Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Национальный исследовательский Университет

Институт Информационных Технологий Математики и Механики

**Отчёт по лабораторной работе**

**Создание класса для подсчёта значений полиномов, работы с различными таблицами.**

Выполнили:

студенты группы 3821Б1ПМ3

БогдашкинС.Е.

Еремеев В.П.

Проверил:

Заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2023 г.

Содержание

[Введение 3](#__RefHeading___Toc89757971)

[1. Постановка задачи 5](#__RefHeading___Toc89757972)

[2. Руководство пользователя 7](#__RefHeading___Toc39350_841689191)

[3. Руководство программиста 10](#__RefHeading___Toc89757977)

[a. Описание структуры программы 10](#__RefHeading___Toc89757978)

[b. Описание структур данных и алгоритмов 12](#__RefHeading___Toc89757979)

[4. Эксперименты 25](#__RefHeading___Toc89757984)

[Заключение 27](#__RefHeading___Toc89757985)

[Литература 28](#__RefHeading___Toc89757986)

# Введение

Программирование - это интересный, полезный и увлекательный процесс, благодаря которому мы можем с помощью специальных команд обучать компьютер, делать для нас разнообразные полезные задачи, от выполнения сложных математических операций с числами и различными уравнениями, расчёта уравнений, не поддаваемых аналитическому исследованию, создания сайтов и упрощения жизни людей с помощью карт и навигации, до автоматического управления самолетами, околоземными спутниками, позволяющими передавать информацию по всей планете, до управления космическими аппаратами, находящимися у других планет.

Чтобы программировать сложные алгоритмы, необходимо постоянно пополнять свои знания о структурах и методах изучаемого языка программирования. Одной из таких сложных и интересных структур в языке С++ являются классы. Это по сути инструмент для создания новых типов переменных, наряду с int, float, bool и т.д. Классы используются, когда нам необходимо описать множество схожих объектов, например, животных в зоопарке, у каждого из которых есть вес, рост, количество особей и т.д. Но неэффективно описывать каждого объекта по отдельности, гораздо проще создать структуру, которая будет содержать в себе данные о каждом объекте в целом. Также классы облегчают работу с разными математическими объектами, так как могут содержать в себе различные функции (методы), присущие каждому из объектов, которые могут выполнить нужную задачу, а также набор разных значений, существующих у объекта (поля). Для объектов класса также можно выполнить перегрузку разных стандартных операций, по сути указать программе как нужно действовать с объектами при выполнении арифметических или других операций, например, как выполнять скалярное произведение векторов. Это значительно упрощает работу с различными объектами описываемого класса. Также, если не понятно, с каким типом данных внутри класса придется работать используют шаблоны, вместо того чтобы кодировать много одинаковых функций, различающихся только типом данных. Шаблоны позволяют в процессе работы использовать нужный тип данных в работе класса. Для проверки используются тесты. Они позволяют убедиться, что написанные классы функционируют правильно, путём визуального (или автоматизировано при помощи Гугл-тестов) исследования работы программы на простых ситуациях, определяя места, в которых программа работает некорректно.

В данной лабораторной работе будут рассматриваться такие структуры данных, как разнообразные таблицы, которые хранят в себе значения по ключам, то есть каждому отдельному ключу (наименованию), будет соответствовать определённое значение. Такие таблицы могут быть отсортированы, для дальнейшего удобного использования в различных целях. Конкретно в этой лабораторной работе таблицы используются для хранения значений для членов алгебраических полиномов, а также самих полиномов, для более компактного их представления в программе. Значения полиномов должны быть вычисляемыми, при передаче соответствующих значений для членов полинома. Также должны быть продемонстрированы примеры работы со всеми таблицами, чтобы показать их работоспособность и научиться грамотно с ними работать для дальнейших проектов. По мере выполнения работы будут совершенствоваться навыки работы с такими базовыми элементами программирования, как шаблонные классы, таблицы, сортировки, ведь они являются теми знаниями, которые лежат в основе программирования.

# Постановка задачи

Используя шаблонные классы и перегрузки операций, разработать программу, выполняющую арифметические операции с полиномами трех переменных (х, у, z): сложение, вычитание, умножение на константу, умножение двух полиномов. Считается, что полином составлен из мономов от трех переменных с ограничением на степень каждой переменой от 0 до 9. Коэффициенты полинома - вещественные числа. Работоспособность программы необходимо проверить с помощью тестов. Кроме того, необходимо разработать пользовательское консольное приложение.

Особенности реализации:

* В качестве структуры хранения полинома использовать список мономов с ненулевыми коэффициентами. Односвязный список удобнее реализовать с фиктивной головой. Элементы списка хранить упорядоченными.
* Степень полинома хранить в «свернутом» виде, т. е. степень должна быть представлена как трехзначное число, где число сотен — это степень при переменной «х», число десятков - степень при переменной «у», число единиц - степень при переменной "г".
* Сложение полиномов осуществлять алгоритмом слияния упорядоченных массивов.
* Вычитание полиномов допускается выполнять через сложение с умножением на константу (С = А-В=А+(-1)\*B)
* При умножении и сложении (вычитании) необходимо следить, чтобы в итоговом полиноме были приведены подобные слагаемые и не хранилось мономов с нулевым коэффициентом.
* Если при умножении полиномов полученные степени переменных больше 9, выводить сообщение об ошибке.
* Считывание полинома у пользователя допускается в любом удобном виде. Необходимо предоставить пользователю правила ввода данных.
* Следует учесть, что пользователь может вводить полином, не упорядочив в нем мономы.
* Необходимо добавить свои файлы с реализацией программы.

После разработки класса полиномов необходимо разработать класс для вычисления арифметических выражений с помощью обратной польской записи и интегрировать в него класс полиномов. Чтобы была возможность принять полином за другое наименование, необходимо написать такую структуру, как таблицу, в которой по каждому ключу, имени переменной сможет находиться не только обычное число, но и целый полином, как значение. С помощью такой структуры реализовать интерфейс для пользователя, позволяющий производить арифметические действия с полиномами, а также вычислять их значения по некоторым начальным условиям. Также необходимо разработать классы для работы с другими видами таблиц, а именно… Последнее задание несёт в себе ознакомительный характер и нужно для освоения алгоритмов работы с различными видами таблиц и не требует интеграции в пользовательский интерфейс, для класса арифметических выражений прекрасно подойдёт самый простой вид таблицы.

Исходные файлы, содержащие описание классов должны быть вынесены в отдельную статистическую библиотеку, для последующей удобной работы с ними. Реализовать потоковые ввод и вывод для каждого класса, а также доступ к защищённым полям, а также провести практическую оценку времени работы программы на самых трудоёмких алгоритмах, и сравнить её с теоретически предполагаемой. К тому же, необходимо протестировать программу, чтобы убедиться, что все алгоритмы работают корректно. Программа должна быть написана на языке «С++».

# Руководство пользователя

В программе реализован интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы. В программе представлен в виде консольного интерфейса и предоставляет доступ к пользовательским операциям модулей программной системы.

Программа выполняет следующие операции:

Операции с полиномом:

* Сложение;
* Вычитание;
* Умножение;
* Деление;
* Нахождение значения в точке;
* Интегрирование;
* Дифференцирование.

Операции с постфиксной формой:

* Те же операции, что и у полинома, кроме нахождения значения в точке;
* Подразумевается использование скобок.

Операции с таблицами:

* Добавление;
* Поиск;
* Удаление.

В начале выводится меню выбора операций или с полиномами или с таблицами, или выход.



Рисунок 1. Начальное меню выбора.

Если мы выберем операции с полиномами, то у нас будет выбор или перейти к калькулятору выражений или вернуться в начальное меню.



Рисунок 2. Меню при выборе операций с полиномами.

При выборе калькулятора выражений нам надо ввести выражение без пробелов и разделителей (например, 2х5) или если пользователь хочет вернуться в начальное меню, то ввести 0.



Рисунок 3. Ввод числа в калькулятор.

После ввода выражения нам предлагают сохранить его, если мы хотим его сохранить, то надо написать Y, иначе любую другую клавишу.



Рисунок 5. Подтверждение сохранения выражения.

И нам теперь нужно дать имя нашему выражению, не используя буквы I,d,x,y,z, и используя максимум 16 символов.

 Рисунок 6. Имя выражения.

После мы переходим в операции с таблицами и нам предлагают: 1) Найти элемент 2) Вставить элемент 3) Удалить элемент 4) Выбрать таблицу 5) Вывести таблицу 0) Вернуться в главное меню.

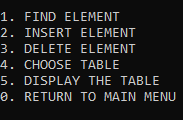


Рисунок 7. Операции с таблицами

При выборе таблиц мы можем выбрать 1) Несортированную таблицу 2) Сортированную таблицу 3)Таблицу списком 4) Красно-чёрное дерево 5) Хэш таблица 6)Таблица хэш списком

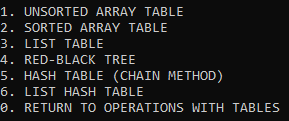


Рисунок 8. Выбор таблицы

И после выбора таблицы пользователь может вывести её.

 Рисунок 9. Несортированная таблица.

# Руководство программиста

## Описание структуры программы

Используемые инструменты

* Система контроля версий Git.
* Фреймворк для написания автоматических тестов Google Test.
* Среда разработки Microsoft Visual Studio.
* Утилита CMake для генерации проектов по сборке исходных кодов.

Общая структура проекта:

* gtest — библиотека Google Test.
* base — каталог с основным проектом ЛР.
* base\_test — каталог с проектом с модульными тестами.
* README.md — информация о проекте.
* mp2-lab6-algebra-polynom.sln - файл с решением (solution) для Microsoft Visual Studio.
* Служебные файлы
* .gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
* CMakeLists.txt — корневой файл для сборки проекта с помощью CMake.

Система классов для работы с полиномом:

* Полином должен быть от трёх переменных;
* Достаточно требовать от пользователя ввода коэффициентов в порядке сначала x, затем y, затем z;
* Пользователь вправе не вводить коэффициент 1 перед мономами, вводить мономы в случайном порядке, вводить сразу несколько мономов с одинаковыми степенями переменных;
* Полином должен быть организован на списке.

Система классов для работы с постфиксной формой:

* Возможность использования в выражениях имена полиномов;
* Проверка на корректность алгебраического выражения.

Система классов для работы с таблицами:

* Ключ в таблице — имя полинома;
* Достаточно консольное визуальное представление таблицы;
* Должны быть представлены следующие 6 таблиц:
* Хэш-таблица (с открытым перемешиванием и со списками);
* Неупорядоченная таблица (на массиве и на списке);
* Упорядоченная таблцица (на массиве и на дереве поиска).

Общее требование — разумное время работы алгоритмов.

## Описание структур данных и алгоритмов

Моном

Содержит коэффициент и степень. Представляется в виде строки со знаком "+" или "-", цифрами и буквами x, y, z. Степени не выделяются отдельно символом "^".

Полином

Строится на основе списка (может либо содержать начальное звено, либо список мономов). Таким образом, реализуется однонаправленный линейный список. Представление аналогично представлению монома за исключением того, что знаков "+" и "-" может быть много.

Алгоритмы, используемые с мономами и полиномами Мономы в данном случае рассматриваются как тривиальные полиномы.

* Сложение. Реализуется при помощи слияния двух отсортированнх списков.
* Вычитание. Реализуется при помощи слияния двух отсортированных списков полиномов (применяется инверсия коэффициентов).
* Умножение. Реализутся при помощи слияния множества (по числу мономов во втором полиноме) отсортированных списков полиномов.
* Деление. Реализуется при использовании алгоритма деления "уголком".
* Дифференцирование. Ненулевая степень уменьшается на 1, коэффициент умножается на показатель соответствующей степени. Моном с нулевой степенью удаляется.

|  |
| --- |
| Polynom Polynom::derivative(char param)  {  Polynom tmp;  Monom tp;  for (List<Monom>::iterator it = polynom.begin();  it != polynom.end(); ++it)  {  tp = \*it;  tp.derivative(param);  if (tp.derivative(param).get\_coef()!=0)  tmp.add\_monom(tp.derivative(param));  }  return tmp;  } |

Фрагмент кода 1.

* Интегрирование. Соответствующая степень увеличивается на 1, а коэффициент делится на эту соответствующую степень.

|  |
| --- |
| Polynom Polynom::integral(char param)  {  Polynom tmp;  Monom tp;  for (List<Monom>::iterator it = polynom.begin();  it != polynom.end(); ++it)  {  tp = \*it;  tp.integral(param);  if (tp.integral(param).get\_coef() != 0)  tmp.add\_monom(tp.integral(param));  }  return tmp;  } |

Фрагмент кода 2.

* Нахождение значения в точке. У каждого монома высчитывается при помощи умножения и возведения в степень, затем все полученные результаты складываются.

|  |
| --- |
| double Polynom::calculate\_in\_point(double x, double y, double z)  {  if (polynom.get\_size() == 0) {  throw EqException(error\_codes::k\_CALCULATE\_EMPTY\_EXPRESSION);  }    int a, b, c;  double result = 0;  for (List<Monom>::iterator it = polynom.begin(); it != polynom.end(); ++it) {  c = (\*it).get\_deg() % BASE;  b = (((\*it).get\_deg() - c) / BASE) % BASE;  a = ((((\*it).get\_deg() - b - c) / BASE) / BASE);  result += (\*it).get\_deg() \* pow(x, a) \* pow(y, b) \* pow(z, c);  }  return result;  } |

Фрагмент кода 3.

Таблица

Содержит структуру хранения строк таблицы. Визуально представляется в виде символов псевдографики, доступных в ASCII. Особенности различных таблиц:

*Неупорядоченная таблица на массиве*

Алгоритмы:

* Поиск. Проход по массиву, пока не встретит совпадение ключа.
* Добавление. Проверяет, не существует ли уже строка с данным ключом, затем вставляет в конец.
* Удаление. Меняет местами последний элемент и удаляемый, а затем удаляет последний.

*Неуподрядоченная таблица на списке*

Алгоритмы:

* Поиск. Проход по списку, пока не встретит совпадение ключа.

|  |
| --- |
| TableBody\* ArrayTable::find(const std::string& key)  {  TableString\* tmp = (\*this).find\_str(key);  if (tmp == nullptr) {  return nullptr;  }  else {  return &tmp->body;  }  } |

Фрагмент кода 4.

* Добавление. Проверяет, не существует ли уже строка с данным ключом, затем вставляет либо только в начало (если список односвязный), либо или в начало, или в конец (если список двусвязный).

|  |
| --- |
| bool ArrayTable::insert(const std::string& key, TableBody& data)  {  if (is\_full()) {  memory\_allocator();  }  TableString\* tmp = find\_str(key);  if (tmp == nullptr) {  tbl[data\_cnt] = new TableString(key, data);  data\_cnt++;  return true;  }  else {  return false;  } |

Фрагмент кода 5.

* Удаление. Стандартное удаление звена списка.

|  |
| --- |
| bool ArrayTable::erase(const std::string& key)  {  TableString\* tmp = find\_str(key);  if (tmp == nullptr) {  return false;  }  else {  tbl[curr\_pos] = tbl[data\_cnt - 1];  tbl[--data\_cnt] = nullptr;  }  return true;  } |

*Фрагмент кода 6.*

*Упорядоченная таблица на массиве.*

Элементы в упорядоченных таблицах располагаются в некотором определенном порядке, задаваемым относительным расположением ключей. Таблица будет организована с помощью статического массива, поскольку для динамических упорядоченных таблиц при включении и удалении элементов приходится выполнять реорганизацию таблиц сразу при выполнении операций, что увеличивает время выполнения этих операций.

Алгоритмы:

* Поиск элемента. В упорядоченной таблице элементы располагаются по возрастанию значений ключей, что позволяет делать некоторые предположения о месте размещения искомого элемента таблицы. Если, например, нам требуется найти элемент с ключом k\*, и нам известно, что в i-ом элементе таблицы ключ ki>k\*, это означает, что искомый элемент может находиться среди первых (i-1)-го элементов таблицы. Если же ki<k \*, то искомый элемент может находиться среди последних элементов таблиц, начиная с (i+1)-го элемента. На этом основан алгоритм двоичного поиска. В соответствии с алгоритмом двоичного поиска определяется индекс записи, находящейся в середине таблицы. Сравнение искомого ключа и ключа найденного элемента позволяет определить (если они не равны), в какой половине таблицы – верхней или нижней – может находиться искомый элемент. В соответствии с принятым решением ненужная половина таблицы, включая и найденную запись, исключается из рассмотрения, в результате чего количество элементов, которые нужно проанализировать, сокращается вдвое. Такое сокращение может происходить до тех пор пока в одной из частей не отсанется единтсвенного нужного нам элемента. Таким образом, через некоторое количество попыток запись с требуемым ключом, если она имеется в таблице, будет найдена. Если же требуемая запись отсутствует в таблице, через некоторое количество попыток все элементы таблицы будут исключены из рассмотрения.

|  |
| --- |
| TableString\* SortTable::find\_str(const std::string& key)  {  int i, i1 = 0, i2 = data\_cnt -1;    while(i1 <= i2) {  i = (i1 + i2) >> 1;    if (tbl[i]->key == key) {  i1 = i + 1;  i2 = i;  break;  }    if (tbl[i]->key > key) {  i2 = i - 1;  }  else {  i1 = i + 1;  }  }  if ((i2 < 0) || (tbl[i2]->key < key)) {  i2++;  }  curr\_pos = i2;  if ((i2 < data\_cnt) && (tbl[i]->key == key)) {  return tbl[i2];  }  return nullptr; |

Фрагмент кода 7.

* Добавление элемента. При включении нового элемента в таблицу удобно использовать алгоритм сортировки вставками. При этом если в таблице не могут находиться элементы с одинаковыми значениями ключей, сначала следует выполнить поиск элемента и, в случае неуспешного поиска, вставить в таблицу новый элемент. При использовании сортировки вставками сравнение ключей элементов осуществляется от конца таблицы к ее началу. Пока очередной элемент таблицы имеет ключ, превышающий ключ включаемого элемента, этот элемент таблицы копируется в следующую за ним позицию таблицы. Как только ключ очередного элемента таблицы окажется меньше ключа нового элемента, новый элемент включается после найденного. Если вся таблица будет просмотрена, новый элемент будет включен в первую позицию таблицы.

|  |
| --- |
| bool SortTable::insert(const std::string& key, TableBody& data)  {  if (is\_full()) {  throw (EqException(error\_codes::k\_OUT\_OF\_MEMORY));  }    TableString\* tmp = find\_str(key);  if (tmp != nullptr) {  return false;  }  else {  for (int i = data\_cnt; i > curr\_pos; i--) {  tbl[i] = tbl[i-1];  }  tbl[curr\_pos] = new TableString(key, data);  data\_cnt++;  return true;  } |

Фрагмент кода 8.

* Удаление элемента. Удаление элемента включает в себя операцию поиска. После нахождения элемента осуществляется непосредственно удаление, при этом структура таблицы не должна быть нарушена. Поэтому требяется реорганизация таблицы, которая в случае таблицы на статическом массиве реализуется достаточно просто: в результате поиска элемента получаем его индекс в таблице, после чего нижняя половина таблицы перемещается на одну позицию вверх.

|  |
| --- |
| bool SortTable::erase(const std::string& key)  {  TableString\* tmp = find\_str(key);  if (tmp == nullptr) {  return false;  }  else {  for (int i = curr\_pos; i < data\_cnt; i++) {  tbl[i] = tbl[i + 1];  }  tbl[--data\_cnt] = nullptr;  return true;  }  } |

Фрагмент кода 9.

*Таблица на Красно-черном дереве.*

Красно-чёрным называется бинарное поисковое дерево, у которого каждому узлу сопоставлен дополнительный атрибут — цвет и для которого выполняются следующие свойства:

* Каждый узел промаркирован красным или чёрным цветом
* Корень и конечные узлы (листья) дерева — чёрные
* У красного узла родительский узел — чёрный
* Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов
* Чёрный узел может иметь чёрного родителя

Алгоритмы:

* Добавление. Каждый элемент вставляется вместо листа, поэтому для выбора места вставки идёт от корня до тех пор, пока указатель на следующего сына не станет нулевым (то есть этот сын — лист). Вставляет вместо него новый элемент с нулевыми потомками и красным цветом. Теперь проверяет балансировку. Если отец нового элемента черный, то никакое из свойств дерева не нарушено. Если же он красный, то нарушается свойство 3, для исправления достаточно рассмотреть два случая: 1) "Дядя" этого узла тоже красный. Тогда, чтобы сохранить свойства 3 и 4, просто перекрашивает "отца" и "дядю" в чёрный цвет, а "деда" — в красный. 2) "Дядя" чёрный. Выполняет поворот. Если добавляемый узел был правым потомком, то необходимо сначала выполнить левое вращение, которое сделает его левым потомком.

|  |
| --- |
| void RedBlackTree::insert\_in(RBTNode\* &root, RBTNode\* node)  {  RBTNode \*x = root;  RBTNode \*y = nullptr;  while (x != nullptr)  {  y = x;  if (node->data->key > x->data->key) {  x = x->right;  }  else {  x = x->left;  }  }  node->parent = y;  if(y!=nullptr) {  if (node->data->key > y->data->key) {  y->right = node;  }  else {  y->left = node;  }  }  else {  root = node;  }  node->color = Red;  insert\_fix\_up(root, node);  }; |

Фрагмент кода 10.

* Удаление вершины. При удалении красной вершины свойства дерева не нарушаются, то восстановление балансировки потребуется только при удалении чёрной. Могут возникнуть три случая в зависимости от количества детей удаляемой вершины: 1) Если у вершины нет детей, то изменяем указатель на неё у родителя на nil. 2) Если у неё только один ребёнок, то делаем у родителя ссылку на него вместо этой вершины. 3) Если же имеются оба ребёнка, то находим вершину со следующим значением ключа. У такой вершины нет левого ребёнка (так как такая вершина находится в правом поддереве исходной вершины и она самая левая в нем, иначе бы мы взяли ее левого ребенка. Удаляет скопировав её ключ в изначальную вершину.

|  |
| --- |
| void RedBlackTree::remove(RBTNode\*&root, RBTNode\*node)  {  RBTNode \*child, \*parent;  RBTColor color;  if (node->left != nullptr && node->right != nullptr) {  RBTNode \*replace = node;  replace = node->right;  while (replace->left != nullptr) {  replace = replace->left;  }  if (node->parent != nullptr) {  if (node->parent->left == node) {  node->parent->left = replace;  }  else {  node->parent->right = replace;  }  }    else {  root = replace;  }    child = replace->right;  parent = replace->parent;  color = replace->color;    if (parent == node) {  parent = replace;  }  else {  if (child != nullptr) {  child->parent = parent;  }  parent->left = child;    replace->right = node->right;  node->right->parent = replace;  }  replace->parent = node->parent;  replace->color = node->color;  replace->left = node->left;  node->left->parent = replace;  if (color == Black) {  remove\_fix\_up(root, child, parent);  }    delete node;  data\_cnt--;  return;  }  if (node->left != nullptr) {  child = node->left;  }  else {  child = node->right;  }    parent = node->parent;  color = node->color;  if (child) {  child->parent = parent;  }    if (parent) {  if (node == parent->left) {  parent->left = child;  }  else {  parent->right = child;  }  }  else {  RedBlackTree::root = child;  }    if (color == Black) {  remove\_fix\_up(root, child, parent);  }  data\_cnt--;  delete node;  } |

Фрагмент кода 11.

* Поиск. Для каждого узла начиная с корня сравнивает значение его ключа с искомым ключом. Если ключи одинаковы, алгоритм закончен, в противном случае функция вызывается рекурсивно для левого или правого поддерева.

|  |
| --- |
| TableString\* RedBlackTree::find\_str(const std::string& key)  {  RBTNode\* tmp = search(root, key);  if (tmp != nullptr) {  return tmp->data;  }  else {  return nullptr;  }    } |

Фрагмент кода 12.

Постфикс для полиномов

Для расчета арифметических операций с полиномами используется постфиксная форма. Она реализуется при помощи стека.

Алгоритм перевода инфиксной формы в постфиксную для полиномов:

1. Перебираем инфиксную запись слева направо.
2. Если входящий элемент число, то он добавляется в постфикс. Если входящий элемент операция (+, -, \*, /) то происходит проверка:
3. Если стек операций пуст или содержит левую скобку в вершине, то входящий элемент добавляется в стек.
4. Если входящий элемент имеет более высокий приоритет чем вершина, он помещается в стек.
5. Если входящий элемент имеет более низкий или равный приоритет, чем вершина, стек выгружается в постфикс, пока не обнаружится элемент с меньшим приоритетом или левая скобка на вершине стека, затем входящий элемент добавляется в стек.
6. Если входящий элемент является левой скобкой, он помещается в стек.
7. Если входящий элемент является правой скобкой, стек выгружается в постфикс, до тех пор пока на вершине стека не окажется левая скобка, которая удаляется из стека.
8. В конце инфиксной записи все элементы, оставшиеся в стеке, выгружаются в постфиксную запись.

# Эксперименты

Были реализованы тесты ко всем видам таблиц.

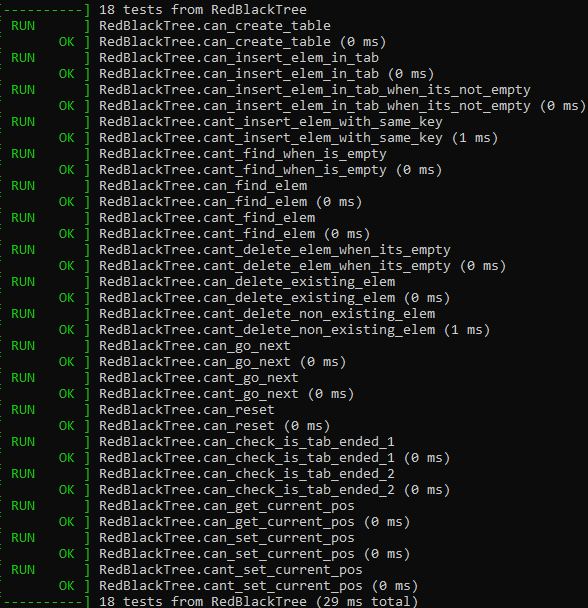


Рисунок 10. Работа тестов для красно-чёрного дерева.

По итогу все тесты выполнились за 281 миллисекунду.

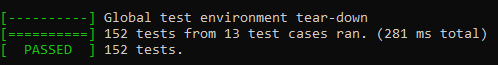
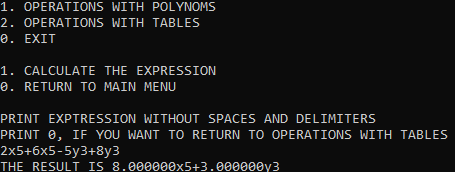


Рисунок 11. Время выполнения всех тестов

Калькулятор успешно выполняет операции с выражениями.



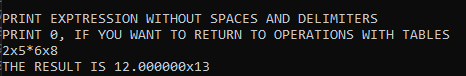


Рисунок 12.Работа калькулятора.

А теперь посмотрим разницу в выводах таблиц.



Рисунок 13. Несортированная таблица.



Рисунок 14. Хэш-таблица

# Заключение

Разработана программа, выполняющую арифметические операции с полиномами трех переменных (х, у, z) (сложение, вычитание, умножение на константу, умножение двух полиномов). В ходе разработки выполнены и учтены все особенности реализации:

* В качестве структуры хранения полинома использован список мономов с ненулевыми коэффициентами
* Степень полинома хранить в «свернутом» виде
* Сложение полиномов осуществлено алгоритмом слияния упорядоченных массивов.
* Вычитание полиномов допускается выполнять через сложение с умножением на константу (С = А-В=А+(-1)\*B)
* При умножении и сложении (вычитании) в итоговом полиноме были приведены подобные слагаемые и не хранятся мономы с нулевым коэффициентом.
* Если при умножении полиномов полученные степени переменных больше 9, выводить сообщение об ошибке.
* Считывание полинома у пользователя допускается в любом удобном виде.
* Учитано, что пользователь может вводить полином, не упорядочив в нем мономы.

Также разработан класс для вычисления арифметических выражений с помощью обратной польской записи и интегрировать в него класс полиномов. Написана таблица, в которой по каждому ключу, имени переменной сможет находиться не только обычное число, но и целый полином, как значение. С помощью неё реализован интерфейс для пользователя, позволяющий производить арифметические действия с полиномами, а также вычислять их значения по некоторым начальным условиям. Также необходимо разработанны классы для работы с другими видами таблиц.

Исходные файлы, содержащие описание классов вынесены в отдельную статистическую библиотеку, для последующей удобной работы с ними. Реализовать потоковые ввод и вывод для каждого класса, а также доступ к защищённым полям, а также проведена практическая оценка времени работы программы на самых трудоёмких алгоритмах.

# Литература

1. Бьерн Страуструп. Язык программирования С++ – Режим доступа: <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>
2. Т.А. Павловская Учебник по программированию на языках высокого уровня(С/С++) – Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>