МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования  
**«Национальный исследовательский  
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Отчет по лабораторной работе**

**«Транслятор полиномов»**

**Выполнили:**

студенты группы 3823Б1ФИ1

Романов Артем

Поташник Максим

**Проверила:**

ассистент каф. ВВСП ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панова Е. А.

:

Нижний Новгород  
2025

# Содержание

[Введение 3](#_Toc198037350)

[1 Постановка задачи 4](#_Toc198037351)

[2 Руководство пользователя 4](#_Toc198037352)

[2.1 Установка 4](#_Toc198037353)

[2.1.1 Клонирование репозитория 4](#_Toc198037354)

[2.1.2 Сборка с помощью CMake 5](#_Toc198037355)

[2.1.3 Запуск в Visual Studio и выбор запускаемого проекта 7](#_Toc198037356)

[2.2 Описание возможностей консольного интерфейса 8](#_Toc198037357)

[2.2.1 Начало работы 8](#_Toc198037358)

[2.2.2 Описание общих возможностей калькулятора 8](#_Toc198037359)

[2.2.3 Операторы и функции 10](#_Toc198037360)

[2.2.4 Переменные 10](#_Toc198037361)

[2.2.5 Полиномы 11](#_Toc198037362)

[3 Руководство программиста 13](#_Toc198037363)

[3.1 Установка 13](#_Toc198037364)

[3.2 Описание структуры кода и программы 13](#_Toc198037365)

[3.3 Описание структур данных 16](#_Toc198037366)

[3.3.1 Упорядоченная таблица 16](#_Toc198037367)

[3.3.2 Неупорядоченная таблица 16](#_Toc198037368)

[3.3.3 AVL-дерево 16](#_Toc198037369)

[3.3.4 Красно-чёрное дерево 17](#_Toc198037370)

[3.3.5 Хеш-таблица с открытой адресацией 17](#_Toc198037371)

[3.3.6 Хеш-таблица с закрытой адресацией 17](#_Toc198037372)

[3.4 Описание алгоритмов 18](#_Toc198037373)

[3.4.1 Преобразование выражения в постфиксную запись 18](#_Toc198037374)

[3.4.2 Вычисление выражения в постфиксной записи 18](#_Toc198037375)

[Заключение 19](#_Toc198037376)

[Список литературы 20](#_Toc198037377)

[Приложение 21](#_Toc198037378)

# Введение

Современные вычислительные системы активно применяются для автоматизации обработки сложных математических выражений, включая полиномиальные уравнения. Одним из направлений, требующих детального изучения, является разработка программных средств, способных выполнять вычисления над полиномами с возможностью сохранения промежуточных результатов, работы с переменными и выбора подходящей структуры хранения данных.

При этом, в современных задачах поднимаются не только вопросы по работе с математическими выражениями, но и вопросы по хранению результатов. Для таких задач существует множество разных видов структур данных, таких как хэш-таблица и поисковое дерево.

# Постановка задачи

В рамках данной лабораторной работы была поставлена цель — создать транслятор полиномов, реализованный в виде консольного калькулятора, поддерживающего работу с числами и полиномами специального вида. Основной задачей стало проектирование интерпретатора, а также реализация и интеграция различных контейнеров данных, таких как хеш-таблицы с открытой и закрытой адресацией, AVL и красно-чёрное деревья, а также упорядоченные и неупорядоченные таблицы.

# Руководство пользователя

Данное руководство пользователя предназначено для ознакомления с функциональностью и принципами работы программы, позволяющей выполнять операции над числами и полиномами специального вида.

# Установка

Для корректной установки программы и её запуска требуется наличие следующего программного обеспечения:

* Microsoft Visual Studio 2022 или более поздняя версия,
* CMake 3.13 или более поздняя версия,
* Git.

Процесс установки описан ниже.

# Клонирование репозитория

Выберите директорию, в которой будет храниться папка проекта. Пусть в текущем примере это будет директория *dev*.

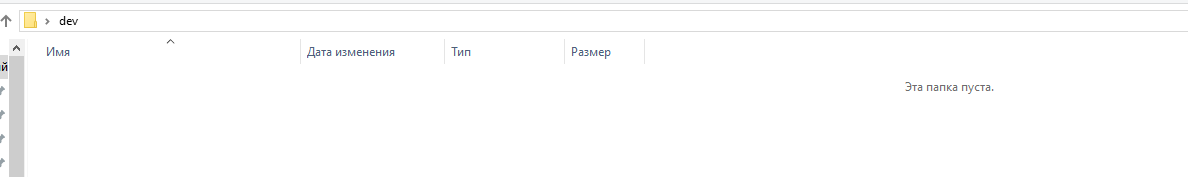


Рисунок 1 – пустая директория dev

Откройте приложение “консоль” в этой папке, введя *cmd* в строке с названием текущей директории.

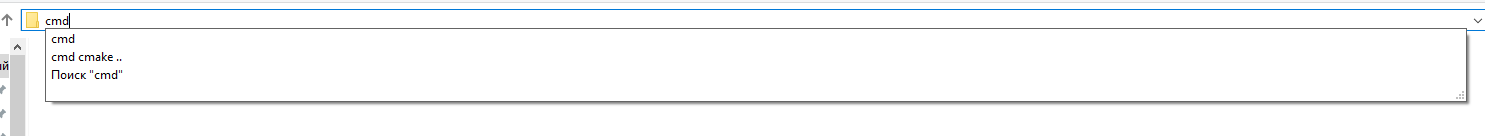


Рисунок 2 – открытие приложения “консоль”

Затем введите следующую команду:

*git clone* [*https://github.com/Quaha/mp2-lab5-PolynomialCalculater*](https://github.com/Quaha/mp2-lab5-PolynomialCalculater)

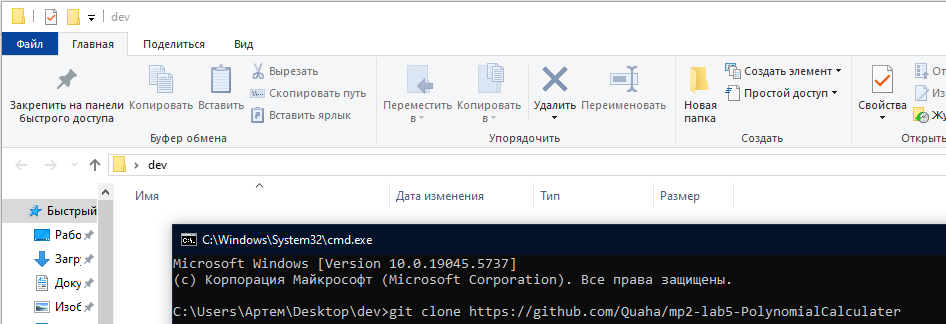
**

Рисунок 3 – клонирование репозитория

В текущей директории появится папка с репозиторием.

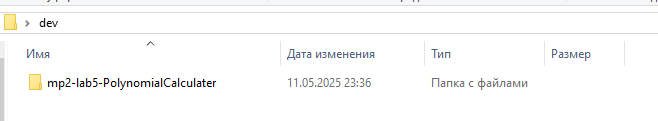


Рисунок 4 – клонированная директория проекта

# Сборка с помощью CMake

Для сборки проекта в его корневой директории следует создать папку *build*.

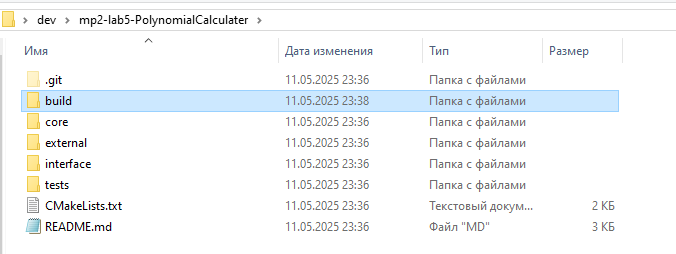


Рисунок 5 – корневая папка проекта

Перейдя в нее, запустите в ней приложение “консоль” (также, как в пункте 2.1.1), введите команду:

*cmake ..*

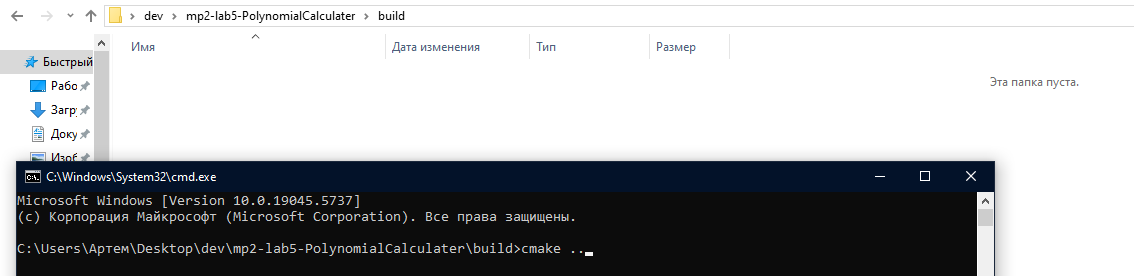


Рисунок 6 – команда для сборки с помощью CMake

Если ПО имеет подходящие версии, то в текущей папке появится схожий набор файлов:

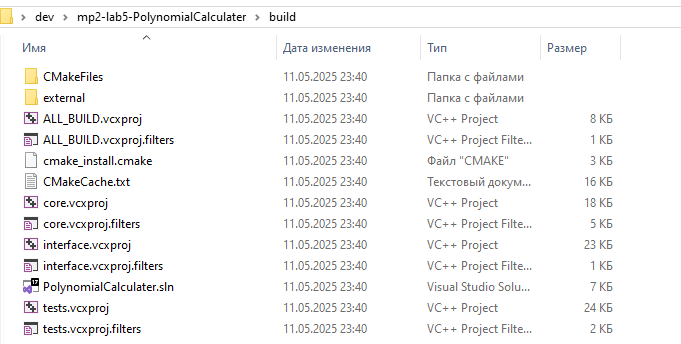


Рисунок 7 – файлы сборки проекта

### Запуск в Visual Studio и выбор запускаемого проекта

Для запуска проекта в Microsoft Visual Studio требуется открыть файл *PolynomialCalculater.sln* с помощью соответствующего приложения.

В “обозревателе решений” находим проект *interface.* Назначаем его в качестве запускаемого проекта, затем производим запуск без отладки.

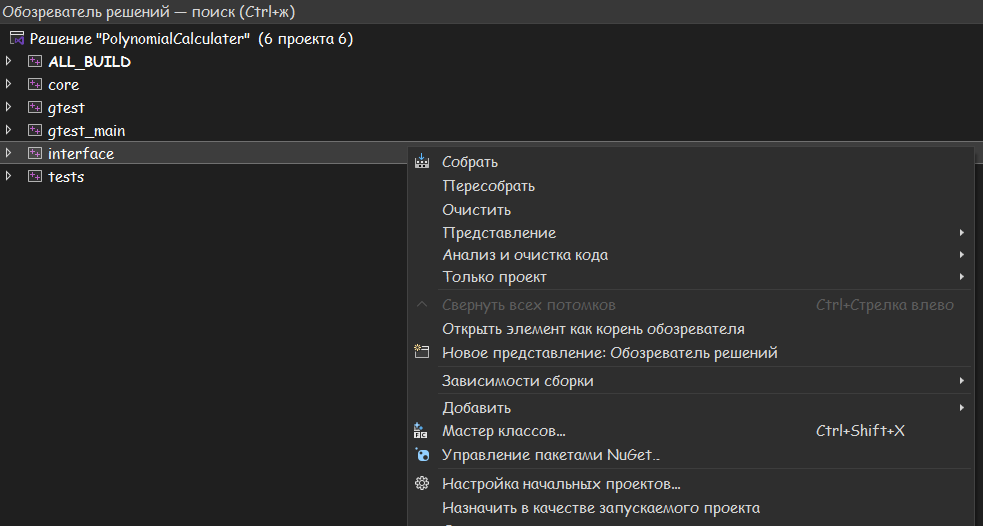


Рисунок 8 – проекты в обозревателе решений

# Описание возможностей консольного интерфейса

# Начало работы

При запуске программы пользователю предлагается выбрать тип контейнера. Программа представляет собой калькулятор полиномов, работающий в режиме интерпретатора, с возможностью записи переменных. Контейнер используется для их хранения, а также для хранения специальных технических данных.

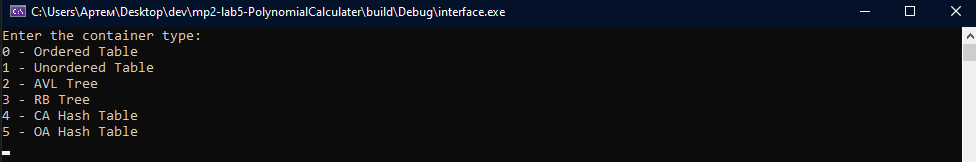


Рисунок 9 – начало работы с консольным интерфейсом

Выбор типа контейнера производится до тех пор, пока не будет верно указан номер контейнера.

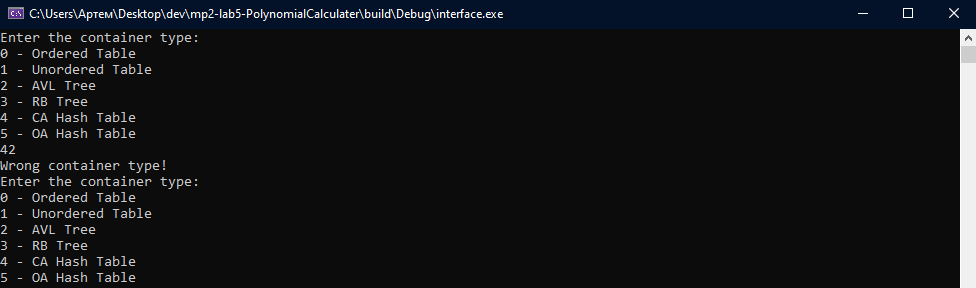


Рисунок 10 – некорректный выбор контейнера

При верном указании выводится пустая строка.

# Описание общих возможностей калькулятора

Калькулятор работает с числами и полиномами специального вида. Работа с числами может осуществляться без какого-либо взаимодействия с полиномами, при этом работа с полиномами организована схожим образом, поэтому для начала следует рассмотреть её. Числа внутри программы представляются типом *long double*, если не указано иное, поэтому в качестве результата любой операции следует ожидать с соответствующей точностью, характерной для этого типа.

Калькулятор фактически является интерпретатором. Каждая строка является отдельной командой. Также команду можно ограничить символом “***;***”, всё, что следует после него, никак не учитывается программой.

Каждая команда разбивается на токены и переводится в постфиксную запись, результат которой частично вычисляется сразу при анализе токенов.

В результате любой корректной команды в стеке постфиксной записи остаётся некоторое значение. Это значение всегда выводится пользователю интерпретатором.

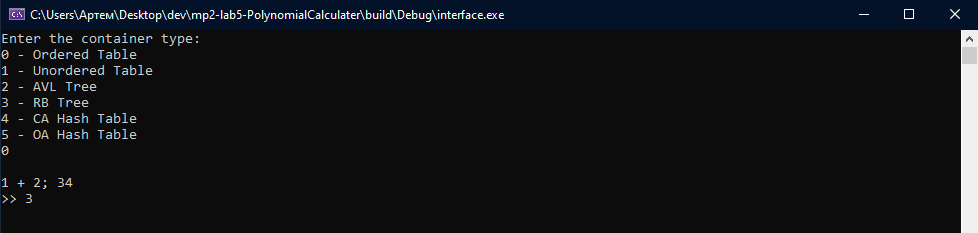


Рисунок 11 – пример вычислений с числами

В случае ввода некорректной команды, пользователю выводится сообщение об ошибке. Такая команда не выполняется, а работы программы продолжается.



Рисунок 12 – пример некорректной команды

Команда может содержать разделяющие символы, которые опускаются при её анализе интерпретатором, и нужны только для более удобного её чтения пользователем. К ним относится пробел “ “ и верхний апостроф “***`***”.



Рисунок 13 – использование разделителей

Действия над числами (а также и над полиномами) могут осуществляться с помощью использования операторов и функций.

# Операторы и функции

В данный момент поддерживаются следующие 4 оператора для работы с числами: “***+***”, “***-***”, “***\****”, “***/***”. Первые два из указанных операторов также имеют унарные версии. Вычисление выражений производится с учетом приоритета операторов, который соответствует принятому в математике порядку действий.



Рисунок 14 – команда с использованием операторов

Также имеется поддержка скобок “(”, “)”, которые позволяют управлять приоритетом вычислений.



Рисунок 15 – изменение приоритета выполнения

Существует поддержка функций. Функции имеют следующий вид: *name(param1, …, paramk)*, при этом число аргументов у отдельных функций может быть неограничено. В данный момент для чисел реализована одна функция *sum()*, которая принимает произвольное число параметров и возвращает их сумму (данная функция также реализована и для полиномов).



Рисунок 16 – использование встроенной функции

# Переменные

Программа поддерживает работу с переменными: их можно создавать, сохранять значения и получать их. Имя переменной может содержать буквы английского алфавита (в любом регистре), цифры и символ подчеркивания, но не может начинаться с цифры. Переменная инициализируется в момент создания с помощью оператора “***=***”, который присваивает ей значение, вычисленное из правого операнда. При этом сам оператор возвращает значение правого операнда.



Рисунок 17 – объявление переменной

Правым операндом может являться любая корректная команда, в частности этой командой может быть число или другая переменная.



Рисунок 18 – оператор присваивания

Использование неинициализированной переменной приводит к уведомлению об ошибке.



Рисунок 19 – использование неинициализированной переменной

Переменные могут быть удалены. Для этого существует функция *delVar(varname)*.

# Полиномы

Калькулятор поддерживает работу с полиномами специального вида, где каждый моном в нём имеет вид: ***x^a1y^a2z^a3***, где ***a1, a2, a3*** - целые числа. В самой программе степени мономов хранятся как целые числа с помощью типа *long long*, из-за чего использование больших степеней может привести к переполнению. Коэффициент при полиноме задается с помощью умножения полинома на число.



Рисунок 20 – пример полинома

Фактически любые числа в программе представлены как полиномы нулевой степени. При выводе таких объектов часть, связанная с полиномиальной формой, как правило, опускается, однако это поведение не гарантируется и может зависеть от конкретного контекста.



Рисунок 21 – полином нулевой степени

Над полиномами поддерживаются все те же операции, что и над числами, за исключением деления одного полинома на другой. При этом деление полинома на число допускается.



Рисунок 22 – операции с полиномами

Для полиномов реализована специальная функция *calcValue(poly, x, y, z)*, которая вычисляет значение полинома в выбранной точке.



Рисунок 23 – вычисление значения полинома в точке

# Руководство программиста

Данное руководство программиста предназначено для ознакомления со структурой, логикой и внутренними механизмами работы программного обеспечения, реализующего операции над полиномами специального вида. Данный раздел описывает ключевые модули, используемые алгоритмы и особенности реализации.

# Установка

Процесс установки полностью совпадает с установкой в руководстве пользователя (пункт 2.1).

# Описание структуры кода и программы

Корневая директория проекта организована следующим образом:

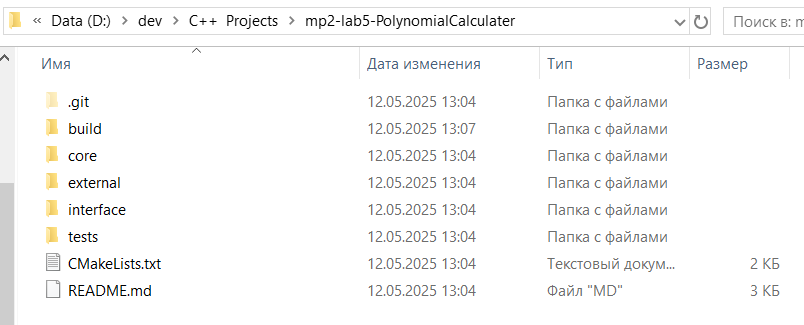


Рисунок 24 – корневая директория проекта

Корневая директория проекта организована следующим образом:

* *.git* — содержит служебные файлы системы контроля версий Git,
* *build* — директория для сборочных файлов, создаваемых системой CMake,
* *core* — основная часть проекта; содержит исходный код (.cpp и .hpp файлы),
* *external* — содержит исходные файлы сторонних библиотек, в данном случае файлы библиотеки googletest,
* *interface* — реализация консольного пользовательского интерфейса,
* *tests* — набор тестов для проверки корректности работы проекта

В директории core находятся две поддиректории:

* *src* — содержит .cpp-файлы с реализациями,
* *include* — содержит заголовочные файлы, а также реализацию шаблонных контейнеров, размещенных в поддиректории containers.

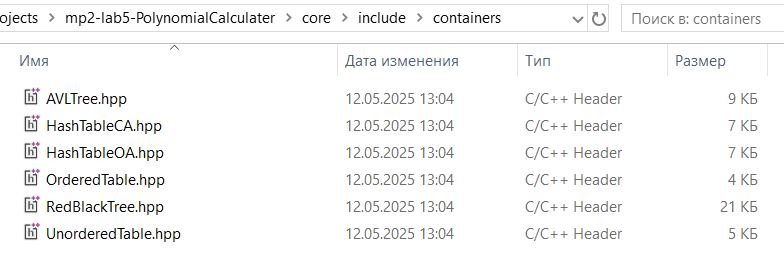


Рисунок 25 – файлы с реализациями контейнеров

Имеются реализации следующих структур данных:

* AVL-дерево,
* хеш-таблица с закрытой адресацией,
* хеш-таблица с открытой адресацией,
* упорядоченная таблица,
* красно-чёрное дерево, а также специальный класс для его тестирования,
* неупорядоченная таблица.

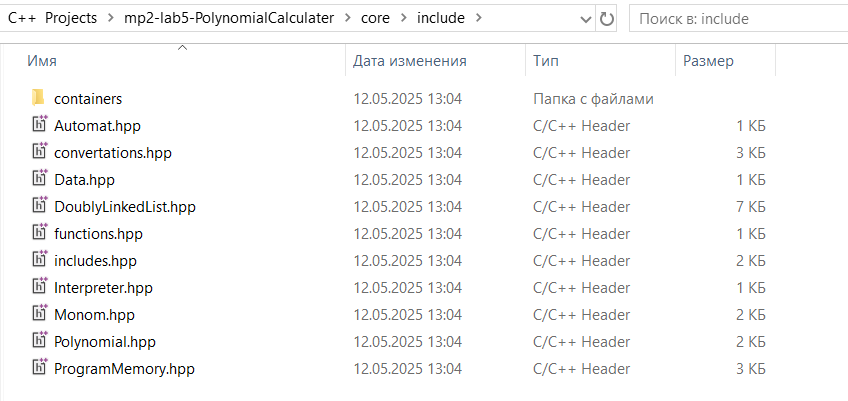


Рисунок 26 – заголовочные файлы проекта

Далее приведено описание основных файлов проекта:

* *Automat* — реализация класса конечного автомата, используемого для разбиения команды на токены;
* *convertations* — функции для преобразования строк в полиномы и числа, а также обратных преобразований;
* *Data* — реализация структуры, предназначенной для хранения значений переменных и внутренних промежуточных результатов;
* *DoublyLinkedList* — реализация двусвязного списка, используемого для хранения мономов полинома;
* *functions* — содержит реализацию операторов и функций, используемых в вычислениях;
* *includes* — содержит необходимые директивы #include, конструкции using и вспомогательные определения;
* *Interpreter* — реализация интерпретатора и связанных с ним внутренних анализаторов;
* *Monom* — реализация класса монома, который используется в составе класса полинома;
* *Polynomial* — реализация основного класса Polynomial;
* *ProgramMemory* — реализация глобальной памяти, отвечающей за выбор контейнера и доступ к данным.

# Описание структур данных

Все структуры данных поддерживают итераторы. В упорядоченных контейнерах они позволяют перебирать элементы в отсортированном порядке.

# Упорядоченная таблица

**Упорядоченная таблица** — это структура данных, представляющая собой массив, в котором элементы отсортированы по ключу. Она позволяет быстро находить элементы с помощью бинарного поиска за логарифмическое время. Вставка и удаление элементов выполняются за линейное время, так как для этих операций может потребоваться сдвиг элементов в массиве для поддержания порядка.

# Неупорядоченная таблица

**Неупорядоченная таблица** — это структура данных, представляющая собой массив, в котором элементы расположены в произвольном порядке. Отсутствие необходимости в сортировке элементов обеспечивает быструю вставку элементов за константное время, но при этом поиск и удаление работают за линейное время.

# AVL-дерево

**AVL-дерево** — это самобалансирующееся двоичное дерево поиска, в котором поддерживается баланс высот левого и правого поддеревьев каждого узла. Это обеспечивает логарифмическую сложность операций поиска, вставки и удаления в худшем случае.

Узлы дерева представлены особой структурой — Node (вершина дерева). В каждой вершине хранятся указатели на правого и левого потомка.

# Красно-чёрное дерево

**Красно-чёрное дерево** — это самобалансирующееся двоичное дерево поиска, в котором каждый узел имеет дополнительный цветовой атрибут: красный или чёрный. Структура поддерживает баланс с помощью набора правил, которые ограничивают порядок следования цветов по дереву. Благодаря этому обеспечивается логарифмическая сложность операций поиска, вставки и удаления в худшем случае.

Узлы дерева представлены особой структурой. Они хранятся в *std::vector*, что делает код безопасным с точки зрения памяти и ускоряет его, за счет локальности данных.

# Хеш-таблица с открытой адресацией

**Хеш-таблица с открытой адресацией** — это структура данных, использующая особые методы для разрешения коллизий. Каждый элемент размещается в ячейке, индекс которой вычисляется с помощью стандартной хэш-функции std::hash от ключа. Если несколько ключей попадают в одну и ту же ячейку, то следующему элементу выбирается иная свободная ячейка. Свободные ячейки перебираются за счёт итерирующей функции.

# Хеш-таблица с закрытой адресацией

Особенность **хеш-таблицы с закрытой адресацией** заключается в том, что, если несколько ключей попадают в одну и ту же ячейку, они хранятся вместе во вложенном векторе. При увеличении числа элементов таблица автоматически расширяется: создается новый массив большего размера, и все элементы перераспределяются согласно новым хешам. Это позволяет поддерживать эффективность операций, которые при равномерном распределении выполняются за амортизированное время O(1).

# Описание алгоритмов

# Преобразование выражения в постфиксную запись

Для преобразования инфиксного выражения (с обычным порядком операций) в постфиксную форму используется алгоритм преобразования выражения в постфиксную запись. Он учитывает приоритеты и ассоциативность операторов, а также корректно обрабатывает скобки. В процессе символы считываются последовательно, и операнды сразу добавляются в выходную очередь, тогда как операторы временно помещаются в стек.

# Вычисление выражения в постфиксной записи

Постфиксное выражение вычисляется с использованием стека: при встрече операнда он помещается в стек, а при встрече оператора — извлекаются необходимые операнды, над ними производится операция, и результат помещается обратно.

# Заключение

В результате выполнения лабораторной работы был успешно реализован транслятор полиномов — программное средство, позволяющее выполнять широкий спектр операций над алгебраическими выражениями. Калькулятор поддерживает работу с переменными, операциями и функциями, обеспечивает обработку полиномов специального формата и предоставляет удобный интерфейс для взаимодействия с пользователем.

Важной особенностью проекта стало внедрение нескольких типов контейнеров для хранения чисел и полиномов. Это дало возможность глубже изучить различные подходы к организации памяти, сравнить их эффективность, особенности реализации и поведение в различных сценариях. В частности, были реализованы и протестированы такие структуры, как AVL-дерево, красно-чёрное дерево, хеш-таблицы с открытой и закрытой адресацией, а также упорядоченные и неупорядоченные таблицы.

В процессе выполнения работы мы закрепили знания по работе с структурами данных, углубить понимание принципов построения интерпретаторов, парсинга выражений и реализации вычислительных алгоритмов. Результаты работы могут быть использованы как основа для создания более сложных систем компьютерной алгебры, а также для дальнейшего изучения тем, связанных с математическими выражениями, компиляторами, трансляцией выражений, структурами данных и оптимизацией хранения данных.

# Список литературы

1. Кормен Т. и др. Алгоритмы. Построение и анализ: [пер. с англ.]. – Издательский дом Вильямс, 2009.
2. Степанов И. Д. Алгоритмы и структуры данных (основной поток). Лекция 13. Красно-чёрное дерево [Электронный ресурс] / МФТИ. – URL:  https://www.youtube.com/watch?v=bEHUSrkATHA&t=5s (дата обращения: 11.05.2025).

# Приложение

1. Исходный код программы может быть найден на ресурсе GitHub по ссылке: <https://github.com/Quaha/mp2-lab5-PolynomialCalculater> (дата обращения: 11.05.2025)