Минобрнауки РФ

ФГБОУ ВО Тверской Государственный Технический Университет

Кафедра  “Программное Обеспечение”

Лабораторная работа N8

По дисциплине “Структуры и алгоритмы обработки данных”

Выполнил: студент группы Б.ПИН.РИС-22.06

Иванов Алексей Михайлович

Проверил: Мальков Александр Анатольевич

г. Тверь, 2023

Оглавление

[**Часть 1. Постановка задачи.** 3](#_Toc145406338)

[**Часть 2. Алгоритм решения.** 3](#_Toc145406339)

[**Часть 3. Оценка временной эффективности в терминах О-функций.** 4](#_Toc145406340)

[**Часть 4. Программный код.** 5](#_Toc145406341)

[**Часть 5. Тестирование и результаты работы.** 9](#_Toc145406342)

# **Часть 1. Постановка задачи.**

# 8. Дек.

# Методы класса "Дек":

# - добавление

# - чтение

# - проверка на пустоту.

# **Часть 2. Алгоритм решения.**

# Дек – расширение очереди, смесь очереди и стэка. Позволяет осуществлять вставку и получение элементов как с начала, так и с конца.

В моей реализации он сделан с помощью узлов, хранящих информацию и ссылки на следующий и предыдущий узел. В самом классе хранятся ссылки на первый и последний узел. Вообще говоря, хранить ссылку на последний узел необязательно, но тут решается проблема времени и памяти. Без нее необходимо будет для вставки в конец каждый раз проходить последовательно все узлы, что очень долго, при том что сама ссылка не весит почти ничего, так что этот вопрос я разрешил в пользу времени. Тогда для вставки и удаления нам необходимо лишь менять по 2 ссылки в конечном и первом узле.

В случае если дек пуст, то необходимо поставить и ссылку на начало и ссылку на конец на новый узел, а также назначить связи между ними.

Также существует поле Length. Его значение изменяется вместе с добавлением и удалением элементов. Оно разрешает вопрос времени и памяти для получения количества элементов в моей реализации в пользу времени. Можно было бы последовательно считать все элементы, но это требует их полного перебора, так что я решил, что эффективнее будет хранить одно int-поле в памяти весом в 16 бит, чем перебирать потенциальные десятки тысяч элементов. Также оно упрощает реализацию проверки на пустоту.

# **Часть 3. Оценка временной эффективности в терминах О-функций.**

Составим сводную таблицу эффективности в соответствии с выбранной реализацией. На пересечении строк с операциями и столбцов с типом очередей будет сложность операции.

|  |  |
| --- | --- |
|  | Сложность |
| Вставка в конец | O(1) |
| Вставка в начало | O(1) |
| Получение последнего | O(1) |
| Получение первого | O(1) |
| Удаление последнего | O(1) |
| Удаление первого | O(1) |
| Проверка на пустоту | O(1) |
| Подсчет элементов | O(1) |

# **Часть 4. Программный код.**

Итоговая библиотека классов имеет следующий вид:

/// <summary>

/// Класс, отображающий узел в двусвязном списке

/// </summary>

/// <typeparam name="T">Тип данных, хранимый в узле</typeparam>

public class MyDoubleNode<T> : MyNode<T>

{

/// <summary>

/// Ссылка на следующий узел

/// </summary>

protected new MyDoubleNode<T> Next;

/// <summary>

/// Ссылка на предыдущий узел

/// </summary>

private MyDoubleNode<T> Prev;

/// <summary>

/// Создает новый узел на основе данного значения

/// </summary>

/// <param name="value">Данное значение</param>

public MyDoubleNode(T value) : base(value) { }

/// <summary>

/// Создает пустой узел

/// </summary>

public MyDoubleNode() { }

/// <summary>

/// Возвращает ссылку на следующий узел в списке

/// </summary>

/// <returns>Ссылка на следующий узел в списке</returns>

public new MyDoubleNode<T> GetNext()

{

return this.Next;

}

/// <summary>

/// Назначает ссылку на следующий узел в списке

/// </summary>

/// <param name="newNode">Ссылка на следующий узел в списке</param>

public void SetNext(MyDoubleNode<T> newNode)

{

this.Next = newNode;

}

/// <summary>

/// Возвращает ссылку на предыдущий узел в списке

/// </summary>

/// <returns>Ссылка на предыдущий узел в списке</returns>

public MyDoubleNode<T> GetPrev()

{

return this.Prev;

}

/// <summary>

/// Назначает ссылку на предыдущий узел в списке

/// </summary>

/// <param name="newNode">Ссылка на предыдущий узел в списке</param>

public void SetPrev(MyDoubleNode<T> newNode)

{

this.Prev = newNode;

}

}

/// <summary>

/// Реализует структуру данных Дек

/// </summary>

/// <typeparam name="T">Тип данных, хранимый в экземпляре дека</typeparam>

public class MyDeque<T>

{

/// <summary>

/// Узел, являющийся первым элементом дека

/// </summary>

private MyDoubleNode<T> first;

/// <summary>

/// Узел, являющийся последним элементом дека

/// </summary>

private MyDoubleNode<T> last;

/// <summary>

/// Количество элементов в деке

/// </summary>

private int Length = 0;

/// <summary>

/// Создает новый пустой дек

/// </summary>

public MyDeque()

{

last = first;

}

/// <summary>

/// Создает новый дек с данным элементом

/// </summary>

/// <param name="first">Первый элемент дека</param>

public MyDeque(T first)

{

this.first = new MyDoubleNode<T>(first);

this.last = this.first;

Length++;

}

/// <summary>

/// Осуществляет вставку данного элемента в начало дека

/// </summary>

/// <param name="elem">Вставляемый элемент</param>

public void InsertFirst(T elem)

{

MyDoubleNode<T> node = new(elem);

if (Length == 0)

{

first = node;

last = first;

first.SetNext(last);

last.SetPrev(first);

Length++;

return;

}

node.SetNext(first);

first.SetPrev(node);

first = node;

Length++;

}

/// <summary>

/// Осуществляет вставку данного элемента в конец дека

/// </summary>

/// <param name="elem">Вставляемый элемент</param>

public void InsertLast(T elem)

{

MyDoubleNode<T> node = new(elem);

if (Length == 0)

{

last = node;

first = last;

first.SetNext(last);

last.SetPrev(first);

Length++;

return;

}

node.SetPrev(last);

last.SetNext(node);

last = node;

Length++;

}

/// <summary>

/// Возвращает первый элемент дека

/// </summary>

/// <returns>Первый элемент дека</returns>

public T GetFirst()

{

return first.GetValue();

}

/// <summary>

/// Возвращает последний элемент дека

/// </summary>

/// <returns>Последний элемент дека</returns>

public T GetLast()

{

return last.GetValue();

}

/// <summary>

/// Возвращает первый элемент дека, удаляя его

/// </summary>

/// <returns>Первый элемент дека</returns>

public T PopFirst()

{

MyDoubleNode<T> node = first;

first = first.GetNext();

first?.SetPrev(null);

Length--;

return node.GetValue();

}

/// <summary>

/// Возвращает последний элемент дека, удаляя его

/// </summary>

/// <returns>Последний элемент дека</returns>

public T PopLast()

{

MyDoubleNode<T> node = last;

last = last.GetPrev();

last?.SetNext(null);

Length--;

return node.GetValue();

}

/// <summary>

/// Возвращает длину дека,

/// т.е. количество элементов

/// </summary>

/// <returns>Длину дека,

/// т.е. количество элементов</returns>

public int GetLength()

{

return this.Length;

}

/// <summary>

/// Проверяет, пуст ли данный дек

/// </summary>

/// <returns>

/// True: дек пуст и не содержит элементов

/// False: все остальные случаи

/// </returns>

public bool IsEmpty()

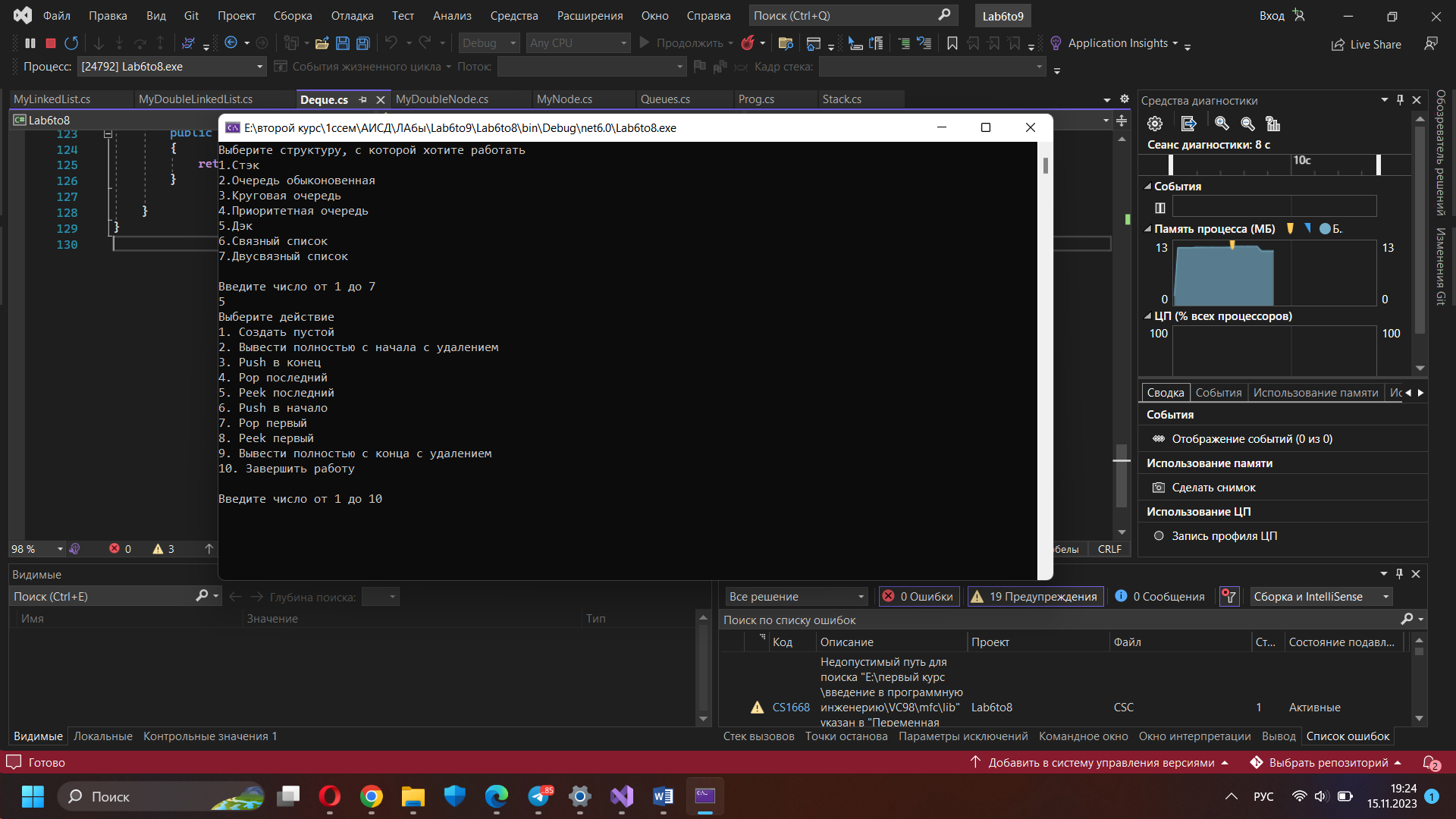
{

return Length == 0;

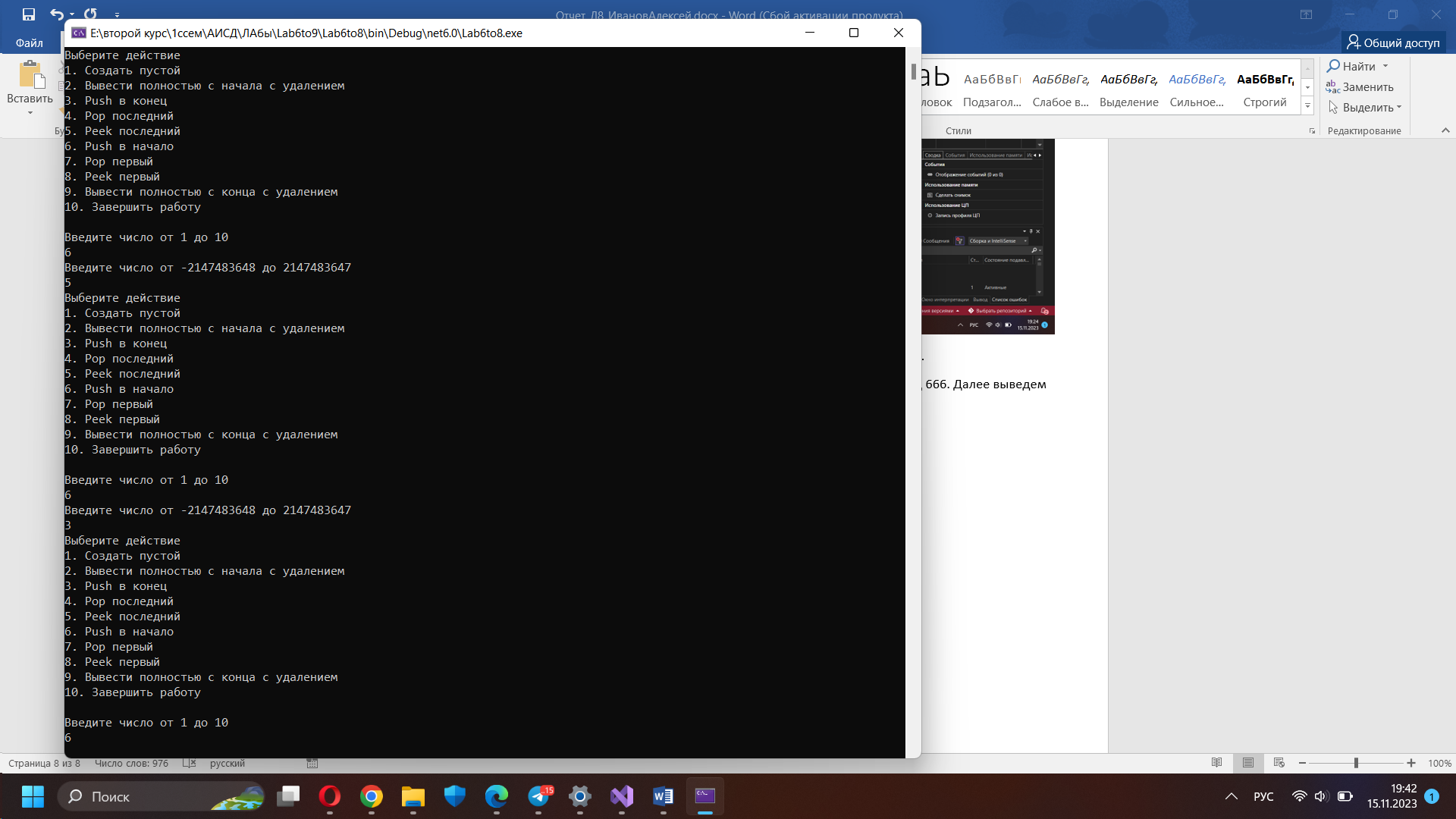
}

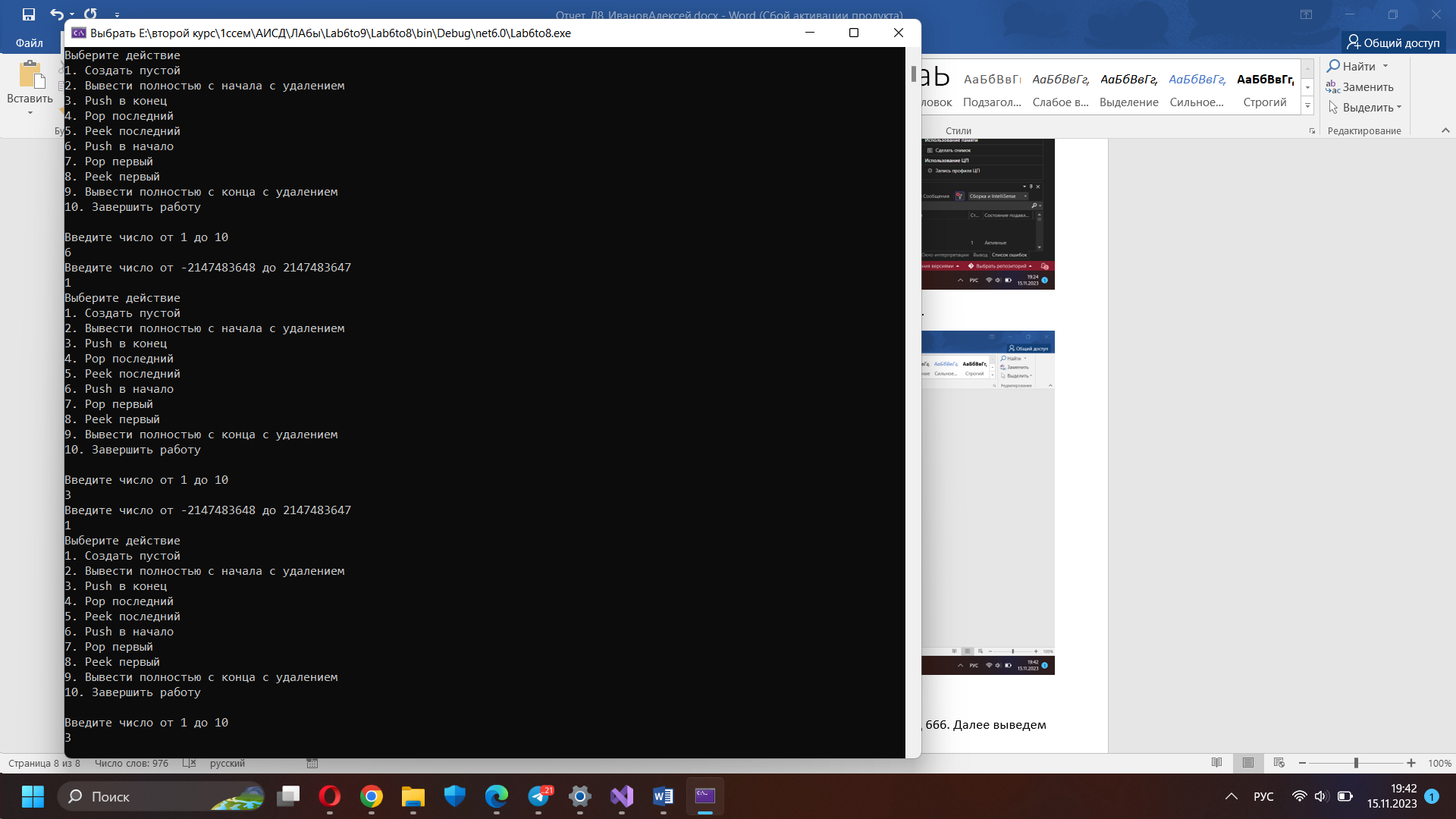
**Часть 5. Тестирование и результаты работы.**

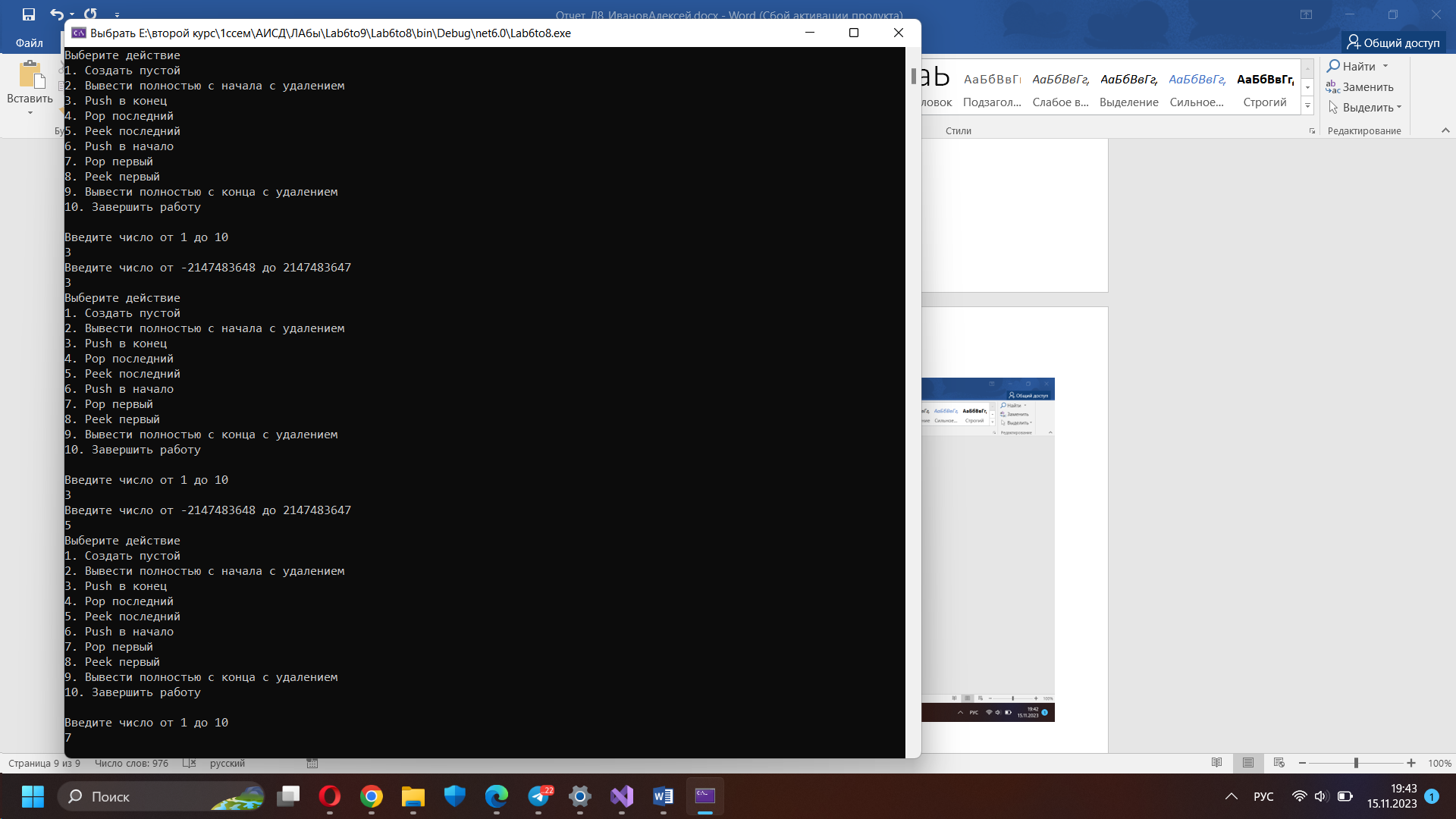
Для тестирования было разработано консольное приложение, проверяющее корректность алгоритмов.



Введем в начало дека последовательно элементы 5,3,1, а далее в конец – 1, 3, 5.







Далее посмотрим первый и последний элементы с удалением и введем в конец 666. Далее выведем дек с конца.

