }
    fout.close();

    while (true) {
        cout << "Выберите команду:" << endl;
        cout << "[1] - Добавить товар." << endl;
        cout << "[2] - Удалить товары по началу ключа." << endl;
        cout << "[3] - Изменить несколько товаров." << endl;
        cout << "[4] - Вывести список товаров." << endl;
        cout << "[5] - Завершить программу." << endl;
        cout << "---->";
        int ch = 0;
        cin >> ch;
        if (ch == 1) {
            cout << "____Введите НОМЕР, НАЗВАНИЕ и ЦЕНУ товара через запятые без пробелов" <<
endl;
            cout << "-------->";
            string new_str;
            cin >> new_str;
            add_in_file(new_str);
            continue;
        }
        if (ch == 2) {
            cout << "____Введите начало ключа" << endl;
            cout << "-------->";
            string new_key;
            cin >> new_key;
            del_elems_part_id(new_key);
            continue;
        }
        if (ch == 3) {
            cout << "____Введите через пробел номера товаров (0 для завершения)" << endl;
            cout << "-------->";
            string new_id = "";
            vector<string> ids;
            cin >> new_id;
```

```
            while (new_id != "0") {
                ids.push_back(new_id);
                cin >> new_id;
            }
            cout << "____Введите новые данные и название поля для вставки" << endl;
            cout << "-------->";
            string data;
            int field;
            cin >> data >> field;
            change_elems(ids, data, field);
            continue;
        }
        if (ch == 4) {
            out_list();
            continue;

        }
        if (ch == 5) {
            break;
        }
    }
}
```

18.Тестирование программы

Ниже представлен результат работы программы с бинарным файлом

```
Выберите команду:
[1] - Добавить товар.
[2] - Удалить товары по началу ключа.
[3] - Изменить несколько товаров.
[4] - Вывести список товаров.
[5] - Завершить программу.
---->1
____Введите НОМЕР, НАЗВАНИЕ и ЦЕНУ товара через запятые без пробелов
-------->248973,AAAAA,100
Выберите команду:
[1] - Добавить товар.
[2] - Удалить товары по началу ключа.
[3] - Изменить несколько товаров.
[4] - Вывести список товаров.
[5] - Завершить программу.
---->1
____Введите НОМЕР, НАЗВАНИЕ и ЦЕНУ товара через запятые без пробелов
-------->388923,BBBB,250
Выберите команду:
[1] - Добавить товар.
[2] - Удалить товары по началу ключа.
[3] - Изменить несколько товаров.
[4] - Вывести список товаров.
[5] - Завершить программу.
---->1
____Введите НОМЕР, НАЗВАНИЕ и ЦЕНУ товара через запятые без пробелов
-------->893212,CCCCCC,1000
Выберите команду:
[1] - Добавить товар.
[2] - Удалить товары по началу ключа.
[3] - Изменить несколько товаров.
[4] - Вывести список товаров.
[5] - Завершить программу.
---->1
____Введите НОМЕР, НАЗВАНИЕ и ЦЕНУ товара через запятые без пробелов
-------->394324,DDD,15000
```

Рис.8 Скриншот добавления товаров в файл

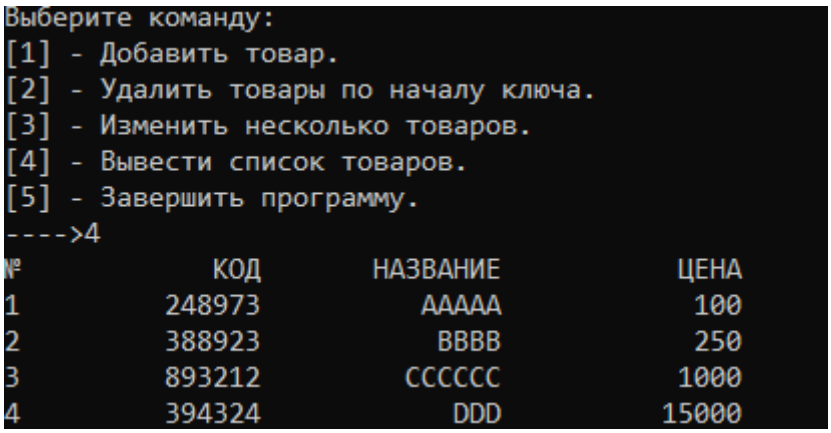


Рис.9 Скриншот вывода списка на экран

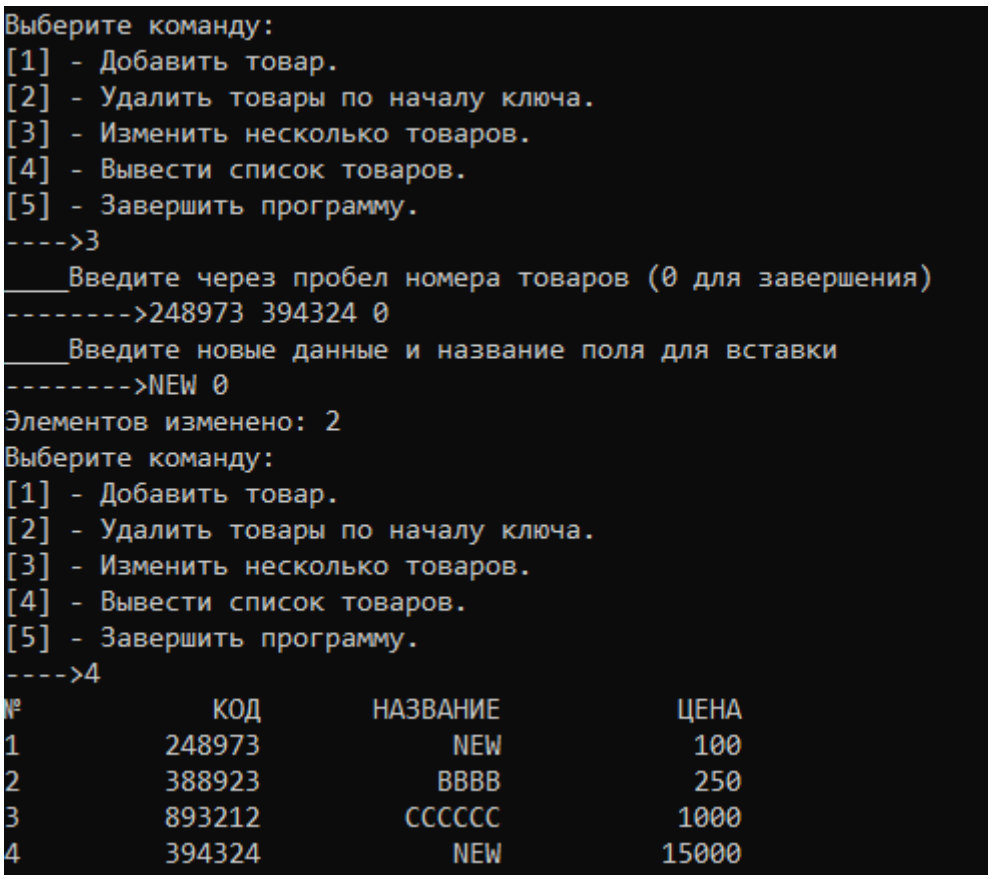


Рис.10 Скриншот изменения названия товаров по ключу и вывод

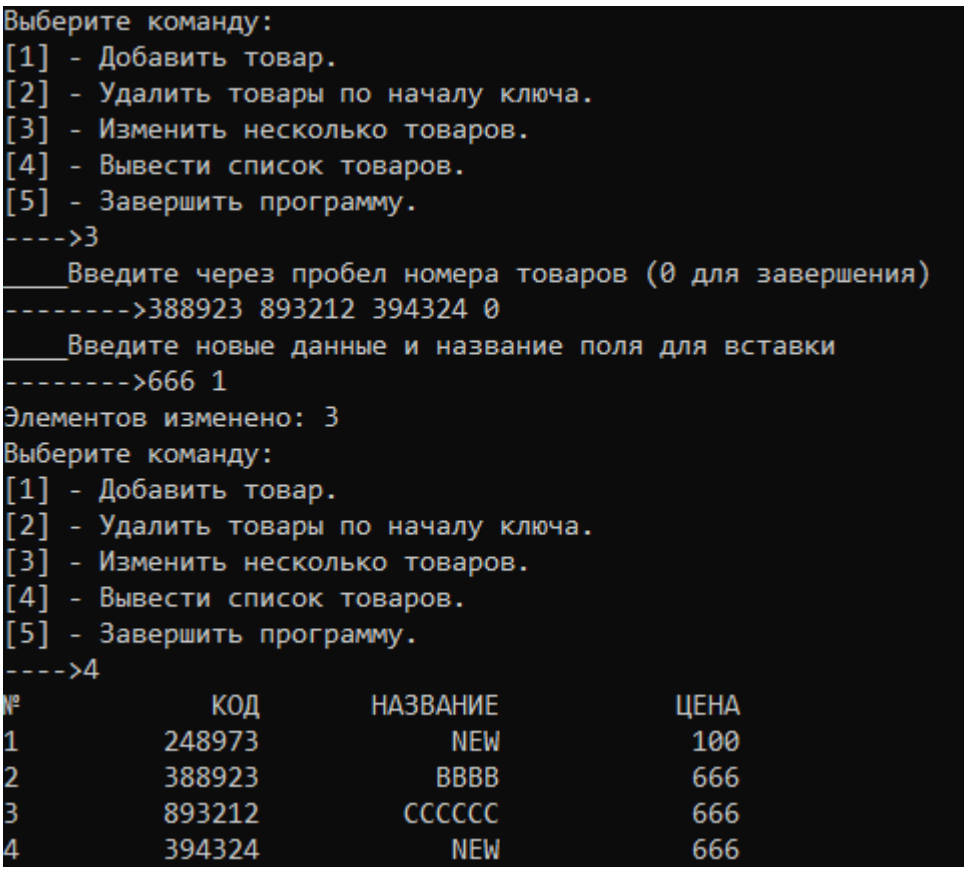


Рис.10 Скриншот изменения цены товаров по ключу и вывод

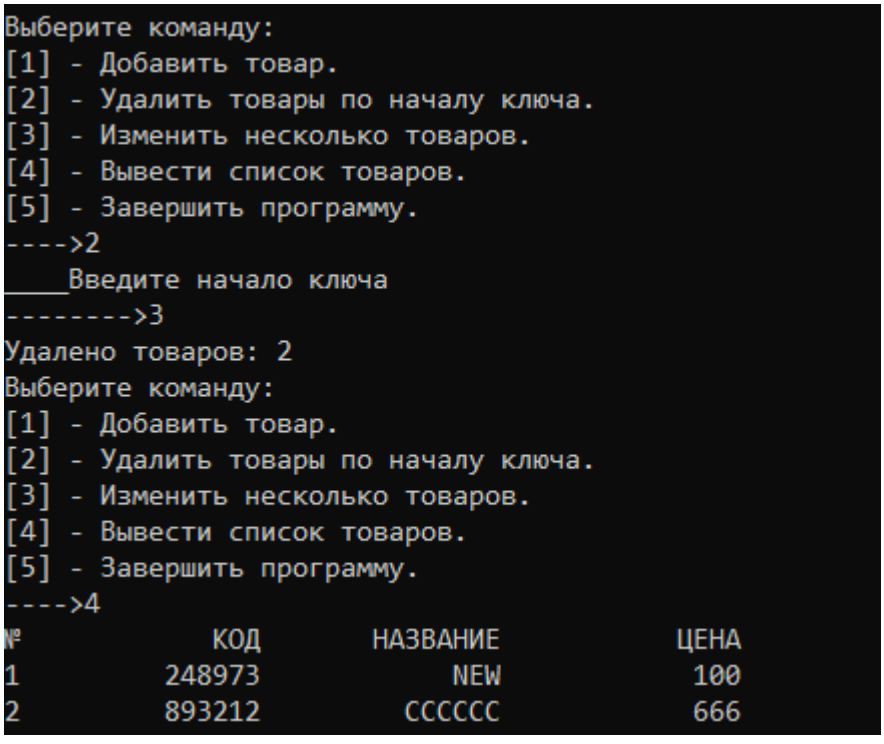


Рис.10 Скриншот удаления товаров по началу ключа

Выводы

1. В ходе работы была создана программа для работы с файлами.
2. Также были реализованы функции записи, чтения, удаления и изменения данных в файле.
3. Были изучены преимущества и недостатки хранения данных в файле:
4. Преимущества: данные в файлах могут храниться даже если устройство отключено от сети. В файлы можно записать гораздо больший объем данных чем в оперативную память
5. Недостатки: для работы необходимо постоянно обращаться к файловой системе.
6. Таким образом, была изучена работа алгоритмов обработки файлов

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание файлов. интернет-ресурс: https://ru.wikipedia.org/wiki/Файл (Дата обращения – 02.11.2020).

Практическая работа № 9

ХЕШИРОВАНИЕ

Вариант-22Ф

Постановка задачи

Разработайте приложение, которое использует хеш-таблицу для организации прямого доступа к элементам множества, реализованного на массиве, структура элементов которого приведена в варианте.

Открытая адресация, Счет в банке: номер счета целое семизначное число, ФИО, адрес

19. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **int get_hash_code** – функция генерации хеш-значения.
- **void add_cell** – функция добавления счета в таблицу.
- **void show_table** – функция вывода таблицы.
- **void find_cell** – функция поиска по владельцу.
- **void delete_cell** – функция удаления записи из таблицы.

В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками. Последовательность, в которой просматриваются ячейки хеш-таблицы, называется последовательностью проб. В общем случае, она зависит только от ключа элемента, то есть это последовательность h0(x), h1(x), …, hn — 1(x), где x — ключ элемента, а hi(x) — произвольные функции, сопоставляющие каждому ключу ячейку в хеш-таблице. Первый элемент в последовательности, как правило, равен значению некоторой хеш-функции от ключа, а остальные считаются от него одним из приведённых ниже способов. Для успешной работы алгоритмов поиска последовательность проб должна быть такой, чтобы все ячейки хеш-таблицы оказались просмотренными ровно по одному разу. Алгоритм поиска просматривает ячейки хеш-таблицы в том же самом порядке, что и при вставке, до тех пор, пока не найдется либо элемент с искомым ключом, либо пока не будет достигнут конец списка. Ошибочно представление, что следует искать до первой свободной ячейки, так как возможно, что элемент из этой ячейки был удален, а искомый элемент находится далее.

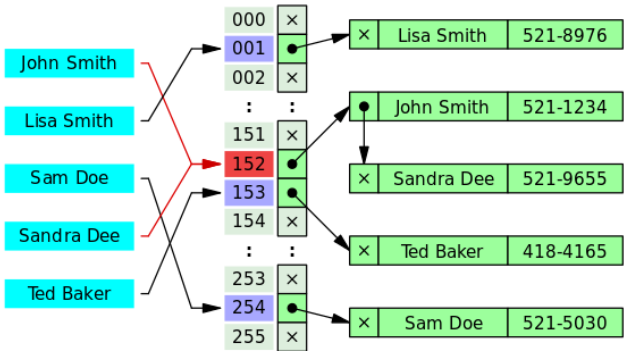


Рис.1 Схема алгоритма открытой адресации

Функция main создает объект класса Table и вызывает меню.

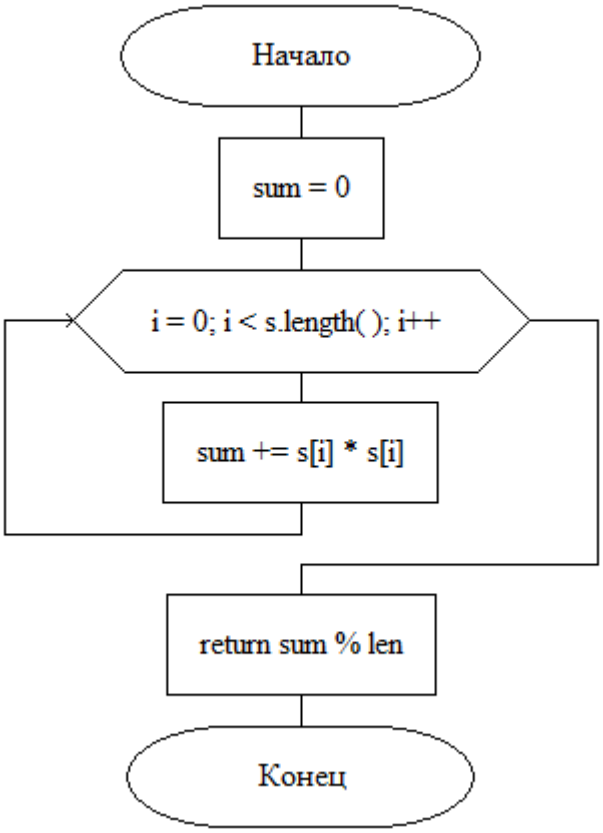


Рис.2 Схема алгоритма функции get_hash_code

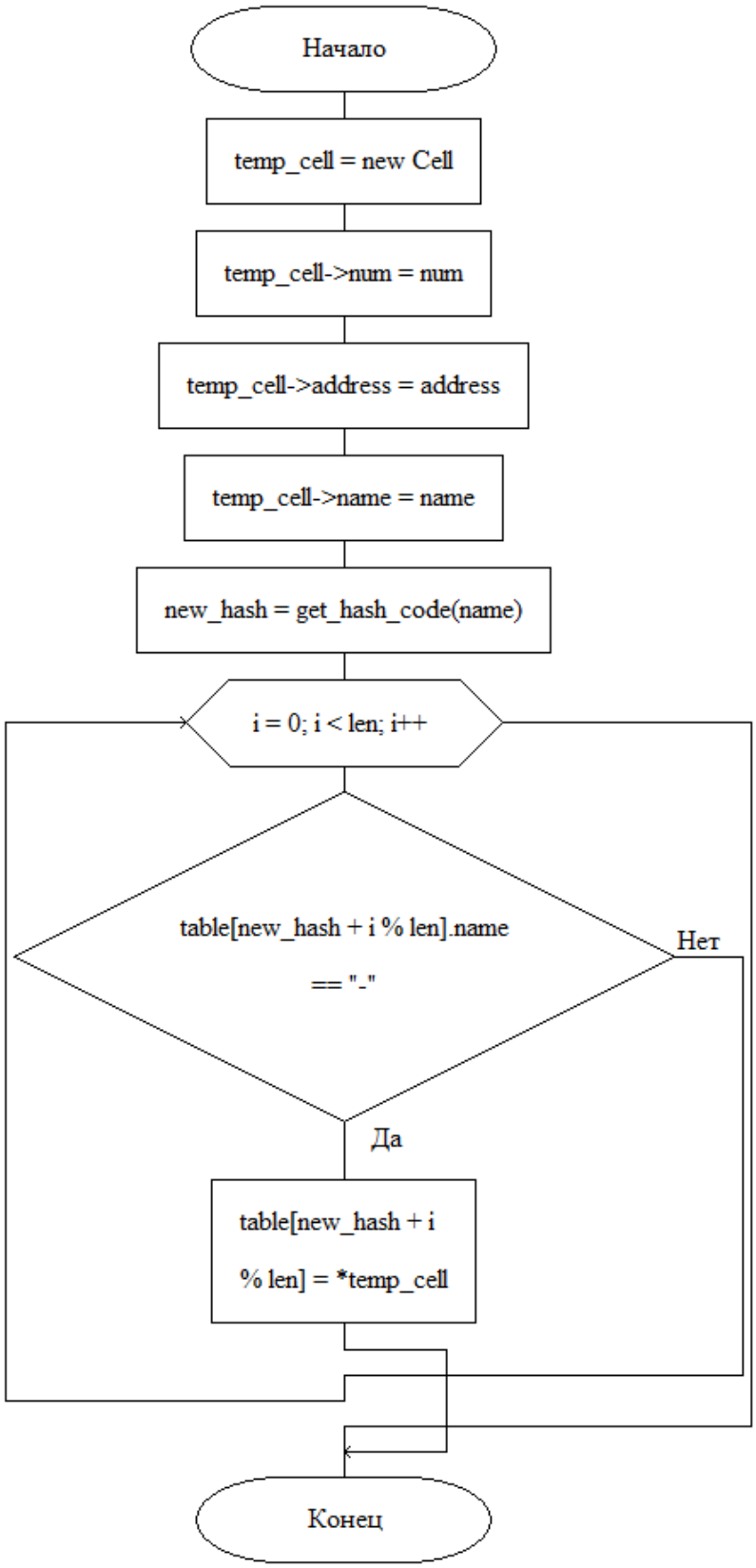


Рис.3 Схема алгоритма функции add_cell

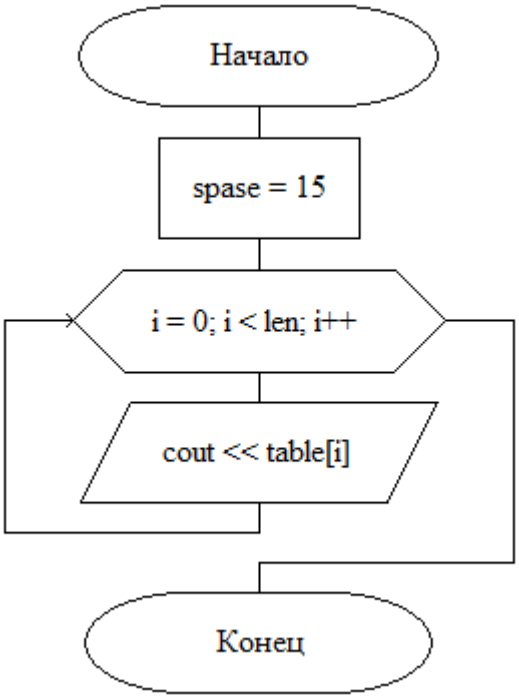


Рис.4 Схема алгоритма функции show_table

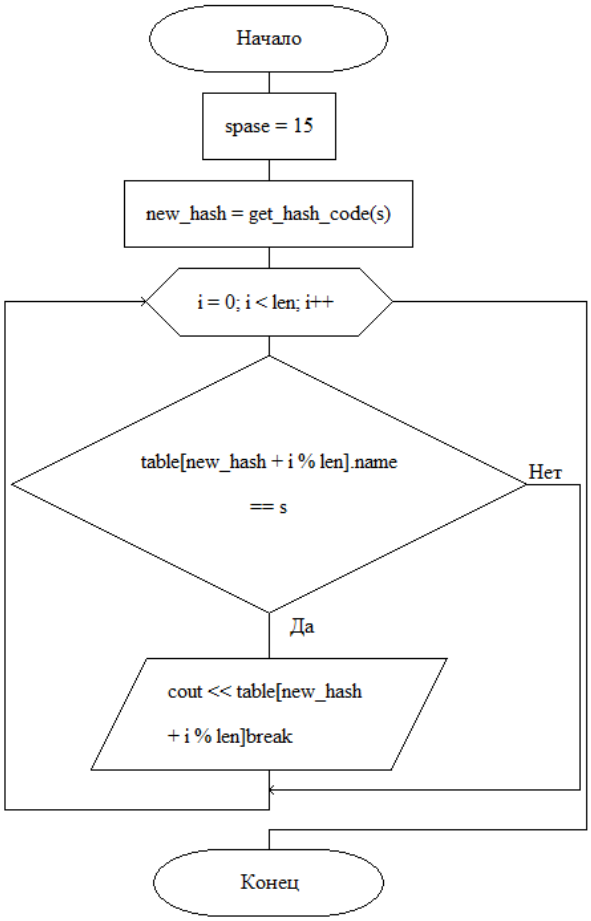


Рис.5 Схема алгоритма функции find_cell

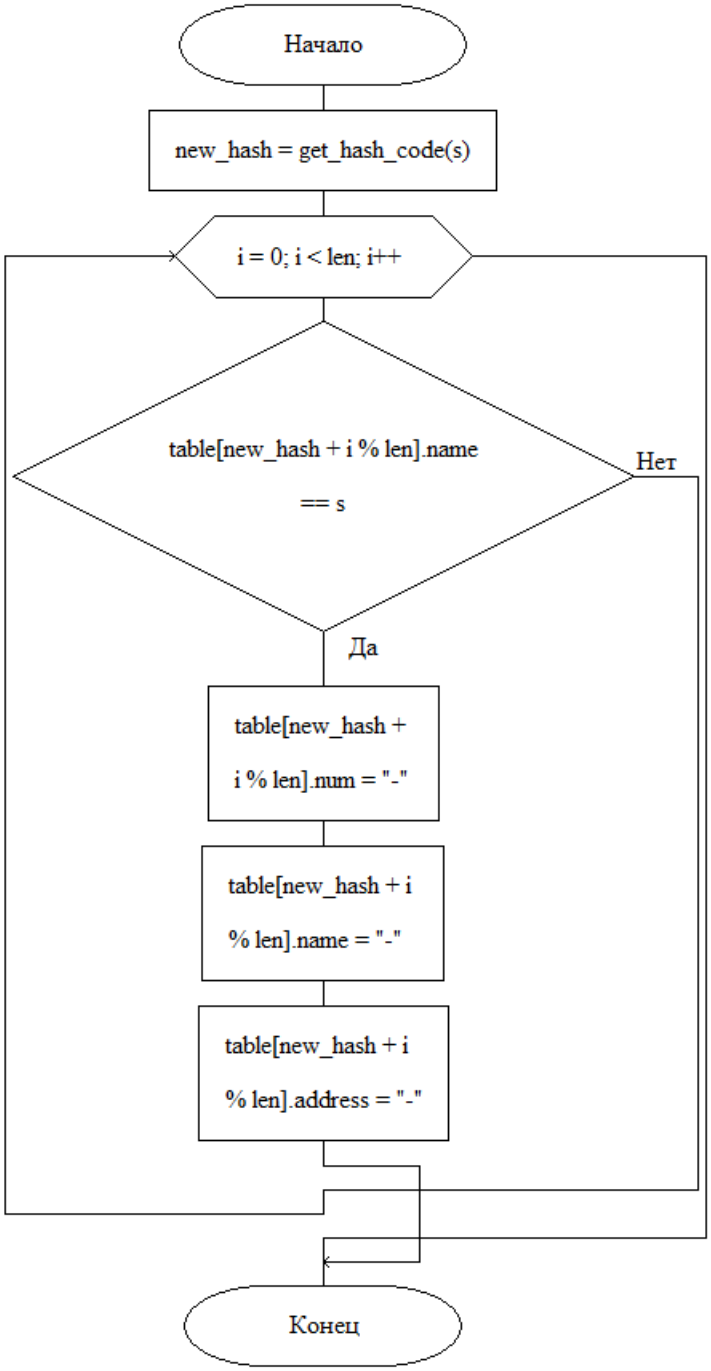


Рис.6 Схема алгоритма функции delete_cell

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Table.h"
void menu(Table table) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить счет." << endl;
	cout << "[2] - Удалить счет." << endl;
	cout << "[3] - Найти счет по владельцу." << endl;
	cout << "[4] - Вывести тыблицу." << endl;
	cout << "[5] - Завершить программу." << endl;
	cout << "---->";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
		cout << "Введите номер счета, имя владельца и адресс через пробелы" << endl;
		string num, name, address;
		cin >> num >> name >> address;
		table.add_cell(num, name, address);
		menu(table);
	}
	else if (ch == 2) {
		cout << "Введите имя владельца счета" << endl;
		string name;
		cin >> name;
		table.delete_cell(name);
		menu(table);
	}

	else if (ch == 3) {
		cout << "Введите имя владельца счета" << endl;
		string name;
		cin >> name;
		table.find_cell(name);
		menu(table);
	}
	else if (ch == 4) {
		table.show_table();
		menu(table);
	}
	else if (ch == 5) {
		cout << "Программа завершена" << endl;
	}

}
int main()
```

```
{
	setlocale(LC_ALL, "Russian");
	int size;
	cout << "Введите размер таблицы" << endl;
	cin >> size;
	Table table(size);
	menu(table);}
```

Table.h

```
#pragma once
#include <iostream>
#include <iomanip>
using namespace std;
struct Cell {
	string num = "-";
	string name = "-";
	string address = "-";
};
class Table
{
private:

	Cell* table;
	int len;
public:
	Table(int size);
	int get_hash_code(string s);
	void find_cell(string s);
	void add_cell(string num, string name, string adress);
	void show_table();
	void delete_cell(string s);
};
```

Table.cpp

```
#include "Table.h"
Table::Table(int size) {
	len = size;
	table = new Cell[size];
}

int Table::get_hash_code(string s) {
	int sum = 0;
	for (int i = 0; i < s.length(); i++) {
		sum += s[i]*s[i];
	}
	return sum % len;
}

void Table::add_cell(string num, string name, string address) {
	Cell* temp_cell = new Cell;
	temp_cell->num = num;
	temp_cell->address = address;
	temp_cell->name = name;

	int new_hash = get_hash_code(name);
	for (int i = 0; i < len; i++) {
		if (table[new_hash + i % len].name == "-") {
			table[new_hash + i % len] = *temp_cell;
```

```
                        break;
                }
                if (i == len - 1)    cout << "Свободных ячеек больше нет" << endl;
        }
}

void Table::show_table() {
        int spase = 15;
        cout << setw(spase) << "№ ячейки" << setw(spase)
                << "Номер счета" << setw(spase)
                << "Имя" << setw(spase)
                << "Адресс" << endl;
        for (int i = 0; i < len; i++) {
                cout << setw(spase) <<i<<setw(spase)
                        << table[i].num << setw(spase)
                        << table[i].name << setw(spase)
                        << table[i].address << endl;
        }
}

void Table::find_cell(string s) {
        int spase = 15;
        int new_hash = get_hash_code(s);
        for (int i = 0; i < len; i++) {
                if (table[new_hash + i % len].name == s) {
                        cout << setw(spase) << "№ ячейки" << setw(spase)
                                << "Номер счета" << setw(spase)
                                << "Имя" << setw(spase)
                                << "Адресс" << endl;
                        cout << setw(spase) << new_hash + i % len << setw(spase)
                                << table[new_hash + i % len].num << setw(spase)
                                << table[new_hash + i % len].name << setw(spase)
                                << table[new_hash + i % len].address << endl;

                        break;
                }
                if (i == len - 1)    cout << "Счет не найден" << endl;
        }


}

void Table::delete_cell(string s) {

        int new_hash = get_hash_code(s);
        for (int i = 0; i < len; i++) {
                if (table[new_hash + i % len].name == s) {
                        table[new_hash + i % len].num = "-";
                        table[new_hash + i % len].name = "-";
                        table[new_hash + i % len].address = "-";
                        break;
                }
                if (i == len - 1)    cout << "Счет не найден" << endl;
        }

}
```

20.Тестирование программы

Ниже представлен результат работы программы с бинарным файлом

```
Введите размер таблицы
5
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->1
Введите номер счета, имя владельца и адресс через пробелы
123786 Aaaa A123a43
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->1
Введите номер счета, имя владельца и адресс через пробелы
786235 Bbbbb B442b99
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->1
Введите номер счета, имя владельца и адресс через пробелы
239345 Ccc C9991c12
```

Рис.7 Скриншот добавления счетов в таблицу

```
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->4
      № ячейки    Номер счета           Имя          Адресс
              0              -             -               -
              1         239345           Ccc        C9991c12
              2         123786          Aaaa        A123a43
              3         786235         Bbbbb        B442b99
              4              -             -               -
```

Рис.8 Скриншот вывода таблицы

```
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->3
Введите имя владельца счета
Ccc
      № ячейки    Номер счета           Имя          Адресс
              1         239345           Ccc        C9991c12
```

Рис.9 Скриншот поиска счета по владельцу

```
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->2
Введите имя владельца счета
Аааа
Выберите команду:
[1] - Добавить счет.
[2] - Удалить счет.
[3] - Найти счет по владельцу.
[4] - Вывести тыблицу.
[5] - Завершить программу.
---->4
       № ячейки    Номер счета            Имя         Адресс
              0              -              -              -
              1         239345            Ccc       C9991c12
              2              -              -              -
              3         786235          Bbbbb       B442b99
              4              -              -              -
```

Рис.10 Скриншот удаления счета по владельцу

Выводы

1. В ходе работы была создана программа для работы с хеш-таблицами.
2. Также были реализованы функции записи, удаления, поиска и вывода данных в таблице.
3. Были изучены преимущества и недостатки хранения данных в хеш-таблице:
4. Преимущества: хеш-таблица обладает невероятной скоростью, что позволяет взаимодействовать с данными даже в очень больших таблицах практически моментально. Это возможно благодаря сложности O(1).
5. Недостатки: требует немного больше памяти чем обычные массивы или списки.
6. Таким образом, была изучена работа алгоритмов хеширования.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание хеш-функций. интернет-ресурс: https://ru.wikipedia.org/wiki/Хеш-функция (Дата обращения – 02.11.2020).

Практическая работа № 10

СЖАТИЕ ДАННЫХ

Постановка задачи

Составить программу, реализующую кодирование строки с помощью алгоритма Хаффмана

1. Описание алгоритма

Построение кода Хаффмана сводится к построению соответствующего бинарного дерева по следующему алгоритму:

1. Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента с весом, равным частоте появления символа в строке.
2. Из списка выберем два узла с наименьшим весом.

3. Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.
4. Добавим к списку только что сформированный узел вместо двух объединенных узлов.
5. Если в списке больше одного узла, то повторим пункты со второго по пятый.

Алгоритм реализован с использованием классов Haffman и Los и одной структуры, являющейся узлом перевернутого дерева.

Методы класса Huffman:

void set_string - функция записывает строку в память.

void make_tree - функция создает дерево шифрования.

void compress – функция зашифровывает строку

string get_code – функция возвращает полученный код

Методы класса Los:

void add_ell – функция добавляет элемент в список

Node *find_ell – функция возвращает указатель на элемент по ключу

void inc_ell – функция инкрементирует поле number структуры

int count_true – функция возвращает количество записей, в которых поле is_last_layer является истиной.

int *pare_min – функция возвращает массив из 2-х id записей с наименьшими полями number.

int get_id – функция возвращает id записи

void inc_id – функция инкрементирует поле counter_id в классе Los

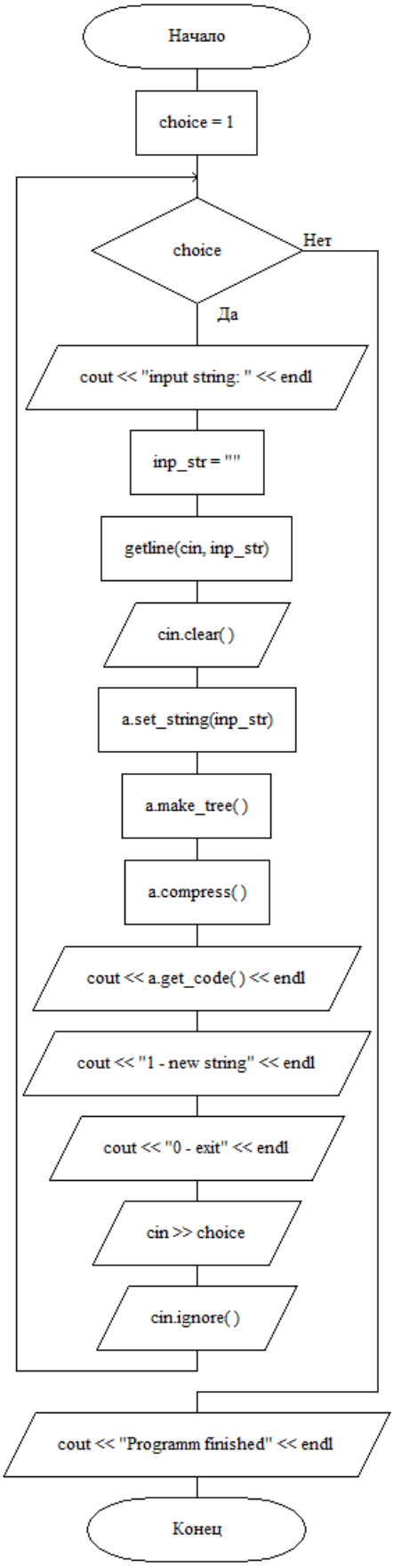


Рис.1 Схема алгоритма функции main

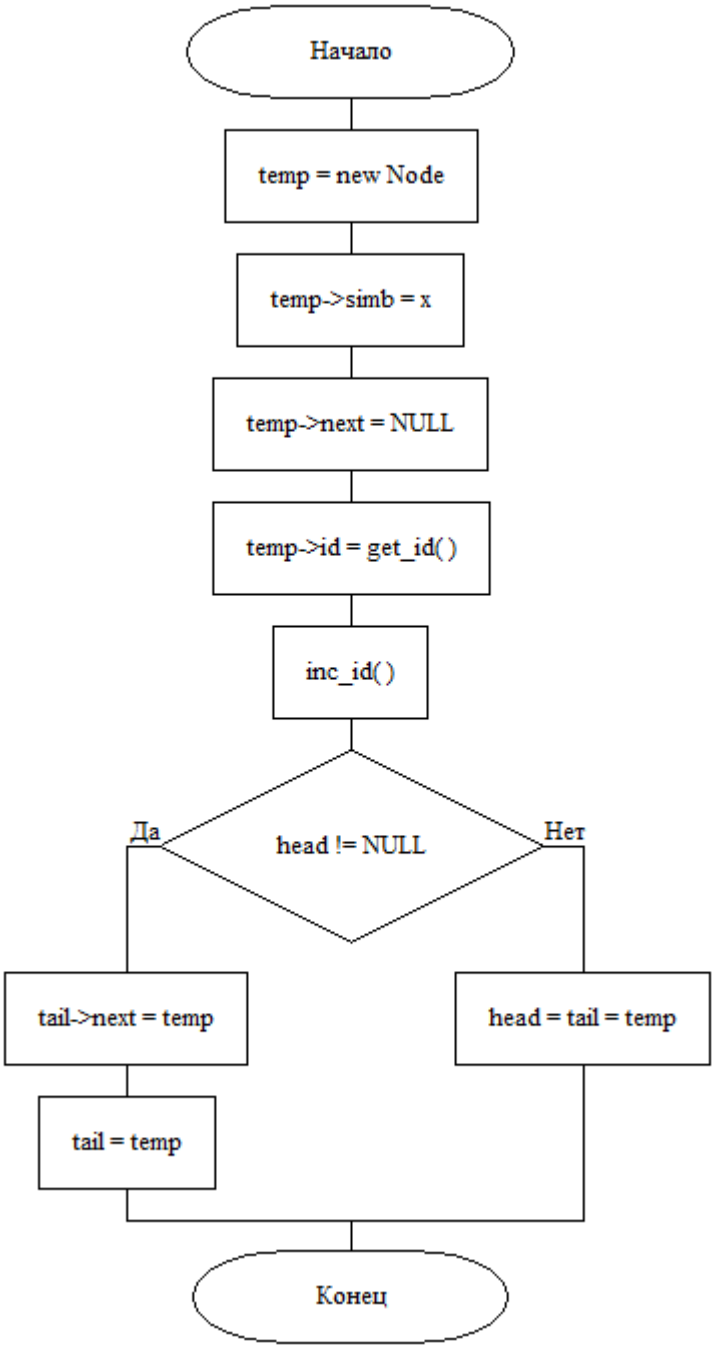


Рис.2 Схема алгоритма функции add_ell(char x)

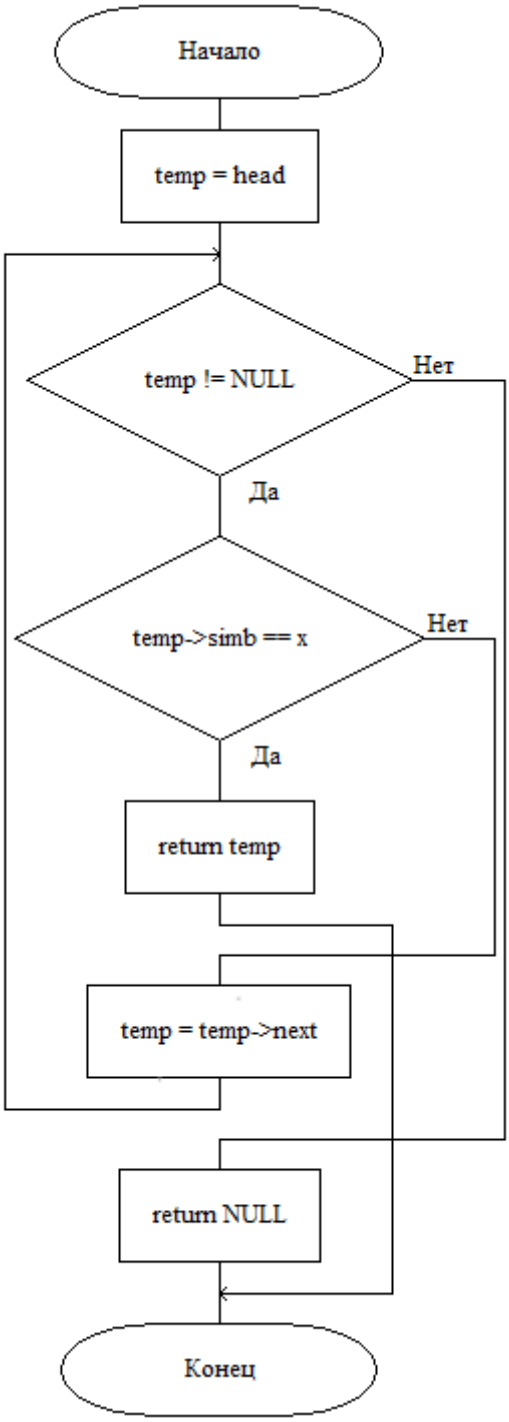


Рис.3 Схема алгоритма функции find_ell(char x)

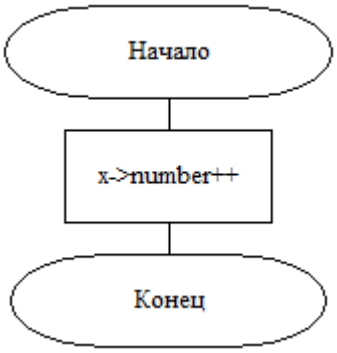


Рис.4 Схема алгоритма функции inc_ell(Node *x)

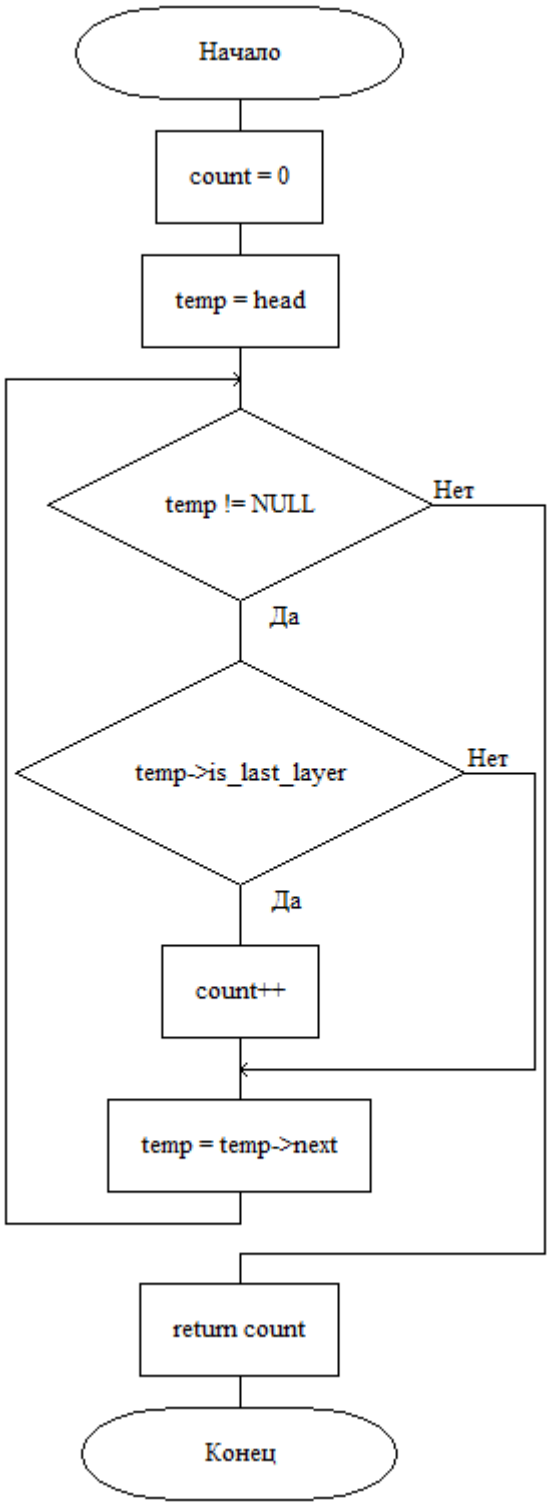


Рис.5 Схема алгоритма функции count_true()

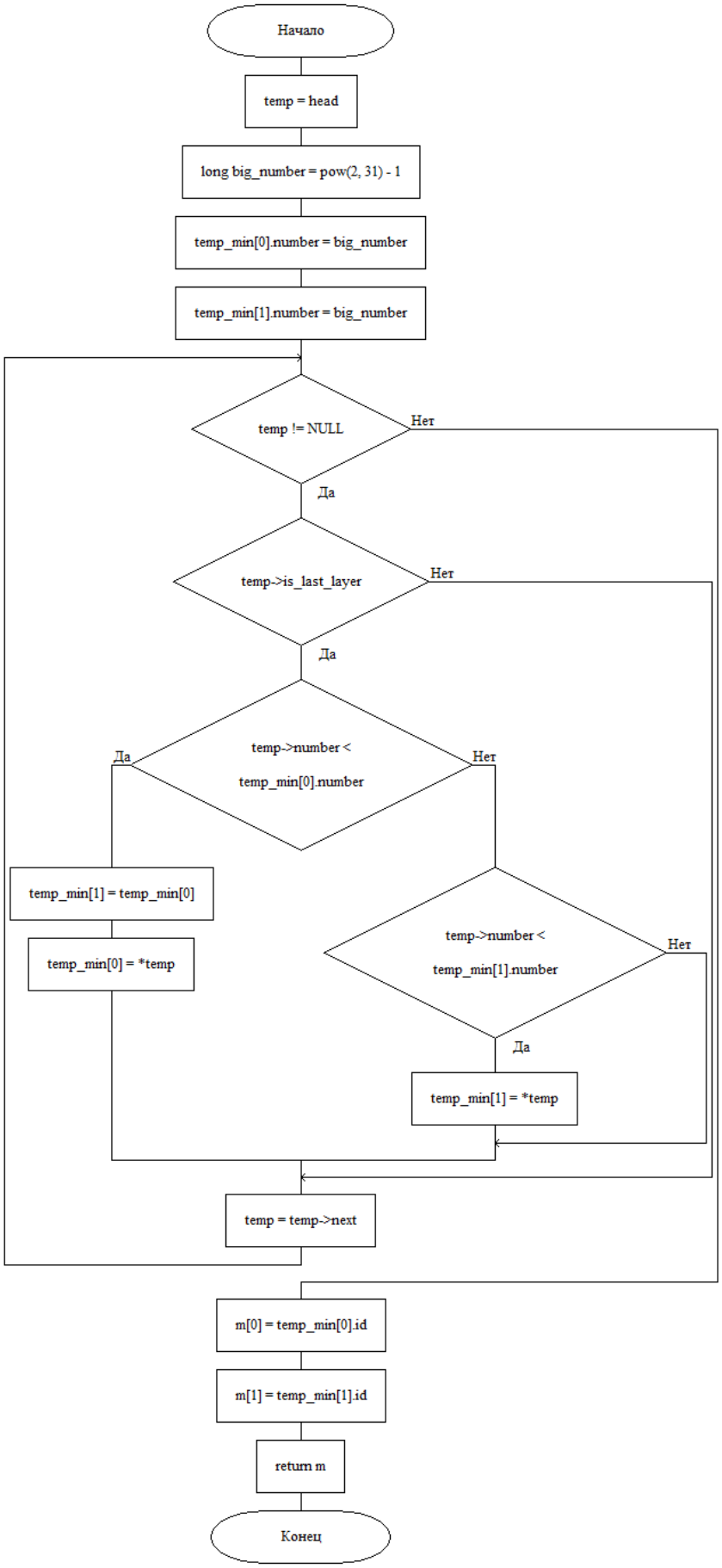

Начало
temp = head
long big_number = pow(2, 31) - 1
temp_min[0].number = big_number
temp_min[1].number = big_number
temp != NULL
Нет
Да
temp->is_last_layer
Нет
Да
temp->number <
temp_min[0].number
Да
Нет
temp_min[1] = temp_min[0]
temp_min[0] = *temp
temp->number <
temp_min[1].number
Нет
Да
temp_min[1] = *temp
temp = temp->next
m[0] = temp_min[0].id
m[1] = temp_min[1].id
return m
Конец

Рис.6 Схема алгоритма функции pare_min()

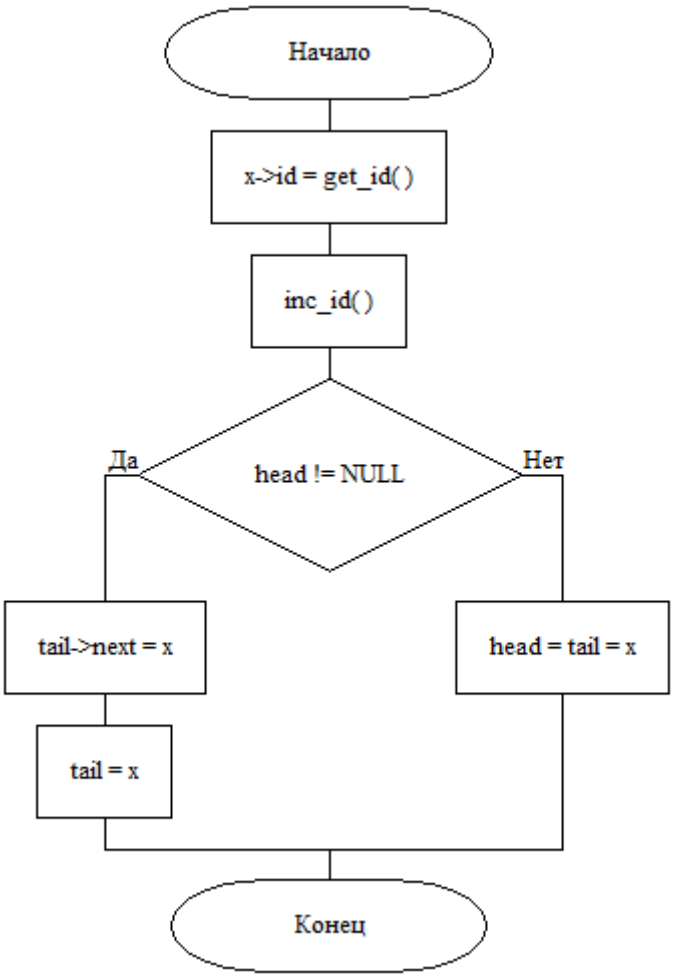


Рис.7 Схема алгоритма функции add_ell(Node *x)

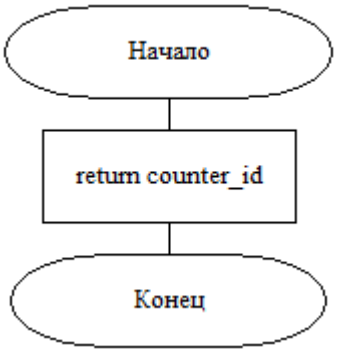


Рис.8 Схема алгоритма функции get_id()

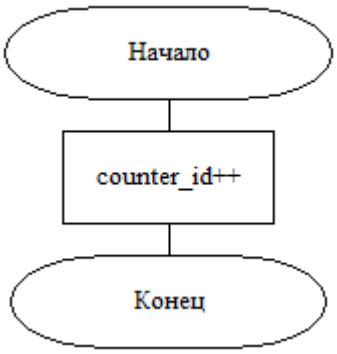


Рис.9 Схема алгоритма функции inc_id()

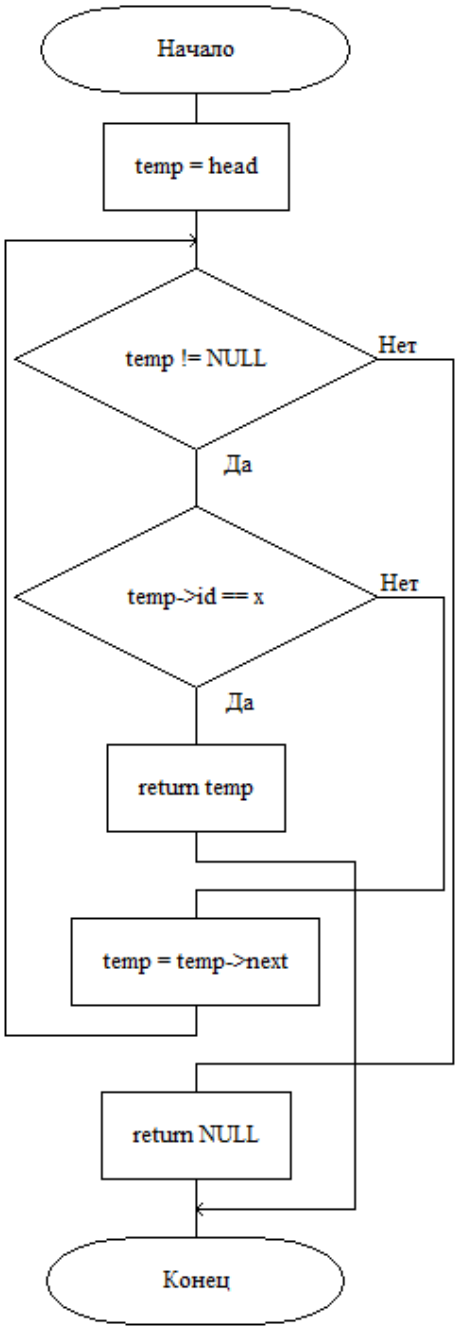


Рис.10 Схема алгоритма функции find_ell(int x)

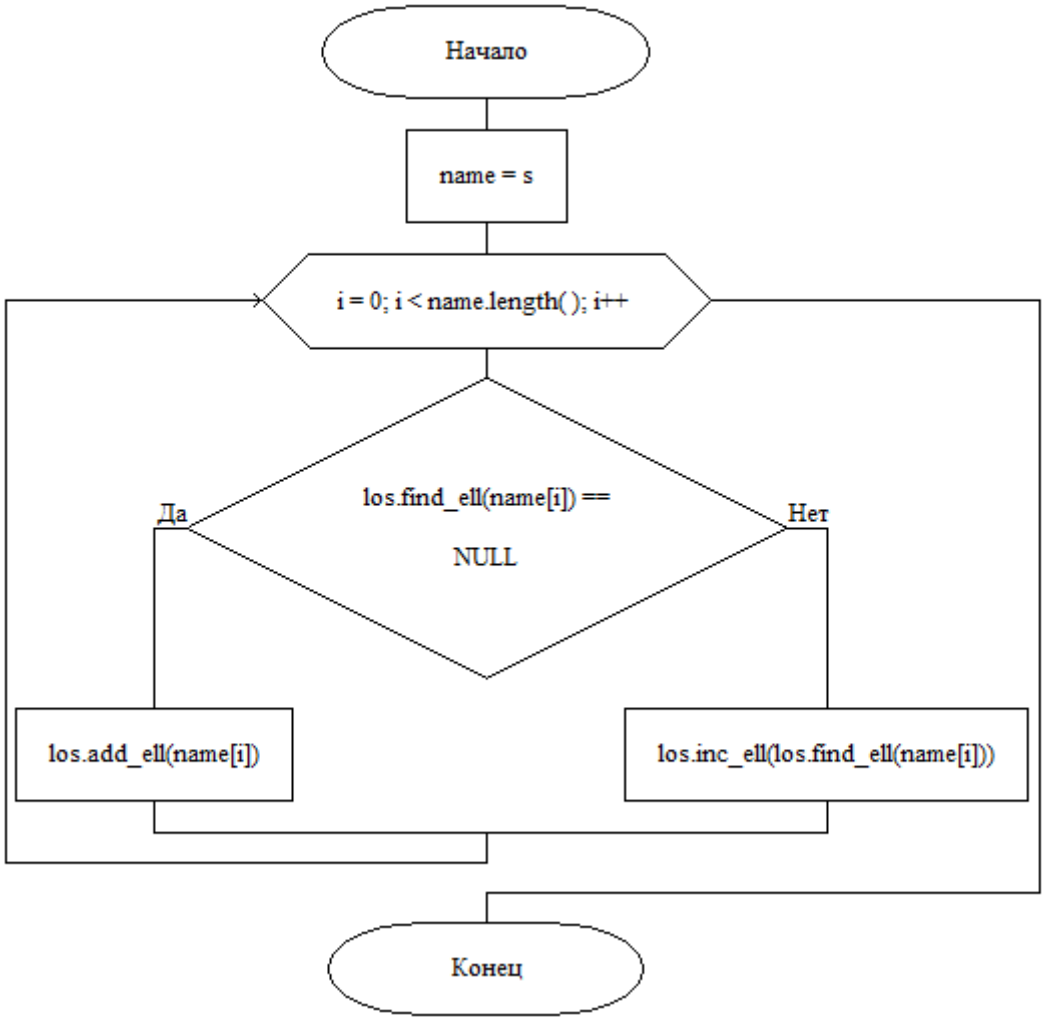


Рис.11 Схема алгоритма функции set_string(string s)

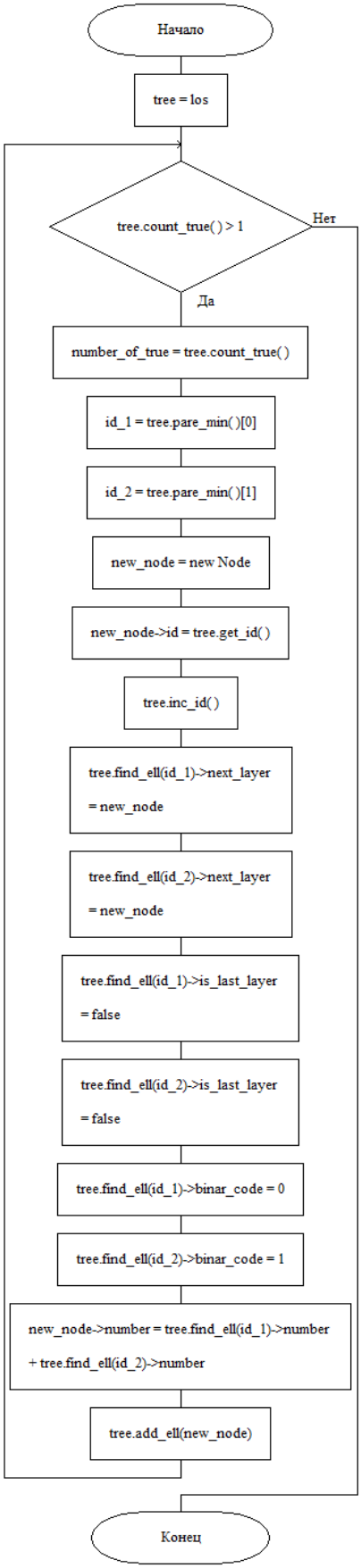


Рис.12 Схема алгоритма функции make_tree()

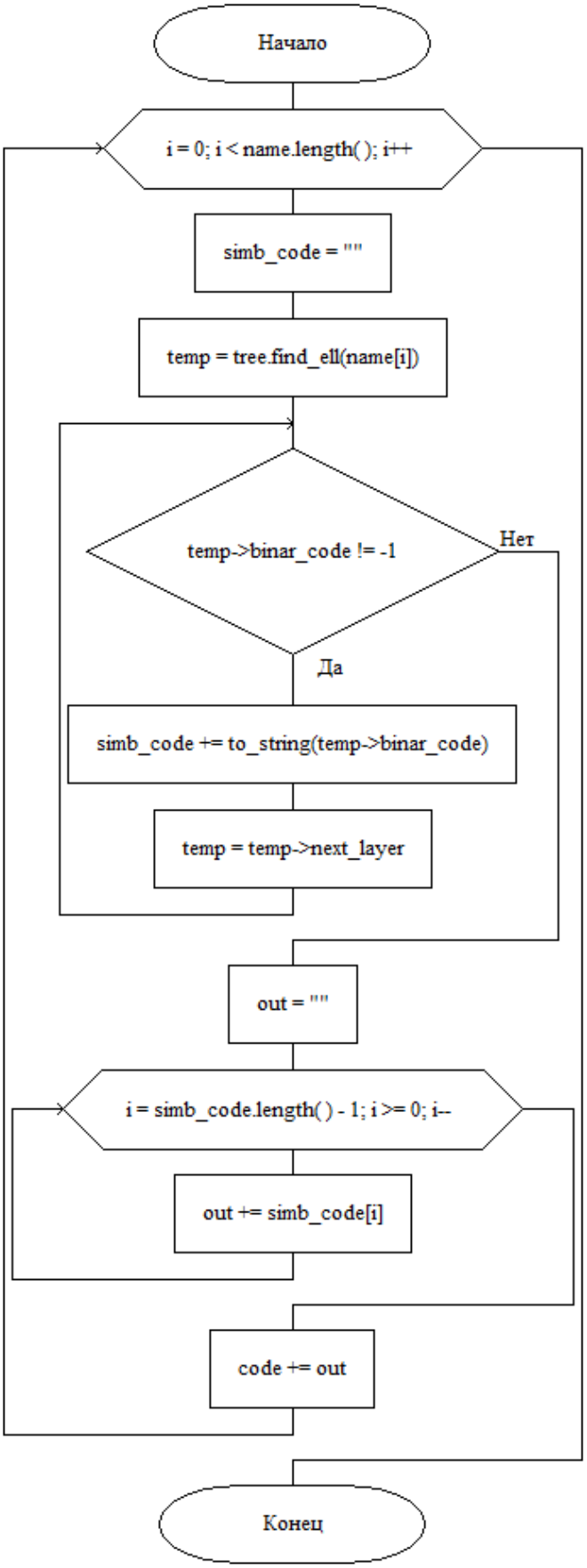

Начало
i = 0; i < name.length( ); i++
simb_code = ""
temp = tree.find_ell(name[i])
temp->binar_code != -1
Нет
Да
simb_code += to_string(temp->binar_code)
temp = temp->next_layer
out = ""
i = simb_code.length( ) - 1; i >= 0; i--
out += simb_code[i]
code += out
Конец

Рис.13 Схема алгоритма функции compress()

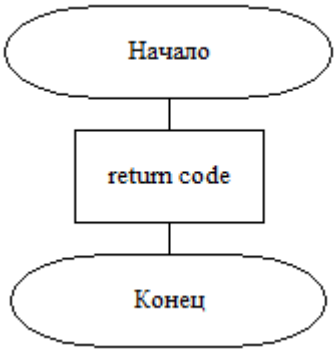


Рис.14 Схема алгоритма функции get_code()

21. Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include <iostream>
#include "Huffman.h"
#include <string>


int main() {

    int choice = 1;
    while (choice) {
        Huffman a;
        cout << "input string: " << endl;
        string inp_str = "";
        getline(cin, inp_str);
        cin.clear();
        a.set_string(inp_str);
        a.make_tree();
        a.compress();
        cout << a.get_code() << endl;
        cout << "1 - new string" << endl;
        cout << "0 - exit" << endl;
        cin >> choice;
        cin.ignore();


    }
    cout << "Programm finished" << endl;


}
```

Los.h

```
#ifndef PR_10_LOS_H
#define PR_10_LOS_H

#include <iostream>

struct Node {
    Node *next = NULL;
    int id = NULL;
    char simb = '_';
    int number = 1;
    Node *next_layer = NULL;
    int binar_code = -1;
    bool is_last_layer = true;

};

class Los {
private:
    Node *head, *tail;
    int counter_id = 0;
public:

    Los() : head(NULL), tail(NULL) {};

    void add_ell(char x);

    void add_ell(Node *x);

    Node *find_ell(char x);

    Node *find_ell(int x);

    void inc_ell(Node *x);

    int count_true();

    int *pare_min();

    int get_id();

    void inc_id();


};


#endif
```

Los.cpp

```
#include "Los.h"
#include <iostream>


void Los::add_ell(char x) {
    Node *temp = new Node;
    temp->simb = x;
```

```
    temp->next = NULL;
    temp->id = get_id();
    inc_id();

    if (head != NULL) {
        tail->next = temp;
        tail = temp;
    } else head = tail = temp;
};

Node *Los::find_ell(char x) {
    Node *temp = head;
    while (temp != NULL) {
        if (temp->simb == x) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

void Los::inc_ell(Node *x) {
    x->number++;
}

int Los::count_true() {
    int count = 0;
    Node *temp = head;
    while (temp != NULL) {
        if (temp->is_last_layer) count++;
        temp = temp->next;

    }
    return count;
}


int *Los::pare_min() {
    int m[2];
    Node *temp = head;
    static Node temp_min[2];
    unsigned long big_number = pow(2, 31) - 1;
    temp_min[0].number = big_number;
    temp_min[1].number = big_number;

    while (temp != NULL) {
        if (temp->is_last_layer) {
            if (temp->number < temp_min[0].number) {
                temp_min[1] = temp_min[0];
                temp_min[0] = *temp;
            } else if (temp->number < temp_min[1].number) { temp_min[1] = *temp;
}

        }
        temp = temp->next;
    }
    m[0] = temp_min[0].id;
    m[1] = temp_min[1].id;


    return m;
```

```
}

void Los::add_ell(Node *x) {
    x->id = get_id();
    inc_id();
    if (head != NULL) {
        tail->next = x;
        tail = x;
    } else head = tail = x;
}

int Los::get_id() {
    return counter_id;
}

void Los::inc_id() {
    counter_id++;
}

Node *Los::find_ell(int x) {
    Node *temp = head;
    while (temp != NULL) {
        if (temp->id == x) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

Haffman.h

```
#ifndef PR_10_HUFFMAN_H
#define PR_10_HUFFMAN_H

#include <iostream>
#include <list>
#include "Los.h"

using namespace std;

class Huffman {
private:
    Los los;
    Los tree;
    string name = "A";
    string code = "";

public:
    void set_string(string s);

    void make_tree();

    void compress();

    string get_code();
```

```
};


#endif
```

Haffman.cpp

```
#include "Huffman.h"

#include <string>

void Huffman::set_string(string s) {
    name = s;
    for (int i = 0; i < name.length(); i++) {
        if (los.find_ell(name[i]) == NULL) {
            los.add_ell(name[i]);
        } else {
            los.inc_ell(los.find_ell(name[i]));
        }
    }

}

void Huffman::make_tree() {
    tree = los;
    while (tree.count_true() > 1) {
        int number_of_true = tree.count_true();

        int id_1 = tree.pare_min()[0];
        int id_2 = tree.pare_min()[1];
        Node *new_node = new Node;
        new_node->id = tree.get_id();
        tree.inc_id();
        tree.find_ell(id_1)->next_layer = new_node;
        tree.find_ell(id_2)->next_layer = new_node;
        tree.find_ell(id_1)->is_last_layer = false;
        tree.find_ell(id_2)->is_last_layer = false;
        tree.find_ell(id_1)->binar_code = 0;
        tree.find_ell(id_2)->binar_code = 1;
        new_node->number = tree.find_ell(id_1)->number + tree.find_ell(id_2)-
>number;
        tree.add_ell(new_node);

    }
}


void Huffman::compress() {

    for (int i = 0; i < name.length(); i++) {
        string simb_code = "";
        Node *temp = tree.find_ell(name[i]);
        while (temp->binar_code != -1) {
            simb_code += to_string(temp->binar_code);
            temp = temp->next_layer;
        }
        string out = "";
        for (int i = simb_code.length() - 1; i >= 0; i--)
```

```
            out += simb_code[i];
        code += out;
    }


}

string Huffman::get_code() {
    return code;
}
```

22.Тестирование программы

```
input string:
aaabbcccc
11111110100000
1 - new string
0 - exit
1
input string:
Mironov Aleksey
01100111100000110010011010101111001101010111011110100000
1 - new string
0 - exit
0
Programm finished
```

Рис.15 Скриншот шифрования двух строк (1 – произвольная, 2 - Фамилия, имя)

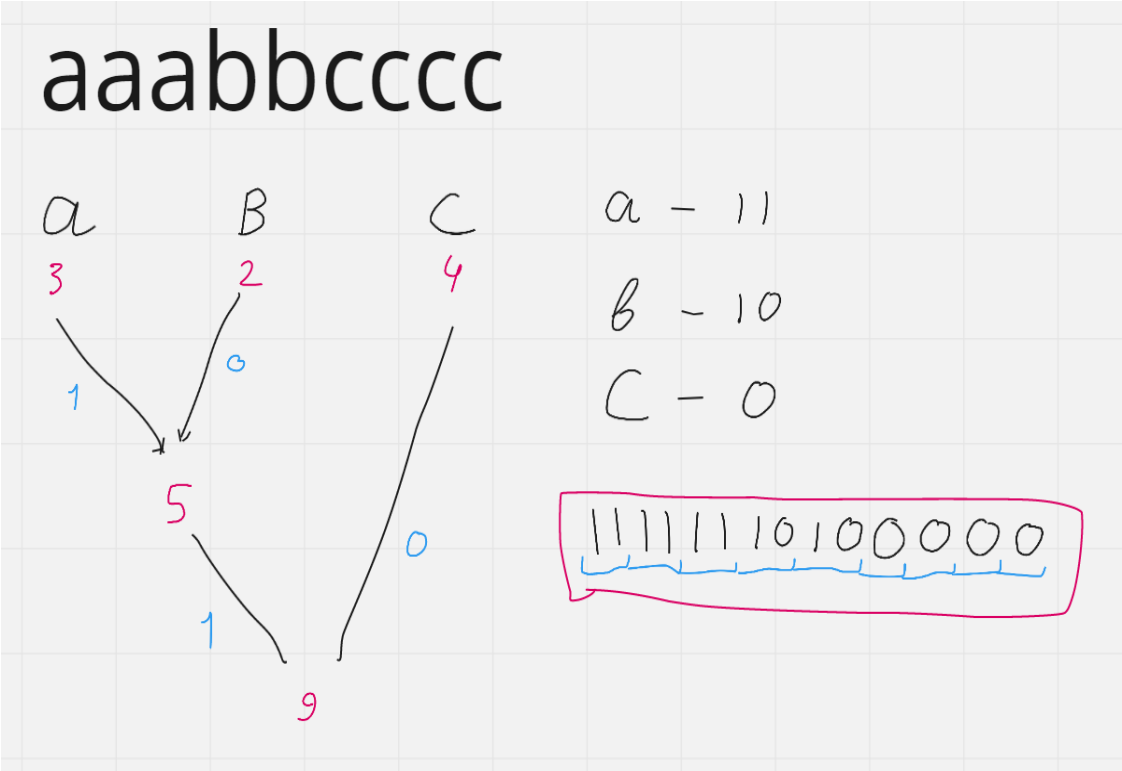


Рис.16 Скриншот ручной проверки алгоритма на строке из рис.1

23.Выводы

1. В ходе данной работы были изучены принципы работы сжатия данных
2. На практике изучен и реализован алгоритм Хаффмана.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание алгоритма Хаффмана. интернет-ресурс: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Хаффмана (Дата обращения – 02.11.2020).

Практическая работа № 11

ПОСТРОЕНИЕ ОСТОВНОГО ДЕРЕВА ГРАФА

Вариант-11.1.3

Постановка задачи

Составить программу реализации алгоритма Крускала построения остовного дерева минимального веса.

Выбрать и реализовать способ представления графа в памяти.

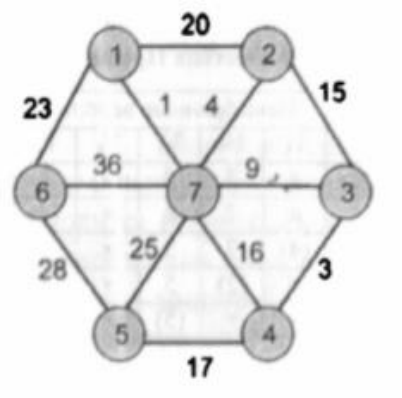
Предусмотреть ввод с клавиатуры произвольного графа.

Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы для заданного графа в соответствии с индивидуальным заданием

Индивидуальное задание:

11.1.3



24. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add** – функция добавления связи в граф.
- **void show**– функция графа.
- **void sort** – функция сортировки графа по весам.
- **void make_min_ostav**– функция создания минимального оставного дерева.

Алгоритм Краскала — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера. В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса. Подробное описание алгоритма можно найти в литературе.

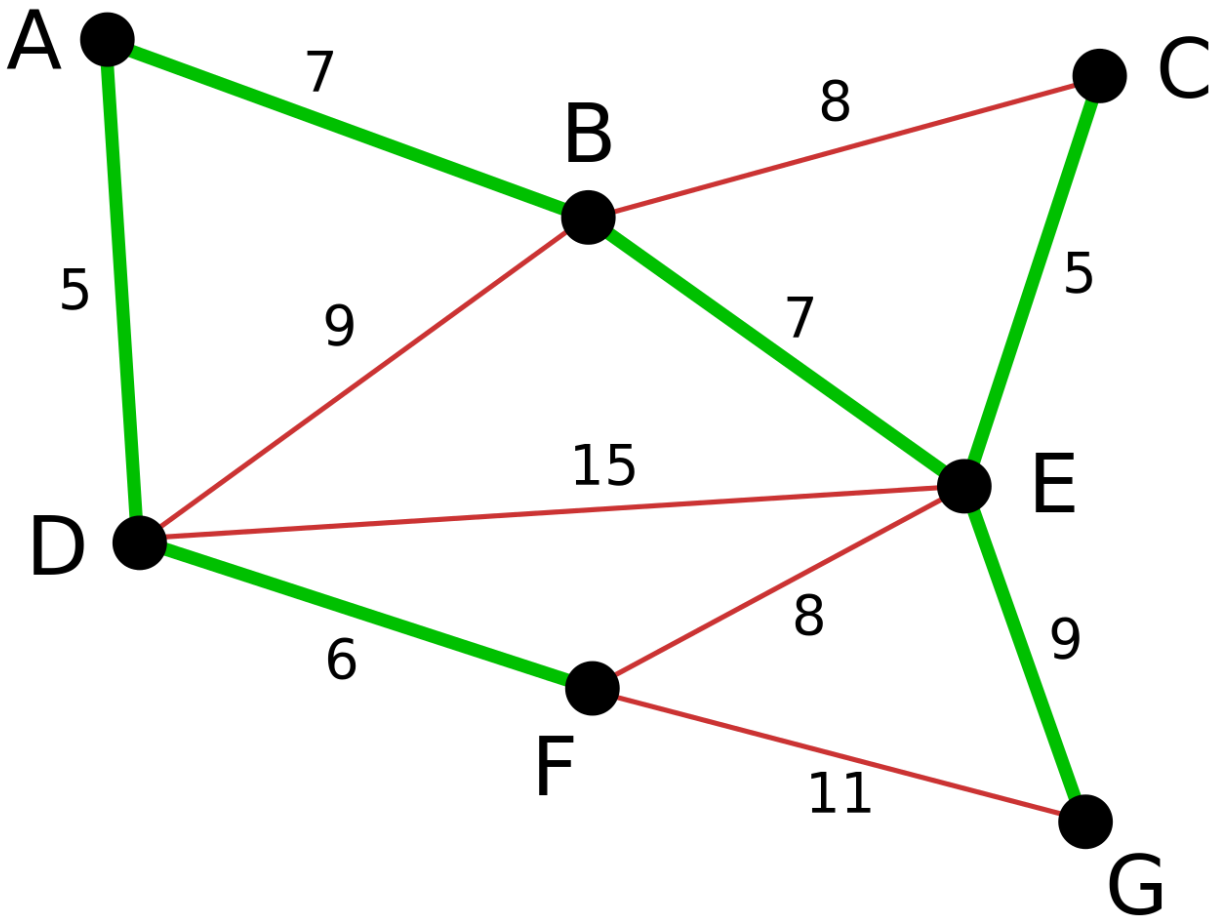


Рис.1 Схема минимального остовного дерева

Функция main создает объект класса Graph и вызывает меню.

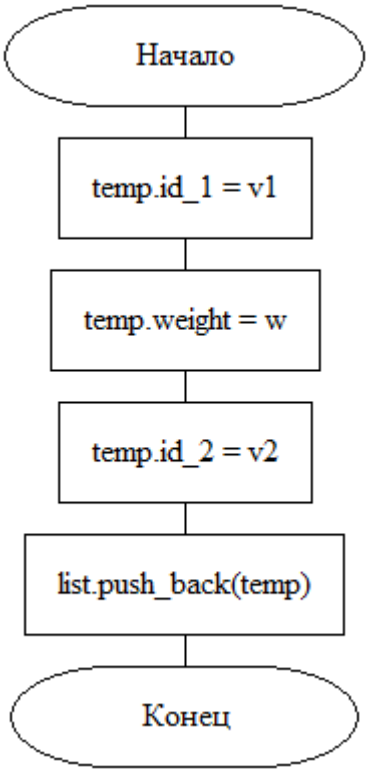


Рис.2 Схема алгоритма функции add

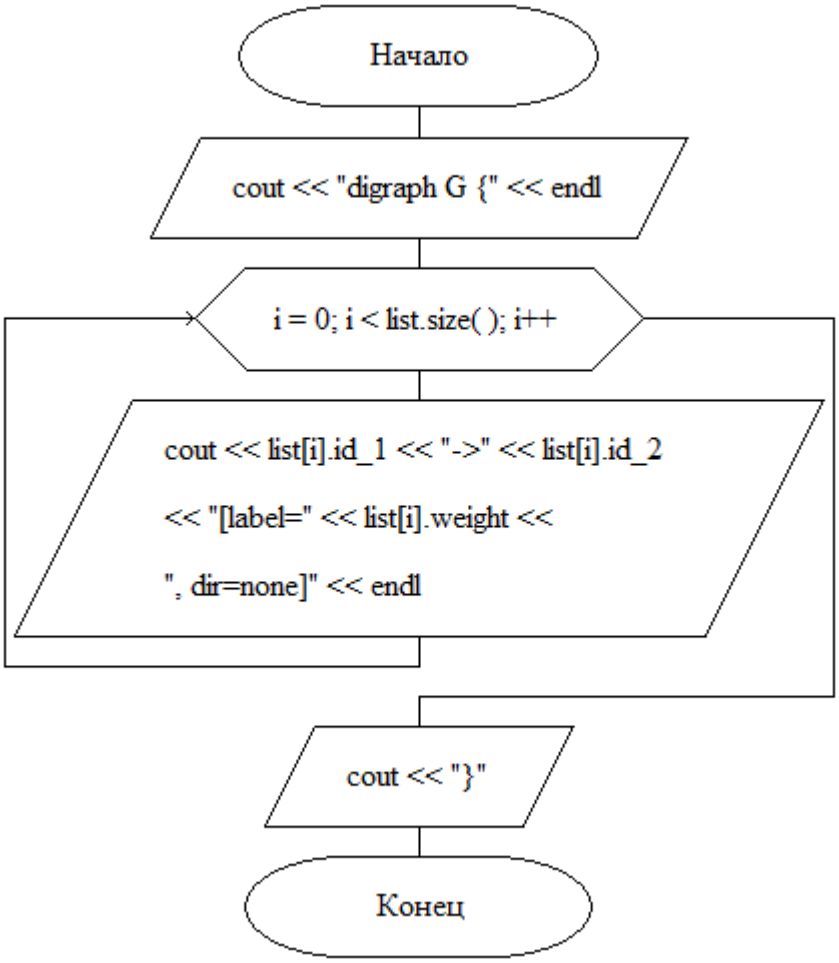


Рис.3 Схема алгоритма функции show

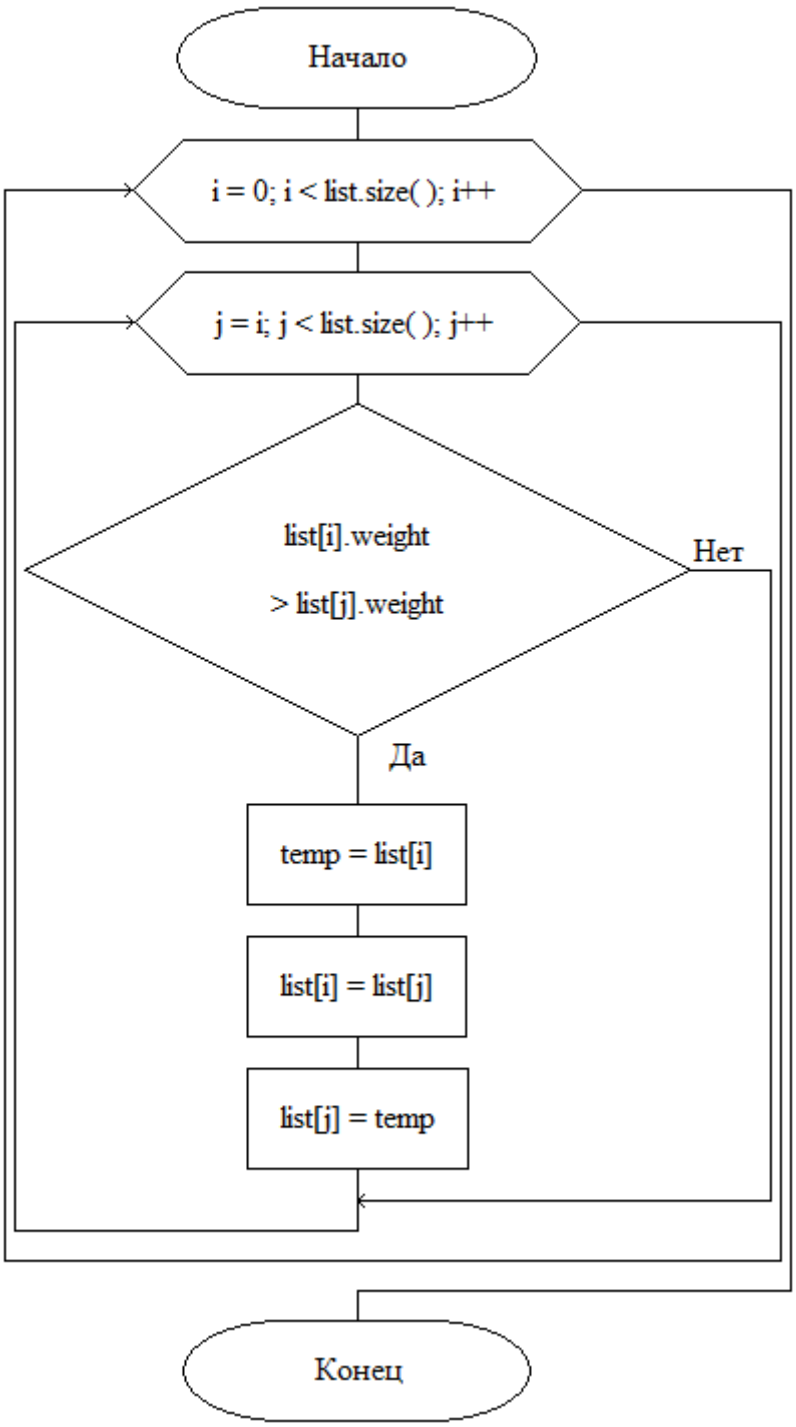


Рис.4 Схема алгоритма функции sort

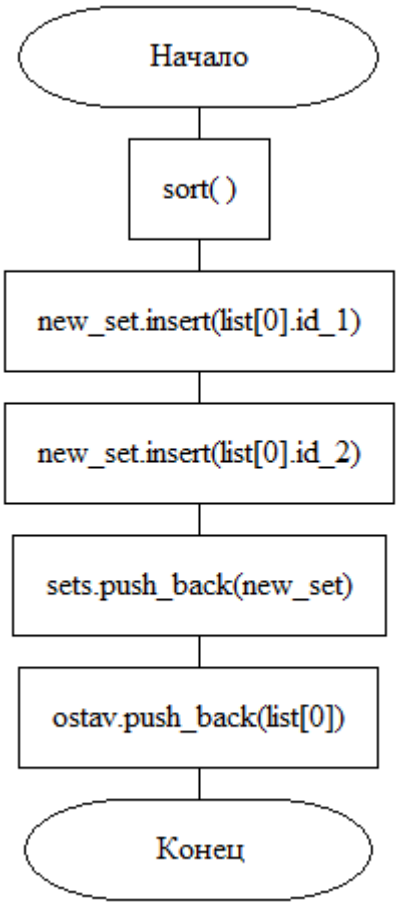


Рис.5 Схема алгоритма функции вставки первой записи

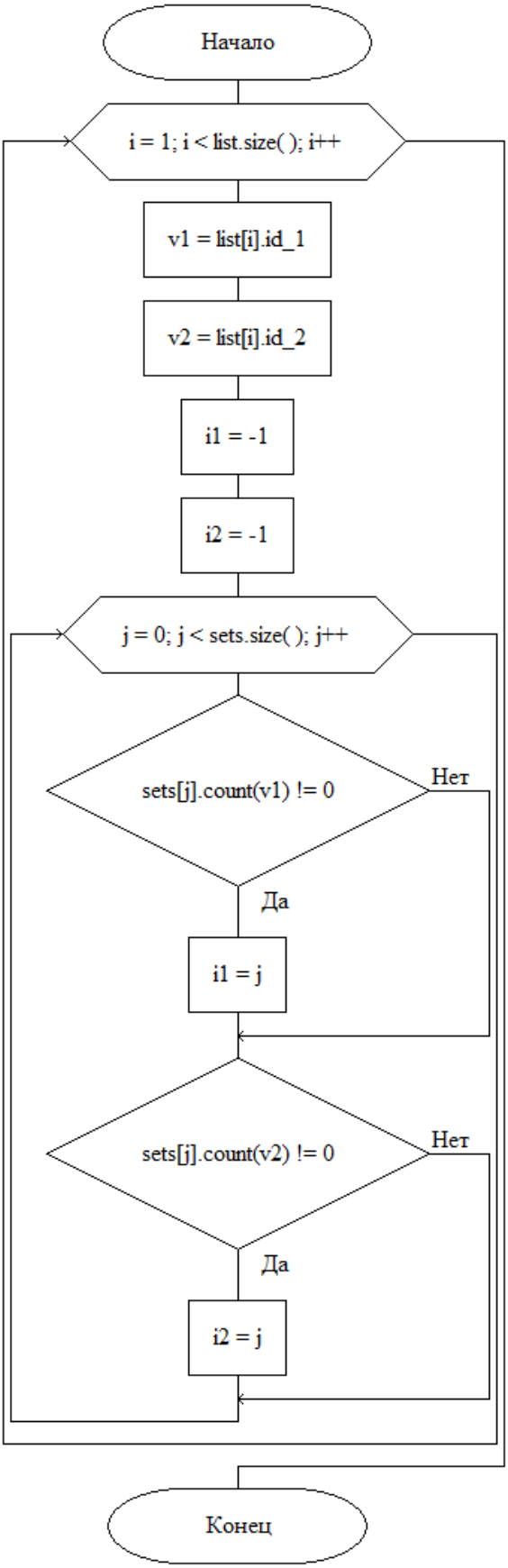


Рис.6 Схема алгоритма функции поиска индексов во множествах

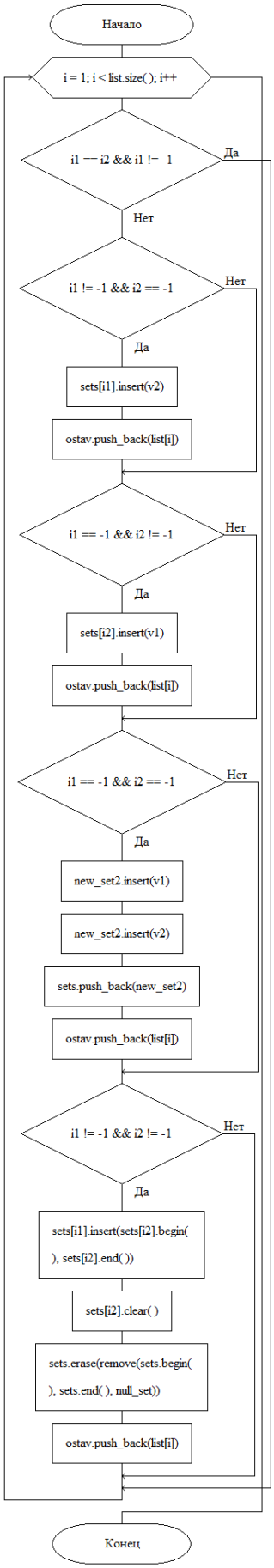

Начало
i = 1; i < list.size( ); i++
i1 == i2 && i1 != -1
Да
Нет
i1 != -1 && i2 == -1
Нет
Да
sets[i1].insert(v2)
ostav.push_back(list[i])
i1 == -1 && i2 != -1
Нет
Да
sets[i2].insert(v1)
ostav.push_back(list[i])
i1 == -1 && i2 == -1
Нет
Да
new_set2.insert(v1)
new_set2.insert(v2)
sets.push_back(new_set2)
ostav.push_back(list[i])
i1 != -1 && i2 != -1
Нет
Да
sets[i1].insert(sets[i2].begin( ), sets[i2].end( ))
sets[i2].clear( )
sets.erase(remove(sets.begin( ), sets.end( ), null_set))
ostav.push_back(list[i])
Конец

Рис.5 Схема алгоритма функции make_min_ostav

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Graph.h"
void menu(Graph g) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить связь." << endl;
	cout << "[2] - Вывести граф (в DOT-нотации)." << endl;
	cout << "[3] - Вывести минимальное оставное дерево (в DOT-нотации)." << endl;
	cout << "[4] - Завершить программу." << endl;
	cout << "---->";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
		cout << "Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)"
<< endl;
		int x1, x2, x3;
		cin >> x1 >> x2 >> x3;
		g.add(x1, x2, x3);
		menu(g);
	}
	if (ch == 2) {
		g.show();
		menu(g);
	}
	if (ch == 3) {
		g.make_min_ostav();
		menu(g);
	}
	if (ch == 4) {
		cout << "Программа завершена";
	}
}

int main()
{
	setlocale(LC_ALL, "Russian");
    Graph a;
    menu(a);
}
```

Graph.h

```
#pragma once
#include <set>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Connection
{
	int id_1 = 0;
	int weight = 0;
```

```
	int id_2 = 0;
};
class Graph
{
private:
	vector<Connection> list;

public:
	void add(int v1, int w, int v2);
	void show();
	void sort();
	void make_min_ostav();
};
```

Graph.cpp

```
#include "Graph.h"

void Graph::add(int v1, int w, int v2) {
	Connection temp;
	temp.id_1 = v1;
	temp.weight = w;
	temp.id_2 = v2;

	list.push_back(temp);

}

void Graph::show() {
	cout << "digraph G {" << endl;
	for (int i = 0; i < list.size(); i++) {

		cout << list[i].id_1 << "->" << list[i].id_2 << "[label=" << list[i].weight <<
", dir=none]" << endl;

	}
	cout << "}";
}

void Graph::sort() {
	for (int i = 0; i < list.size(); i++) {
		for (int j = i; j < list.size(); j++) {
			if (list[i].weight > list[j].weight) {
				Connection temp = list[i];
				list[i] = list[j];
				list[j] = temp;
			}
		}
	}
}

void Graph::make_min_ostav() {
	sort();
	set<int> null_set;
	vector<set<int>> sets;
	vector<Connection> ostav;
	set<int> new_set;
	new_set.insert(list[0].id_1);
	new_set.insert(list[0].id_2);
	sets.push_back(new_set);
```

```
	ostav.push_back(list[0]);
	for (int i = 1; i < list.size(); i++) {
		int v1 = list[i].id_1;
		int v2 = list[i].id_2;
		int i1 = -1;
		int i2 = -1;
		for (int j = 0; j < sets.size(); j++) {
			if (sets[j].count(v1) != 0) i1 = j;
			if (sets[j].count(v2) != 0) i2 = j;
		}
		if (i1 == i2 && i1!=-1) {
			continue;
		}
		if (i1 != -1 && i2 == -1) {
			sets[i1].insert(v2);
			ostav.push_back(list[i]);
		}
		if (i1 == -1 && i2 != -1) {
			sets[i2].insert(v1);
			ostav.push_back(list[i]);
		}
		if (i1 == -1 && i2 == -1) {
			set<int> new_set2;
			new_set2.insert(v1);
			new_set2.insert(v2);
			sets.push_back(new_set2);
			ostav.push_back(list[i]);
		}
		if (i1 != -1 && i2 != -1) {
			sets[i1].insert(sets[i2].begin(), sets[i2].end());
			sets[i2].clear();
			sets.erase(remove(sets.begin(), sets.end(), null_set));
			ostav.push_back(list[i]);
		}

	}
	cout << "digraph G {" << endl;
	for (int i = 0; i < ostav.size(); i++) {

		cout << ostav[i].id_1 << "->" << ostav[i].id_2 <<"[label="<<ostav[i].weight<<",
dir=none]"<<endl;

	}
	cout << "}";
	}
```

25.Тестирование программы

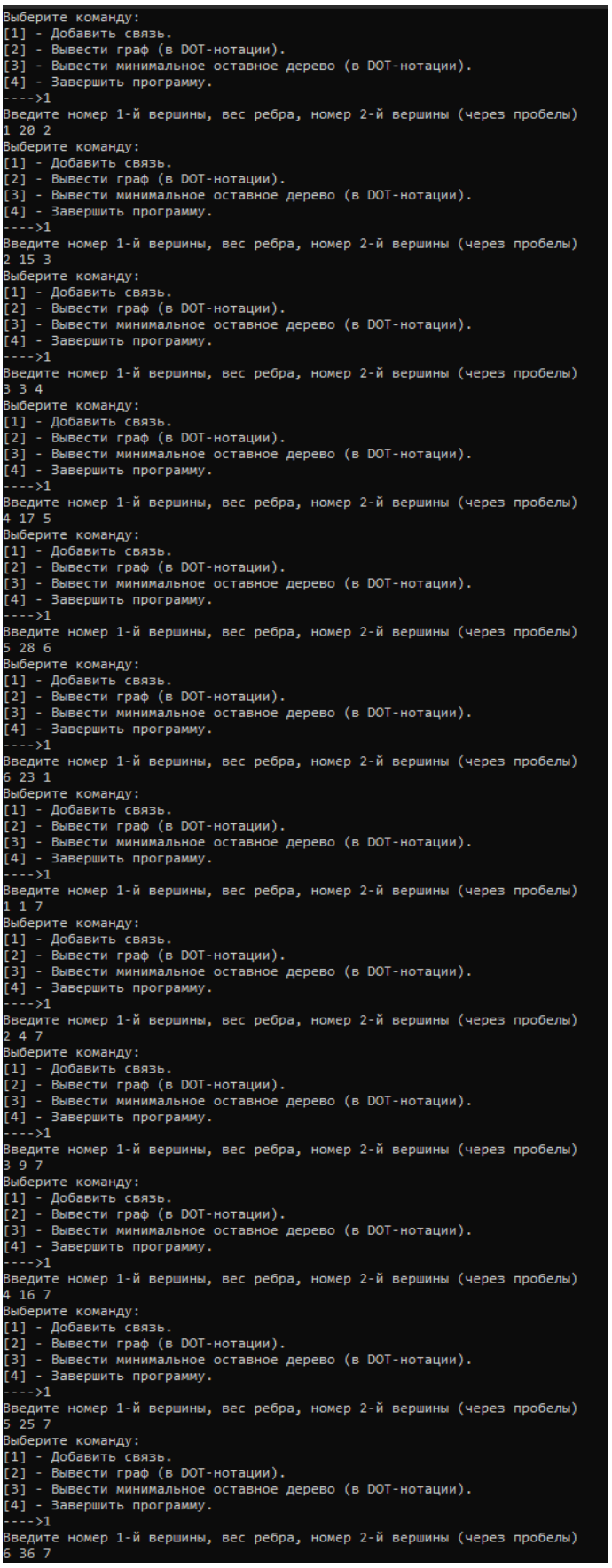


Рис.6 Скриншот добавления связей в граф

```
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->
2
digraph G {
1->2[label=20, dir=none]
2->3[label=15, dir=none]
3->4[label=3, dir=none]
4->5[label=17, dir=none]
5->6[label=28, dir=none]
6->1[label=23, dir=none]
1->7[label=1, dir=none]
2->7[label=4, dir=none]
3->7[label=9, dir=none]
4->7[label=16, dir=none]
5->7[label=25, dir=none]
6->7[label=36, dir=none]
```

Рис.7 Скриншот вывода исходного графа в DOT-нотации

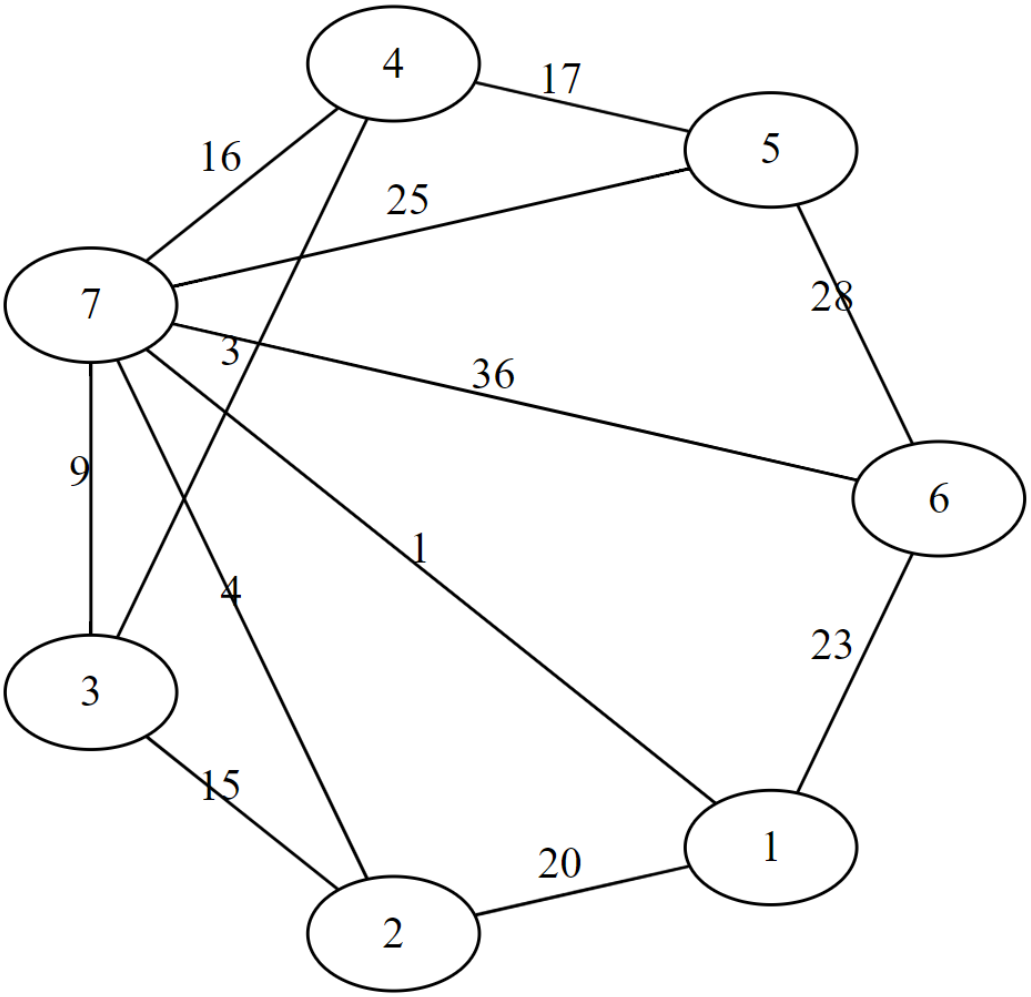


Рис.8 Скриншот построения графа по полученной DOT-нотации

```
}Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->3
digraph G {
1->7[label=1, dir=none]
3->4[label=3, dir=none]
2->7[label=4, dir=none]
3->7[label=9, dir=none]
4->5[label=17, dir=none]
6->1[label=23, dir=none]
```

Рис.9 Скриншот вывода минимального остовного дерева в DOT-нотации

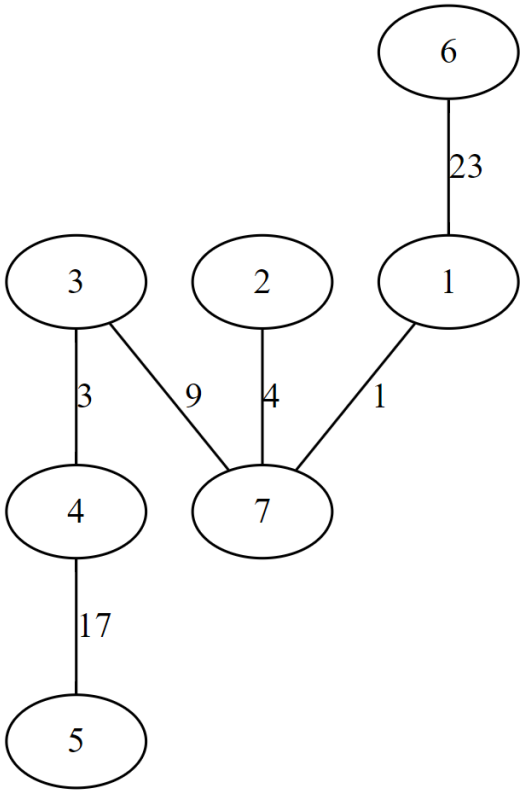


Рис.10 Скриншот построения минимального остовного дерева по полученной DOT-нотации

Выводы

1. В ходе работы была создана программа для работы с графами.
2. Также были реализованы функции добавления, вывода исходного графа и остовного дерева.
3. Были изучены особенности алгоритма Крускала:
4. Преимущества: Алгоритм Крускала позволяет эфективно строить минимальное остовное дерево благодаря своей линейной сложности. Для построения достаточно одного прохода по всем ребрам.
5. Недостатки: Алгоритм Крускала требует сортировки ребер графа по весу по убыванию, что в больших графах может некоторого времени.
6. Таким образом, была изучена работа Алгоритма Крускала и принцип представления графов в памяти компьютера .

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 30.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 30.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 030.11.2020).
5. Описание алгоритма Крускала. интернет-ресурс: https://ru.wikipedia.org/wiki/Алгоритм_Крускала (Дата обращения – 30.11.2020).
6. Построение графов по DOT-нотации. интернет-ресурс: https://dreampuf.github.io/GraphvizOnline (Дата обращения – 30.11.2020).

Практическая работа № 12

НАХОЖДЕНИЕ КРАТЧАЙШИХ ПУТЕЙ В ГРАФЕ

Вариант-12.2.3.2

Постановка задачи

Составить программу нахождения кратчайших путей в графе алгоритмом Йена.

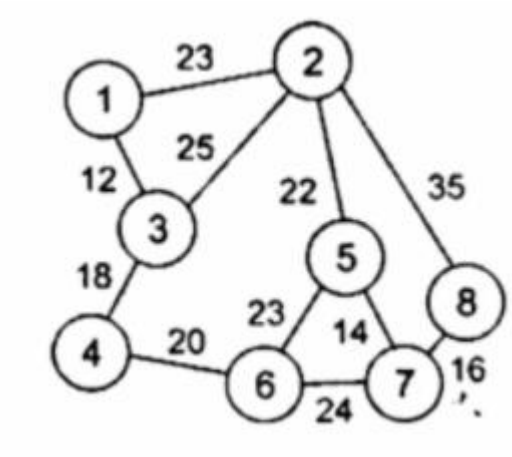
Выбрать и реализовать способ представления графа в памяти.

Предусмотреть ввод с клавиатуры произвольного графа.

Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы для заданного графа.

Индивидуальное задание:



26. Описание алгоритма

Алгоритм Йена вычисляет K -самые короткие пути без петель с одним источником для графа с неотрицательной стоимостью ребра. Алгоритм был опубликован Jin Y. Yen в 1971 году и использует любой алгоритм кратчайшего пути для поиска наилучшего пути, а затем переходит к поиску K - 1 отклонений от наилучшего пути. Алгоритм может быть разбит на две части, определение первого к-кратчайший путь, и затем определяют все другие K - кратчайшие пути. Предполагается, что контейнер будет содержать k- самый короткий путь, тогда как контейнер будет содержать потенциальные k- самый короткий путь. Чтобы определить, в кратчайшем пути от источника к раковине, любой эффективный алгоритм кратчайшего пути может быть использован. Чтобы найти, где находится в диапазоне от до, алгоритм предполагает, что все пути от до были ранее найдены. Итерации могут быть разделены на два процесса, находя все отклонения и выбрать минимальную длину пути, чтобы стать. Обратите внимание, что в этой итерации диапазон от до. Первый процесс можно подразделить на три операции: выбор, поиск и добавление в контейнер. Корневой путь выбирается путем нахождения подпути в том, что следует за первыми узлами, где находится в диапазоне от до. Затем, если найден путь, стоимость края от устанавливается на бесконечность. Затем, ответвительный путь находится путем вычисления кратчайшего пути от ответвительного узла, узла, до приемника. Удаление ранее использовавшихся кромок от до гарантирует, что путь ответвления будет другим. , добавление корневого пути и ответвления добавляется к . Затем ребра, которые были удалены, т.е. для которых

была установлена бесконечная стоимость, восстанавливаются до своих исходных значений. Второй процесс определяет подходящий путь, находя путь в контейнере с наименьшими затратами. Этот путь удаляется из контейнера и вставляется в контейнер, и алгоритм переходит к следующей итерации.

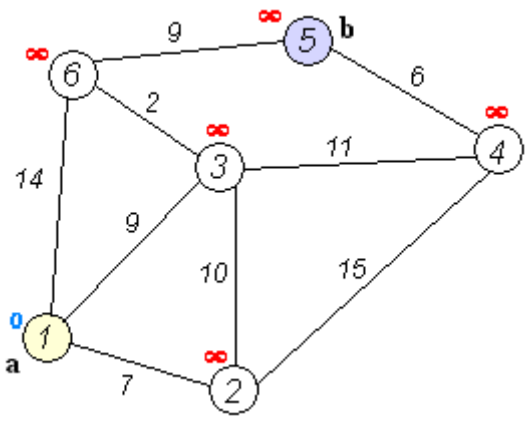


Рис.1 Поиск кратчайшего пути алгоритмом Дейкстры

Функция main вызывает меню.

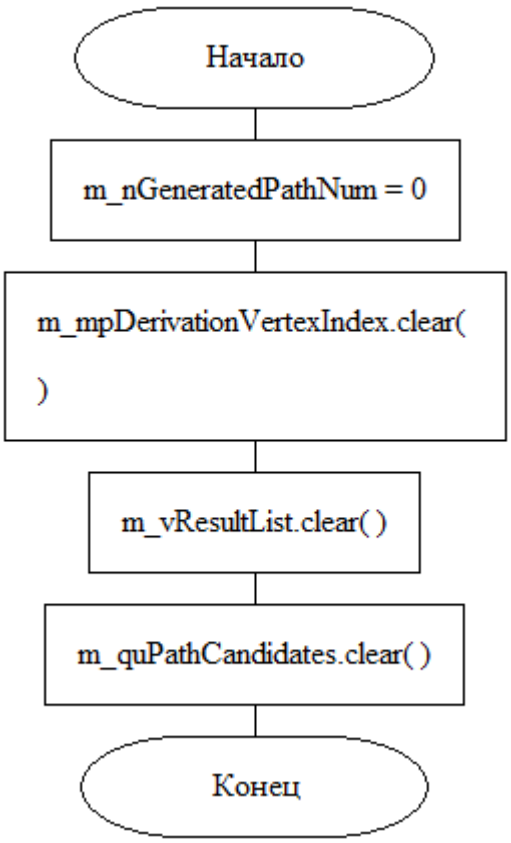


Рис.2 Схема алгоритма функции clear

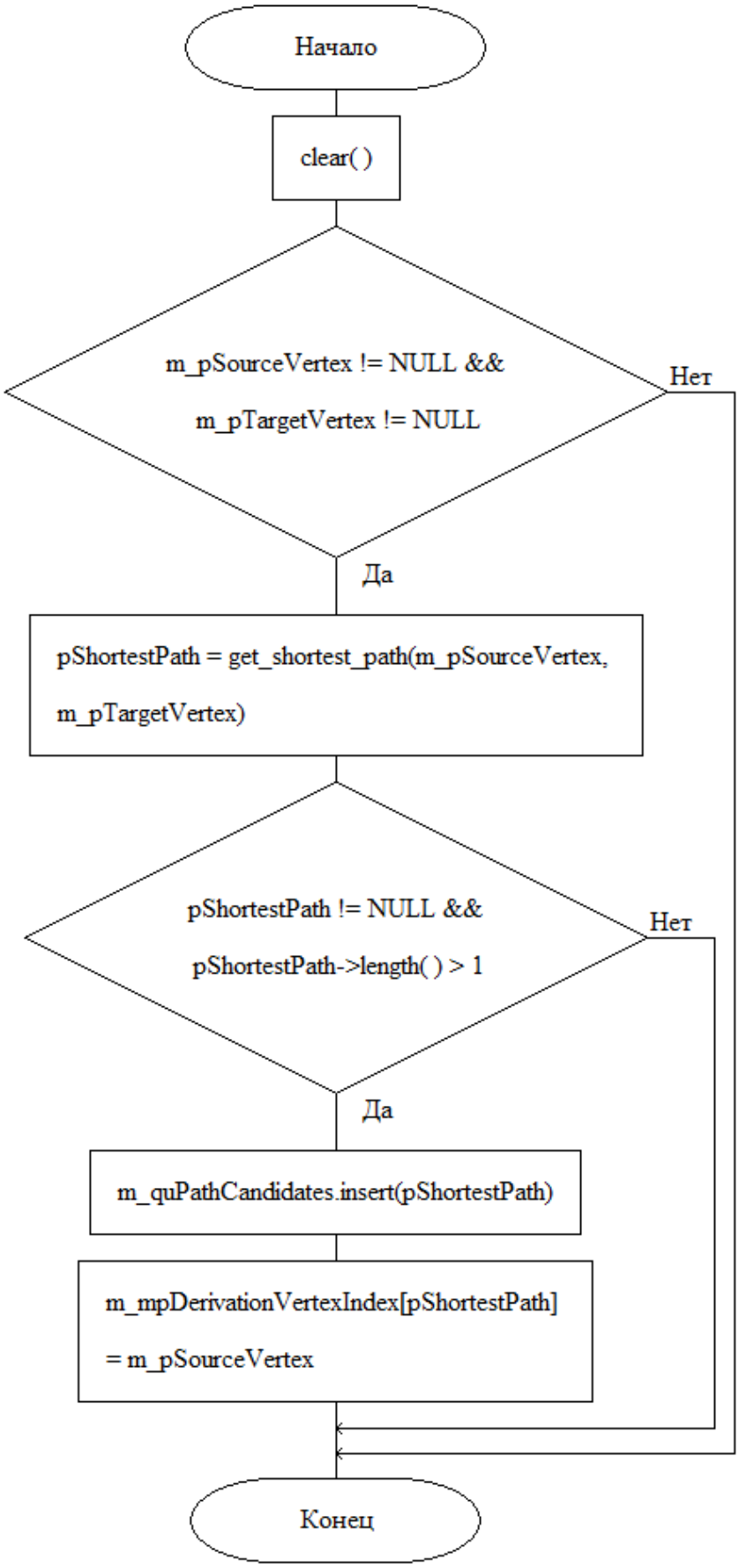

Начало
clear( )
m_pSourceVertex != NULL &&
m_pTargetVertex != NULL
Нет
Да
pShortestPath = get_shortest_path(m_pSourceVertex,
m_pTargetVertex)
pShortestPath != NULL &&
pShortestPath->length( ) > 1
Нет
Да
m_quPathCandidates.insert(pShortestPath)
m_mpDerivationVertexIndex[pShortestPath]
= m_pSourceVertex
Конец

Рис.3 Схема алгоритма функции init

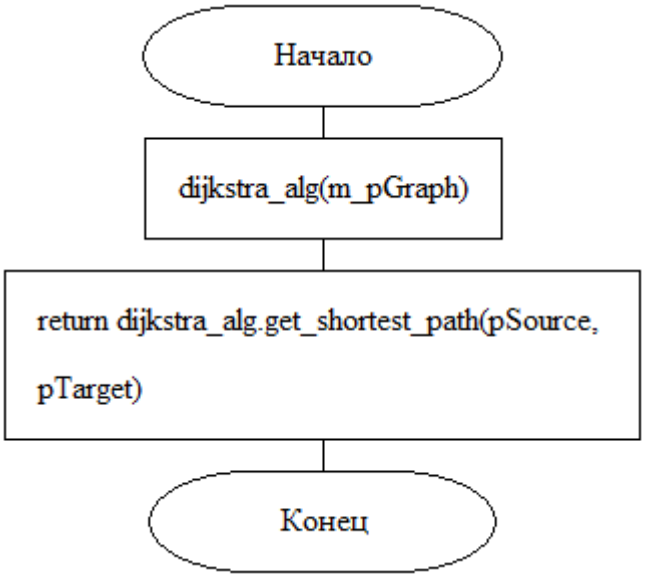


Рис.4 Схема алгоритма функции get_shortest_path

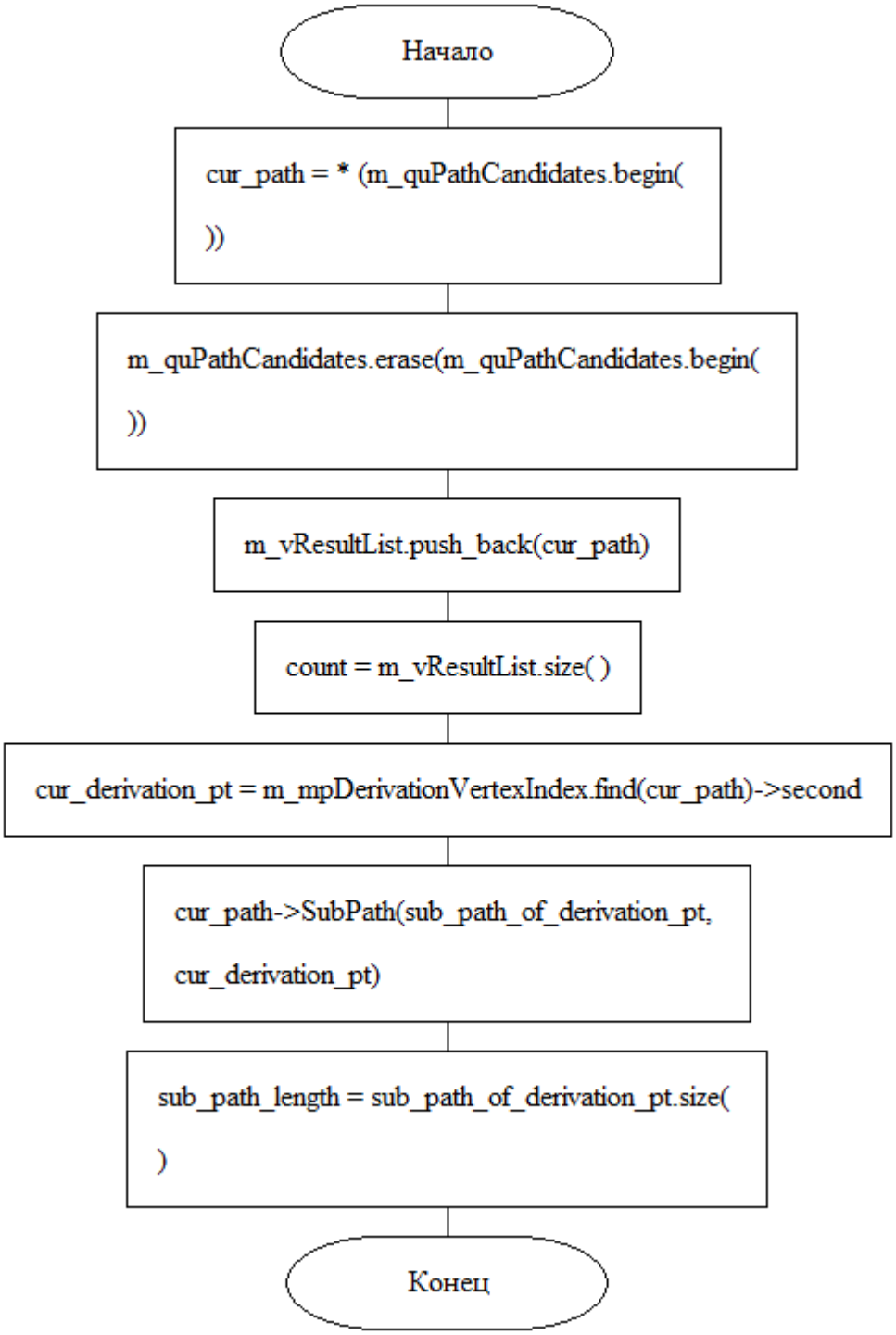


Рис.5 Схема алгоритма функции next

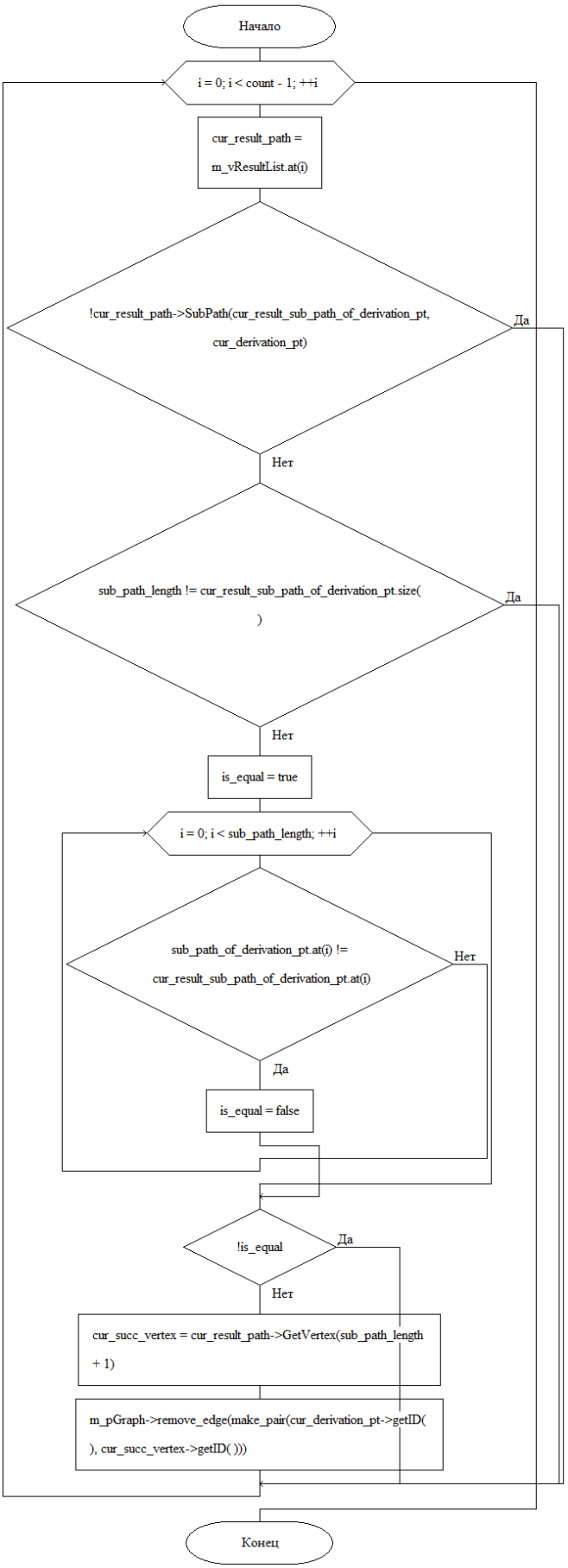

Начало
i = 0; i < count - 1; ++i
cur_result_path = m_vResultList.at(i)
!cur_result_path->SubPath(cur_result_sub_path_of_derivation_pt, cur_derivation_pt)
Да
Нет
sub_path_length != cur_result_sub_path_of_derivation_pt.size( )
Да
Нет
is_equal = true
i = 0; i < sub_path_length; ++i
sub_path_of_derivation_pt.at(i) != cur_result_sub_path_of_derivation_pt.at(i)
Нет
Да
is_equal = false
!is_equal
Да
Нет
cur_succ_vertex = cur_result_path->GetVertex(sub_path_length + 1)
m_pGraph->remove_edge(make_pair(cur_derivation_pt->getID( ), cur_succ_vertex->getID( )))
Конец

Рис.6 Схема алгоритма функции поиска следующего пути

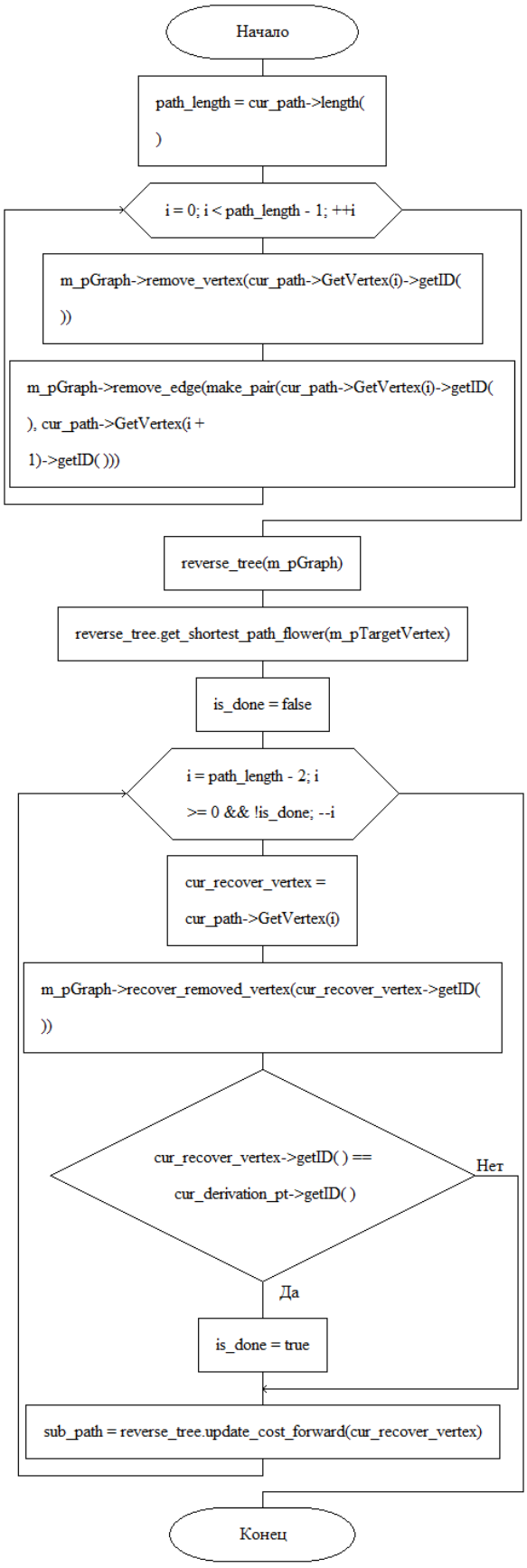


Рис.5 Схема алгоритма функции удаления векторов

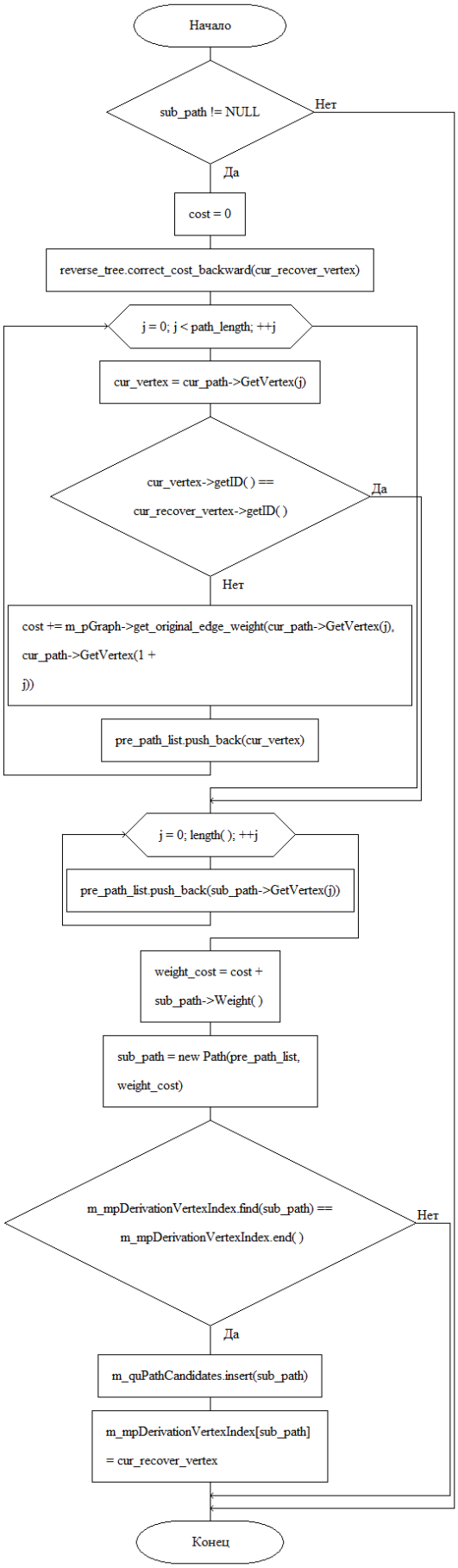

Начало
sub_path != NULL
Нет
Да
cost = 0
reverse_tree.correct_cost_backward(cur_recover_vertex)
j = 0; j < path_length; ++j
cur_vertex = cur_path->GetVertex(j)
cur_vertex->getID( ) ==
cur_recover_vertex->getID( )
Да
Нет
cost += m_pGraph->get_original_edge_weight(cur_path->GetVertex(j),
cur_path->GetVertex(1 +
j))
pre_path_list.push_back(cur_vertex)
j = 0; length( ); ++j
pre_path_list.push_back(sub_path->GetVertex(j))
weight_cost = cost +
sub_path->Weight( )
sub_path = new Path(pre_path_list,
weight_cost)
m_mpDerivationVertexIndex.find(sub_path) ==
m_mpDerivationVertexIndex.end( )
Нет
Да
m_quPathCandidates.insert(sub_path)
m_mpDerivationVertexIndex[sub_path]
= cur_recover_vertex
Конец

Рис.6 Схема алгоритма функции удаления векторов

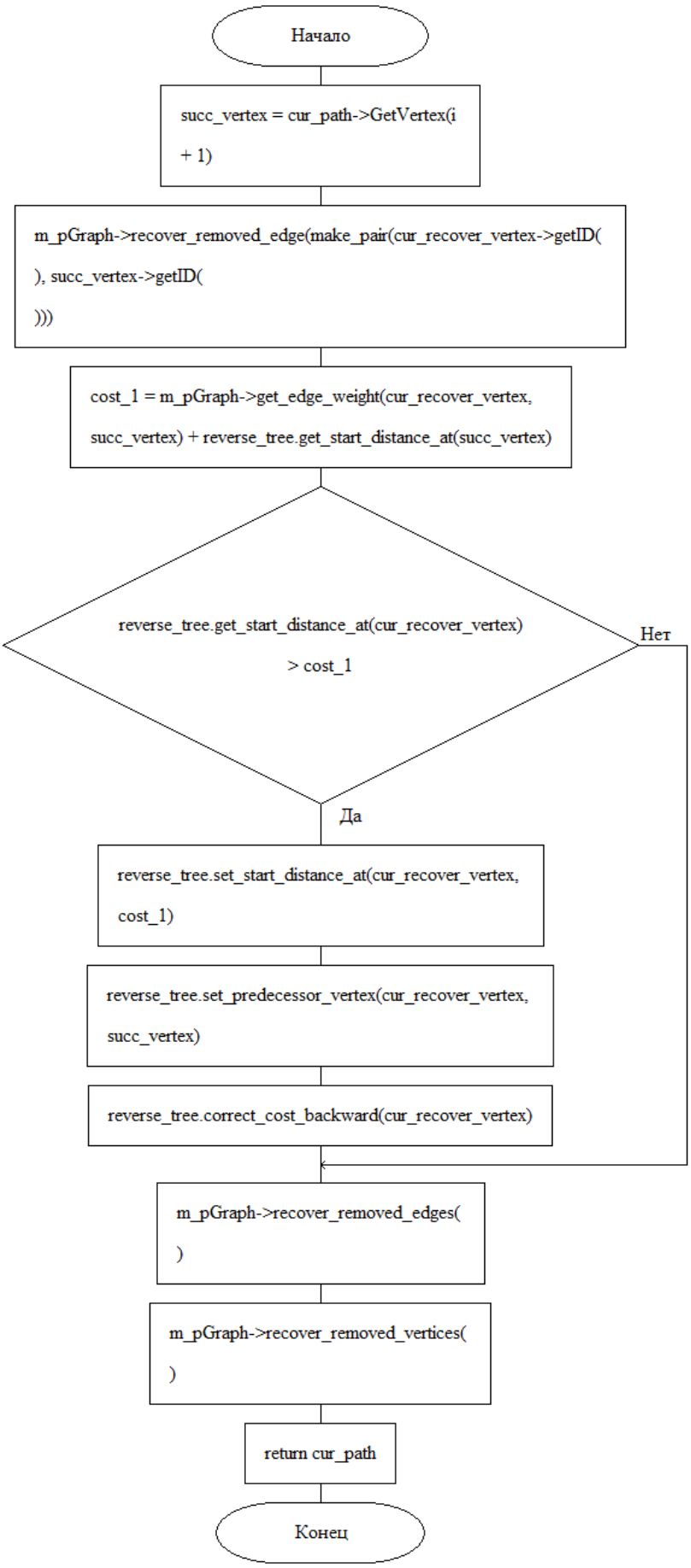

Начало
succ_vertex = cur_path->GetVertex(i + 1)
m_pGraph->recover_removed_edge(make_pair(cur_recover_vertex->getID(), succ_vertex->getID()))
cost_1 = m_pGraph->get_edge_weight(cur_recover_vertex, succ_vertex) + reverse_tree.get_start_distance_at(succ_vertex)
reverse_tree.get_start_distance_at(cur_recover_vertex) > cost_1
Нет
Да
reverse_tree.set_start_distance_at(cur_recover_vertex, cost_1)
reverse_tree.set_predecessor_vertex(cur_recover_vertex, succ_vertex)
reverse_tree.correct_cost_backward(cur_recover_vertex)
m_pGraph->recover_removed_edges()
m_pGraph->recover_removed_vertices()
return cur_path
Конец

Рис.7 Схема алгоритма функции разворота дерева

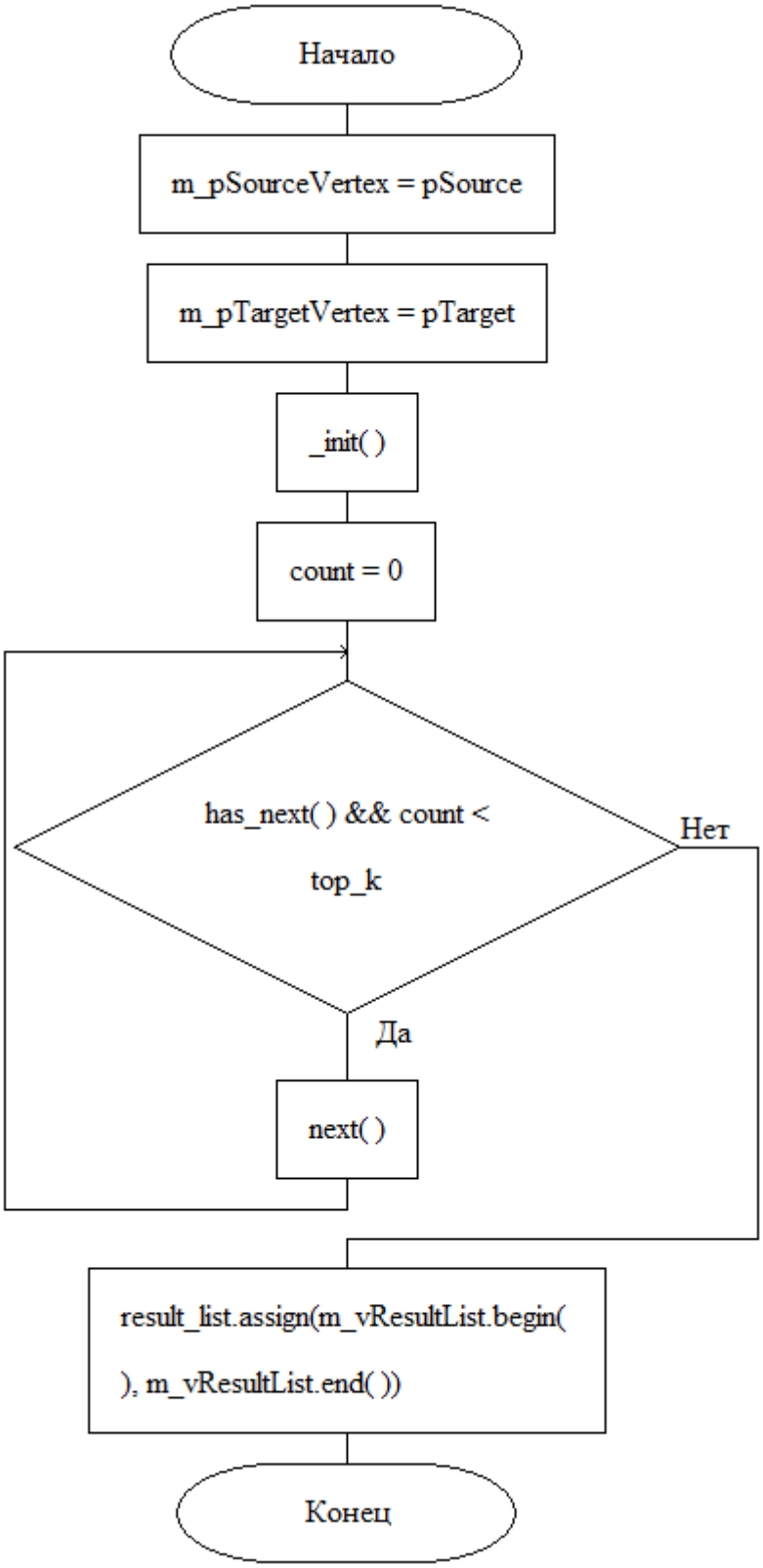


Рис.8 Схема алгоритма функции get_shortest_paths

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include <limits>
#include <set>
#include <map>
#include <queue>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "GraphElements.h"
#include "Graph.h"
#include "Yen.h"

using namespace std;

void testYenAlg()
{
	Graph my_graph("yen_8.txt");

	Yen yenAlg(my_graph, my_graph.get_vertex(0),
		my_graph.get_vertex(7));

	int i = 0;
	while (yenAlg.has_next())
	{
		++i;
		yenAlg.next()->PrintOut(cout);
	}
}

void menu() {
	int choice = 0, count = 0, heights = 0;
	int v1 = 0, v2 = 0, weigth = 0;
	bool flag = true;
	string line = "", str = "";
	while (flag == true) {
		cout << "\n[1] - Добавить элемент в граф\n[2] - Вывод графа в виде таблицы\n[3]
- Вывод K-кратчайших путей во взвешенном графе\n[4] - Выход из программы\n\nВведите номер
команды: ";
		cin >> choice;
		if (choice == 1) {
			std::ofstream f;
			f.open("yen_8.txt", std::ios::app);
```

```
                        cout << endl;
                        cout << "Введите номер начальной вершины: ";
                        cin >> v1;
                        cout << "Введите номер конечной вершины: ";
                        cin >> v2;
                        cout << "Введите вес ребра между вершинами: ";
                        cin >> weigth;
                        f << endl << v1 << " " << v2 << "  " << weigth << "     ";

                f.close();
                cout << endl;
        }
        else if (choice == 2) {
                cout << endl;
                ifstream fin;
                fin.open("yen_8.txt");
                while (!fin.eof()) {
                        if (count == 0) {
                                cout << "Кол-во вершин в графе:";
                                count++;
                        }
                        else if (count == 1) {
                                cout << "_____________________" << endl;
                                cout << "|начало| конец |вес|" << endl;
                                count++;
                        }
                        getline(fin, str);
                        cout << "|" << str << " |" << endl;
                }
                cout << "---------------------" << endl;
                fin.close();
                cout << endl;
        }
        else if (choice == 3) {
                testYenAlg();
                cout << endl;
        }
        else if (choice == 4) {
                cout << endl;
                cout << "Подтвердите выход\n1 - Да | 2 - Нет\nВведите выбранную команду:
";
                cin >> choice;
                if (choice == 1)
                {
                        cout << endl;
                        flag = false;
                }
                else if (choice == 2)
                {
                        flag = true;
                        cout << endl;
                }
        }
    }
}

int main()
{
    setlocale(0, "");

    menu();
}
```

Graph.h

```
#pragma once

using namespace std;

class Path : public BasePath
{
public:

	Path(const	std::vector<BaseVertex*>&	vertex_list,	double	weight)
:BasePath(vertex_list, weight) {}


	void PrintOut(std::ostream& out_stream) const
	{
		out_stream << "w:" << m_dWeight << "d:" << m_vtVertexList.size() << std::endl;
		for (std::vector<BaseVertex*>::const_iterator pos = m_vtVertexList.begin(); pos
!= m_vtVertexList.end(); ++pos)
		{
			out_stream << (*pos)->getID() << " ";
		}
		out_stream		<<		std::endl		<<
"_____" << std::endl;
	}
};

class Graph
{
public:

	const static double DISCONNECT;

	typedef set<BaseVertex*>::iterator VertexPtSetIterator;
	typedef map<BaseVertex*, set<BaseVertex*>*>::iterator BaseVertexPt2SetMapIterator;

protected:


	map<BaseVertex*, set<BaseVertex*>*> m_mpFanoutVertices;
	map<BaseVertex*, set<BaseVertex*>*> m_mpFaninVertices;
	map<int, double> m_mpEdgeCodeWeight;
	vector<BaseVertex*> m_vtVertices;
	int m_nEdgeNum;
	int m_nVertexNum;

	map<int, BaseVertex*> m_mpVertexIndex;


	set<int> m_stRemovedVertexIds;
	set<pair<int, int> > m_stRemovedEdge;

public:


	Graph(const string& file_name);
	Graph(const Graph& rGraph);
	~Graph(void);

	void clear();

	BaseVertex* get_vertex(int node_id);
```

```
	int get_edge_code(const BaseVertex* start_vertex_pt, const BaseVertex* end_vertex_pt)
const;
	set<BaseVertex*>*	get_vertex_set_pt(BaseVertex*	vertex_,	map<BaseVertex*,
set<BaseVertex*>*>& vertex_container_index);

	double get_original_edge_weight(const BaseVertex* source, const BaseVertex* sink);

	double get_edge_weight(const BaseVertex* source, const BaseVertex* sink);
	void get_adjacent_vertices(BaseVertex* vertex, set<BaseVertex*>& vertex_set);
	void get_precedent_vertices(BaseVertex* vertex, set<BaseVertex*>& vertex_set);


	void remove_edge(const pair<int, int> edge)
	{
		m_stRemovedEdge.insert(edge);
	}

	void remove_vertex(const int vertex_id)
	{
		m_stRemovedVertexIds.insert(vertex_id);
	}

	void recover_removed_edges()
	{
		m_stRemovedEdge.clear();
	}

	void recover_removed_vertices()
	{
		m_stRemovedVertexIds.clear();
	}

	void recover_removed_edge(const pair<int, int> edge)
	{
		m_stRemovedEdge.erase(m_stRemovedEdge.find(edge));
	}

	void recover_removed_vertex(int vertex_id)
	{
		m_stRemovedVertexIds.erase(m_stRemovedVertexIds.find(vertex_id));
	}

private:
	void _import_from_file(const std::string& file_name);

};
```

Graph.cpp

```
#include <limits>
#include <set>
#include <map>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>
#include "GraphElements.h"
#include "Graph.h"
```

```
const double Graph::DISCONNECT = (numeric_limits<double>::max)();


Graph::Graph(const string& file_name)
{
	_import_from_file(file_name);
}

Graph::Graph(const Graph& graph)
{
	m_nVertexNum = graph.m_nVertexNum;
	m_nEdgeNum = graph.m_nEdgeNum;
	m_vtVertices.assign(graph.m_vtVertices.begin(), graph.m_vtVertices.end());
	m_mpFaninVertices.insert(graph.m_mpFaninVertices.begin(),
graph.m_mpFaninVertices.end());
	m_mpFanoutVertices.insert(graph.m_mpFanoutVertices.begin(),
graph.m_mpFanoutVertices.end());
	m_mpEdgeCodeWeight.insert(graph.m_mpEdgeCodeWeight.begin(),
graph.m_mpEdgeCodeWeight.end());
	m_mpVertexIndex.insert(graph.m_mpVertexIndex.begin(), graph.m_mpVertexIndex.end());
}

Graph::~Graph(void)
{
	clear();
}


void Graph::_import_from_file(const string& input_file_name)
{
	const char* file_name = input_file_name.c_str();


	ifstream ifs(file_name);
	if (!ifs)
	{
		cerr << "Graph.cpp1 " << file_name << " Graph.cpp1!!!" << endl;
		exit(1);
	}

	clear();


	ifs >> m_nVertexNum;

	int start_vertex, end_vertex;
	double edge_weight;
	int vertex_id = 0;

	while (ifs >> start_vertex)
	{
		if (start_vertex == -1)
		{
			break;
		}
		ifs >> end_vertex;
		ifs >> edge_weight;


		BaseVertex* start_vertex_pt = get_vertex(start_vertex);
		BaseVertex* end_vertex_pt = get_vertex(end_vertex);
```

```
                m_mpEdgeCodeWeight[get_edge_code(start_vertex_pt,       end_vertex_pt)]       =
edge_weight;


                get_vertex_set_pt(end_vertex_pt, m_mpFaninVertices)->insert(start_vertex_pt);


                get_vertex_set_pt(start_vertex_pt, m_mpFanoutVertices)->insert(end_vertex_pt);

        }

        if (m_nVertexNum != m_vtVertices.size())
        {
                cerr << "Graph.cpp: " << m_vtVertices.size() << "Graph.cpp:" << m_nVertexNum <<
endl;
                exit(1);
        }

        m_nVertexNum = m_vtVertices.size();
        m_nEdgeNum = m_mpEdgeCodeWeight.size();

        ifs.close();
}

BaseVertex* Graph::get_vertex(int node_id)
{
        if (m_stRemovedVertexIds.find(node_id) != m_stRemovedVertexIds.end())
        {
                return NULL;
        }
        else
        {
                BaseVertex* vertex_pt = NULL;
                const map<int, BaseVertex*>::iterator pos = m_mpVertexIndex.find(node_id);
                if (pos == m_mpVertexIndex.end())
                {
                        int vertex_id = m_vtVertices.size();
                        vertex_pt = new BaseVertex();
                        vertex_pt->setID(node_id);
                        m_mpVertexIndex[node_id] = vertex_pt;

                        m_vtVertices.push_back(vertex_pt);
                }
                else
                {
                        vertex_pt = pos->second;
                }

                return vertex_pt;
        }
}

void Graph::clear()
{
        m_nEdgeNum = 0;
        m_nVertexNum = 0;

        for       (map<BaseVertex*,       set<BaseVertex*>*>::const_iterator       pos       =
m_mpFaninVertices.begin();
                pos != m_mpFaninVertices.end(); ++pos)
        {
                delete pos->second;
        }
```

```
	m_mpFaninVertices.clear();

	for	(map<BaseVertex*,	set<BaseVertex*>*>::const_iterator	pos	=
m_mpFanoutVertices.begin();
		pos != m_mpFanoutVertices.end(); ++pos)
	{
		delete pos->second;
	}
	m_mpFanoutVertices.clear();


	m_mpEdgeCodeWeight.clear();


	for_each(m_vtVertices.begin(), m_vtVertices.end(), DeleteFunc<BaseVertex>());
	m_vtVertices.clear();
	m_mpVertexIndex.clear();

	m_stRemovedVertexIds.clear();
	m_stRemovedEdge.clear();
}

int Graph::get_edge_code(const BaseVertex* start_vertex_pt, const BaseVertex* end_vertex_pt)
const
{

	return start_vertex_pt->getID() * m_nVertexNum + end_vertex_pt->getID();
}


set<BaseVertex*>*	Graph::get_vertex_set_pt(BaseVertex*	vertex_,	map<BaseVertex*,
set<BaseVertex*>*>& vertex_container_index)
{
	BaseVertexPt2SetMapIterator pos = vertex_container_index.find(vertex_);

	if (pos == vertex_container_index.end())
	{
		set<BaseVertex*>* vertex_set = new set<BaseVertex*>();
		pair<BaseVertexPt2SetMapIterator, bool> ins_pos =
			vertex_container_index.insert(make_pair(vertex_, vertex_set));

		pos = ins_pos.first;
	}

	return pos->second;
}


double Graph::get_edge_weight(const BaseVertex* source, const BaseVertex* sink)
{
	int source_id = source->getID();
	int sink_id = sink->getID();

	if (m_stRemovedVertexIds.find(source_id) != m_stRemovedVertexIds.end()
		|| m_stRemovedVertexIds.find(sink_id) != m_stRemovedVertexIds.end()
		||	m_stRemovedEdge.find(make_pair(source_id,	sink_id))	!=
m_stRemovedEdge.end())
	{
		return DISCONNECT;
	}
	else
	{
		return get_original_edge_weight(source, sink);
```

```
	}
}


void Graph::get_adjacent_vertices(BaseVertex* vertex, set<BaseVertex*>& vertex_set)
{
	int starting_vt_id = vertex->getID();

	if (m_stRemovedVertexIds.find(starting_vt_id) == m_stRemovedVertexIds.end())
	{
		set<BaseVertex*>*	vertex_pt_set	=	get_vertex_set_pt(vertex,
m_mpFanoutVertices);
		for (set<BaseVertex*>::const_iterator pos = (*vertex_pt_set).begin();
			pos != (*vertex_pt_set).end(); ++pos)
		{
			int ending_vt_id = (*pos)->getID();
			if		(m_stRemovedVertexIds.find(ending_vt_id)		!=
m_stRemovedVertexIds.end()
				||	m_stRemovedEdge.find(make_pair(starting_vt_id,	ending_vt_id))
!= m_stRemovedEdge.end())
			{
				continue;
			}

			vertex_set.insert(*pos);
		}
	}
}

void Graph::get_precedent_vertices(BaseVertex* vertex, set<BaseVertex*>& vertex_set)
{
	if (m_stRemovedVertexIds.find(vertex->getID()) == m_stRemovedVertexIds.end())
	{
		int ending_vt_id = vertex->getID();
		set<BaseVertex*>*	pre_vertex_set	=	get_vertex_set_pt(vertex,
m_mpFaninVertices);
		for (set<BaseVertex*>::const_iterator pos = (*pre_vertex_set).begin();
			pos != (*pre_vertex_set).end(); ++pos)
		{
			int starting_vt_id = (*pos)->getID();
			if		(m_stRemovedVertexIds.find(starting_vt_id)		!=
m_stRemovedVertexIds.end()
				||	m_stRemovedEdge.find(make_pair(starting_vt_id,	ending_vt_id))
!= m_stRemovedEdge.end())
			{
				continue;
			}

			vertex_set.insert(*pos);
		}
	}
}

double Graph::get_original_edge_weight(const BaseVertex* source, const BaseVertex* sink)
{
	map<int, double>::const_iterator pos =
		m_mpEdgeCodeWeight.find(get_edge_code(source, sink));

	if (pos != m_mpEdgeCodeWeight.end())
	{
		return pos->second;
	}
	else
```

```
	{
		return DISCONNECT;
	}
}
```

Yen.h

```
#pragma once

using namespace std;

class Yen
{
	Graph* m_pGraph;

	vector<BasePath*> m_vResultList;
	map<BasePath*, BaseVertex*> m_mpDerivationVertexIndex;
	multiset<BasePath*, WeightLess<BasePath> > m_quPathCandidates;

	BaseVertex* m_pSourceVertex;
	BaseVertex* m_pTargetVertex;

	int m_nGeneratedPathNum;

private:

	void _init();

public:

	Yen(const Graph& graph)
	{
		Yen(graph, NULL, NULL);
	}

	Yen(const Graph& graph, BaseVertex* pSource, BaseVertex* pTarget)
		:m_pSourceVertex(pSource), m_pTargetVertex(pTarget)
	{
		m_pGraph = new Graph(graph);
		_init();
	}

	~Yen(void) { clear(); }

	void clear();
	bool has_next();
	BasePath* next();

	BasePath* get_shortest_path(BaseVertex* pSource, BaseVertex* pTarget);
	void get_shortest_paths(BaseVertex* pSource, BaseVertex* pTarget, int top_k,
		vector<BasePath*>&);
	};
```

Yen.cpp

```
#include <set>
#include <map>
```

```
#include <queue>
#include <vector>
#include "GraphElements.h"
#include "Graph.h"
#include "Dx.h"
#include "Yen.h"

using namespace std;

void Yen::clear()
{
	m_nGeneratedPathNum = 0;
	m_mpDerivationVertexIndex.clear();
	m_vResultList.clear();
	m_quPathCandidates.clear();
}

void Yen::_init()
{
	clear();
	if (m_pSourceVertex != NULL && m_pTargetVertex != NULL)
	{
		BasePath* pShortestPath = get_shortest_path(m_pSourceVertex, m_pTargetVertex);
		if (pShortestPath != NULL && pShortestPath->length() > 1)
		{
			m_quPathCandidates.insert(pShortestPath);
			m_mpDerivationVertexIndex[pShortestPath] = m_pSourceVertex;
		}
	}
}

BasePath* Yen::get_shortest_path(BaseVertex* pSource, BaseVertex* pTarget)
{
	Dx dijkstra_alg(m_pGraph);
	return dijkstra_alg.get_shortest_path(pSource, pTarget);
}

bool Yen::has_next()
{
	return !m_quPathCandidates.empty();
}

BasePath* Yen::next()
{

	BasePath* cur_path = *(m_quPathCandidates.begin());


	m_quPathCandidates.erase(m_quPathCandidates.begin());
	m_vResultList.push_back(cur_path);

	int count = m_vResultList.size();

	BaseVertex* cur_derivation_pt = m_mpDerivationVertexIndex.find(cur_path)->second;
	vector<BaseVertex*> sub_path_of_derivation_pt;
	cur_path->SubPath(sub_path_of_derivation_pt, cur_derivation_pt);
	int sub_path_length = sub_path_of_derivation_pt.size();


	for (int i = 0; i < count - 1; ++i)
	{
		BasePath* cur_result_path = m_vResultList.at(i);
		vector<BaseVertex*> cur_result_sub_path_of_derivation_pt;
```

```
                if            (!cur_result_path->SubPath(cur_result_sub_path_of_derivation_pt,
cur_derivation_pt)) continue;

                if (sub_path_length != cur_result_sub_path_of_derivation_pt.size()) continue;

                bool is_equal = true;
                for (int i = 0; i < sub_path_length; ++i)
                {
                        if                  (sub_path_of_derivation_pt.at(i)                  !=
cur_result_sub_path_of_derivation_pt.at(i))
                        {
                                is_equal = false;
                                break;
                        }
                }
                if (!is_equal) continue;


                BaseVertex* cur_succ_vertex = cur_result_path->GetVertex(sub_path_length + 1);
                m_pGraph->remove_edge(make_pair(cur_derivation_pt->getID(),    cur_succ_vertex-
>getID()));
        }


        int path_length = cur_path->length();
        for (int i = 0; i < path_length - 1; ++i)
        {
                m_pGraph->remove_vertex(cur_path->GetVertex(i)->getID());
                m_pGraph->remove_edge(make_pair(
                        cur_path->GetVertex(i)->getID(), cur_path->GetVertex(i + 1)->getID()));
        }


        Dx reverse_tree(m_pGraph);
        reverse_tree.get_shortest_path_flower(m_pTargetVertex);


        bool is_done = false;
        for (int i = path_length - 2; i >= 0 && !is_done; --i)
        {

                BaseVertex* cur_recover_vertex = cur_path->GetVertex(i);
                m_pGraph->recover_removed_vertex(cur_recover_vertex->getID());


                if (cur_recover_vertex->getID() == cur_derivation_pt->getID())
                {
                        is_done = true;
                }


                BasePath* sub_path = reverse_tree.update_cost_forward(cur_recover_vertex);


                if (sub_path != NULL)
                {
                        ++m_nGeneratedPathNum;

                        double cost = 0;
                        reverse_tree.correct_cost_backward(cur_recover_vertex);

                        vector<BaseVertex*> pre_path_list;
```

```
                        for (int j = 0; j < path_length; ++j)
                        {
                                BaseVertex* cur_vertex = cur_path->GetVertex(j);
                                if (cur_vertex->getID() == cur_recover_vertex->getID())
                                {

                                        break;
                                }
                                else
                                {
                                        cost += m_pGraph->get_original_edge_weight(
                                                cur_path->GetVertex(j),   cur_path->GetVertex(1   +
j));

                                        pre_path_list.push_back(cur_vertex);
                                }
                        }

                        for (int j = 0; j < sub_path->length(); ++j)
                        {
                                pre_path_list.push_back(sub_path->GetVertex(j));
                        }


                        double weight_cost = cost + sub_path->Weight();
                        delete sub_path;
                        sub_path = new Path(pre_path_list, weight_cost);


                        if              (m_mpDerivationVertexIndex.find(sub_path)              ==
m_mpDerivationVertexIndex.end())
                        {
                                m_quPathCandidates.insert(sub_path);
                                m_mpDerivationVertexIndex[sub_path] = cur_recover_vertex;
                        }
                }


                BaseVertex* succ_vertex = cur_path->GetVertex(i + 1);
                m_pGraph->recover_removed_edge(make_pair(cur_recover_vertex->getID(),
succ_vertex->getID()));


                double cost_1 = m_pGraph->get_edge_weight(cur_recover_vertex, succ_vertex)
                        + reverse_tree.get_start_distance_at(succ_vertex);

                if (reverse_tree.get_start_distance_at(cur_recover_vertex) > cost_1)
                {
                        reverse_tree.set_start_distance_at(cur_recover_vertex, cost_1);
                        reverse_tree.set_predecessor_vertex(cur_recover_vertex, succ_vertex);
                        reverse_tree.correct_cost_backward(cur_recover_vertex);
                }
        }


        m_pGraph->recover_removed_edges();
        m_pGraph->recover_removed_vertices();

        return cur_path;
}

void Yen::get_shortest_paths(BaseVertex* pSource,
        BaseVertex* pTarget, int top_k, vector<BasePath*>& result_list)
{
```

```
m_pSourceVertex = pSource;
m_pTargetVertex = pTarget;

_init();
int count = 0;
while (has_next() && count < top_k)
{
	next();
	++count;
}

result_list.assign(m_vResultList.begin(), m_vResultList.end());
}
```

27.Тестирование программы

```
[1] - Добавить элемент в граф
[2] - Вывод графа в виде таблицы
[3] - Вывод K-кратчайших путей во взвешенном графе
[4] - Выход из программы

Введите номер команды: 2

Кол-во вершин в графе:|8 |
_____________________
|начало| конец |вес|
|0      1       23 |
|0      2       12 |
|1      2       24 |
|1      4       22 |
|1      7       35 |
|2      3       18 |
|3      4       20 |
|4      5       23 |
|4      6       14 |
|5      6       24 |
|6      7       16 |
--------------------
```

Рис.9 Скриншот вывода графа в виде таблицы

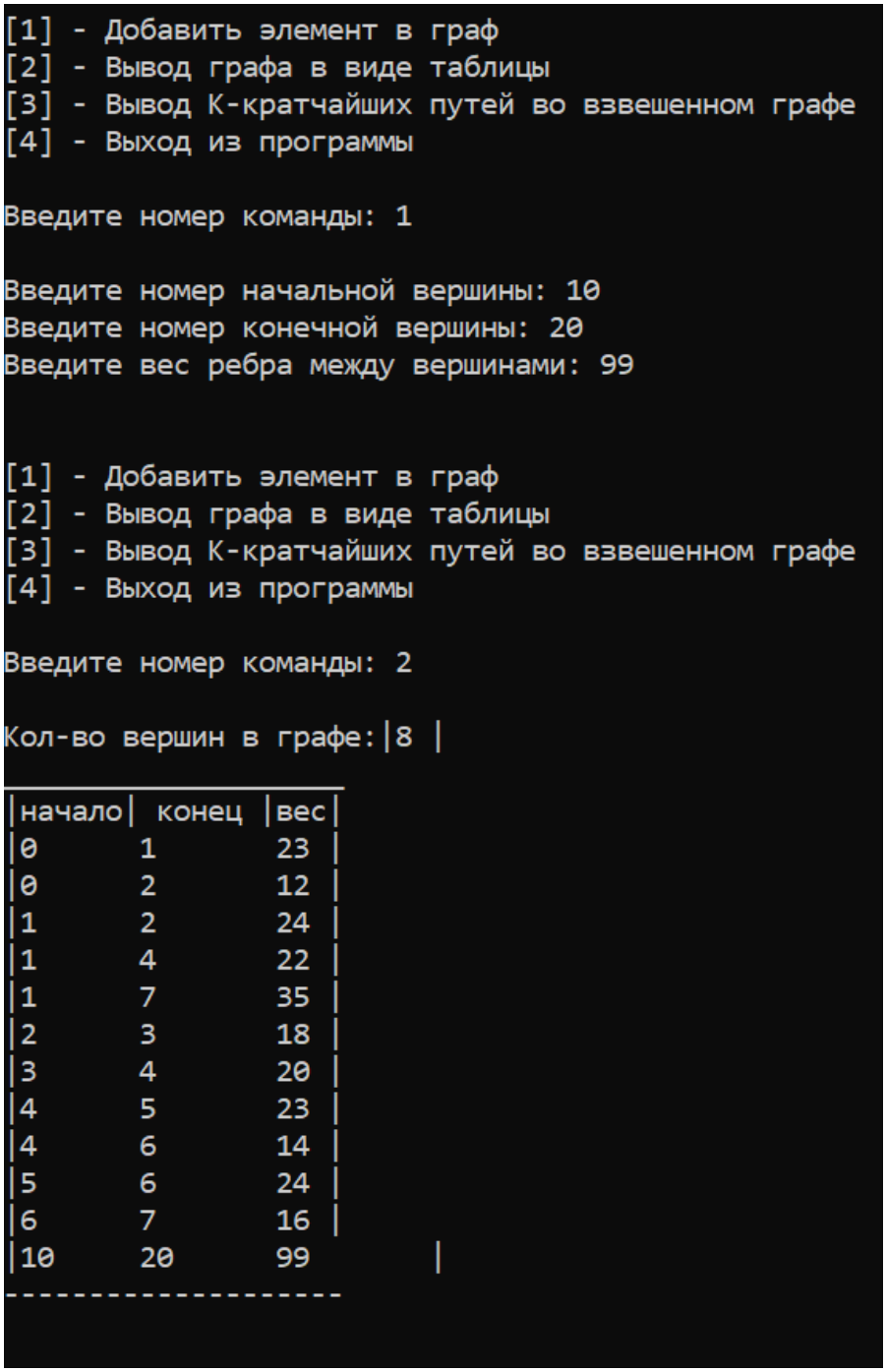


Рис.9 Скриншот добавления связи в граф

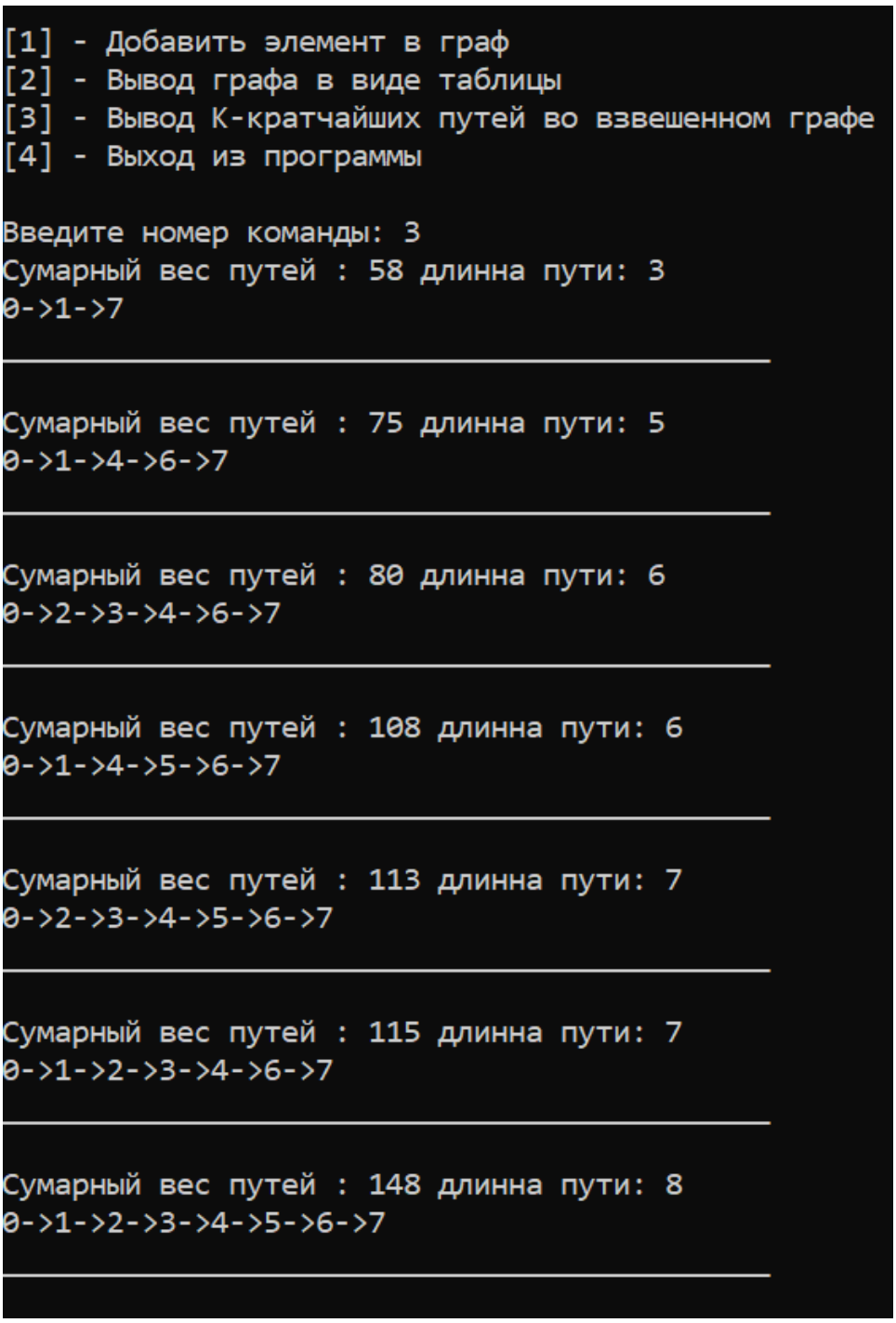


Рис.10 Скриншот вывода k- кратчайших путей во взвешенном графе

Выводы

1. В ходе работы была создана программа для работы с графами.
2. Также были реализованы функции добавления, вывода исходного графа, вывода k-кратчайших путей во взвешенном графе.
3. Были изучены особенности алгоритма Йена и алгоритма Дейкстры:
4. Преимущества: Алгоритм Йена позволяет найти наиболее оптимальные пути не перебирая все возможные.
5. Недостатки: Сложность времени алгоритм Йена напрямую зависит от сложности алгоритма кратчайшего пути. В данном случае это алгоритм Дейкстры, который в худшем случае имеет сложность O(n^2)
6. Таким образом, была изучена работа Алгоритма Йена и принцип представления графов в памяти компьютера .

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 10.12.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 10.12.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 10.12.2020).
5. Описание алгоритма Йена. интернет-ресурс: https://ru.qaz.wiki/wiki/Yen%27s_algorithm (Дата обращения – 10.12.2020).
6. Описание алгоритма Дейкстры. интернет-ресурс: https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры (Дата обращения – 10.12.2020).
7. Описание оформления таблицы в консоли C++. интернет-ресурс: https://ru.stackoverflow.com/questions/437788 (Дата обращения – 10.12.2020).

Заключение

В данном курсе были изучены основные структуры и алгоритмы обработки данных: линейный односвязный список, очередь, стек, линейный двусвязный список, дек, бинарное дерево поиска, АВЛ - дерево, текстовые и двоичные файлы, хеширование данных, сжатие данных, построение остовного дерева графа и нахождение кратчайшего пути в графе, выявлены их сильные и слабые стороны, применения и возможности. Так как небыло четких рамок решения поставленных задач, можно было применить свое воображение, благодаря которому созданно именно свое видение програм. Было проведено множество сравнений различных реализаций, изучены тонкости той или иной из них. Такой подход хорошо подготавливает к реальным рабочим условиям, где никто не будет разжевывать задачу. Всё это дало бесценный опыт, который пригодится в дальнейшем. После второго семестра обучения структурам и алгоритмам данных не остается сомненпий в важности данного предмета для программиста, но нам пора двигаться дальше и достигать новые вершины благодаря полученным знаниям.