Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рязанский государственный радиотехнический университет

имени В. Ф. Уткина»

Кафедра «Вычислительная и прикладная математика»

**Отчет**

**по лабораторной работе № 5**

**по дисциплине**

**«Объектно-ориентированное программирование»**

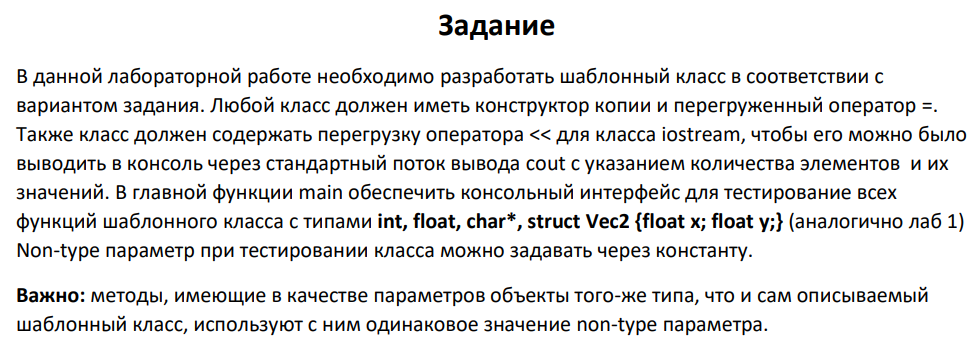
**на тему**

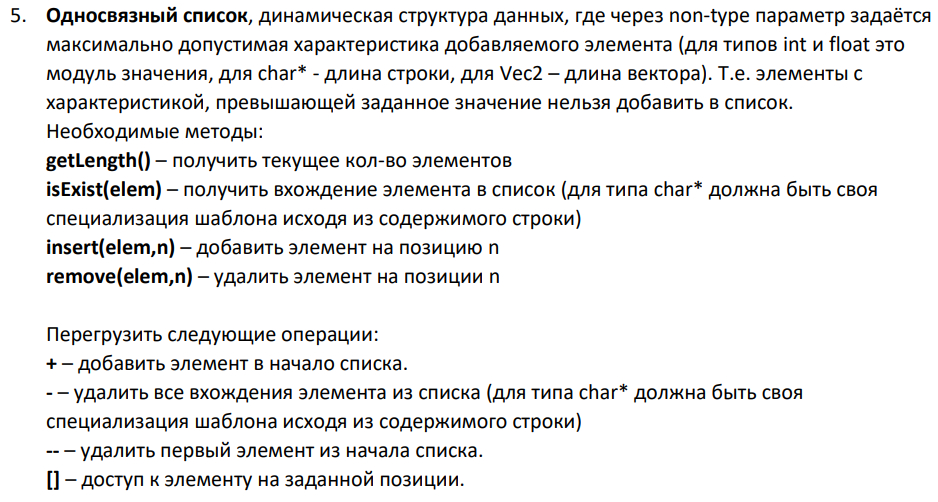
**«Шаблонные классы»**

Выполнил:   
студент гр. 143  
Вербицкая И. С.

Проверил:  
Антипов О.В.

Рязань 2022





**Описание структуры программы**

Созданный для работы с динамическим односвязным списком класс List – шаблонный класс-потомок абстрактного класса BaseList. Для List имеется несколько реализаций для различных типов, а также общая реализация. Шаблон абстрактного класса BaseList не содержит специализаций. Кроме того, дополнительно есть шаблон структуры Node, представляющей из себя структуру узла односвязного списка.

Рассмотрим состав структур и классов подробнее.

**Структура Node**

Шаблон этой структуры принимает некоторый тип данных T.

***Поля структуры:***

* T Val – значение узла
* Node<T>\* Head - указатель на следующий узел

**Класс BaseList**

Предусмотрена лишь одна общая реализация шаблона, принимающая тип данных T и некоторое целочисленное значение MaxSize.

***Поля класса:***

* int Len – количество элементов списка
* Node<T>\* Head - указатель на первый элемент списка

***Методы класса:***

* bool Empty() const – возвращает True, если список пуст
* virtual bool isCompatible (Node<T>\* Elem) – возвращает True, если элемент соответствует значению MaxSize, это чистая виртуальная функция, которая реализуется в классе List
* virtual bool isEqual (T a, T b) – возвращает True, если элементы равны
* virtual void DelAddr(Node<T>\* Addr) – удаляет из списка элемент с адресом Addr
* virtual void CreateValue (T& Value, T Elem) – записывает в Value значение Elem
* BaseList() – конструктор класса
* virtual ~BaseList() – деструктор класса
* int getLength() const – возвращает Len
* bool isIndex(int ind) const – возвращает True, если индекс ind входит в пределы списка (больше нуля и меньше его длины)
* void Clear() – полностью очищает список вместе с памятью, выделенной под элементы
* bool isExist(T Elem) – возвращает True, если элемент присутствует в списке
* virtual void print () – выводит список в консоль
* T& operator- (T Elem) – удаляет всех вхождения элемента в список
* T& operator-- () - удаляет первый элемент списка
* T& operator+ (T Elem) – добавляет элемент в начало списка
* void insert(T Elem, int n) - добавляет элемент в список по его индексу
* void remove(T Elem, int n) – удаляет элемент из списка по его индексу
* T& operator[](int index) – возвращает значение элемента по его индексу

**Класс List**

Шаблон этого класса, так же, как его родитель BaseList, принимает тип данных T и некоторое целочисленное значение MaxSize.

Для List предусмотрены следующие реализации:

* Общая реализация (базовый шаблон)
* Реализация для типа Vec2 (частичная специализация шаблона) (дополнительно к специализации – структура Vec2 – структура вектора)
* Реализация для типа char\* (частичная специализация шаблона)

**Класс List (общая реализация)**

***Методы класса:***

* bool isCompatible (Node<T>\* Elem) override – перегрузка метода BaseList, вычисляющая модуль значения элемента и сравнивающая его со значением MaxSize

**Структура Vec2**

***Поля структуры:***

* int x, int y – координаты вектора

***Методы структуры:***

* Vec2& operator= (Vec2 Value) – перегрузка оператора копирования
* bool operator== (Vec2 Value), bool operator!= (Vec2 Value) – перегрузка операторов == и !=

Кроме того, для Vec2 реализована перегрузка оператора << для вывода вектора в стандартном потоке вывода, однако она вынесена за пределы структуры:

* ostream& operator<<(ostream& os, const Vec2& vec)

**Класс List (реализация для типа Vec2)**

***Методы класса:***

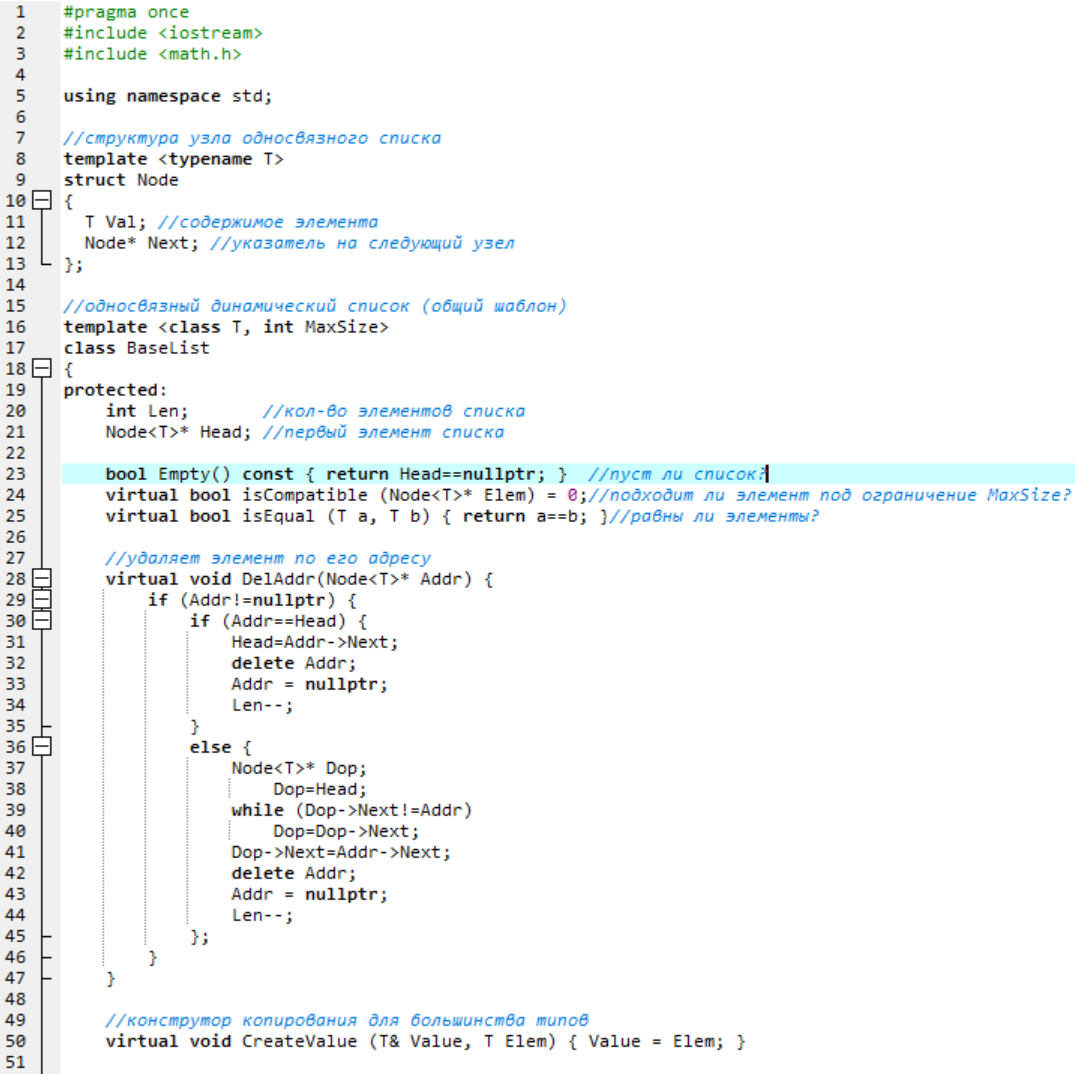
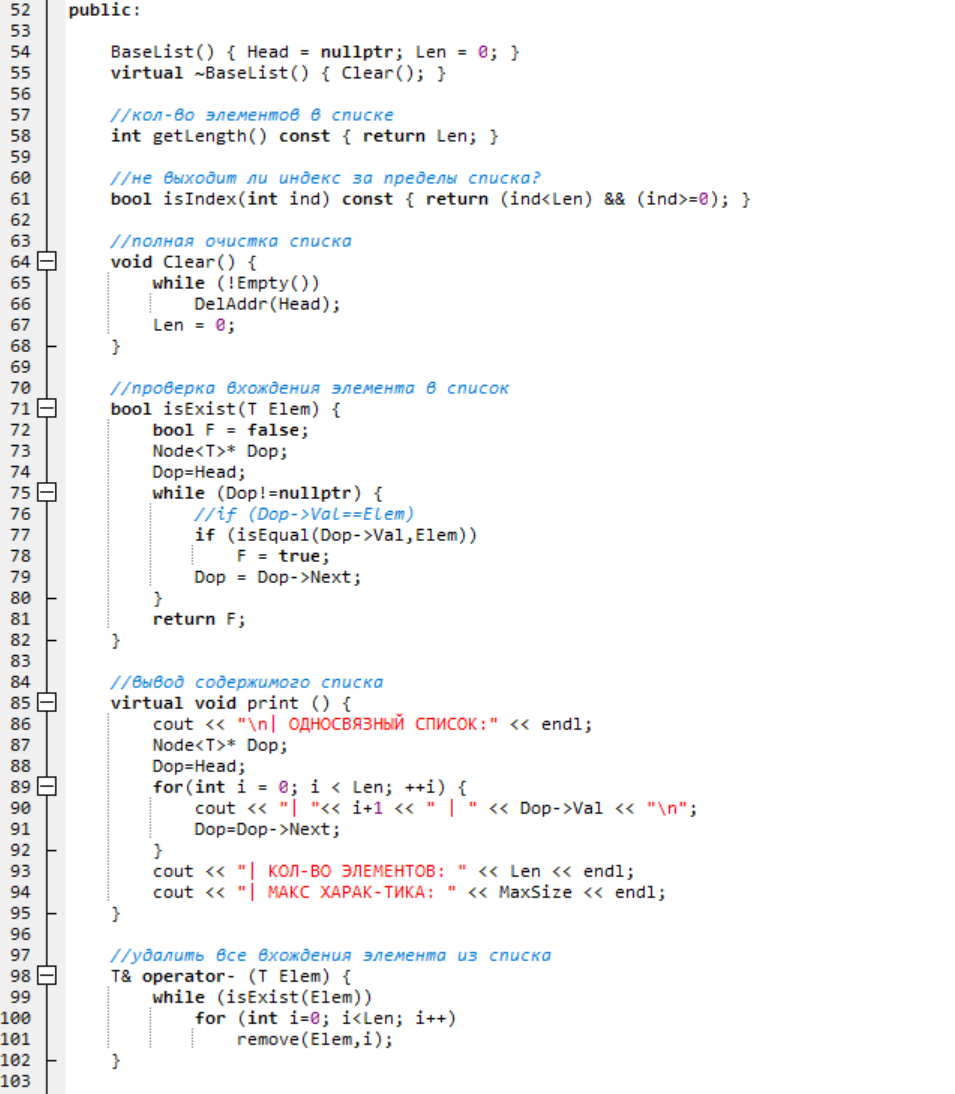
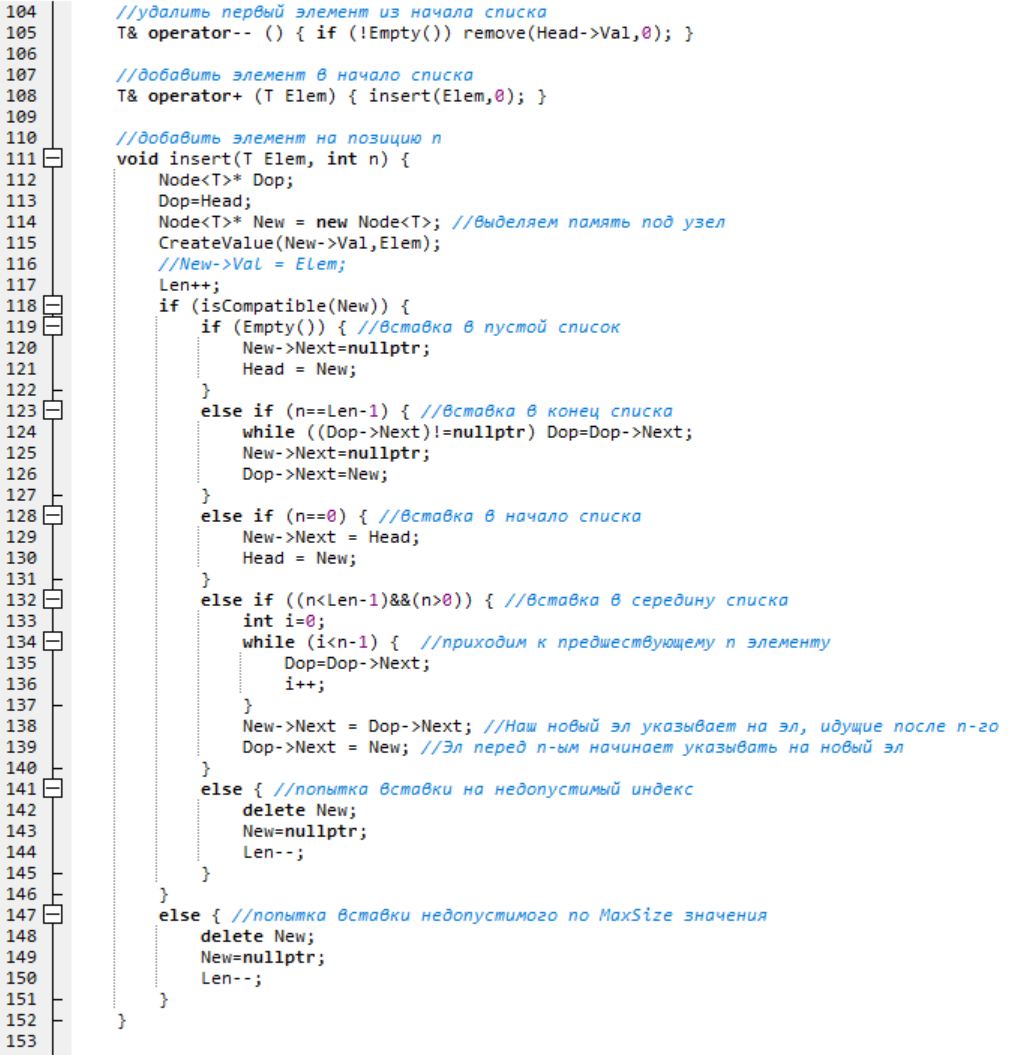
* bool isCompatible (Node<T>\* Elem) override – перегрузка метода BaseList, вычисляющая длину вектора и сравнивающая ее со значением MaxSize

**Класс List (реализация для типа char\*)**

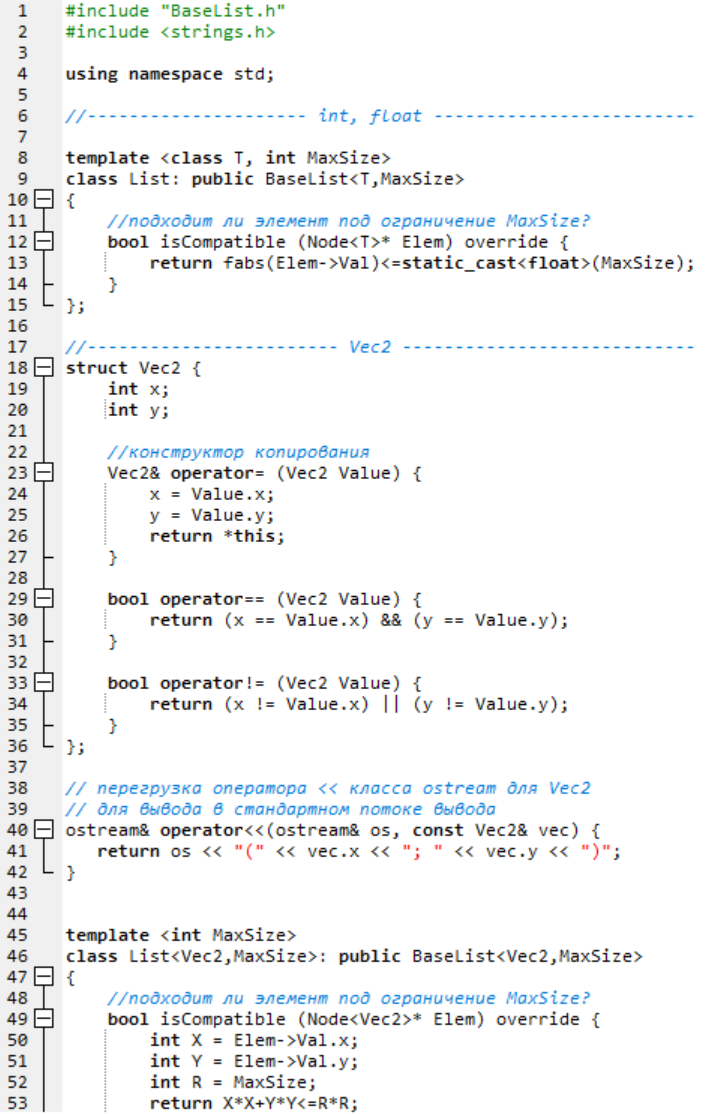
***Методы класса:***

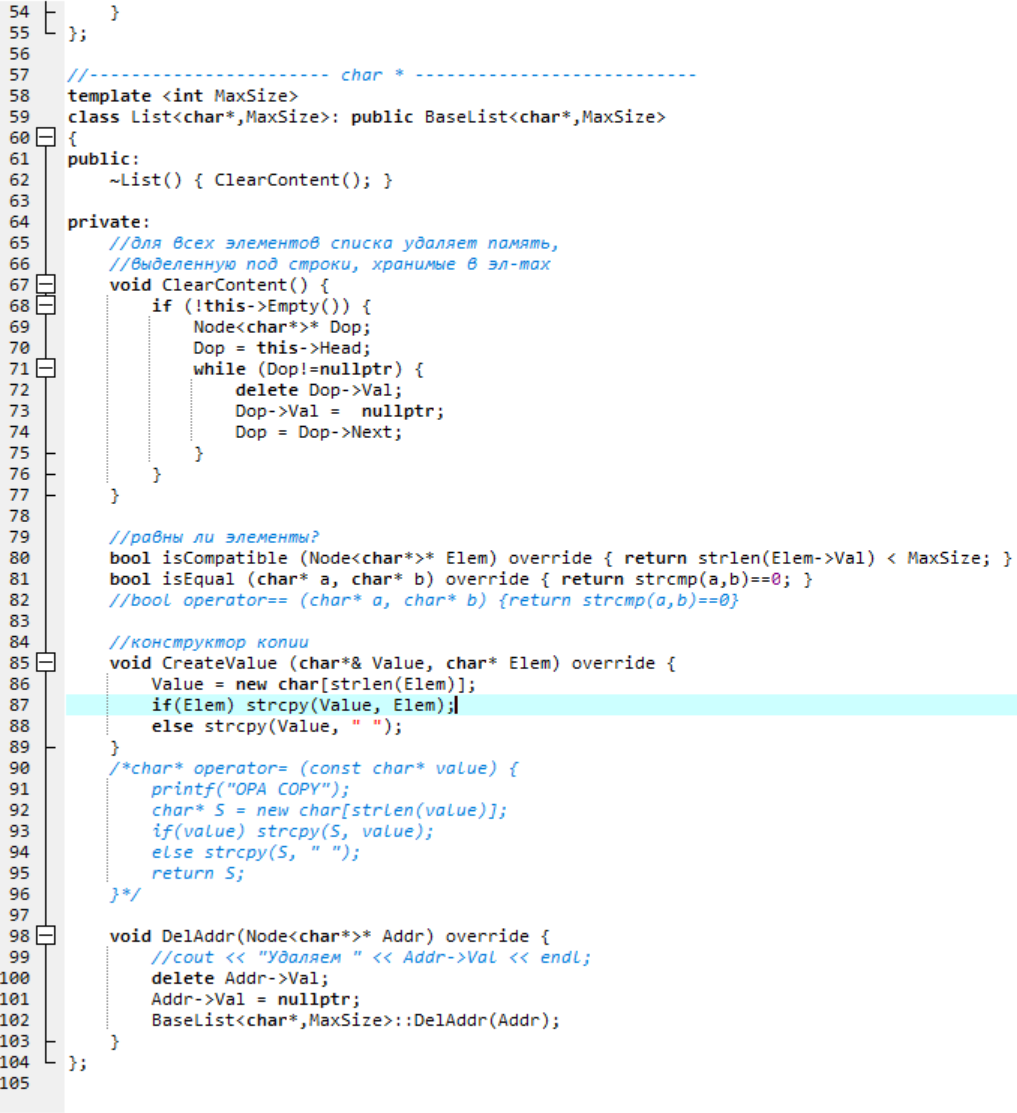
* **~**List() – измененный деструктор класса, предусматривающий также очистку динамической памяти, выделенной под строки, хранящиеся в элементах, также выделенных динамически.
* void ClearContent() – очищает память, выделенную под строки во всех элементах списка
* bool isCompatible (Node<char\*>\* Elem) override - перегрузка метода BaseList, вычисляющая длину строки и сравнивающая ее со значением MaxSize
* bool isEqual (char\* a, char\* b) override – сравнивает содержимое двух строк, и если оно совпадает возвращает True
* void CreateValue (char\*& Value, char\* Elem) override – записывает в Value содержимое строки Elem
* void DelAddr(Node<char\*>\* Addr) override – перегрузка метода удаления элемента по его адресу, предусматривающая также очистку памяти, которая выделяется на хранение самих строк в содержимом элементов списка

**Листинг программы:**

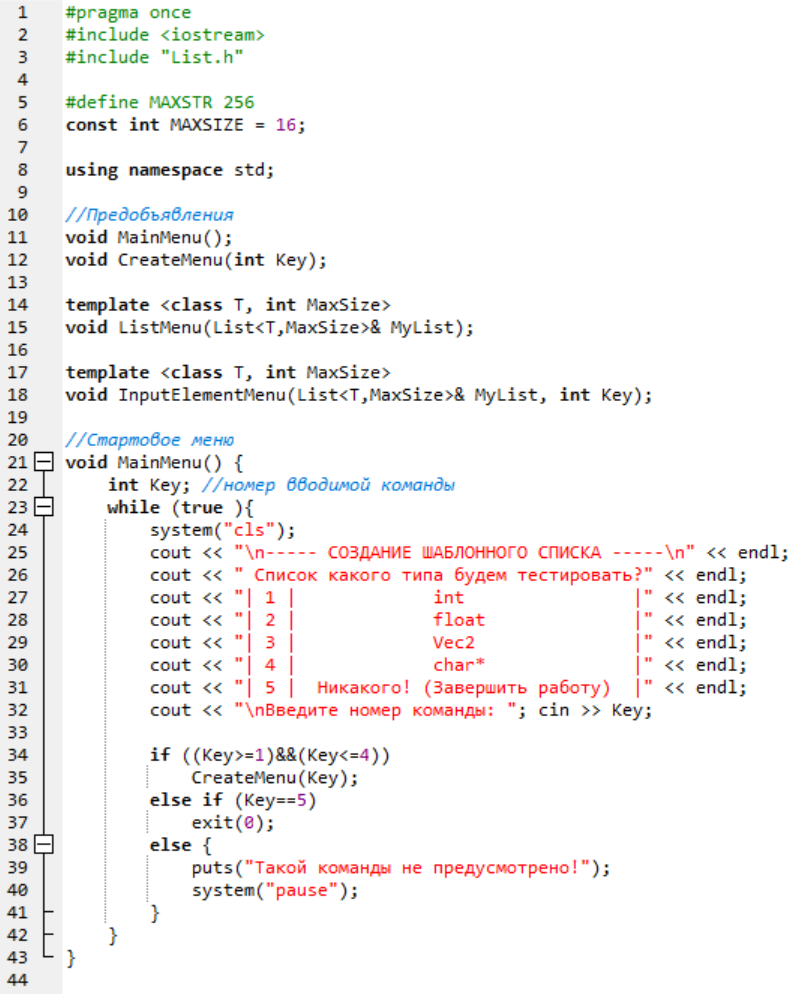
*Файл BaseList.h* **  

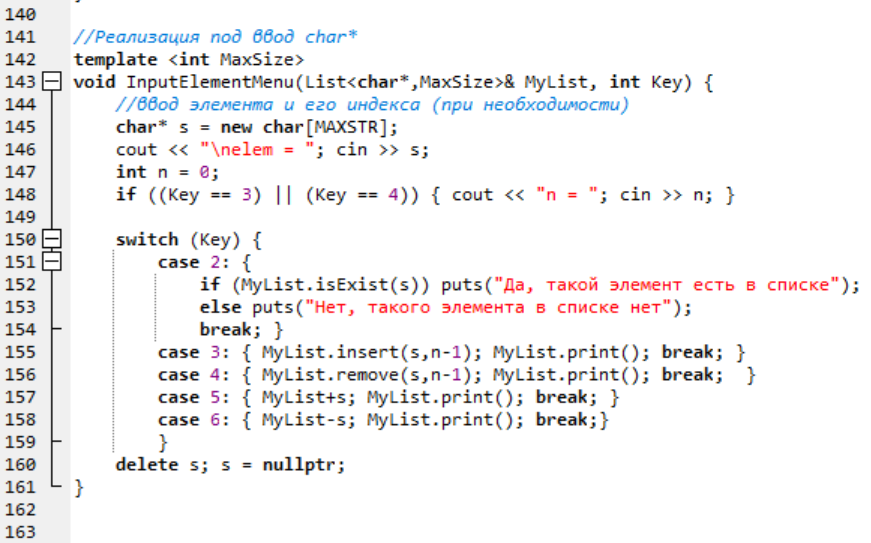
*Файл List.h*



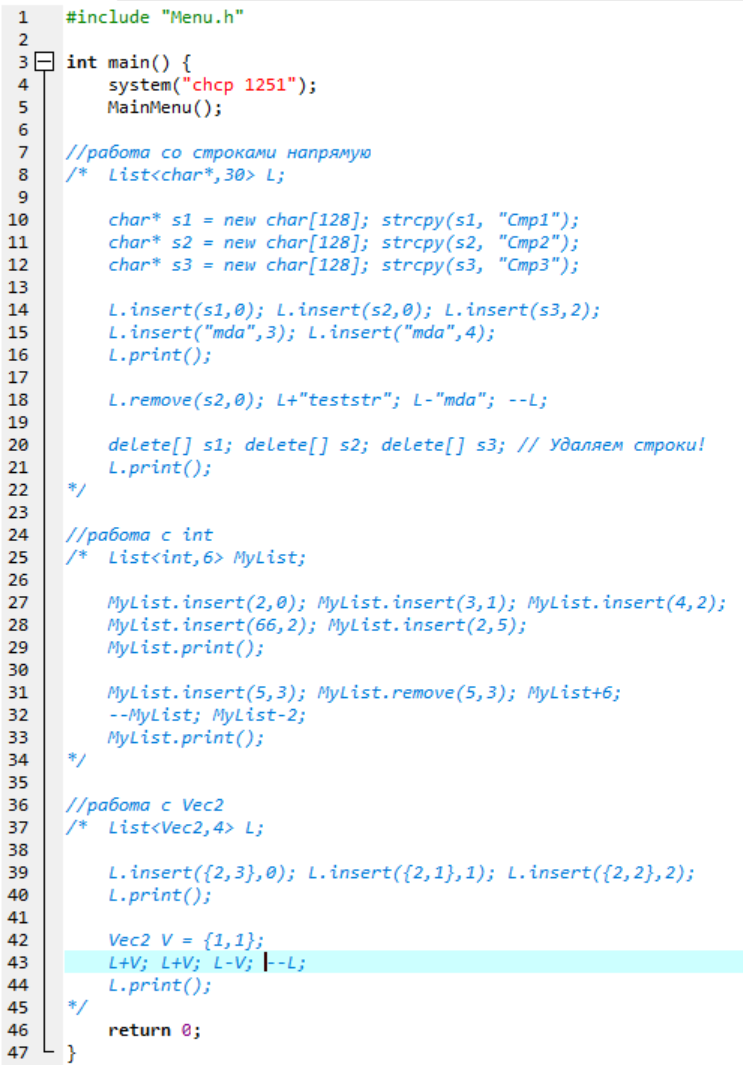


*Файл Menu.h*

**

**

*Файл main.cpp*

**

**Результаты работы программы:**

В результате выполнения лабораторной было разработано консольное приложение с двумя страницами меню (рисунок 1-2), позволяющими тестировать разработанный шаблон односвязного динамического списка, выбирая один из четырех предложенных типов данных и выполняя над списком различные действия.

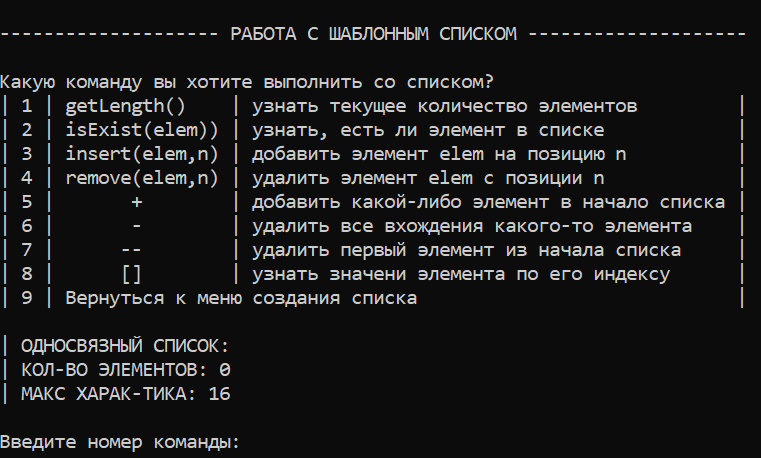
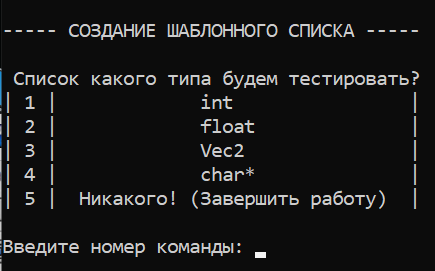


Рисунок 1 – Главное меню Рисунок 2 – Меню работы со списком

В качестве примера приведем работу списка, хранящего значения типа char\* (рисунки3-13)

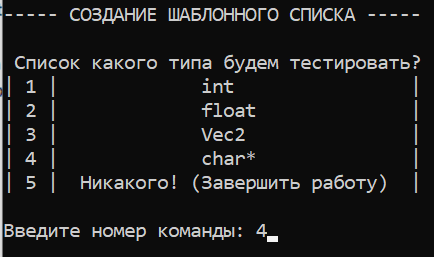


Рисунок 3 – Создание строкового списка

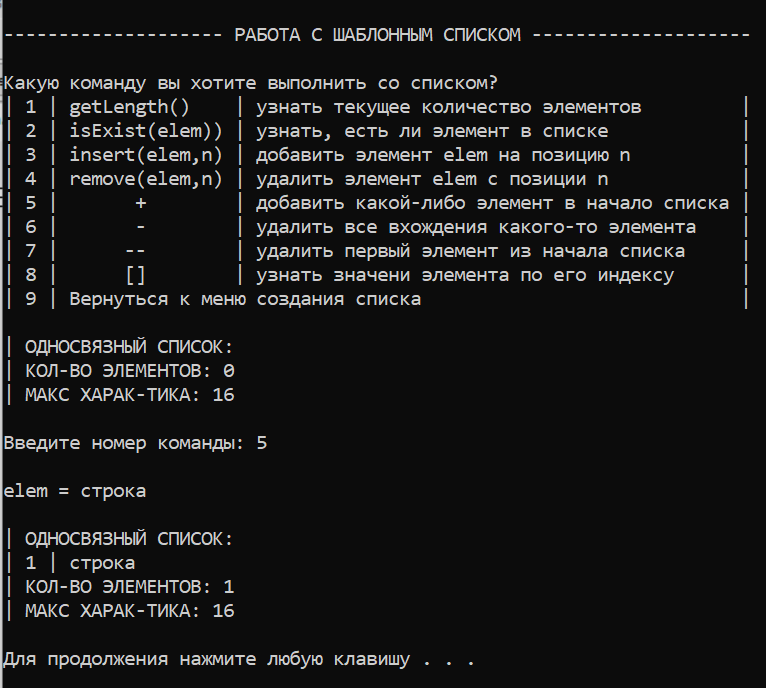


Рисунок 4 – Работа оператора +

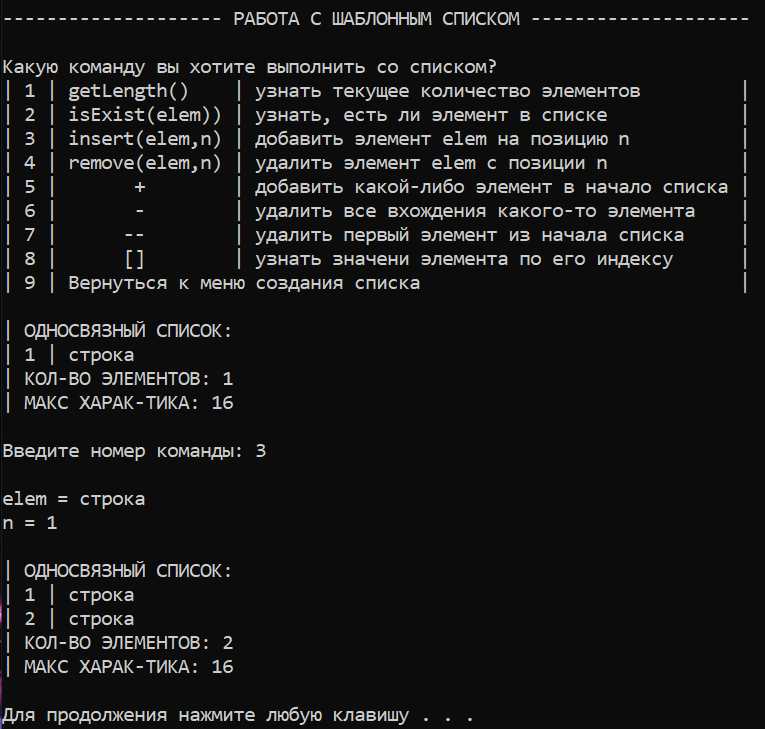


Рисунок 5 – Работа метода insert

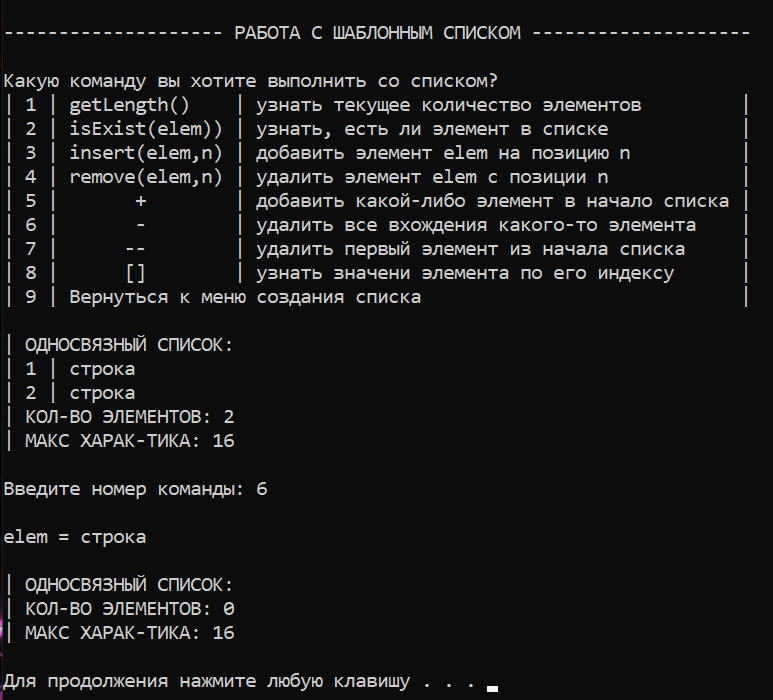


Рисунок 6 – Работа оператора -

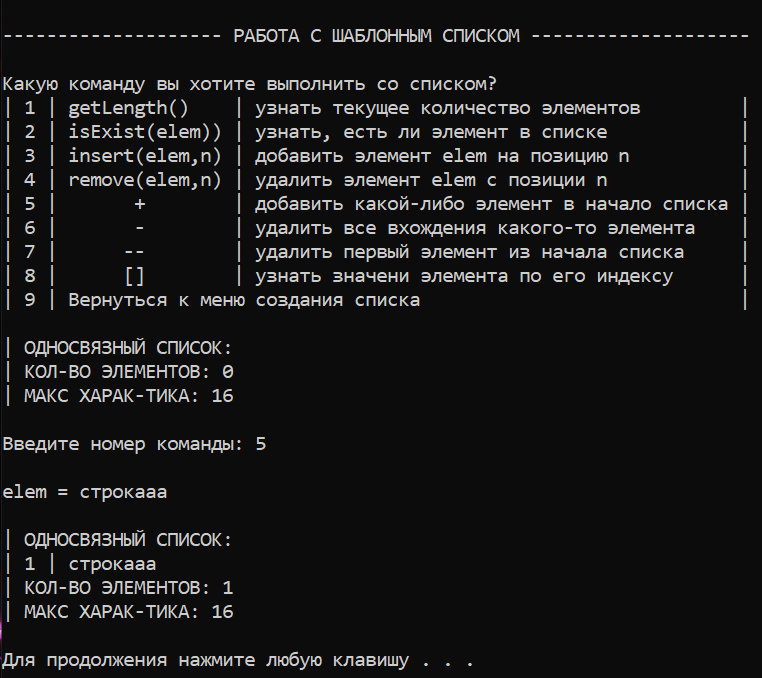


Рисунок 7 – Работа оператора +

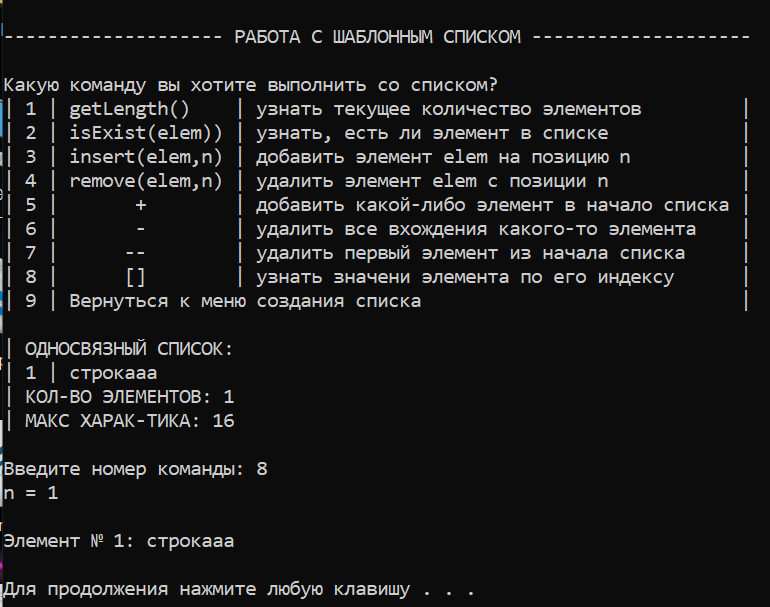


Рисунок 8 – Работа оператора []

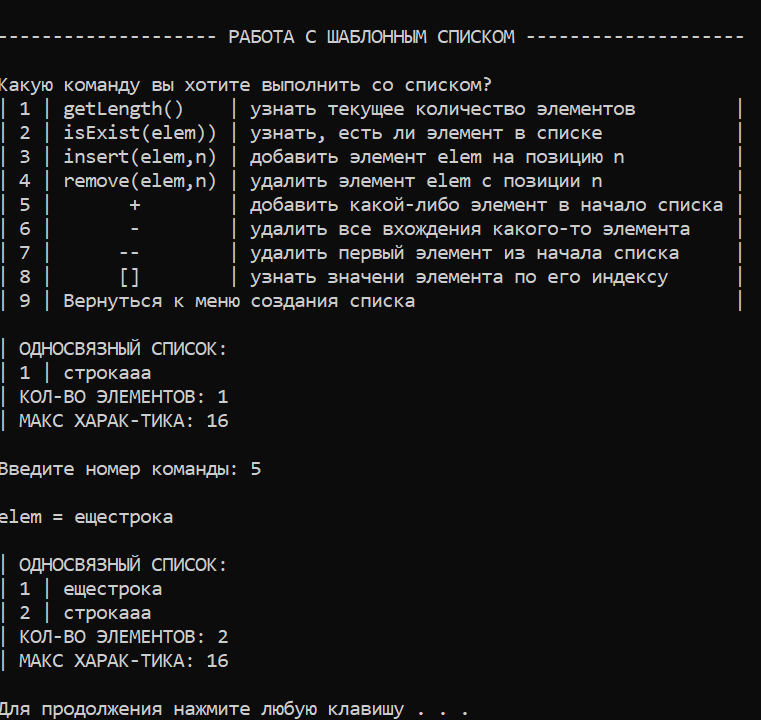


Рисунок 9 – Работа оператора +

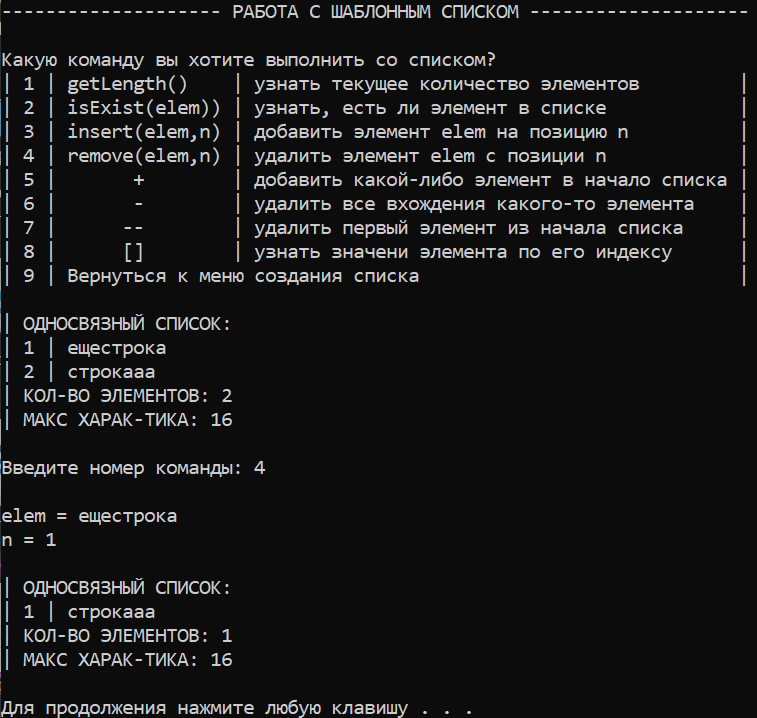


Рисунок 10 – Работа метода remove

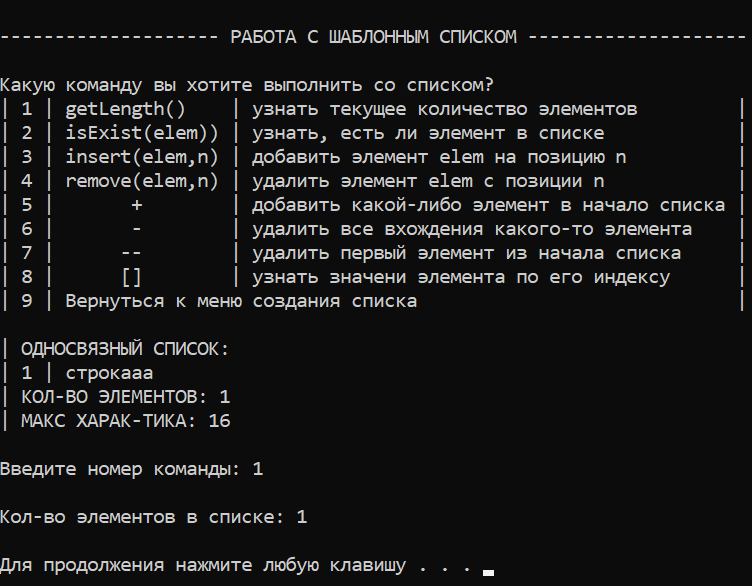


Рисунок 11 – Работа метода getLength

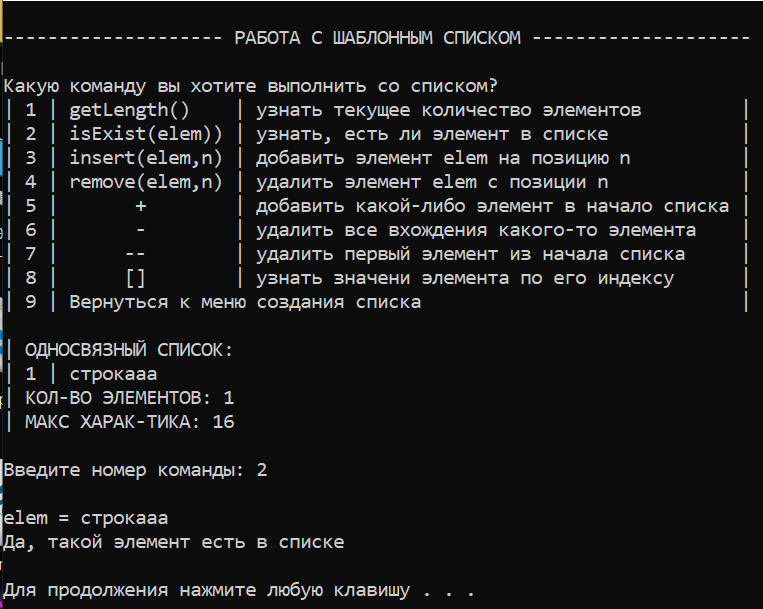


Рисунок 12 – Работа метода isExist

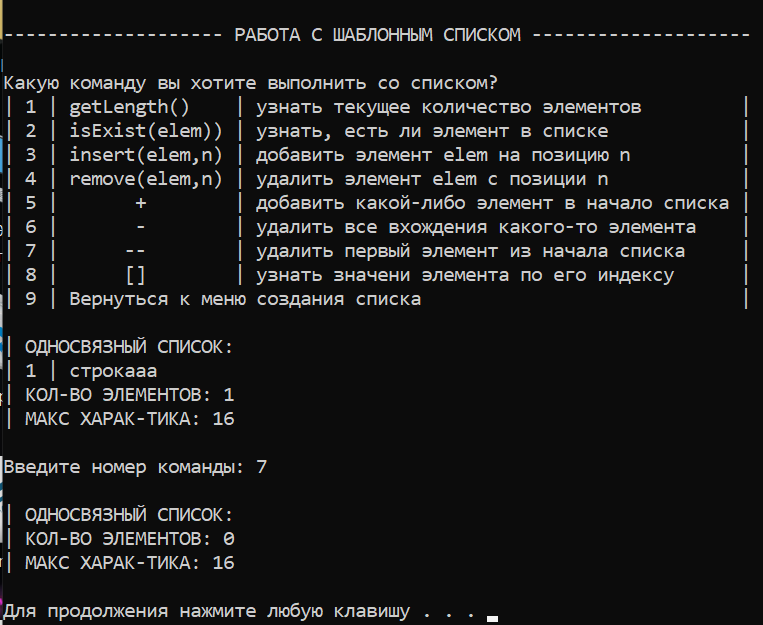


Рисунок 13 – Работа оператора --

Программа работает корректно, доступные пользователю команды работают без ошибок, некорректные действия пользователя и выходы из ОДЗ также учтены. Программа справляется с поставленной задачей.