|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
|  |
| Ханжин Александр Евгеньевич  **ЛАБОРАТОРНАЯ РАБОТА №12**  студента образовательной программы «Разработка информационных систем для бизнеса» по направлению подготовки *09.03.04 Программная инженерия*   |  |  | | --- | --- | |  | Старший преподаватель кафедры ИТБ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Марквирер В.Д. |   Пермь, 2024 |

# Оглавление

[Оглавление 2](#_Toc166604133)

[Задача 1 3](#_Toc166604134)

[1.1 Постановка задачи 3](#_Toc166604135)

[1.2 Диаграмма классов 3](#_Toc166604136)

[1.3 Алгоритм (блок-схема) 4](#_Toc166604137)

[1.4 Программа (листинг) 7](#_Toc166604138)

[1.5 Код тестов (листинг) 7](#_Toc166604139)

[1.6 Покрытие тестами кода 7](#_Toc166604140)

[Задача 2 8](#_Toc166604141)

[2.1 Постановка задачи 8](#_Toc166604142)

[2.2 Диаграмма классов 8](#_Toc166604143)

[2.3 Алгоритм (блок-схема) 9](#_Toc166604144)

[2.4 Программа (листинг) 11](#_Toc166604145)

[2.5 Код тестов (листинг) 11](#_Toc166604146)

[2.6 Покрытие тестами кода 11](#_Toc166604147)

[Задача 3 12](#_Toc166604148)

[3.1 Постановка задачи 12](#_Toc166604149)

[3.2 Диаграмма классов 12](#_Toc166604150)

[3.3 Алгоритм (блок-схема) 13](#_Toc166604151)

[3.4 Программа (листинг) 15](#_Toc166604152)

[3.5 Код тестов (листинг) 15](#_Toc166604153)

[3.6 Покрытие тестами кода 16](#_Toc166604154)

[Задача 4 17](#_Toc166604155)

[4.1 Постановка задачи 17](#_Toc166604156)

[4.2 Диаграмма классов 17](#_Toc166604157)

[4.3 Программа (листинг) 18](#_Toc166604158)

[4.4 Код тестов (листинг) 18](#_Toc166604159)

[4.5 Покрытие тестами кода 18](#_Toc166604160)

# Задача 1

## Постановка задачи

В первой части лабораторной работы №12 мы знакомимся с реализацией двунаправленных списков в C#. Двунаправленный список представляет собой структуру данных, в которой каждый элемент содержит ссылки на предыдущий и следующий элементы. Это позволяет эффективно добавлять, удалять и обращаться к элементам списка как с начала, так и с конца.

В данной работе мы создадим двунаправленный список, заполним его объектами из иерархии классов, выполним обработку элементов списка и выполним глубокое клонирование для правильного управления памятью. Работа с двунаправленными списками позволит нам лучше понять принципы работы со структурами данных и их применение в реальных задачах программирования.

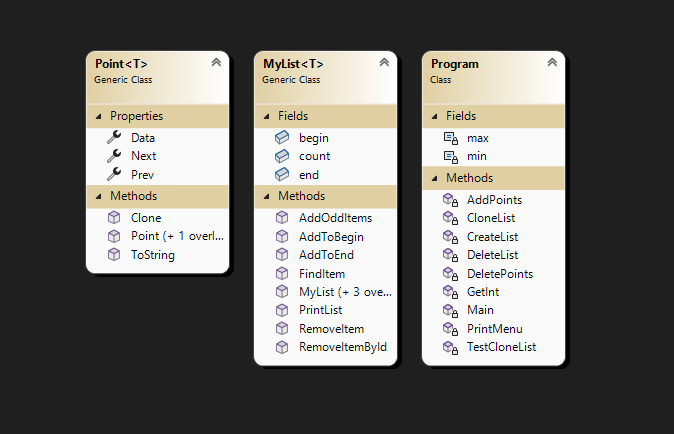
Программа должна выполнять следующие функции:

1. Формировать двунаправленный список, в информационные поля записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученный список.
3. Добавлять с список элементы с номерами 1, 3, 5 и тд.
4. Удалять все элементы начиная с элемента с заданным информационным полем.
5. Выполнять глубокое клонирование списка, т.е. под объекты, хранящиеся в коллекции должна быть выделена память.
6. Удалять список из памяти.

## Диаграмма классов

Диаграммы классов являются важным инструментом в объектно-ориентированном программировании для визуализации структуры классов и их взаимосвязей в программе или системе. Они представляют собой графическое представление классов, интерфейсов, атрибутов и методов, а также отношений между ними.

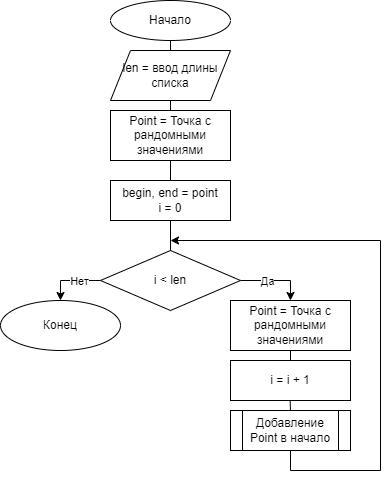
Для более наглядного представления добавим созданные классы в диаграмму классов. Класс Point<T> элемент моего списка, у которого есть ссылки на предыдущий и следующий элемент Point<T>. Класс MyList<T> реализует работу списка. В классе Program реализуется основная работа консольного меню.



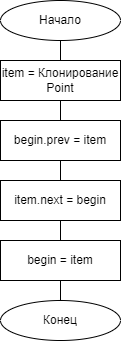
***Рис. 1.2.1. Диаграмма классов***

## Алгоритм (блок-схема)

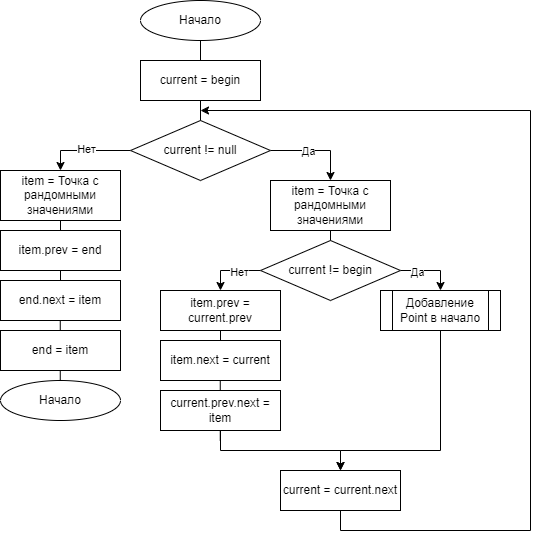
Для заданий по варианту реализованы блок-схемы, которые упрощают понимание реализации алгоритмов.



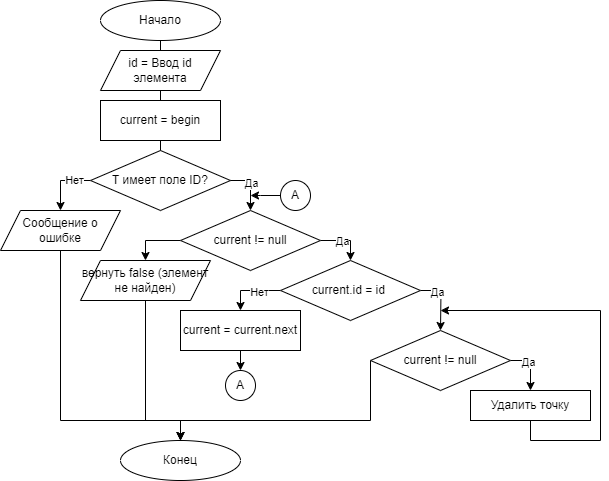
***Рис. 1.3.1. Блок-схема конструктора списка.***

******

***Рис. 1.3.2. Блок-схема добавление элемента в начало списка.***

******

***Рис. 1.3.3. Блок-схема добавления элементов с номерами 1, 3, 5 и тд.***

******

***Рис. 1.3.4. Блок-схема операции удаления всех точек начиная с заданной информационным полем.***

## Программа (листинг)

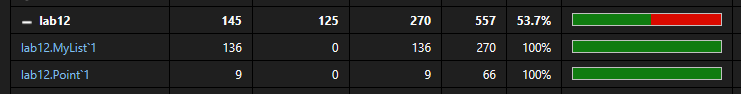
Полный код лабораторной работы можно посмотреть по ссылке <https://github.com/Asklit/lab12>. С частью с двунаправленными списками можно ознакомиться по ссылке <https://github.com/Asklit/lab12/tree/master/lab12>.

## Код тестов (листинг)

Полный код unit тестов списков можно посмотреть по ссылке <https://github.com/Asklit/lab12/tree/master/ListTest>.

## Покрытие тестами кода

Для покрытия кода тестами я исключил все сторонние функции интерфейса, за счет чего получилось достичь 100% покрытия тестами кода.



# Задача 2

## Постановка задачи

Во второй части лабораторной работы №12 мы знакомимся с реализацией хештаблиц в C#. Хеш-таблица является одной из ключевых структур данных в программировании, обеспечивающей эффективное хранение и быстрый доступ к данным. Каждому элементу в хеш-таблице соответствует уникальный ключ, по которому можно быстро осуществлять доступ к соответствующему значению.

Реализация хеш-таблицы в C# позволяет эффективно выполнять операции вставки, удаления и поиска элементов за константное время в среднем случае. При этом хеш-таблицы позволяют хранить данные в отсортированном порядке, что делает их удобным инструментом для работы с большим объемом информации.

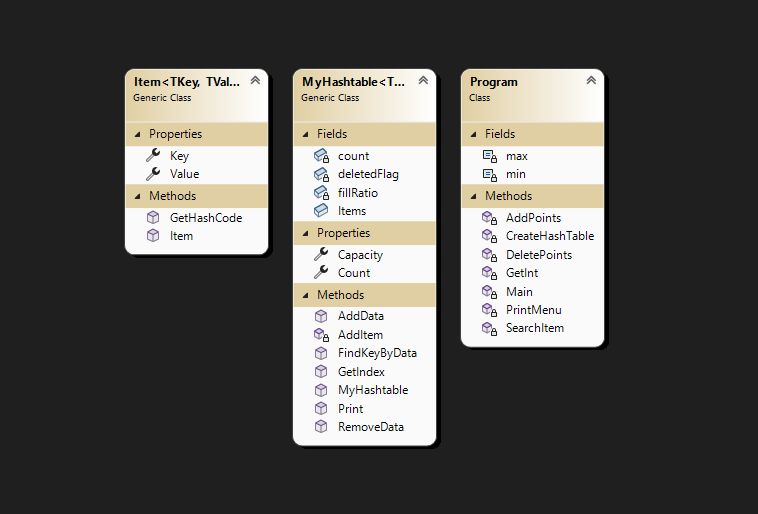
При использовании хеш-таблиц в C# важно правильно выбирать хеш-функцию для равномерного распределения ключей по ячейкам таблицы, что поможет избежать коллизий и обеспечить высокую производительность структуры данных.

Программа должна выполнять следующие функции:

1. Формировать хеш-таблицу, в информационные поля записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученную хеш-таблицу.
3. Выполнять поиск по информационным полям.
4. Удалять найденный элемент по информационному полю.
5. Добавлять в таблицу рандомно сформированные значения.

## Диаграмма классов

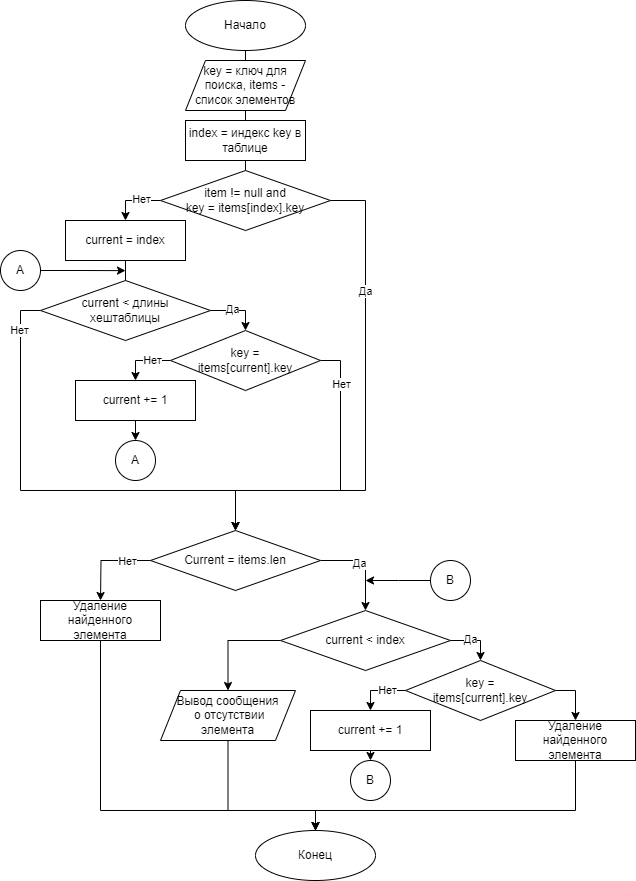
Для более наглядного представления добавим созданные классы в диаграмму классов. Класс Item<Tkey, TValue> элемент моей хештаблицы реализующей работу обобщения TKey TValue, у которого есть функция вычисления хеша. Класс MyHashtable<TKey, TValue> реализует работу хештаблицы. В классе Program реализуется основная работа консольного меню.



***Рис. 2.2.1. Диаграмма классов***

## Алгоритм (блок-схема)

Для заданий по варианту реализованы блок-схемы, которые упрощают понимание реализации алгоритмов.



***Рис. 2.3.1. Блок-схема удаления элемента из хештаблицы***

## Программа (листинг)

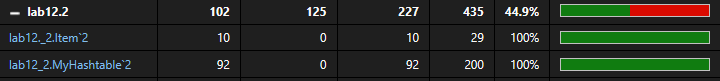
Полный код лабораторной работы можно посмотреть по ссылке <https://github.com/Asklit/lab12>. С частью с двунаправленными списками можно ознакомиться по ссылке <https://github.com/Asklit/lab12/tree/master/lab12.1>.

## Код тестов (листинг)

Полный код unit тестов списков можно посмотреть по ссылке https://github.com/Asklit/lab12/tree/master/HashTableTest.

## Покрытие тестами кода

Для покрытия кода тестами я исключил все сторонние функции интерфейса, за счет чего получилось достичь 100% покрытия тестами кода.



# Задача 3

## Постановка задачи

В третьей части лабораторной работы №12 мы знакомимся с реализацией идеально сбалансированных деревьев и деревьев поиска в C#. Идеально сбалансированное дерево представляет собой особую структуру данных, где каждый узел имеет двух потомков, и разница в высоте поддеревьев для каждого узла не превышает одного уровня. Это обеспечивает оптимальное распределение данных и обеспечивает эффективность операций поиска, вставки и удаления элементов.

Бинарные деревья являются одной из основных структур данных в информатике и программировании. В C# реализация бинарных деревьев предоставляет эффективный способ хранения и организации данных. Бинарное дерево состоит из узлов, каждый из которых имеет не более двух потомков: левого и правого. Эта структура данных широко применяется для решения различных задач, таких как поиск, сортировка, обход дерева и другие операции.

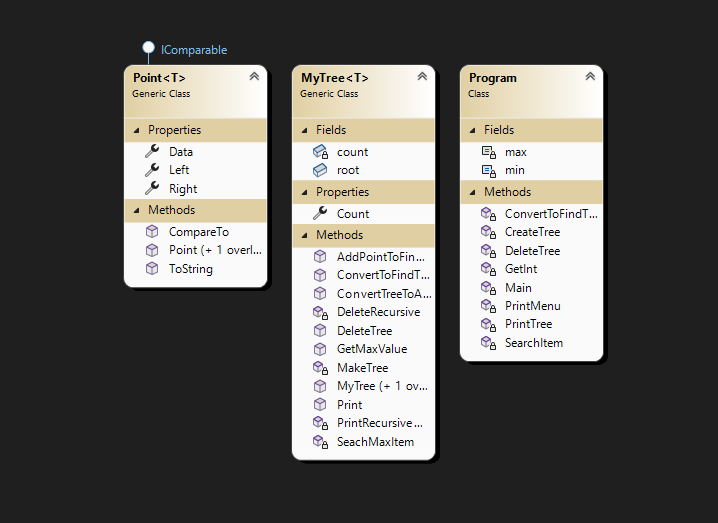
Реализация бинарных деревьев в C# может быть осуществлена с использованием классов и структур, представляющих узлы дерева, а также методов для добавления и обхода узлов. Правильная реализация бинарного дерева позволяет эффективно выполнять операции вставки, поиска элементов, что делает его мощным инструментом в различных прикладных задачах.

Программа должна выполнять следующие функции:

1. Формировать идеально сбалансированное дерево, в информационные поля записать объекты из иерархии классов лабораторной работы №10.
2. Распечатывать ИДС.
3. Находить максимальный элемент в дереве (элемент с максимальным id).
4. Преобразовывать ИДС в дерево поиска.
5. Распечатывать дерево поиска.
6. Удалять ИДС из памяти.
7. Удалять дерево поиска из памяти.

## Диаграмма классов

Для более наглядного представления добавим созданные классы в диаграмму классов. Класс Point<T> элемент моего дерева в котором хранятся ссылки на левое и правое поддерево класса Point<T>. Помимо этого класс Point<T> реализует работу интерфейса IComparable для сравнения вершин дерева между собой. Класс MyTree<T> реализует работу дерева, у которого задана вершина. В классе Program реализуется основная работа консольного меню.



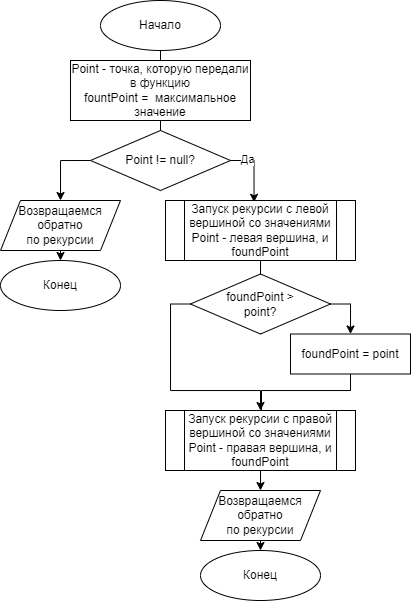
***Рис. 3.2.1. Диаграмма классов***

## Алгоритм (блок-схема)

Для заданий по варианту реализованы блок-схемы, которые упрощают понимание реализации алгоритмов.

******

***Рис. 3.3.1. Блок-схема функции поиска максимального элемента в дереве***

******

***Рис. 3.3.2. Блок-схема вспомогательной рекурсивной функции поиска максимального элемента***

## Программа (листинг)

Полный код лабораторной работы можно посмотреть по ссылке <https://github.com/Asklit/lab12>. С частью с двунаправленными списками можно ознакомиться по ссылке <https://github.com/Asklit/lab12/tree/master/lab12.3>.

## Код тестов (листинг)

Полный код unit тестов списков можно посмотреть по ссылке <https://github.com/Asklit/lab12/tree/master/TreeTest>.

## Покрытие тестами кода

Для покрытия кода тестами я исключил все сторонние функции интерфейса, за счет чего получилось достичь 100% покрытия тестами кода.



# Задача 4

## Постановка задачи

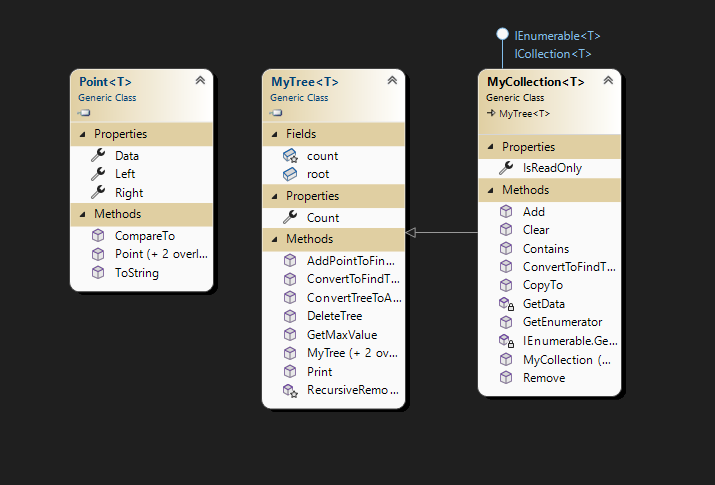
В четвертой части лабораторной работы №12 мы знакомимся с реализацией интерфейсов IEnumerable и ICollection. В качестве структуры данных, для которой я буду реализовывать эти интерфейсы я выбрал дерево поиска.

1. Изучение интерфейсов IEnumerable и ICollection в контексте C# и их обобщенной реализации.
2. Проектирование классов и методов, необходимых для реализации интерфейсов IEnumerable и ICollection для бинарного дерева.
3. Написание кода с учетом обобщенности и безопасности типов для обеспечения гибкости использования интерфейсов.
4. Тестирование разработанных классов и методов на корректность работы с элементами бинарного дерева.

## Диаграмма классов

Для более наглядного представления добавим созданные классы в диаграмму классов. Класс Point<T> элемент моего дерева в котором хранятся ссылки на левое и правое поддерево класса Point<T>. Помимо этого, класс Point<T> реализует работу интерфейса IComparable для сравнения вершин дерева между собой. Класс MyTree<T> реализует работу дерева, у которого задана вершина. В классе Program реализуется основная работа консольного меню.

Класс реализующий работу IEnumerable и ICollection наследуется от класса MyTree и переопределяет нужные методы и конструкторы.



***Рис. 4.2.1. Диаграмма классов***

## Программа (листинг)

Полный код лабораторной работы можно посмотреть по ссылке <https://github.com/Asklit/lab12>. С частью с двунаправленными списками можно ознакомиться по ссылке https://github.com/Asklit/lab12/tree/master/lab12.4.

## Код тестов (листинг)

Полный код unit тестов списков можно посмотреть по ссылке https://github.com/Asklit/lab12/tree/master/MyCollectionTest.

## Покрытие тестами кода

При тестированнии написанной коллекции я протестировал все написанные мной функции кроме стандартных по типу IEnumerable.GetEnumerator(), которую я не реализовывал.

