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

Ханойские башни...................................................................................................4

Фрактал «Дерево Пифагора»................................................................................7

Модель проекта....................................................................................................10

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

I. Стек.

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

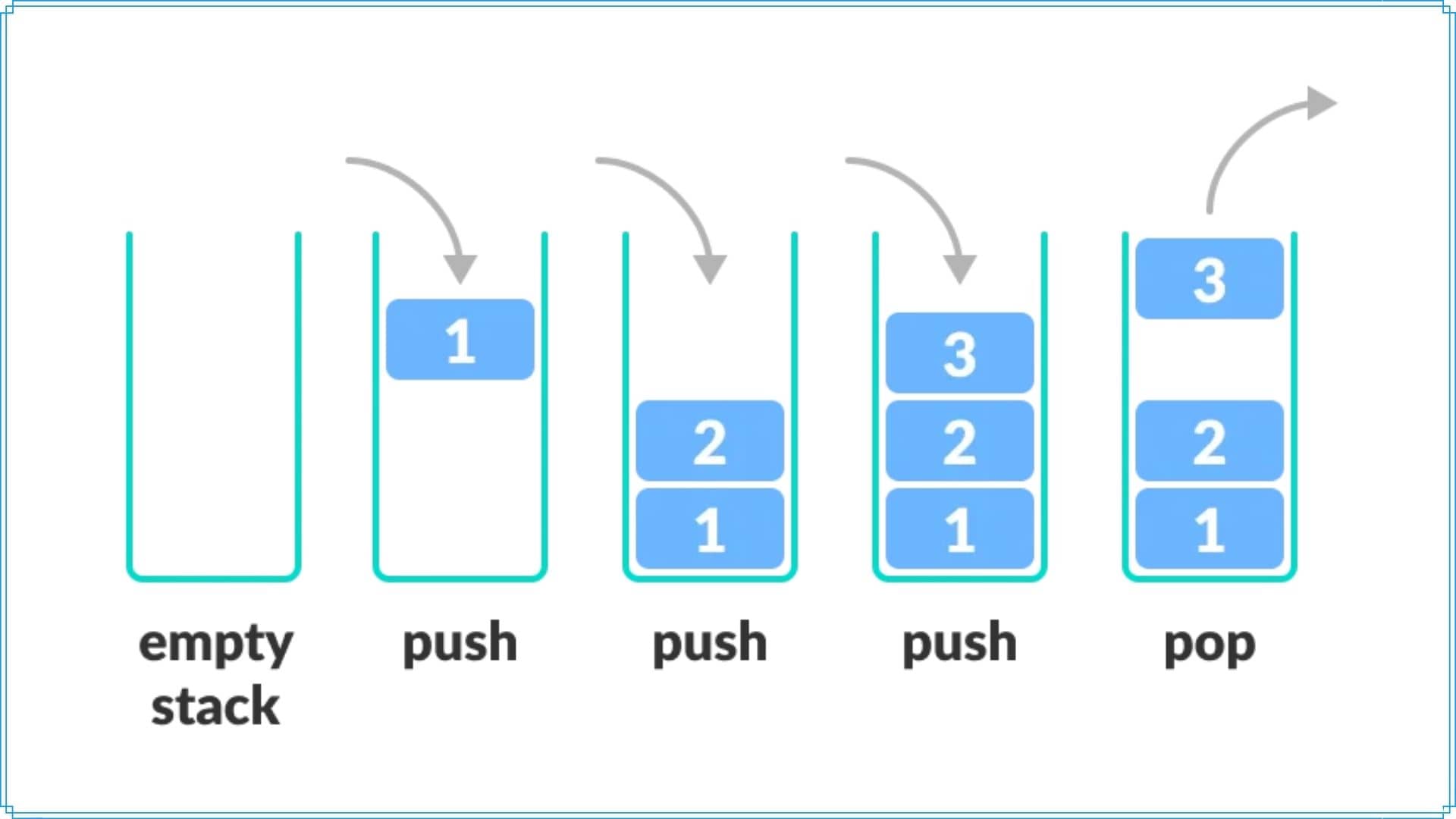


Рис 1.1. Принцип организации работы стека

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 (рис 1.2):

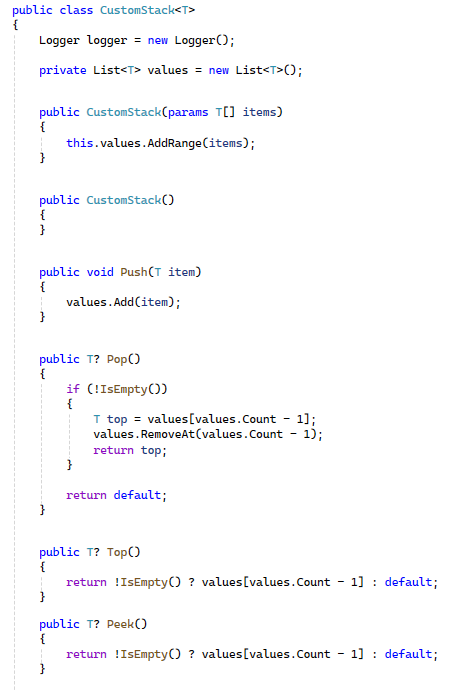
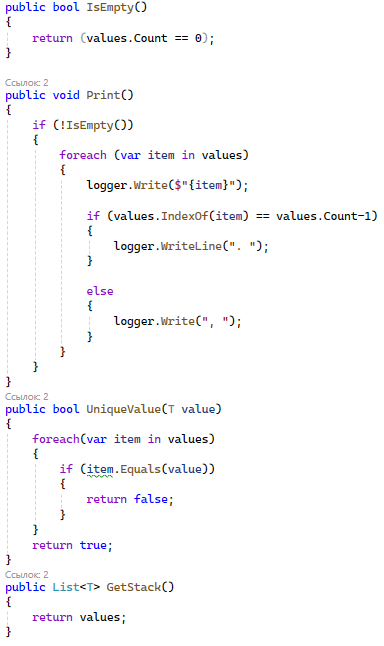


Рис 1.2.1. Программный код стека на языке C#.



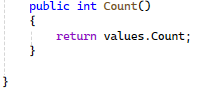


рис 1.2.2. Программный код стека на языке C#.

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#(рис 1.3):

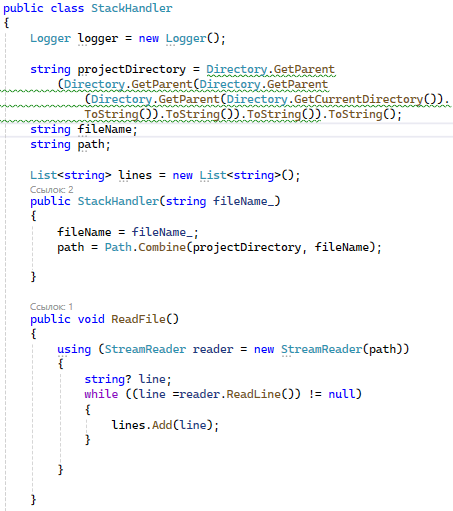


Рис 1.3.1. Программный код обработчика стека на языке C#.

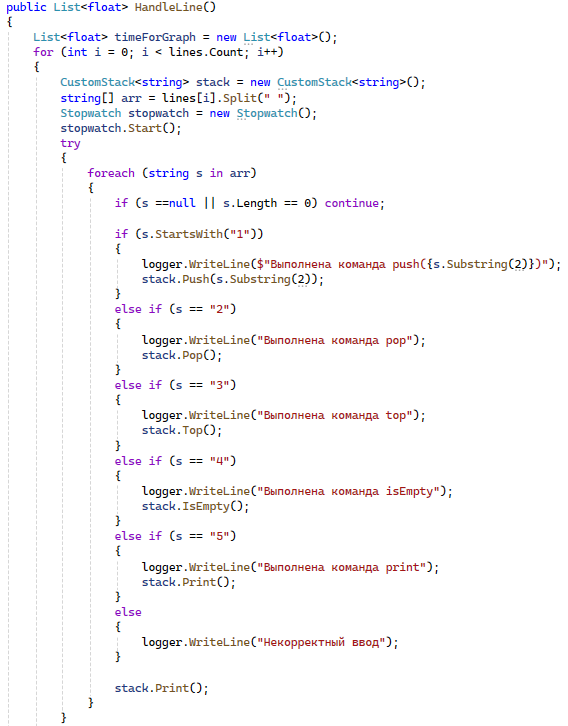


Рис 1.3.2. Программный код обработчика стека на языке C#



Рис 1.3.3. Программный код обработчика стека на языке C#

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.

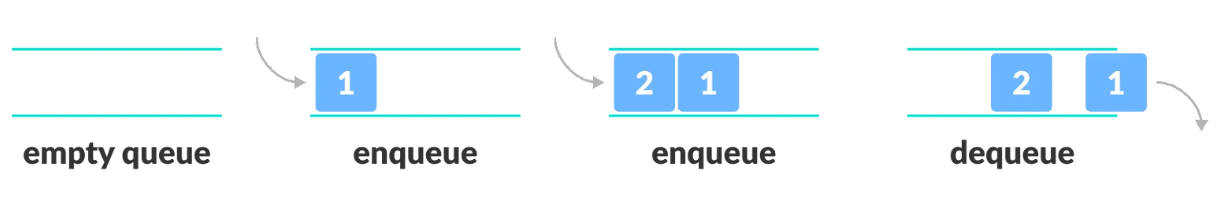


Рис 2.1. Как работает очередь в программировании.

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 (рис 2.2):

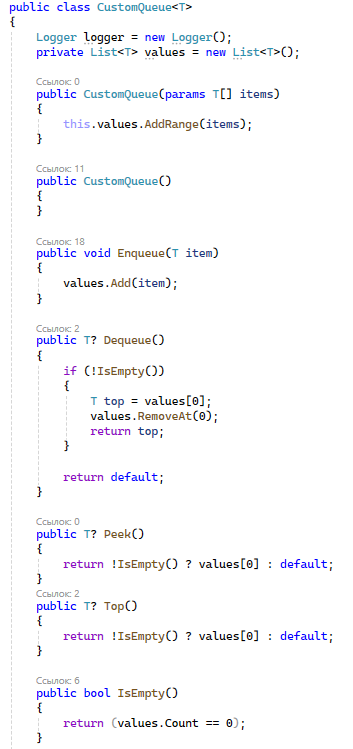


Рис 2.2.1. Реализация очереди на языке C#

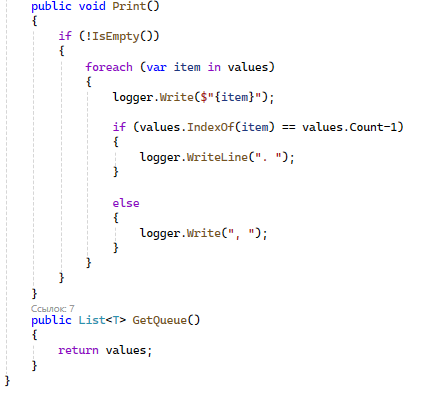


Рис 2.2.2. Реализация очереди на языке C#

Обработчик для Стека

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#(рис 2.3):

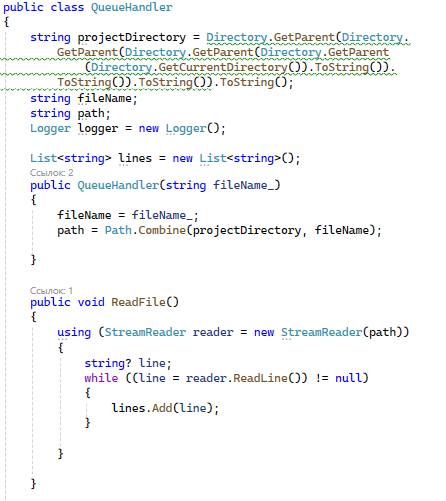


Рис 2.3.1. Реализация обработчика очереди на языке C#.

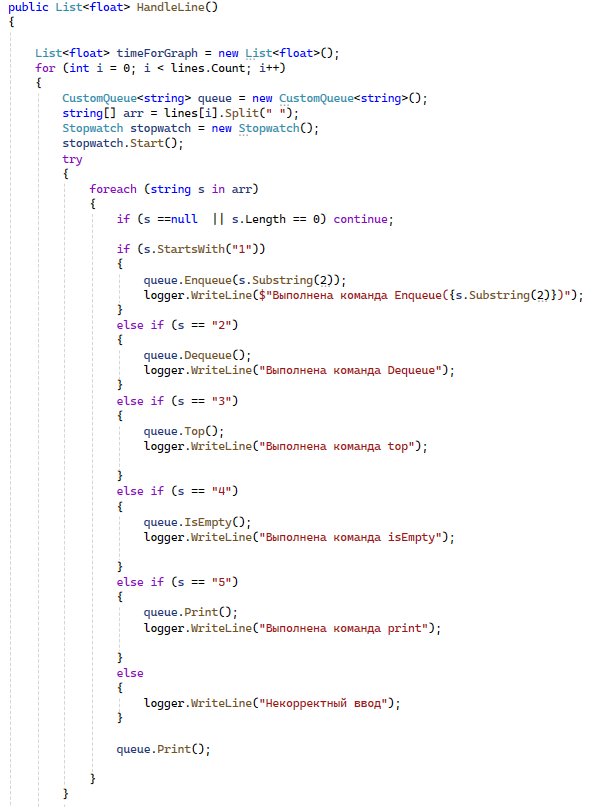


Рис 2.3.2. Реализация обработчика очереди на языке C#.



Рис 2.3.3. Реализация обработчика очереди на языке C#.