Минобрнауки РФ

ФГБОУ ВО Тверской Государственный Технический Университет

Кафедра  “Программное Обеспечение”

Лабораторная работа N9

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.**

# Алгоритмы работы со списками.

# Методы класса "Список":

# - добавление

# - удаление

# - копирование части списка

# - слияние двух списков (2 варианта: с и без создания нового списка)

# - проверка на пустоту.

# **Часть 2. Алгоритм решения.**

# Связный список – динамическая структура, состоящая из узлов, каждый из который ссылается на следующий и хранит информацию, а также хранящая ссылку на первый из списка узлов. Для вставки в начало нам необходимо лишь назначить текущий первый элемент следующим для нового узла, а далее ссылку на первый узел переместить на него. Для вставки в конец нам необходимо пройти последовательно все узлы и для последнего следующим поставить новый. Для удаления необходимо лишь разорвать связь между узлами и переместить ссылку. Для проверки на пустоту необходимо лишь проверить, не ведет ли в null ссылка на первый элемент.

Для слияния списков без создания нового необходимо лишь последнему элементу первого назначить следующим первый элемент второго сливаемого списка. Если необходимо слить два списка с созданием нового, то можно последовательно копировать элементы первого, связывая их между собой, а далее сделать то же самое с элементами второго, связав его первый элемент с последним элементом первого списка.

Для копирования части списка необходимо, последовательно перебирая элементы, дойти до элемента с нужным номером, а далее их последовательно копировать и вставлять, пока это необходимо.

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

Составим сводную таблицу эффективности операций.

|  |  |
| --- | --- |
|  | Сложность (n длина списка) |
| Вставка в начало | O(1) |
| Вставка в конец | O(n) |
| Получение первого элемента | O(1) |
| Получение последнего элемента | O(n) |
| Проверка на пустоту | O(1) |
| Слияние без создания нового списка | O(n) , от длины второго списка не зависит |
| Слияние с созданием нового списка | O(n + m), где m – длина второго списка |

# **Часть 4. Программный код.**

Итоговая библиотека классов имеет следующий вид:

/// <summary>

/// Класс, отображающий узел в связном списке

/// </summary>

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

public class MyNode<T>

{

/// <summary>

/// Значение, хранимое в узле

/// </summary>

protected T Value;

/// <summary>

/// Ссылка на следующий узел

/// </summary>

protected MyNode<T> Next;

/// <summary>

/// Создает новый узел с данным значением

/// </summary>

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

public MyNode(T value)

{

Value = value;

Next = null;

}

/// <summary>

/// Создает пустой узел

/// </summary>

public MyNode()

{

Value = default(T);

Next = null;

}

/// <summary>

/// Возвращает значение в текущем узле

/// </summary>

/// <returns>Значение в текущем узле</returns>

public T GetValue()

{

return this.Value;

}

/// <summary>

/// Возвращает ссылку на следующий элемент

/// </summary>

/// <returns>Ссылка на следующий элемент</returns>

public MyNode<T> GetNext()

{

return this.Next;

}

/// <summary>

/// Назначает ссылку на следующий элемент

/// </summary>

/// <returns>Ссылка на следующий элемент</returns>

public void SetNext(MyNode<T> newNode)

{

this.Next = newNode;

}

/// <summary>

/// Реализует структуру данных стэк

/// </summary>

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

/// <summary>

/// Реализует структуру данных связный список

/// </summary>

/// <typeparam name="T">Тип данных в связнном списке</typeparam>

public class MyLinkedList<T>

{

/// <summary>

/// Ссылка на первый элемент в списке

/// </summary>

public MyNode<T> First { get; set; }

/// <summary>

/// Создает новый пустой двусвязный список

/// </summary>

public MyLinkedList()

{

}

/// <summary>

/// Создает новый связный список с единственным элементом

/// </summary>

/// <param name="val">Элемент в связном списке</param>

public MyLinkedList(T val)

{

First = new MyNode<T>(val);

}

/// <summary>

/// Возвращает первый узел связного списка

/// </summary>

/// <returns>Первый узел связного списка</returns>

public MyNode<T> GetFirstNode()

{

return this.First;

}

/// <summary>

/// Вставляет узел в начало списка

/// </summary>

/// <param name="newNode">Вставляемый узел</param>

protected void InsertFirstNode(MyNode<T> newNode)

{

newNode.SetNext(First);

First = newNode;

}

/// <summary>

/// Вставляет узел в конец списка

/// </summary>

/// <param name="newNode">Вставляемый узел</param>

protected void InsertLastNode(MyNode<T> newNode)

{

if (First == null)

{

First = newNode;

return;

}

MyNode<T> cur = First;

while(cur.GetNext() != null)

{

cur = cur.GetNext();

}

cur.SetNext(newNode);

}

/// <summary>

/// Вставляет элемент в начало списка

/// </summary>

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

public void InsertFirst(T val)

{

this.InsertFirstNode(new MyNode<T>(val));

}

/// <summary>

/// Вставляет элемент в конец списка

/// </summary>

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

public void InsertLast(T val)

{

this.InsertLastNode(new MyNode<T>(val));

}

/// <summary>

/// Удаляет элемент из списка по значению

/// </summary>

/// <param name="val">Удаляемый элемент</param>

public void DeleteByVal(T val)

{

MyNode<T> prev = First;

while (prev.GetNext() != null)

{

if (prev.GetNext().GetValue().Equals(val))

{

prev.SetNext(prev.GetNext().GetNext());

break;

}

prev = prev.GetNext();

}

}

/// <summary>

/// Удаляет первый элемент из списка

/// </summary>

public void DeleteFirst()

{

First = First.GetNext();

}

/// <summary>

/// Копирует отрезок из связного списка в новый список

/// </summary>

/// <param name="from">Порядковый номер первого копируемого элемента</param>

/// <param name="to">Порядковый номер предпоследнего копируемого элемента</param>

public MyLinkedList<T> Copy(int from, int to)

{

MyLinkedList<T> list = new MyLinkedList<T>();

int i = 0;

MyNode<T> cur = First;

for (i = 0; i < from; i++)

cur = cur.GetNext();

while (i < to)

{

list.InsertFirst(cur.GetValue());

i++;

}

return list;

}

/// <summary>

/// Объединяет текущий список с данным, присоединяя его в конец

/// </summary>

/// <param name="ll">Присоединяемый в конец связный список</param>

public void UnifyWith(MyLinkedList<T> ll)

{

MyNode<T> myNode = First;

while (myNode.GetNext() != null)

myNode = myNode.GetNext();

myNode.SetNext(ll.GetFirstNode());

}

/// <summary>

/// Возвращает новый связный список, соединяющий в себе

/// текущий и данный

/// </summary>

/// <param name="ll">Присоединяемый в конец связный список</param>

/// <returns>Новый связный список, соединяющий в себе

/// текущий и данный</returns>

public MyLinkedList<T> UnifiedWith(MyLinkedList<T> ll)

{

MyNode<T> cur;

MyLinkedList<T> res = new();

MyStack<T> Stack = new();

cur = First;

while (cur != null)

{

Stack.InsertElem(cur.GetValue());

cur = cur.GetNext();

}

cur = ll.GetFirstNode();

while (cur != null)

{

Stack.InsertElem(cur.GetValue());

cur = cur.GetNext();

}

while (!Stack.IsEmpty())

{

res.InsertFirst(Stack.PopElem());

}

return res;

}

/// <summary>

/// Проверяет текущий список на пустоту

/// </summary>

/// <returns>

/// True: текущий список пуст

/// False: все иные случаи

/// </returns>

public bool IsEmpty()

{

return this.First == null;

}

/// <summary>

/// Возвращает список, состоящий из элементов

/// текущего связного списка

/// </summary>

/// <returns>Список, состоящий из элементов

/// текущего связного списка</returns>

public List<T> GetValues()

{

List<T> res = new();

MyNode<T> myNode = First;

while (myNode != null)

{

res.Add(myNode.GetValue());

myNode = myNode.GetNext();

}

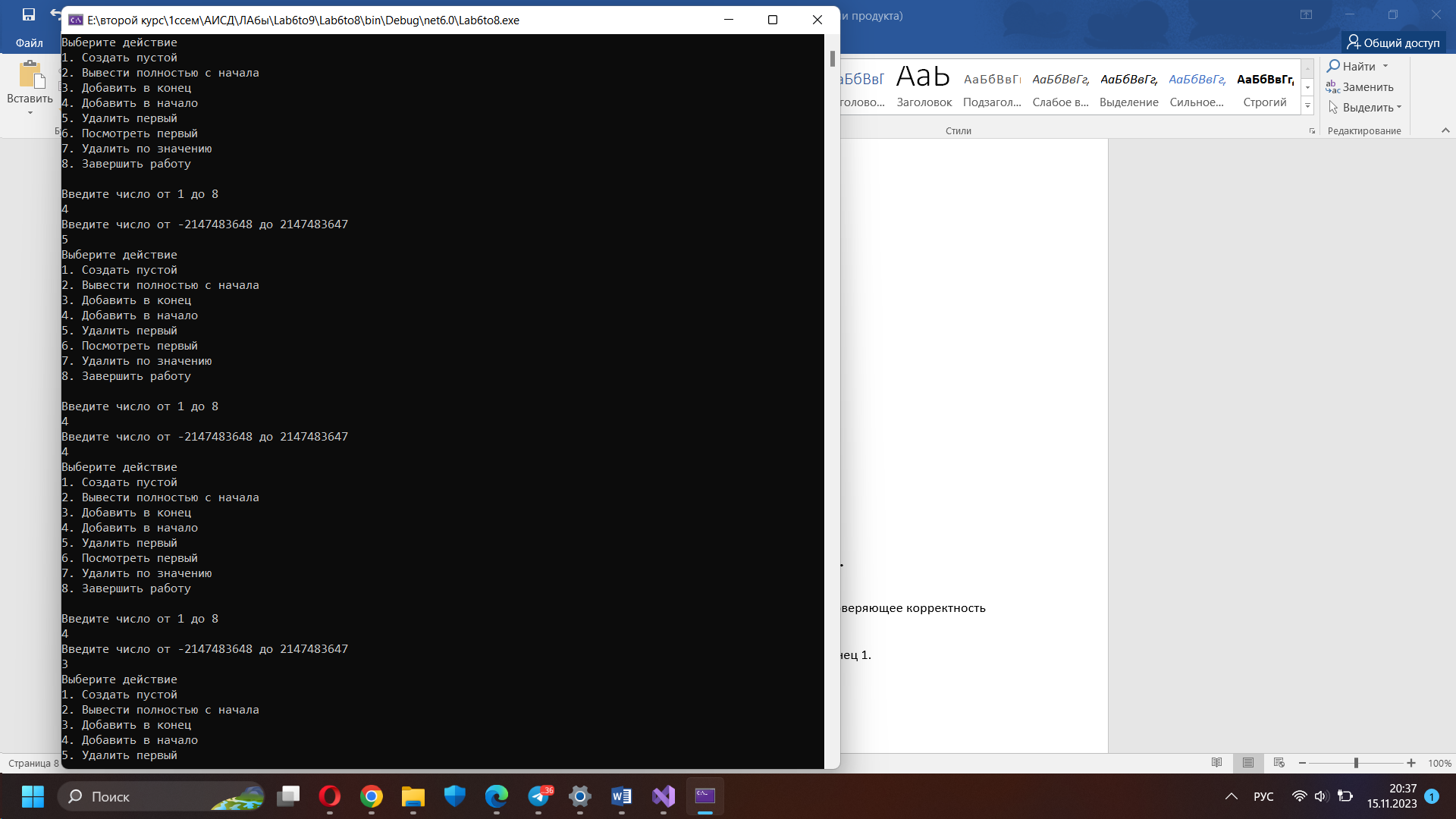
return res;

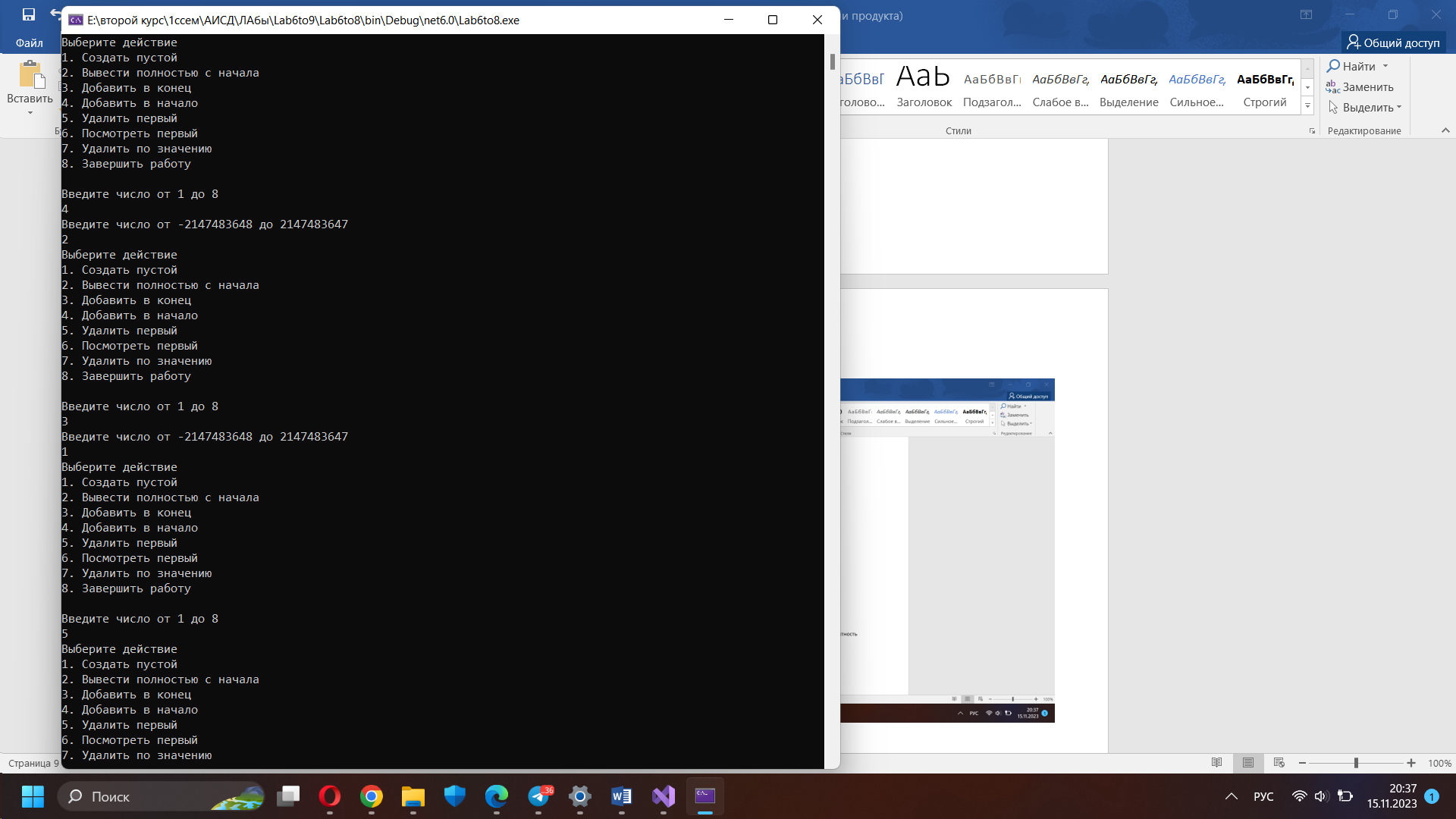
}

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

Для тестирования было разработано консольное приложение, проверяющее корректность алгоритмов.

Вставим в начало последовательно элементы 5, 4, 3, 2. Далее в конец 1.





Далее удалим первые 2, и выведем список полностью.

