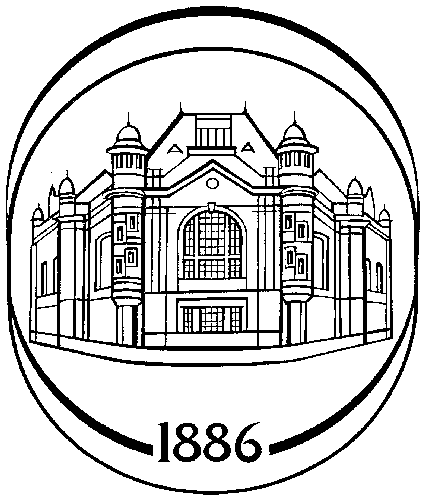
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В. И. Ульянова (Ленина) (СПбГЭТУ «ЛЭТИ»)**



Кафедра САПР

ЛАБОРАТОРНАЯ РАБОТА №1

**по дисциплине «АЛГОРИТМЫ И СТРУКТУРЫ ДАННХ»**

Тема: «Списки»

Вариант № 15

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 9301, ФКТИ |  | Кряжевских Е.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020 г.

**Постановка задачи**

Реализовать класс односвязного списка с набором методов:

1. void push\_back(int); // добавление в конец списка
2. void push\_front(int); // добавление в начало списка
3. void pop\_back(); // удаление последнего элемента
4. void pop\_front(); // удаление первого элемента
5. void insert(int, size\_t) // добавление элемента по индексу (вставка перед элементом, который был ранее доступен по этому индексу)
6. int at(size\_t); // получение элемента по индексу. Можно сделать типа size\_t
7. void remove(size\_t); // удаление элемента по индексу
8. size\_t get\_size(); // получение размера списка
9. void print\_to\_console(); // вывод элементов списка в консоль через разделитель, не использовать at
10. void clear(); // удаление всех элементов списка

11.void set(size\_t, int); // замена элемента по индексу на передаваемый элемент

12.bool isEmpty(); // проверка на пустоту списка

13.void swap(size\_t, size\_t); // обмен двух элементов списка по индексам

Программа также обязательно должна включать в себя реализацию конструктора, деструктора и содержать unit-тесты ко всем реализуемым методам. При выполнении задания запрещено пользоваться библиотекой <algorithm>.

**Описание реализуемого класса и методов.**

В своей программе я реализовала класс List, в который включила 13 методов для работы со списком, конструктор, деструктор и класс Node. Класс Node включает в себя создание элемента односвязного списка, а именно: указатель на следующий элемент, указатель на головной элемент, значение элемента, размер списка и конструктор Node. Конструктор Node принимает на вход значение элемента и указатель, после чего инициализирует элемент.

1. Метод push\_back.

Метод вставляет новый элемент в конец списка. На вход принимает значение элемента. Функция проходит по списку в поиске последнего элемента и на месте следующего элемента по адресу создает новый. Если список пустой, то создает первый элемент. После чего увеличиваем размер списка на один. Временная сложность: O(n).

1. Метод push\_front.

Метод вставляет новый элемент в начало списка. На вход принимает значение элемента. Функция создает новый головной элемент по адресу головного элемента. После чего увеличиваем размер списка на один. Временная сложность: O(1).

1. Метод insert.

Метод вставляет новый элемент в список по заданному индексу. Вставка должна быть перед элементом, который был ранее доступен по этому индексу. На вход функция принимает значение элемента и индекс. Функция проходит по списку в поиске предыдущего элемента и создает элемент по нужному индексу (следующий элемент после предыдущего). После чего увеличиваем размер списка на один. Временная сложность: O(n).

1. Метод at.

Метод получает значение элемента по определенному индексу. На вход функция принимает индекс элемента. Функция проходит по списку в поиске нужно элемента и возвращает значение этого элемента. Временная сложность: O(n).

1. Метод print\_to\_console.

Метод выводит на экран весь список элементов. На вход ничего не принимает. Функция проходит по списку и выводит на экран один элемент за другим. Временная сложность: O(n).

1. Метод isEmpty.

Метод проверяет список на отсутствие элементов. На вход ничего не принимает. Если головной элемент нулевой, то возвращает значение truth, иначе возвращает значение false. Временная сложность: O(1).

1. Метод pop\_front.

Метод удаляет первый элемент списка. На вход ничего не принимает. Функция запоминает первый элемент, второй элемент делает головным и удаляет первый элемент. После чего уменьшает размер списка на один. Временная сложность: O(1).

1. Метод remove.

Метод удаляет элемент по заданному индексу. На вход принимает индекс удаляемого элемента. Функция ищет предыдущий элемент, запоминает элемент, следующий после предыдущего (нужный нам элемент), на его место вставляет элемент, идущий после нужного индекса, и удаляет элемент по нужному индексу. После чего уменьшает размер списка на один. Временная сложность: O(n).

1. Метод pop\_back.

Метод удаляет элемент в конце списка. На вход ничего не принимает. Функция ищет предпоследний элемент, запоминает последний элемент, следующему элементу после удаляемого присваивает нулевой адрес и удаляем последний элемент списка. После чего уменьшает размер списка на один. Временная сложность: O(n).

1. Метод set.

Метод устанавливает новое значение вместо значения элемента по индексу. На вход принимает новое значение и индекс. Функция проходит по списку в поисках нужного элемента и присваивает этому элементу новое значение. Временная сложность: O(n).

1. Метод swap.

Метод меняет два элемента из списка местами. На вход получает два индекса: индекс первого элемента и индекс второго элемента. Функция ищет нужные элементы по заданным индексам, запоминает значение первого элемента, первому элементу присваивает значение второго элемента, а второму элементу – ранее запомненное значение первого элемента. Временная сложность: O(n).

1. Метод get\_size.

Метод выводит размер списка. На вход ничего не принимает. Функция возвращает значение переменной, которая отвечает за подсчет размера списка. Временная сложность: O(1).

1. Метод clear.

Метод удаляет все элементы из списка. На вход ничего не принимает. Пока размер списка не равен нулю, функция удаляет первый элемент. Временная сложность: O(n).

**Описание реализованных unit-тестов.**

Для проверки работоспособности программы реализовано 29 unit-тестов.

Три теста для метода At: ввод верного индекса, ввод неверного индекса, ввод индекса при пустом списке.

Два теста для метода push\_back: вставка обычная, вставка при пустом списке.

Два теста для метода push\_front: вставка обычная, вставка при пустом списке.

Пять тестов для метода insert: вставка в конец, вставка в начало, вставка с середину, вставка при пустом списке, вставка при вводе неверного индекса.

Два теста для метода pop\_front: обычное удаление, удаление при пустом списке.

Два теста для метода pop\_back: обычное удаление, удаление при пустом списке.

Четыре теста для метода remove: удаление в конце, удаление в начале, удаление в середине, удаление при пустом списке.

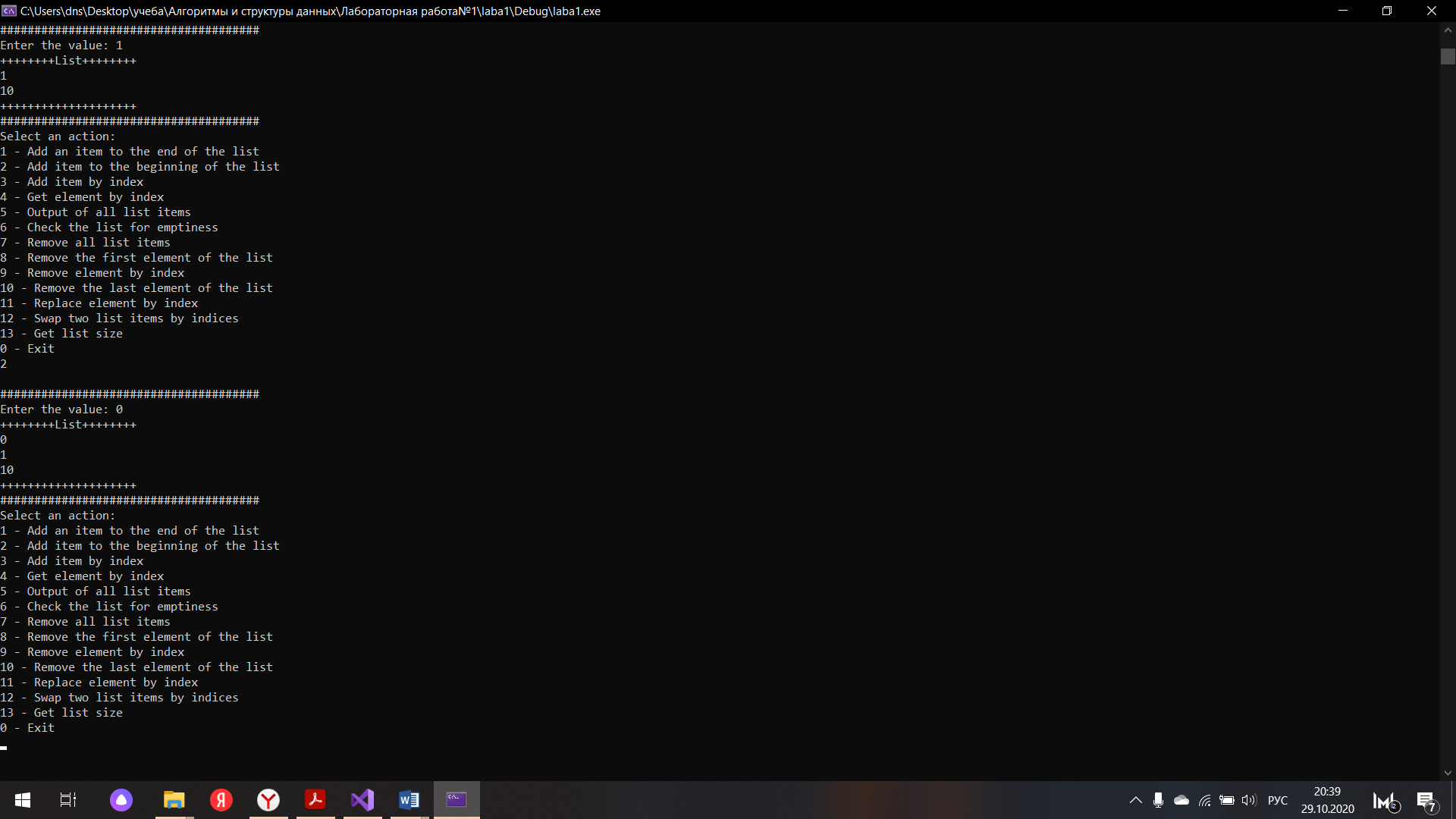
Три теста для метода set: некорректный индекс, обычная замена значения, замена значения при пустом списке.

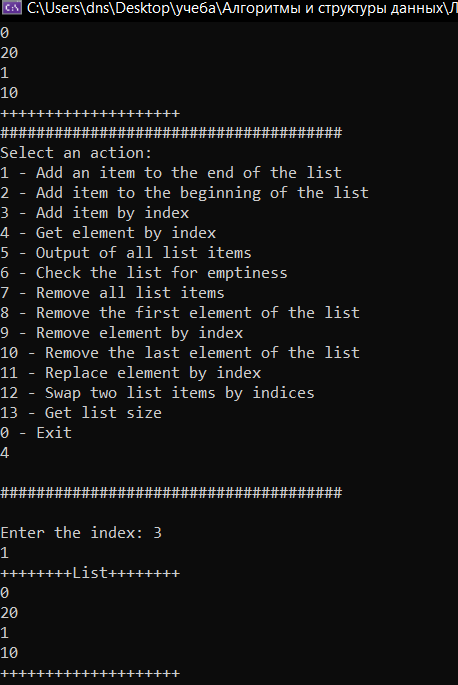
Четыре теста для метода swap: обмен значениями при некорректном индексе, при пустом списке, при равных индексах и в середине списка.

Два теста для метода get\_size: размер списка при непустом списке и размер списка при пустом списке.

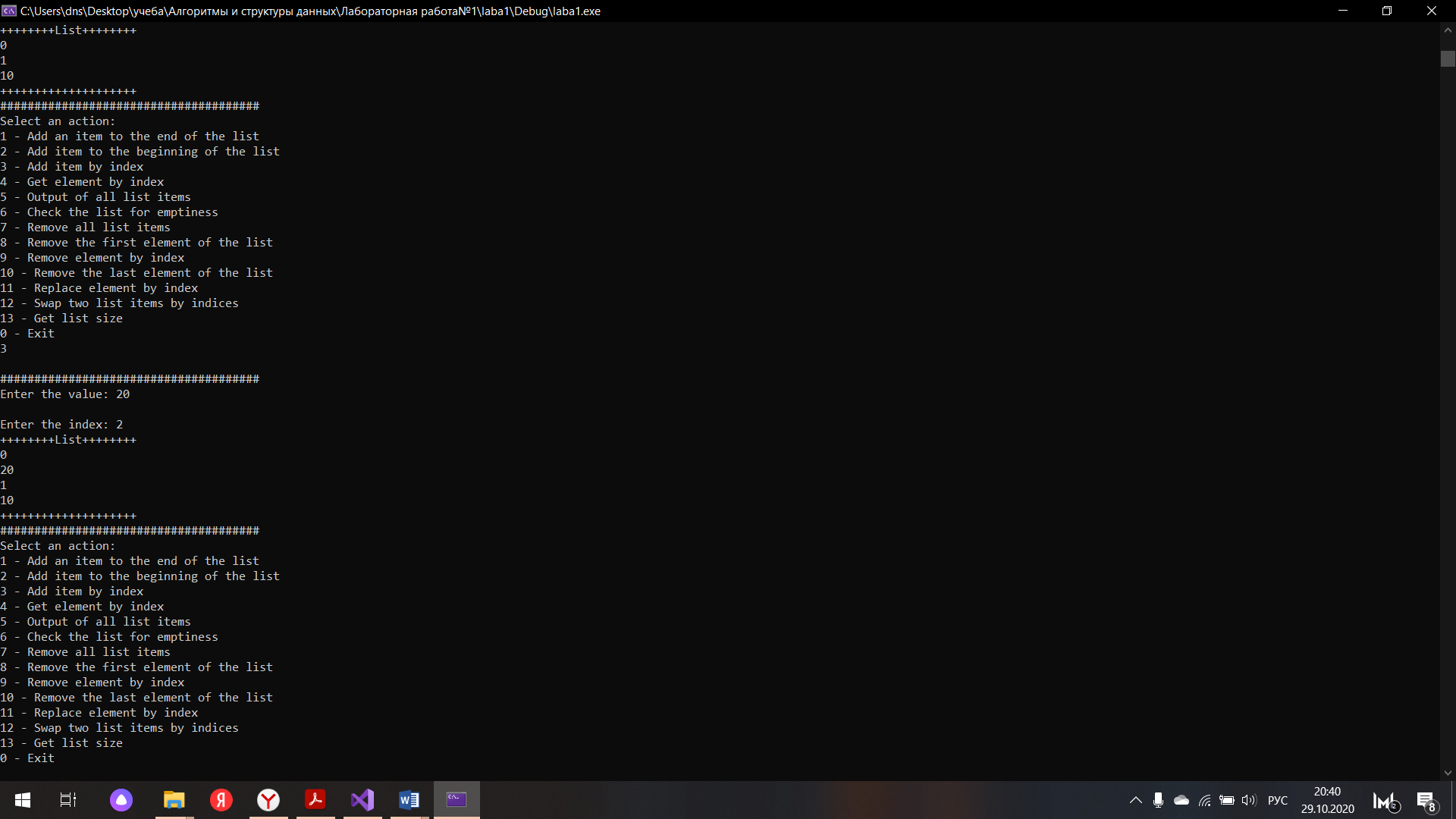
**Пример работы.**

Пример 1 – Вставка в конец списка.

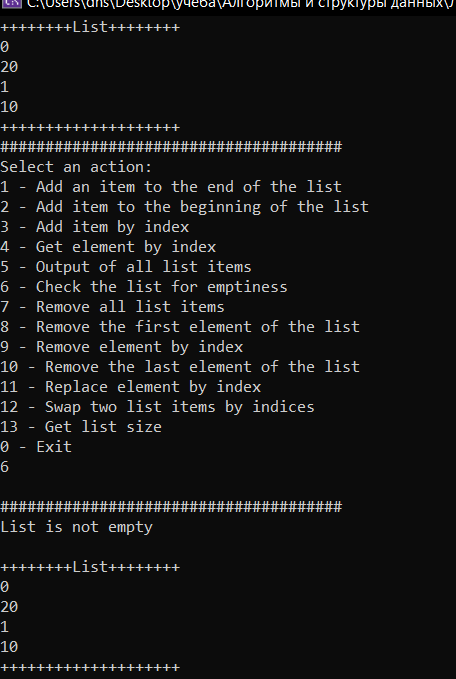
****

Пример 3 – Получение элемента из списка.  


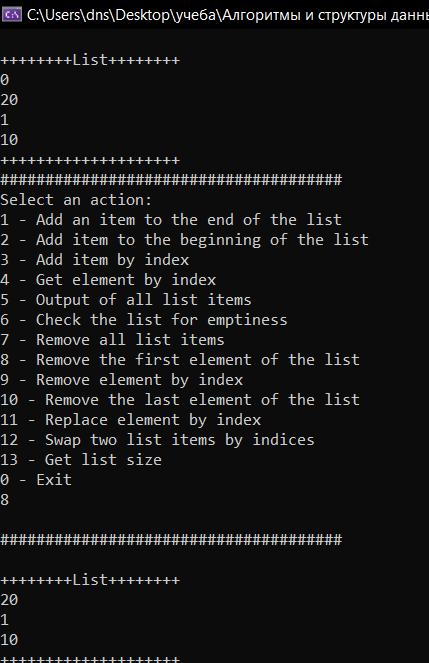
Пример 2 – Вставка в список по индексу.

****

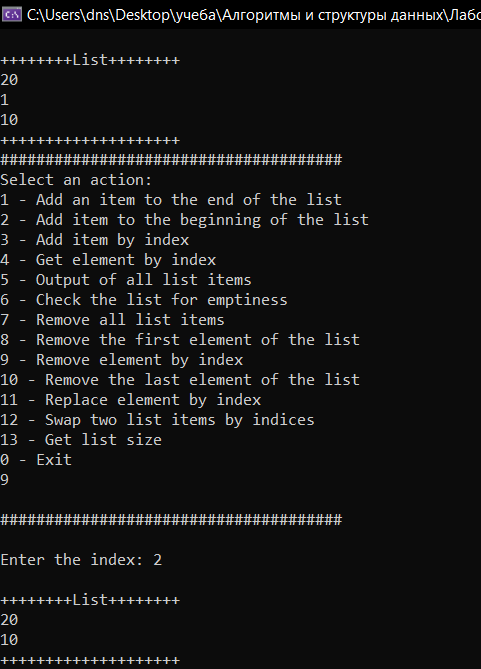
Пример 4 – Проверка списка на пустоту.



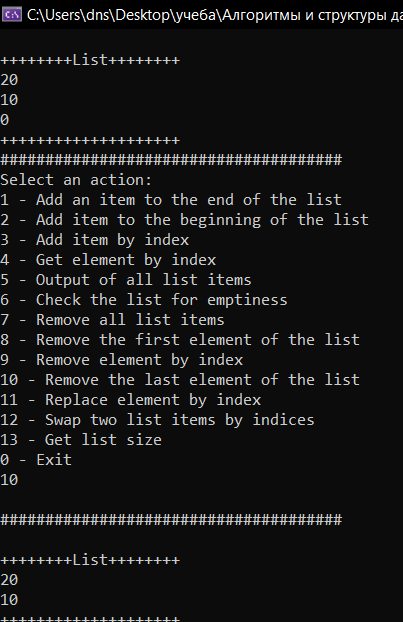
Пример 5 – Удаление первого элемента.



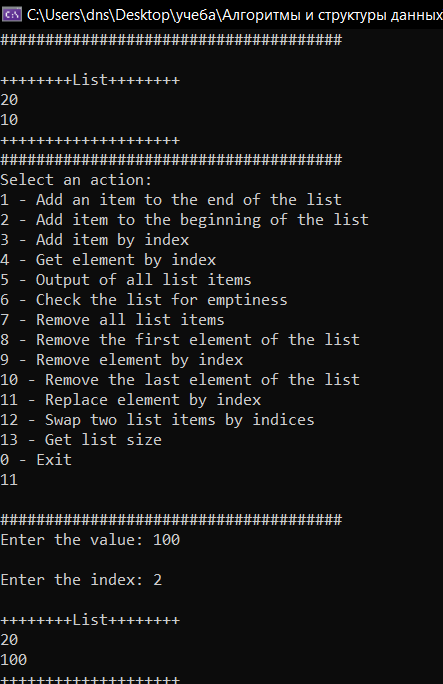
Пример 7 – Удаление элемента по индексу.



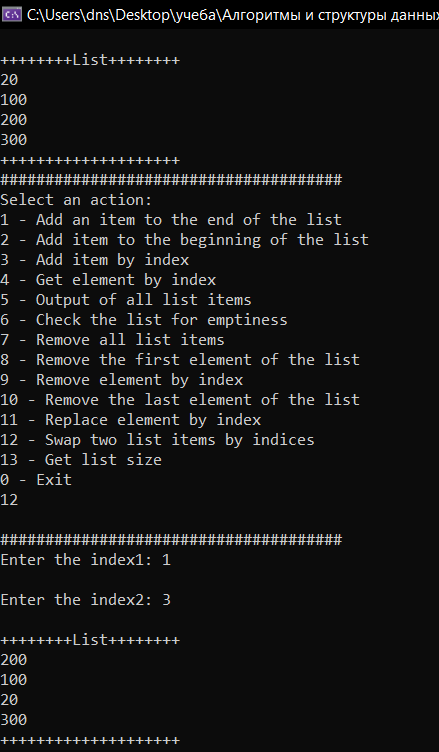
Пример 6 – Удаление последнего элемента.



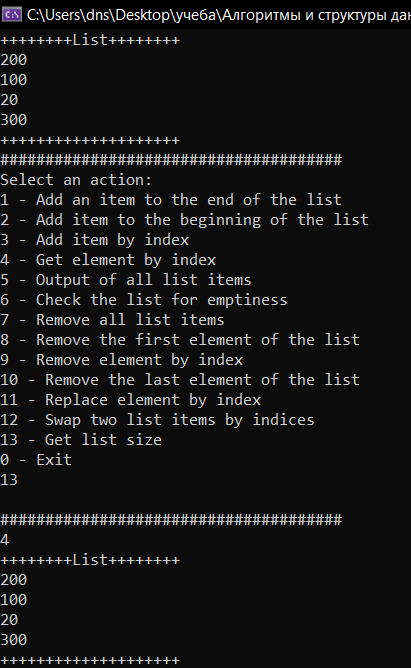
Пример 8 – Замена элемента по индексу.



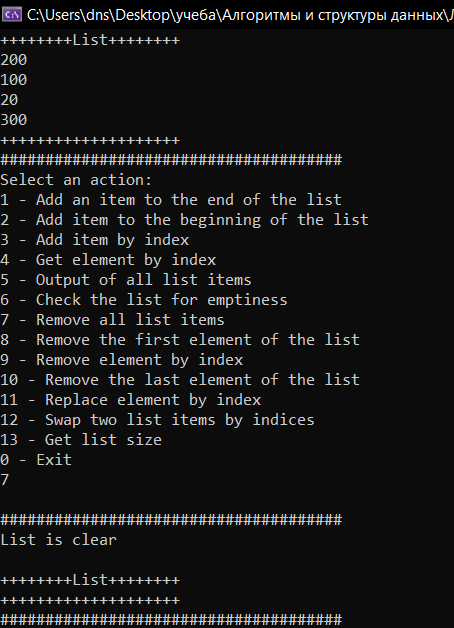
Пример 9 – Поменять два элемента местами.



Пример 11 – Вывод размера списка.



Пример 10 – Очищение списка.



**Листинг.**

<https://github.com/Elizaveta-git/Laba1.git>