МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по лабораторной работе №3

Выполнил:

Студент группы ПрИ-202 Приходько Даниил Александрович

Студент группы ПрИ-202 Саламатин Алексей Юрьевич

Студент группы ПрИ-202 Скоробогатов Максим Дмитриевич

Принял:

Преподаватель ИИТ Николаев Иван Евгеньевич

Отчет защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата оценка

Челябинск 2024 г.

Содержание

Введение..................................................................................................................3

LinkedList……….....................................................................................................6

I. Стек......................................................................................................................11

II.Очередь...............................................................................................................25

III.Применения...................................................................................................... 37

IV. Задачи...............................................................................................................56

Анализ…………………………………………………………………………….68

Введение

Цель работы: реализовать динамические структуры в коде программы

Задачи:

1. Реализовать структуру данных – Стек

1.1 Реализовать стек на основе своего связного списка, умеющий выполнять следующие операции: Push(elem), Pop(), Top(), isEmpty(), Print().

1.2 Реализовать считывание команд из файла в котором записаны числа от 1 до 5 через пробел. Каждому числу соответствует своя операция: 1 - Push(elem), 2 - Pop(), 3 - Top(), 4 - isEmpty(), 5 - Print(). Для операции push после единицы через запятую указывается значение помещаемого элемента (это может быть число либо слово). Вывести все операции последовательно пользователю на экран

1.3 Создать тестовый файл с различной длиной последовательности команд а также их сочетанием, измерить время их выполнения и отобразить на графике.

1.4 Реализовать алгоритм вычисления выражения, записанного в постфиксной записи (используя постфиксные вычисления). Постфиксная запись считывается из файла. В выражение входят только числа и знаки операций (+, -, \*, :, ^, ln, cos, sin, sqrt, «)».

После реализации алгоритма вычисления постфиксной записи, необходимо произвести расчет оценки сложности алгоритма.

2. Реализовать структуру данных – Очередь

2.1 Реализовать Очередь с помощью списка и с помощью стандартного класса Queue. Должны поддерживаться следующие операции работы с очередью: вставка(Enqueue)/удаление(Dequeue)) элемента, проверка на пустоту(IsEmpty), печать(Print), вывод первого элемента(Top или Peek).

2.2 Реализовать считывание команд из файла в котором записаны числа от 1 до 5 через пробел. Каждому числу соответствует своя операция: 1 - вставка, 2 - удаление, 3 – просмотр начала очереди, 4 – проверка на пустоту, 5 - печать. Для операции добавления в очередь после единицы через запятую указывается значение помещаемого элемента (это может быть число либо слово). Результат выполнения каждой операции выводится на экран.

2.3 Создать тестовый файл с различной длиной последовательности команд, а также их сочетанием, измерить время их выполнения и отобразить на графике. Исследовать последовательности на максимально время затратные и наоборот самые быстро-выполняемые.

3. Найти где и как применяются динамические структуры в решении задач. Для каждого члена команды найти по 1 примеру применения структур данных – Список, Стек, Очередь, Дерево, в целях решения какой – либо задачи. Написать программную реализацию.

4. Реализовать алгоритмы работы со своими связными списками.

4.1 Написать функцию, которая переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке.

4.2 Написать функцию, которая переносит в начало (в конец) непустого списка L его последний (первый) элемент.

4.3 Написать функцию, которая определяет количество различных элементов списка, содержащего целые числа.

4.4 Написать функцию, которая удаляет из списка L неуникальные элементы.

4.5 Написать функцию вставки списка самого в себя вслед за первым вхождением числа х.

4.6 Написать функцию, которая вставляет в непустой список L, элементы которого упорядочены по не убыванию, новый элемент Е так, чтобы сохранилась упорядоченность.

4.7 Написать функцию, которая удаляет из списка L все элементы Е, если таковые имеются.

4.8 Написать функцию, которая вставляет в список L новый элемент F перед первым вхождением элемента Е, если Е входит в L.

4.9 Написать функцию, которая дописывает к списку L список E. Оба списка содержат целые числа. В основной программе считать их из файла.

4.10 Написать функцию, которая разбивает список целых чисел на два списка по первому вхождению заданного числа. Если этого числа в списке нет, второй список будет пустым, а первый не изменится.

4.11 Написать функцию, которая удваивает список, т.е. приписывает в конец списка себя самого.

4.12 Написать функцию, которая меняет местами два элемента списка, заданные пользователем

LinkedList

По условию лабораторной работы, нужно было реализовать некоторые динамические структуры с помощью своего связного списка.

Нашей командой специально для этого был реализован класс LinkedList (рисунки 1.1.1 – 1.1.5). В этой реализации каждый элемент списка представляется объектом типа Node<T> (рис.1.2), который содержит данные и ссылку на следующий элемент списка. В связном списке элементы не хранятся в массиве, как в обычном списке, а каждый элемент "ссылается" на следующий через указатели, что позволяет эффективно изменять структуру (добавлять и удалять элементы) без необходимости перемещать другие элементы.

Методы класса LinkedList<T>:

1. Индексатор (this[int index]):

Позволяет получать элементы по индексу. Индексация начинается с нуля. Используется метод GetAt(int index) для получения элемента по индексу.

1. Метод ToList:

Преобразует связанный список в обычный список List<T>. Проходит по всем элементам, добавляя их в новый список.

1. Метод GetAt(int index):

Возвращает элемент списка по индексу. Проверяет, что индекс не меньше нуля и не выходит за пределы списка.

1. Метод RemoveAt(int index):

Удаляет элемент по заданному индексу. Если индекс равен 0, то удаляется первый элемент, иначе нужно найти предыдущий элемент и переподключить его к следующему.

1. Метод AddFirst(T data):

Добавляет элемент в начало списка. Создается новый узел с данным значением, и он становится новым head.

1. Метод AddLast(T data):

Добавляет элемент в конец списка. Если список пуст, новый элемент становится head. В противном случае, проходится по всем элементам до последнего и добавляется новый элемент.

1. Метод RemoveFirst():

Удаляет первый элемент списка и возвращает его значение. Если список пуст, выбрасывает исключение.

1. Метод IsEmpty():

Проверяет, пуст ли список, возвращая true, если head равен null.

1. Метод Peek():

Возвращает первый элемент списка (данные из head), но не удаляет его. Если список пуст, выбрасывает исключение.

1. Метод Count():

Возвращает количество элементов в списке, проходя по всем узлам и увеличивая счетчик.

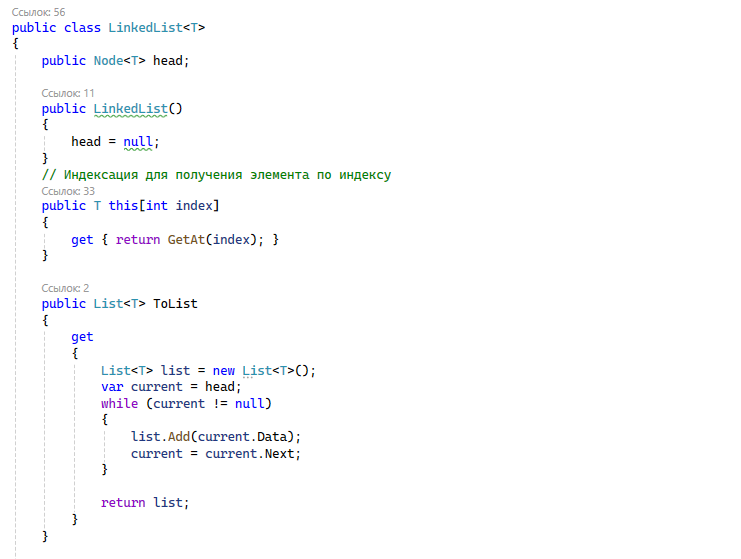


Рис.1.1.1. Код класса LinkedList



Рис.1.1.2. Код класса LinkedList

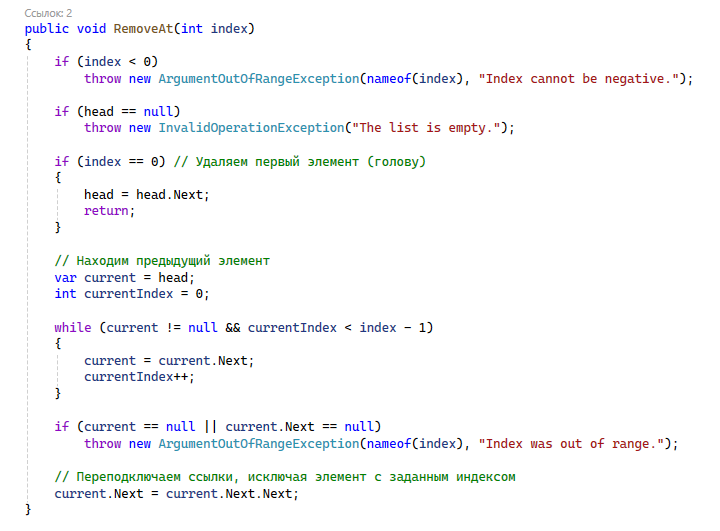


Рис.1.1.3. Код класса LinkedList

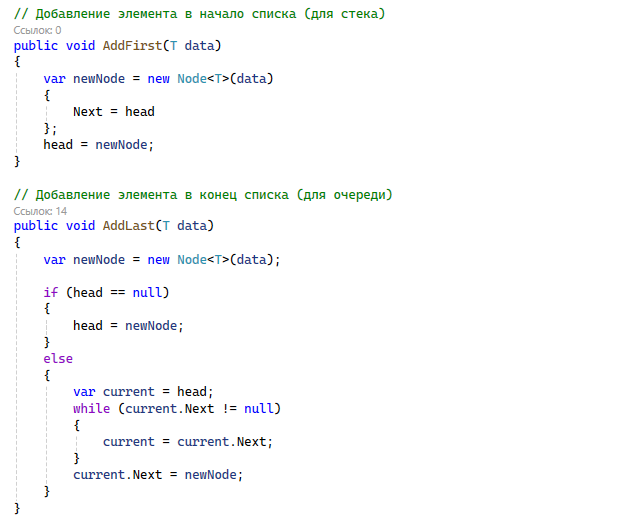


Рис.1.1.4. Код класса LinkedList



Рис.1.1.5. Код класса LinkedList

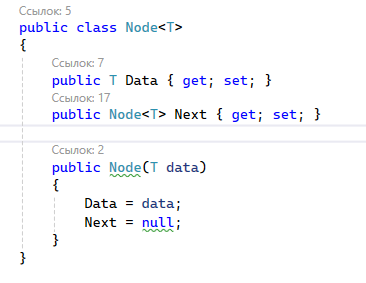


Рис.1.2. Код класса Node

I. Стек.

Стек -  абстрактный тип данных, представляющий собой список элементов, организованных по принципу *LIFO* (*last in — first out*, «последним пришёл — первым вышел»).

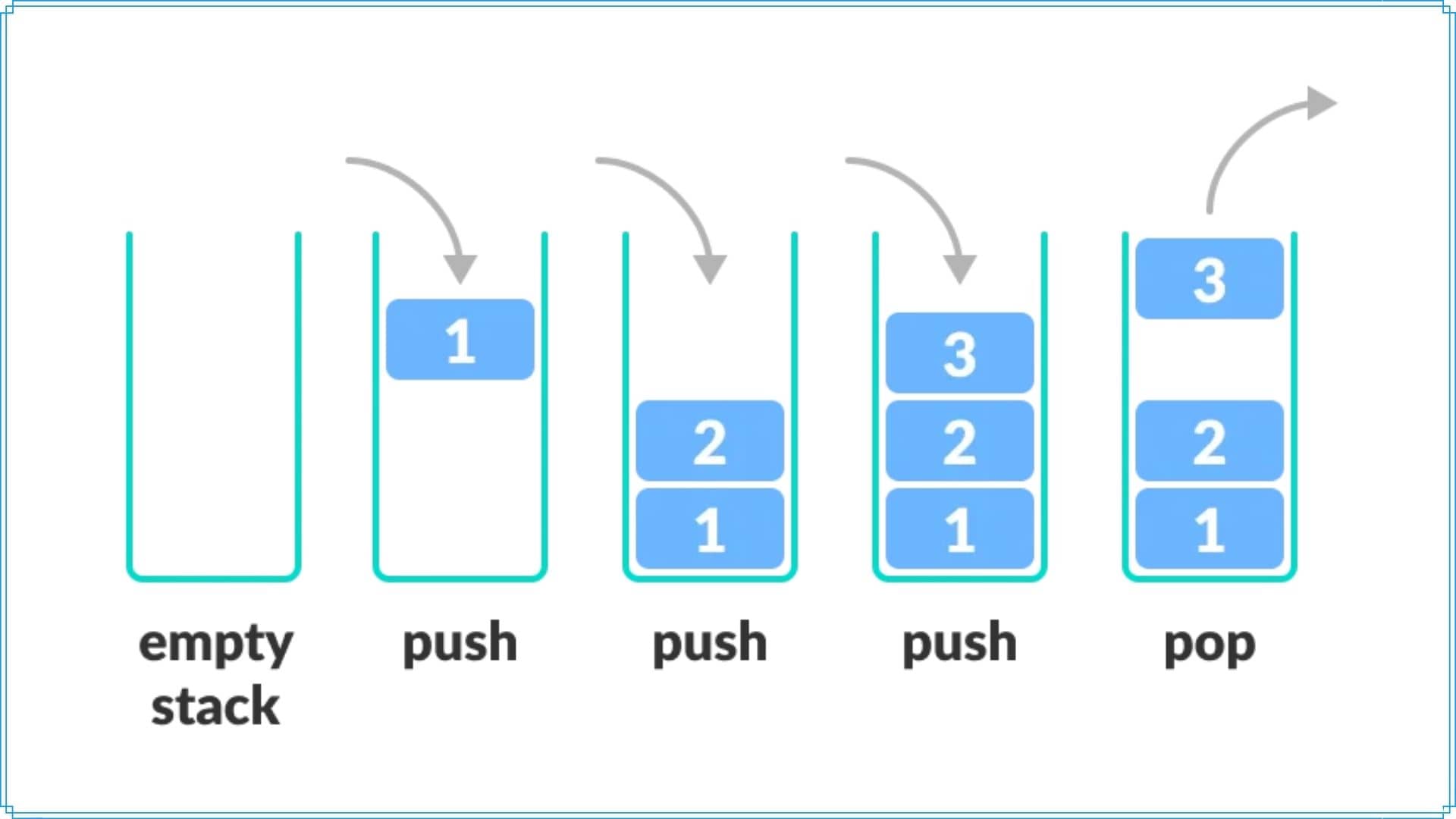


Рис 2.1. Принцип организации работы стека

В нашем коде стек основан как отдельный класс, имеющий реализацию на базе дженериков чтобы описать его работу сразу для всех типов данных (а также типы данных созданные пользователем). Наш стек имеет следующие методы:

Push(T name) – этот метод добавляет элемент любого типа данных в том числе пользовательского на вершину нашего стека.

Pop() – возвращает из стека верхний элемент одновременно удаляя его из стека.

IsEmpty() – метод который проверяет пустой ли стек, в случае если это так возвращает булевое значение True, в ином False.

Top() или Peek() – это два метода с одинаковым функционалом (в псевдокоде используют Top, в C# это Peek()), которые возвращают значение находящееся в верхнем элементе стека.

Print() – метод который распечатывает пользовательский стек на экран.

UniqueValue(T value) – метод который проверяет является ли поданный пользователем элемент на вход уникальным или уже присутствует в структуре.

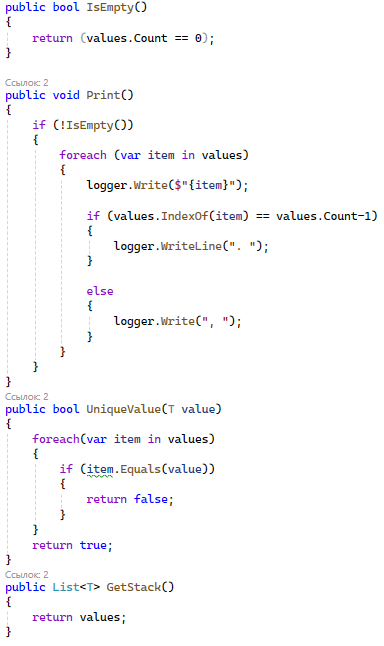
GetStack() – возвращает стек в виде списка находящихся в нем элементов.

Count() – подсчитывает количество элементов в стеке.

Ниже представлена реализация стека на языке C# в нашей программе (рисунки 2.2.1 – 2.2.2):



Рис 2.2.1. Программный код стека на языке C#.



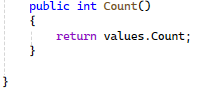


рис 2.2.2. Программный код стека на языке C#.

Обработчик для Стека

В нашей программе для обработки и подсчета времени выполнения последовательности используется класс StackHandler, которому на вход подается имя файла а он возвращает список из замеров времени.

Замеры времени реализованы через TimeSpan. Время последовательно записывается в список и измеряется в мс\*100.

Программный код для этого класса нашей программы на языке C#(рисунки 2.3.1 – 2.3.3):

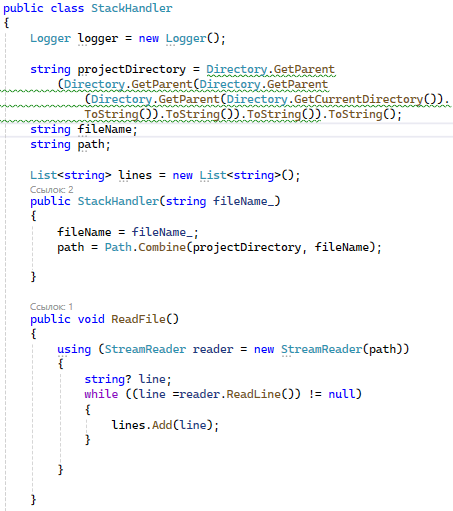


Рис 2.3.1. Программный код обработчика стека на языке C#.

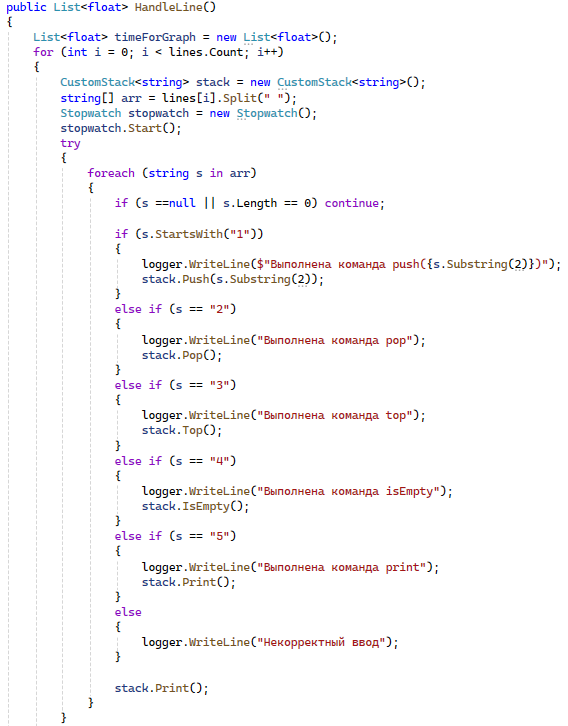


Рис 2.3.2. Программный код обработчика стека на языке C#



Рис 2.3.3. Программный код обработчика стека на языке C#

LinkedListStack

Так же был реализован стек, который основан на нашем связном списке LinkedList. Он выполняет те же функции и имеет ту же суть. Цель его создания – это получение знаний и представления о том как работает динамическая струтктура Связной Список

Программный код для этого класса нашей программы на языке C#(рисунки 2.4.1 – 2.4.3):

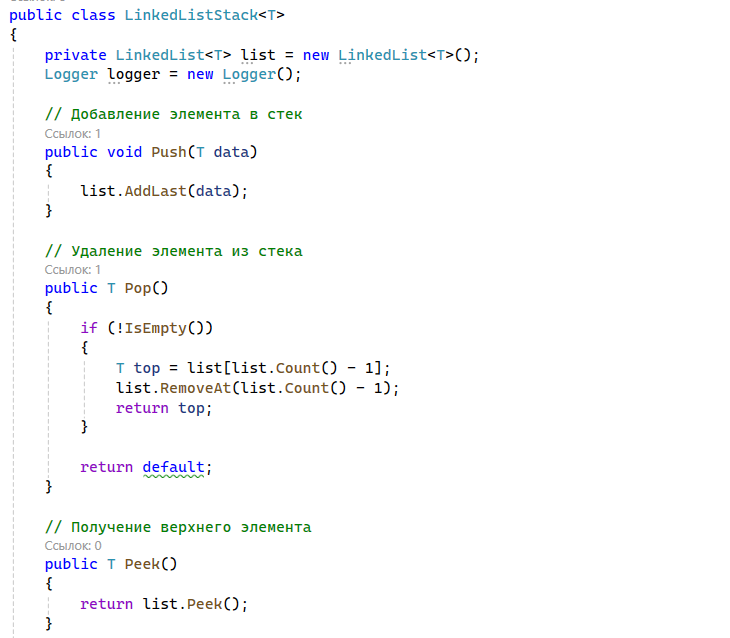


Рис.2.4.1. Код LinkedListStack



Рис.2.4.2. Код LinkedListStack

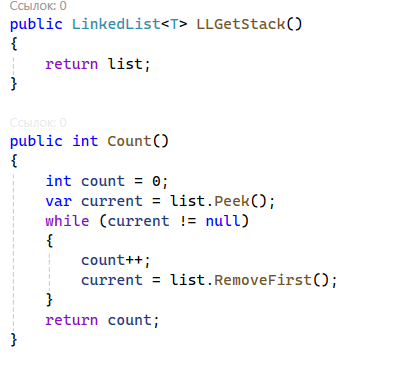


Рис.2.4.3. Код LinkedListStack

Под этот класс был реализован свой обработчик LinkedListStackHandler, с таким же функционалом как и обычный.

Программный код для этого класса нашей программы на языке C#(рисунки 2.5.1 – 2.5.3):



Рис.2.5.1 Код обработчика для LinkedListStack

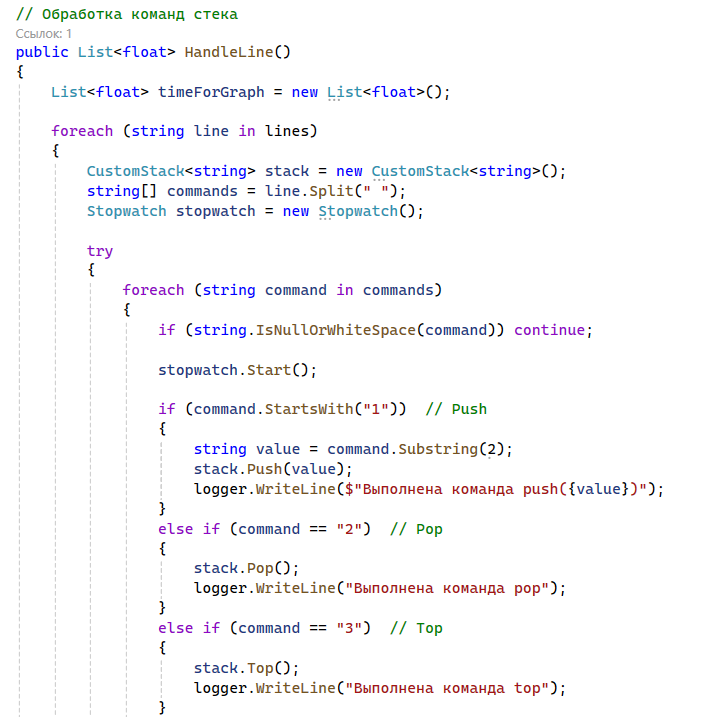


Рис.2.5.2. Код обработчика для LinkedListStack



рис.2.5.3. Код обработчика для LinkedListStack

ОПЗ калькулятор

В нашей программе реализован калькулятор на основе обратной польской записи который работает на нашем собственном стеке и умеет:

1) Базовые арифметические операции: +, -, \*, / , ^

2) Тригонометрические операции: sin, cos, tg, ctg

3) Логарифмы: log, ln, lg

4) Корни: sqrt, cbrt, rt

Он состоит из 7 вложенных классов и занимает более двух тысяч строк, поэтому будет приведено краткое описание работы классов:

Delimiter – отвечает за то, какой будет разделитель в операциях подразумевающих на вход два операнда

Number – отвечает за численные составляющие при подсчете нашего выражения

Operation – отвечает за операции и их приоритет.

Parenthesis – отвечает за работу со скобками и их приоритет.

RPNCalculator – собирает все классы и возвращает ответ (посчитанное выражение).

Tokens – общий базовый родительский класс, от которого наследуются все составляющие выражения (числа, скобки, операции, переменные).

Variable – отвечает за работу с переменными в выражении если таковые присутствуют.

II.Очередь.

Очередь — **это структура данных, представляющая собой последовательность элементов.** Доступ к этим элементам возможен только по принципу **FIFO (First In, First Out)**: из очереди можно быстро и легко извлечь элемент, который расположен в самом её начале и находится в ней дольше всего.

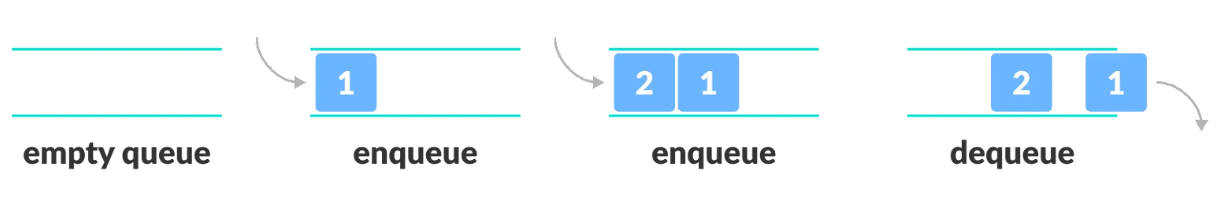


Рис 3.1. Как работает очередь в программировании.

Очередь в нашем программном коде работает на основе дженериков и обладает следующими методами:

Enqueue(T name) – этот метод добавляет элемент любого типа данных в том числе пользовательского в конец очереди.

Dequeue() – возвращает из стека верхний элемент одновременно удаляя его из стека.

IsEmpty() – метод который проверяет пуста ли очередь, в случае если это так возвращает булевое значение True, в ином False.

Top() или Peek() – это два метода с одинаковым функционалом (в псевдокоде используют Top, в C# это Peek()), которые возвращают значение находящееся в начале очереди.

Print() – метод который распечатывает пользовательскую очередь на экран.

GetQueue() – возвращает очередь в виде списка находящихся в нем элементов.

Count() – подсчитывает количество элементов в очереди.

Ниже представлена реализация очереди на языке C# в нашей программе (рисунки 3.2.1 – 3.2.2):

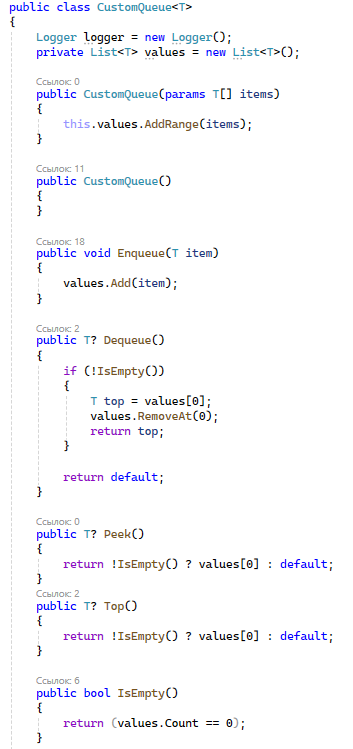


Рис 3.2.1. Реализация очереди на языке C#

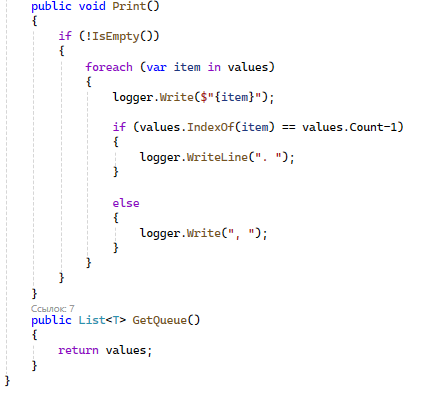


Рис 3.2.2. Реализация очереди на языке C#

Обработчик для Очереди

В нашей программе для обработки и подсчета времени выполнения последовательности используется класс QueueHandler, которому на вход подается имя файла а он возвращает список из замеров времени.

Замеры времени реализованы через TimeSpan. Время последовательно записывается в список и измеряется в мс\*100.

Программный код для этого класса нашей программы на языке C#(рисунки 3.3.1 – 3.3.3):

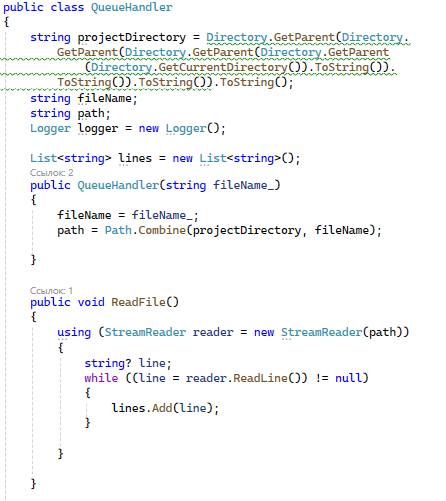


Рис 3.3.1. Реализация обработчика очереди на языке C#.

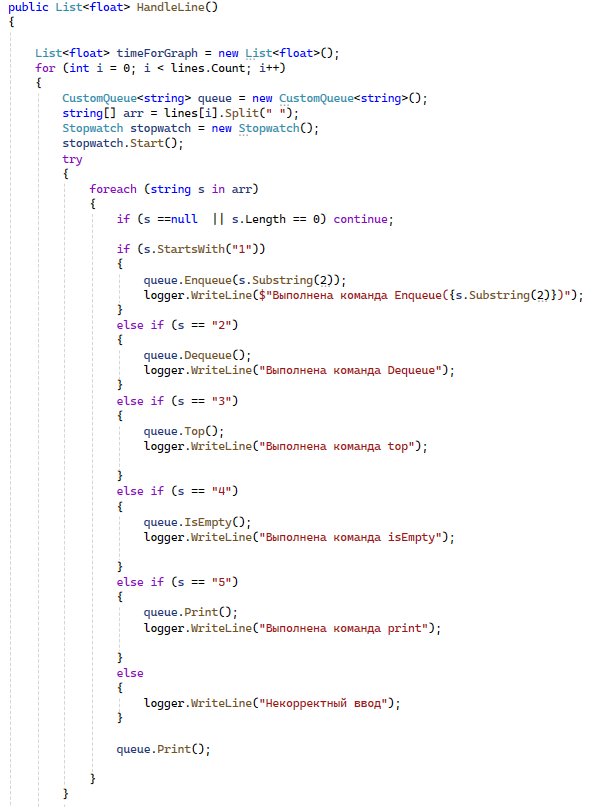


Рис 3.3.2. Реализация обработчика очереди на языке C#.



Рис 3.3.3. Реализация обработчика очереди на языке C#.

LinkedListStack

Так же была реализована очередь, которая основана на нашем связном списке LinkedList. Она выполняет те же функции и имеет ту же суть. Цель ее создания – это получение знаний и представления о том как работает динамическая струтктура Связной Список

Программный код для этого класса нашей программы на языке C#(рисунки 3.4.1 – 3.4.3):

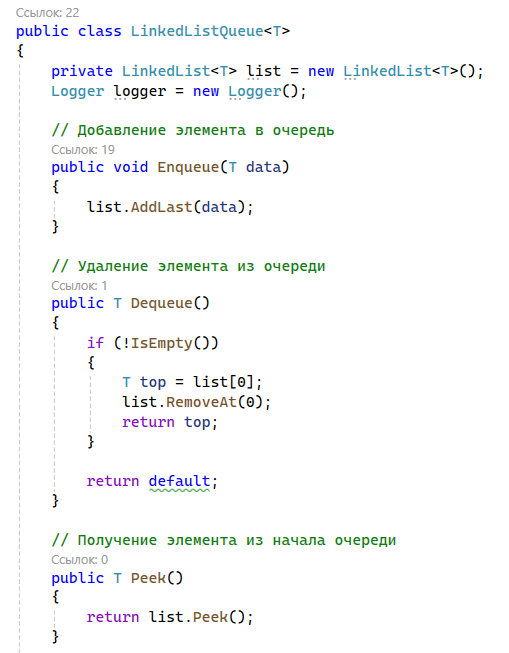


Рис.3.4.1 Код LinkedListQueue

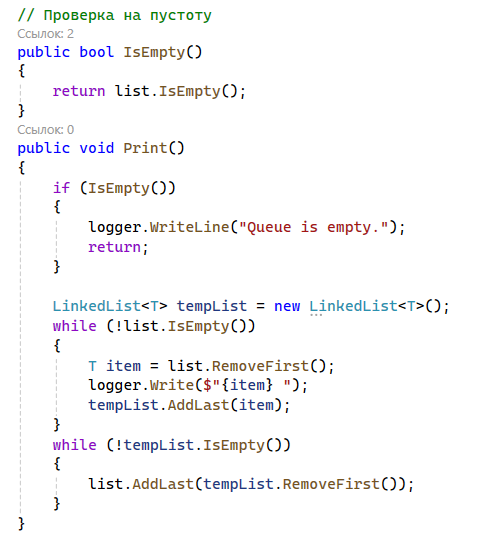


Рис.3.4.2 Код LinkedListQueue



Рис.3.4.3 Код LinkedListQueue

Под этот класс был реализован свой обработчик LinkedListQueueHandler, с таким же функционалом как и обычный.

Программный код для этого класса нашей программы на языке C#(рисунки 3.5.1 – 3.5.3):



Рис.3.5.1. Код обработчика для LinkedListQueue

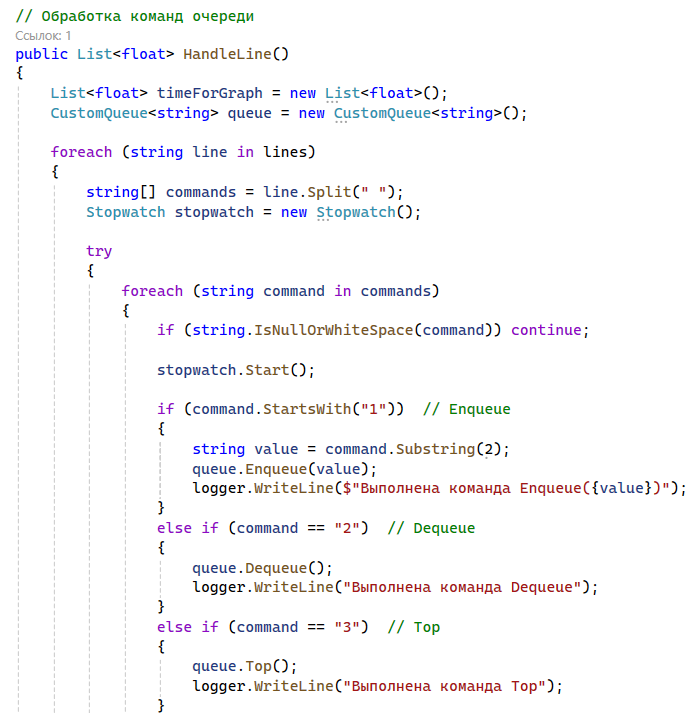


Рис.3.5.2. Код обработчика для LinkedListQueue



Рис.3.5.3. Код обработчика для LinkedListQueue

III.Применение

Задача этой части – показать на примерах, где используются динамические структуры, такие как стек, очередь, дерево и список.

Редактор

Начнем со стека:

В качестве примера мы реализовали текстовый редактор, который содержит базовые функции: запись текста, отмена последнего действия, возвращение отмененного действия, сохранение файла в формате txt.

Редактор состоит из двух классов.

Класс TextAction (рис. 4.1) является вспомогательным и отвечает за хранение информации о каждом действии, которое можно отменить или повторить.

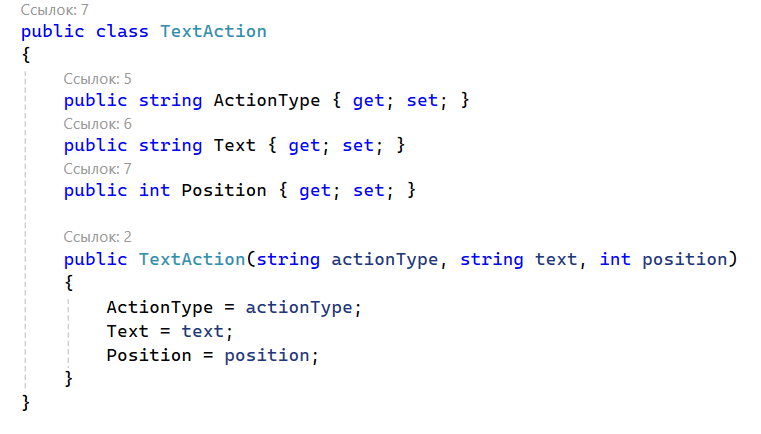


Рис.4.1. Код класса TextAction

TextEditor (рисунки 4.2.1 – 4.2.4) - это основной класс, отвечающий за функциональность текстового редактора. Он содержит в себе главные методы:

1. Undo() - отменяет последнее действие из стека Undo.
2. Redo() - повторяет последнее отмененное действие из стека Redo.
3. SaveToFile() - сохраняет текст в файл.



Рис.4.2.1. Код класса TextEditor



Рис.4.2.2. Код класса TextEditor

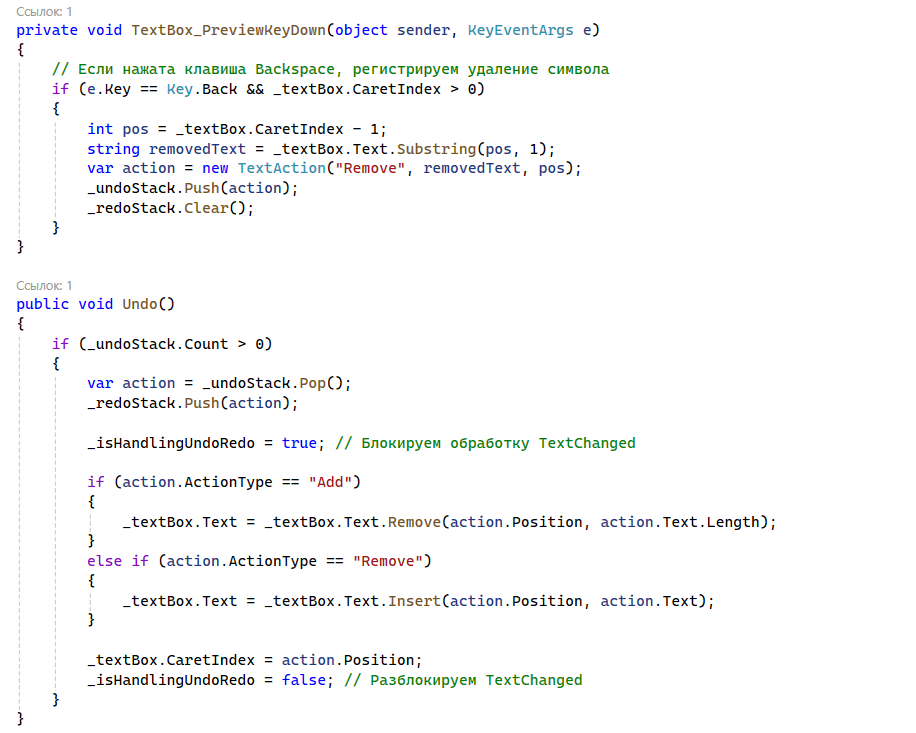


Рис.4.2.3. Код класса TextEditor



Рис.4.2.4. Код класса TextEditor

Визуализация проекта показана на рисунке 4.2.5

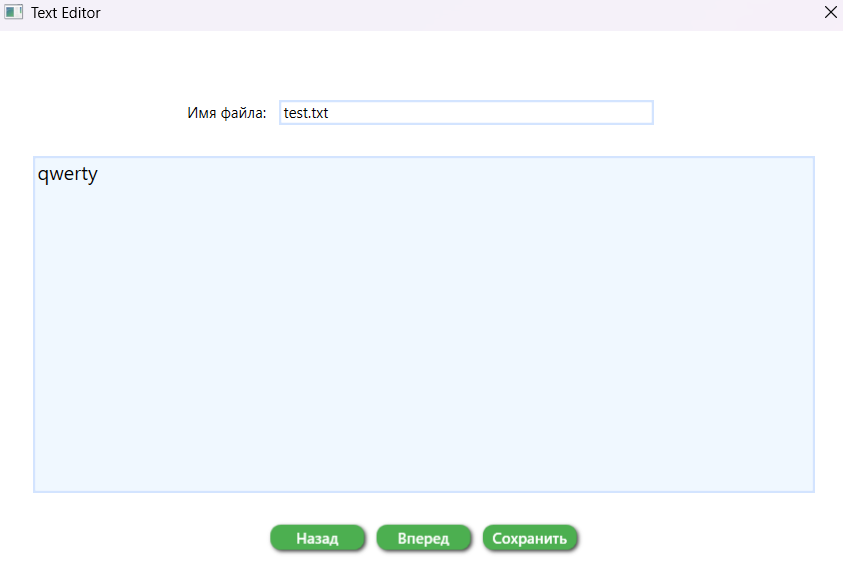


Рис.4.2.5 Визуализация редактора

Банк

Дальше на очереди идет Очередь. На основе этой динамической струтуры была написана программа BankQueueSystem. Это имитация очереди в банке, где n количество поситителей обслуживается m количеством банкиров.

Код состоит из частей PlaceholderTextBox и MainWindow.

PlaceholderTextBox (рисунки 4.3.1 – 4.3.2) - это пользовательский элемент управления, который реализует текстовое поле с подсказкой. Он использует свойства зависимости Text и Placeholder для хранения текста и подсказки соответственно. Подсказка отображается, когда поле пусто, и скрывается, когда поле получает фокус или в него вводится текст.

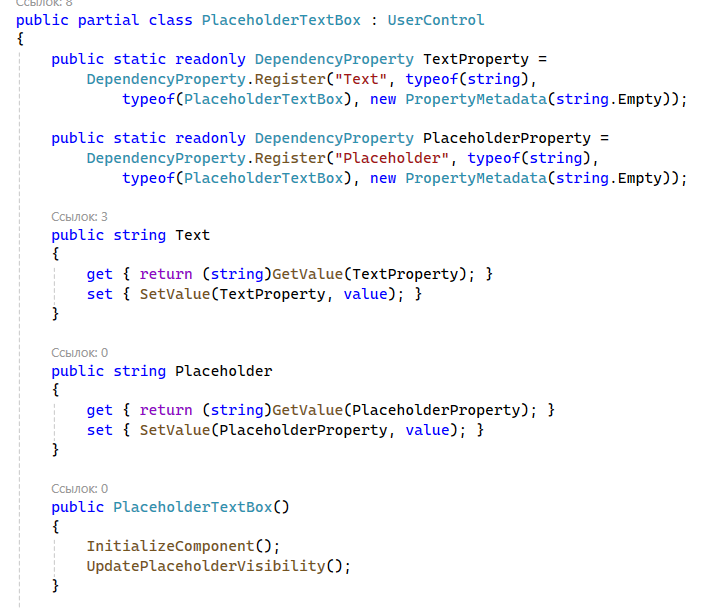


Рис.4.3.1 Код PlaceholderTextBox

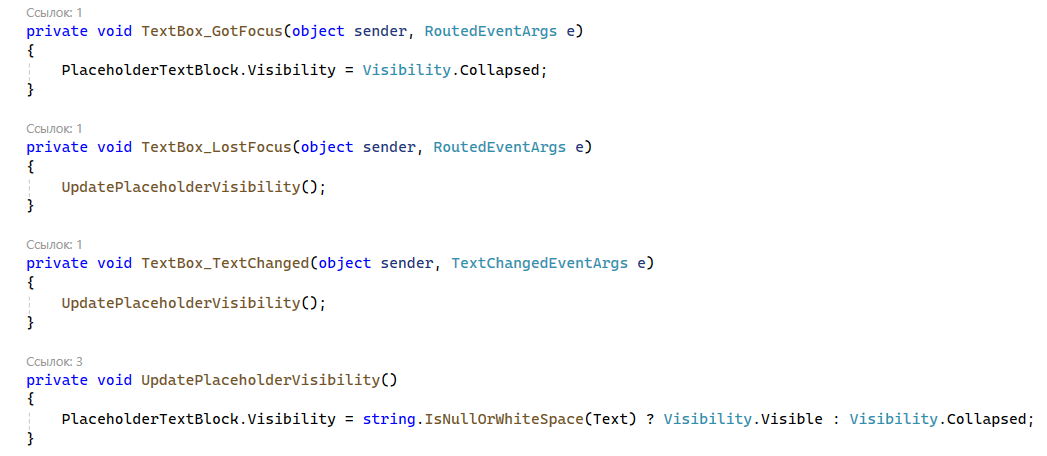


Рис.4.3.2 Код PlaceholderTextBox

MainWindow (рисунки 4.4.1 - 4.4.4) - это главный класс приложения, управляющий логикой и интерфейсом программы. Он содержит очередь клиентов (\_queue), список полос прогресса кассиров (\_bankersProgressBars) и список, указывающий, свободен ли кассир (\_bankersAvailability).

В MainWindow реализованы обработчики событий "Добавить клиента" и "Установить кассиров". Функция "Добавить клиента" добавляет нового клиента в очередь, а функция "Установить кассиров" задает количество кассиров в системе.

Метод ProcessQueue() асинхронно обрабатывает очередь клиентов, извлекая клиентов из очереди, находя свободного кассира и назначая ему клиента. Метод ServeClient() имитирует обслуживание клиента, постепенно увеличивая значение полосы прогресса кассира.

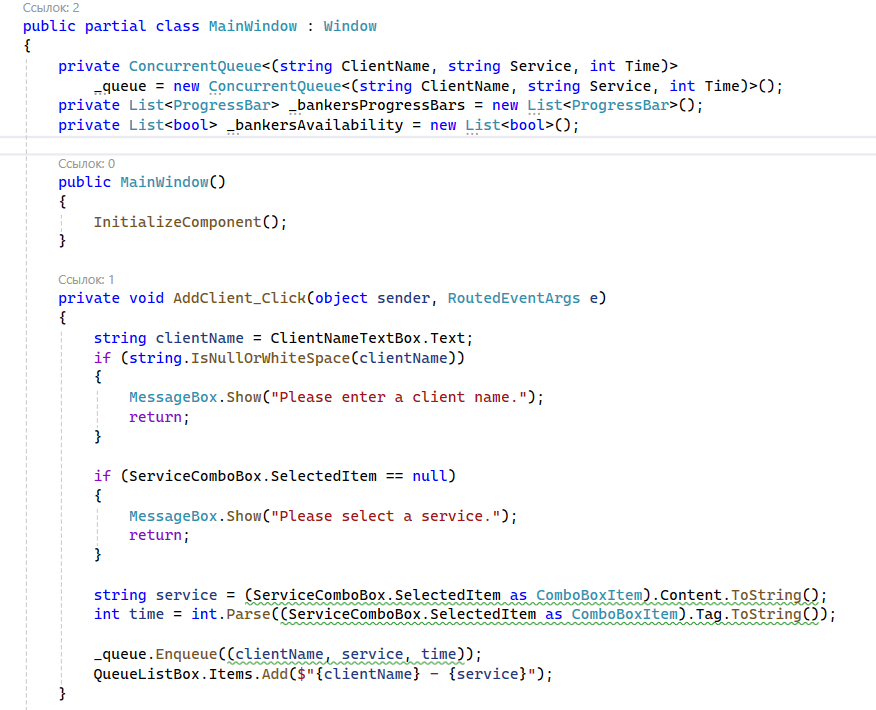


Рис.4.4.1 Код MainWindow

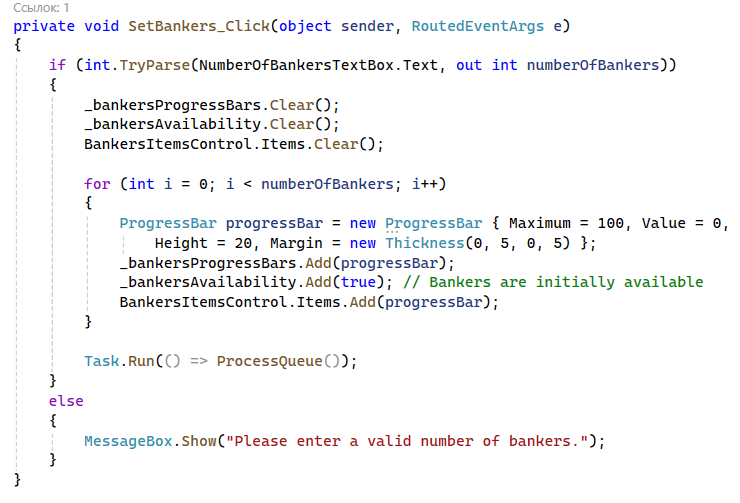


Рис.4.4.2 Код MainWindow

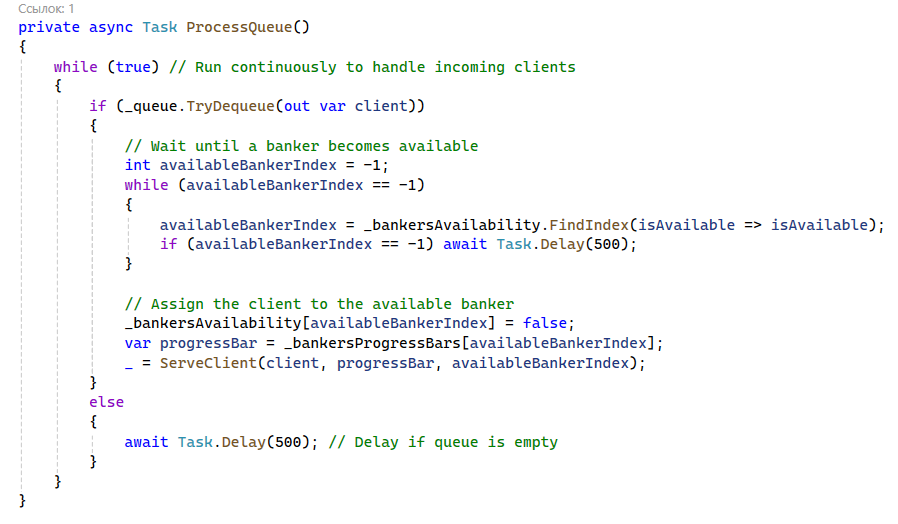


Рис.4.4.3 Код MainWindow



Рис.4.4.4 Код MainWindow

Визуализация проекта показана на рисунке 4.4.5

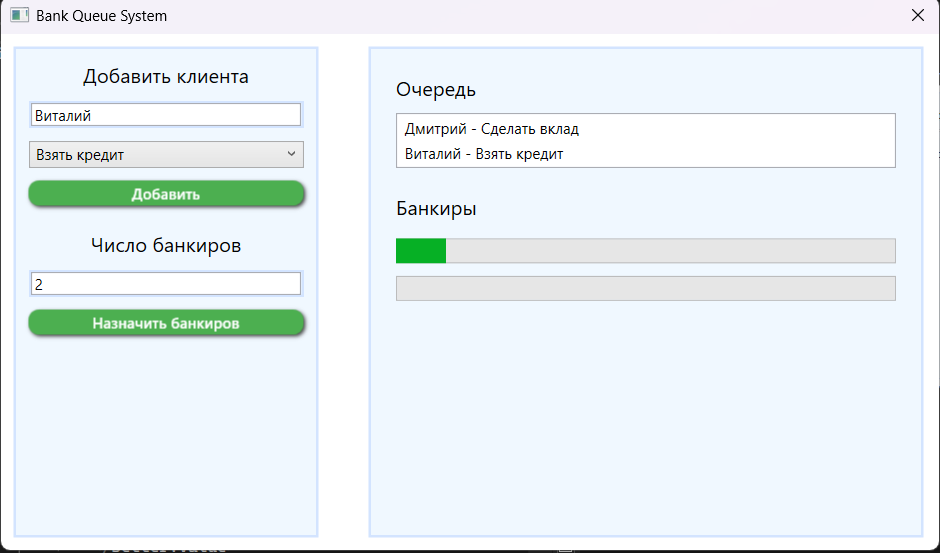


Рис.4.4.5 Визуализация очереди в банке

Дерево

Чтобы показать применение динамической струтуры Дерево была реализована программа, представляющая из себя иерархическую струтуру сотрудников.

Класс Employee (рисунки 4.5.1 – 4.5.2) описывает отдельного сотрудника. Он хранит информацию о его имени, зарплате, должности, руководителе и списке подчиненных. С помощью методов AddSubordinate и RemoveSubordinate можно добавлять и удалять подчиненных сотрудников. Метод CalculateTotalSalary рекурсивно подсчитывает общую зарплату сотрудника, включая зарплаты всех его подчиненных.

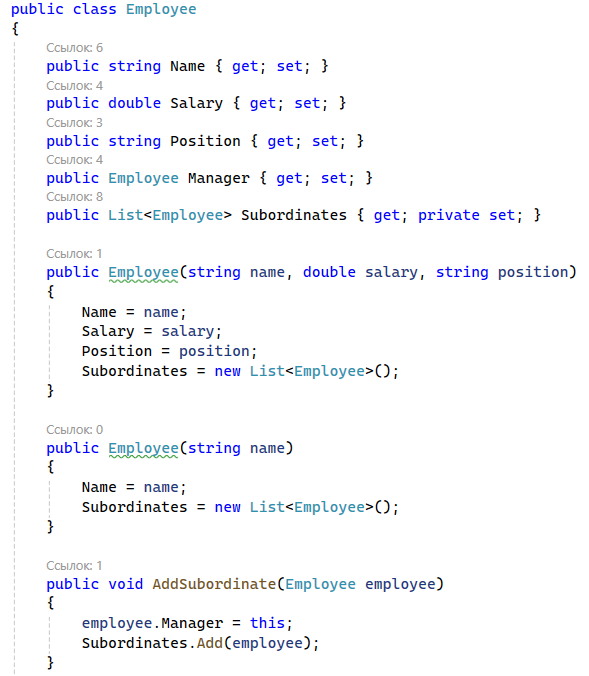


Рис.4.5.1 Код Employee

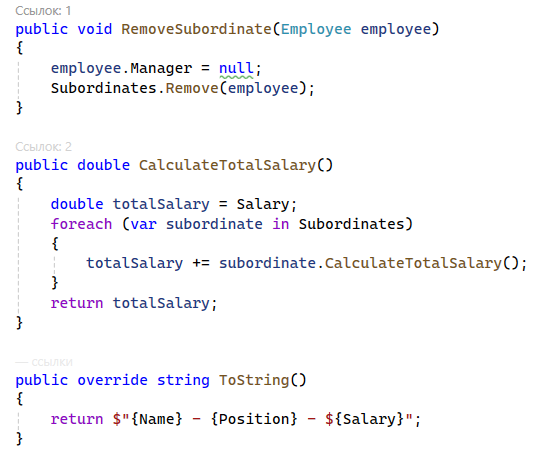


Рис.4.5.2 Код Employee

Класс EmployeeManager (рисунки 4.6.1 – 4.6.3) отвечает за управление всей организацией. Он хранит информацию о генеральном директоре (CEO) и предоставляет методы для добавления, удаления и поиска сотрудников.

EmployeeManager использует рекурсию для поиска сотрудников по имени, зарплате или должности. Метод FindEmployee ищет конкретного сотрудника по имени, а метод SearchEmployees выполняет поиск по заданным критериям, возвращая список соответствующих сотрудников.

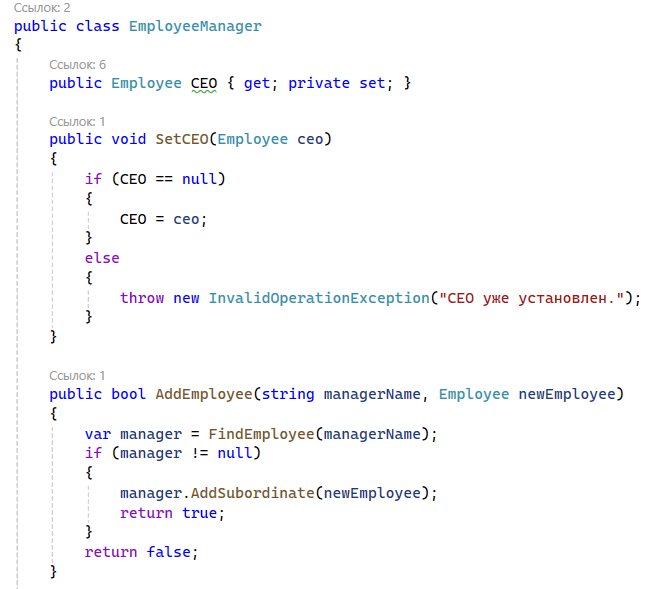


Рис.4.6.1 Код EmployeeManager



Рис.4.6.2 Код EmployeeManager

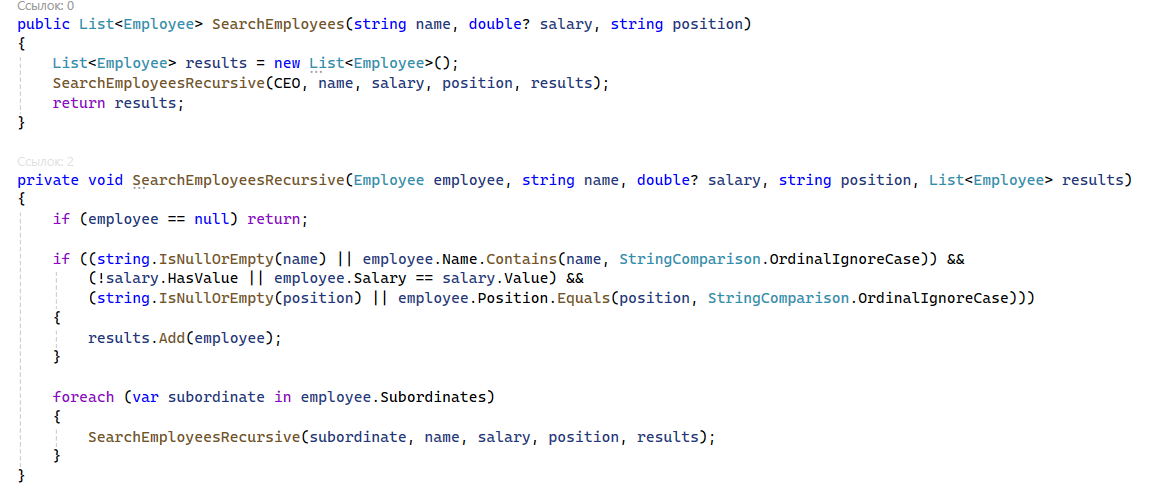


Рис.4.6.3 Код EmployeeManager

Визуализация проекта показана на рисунке 4.6.4

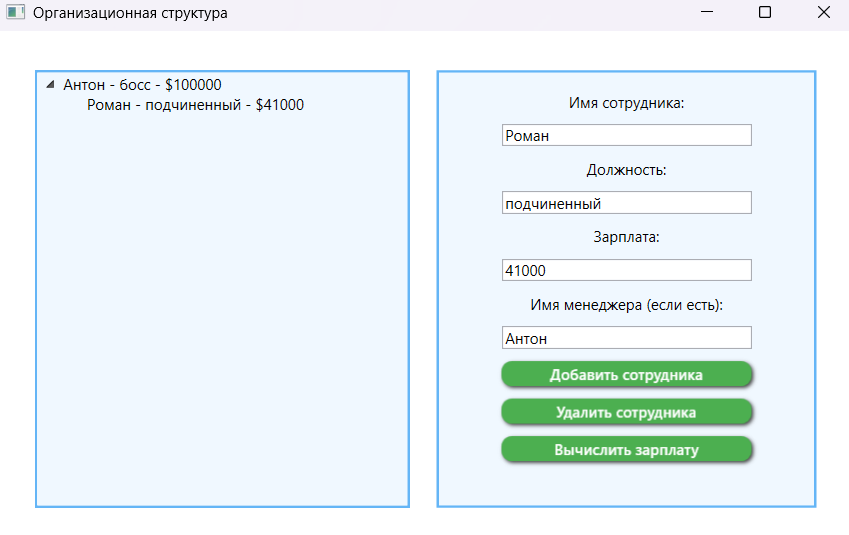


Рис.4.6.4 Визуализации иерархии рабочих

Список контактов

Чтобы показать на примере как работает динамическая структура Список была реализована программа Список контактов. Она выполняет все основные функции списка контактов, в телефоне.

Код состоит из двух классов: Person и ContactList.

Класс Person (рис.4.7) описывает информацию о конкретном человеке, храня информацию об имени, фамилии и номере телефона. Он предоставляет два конструктора: один для создания объекта с полным именем и номером телефона, а другой для создания объекта только с именем и номером телефона.

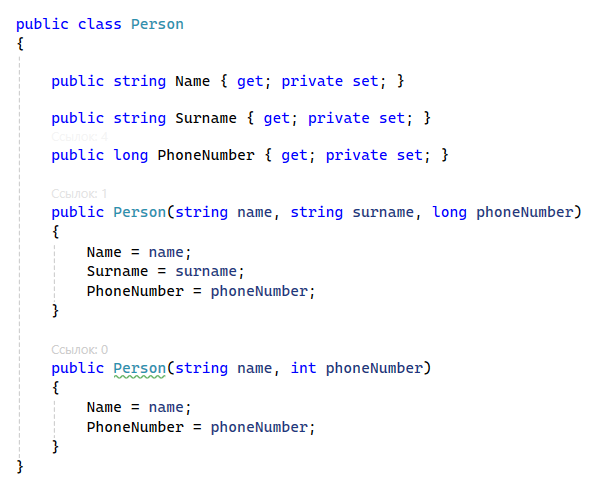


Рис.4.7. Код Person

Класс ContactList (рисунки 4.8.1 – 4.8.4) является основным классом, отвечающим за управление списком контактов. Он хранит список объектов Person и предоставляет различные методы для работы с ним:

1. AddPerson: Добавляет новый контакт в список.
2. GetPersons: Возвращает список всех контактов.
3. RemovePerson: Удаляет контакт из списка.

Так же была реализована сортировка списка контактов по разным параметрам.

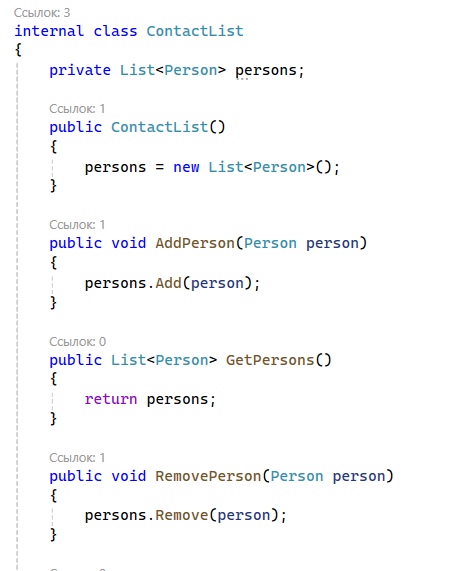


Рис.4.8.1. Код ContactList

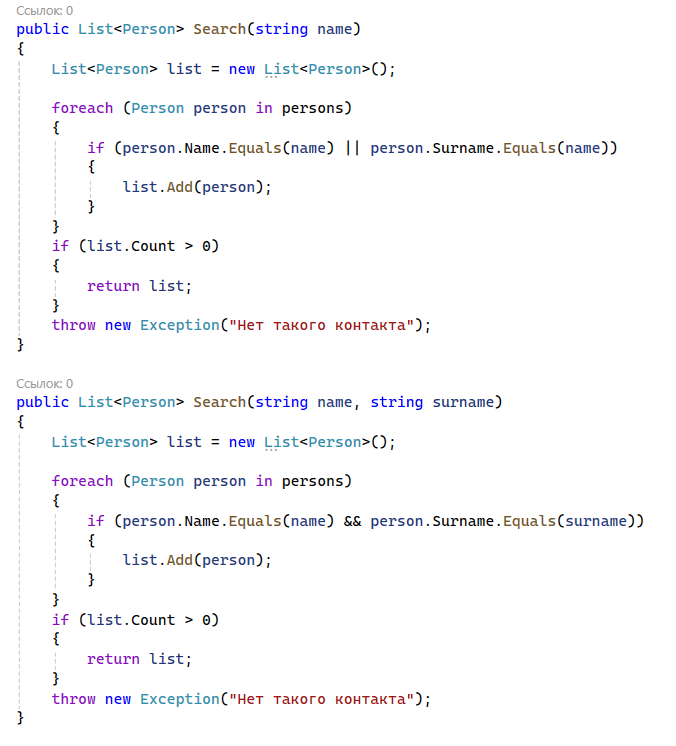


Рис.4.8.2. Код ContactList



Рис.4.8.3. Код ContactList

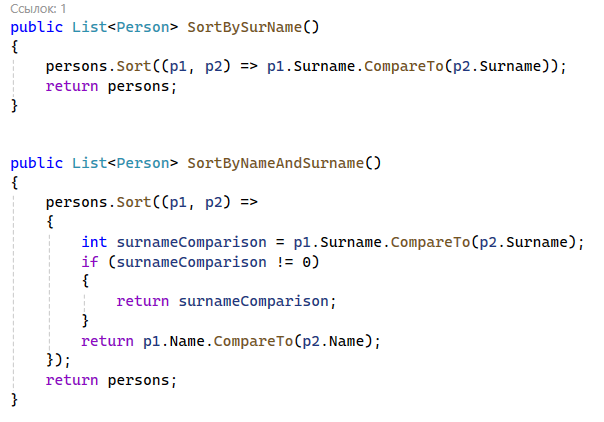


Рис.4.8.4. Код ContactList

Визуализация проекта показана на рисунке 4.8.5

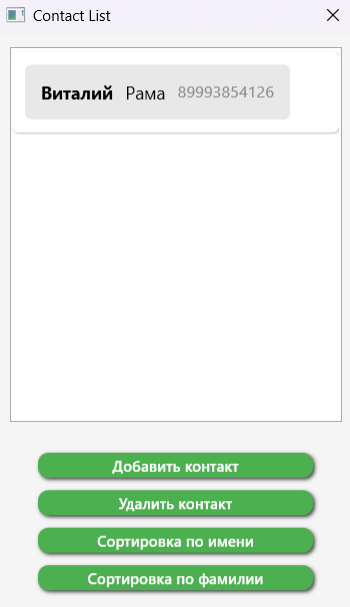


Рис.4.8.5 Визуализация списка контактов

IV. Задачи

В этом блоке будет описание задачь с использованием связного списка LinkedList<T>.

1. Написать функцию, которая переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке.

Код решения представлен на рисунке 5.1

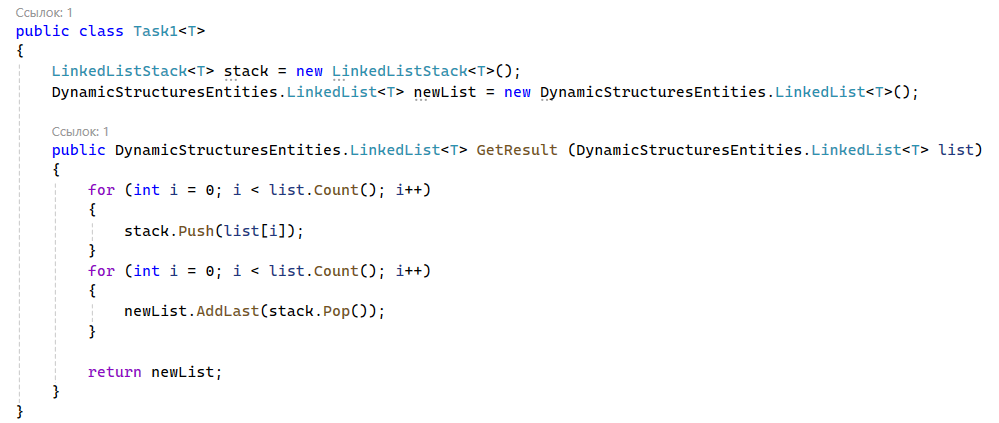


Рис.5.1. Код задачи 1

1. Написать функцию, которая переносит в начало (в конец) непустого списка L его последний (первый) элемент.

Код решения представлен на рисунке 5.2



Рис.5.2. Код задачи 2

1. Написать функцию, которая определяет количество различных элементов списка, содержащего целые числа.

Код решения представлен на рисунке 5.3

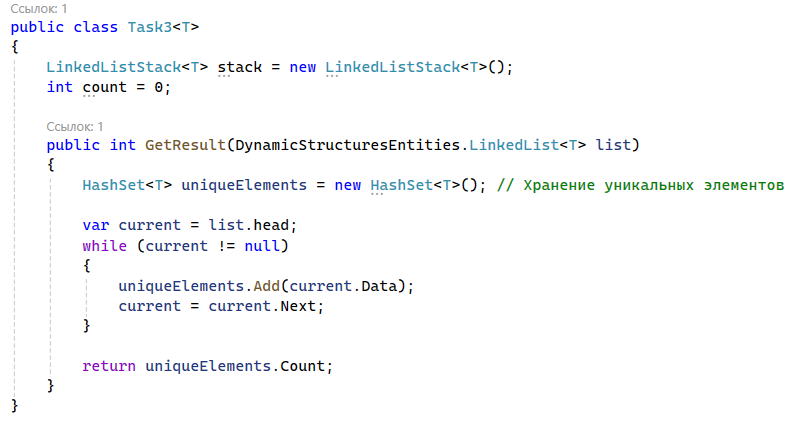


Рис.5.3. Код задачи 3

1. Написать функцию, которая удаляет из списка L неуникальные элементы.

Код решения представлен на рисунке 5.4

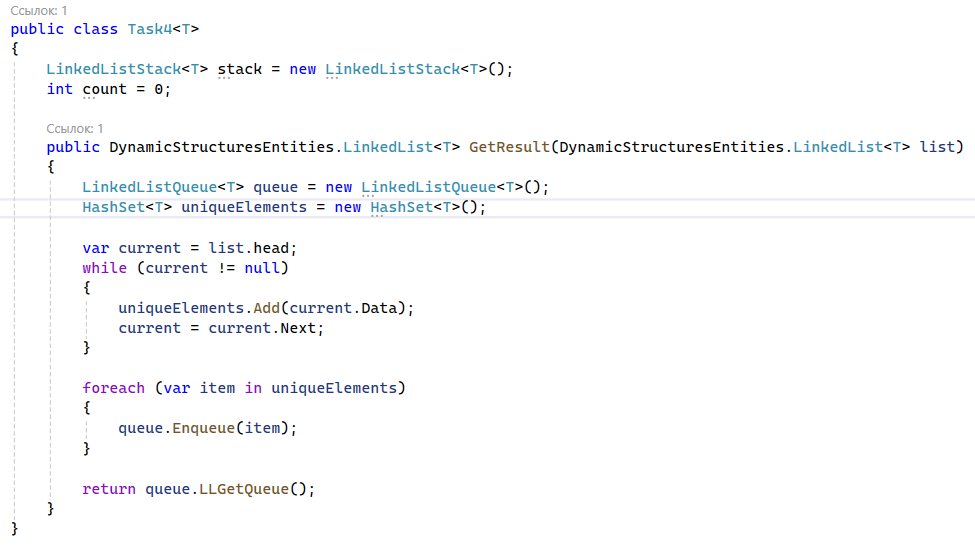


Рис.5.4. Код задачи 4

1. Написать функцию вставки списка самого в себя вслед за первым вхождением числа х.

Код решения представлен на рисунке 5.5

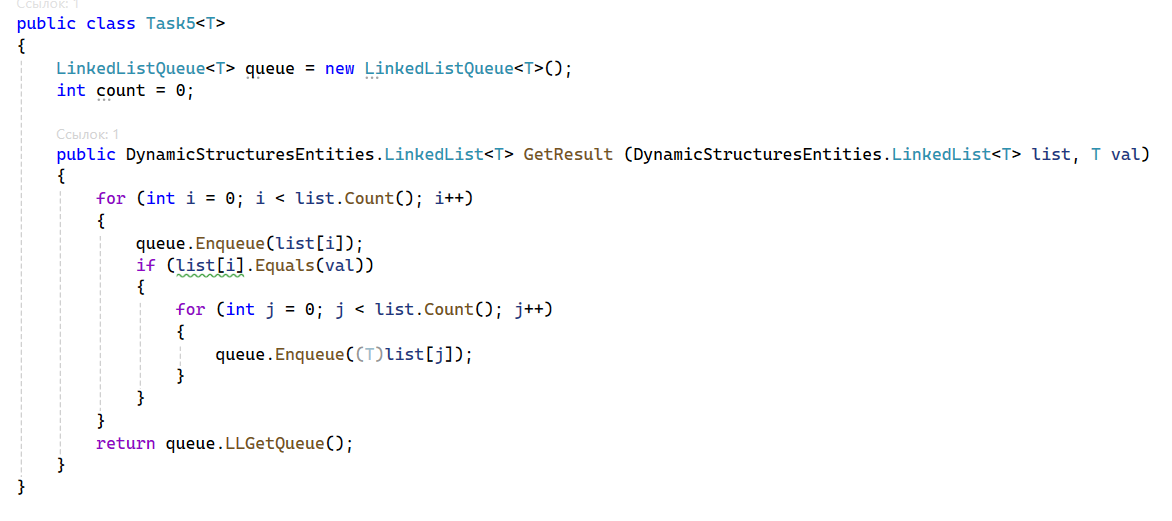


Рис.5.5. Код задачи 5

1. Написать функцию, которая вставляет в непустой список L, элементы которого упорядочены по не убыванию, новый элемент Е так, чтобы сохранилась упорядоченность.

Код решения представлен на рисунке 5.6

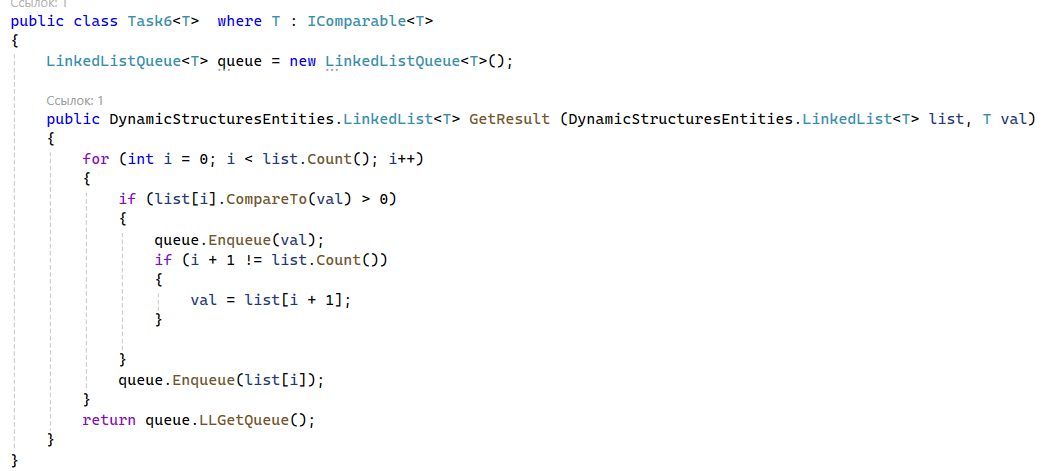


Рис.5.6. Код задачи 6

1. Написать функцию, которая удаляет из списка L все элементы Е, если таковые имеются.

Код решения представлен на рисунке 5.7

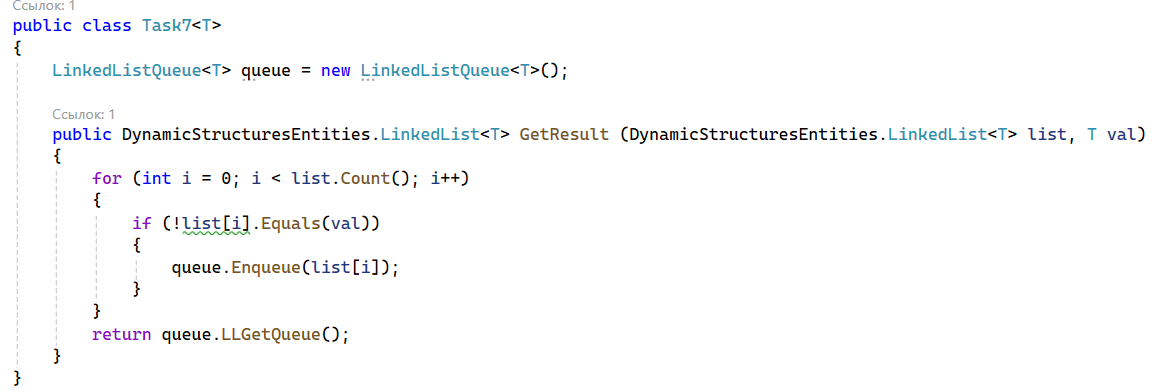


Рис.5.7. Код задачи 7

1. Написать функцию, которая вставляет в список L новый элемент F перед первым вхождением элемента Е, если Е входит в L.

Код решения представлен на рисунке 5.8

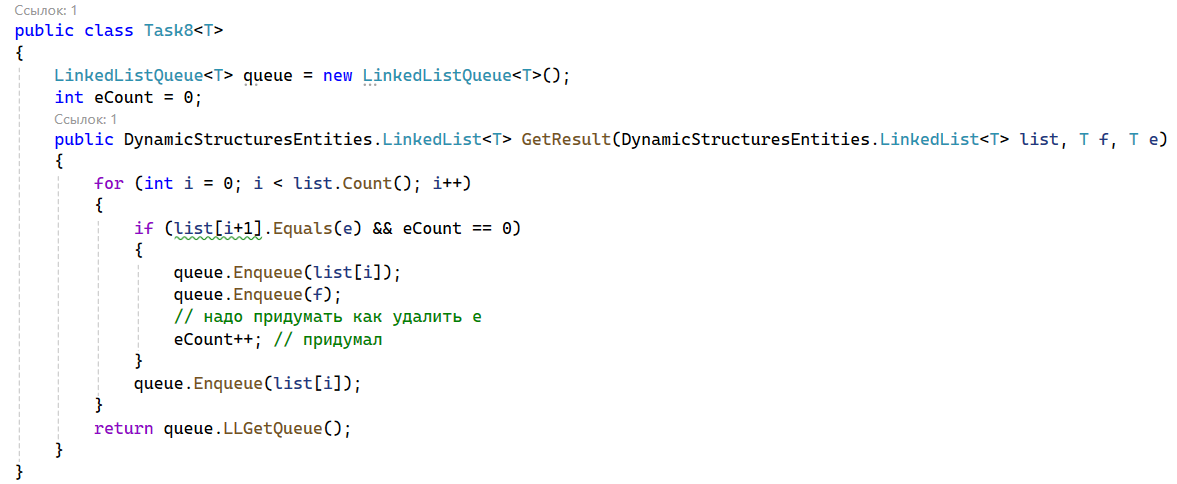


Рис.5.8. Код задачи 8

1. Функция дописывает к списку L список E. Оба списка содержат целые числа. В основной программе считать их из файла.

Код решения представлен на рисунке 5.9

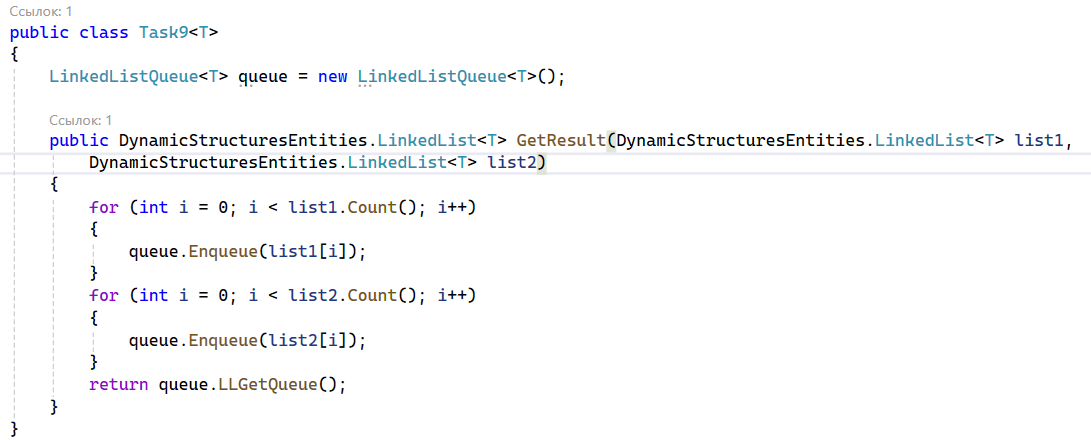


Рис.5.9. Код задачи 9

1. Функция разбивает список целых чисел на два списка по первому вхождению заданного числа. Если этого числа в списке нет, второй список будет пустым, а первый не изменится.

Код решения представлен на рисунке 5.10

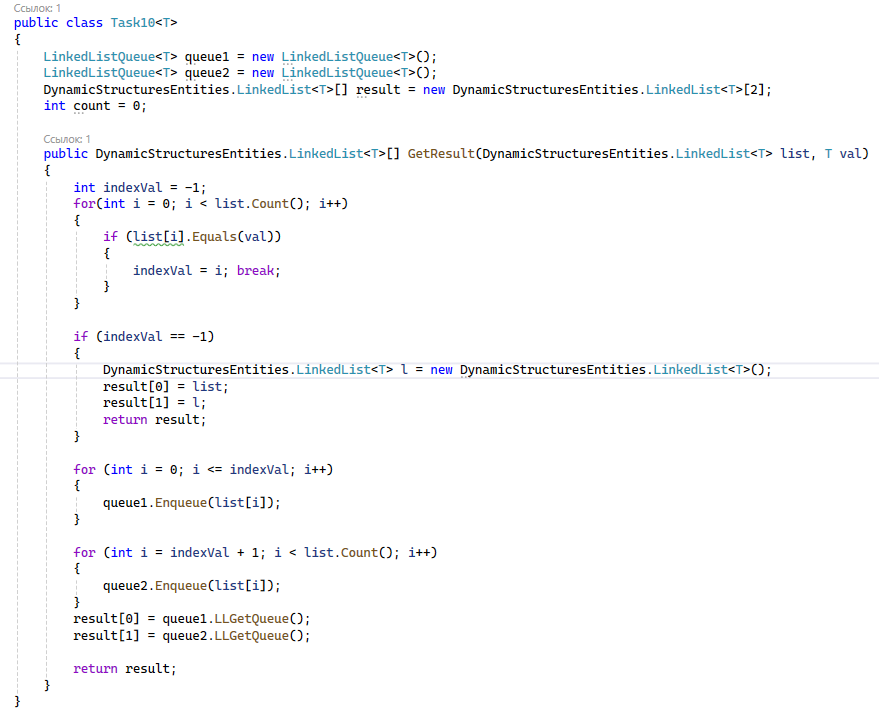


Рис.5.10. Код задачи 10

1. Функция удваивает список, т.е. приписывает в конец списка себя самого.

Код решения представлен на рисунке 5.11

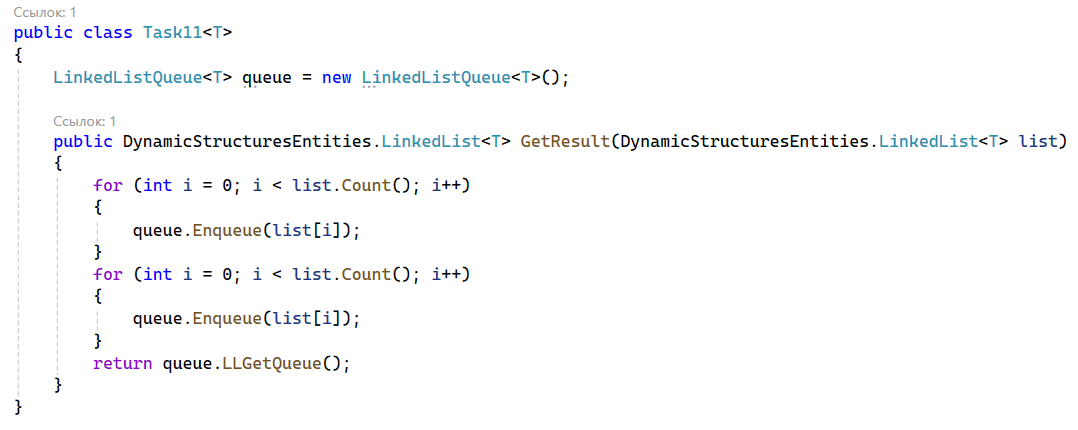


Рис.5.11. Код задачи 11

1. Функция меняет местами два элемента списка, заданные пользователем.

Код решения представлен на рисунке 5.12

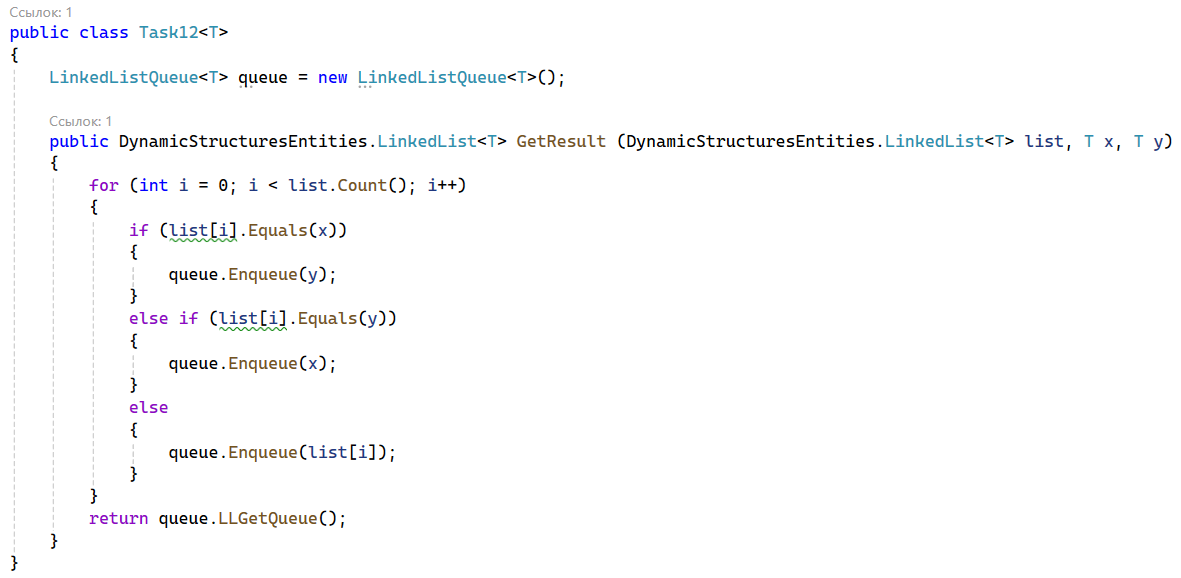


Рис.5.12. Код задачи 12

Анализ

В результате анализа было выявлено, что самой легковесной командой для стека является команда Print(). Так как исполняя ее программа проходит по каждому элементу связного списка.

В случае добавления и выкидывания элемента нужно поменять ссылку только в верхнем Node.

В случае выполнения команды IsEmpty() проверяется, есть ли в списке хотя бы один элемент. Поэтому он исполняется за кратчайшее время.

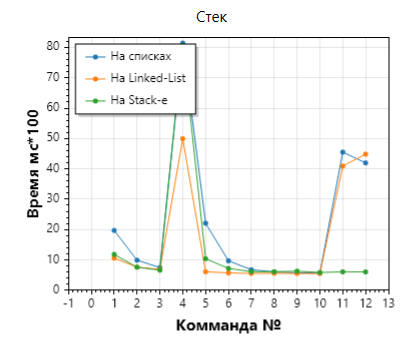


Рис.6.1 Сложность выполнения команд стека

Результаты для очереди аналогичны результатам стека (рис.6.2).

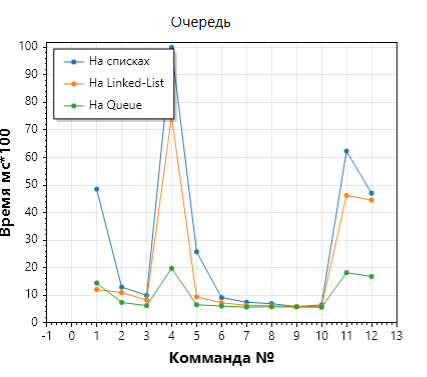


Рис.6.2 Сложность выполнения команд очереди

Последовательность команд для обеих динамических структур:

**1,2 1,5 1,queueu 5 2 2 2 1,2 1,1 1,123456 5 5,** где:

1 - Push(elem), 2 - Pop(), 3 - Top(), 4 - isEmpty(), 5 - Print() - для стека.

1 - вставка, 2 - удаление, 3 – просмотр начала очереди, 4 – проверка на пустоту, 5 – печать - для очереди.