МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
тема «Работа со списками. Вариант 7»

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Пестерев Д.О.

Санкт-Петербург
2023

1. Постановка задачи

Реализовать объект в виде двухсвязного списка с набором методов/функций:

1. Добавление в конец списка.
2. Добавление в начало списка.
3. Удаление последнего элемента.
4. Удаление первого элемента.
5. Добавление элемента по индексу.
6. Получение элемента по индексу.
7. Удаление элемента по индексу.
8. Получение размера списка.
9. Удаление всех элементов списка.
10. Замена элемента по индексу на передаваемый элемент.
11. Проверка на пустоту списка.
12. Меняет порядок элементов в списке на обратный.
13. Вставка другого списка в список, начиная с индекса.
14. Вставка другого списка в конец.
15. Вставка другого списка в начало.
16. Проверка на содержание другого списка в списке.
17. Поиск первого вхождения другого списка в список.
18. Поиск последнего вхождения другого списка в список.
19. Обмен двух элементов списка по индексам.

2. Описание алгоритма

Программа «Реализация двусвязного списка» написана на языке программирования C++ при помощи компилятора Visual Studio 2019.

Программа реализует двусвязный список с наиболее используемыми и актуальными методами/функциями.

Также в программе предусматривается создание второго двусвязного списка, в который можно добавлять элементы в конец. Он нужен лишь для реализации нескольких методов/функций основного списка (вставка одного списка в другой, поиск первого и последнего вхождения другого списка в список)

Каждая функция/метод списка в программе реализована с помощью собственной функции, в которую передаются определенные параметры (информация о списке).

Список реализуется с помощью структуры Node – отдельные блоки, из которых составляется список. Блок состоит из значения, ссылки на след элемент, ссылки на предыдущий элемент. Изначально список имеет 2 элемента (head tail), которые не хранят в себе никакого значения, не выводятся на экран, служат лишь облегчением логики составления списка.

В основной ветке программы (main) происходит:

Заполнение списка, который будет служить вставным для основного. При вводе 0, заполнение заканчивается.

Начало заполнения основного списка, ввод первого элемента.

Вывод всех функций/методов списка, которые возможно выполнить в данный момент. Некоторые функции изначально не показываются. К примеру, нахождение первого и последнего вхождения списка в список (так как в список еще не вставлен другой список)

Выбор метода/функции пользователем через консоль (при выборе несуществующей функции, ничего не выполнится, список функций заново выведется на консоль)

Выполнение функции/метода, которую выбрал пользователь (реализуется с помощью конструкции switch-case)

При вводе с консоли 0 программа завершается. (0 присутствует в списке функций)

Описание функций/методов:

Идентификатор	Описание	Вр-нная сложность алгоритма
void printList(Node* start, int& length)	Вывод списка на экран Передаваемые параметры: начало списка, длина списка	$\Theta(n)$ $O(n)$
void addLast(Node* end, int& length, int toAdd)	Добавление элемента в конец списка. Передаваемые параметры: конец списка, длина списка, добавляемое значение	$\Theta(1)$ $O(1)$
void addFirst(Node* start, int& length, int toAdd)	Добавление элемента в начало списка. Передаваемые параметры: начало списка, длина списка, добавляемый элемент	$\Theta(1)$ $O(1)$
void deleteLast(Node* end, int& length)	Удаление последнего элемента списка Передаваемые параметры: конец списка, длина списка	$\Theta(1)$ $O(1)$
void deleteFirst (Node* start, int& length)	Удаление первого элемента списка. Передаваемые параметры: начало списка, длина списка	$\Theta(n)$ $O(1)$
void addElem(Node* start, Node* end, int& length, int num)	Добавление элемента по индексу. Алгоритм: если инд=0, то вызов функции addFirst. Иначе, в зависимости от индекса, выбирается обход списка. Если индекс в первой половине, идем с начала, иначе с конца. Далее, список выводится на экран. Если индекс вышел за пределы списка выводится ошибка. Примечание. Элемент добавляется слева от элемента с введенным индексом, поэтому, идя с начала, можно дойти до того элемента, где добавление будет происходить справа (таким образом уменьшается кол-во действий) Передаваемые параметры: начало списка, конец списка, длина списка, индекс.	$\Theta(n/2)$ $O(n)$
Node* findElem(Node* start, Node* end, int& length, int num)	Нахождение элемента по индексу. Алгоритм: в зависимости от индекса, выбирается обход списка. Если индекс в первой половине, идем с начала, иначе с конца. Если индекс вышел за границы списка выводится ошибка. Возвращаемое значение: адрес на элемент Передаваемые параметры: начало списка, конец списка, длина списка, индекс	$\Theta(n/2)$ $O(n)$

void delElem(Node* start, Node* end, int& length, int num)	Удаление элемента по индексу. Находим элемент по индексу и удаляем его. Передаваемые параметры: начало списка, конец списка, длина списка, индекс.	$\Theta(n/2)$ $O(n)$
void delArr(Node* start, int& length)	Удаление всех элементов списка. Алгоритм: цикл, пока длина !=0, вызывается функция deleteFirst().	$O(n)$
void newName (Node* start, Node* end, int& length, int num)	Изменение значения элемента по индексу. Находим элемент по индексу и меняем его значение. Передаваемые параметры: начало списка, конец списка, длина списка, индекс.	$\Theta(n/2)$ $O(n)$
void reverse(Node*& start, Node*& end, int& length)	Порядок элементов списка становится обратным. Алгоритм: проходим циклом по списку и меняем адреса каждого элемента на след и прошлый эл-нты местами Отдельно меняем адреса начала и конца Передаваемые параметры: начало списка, конец списка, длина списка	$O(n^2)$
void insertList (Node* start, Node* end, int& length, bool& insert, Node* startBegin, Node* endBegin, int& lengthBegin, int num)	Вставка одного списка в другой по индексу. Алгоритм: находим нужный элемент по индексу. Привязываем первый элемент вставл. списка к элементу, найденного по индексу. Последний элемент вставл. списка привязываем к след. элементу за элементом, найденным по индексу. Меняем длину, меняем значение bool переменной Передаваемые параметры: начало основного списка, конец основного списка, длина основного списка, проверка вставки списка в список, начало вставл списка, конец вставл списка, длина вставл списка, индекс	$\Theta(n)$ $O(n)$
void insertListBegin (Node* start, int& length, bool& insert, Node* startBegin, Node* endBegin, int& lengthBegin)	Вставка одного списка в другой в начало. Передаваемые параметры: начало основного списка, длина основного списка, проверка вставки списка в список, начало вставл списка, конец вставл списка, длина вставл списка	$O(n)$
void insertListEnd (Node* end, int& length, bool& insert, Node* startBegin, Node* endBegin, int& lengthBegin)	Вставка одного списка в другой в конец. Передаваемые параметры: конец основного списка, длина основного списка, проверка вставки списка в список, начало вставл списка, конец вставл списка, длина вставл списка	$\Theta(n)$ $O(n)$
void enterEnd(Node* start, Node* end, Node* startBegin, Node* endBegin, int& length)	Проверка последнего вхождения списка в список. Алгоритм: идем по списку до тех пор(с конца списка), пока адрес последнего элемента вставл списка не совпадет с адресом в основном списке Передаваемые параметры: начало основного списка, конец основного списка, начало вставл списка, конец	$\Theta(n/2)$ $O(n)$

	вставл списка, длина основного списка	
void enterFirst (Node* start, Node* end, Node* startBegin, Node* endBegin, int& length)	Проверка первого вхождения списка в список. Алгоритм: идем по списку до тех пор(с начала списка), пока адрес первого элемента вставл списка не совпадет с адресом в основном списке Передаваемые параметры: начало основного списка, конец основного списка, начало вставл списка, конец вставл списка, длина основного списка	$\Theta(n)$ $O(n)$
void swap(Node* start, Node* end, int& length, int num1, int num2)	Меняем значения двух элементов местами по индексам. Алгоритм: находим оба элемента, меняем значения местами Передаваемые параметры: начало списка, конец списка, длина списка, 1 индекс, 2 индекс	$\Theta(n)$ $O(n)$
void listInlist(bool check)	Проверка вхождения одного списка в другой. Вывод сообщения о вхождении списка в список в зависимости от значения bool переменной (она меняется при вставке списка в список) Передаваемые параметры: bool переменная проверки вхождения списка в список	$\Theta(1)$ $O(1)$
void listEmpty(int length)	Проверка на пустоту списка Передаваемые параметры: длина списка	$O(1)$

3. Пример работы программы

```
Введите список, который будет служить вставным

Введите первый элемент для добавления в список: 1
Список:
1
----------------------------------------
Выберите действие:
1. Добавить элемент в конец списка
0. Если ввод закончен
----------------------------------------
1
Введите элемент для добавления: 2
Список:
1 2
----------------------------------------
```

Рисунок 1 Пример ввода вставл списка

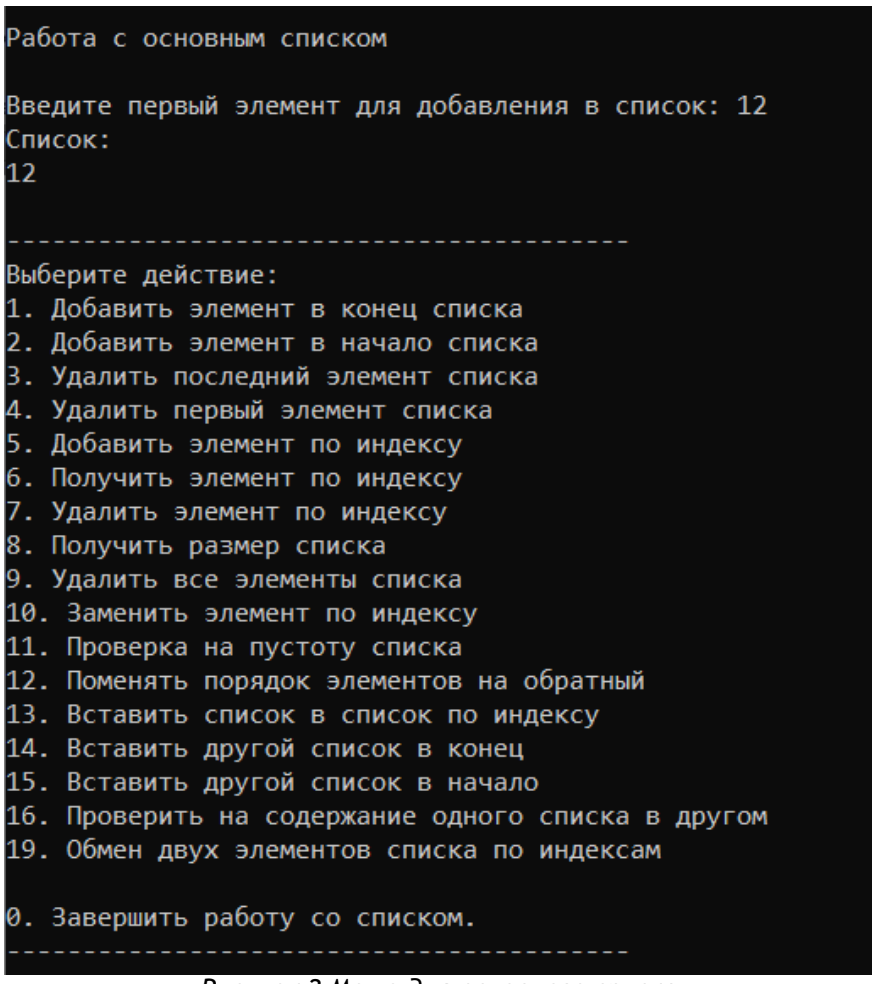
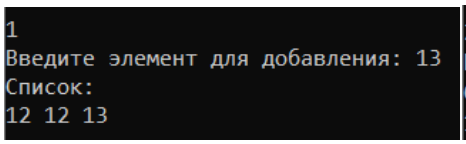
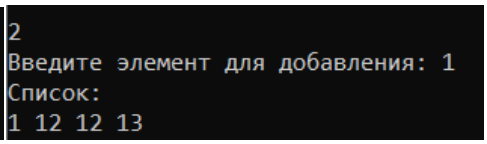
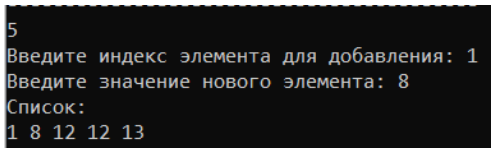


Рисунок 2 Меню для основного списка







Рисунки 3-5 Примеры добавления элементов в список

Рисунок 6 Пример замены порядка списка на обратный

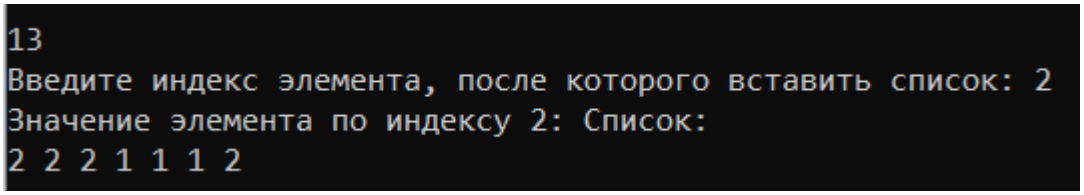


Рисунок 7 Вставка списка в список (список вставл= 1 1 1, список осн= 2 2 2 2)

```
16
Список был вставлен в список
```

Рисунок 8 Проверка вхождения списка в список

```
17
Первое вхождение списка в список (индекс): 3
```

```
18
Последнее вхождение списка в список (индекс): 5
```

Рисунки 9-10 Нахождение первого и последнего вхождения списка в список

Графики оценки временной сложности. График красного цвета – экспериментальный. График синего цвета – теоретический. Зеленый – теоретический для θ.

1. Метод добавления элемента в конец списка. Теоретическая сложность: O(1)

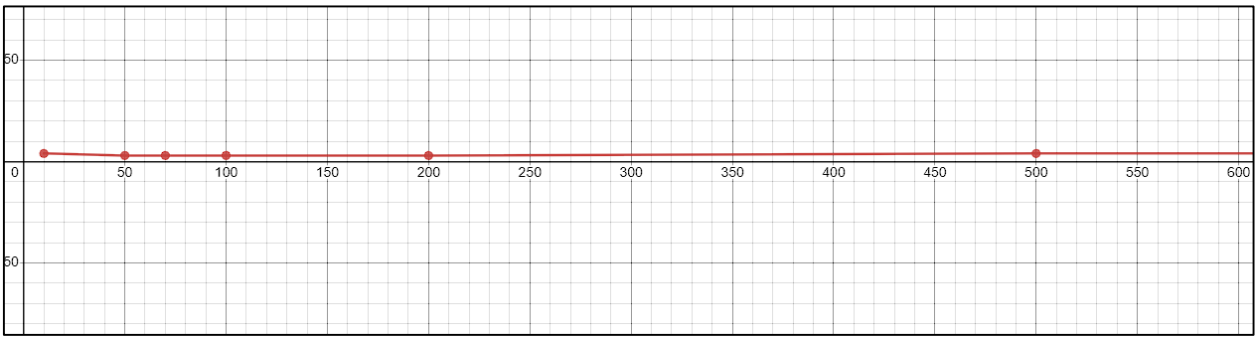


Рисунок 11 График оценки временной сложности добавления в конец списка

2. Метод добавления элемента в начало списка. Теоретическая сложность: O(1)

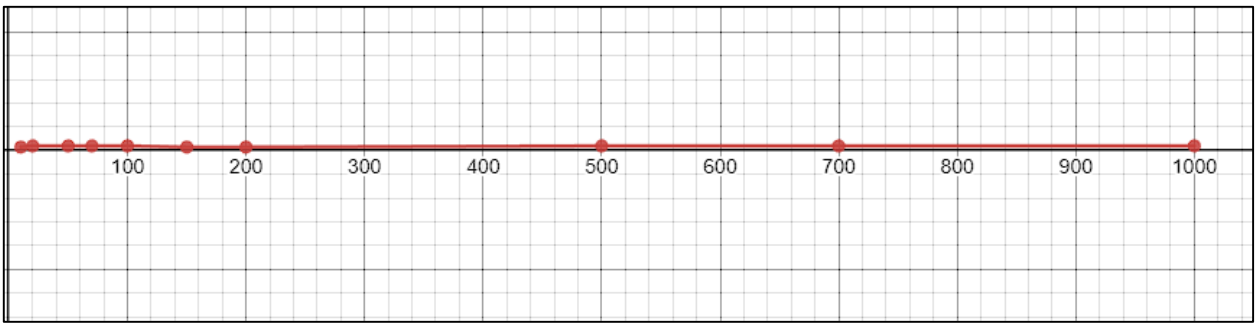


Рисунок 12 График оценки временной сложности добавления в начало

3. Метод удаления элемента из начала списка. Теоретическая сложность: O(1)

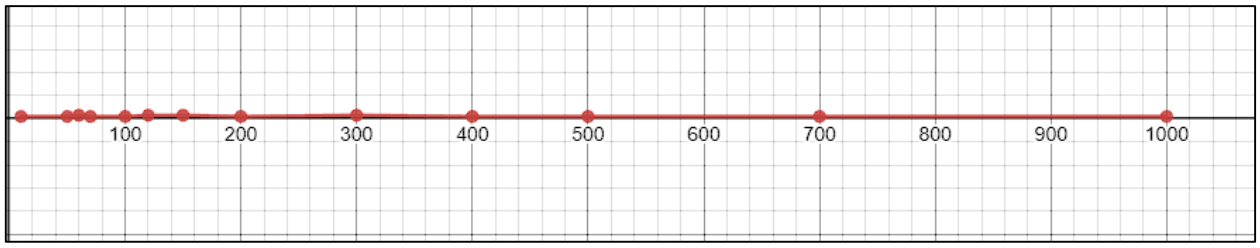


Рисунок 13 График оценки временной сложности удаления из начала

4. Метод удаления элемента из конца списка. Теоретическая сложность: O(1)

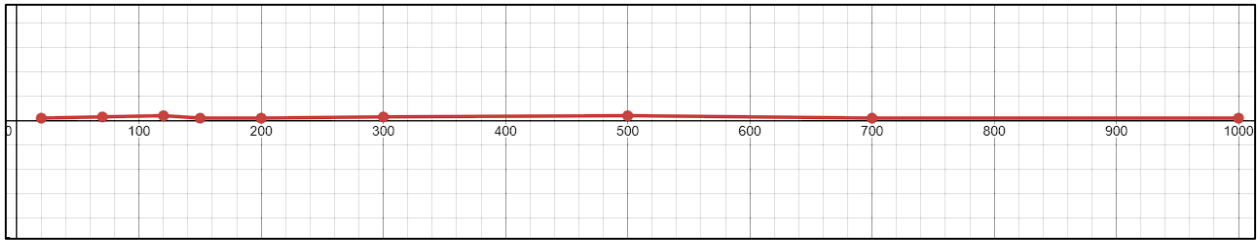


Рисунок 14 График оценки временной сложности удаления из конца

5. Метод реализации добавления элемента по индексу. Теоретическая сложность: O(n), θ(n/2)

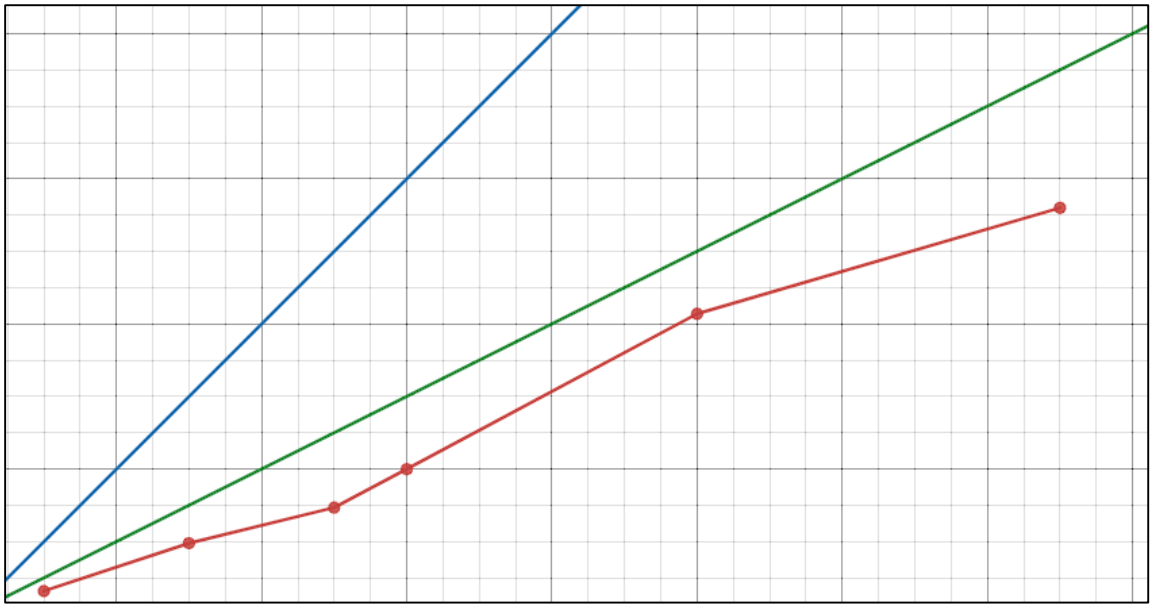


Рисунок 15 График оценки временной сложности реализации добавления элемента по индексу

6. Метод реализации нахождения элемента по индексу. Теоретическая сложность: O(n), θ(n/2)

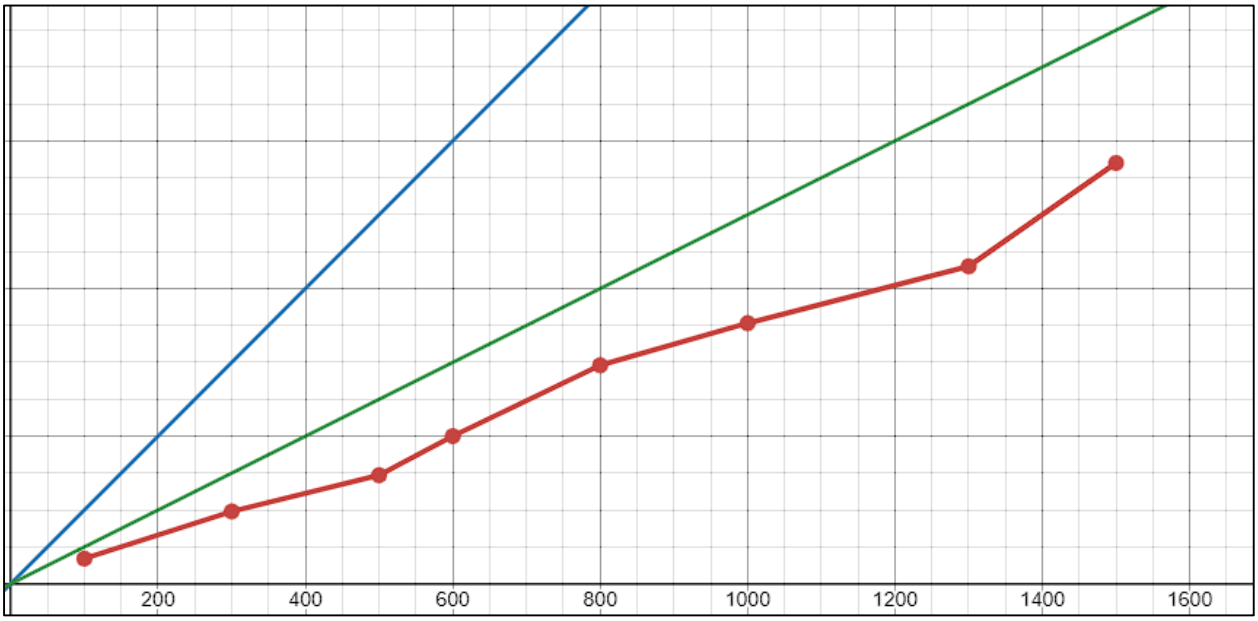


Рисунок 16 График оценки временной сложности реализации нахождения элемента по индексу

7. Метод реализации удаления элемента по индексу. Теоретическая сложность: O(n), θ(n/2)

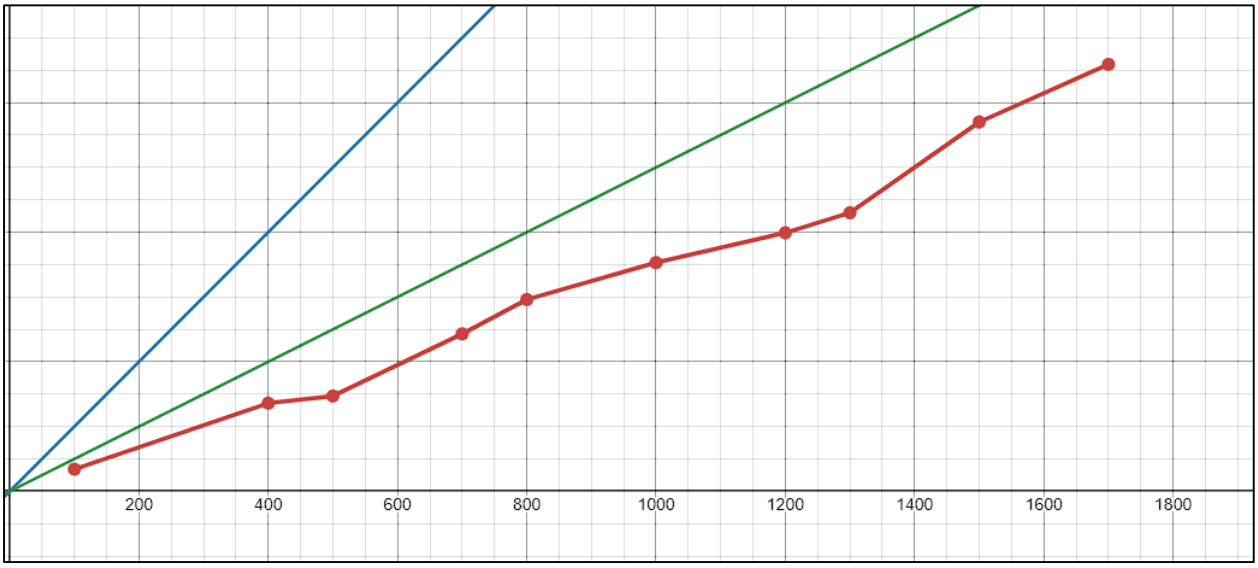


Рисунок 17 График оценки временной сложности реализации удаления элемента по индексу

8. Метод удаления списка (очистки). Теоретическая сложность: O(n)

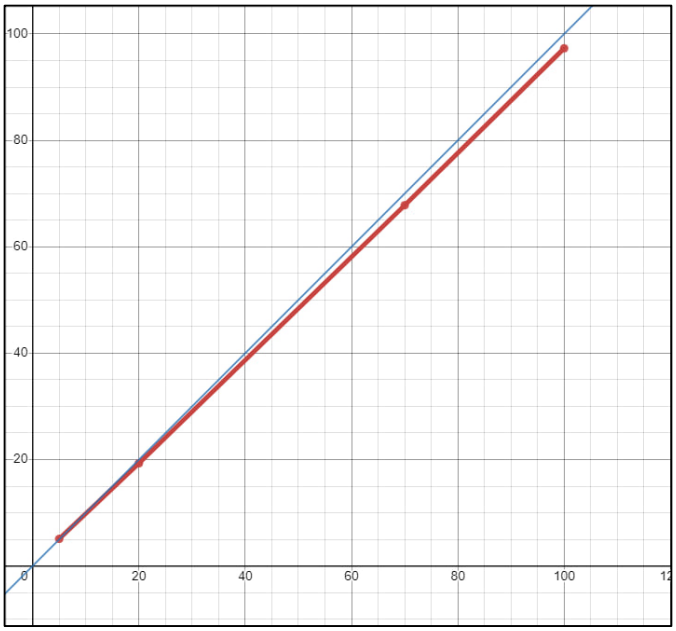


Рисунок 18 График оценки временной сложности очистки списка

9. Метод реализации изменение значения элемента по индексу. Теоретическая сложность: O(n), θ(n/2)

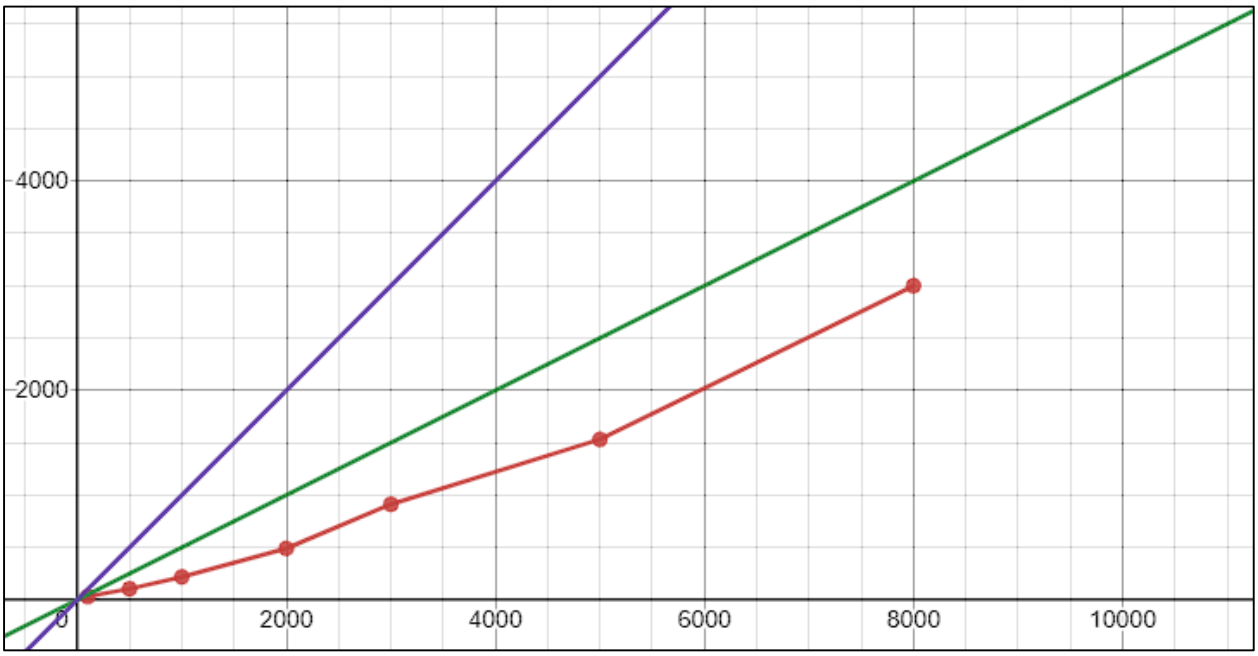


Рисунок 19 График оценки временной сложности реализации изменения зн. элемента по индексу

10.Метод изменения порядка списка. Теоретическая сложность: $O(n^2)$

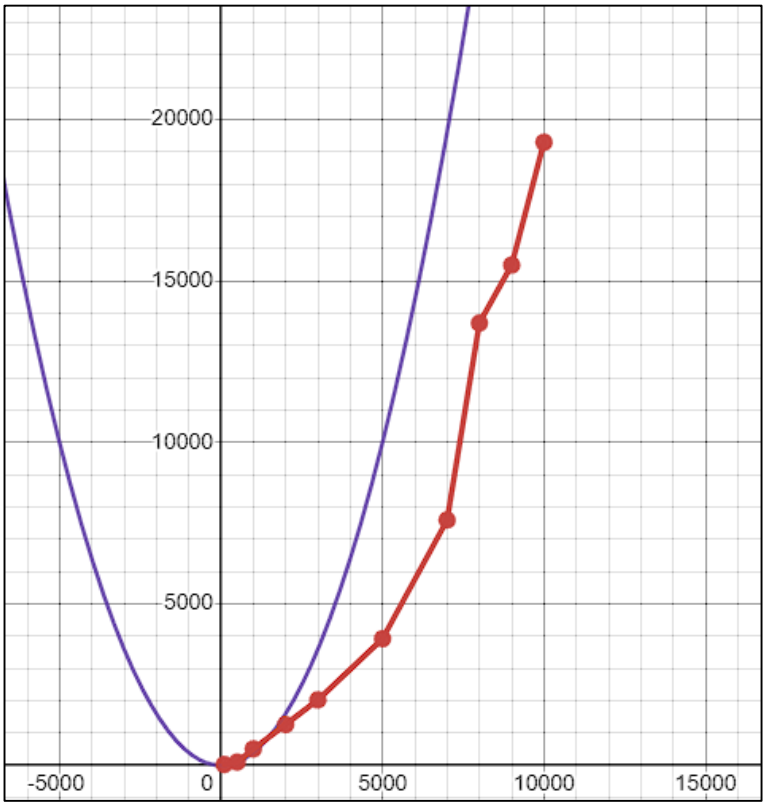


Рисунок 20 График оценки временной сложности изменения порядка списка

11.Метод вставки списка в список по индексу. Теоретическая сложность: O(n)

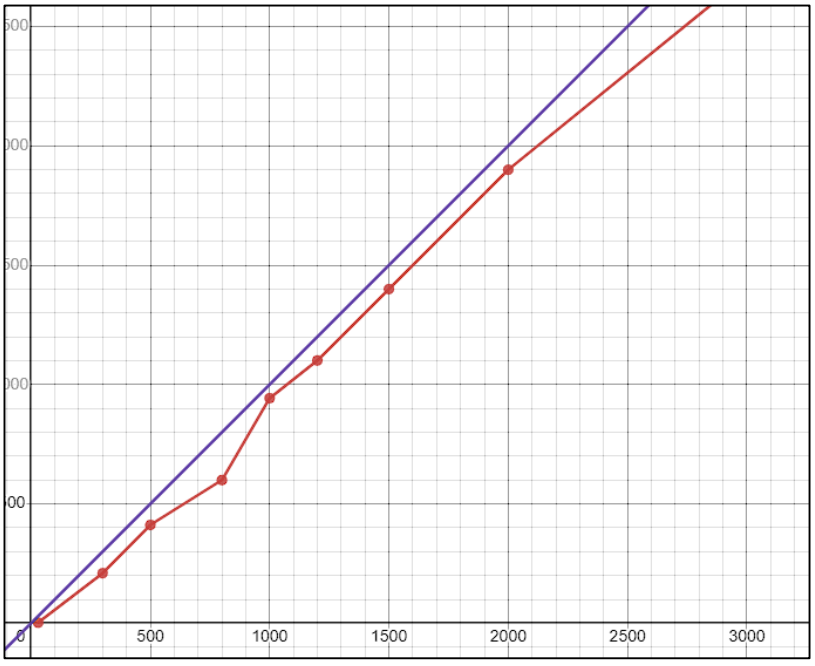


Рисунок 21 График оценки временной сложности вставки списка в список

12. Метод вставки списка в список в начало. Теоретическая сложность: O(n)

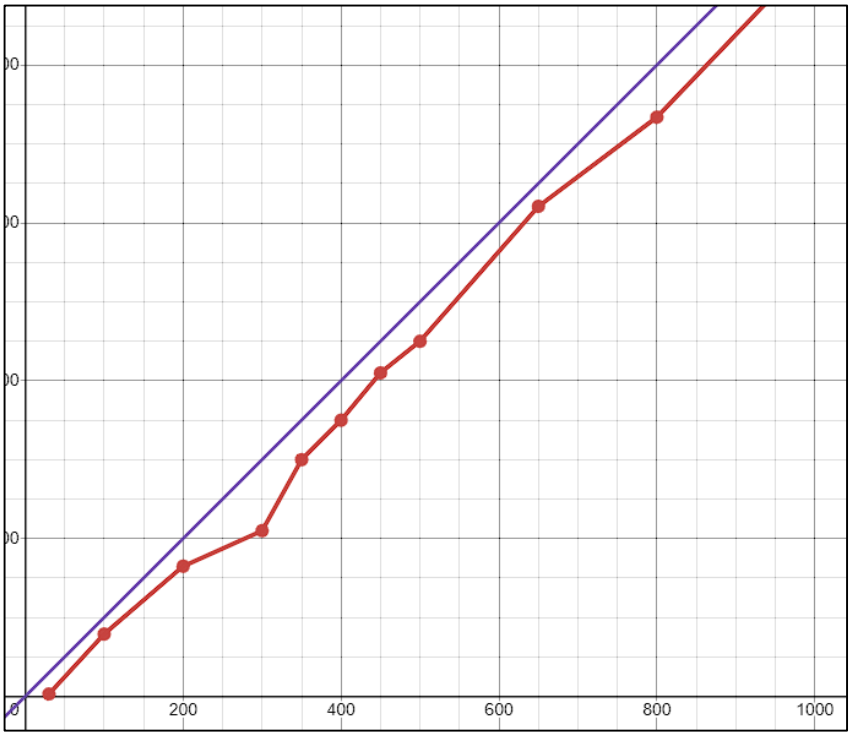


Рисунок 22 График оценки временной сложности вставки списка в список в начало

13.Метод вставки списка в список в конец. Теоретическая сложность: O(n)

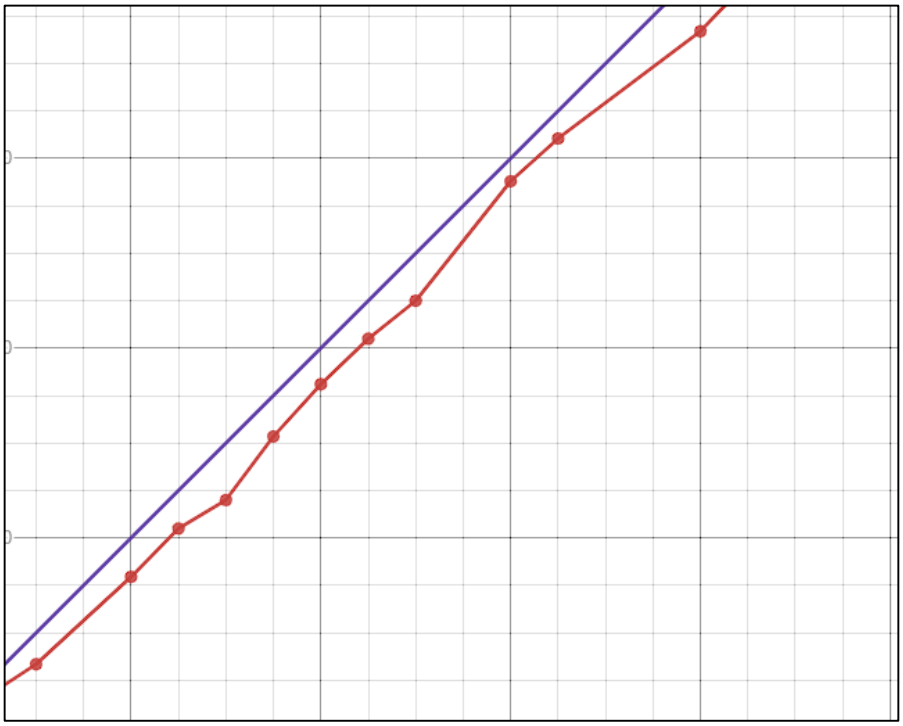


Рисунок 23 График оценки временной сложности вставки списка в список

14.Метод поиска вставки списка в список. Теоретическая сложность: O(n) (и для конца, и для начала, так как алгоритм одинаковый)

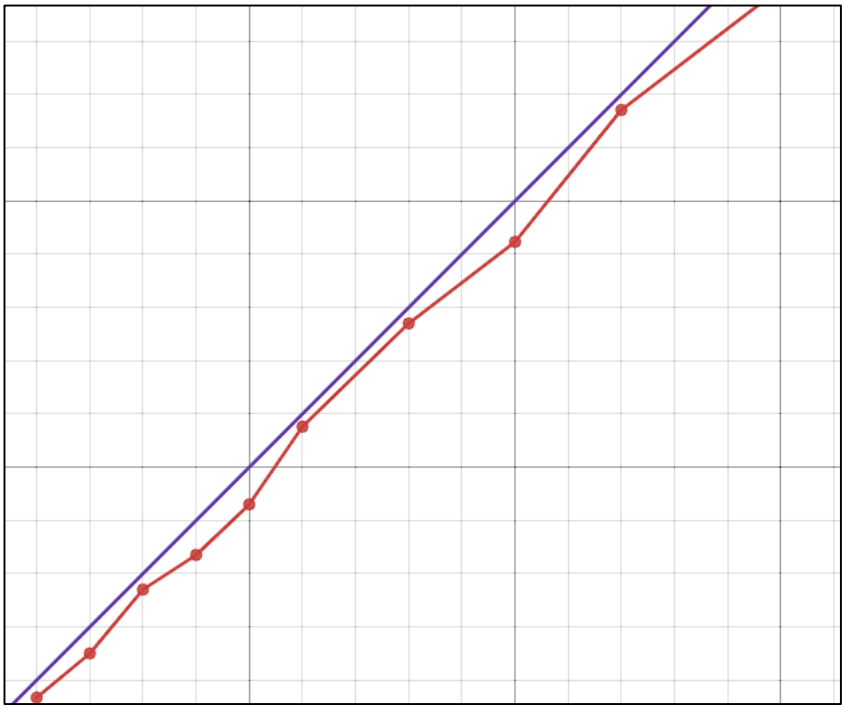


Рисунок 24 График оценки временной сложности вставки списка в список

15.Метод поиска вставки списка в список. Теоретическая сложность: O(n) (и для конца, и для начала, так как алгоритм одинаковый)

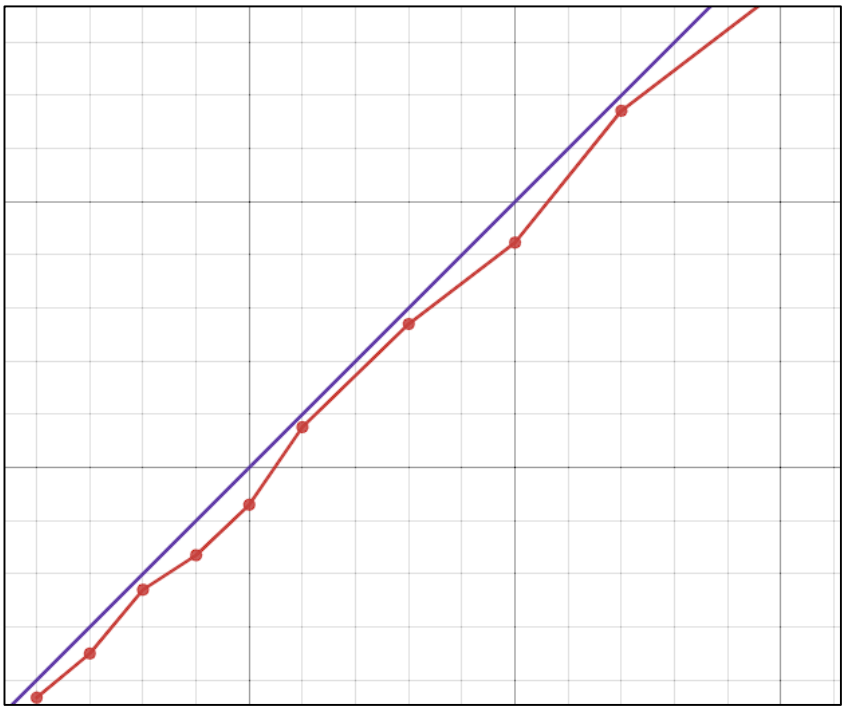


Рисунок 24 График оценки временной сложности вставки списка в список

16. Метод реализации свапа двух элементов. Теоретическая сложность: O(n), θ(n/2)

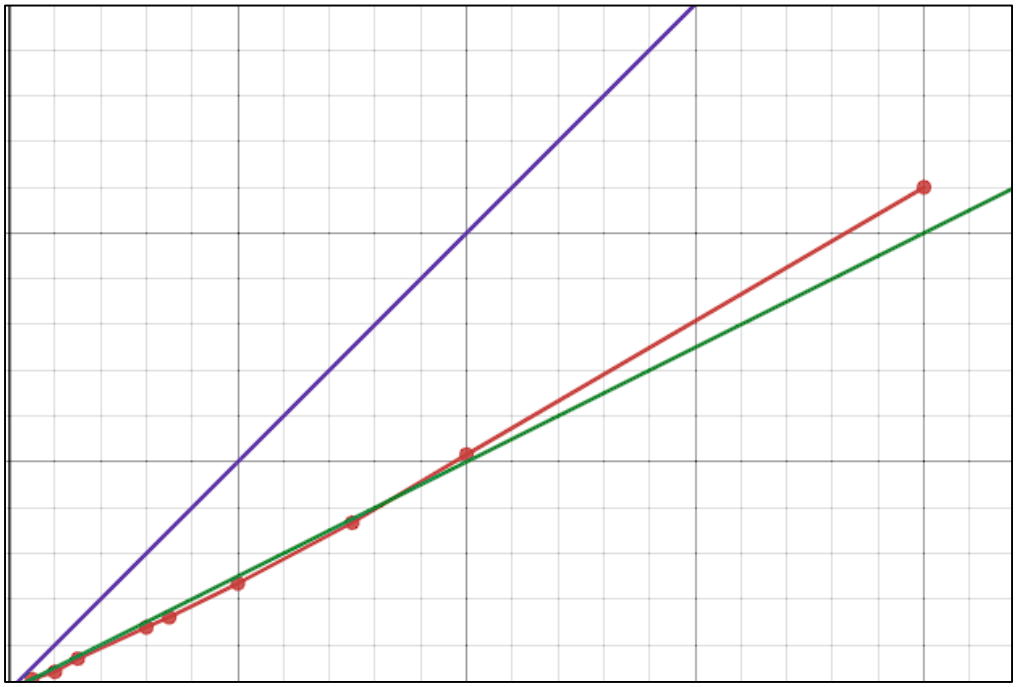


Рисунок 25 График оценки временной сложности реализации свапа двух элементов

17. Метод проверки списка на пустоту, на вход списка в список, вывод длины списка. Теоретическая сложность: O(1). Эти методы делают лишь одно сравнение или проверку

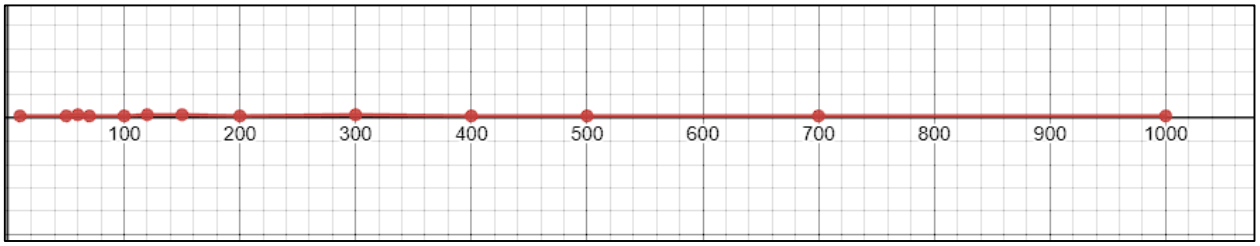


Рисунок 26 График оценки временной сложности очистки списка

4. Листинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <conio.h>
#include <windows.h>
#include <iostream>
#include <stdio.h>
#include <cstdlib>
#define LANGUAGE 1251
#define INITADD 5

using namespace std;

struct Node
{
    int number;
    struct Node* next;
    struct Node* previous;
```

```
};

void printList(Node* start, int& length)
{
    cout << "Список: " << endl;
    Node* add = start->next;
    for (int i = 0; i < length; i++)
    {
        cout << add->number << " ";
        add = add->next;
    }
    cout << endl;
}

void addLast(Node* end, int& length, int toAdd)
{
    Node* adder = new Node;
    adder->number = toAdd;
    end->previous->next = adder;
    adder->next = end;
    adder->previous = end->previous;
    end->previous = adder;
    length++;

}

void addFirst(Node* start, int& length, int toAdd)
{
    Node* adder = new Node;
    adder->number = toAdd;
    adder->next = start->next;
    start->next->previous = adder;
    start->next = adder;
    adder->previous = start;
    length++;

}

void deleteLast(Node* end, int& length) {
    Node* add = end->previous;
    end->previous = add->previous;
    add->previous->next = end;
    delete add;
    length--;
}

void deleteFirst(Node* start, int& length) {
    Node* add = start->next;
    start->next = add->next;
    add->next->previous = start;
    delete add;
    length--;
}

void addElem(Node* start, Node* end, int& length, int num) {
    if (num == 0) {
        Node* addNew = new Node;
        cout << "Введите значение нового элемента: ";
        int elem;
        cin >> elem;
        addFirst(start, length, elem);
    }
    else if ((num < length) and (num >= 0)) {
        cout << "Введите значение нового элемента: ";
        int elem;
        cin >> elem;
        if (num < ceil(length / 2)) {
            Node* add = start->next;
            for (int i = 1; i < num; i++) {
```

```
                add = add->next;
            }
            Node* addNew = new Node;
            addNew->number = elem;
            add->next->previous = addNew;
            addNew->next = add->next;
            add->next = addNew;
            addNew->previous = add;
        }
        else {
            Node* add = end->previous;
            for (int i = 1; i < length - num; i++) {
                add = add->previous;
            }
            Node* addNew = new Node;
            addNew->number = elem;
            add->previous->next = addNew;
            addNew->previous = add->previous;
            addNew->next = add;
            add->previous = addNew;
        }

        length++;

        printList(start, length);
    }
    else { cout << "Ошибка ввода, такого индекса нет" << endl; }
}

Node* findElem(Node* start, Node* end, int& length, int num) {

    if ((num < length) and (num >= 0)) {


        if (num < ceil(length / 2)) {
            Node* add = start->next;
            for (int i = 0; i < num; i++) {
                add = add->next;
            }
            return add;
        }
        else {
            Node* add = end->previous;
            for (int i = 1; i < length - num; i++) {
                add = add->previous;
            }
            return add;
        }
    }
    else { cout << "Ошибка ввода, такого индекса нет" << endl; }

}

void delElem(Node* start, Node* end, int& length, int num) {

    if ((num < length) and (num >= 0)) {

        Node* del = findElem(start, end, length, num);
        del->next->previous = del->previous;
        del->previous->next = del->next;
        length--;

    }
    else { cout << "Ошибка ввода, такого индекса нет" << endl; }
}

void delArr(Node* start, int& length) {
    Node* del = start->next;
    while (length != 0) {
```

```
        deleteFirst(start, length);

    }
    cout << "Список удален" << endl;
}

void newName(Node* start, Node* end, int& length, int num) {

    if ((num < length) and (num >= 0)) {
        int name;
        cout << "Введите новое значение элемента: ";
        cin >> name;

        Node* add = findElem(start, end, length, num);
        add->number = name;

        printList(start, length);
    }
    else { cout << "Ошибка ввода, такого индекса нет" << endl; }

}

void reverse(Node*& start, Node*& end, int& length) {
    Node* add = start->next;
    Node* temp = new Node;
    for (int i = 1; i <= length; i++) {
        temp->next = add->next;
        temp->previous = add->previous;
        add->next = temp->previous;
        add->previous = temp->next;
        add = temp->next;
    }

    temp = start;
    start = end;
    end = temp;
    start->next = start->previous;
    start->previous = NULL;
    end->previous = end->next;
    end->next = NULL;


}

void insertList(Node* start, Node* end, int& length, bool& insert, Node* startBegin, Node* endBegin,
 int& lengthBegin, int num) {
    if ((num < length) and (num >= 0)) {
        cout << "Значение элемента по индексу " << num << ": ";

        Node* add = findElem(start, end, length, num);
            add->next->previous = endBegin->previous;
            endBegin->previous->next = add->next;
            startBegin->next->previous = add;
            add->next = startBegin->next;

        length += lengthBegin;
        insert = true;
        printList(start, length);
    }
    else { cout << "Ошибка ввода, такого индекса нет" << endl; }
}

void insertListBegin(Node* start, int& length, bool& insert, Node* startBegin, Node* endBegin, int&
lengthBegin) {

    endBegin->previous->next = start->next;
    start->next->previous = endBegin->previous;
    start->next = startBegin->next;
    startBegin->next->previous = start;
```

```
    length += lengthBegin;
    insert = true;



}

void insertListEnd(Node* end, int& length, bool& insert, Node* startBegin, Node* endBegin, int& leng
thBegin) {

    startBegin->next->previous = end->previous;
    end->previous->next = startBegin->next;
    endBegin->previous->next = end;
    end->previous = endBegin->previous;
    length += lengthBegin;
    insert = true;
}

void enterEnd(Node* start, Node* end, Node* startBegin, Node* endBegin, int& length) {
    Node* add = end;
    int index = 1;
    while (!(add->previous == endBegin->previous)) {
        index += 1;
        add = add->previous;
    }
    index = length - index;
    cout << "Последнее вхождение списка в список (индекс): " << index << endl;
}

void enterFirst(Node* start, Node* end, Node* startBegin, Node* endBegin, int& length) {
    Node* add = start;
    int index = 0;
    while (!(add->next == startBegin->next)) {
        index += 1;
        add = add->next;
    }
    cout << "Первое вхождение списка в список (индекс): " << index << endl;
}

void swap(Node* start, Node* end, int& length, int num1, int num2) {
    int temp;
    Node* add1 = findElem(start, end, length, num1);
    Node* add2 = findElem(start, end, length, num2);
    temp = add1->number;
    add1->number = add2->number;
    add2->number = temp;
}

void listInlist(bool check) {
    if (check)cout << "Список был вставлен в список" << endl;
    else cout << "Вставки списка в список не было" << endl;
}

void listEmpty(int length) {
    if (length == 0) {
        cout << "Список пустой" << endl;
    }
    else cout << "Список не пустой" << endl;
}


int main()
{
    SetConsoleCP(LANGUAGE);
    SetConsoleOutputCP(LANGUAGE);

    int lengthMain = 0;
    int lengthBegin = 0;
    Node* headMain = new Node;
```

```
Node* headBegin = new Node;
Node* tailMain = new Node;
Node* tailBegin = new Node;
Node* add = NULL;
int firstElem;
int countAdded = 0;
int choice;
int choiceBegin;
bool insert = false;

cout << "Введите список, который будет служить вставным" << endl << endl;
if (lengthBegin == 0)
{
    cout << "Введите первый элемент для добавления в список: ";
    cin >> firstElem;
    add = new Node;
    add->number = firstElem;
    add->next = tailBegin;
    add->previous = headBegin;
    headBegin->next = add;
    tailBegin->previous = add;
    lengthBegin++;
    printList(headBegin, lengthBegin);
}
do {
    cout << "----------------------------------------" << endl;
    cout << "Выберите действие: " << endl;
    cout << "1. Добавить элемент в конец списка" << endl;
    cout << "0. Если ввод закончен" << endl;
    cout << "----------------------------------------" << endl;
    cin >> choiceBegin;
    switch (choiceBegin)
    {
    case 1:
        int lastElem;
        cout << "Введите элемент для добавления: ";
        cin >> lastElem;
        addLast(tailBegin, lengthBegin, lastElem);
        printList(headBegin, lengthBegin);
        break;
    }
} while (choiceBegin != 0);

cout << endl << "Работа с основным списком" << endl << endl;
do
{
    if (lengthMain == 0)
    {
        cout << "Введите первый элемент для добавления в список: ";
        cin >> firstElem;
        add = new Node;
        add->number = firstElem;
        add->next = tailMain;
        add->previous = headMain;
        headMain->next = add;
        tailMain->previous = add;
        lengthMain++;
        printList(headMain, lengthMain);
    }

    cout << endl << "----------------------------------------" << endl;
    cout << "Выберите действие: " << endl;
    cout << "1. Добавить элемент в конец списка" << endl;
    cout << "2. Добавить элемент в начало списка" << endl;
    cout << "3. Удалить последний элемент списка" << endl;
    cout << "4. Удалить первый элемент списка" << endl;
    cout << "5. Добавить элемент по индексу" << endl;
    cout << "6. Получить элемент по индексу" << endl;
    cout << "7. Удалить элемент по индексу" << endl;
```

```
cout << "8. Получить размер списка" << endl;
cout << "9. Удалить все элементы списка" << endl;
cout << "10. Заменить элемент по индексу" << endl;
cout << "11. Проверка на пустоту списка" << endl;
cout << "12. Поменять порядок элементов на обратный" << endl;
cout << "13. Вставить список в список по индексу" << endl;
cout << "14. Вставить другой список в конец" << endl;
cout << "15. Вставить другой список в начало" << endl;
cout << "16. Проверить на содержание одного списка в другом" << endl;
if (insert) {
    cout << "17. Поиск первого вхождения списка в список" << endl;
    cout << "18. Поиск последнего вхождения списка в список" << endl;
}
cout << "19. Обмен двух элементов списка по индексам" << endl;
cout << endl << "0. Завершить работу со списком." << endl;
cout << "----------------------------------------" << endl;
cin >> choice;
switch (choice)
{
case 1:
    int lastElem;
    cout << "Введите элемент для добавления: ";
    cin >> lastElem;
    addLast(tailMain, lengthMain, lastElem);
    printList(headMain, lengthMain);
    break;
case 2:
    int firstElem;
    cout << "Введите элемент для добавления: ";
    cin >> firstElem;
    addFirst(headMain, lengthMain, firstElem);
    printList(headMain, lengthMain);
    break;
case 3:
    deleteLast(tailMain, lengthMain);
    printList(headMain, lengthMain);
    break;

case 4:
    deleteFirst(headMain, lengthMain);

    printList(headMain, lengthMain);
    break;

case 5:
    cout << "Введите индекс элемента для добавления: ";
    int indexAdd;
    cin >> indexAdd;
    addElem(headMain, tailMain, lengthMain, indexAdd);
    break;

case 6:
    cout << "Введите индекс элемента: ";
    int indexFind;
    cin >> indexFind;
    cout << "Значение элемента по индексу " << indexFind << ": ";
    add = findElem(headMain, tailMain, lengthMain, indexFind);

    cout << add->number << endl;
    break;

case 7:
    cout << "Введите индекс элемента для удаления: ";
    int indexDel;
    cin >> indexDel;
    delElem(headMain, tailMain, lengthMain, indexDel);
    printList(headMain, lengthMain);
    break;
```

```
        case 8:
            cout << "Длина списка: " << lengthMain << endl;
            break;

        case 9:
            delArr(headMain, lengthMain);
            break;

        case 10:
            cout << "Введите индекс элемента: ";
            cin >> indexFind;
            newName(headMain, tailMain, lengthMain, indexFind);
            break;

        case 11:
            listEmpty(lengthMain);
            break;
        case 12:
            reverse(headMain, tailMain, lengthMain);
            printList(headMain, lengthMain);
            break;
        case 13:
            cout << "Введите индекс элемента, после которого вставить список: ";
            int indexInsert;
            cin >> indexInsert;
            insertList(headMain, tailMain, lengthMain, insert, headBegin, tailBegin, lengthBegin, in
dexInsert);
            break;
        case 14:
            insertListEnd(tailMain, lengthMain, insert, headBegin, tailBegin, lengthBegin);
            printList(headMain, lengthMain);
            break;

        case 15:
            insertListBegin(headMain, lengthMain, insert, headBegin, tailBegin, lengthBegin);
            printList(headMain, lengthMain);
            break;

        case 16:
            listInlist(insert);
            break;

        case 17:
            if (insert) {
                enterFirst(headMain, tailMain, headBegin, tailBegin, lengthMain);
            }
            break;

        case 18:
            if (insert) {
                enterEnd(headMain, tailMain, headBegin, tailBegin, lengthMain);
            }
            break;

        case 19:
            cout << "Введите индексы элементов, которые нужно поменять местами: ";
            int indexFirst;
            int indexSecond;
            cin >> indexFirst;
            cin >> indexSecond;
            swap(headMain, tailMain, lengthMain, indexFirst, indexSecond);
            printList(headMain, lengthMain);
            break;
        }
    } while (choice != 0);

    return 0;
    _getch();
}
```

Ссылка на репозиторий
https://github.com/pavlichek121/2301KomissarovPElr1.git