МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

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

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

«Аналитические преобразования полиномов

от нескольких переменных

(списки)»

**Выполнил:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Холин К.И./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2023

Содержание

[1 Введение 4](#_Toc163506229)

[2 Постановка задачи 5](#_Toc163506230)

[3 Руководство пользователя 6](#_Toc163506231)

[3.1 Приложение для демонстрации работы узла списка 6](#_Toc163506232)

[3.2 Приложение для демонстрации работы списка 7](#_Toc163506233)

[3.3 Приложение для демонстрации работы мономов 9](#_Toc163506234)

[3.4 Приложение для демонстрации работы полиномов 11](#_Toc163506235)

[4 Руководство программиста 16](#_Toc163506236)

[4.1 Описание алгоритмов 16](#_Toc163506237)

[4.1.1 Шаблонная структура Node 16](#_Toc163506238)

[4.1.2 Шаблонный класс TList 16](#_Toc163506239)

[4.1.3 Шаблонный класс TRingList 25](#_Toc163506240)

[4.1.4 Класс TMonom 25](#_Toc163506241)

[4.1.5 Класс TPolynom 27](#_Toc163506242)

[4.2 Описание программной реализации 33](#_Toc163506243)

[4.2.1 Описание класса Node 33](#_Toc163506244)

[4.2.2 Описание класса TList 34](#_Toc163506245)

[4.2.3 Описание класса TRingList 37](#_Toc163506246)

[4.2.4 Описание класса TMonom 39](#_Toc163506247)

[4.2.5 Описание класса TPolynom 40](#_Toc163506248)

[5 Заключение 43](#_Toc163506249)

[6 Литература 44](#_Toc163506250)

[7 Приложения 45](#_Toc163506251)

[7.1 Приложение А. Реализация класса Node 45](#_Toc163506252)

[7.2 Приложение Б. Реализация класса TList 46](#_Toc163506253)

[7.3 Приложение C. Реализация класса TRingList. 52](#_Toc163506254)

[7.4 Приложение D. Реализация класса TMonom. 53](#_Toc163506255)

[7.5 Приложение E. Реализация класса TPolynom. 57](#_Toc163506256)

# Введение

Вам, может быть, известно, что в некоторых типовых контейнерах языка c++ более удобным и правильным способом расположения элементов в памяти является непрерывное хранение элементов в памяти. Это позволяет,например, не перепаковывать элементы при вставке и удалении.

Примерами контейнеров, работающих по непрерывному размещению элементов памяти, могут послужить контейнер std::vector, std::deque, std::forward\_list. Нас же интересует односвязный список.

Односвязный список(forward\_list) до сих пор остаётся актуальным вариантом выбора структуры данных. Например, односвязный список применим в реализации хэш-таблиц и полиномов. Именно forward\_list позволяет эффективно структурировать данные полинома(коэффициенты и степени мономов), а также значительно ускорить выполнение программы, так как операции с полиномами могут быть трудоёмкими и требовать большие затраты по памяти, но доступ к элементу затрудняется.

Поэтому в данной лабораторной работе мы будем использовать такую структуру данных, как линейный односвязный список, чтобы продолжить работу с такими арифметическими выражениями, как полиномы.

# Постановка задачи

Цель: реализовать классы для работы с линейными односвязными списками и разработать класс полиномы.

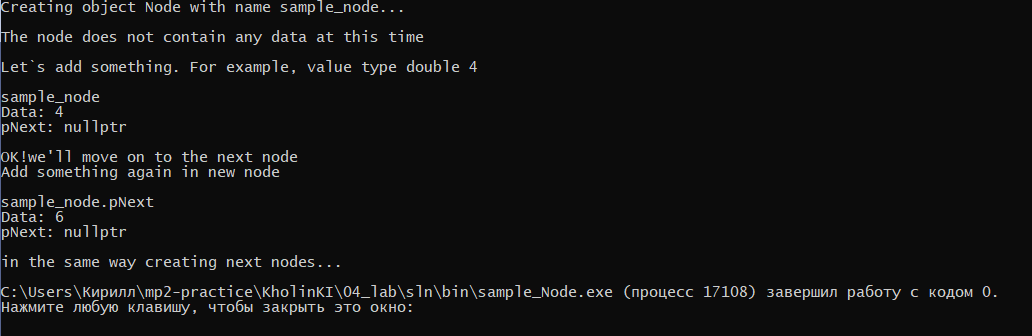
Задачи:

1. Разработать шаблонный класс Node(узел списка): структура узла.
2. На основании структуры узла реализовать шаблонный класс TList, который должен поддерживать следующие операции: поиск узла, вставка в начало списка, в конец, до, после, упорядоченная вставка, удаление узла, подсчёт узлов списка. Также следует добавить ко всему выше специфические операции: проверка на полноту, на пустоту, на конец списка. Навигационные методы: получение текущего узла, переход к следующему узлу списка, переход в начало списка.
3. Дополнительно создать шаблонный класс TRingList(кольцевой список), поддерживающий те же самые операции списка, но с некоторым отличием.
4. Реализовать класс TMonom(мономы), который будет являться параметром шаблонного класса TList.
5. В качестве финальной задачи разработать класс TPolynom на основе списков и мономов.

# Руководство пользователя

## Приложение для демонстрации работы узла списка

1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (рис. 1).

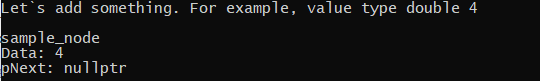


1. Основное окно программы
2. В первых двух строчках содержится информации о создании объекта Node, который на данный момент не содержит никаких данных (рис. 2).

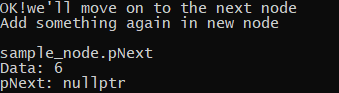


1. Создание пустого узла
2. Далее в качестве тестовых данных предлагается инициализировать первый узел.

Информация о текущем узле приводится, как показано на (рис. 3).



1. Инициализация узла
2. Инициализируется указатель на следующий узел. После полю Data созданного узла присваивается значение (рис. 4).



1. Инициализация указателя на следующий узел

и добавление даннных

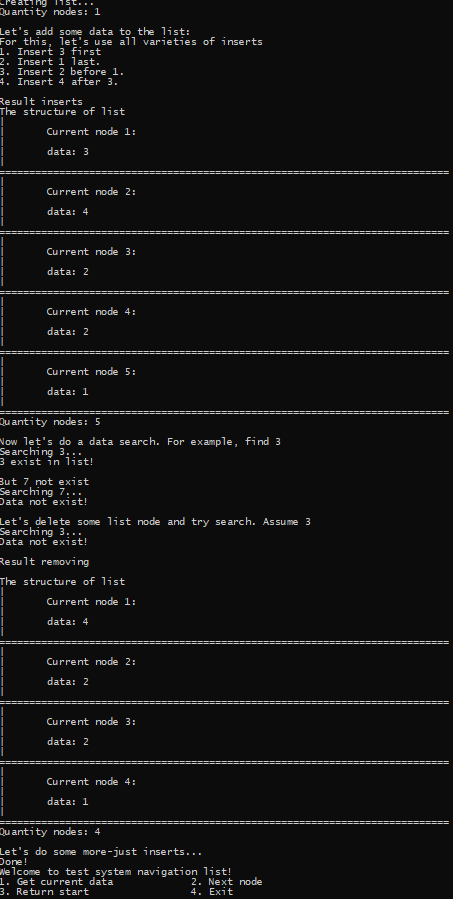
1. Сообщение на рисунке ниже информирует об аналогичном создании следующих узлов (рис. 5).



1. Сообщение об аналогичном создании узлов.

## Приложение для демонстрации работы списка

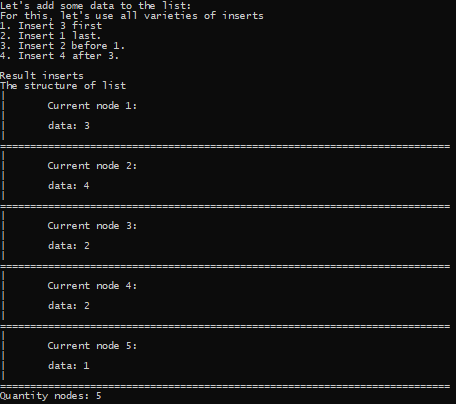
1. .Запустите приложением с названием \*.exe, как показано на рисунке (рис. 6).



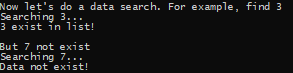
1. Основное окно программы.
2. Сначала создаётся список из одного узла (рис. 7).



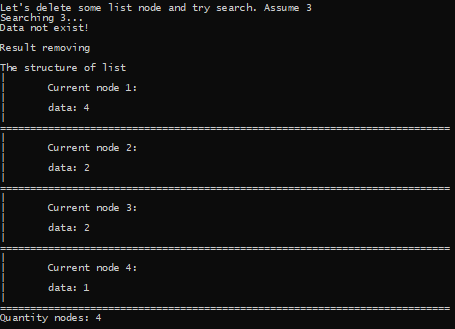
1. Создание списка
2. .На следующем шаге приводится полный список вставок, которые будут осуществляться при добавлении новых узлов односвязного списка как СД. Результат выводится на экран (рис. 8).



1. Вставки и их результат.
2. Теперь, когда у нас есть данные, можно продемонстрировать работу операции поиска узла списка (рис. 9).



1. Операция поиска.
2. Удалим какой-нибудь узел списка. Результат удаления приведён на рисунке ниже (рис. 10).



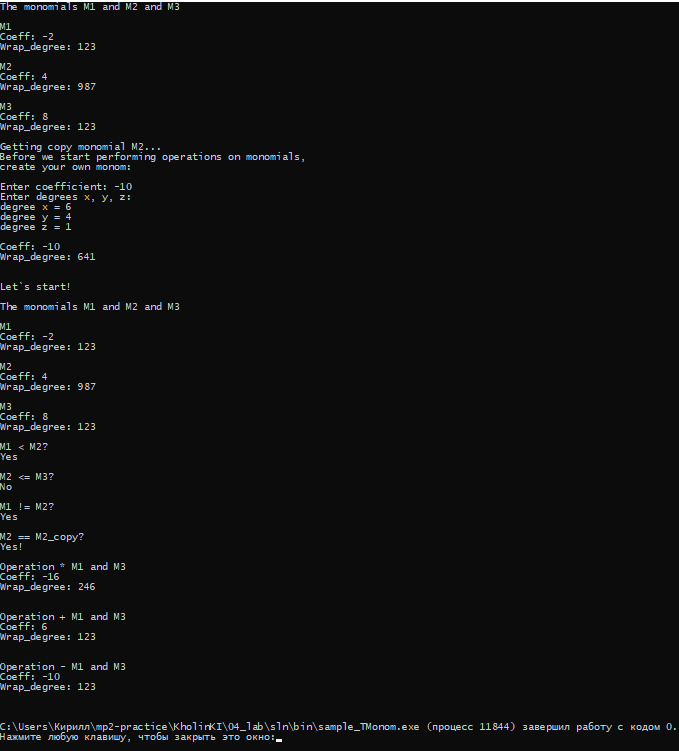
1. Удаление узла списка и результат.
2. В завершение вам предлагается интерфейс для навигации списка. Можно получить информацию о текущем узле, перейти к следующему, вернуться в начало списка и выйти из интерфейса (рис. 11).



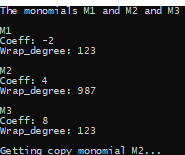
1. Навигация.

## Приложение для демонстрации работы мономов

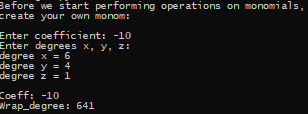
1. .Запустите приложение с названием \*.exe. В результате появится окно, как показано на рисунке ниже (рис. 12).



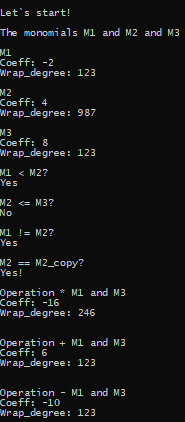
1. Основное окно программы.
2. В первую очередь создаются мономы с названиями M1,M2,M3 в порядке, указанном на рисунке (рис. 13). Дополнительно создаётся копия монома M2.



1. Создание мономов.
2. После пользователю предлагается создать собственный моном (рис. 14).



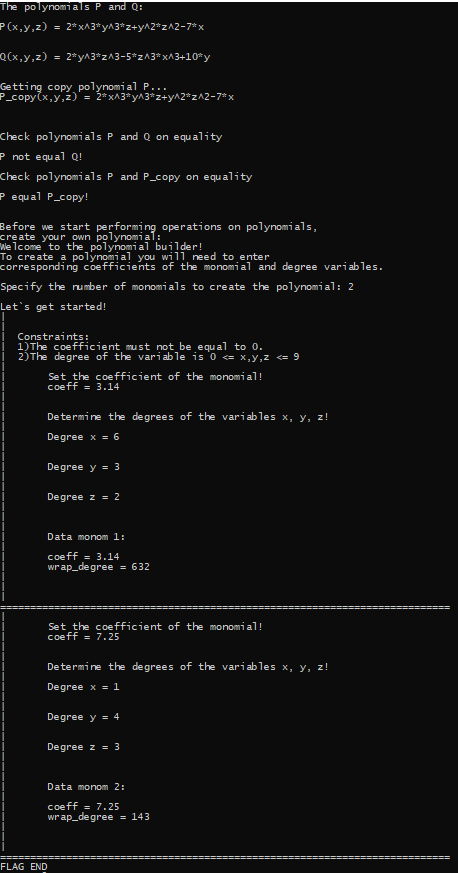
1. Пользовательский моном.
2. Выполняются такие операции над мономами, как операции сравнения и арифметические операции (рис. 15).



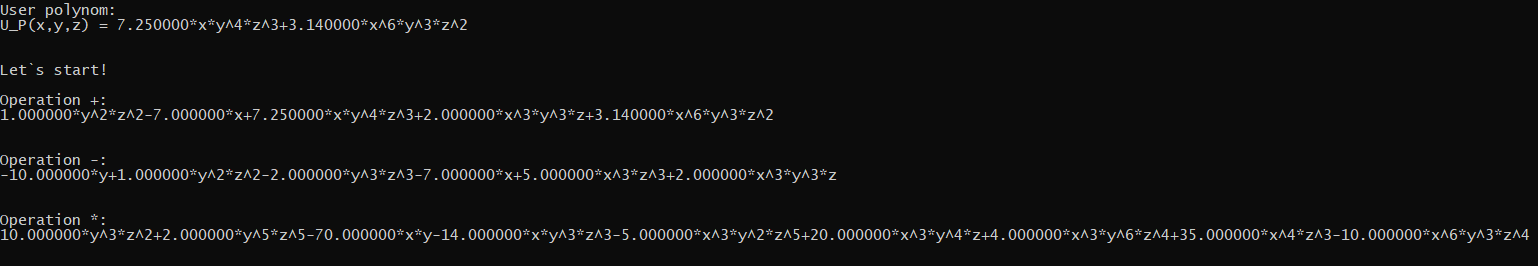
1. Операции над мономами.

## Приложение для демонстрации работы полиномов

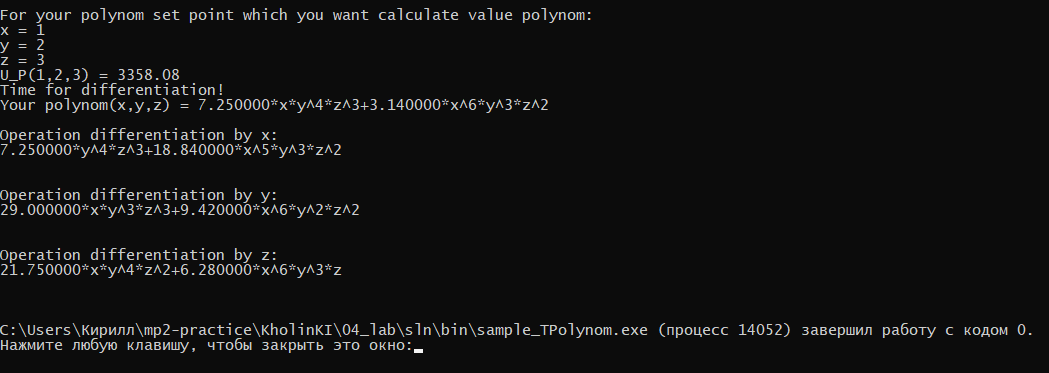
1. Запустите приложение с названием \*.exe. В результате появится окно, как показано на рисунках ниже (рис. 16), (рис. 17),(рис. 18).



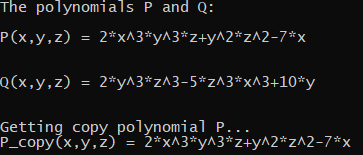
1. Основное окно программы. Часть 1.



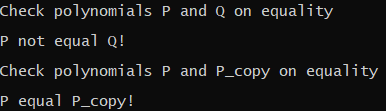
1. Основное окно программы. Часть 2.



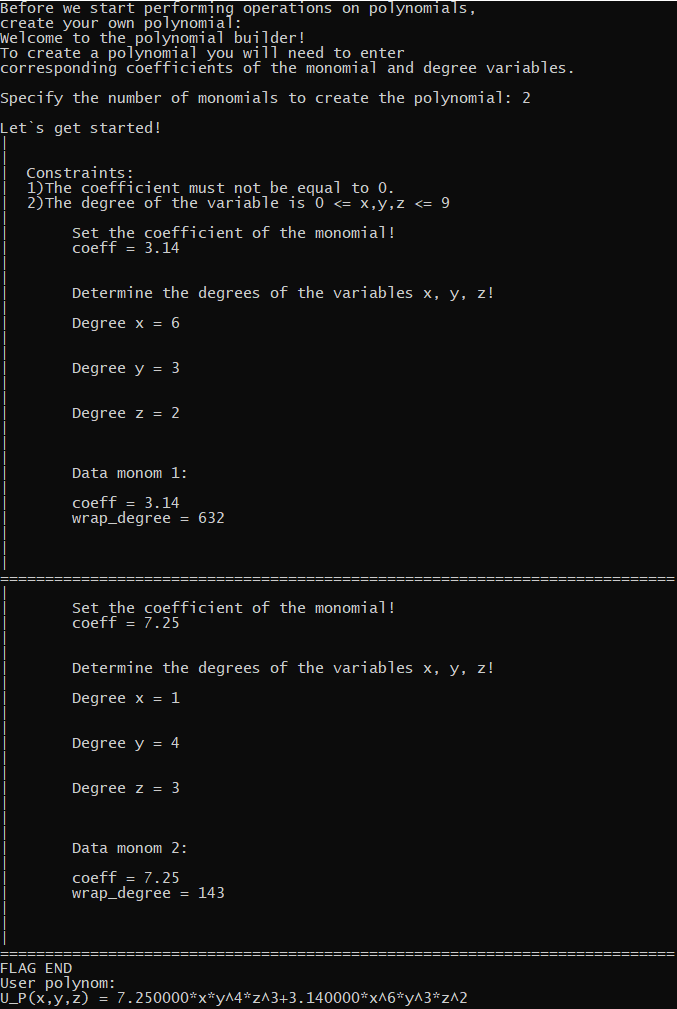
1. Основное окно программы. Часть 3.
2. В начале работы программы создаются два полинома с названиями P и Q, как показано на рисунке (рис. 19). Дополнительно создаётся копия полинома P.



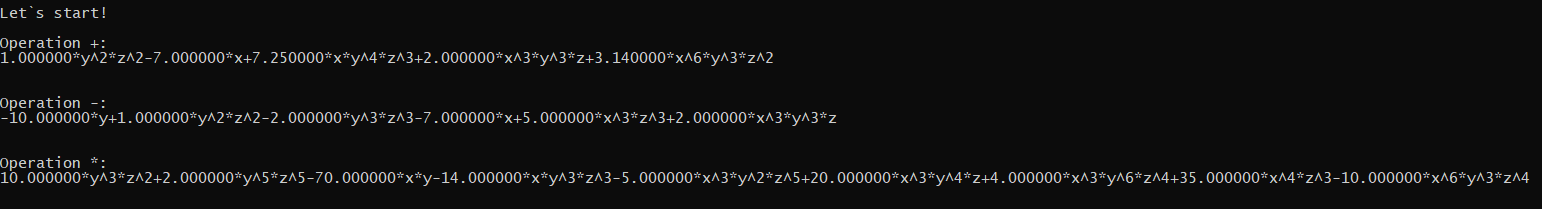
1. Создание полиномов.
2. Проверка, что полиномы различны (рис. 20).



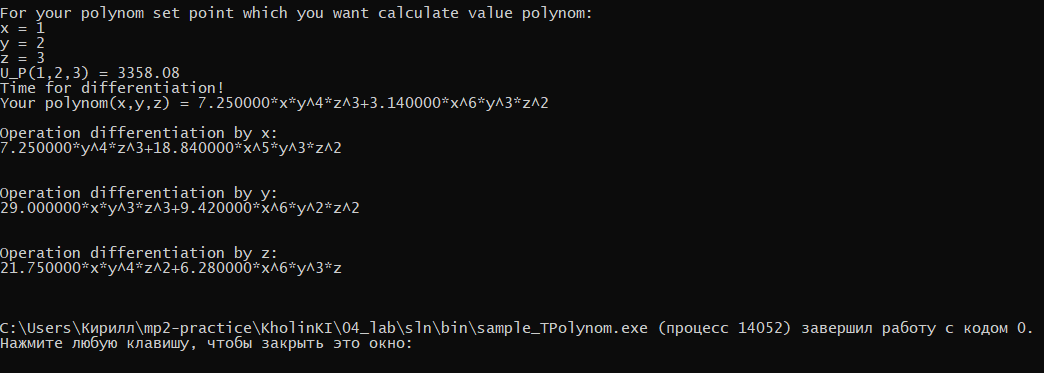
1. Проверка на равенство.
2. Наиболее важным и показательным моментом является создание пользовательского полинома. Пример вы можете увидеть на рисунке ниже (рис. 21).



1. Создание пользовательского полинома.
2. И в заключение выполняются различные операции над полиномами: сложение, вычитание, умножение, вычисление в точке, дифференцирование (рис. 22),(рис. 23).



1. Операции над полиномами. Часть 1.

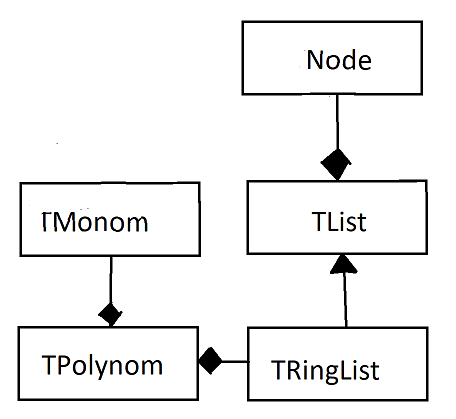


1. Операции над полиномами. Часть 2.

# Руководство программиста

## Описание алгоритмов

### Схема наследования классов.



### Шаблонная структура Node

Шаблонная структура Node – это узел линейного односвязного списка.

Он представляет из себя следующее:

|  |  |
| --- | --- |
| Data | pNext |

**Data** – это данные произвольного типа.

**pNext** – это указатель на следующее звено.

### Шаблонный класс TList

Шаблонный класс TList – это класс, который реализует линейный односвязный список. Линейный односвязный список – это структура данных, которая имеет следующее представление о себе:

pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

pPrev pCurr pLast

**pFirst –** указатель на начало списка.

**pCurr –** указатель на текущий узел списка.

**pPrev –** указатель на предшествующий узел.

**pLast –** указатель на последний узел.

**pStop –** указатель со значением NULL.

Примечание

Другие значения pStop могут использоваться по необходимости.

Рассмотрим операции со списком:

1. Поиск.

Поиск служит как средство обхода списка.

Алгоритм:

1. Если список пустой, ты выходим из функции поиска.
2. pCurr указывает на начало списка, а pPrev имеет значение nullptr;
3. В цикле, пока значение pCurr не совпало со значением pStop и Data pCurr не совпала с искомой Data, сначала pCurr двигаем на следующий узел, а потом pPrev сдвигаем на следующей узел, на который указывал до этого pCurr.
4. После выхода из цикла, если данные найдены, возвращаем pCurr. Иначе – возвращаем значение nullptr(данные не найдены).
5. Вставка в начало.

Узел с новыми данными вставляется в начало списка.

pNode

|  |  |
| --- | --- |
| NewData | pNext |

pFirst

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |

pCurr pLast

Алгоритм:

1. Создаём указатель pNode на новый узел списка , отдав в параметры конструктора узла новые данные.
2. pNext нового узла устанавливаем на pFirst.
3. Если список пустой, то сначала pFirst присваиваем указатель pNode, потому что pNode стал первым узлом списка, а затем pCurr и pLast – указатель pFirst. Иначе pFirst присваиваем указатель pNode, а потом pCurr присваиваем pFirst.
4. .Вставка в конец.

Узел с новыми данными вставляется в конец списка.

pNode

|  |  |
| --- | --- |
| NewData | pNext |

pFirst

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |

pCurr pLast

Алгоритм:

1. Если список пустой, то делаем вставку в начало и выходим из функции.
2. Создаём указатель на новый узел списка pNode, отдав в параметры конструктора узла новые данные.
3. Указатель pNext нового узла делаем равным pStop, а pNext указателя pLast делаем равным pNode.
4. Соответственно последним узлом списка будет pNode,то есть pLast делаем равным pNode.
5. Вставка before.

Узел с новыми данными вставляется **до** тех данных списка, которые ищем с помощью операции поиска узла списка.

pNode

|  |  |
| --- | --- |
| NewData | pNext |

pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

pPrev pCurr pLast

. Алгоритм:

1. Если список пустой, то делаем вставку в начало и выходим из функции.
2. Создаём временный указатель tmp на узел, поиск которого осуществляется с помощью операции поиска узла списка.
3. Если узел не найден, то бросаем исключение. Иначе создаём указатель pNode на новый узел списка, отдав в параметры конструктора узла новые данные.
4. Если операция поиска вернула указатель на первый узел списка, делаем вставку в начало списка. Иначе pNext указателя pNode присваиваем tmp и pNext указателя pPrev устанавливаем pNode.
5. С помощью операции reset() устанавливаем pCurr на pFirst, то есть возвращаем в исходную позицию.
6. Вставка after.

Узел с новыми данными вставляем **после** тех данных, которые ищем с помощью операции поиска узла списка.

pNode

|  |  |
| --- | --- |
| Data | pNext |

pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data | pNext |  | Data | pNext |  | Data | pNext |  | Data | pNext |

pPrev pCurr pLast

. Алгоритм:

1. Если список пустой, то делаем вставку в начало и выходим из функции.
2. Создаём временный указатель tmp на узел, поиск которого осуществляется с помощью операции поиска узла списка.
3. Если узел не найден, то бросаем исключение. Иначе создаём указатель pNode на новый узел списка, отдав в параметры конструктора узла новые данные.
4. Если операция поиска вернула указатель на последний узел списка, делаем вставку в конец списка. Иначе pNext указателя tmp присваиваем pNode, а pNext указателя pNode устанавливаем pNext указателя tmp.
5. В конце алгоритма не забыть прописать reset()
6. Упорядоченная вставка.

Упорядоченная вставка позволяет упорядочить данные списка.

Пусть у нас есть данные, которые надо вставить.

|  |
| --- |
| NewData |

Для удобства рассмотрим какой-нибудь упорядоченный по возрастанию список, но возьмём от него только данные без указателей.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data0 | Data1 | Data2 | Data3 | Data4 |

Теперь просто берём NewData и сравниваем с каждым Data узла списка. Если NewData меньше Data или совпадают, то вставляем новые данные в список.

Алгоритм:

1. В цикле, пока не дошли до конца списка, получаем данные узла, на который в текущий момент указывает pCurr. Обозначим такой узел за Comp\_data.
2. Если Newdata меньше Comp\_data или совпадает, то вызываем вставку before и делаем после этого reset(). Выходим из функции.
3. В противном случае применяем метод next(), который сдвигает указатель pCurr на следующий узел списка.
4. После выхода из цикла, если мы дошли до конца списка, делаем вставку в конец и выходим из функции.
5. Удаление.

Функция удаление исключает узел из списка.

Предположим, что нужно удалить узел с Data2.

pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

pPrev pCurr pLast

. Алгоритм:

1. Если список пустой, то выходим из функции.
2. Создаём временный указатель tmp на узел, поиск которого осуществляется с помощью операции поиска узла списка.
3. Если операция поиска вернула указатель на первый узел списка, то pFirst сдвигаем на следующий узел списка,а после этого pCurr присваиваем pFirst.
4. Далее важно pNext указателя tmp присвоить значение nullptr, чтобы не нарушить целостность списка. Теперь можно выполнить команду delete tmp, которая удаляет узел списка
5. Иначе, если мы дошли до конца списка, то pNext указателя pPrev присваиваем pNext указателя tmp. Выполняем пункт 4.
6. Не забудем проконтролировать pLast! pLast присвоить pPrev, поскольку удалялся последний узел списка.
7. .Очистка.

Функция очистка последовательно удаляет все узлы списка. В качестве примера приведено несколько итераций удаления.

I = 1

pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

tmp pLast

I = 2

pFirst

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

tmp pLast

. Алгоритм:

1. Если список пустой, то выходим из функции.
2. Создаём временный указатель tmp на первый узел списка.
3. В цикле, пока не дошли до конца списка, сдвигаем указатель pFirst на следующий узел. Выполнить процедуру удаления узла, как было написано в пункте 4 алгоритма удаления
4. Установить все указатели списка, кроме pStop, в nullptr.
5. Подсчёт количества узлов.

Функция подсчёта количества узлов считает узлы списка и возвращает число подсчитанных узлов.

Алгоритм:

1. Если список пустой, то выходим из функции.
2. Создаём временный указатель tmp на первый узел списка.
3. Заводим переменную под количество узлов списка(счётчик). Стартовое значение переменной имеет 0.
4. В цикле, пока не дошли до конца списка, увеличиваем счётчик на единицу и сдвигаем указатель tmp на следующий узел.
5. В результате функция вернёт количество подсчитанных узлов.
6. .Проверка на полноту и пустоту.

Определение 1

Список считается пустым, если он не содержит ни одного узла.

Алгоритм всего лишь заключается в проверке pFirst на значение nullptr.

Если pFirst равен nullptr, то функция проверки вернёт true,а иначе – false.

Определение 2

Список считается полным, если не удалось создать новый узел списка.

Алгоритм всего лишь заключается в проверки указателя на новый узел на значение nullptr в результате использования операции new.

Если указатель на новый узел равен nullptr, то функция вернёт true, а иначе – false.

1. .Сброс указателя pCurr.

Функция сброса указателя pCurr служит установки pCurr в начало списка.

1. .Переход к следующему узлу списка.

Сдвигает pCurr на следующий узел списка.

1. Проверка на конец списка.

Во время использования операции перехода к следующему узлу возможна ситуация, когда значение pCurr будет совпадать со значением pStop. Это означает, что мы дошли до конца списка. Функция проверки на конец списка вернёт в этом случае true, а иначе – false.

### Шаблонный класс TRingList

Шаблонный класс TRingList – это класс, который реализует кольцевой линейный односвязный список. Отличие от обычного списка заключается в том что добавляется ещё один указатель. Назовём его pHead. pHead – это указатель на фиктивный узел списка. Теперь признаком конца списка будет не значение NULL у pStop, а pHead. Вот как выглядит список в данном случае:

pHead pFirst

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data1 | pNext |  | Data2 | pNext |  | Data3 | pNext |  | Data4 | pNext |

pPrev pCurr pLast

Операции совершенно аналогичны операциям, которые были описаны у класса TList.

### Класс TMonom

Класс TMonom – это класс, который используется в качестве Data(данные) узла списка.

Всего два поля: коэффициент монома и свёртка степеней. Коэффициент может быть любым действительным числом, но в данной лабораторной работе мы исключаем нулевые коэффициенты. Также моном состоит из трёх переменных: x, y, z, однако одна из них может отсутствовать, когда мы будем говорить о полиномах. \

Свёртка степеней представляет собой следующее:

Пусть A, B и C – это степени переменных x, y и z соответственно. Для удобного хранения степеней мы можем использовать эту свёртку, которая находится по формуле:

ABC = A\*100+B\*10+C

Учитывая ограничение, в лабораторной работе A,B,C могут принимать целочисленные значения 0≤A,B,C≤9.

Обратное преобразование в степени переменных имеет вид:

A= (ABC%100),

B= (ABC-A\*100)%10,

C= ABC-A\*100-B\*10.

Рассмотрим операции над мономами:

Обозначим за коэффициент *Coeff*, а за свёртку степеней *Wp*.

1. Сложение

Если два монома имеют одинаковую свёртку степеней, то их коэффициенты складываются.

Monom1: 6\*x^2\*y^3.

Data1: Coeff1 = 6, Wp1 = 230.

|  |  |
| --- | --- |
| Data1 | pNext |

Monom2: -4\*x^2\*y^3.

Data2: Coeff2 = -4, Wp2 = 230.

|  |  |
| --- | --- |
| Data2 | pNext |

ResultMonom: 6\*x^2\*y^3 + -4\*x^2\*y^3 = -2\*x^2\*y^3.

ResData: Coeff = coeff1 + coeff2 = -2, Wp = 230.

|  |  |
| --- | --- |
| ResData | pNext |

1. Вычитание.

Если два монома имеют одинаковую свёртку степеней, то их коэффициенты вычитаются.

Monom1: 6\*x^2\*y^3.

Data1: Coeff1 = 6, Wp1 = 230.

Monom2: -4\*x^2\*y^3.

Data2: Coeff2 = -4, Wp2 = 230.

ResultMonom: : 6\*x^2\*y^3 - (-4\*x^2\*y^3) = 10\*x^2\*y^3.

ResData: Coeff = coeff1 – coeff2 = 10, Wp = 230.

1. Умножение.

Monom1: 6\*x^2\*y^3.

Data1: Coeff1 = 6, Wp1 = 230.

|  |  |
| --- | --- |
| Data1 | pNext |

Monom2: 7\*x^5\*y\*z^3.

Data2: Coeff2 = 7, Wp2 = 513.

|  |  |
| --- | --- |
| Data2 | pNext |

ResultMonom: 6\*x^2\*y^3 \* 7\*x^5\*y\*z^3 = 42\*x^7\*y^4\*z^3.

ResData: Coeff = coeff1 \* coeff2 = 42, Wp = Wp1 + Wp2 = 743.

|  |  |
| --- | --- |
| ResData | pNext |

При умножении мономов коэффициенты перемножаются, а соответствующие степени складываются.

1. Операции сравнения(<, ≤, !=, ==)

Все операции сравнения у монома сравнивают степени мономов. Например,

Monom1: 6\*x^2\*y^3.

Data1: Coeff1 = 6, Wp1 = 230.

|  |  |
| --- | --- |
| Data1 | pNext |

Monom2: 7\*x^5\*y\*z^3.

Data2: Coeff2 = 7, Wp2 = 513.

|  |  |
| --- | --- |
| Data2 | pNext |

Wp1 < Wp2, поэтому результат операции будет true. Аналогично для остальных операций.

### Класс TPolynom

Класс TPolynom – это класс, поддерживающий работу с полиномами.

Под полиномом понимается выражение вида:

Pn(x)=anxn + an-1xn-1 + … + a1x1 + a0 , где n – степень полинома, a – коэффициент монома,

x – переменная.

Другими словами, как функция от n переменных, представляющая собой сумму мономов.

С точки зрения структуры данных, полином можно представить как линейный односвязный список. Для удобства выбрали кольцевой список.

Описание узла кольцевого списка для полиномов.

|  |  |
| --- | --- |
| coeff | pNext |
| wp |

**coeff –** коэффициент монома.

**wp –** свёртка степеней.

**pNext –** указатель на следующий узел списка.

Как уже упоминалось ранее, мы будем рассматривать полином от трёх переменных: x, y и z.

Пример: 1+x + xy + z2yx6

Определение 1

Назовём два монома подобными, если их степени совпадают, а коэффициенты пропорциональны.

Определение 2

Назовём два монома равными, если их степени и коэффициенты равны.

Для организации быстрого доступа к данным мономов необходимо упорядочить полином (по возрастанию или по убыванию). Будем упорядочивать по возрастанию. Делается это очень просто:

Пусть даны свёртки степеней соответствующих мономов A1B1C1 и A2B2C2.

Чтобы упорядочить по возрастанию, нужно проверять условие A1B1C1 < A2B2C2.

Таким образом, моном с наименьшей свёрткой будет располагаться первым, а наибольший – последним. Для убывания аналогично.

Переходим к рассмотрению основных операций над полиномами:

1. Сложение.

При сложении полиномов соответствующие мономы с равными свёртками степеней приводятся к подобным мономам, а с неравными степенями – остаются без изменений.

P(x, y, z) = 1+ xy - 4x5z

Q(x, y, z) = 2yz2 - 4xy + z3x5

P + Q = 1 + 2yz2 – 3xy – 4x5z + x5z3

P(x, y, z)

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| 1 |  |
| 110 |

|  |  |
| --- | --- |
| -4 |  |
| 501 |

Q(x, y, z)

|  |  |
| --- | --- |
| -4 |  |
| 110 |

|  |  |
| --- | --- |
| 1 |  |
| 503 |

|  |  |
| --- | --- |
| 2 |  |
| 12 |

|  |  |
| --- | --- |
| 1 |  |
| 0 |

P + Q

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| 2 |  |
| 12 |

|  |  |
| --- | --- |
| -3 |  |
| 110 |

|  |  |
| --- | --- |
| -4 |  |
| 501 |

|  |  |
| --- | --- |
| 1 |  |
| 503 |

1. Вычитание.

Вычитание полиномов удобно реализовать через сложение полиномов. Соответствующие мономы с равными свёртками степеней приводятся к подобным мономам. Иначе знак коэффициента одного монома меняется на противоложный, и затем мономы складываются.

P(x, y, z) = 1+ xy - 4x5z

Q(x, y, z) = 2yz2 - 4xy + z3x5

P - Q = 1 - 2yz2 + 5xy – 4x5z - x5z3

P(x, y, z)

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| 1 |  |
| 110 |

|  |  |
| --- | --- |
| -4 |  |
| 501 |

Q(x, y, z)

|  |  |
| --- | --- |
| -4 |  |
| 110 |

|  |  |
| --- | --- |
| 1 |  |
| 503 |

|  |  |
| --- | --- |
| 2 |  |
| 12 |

|  |  |
| --- | --- |
| 1 |  |
| 0 |

P - Q

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| -2 |  |
| 12 |

|  |  |
| --- | --- |
| 5 |  |
| 110 |

|  |  |
| --- | --- |
| -4 |  |
| 501 |

|  |  |
| --- | --- |
| -1 |  |
| 503 |

1. Умножение

Каждый моном одного полинома перемножается со всеми мономами другого полинома.

P(x, y, z) = 7z2x +5x3y

Q(x, y, z) = -2xyz + 3x2y3

P\*Q = -14x2y3 + 21x3y3z2 -10x4y2z + 15x5y4

P(x, y, z)

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| 7 |  |
| 102 |

|  |  |
| --- | --- |
| 5 |  |
| 310 |

Q(x, y, z)

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| -2 |  |
| 111 |

|  |  |
| --- | --- |
| 3 |  |
| 230 |

P\*Q

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| -14 |  |
| 213 |

|  |  |
| --- | --- |
| 21 |  |
| 332 |

|  |  |
| --- | --- |
| 10 |  |
| 421 |

|  |  |
| --- | --- |
| 15 |  |
| 540 |

1. Вычисление в точке.

Пусть дана точка M с координатами (1, 2, 3).

Вычислим значение некоторого полинома P(x, y, z) в точке M.

Для этого вместо переменных полинома подставим значения координат

точки M соответственно.

Алгоритм

1. Исходный полином

P(x, y, z) = 1+ x + y2 – z3

1. Вычисляем значение

P(1, 2, 3) = 1 + 1 + 22 – 33 = 2 + 4 – 27 = -21

1. Получаем результат

P(1, 2, 3) = -21

1. .Дифференцирование.

Рассмотрим дифференцирование полинома по переменной x.

Для этого нужно взять частную производную по переменной икс

каждого монома, поскольку по свойствам производных производная

от суммы равна сумме производных.

Пусть дан полином P(x, y, z) = 10 – 4yz5 + 10x2z2 + x3y3

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| 10 |  |
| 0 |

|  |  |
| --- | --- |
| -4 |  |
| 15 |

|  |  |
| --- | --- |
| 10 |  |
| 202 |

|  |  |
| --- | --- |
| 1 |  |
| 330 |

Применяя дифференцирование по переменной икс для каждого монома,

получим новый полином Q:

Q(x, y, z) = 20xz2 + 3x2y3

|  |  |
| --- | --- |
| 1 |  |
| 0 |

|  |  |
| --- | --- |
| -20 |  |
| 102 |

|  |  |
| --- | --- |
| 3 |  |
| 230 |

Дифференцирование по остальным переменным аналогичное.

## Описание программной реализации

### Описание класса Node

template<typename T> class Node {

public:

T data;

Node\* pNext;

Node();

Node(const T& data\_);

Node(const Node& Node\_);

~Node();

};

Поля:

data – произвольные данные.

pNext – указатель на следующий узел.

Конструкторы:

Node();

Назначение: установка значений полей класса по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

Node(const T& data\_);

Назначение: инициализация полей класса значениями,

переданных в качестве параметров конструктора.

Входные параметры: data\_ - ссылка на данные.

Выходные параметры: отсутствуют.

Node(const Node& Node\_);

Назначение: копирование узла.

Входные параметры: Node\_ – ссылка на узел, который копируется.

Выходные параметры: отсутствуют.

~Node();

Назначение: освобождение памяти из-под узла.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

### **Описание класса TList**

template<typename T> class TList {

protected:

Node<T>\* pCurr;

public:

Node<T>\* pFirst;

Node<T>\* pPrev;

Node<T>\* pLast;

Node<T>\* pStop;

public:

TList();

TList(Node<T>\* pFirst\_);

TList(const TList<T>& list);

virtual ~TList();

Node<T>\* search(const T& data\_);

virtual void insert\_first(const T& data\_);

void insert\_last(const T& data\_);

void insert\_before(const T& data\_, const T& next\_data\_);

void insert\_after(const T& data\_, const T& next\_data\_);

void insertion\_sort(const T& data);

virtual void remove(const T& data\_);

int GetSize()const;

bool IsEmpty()const;

bool IsFull()const;

void clear();

void reset();

void next();

bool Is\_Ended() const;

Node<T>\* getCurrent() const{ return pCurr; }

friend ostream& operator<<(ostream& ostr, const TList<T>& l) {

Node<T>\* tmp = l.pCurr;

int number = 1;

cout << "The structure of list" << endl;

while (tmp != l.pStop) {

cout << "| " << endl;

cout << "| Current node " << number << ":" << "" << endl;

cout << "| " << endl;

cout << "| data: " << tmp->data << " " << endl;

cout << "| " << endl;

cout <<"=========================================" << endl;

number++;

tmp = tmp->pNext;

}

return ostr;

}

};

**Поля:**

pCurr – указатель на текущий узел списка.

pFirst – указатель на первый узел.

pPrev – указатель на предшествующий узел.

pLast – указатель на последний узел.

pStop – указатель со значением NULL.

**Конструкторы:**

TList();

Назначение: установка значений полей класса по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TList(Node<T>\* pFirst\_);

Назначение: инициализация указателей.

Входные параметры: pFirst\_ - указатель на первый узел списка.

Выходные параметры: отсутствуют.

TList(const TList<T>& list);

Назначение: копирование списка.

Входные параметры: list – ссылка на список, который копируется.

Выходные параметры: отсутствуют.

~TList();

Назначение: освобождение памяти из-под узлов списка.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

**Методы:**

Node<T>\* search(const T& data\_);

Назначение: поиск данных.

Входные параметры: data\_ - ссылка на данные, которые нужно найти.

Выходные параметры: указатель на звено списка.

virtual void insert\_first(const T& data\_);

Назначение: вставка данных в начало списка.

Входные параметры: data\_ - данные, которые нужно вставить.

Выходные параметры: отсутствуют.

void insert\_last(const T& data\_);

Назначение: вставка данных в конец списка.

Входные параметры: data\_ - данные, которые нужно вставить.

Выходные параметры: отсутствуют.

void insert\_before(const T& data\_, const T& next\_data\_);

Назначение: вставка данных до элемента списка.

Входные параметры: data\_ - данные, которые нужно вставить,

next\_data\_ - данные, до которых вставляем.

Выходные параметры: отсутствуют.

void insert\_after(const T& data\_, const T& next\_data\_);

Назначение: вставка данных после элемента списка.

Входные параметры: data\_ - данные, которые нужно вставить,

next\_data\_ - данные, после которых вставляем.

Выходные параметры: отсутствуют.

void insertion\_sort(const T& data);

Назначение: упорядочивание списка.

Входные параметры: data – данные, вставляемые в список

в соответствие с его заданным порядком.

Выходные параметры: отсутствуют.

virtual oid remove(const T& data\_);

Назначение: удаление узла.

Входные параметры: data\_ - данные, которые нужно удалить.

Выходные параметры: отсутствуют.

void clear();

Назначение: очистка списка.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

int GetSize()const;

Назначение: подсчёт узлов списка.

Входные параметры: отсутствуют.

Выходные параметры: количество подсчитанных узлов.

Node<T>\* getCurrent()const;

Назначение: получение значения поля pCurr.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущий узел списка.

bool IsEmpty()const;

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: true, если список пустой. Иначе false.

bool IsFull()const;

Назначение: проверка на полноту.

Входные параметры: отсутствуют.

Выходные параметры: true, если список полный. Иначе false.

bool IsEnded()const;

Назначение: проверка на достижение конца списка.

Входные параметры: отсутствуют.

Выходные параметры: true, если дошли до конца списка. Иначе false.

void next();

Назначение: перейти к следующему узлу.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void reset();

Назначение: вернуться в начало.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

### Описание класса TRingList

template <typename T>

class TRingList : public TList<T> {

protected:

Node<T>\* pHead;

public:

TRingList();

TRingList(Node<T>\* pFirst\_);

TRingList(const TRingList<T>& rList);

virtual ~TRingList();

void insert\_first(const T& data\_) override;

void remove(const T& data\_) override;

};

**Поля:**

pHead – указатель на фиктивный узел списка.

**Конструкторы:**

TRingList();

Назначение: установка значений полей класса по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TRingList(Node<T>\* pFirst\_);

Назначение: инициализация указателя.

Входные параметры: pFirst\_ - указатель на первый узел списка.

Выходные параметры: отсутствуют.

TRingList(const TRingList<T>& rList);

Назначение: копирование списка

Входные параметры: rList – ссылка на список, который копируется.

Выходные параметры: отсутствуют.

Деструктор.

Назначение: освобождение памяти из-под pHead.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

Методы:

Методы в классе TRingList являются переопределёнными, поэтому их назначение и параметры остаётся неизменными.

### Описание класса TMonom

class TMonom {

friend ostream& operator<<(ostream& ostr, const TMonom& monom);

friend istream& operator>>(istream& istr, TMonom& monom);

public:

double coeff;

int wrap\_degree;

TMonom(double c\_val = 0, int wd\_val = -1);

TMonom(const TMonom& mon);

bool operator<(const TMonom& monom)const;

bool operator<=(const TMonom& monom)const;

bool operator!=(const TMonom& monom)const;

bool operator==(const TMonom& monom)const;

TMonom operator\*(const TMonom& monom)const;

TMonom operator+(const TMonom& monom)const;

TMonom operator-()const;

};

**Поля:**

Coeff **–**  коэффициент монома.

Wrap\_degree **–** свёртка степеней.

**Конструкторы:**

TMonom(double c\_val = 0, int wd\_val = -1);

Назначение: инициализация полей класса.

Входные параметры: c\_val – коэффициент, wd\_val – свёртка степеней.

Выходные параметры: отсутствуют.

TMonom(const TMonom& mon);

Назначение: копирование мономов.

Входные параметры: mon – ссылка на моном, который копируется.

Выходные параметры: отсутствуют.

**Методы:**

Bool operator<(const TMonom& monom)const;

Назначение: сравнение мономов.

Входные параметры: monom – ссылка на моном, с которым сравниваем.

Выходные параметры: true, если текущий моном меньше monom. Иначе False.

Bool operator<=(const TMonom& monom)const;

Назначение: сравнение мономов.

Входные параметры: monom – ссылка на моном, с которым сравниваем.

Выходные параметры: true, если текущий моном меньше или равен monom. Иначе False.

Bool operator!=(const TMonom& monom)const;

Назначение: сравнение мономов.

Входные параметры: monom – ссылка на моном, с которым сравниваем.

Выходные параметры: true, если текущий моном не равен monom. Иначе False.

Bool operator==(const TMonom& monom)const;

Назначение: сравнение мономов.

Входные параметры: monom – ссылка на моном, с которым сравниваем.

Выходные параметры: true, если текущий моном равен monom. Иначе False.

TMonom operator+(const TMonom& monom)const;

Назначение: сложение мономов.

Входные параметры: monom – ссылка на моном, с которым складываем.

Выходные параметры: моном с результатом сложения.

TMonom operator\*(const TMonom& monom)const;

Назначение: умножение мономов.

Входные параметры: monom – ссылка на моном, с которым умножаем.

Выходные параметры: моном с результатом умножения.

TMonom operator-()const;

Назначение: изменение знака коэффициента монома.

Входные параметры: отсутствуют.

Выходные параметры: моном с противоположным знаком коэффициента.

### Описание класса TPolynom

class TPolynom {

friend istream& operator>>(istream& istr, TPolynom& Q);

friend ostream& operator<<(ostream& ostr, const TPolynom& Q);

public:

TRingList<TMonom>\* monoms;

private:

string polynom;

map<string, double> values\_xyz;

public:

TPolynom();

TPolynom(const string& str);

TPolynom(const TRingList<TMonom>\* monoms\_list);

TPolynom(const TPolynom& obj);

~TPolynom();

const TPolynom& operator=(const TPolynom& Q);

bool operator==(const TPolynom& Q)const;

TPolynom operator+(const TPolynom& Q);

TPolynom operator-(const TPolynom& Q);

TPolynom operator\*(const TPolynom& Q);

double operator()(double x,double y, double z);

string GetPolynomString()const;

TPolynom differentiate\_by\_x()const;

TPolynom differentiate\_by\_y()const;

TPolynom differentiate\_by\_z()const;

};

**Поля:**

monoms – список мономов.

polynom – строка с полиномом.

values\_xyz – значения переменных x, y, z.

**Конструкторы:**

TPolynom();

Назначение: установка полей класса по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TPolynom(const string& str);

Назначение: преобразование строки с полиномом в кольцевой список мономов.

Входные параметры: str – строка с полиномом.

Выходные параметры: отсутствуют.

TPolynom(const TRingList<TMonom>\* monoms\_list);

Назначение: преобразование кольцевого списка мономов в строку.

Входные параметры: monoms\_list – указатель на кольцевой список мономов.

Выходные параметры: отсутствуют.

TPolynom(const TPolynom& obj);

Назначение: копирование полиномов.

Входные параметры: obj – ссылка на полином, который копируется.

Выходные параметры: отсутствуют.

~TPolynom();

Назначение: освобождение выделенной памяти под список мономов.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

**Методы:**

TPolynom operator+(const TPolynom& obj);

Назначение: сложение полиномов.

Входные параметры: obj – ссылка на полином, с которым складываем.

Выходные параметры: полином с результатом сложения.

TPolynom operator-(const TPolynom& obj);

Назначение: вычитание полиномов.

Входные параметры: obj – ссылка на полином, который вычитаем.

Выходные параметры: полином с результатом вычитания.

TPolynom operator\*(const TPolynom& obj);

Назначение: умножение полиномов.

Входные параметры: obj – ссылка на полином, с которым умножаем.

Выходные параметры: полином с результатом умножения.

Double operator()(double x, double y, double z);

Назначение: вычисление значения полинома в точке.

Входные параметры: x, y, z – координаты точки по OX,OY,OZ соответственно.

Выходные параметры: значение полинома в точке.

Const TPolynom& operator=(const TPolynom& Q);

Назначение: присваивание полиномов.

Входные параметры: Q- ссылка на полином, который нужно присвоить.

Выходные параметры: ссылка на полином, которому присвоили Q.

Bool operator==(const TPolynom& Q)const;

Назначение: сравнение полиномов.

Входные параметры: Q- ссылка на полином, с которым сравниваем.

Выходные параметры: true, если полиномы равны. Иначе false.

String GetPolynomString()const;

Назначение: получение строки с полиномом.

Входные параметры: отсутствуют.

Выходные параметры: строка с полиномом.

TPolynom differentiate\_by\_x()const;

Назначение: дифференцирование по переменной x.

Входные параметры: отсутствуют.

Выходные параметры: полином с результатом дифференцирования по x.

TPolynom differentiate\_by\_y()const;

Назначение: дифференцирование по переменной y.

Входные параметры: отсутствуют.

Выходные параметры: полином с результатом дифференцирования по y.

TPolynom differentiate\_by\_z()const;

Назначение: дифференцирование по переменной z.

Входные параметры: отсутствуют.

Выходные параметры: полином с результатом дифференцирования по z.

# Заключение

По результатам лабораторной работы была разработана такая эффективная структура данных, как линейный односвязный список, на основе которого был реализован класс для работы с полиномами. Все основные операции классов написаны: операции с полиномам и со списками. Состав операций расширен специфическими методами, позволяющими совершать обход структуры данных максимально эффективно. Благодаря дополнительному функционалу разработана диалоговая управляющая программа с наглядным визуальным интерфейсом.

# Литература

1. Льюис, Чейз <Современное программирование на C++: структуры данных и алгоритмы> - Издательство "БХВ-Петербург" - 960 страниц
2. Майкл Т. Гудріч, Роберт Лафоре <Структуры данных и алгоритмы в Java> - Издательство "Вильямс" - 720 страниц.
3. Майкл Доусон, Эд Вагнер <Структуры данных и алгоритмы в С++> - Издательство "Питер" - 352 страницы.
4. Майкл Майерс <Эффективное использование STL: 50 способов улучшить свои программы и проекты на C++> - Издательство "O'Reilly Media" - 288 страниц.
5. Стэнли Б. Липман, Лайонел Ленихан <C++ Primer> - Издательство "Addison-Wesley Professional" - 1376 страниц.

# Приложения

## Приложение А. Реализация класса Node

template<typename T>

Node<T>::Node() {

data = {};

pNext = nullptr;

}

template<typename T>

Node<T>::Node(const T& data\_) {

data = data\_;

pNext = nullptr;

}

template<typename T>

Node<T>::Node(const Node& Node\_) {

data = Node\_.data;

pNext = Node\_.pNext;

}

template<typename T>

Node<T>::~Node() {

delete pNext;

data = {};

}

Пример:

int main() {

cout << "Creating object Node with name sample\_node..." << endl;

Node<double> sample\_node;

cout << endl;

cout << "The node does not contain any data at this time" << endl;

cout << endl;

cout << "Let`s add something. For example, value type double 4" << endl; cout << endl;

sample\_node.data = 4;

cout << "sample\_node" << endl;

cout << "Data: " << sample\_node.data << endl;

cout << "pNext: " << "nullptr" << endl;

cout << endl;

cout << "OK!we'll move on to the next node" << endl;

sample\_node.pNext = new Node<double>;

cout << "Add something again in new node" << endl;

cout << endl;

sample\_node.pNext->data = 6;

cout << "sample\_node.pNext" << endl;

cout << "Data: " << sample\_node.pNext->data << endl;

cout << "pNext: " << "nullptr" << endl;

cout << endl;

cout << "in the same way creating next nodes..." << endl;

return 0;

}

## Приложение Б. Реализация класса TList

template<typename T>

TList<T>::TList() { // pPrev

pFirst = nullptr;

pLast = nullptr;

pCurr = nullptr;

pStop = nullptr;

pPrev = nullptr;

}

template<typename T>

TList<T>::TList(Node<T>\* pFirst\_) {

if (pFirst\_ == nullptr)

{

pFirst = nullptr;

return;

}

pFirst = new Node<T>(pFirst\_->data);

pCurr = pFirst;

Node<T>\* tmp = pFirst->pNext;

while (tmp != nullptr) {

pCurr->pNext = new Node<T>(tmp->data);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pLast = pCurr;

pStop = nullptr;

pCurr = pFirst;

pPrev = nullptr;

}

template<typename T>

TList<T>::TList(const TList<T>& list) {

if (list.pFirst == nullptr) {

return;

}

pFirst = new Node<T>(list.pFirst->data);

Node<T>\* tmp = list.pFirst->pNext;

pCurr = pFirst;

while (tmp != list.pStop) {

pCurr->pNext = new Node<T>(tmp->data);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pStop = nullptr;

pLast = pCurr;

pCurr = pFirst;

pPrev = nullptr;

}

template<typename T>

TList<T>::~TList() {

clear();

}

template<typename T>

Node<T>\* TList<T>::search(const T& data\_) {

pCurr = pFirst;

pPrev = nullptr;

if (pFirst == nullptr) {

return nullptr;

}

while (pCurr != pStop && pCurr->data != data\_) {

pPrev = pCurr;

pCurr = pCurr->pNext;

}

if (pCurr == pStop) {

return nullptr;

}

return pCurr;

}

template<typename T>

void TList<T>::insert\_first(const T& data\_) {

Node<T>\* tmp = new Node<T>(data\_);

tmp->pNext = pFirst;

if (pFirst == nullptr) {

pFirst = tmp;

pLast = pFirst;

pCurr = pFirst;

}

else {

pFirst = tmp;

pCurr = pFirst;

}

}

template<typename T>

void TList<T>::insert\_last(const T& data\_) {

if (pFirst == nullptr) {

insert\_first(data\_);

return;

}

Node<T>\* tmp = pLast;

Node<T>\* new\_node = new Node<T>(data\_);

new\_node->pNext = pStop;

tmp->pNext = new\_node;

pLast = new\_node;

}

template<typename T>

void TList<T>::insert\_before(const T& data\_, const T& next\_data\_) {

if (pFirst == nullptr) {

insert\_first(data\_);

return;

}

Node<T>\* tmp = search(next\_data\_);

if (tmp == nullptr) {

throw "Data not found!";

}

Node<T>\* node = new Node<T>(data\_);

if (tmp == pFirst) {

insert\_first(data\_);

}

else {

node->pNext = tmp;

pPrev->pNext = node;

}

reset();

}

template<typename T>

void TList<T>::insert\_after(const T& data\_, const T& next\_data\_) {

if (pFirst == nullptr) {

insert\_first(data\_);

return;

}

Node<T>\* tmp = search(next\_data\_);

if (tmp == nullptr) {

throw "Data not found!";

}

if (tmp->pNext == pStop) {

insert\_last(data\_);

}

else {

Node<T>\* node = new Node<T>(data\_);

node->pNext = tmp->pNext;

tmp->pNext = node;

}

reset();

}

template<typename T>

void TList<T>::insertion\_sort(const T& data) {

while (!Is\_Ended()) {

T comp\_data = getCurrent()->data;

if (data <= comp\_data) {

insert\_before(data, comp\_data);

reset();

break;

}

next();

}

if (Is\_Ended()) {

insert\_last(data);

reset();

}

}

template<typename T>

void TList<T>::remove(const T& data\_) {

if (pFirst == nullptr) {

return;

}

Node<T>\* tmp = search(data\_);

if (tmp == nullptr) {

throw "Data not found!";

}

if (tmp == pFirst) {

pFirst = pFirst->pNext;

pCurr = pFirst;

tmp->pNext = nullptr;

delete tmp;

}

else {

if (tmp->pNext == pStop) {

pLast = pPrev;

}

pPrev->pNext = tmp->pNext;

tmp->pNext = nullptr;

delete tmp;

}

}

template<typename T>

void TList<T>::clear() {

if (pFirst == nullptr) {

return;

}

Node<T>\* tmp = pFirst;

while (tmp != pStop) {

pFirst = pFirst->pNext;

tmp->pNext = nullptr;

delete tmp;

tmp = pFirst;

}

pFirst = nullptr;

pCurr = nullptr;

pPrev = nullptr;

pLast = nullptr;

}

template<typename T>

int TList<T>::GetSize()const {

if (pFirst == nullptr) {

return 0;

}

int count = 0;

Node<T>\* tmp = pFirst;

while (tmp != pStop) {

count++;

tmp = tmp->pNext;

}

return count;

}

template<typename T>

bool TList<T>::IsEmpty()const {

return pFirst == nullptr ? true : false;

}

template<typename T>

bool TList<T>::IsFull()const {

Node<T>\* tmp = new Node<T>();

if (tmp == nullptr) {

throw "Failed allocate memory!";

}

else { return false; }

}

template<typename T>

void TList<T>::reset() {

pCurr = pFirst;

}

template<typename T>

void TList<T>::next() {

pCurr = pCurr->pNext;

}

template<typename T>

bool TList<T>::Is\_Ended()const {

return pCurr == pStop ? true : false;

}

Пример:

int main() {

cout << "Creating list..." << endl;

TList<int> my\_list(new Node<int>(2));

if (my\_list.IsEmpty()) {

cout << "empty list" << endl;

}

cout << "Quantity nodes: " << my\_list.GetSize() << endl << endl;

cout << "Let's add some data to the list: " << endl;

cout << "For this, let's use all varieties of inserts" << endl;

cout << "1. Insert 3 first" << endl;

my\_list.insert\_first(3);

cout << "2. Insert 1 last." << endl;

my\_list.insert\_last(1);

cout << "3. Insert 2 before 1." << endl;

my\_list.insert\_before(2, 1);

cout << "4. Insert 4 after 3." << endl;

my\_list.insert\_after(4, 3);

cout << endl;

cout << "Result inserts" << endl;

cout << my\_list;

cout << "Quantity nodes: " << my\_list.GetSize() << endl;

cout << endl;

cout << "Now let's do a data search. For example, find 3" << endl;

cout << "Searching 3..." << endl;

Node<int>\* found = my\_list.search(3);

my\_list.reset();

if (found != nullptr) {

cout << found->data << " exist in list!" << endl << endl;

}

else {

cout << "Data not exist!" << endl << endl;

}

cout << "But 7 not exist" << endl;

cout << "Searching 7..." << endl;

found = my\_list.search(7);

my\_list.reset();

if (found != nullptr) {

cout << found->data << " exist in list!" << endl << endl;

}

else {

cout << "Data not exist!" << endl << endl;

}

cout << "Let's delete some list node and try search. Assume 3" << endl;

my\_list.remove(3);

cout << "Searching 3..." << endl;

found = my\_list.search(3);

my\_list.reset();

if (found != nullptr) {

cout << found->data << " exist in list!" << endl << endl;

}

else {

cout << "Data not exist!" << endl << endl;

}

cout << "Result removing" << endl << endl;

cout << my\_list;

cout << "Quantity nodes: " << my\_list.GetSize() << endl << endl;

cout << "Let's do some more-just inserts..." << endl;

my\_list.insert\_last(6);

my\_list.insert\_last(8);

my\_list.insert\_last(4);

my\_list.insert\_last(20);

cout << "Done!" << endl;

bool mode = true;

int choice = 0;

cout << "Welcome to test system navigation list!" << endl;

do {

cout << "1. Get current data" << " " << "2. Next node " << endl;

cout << "3. Return start " << " " << "4. Exit " << endl;

cout << endl << endl << endl;

do {

cin >> choice;

} while (choice < 0 || choice > 4);

switch (choice) {

case 1:{

cout << "| " << endl;

cout << "| Current node:" << endl;

cout << "| " << endl;

cout << "| data: " << my\_list.getCurrent()->data << "“<<endl;

cout << "| " << endl;

break;

}

case 2:{

if (!my\_list.Is\_Ended()) {

my\_list.next();

}

else {

cout << "You've reached the end of the list,";

cout << "so you're back at the beginning" << endl;

my\_list.reset();

}

break;

}

case 3:{

my\_list.reset();

cout << "You returned in start list!" << endl;

break;

}

case 4:{

mode = false;

my\_list.reset();

cout << "Completion of work..." << endl;

break;

}

}

} while (mode != false);

cout << "Have a nice day!" << endl;

}

## Приложение C. Реализация класса TRingList.

template<typename T>

TRingList<T>::TRingList(){

pHead = new Node<T>();

pHead->pNext = pHead;

pStop = pHead;

}

template <typename T>

TRingList<T>::TRingList(Node<T>\* pFirst\_):TRingList<T>(){

if (pFirst\_ == nullptr)

{

return;

}

pFirst = new Node<T>(pFirst\_->data);

pHead->pNext = pFirst;//почему менять местами нельзя?

Node<T>\* tmp = pFirst->pNext;

pCurr = pFirst;

while (tmp != nullptr) {

pCurr->pNext = new Node<T>(tmp->data);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pCurr->pNext = pHead;

pLast = pCurr;

pCurr = pFirst;

pPrev = nullptr;

pStop = pHead;

}

template <typename T>

TRingList<T>::TRingList(const TRingList<T>& rList){

if (rList.pFirst == nullptr) {

return;

}

pHead = new Node<T>();

pFirst = new Node<T>(rList.pFirst->data);

pHead->pNext = pFirst;

Node<T>\* tmp = rList.pFirst->pNext;

pCurr = pFirst;

while (tmp != rList.pStop) {

pCurr->pNext = new Node<T>(tmp->data);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pCurr->pNext = pHead;

pLast = pCurr;

pCurr = pFirst;

pPrev = nullptr;

pStop = pHead;

}

template<typename T>

TRingList<T>::~TRingList() {

pHead->pNext = nullptr;

delete pHead;

}

template<typename T>

void TRingList<T>::insert\_first(const T& data\_) {

Node<T>\* tmp = new Node<T>(data\_);

tmp->pNext = pFirst;

if (pFirst == nullptr) {

pFirst = tmp;

pFirst->pNext = pHead;

pHead->pNext = pFirst;

pLast = pFirst;

pCurr = pFirst;

}

else {

pFirst = tmp;

pHead->pNext = pFirst;

pCurr = pFirst;

}

}

template<typename T>

void TRingList<T>::remove(const T& data\_) {

if (pFirst == nullptr) {

return;

}

Node<T>\* tmp = search(data\_);

//if

if (tmp == pFirst) {

pFirst = pFirst->pNext;

pHead->pNext = pFirst;

pCurr = pFirst;

tmp->pNext = nullptr;

delete tmp;

}

else {

if (tmp->pNext == pStop) {

pLast = pPrev;

}

pCurr = pCurr->pNext;

pPrev->pNext = tmp->pNext;

tmp->pNext = nullptr;

delete tmp;

}

}

## Приложение D. Реализация класса TMonom.

TMonom::TMonom(double c\_val, int wd\_val) {

coeff = c\_val;

wrap\_degree = wd\_val;

}

TMonom::TMonom(const TMonom& mon) {

coeff = mon.coeff;

wrap\_degree = mon.wrap\_degree;

}

bool TMonom::operator<(const TMonom& monom)const {

if (wrap\_degree < monom.wrap\_degree) {

return true;

}

else {

return false;

}

}

bool TMonom::operator<=(const TMonom& monom)const {

if (wrap\_degree <= monom.wrap\_degree) {

return true;

}

else {

return false;

}

}

bool TMonom::operator!=(const TMonom& monom)const {

if (wrap\_degree != monom.wrap\_degree) {

return true;

}

else {

return false;

}

}

bool TMonom::operator==(const TMonom& monom)const {

if (wrap\_degree == monom.wrap\_degree) {

return true;

}

else {

return false;

}

}

TMonom TMonom::operator\*(const TMonom& monom)const {

int d\_x\_t = wrap\_degree / 100;

int d\_y\_t = (wrap\_degree % 100) / 10;

int d\_z\_t = (wrap\_degree % 10);

int d\_x\_m = monom.wrap\_degree / 100;

int d\_y\_m = (monom.wrap\_degree % 100) / 10;

int d\_z\_m = (monom.wrap\_degree % 10);

if (d\_x\_t + d\_x\_m < 0 || d\_x\_t + d\_x\_m > 9) {

throw "Degree limit!";

}

else if (d\_y\_t + d\_y\_m < 0 || d\_y\_t + d\_y\_m > 9) {

throw "Degree limit";

}

else if (d\_z\_t + d\_z\_m < 0 || d\_z\_t + d\_z\_m > 9) {

throw "Degree limit";

}

return TMonom(coeff \* monom.coeff, wrap\_degree + monom.wrap\_degree);

}

TMonom TMonom::operator+(const TMonom& monom)const {

if (this->wrap\_degree != monom.wrap\_degree) {

throw "Monomialss have different degrees!";

}

TMonom res(\*this);

res.coeff = coeff + monom.coeff;

return res;

}

TMonom TMonom::operator-()const {

TMonom res(\*this);

res.coeff = -coeff;

return res;

}

istream& operator>>(istream& istr, TMonom& monom) {

double d\_x; double d\_y; double d\_z;

cout << "Enter coefficient: ";

cin >> monom.coeff;

cout << "Enter degrees x, y, z:" << endl;

cout << "degree x = ";

cin >> d\_x;

cout << "degree y = ";

cin >> d\_y;

cout << "degree z = ";

cin >> d\_z;

monom.wrap\_degree = d\_x \* 100 + d\_y \* 10 + d\_z;

return istr;

}

ostream& operator<<(ostream& ostr, const TMonom& monom) {

cout << "Coeff: " << monom.coeff << endl;

cout << "Wrap\_degree: " << monom.wrap\_degree << endl;

cout << endl;

return ostr;

}

Пример:

int main(){

cout << "The monomials M1 and M2 and M3" << endl << endl;

TMonom M1(-2, 123); TMonom M2(4, 987); TMonom M3(8, 123); TMonom U\_M;

cout << "M1" << endl;

cout << M1;

cout << "M2" << endl;

cout << M2;

cout << "M3" << endl;

cout << M3;

cout << "Getting copy monomial M2..." << endl;

TMonom M2\_copy(M2);

cout << "Before we start performing operations on monomials," << endl;

cout << "create your own monom: " << endl << endl;

cin >> U\_M;

cout << endl;

cout << U\_M;

cout << endl;

cout << "Let`s start!" << endl;

cout << endl;

cout << "The monomials M1 and M2 and M3" << endl << endl;

cout << "M1" << endl;

cout << M1;

cout << "M2" << endl;

cout << M2;

cout << "M3" << endl;

cout << M3;

cout << "M1 < M2?" << endl;

if (M1 < M2) {

cout << "Yes" << endl;

}

else {

cout << "No" << endl;

cout << endl;

cout << "M1 == M2?" << endl;

if (M1 == M2) {

cout << "Yes,it`s equal monomials" << endl;

}

else {

cout << "No" << endl;

}

cout << endl;

}

cout << endl;

cout << "M2 <= M3?" << endl;

if (M2 <= M3) {

if (M2 == M3) {

cout << "Yes,it`s equal monomials" << endl;

}

else{

cout << "Yes, M2 < M3" << endl;

}

}

else {

cout << "No" << endl;

}

cout << endl;

cout << "M1 != M2?" << endl;

if (M1 != M2) {

if (M1.coeff != M2.coeff) {

cout << "Yes" << endl;

}

}

else {

if (M1.coeff == M2.coeff) {

cout << "Yes,it`s equal monomials" << endl;

}

else {

cout << "No" << endl;

}

cout << endl;

}

cout << endl;

cout << "M2 == M2\_copy?" << endl;

if (M2 == M2\_copy) {

cout << "Yes!" << endl;

}

else {

cout << "No!" << endl;

}

cout << endl;

cout << "Operation \* M1 and M3" << endl;

TMonom res1 = M1 \* M3;

cout << res1;

cout << endl;

cout << "Operation + M1 and M3" << endl;

TMonom res2 = M1 + M3;

cout << res2;

cout << endl;

cout << "Operation - M1 and M3" << endl;

TMonom res3 = M1 + (-M3);

cout << res3;

cout << endl;

return 0;

}

## Приложение E. Реализация класса TPolynom.

TPolynom::TPolynom() {

monoms = new TRingList<TMonom>();

polynom = "";

}

TPolynom::TPolynom(const string& str) {

monoms = nullptr;

int size\_str = str.size();

string infix;

if (size\_str != 0) {

ArithmeticExpression expr(str);

infix = expr.GetInfix();

}

polynom = infix;

int i = -1;

string tmp;

bool minus\_status = false;

string str\_test;

int degree\_x = 0;

int degree\_y = 0;

int degree\_z = 0;

int wrap\_degree = 0;

double coeff\_monom = 1;

int size = infix.size();

bool stop = false;

bool create\_monom = true;

while (i < size && size != 0){

do {

i++;

tmp = infix[i];

if (tmp[0] == '(' || tmp[0] == ')') {

continue;

}

if (tmp[0] >= 65 && tmp[0] <= 90 || tmp[0] >= 97 && tmp[0] <= 122) {

if ((i - 1) >= 0) {

if(infix[i-1] == '-'){

coeff\_monom = -1;

}

else if (infix[i - 1] == '+') {

coeff\_monom = 1;

}

}

break;

}

} while (!(tmp[0] >= 48 && tmp[0] <= 57));

if(tmp[0] >= 48 && tmp[0] <= 57) {

if ((i - 1) >= 0) {

if (infix[i - 1] == '-') {

minus\_status = true;

}

}

while (tmp[0] >= 48 && tmp[0] <= 57) {

str\_test = str\_test + tmp;

i++;

tmp = infix[i];

}

if (minus\_status == true) {

coeff\_monom = stod(str\_test) \* (-1);

if (coeff\_monom == 0) {

while (tmp[0] != '-' && tmp[0] != '+'&& i< size) {

i++;

tmp = infix[i];

}

create\_monom = false;

minus\_status = false;

}

}

else {

coeff\_monom = stod(str\_test);

if (coeff\_monom == 0) {

while (tmp[0] != '-' && tmp[0] != '+'&& i< size) {

i++;

tmp = infix[i];

}

create\_monom = false;

}

}

if (tmp[0] == '\0') {

stop = true;

}

str\_test = "";

minus\_status = false;

}

while (tmp[0] != '-' && tmp[0] != '+' && stop == false && create\_monom == true) {

switch (tmp[0]) {

case 'x':

{

if (infix[i + 1] == '^') {

str\_test = str\_test + infix[i + 2];

degree\_x = stod(str\_test);

if (degree\_x < 0 || degree\_x > 9) {

throw "degree limit!";

}

str\_test = "";

}

else {

degree\_x = 1;

}

break;

}

case 'y':

{

if (infix[i + 1] == '^') {

str\_test = str\_test + infix[i + 2];

degree\_y = stod(str\_test);

if (degree\_y < 0 || degree\_y > 9) {

throw "degree limit!";

}

str\_test = "";

}

else {

degree\_y = 1;

}

break;

}

case 'z':

{

if (infix[i + 1] == '^') {

str\_test = str\_test + infix[i + 2];

degree\_z = stod(str\_test);

if (degree\_z < 0 || degree\_z > 9) {

throw "degree limit!";

}

str\_test = "";

}

else {

degree\_z = 1;

}

break;

}

}

i++;

if (i == size) {

break;

}

tmp = infix[i];

}

if (create\_monom == true) {

if (monoms == nullptr) {

wrap\_degree = degree\_x \* 100 + degree\_y \*10+ degree\_z;

TMonom m(coeff\_monom, wrap\_degree);

monoms = new TRingList<TMonom>();

monoms->insert\_first(m);

}

else {

wrap\_degree = degree\_x \* 100 + degree\_y \*10+ degree\_z;

TMonom m(coeff\_monom, wrap\_degree);

Node<TMonom>\* found = monoms->search(m);

monoms->reset();

if (found != nullptr && found != monoms->pStop) {

TMonom result = found->data + m;

if (result.coeff != 0) {

found->data = result;

}

if (result.coeff == 0) {

monoms->remove(found->data);

}

}

else {

monoms->insertion\_sort(m);

}

}

}

degree\_x = 0;

degree\_y = 0;

degree\_z = 0;

coeff\_monom = 1;

stop = false;

create\_monom = true;

}

if (size == 0) {

monoms = new TRingList<TMonom>();

}

if (monoms == nullptr && size != 0) {

monoms = new TRingList<TMonom>();

}

}

TPolynom::TPolynom(const TRingList<TMonom>\* monoms\_list) {

monoms = new TRingList<TMonom>(\*monoms\_list);

string \_polynom = "";

if (monoms->pFirst == nullptr) {

return;

}

int degree\_x = 0;

int degree\_y = 0;

int degree\_z = 0;

while (!monoms->Is\_Ended()) {

TMonom m = monoms->getCurrent()->data;

if (m.coeff > 0) {

if (monoms->getCurrent() != monoms->pFirst) {

\_polynom.append("+");

\_polynom.append(to\_string(m.coeff));

}

else {

\_polynom.append(to\_string(m.coeff));

}

}

else if (m.coeff < 0) {

\_polynom.append(to\_string(m.coeff));

}

degree\_x = m.wrap\_degree / 100;

degree\_y = (m.wrap\_degree % 100) / 10;

degree\_z = (m.wrap\_degree % 10);

if (degree\_x != 0) {

if (degree\_x == 1) {

\_polynom.append("\*x");

}

else {

\_polynom.append("\*x^");

\_polynom.append(to\_string(degree\_x));

}

}

if (degree\_y != 0) {

if (degree\_y == 1) {

\_polynom.append("\*y");

}

else {

\_polynom.append("\*y^");

\_polynom.append(to\_string(degree\_y));

}

}

if (degree\_z != 0) {

if (degree\_z == 1) {

\_polynom.append("\*z");

}

else {

\_polynom.append("\*z^");

\_polynom.append(to\_string(degree\_z));

}

}

monoms->next();

}

monoms->reset();

polynom = \_polynom;

}

string TPolynom::GetPolynomString()const {

return polynom;

}

TPolynom::TPolynom(const TPolynom& obj) {

monoms = new TRingList<TMonom>(\*obj.monoms);

polynom = obj.GetPolynomString();

}

TPolynom::~TPolynom() {

delete monoms;

polynom = "";

}

TPolynom TPolynom::operator+(const TPolynom& Q) {

TPolynom R;

while (!this->monoms->Is\_Ended() && !Q.monoms->Is\_Ended()) {

TMonom mThis = this->monoms->getCurrent()->data;

TMonom mQ = Q.monoms->getCurrent()->data;

if (mThis.wrap\_degree < mQ.wrap\_degree) {

Node<TMonom>\* pNode = R.monoms->search(mThis);

if (pNode == nullptr) {

R.monoms->insert\_last(mThis);

}

this->monoms->next();

}

else if (mThis.wrap\_degree > mQ.wrap\_degree) {

Node<TMonom>\* pNode = R.monoms->search(mQ);

if (pNode == nullptr) {

R.monoms->insert\_last(mQ);

}

Q.monoms->next();

}

else{

TMonom result = mThis + mQ;

Node<TMonom>\* pNode = R.monoms->search(result);

if ( pNode == nullptr && result.coeff != 0) {

R.monoms->insert\_last(result);

}

this->monoms->next();

Q.monoms->next();

}

}

if (this->monoms->Is\_Ended() && !Q.monoms->Is\_Ended()) {

while (!Q.monoms->Is\_Ended()) {

TMonom mQ = Q.monoms->getCurrent()->data;

R.monoms->insert\_last(mQ);

Q.monoms->next();

}

}

else if(Q.monoms->Is\_Ended() && !this->monoms->Is\_Ended()){

while (!this->monoms->Is\_Ended()) {

TMonom mThis= this->monoms->getCurrent()->data;

R.monoms->insert\_last(mThis);

this->monoms->next();

}

}

Q.monoms->reset();

this->monoms->reset();

return TPolynom(R.monoms);

}

TPolynom TPolynom::operator-(const TPolynom& Q) {

TPolynom R;

while (!this->monoms->Is\_Ended() && !Q.monoms->Is\_Ended()) {

TMonom mThis = this->monoms->getCurrent()->data;

TMonom mQ = Q.monoms->getCurrent()->data;

if (mThis.wrap\_degree < mQ.wrap\_degree) {

Node<TMonom>\* pNode = R.monoms->search(mThis);

if (pNode == nullptr) {

R.monoms->insert\_last(mThis);

}

this->monoms->next();

}

else if (mThis.wrap\_degree > mQ.wrap\_degree) {

Node<TMonom>\* pNode = R.monoms->search(mQ);

if (pNode == nullptr) {

mQ = -mQ;

R.monoms->insert\_last(mQ);

}

Q.monoms->next();

}

else {

TMonom result = mThis + (-mQ);

Node<TMonom>\* pNode = R.monoms->search(result);

if (pNode == nullptr && result.coeff != 0) {

R.monoms->insert\_last(result);

}

this->monoms->next();

Q.monoms->next();

}

}

if (this->monoms->Is\_Ended() && !Q.monoms->Is\_Ended()) {

while (!Q.monoms->Is\_Ended()) {

TMonom mQ = Q.monoms->getCurrent()->data;

mQ = -mQ;

R.monoms->insert\_last(mQ);

Q.monoms->next();

}

}

else if (Q.monoms->Is\_Ended() && !this->monoms->Is\_Ended()) {

while (!this->monoms->Is\_Ended()) {

TMonom mThis = this->monoms->getCurrent()->data;

R.monoms->insert\_last(mThis);

this->monoms->next();

}

}

Q.monoms->reset();

this->monoms->reset();

return TPolynom(R.monoms);

}

TPolynom TPolynom::operator\*(const TPolynom& Q) {

TPolynom R;

while(!this->monoms->Is\_Ended()){

TMonom mThis = this->monoms->getCurrent()->data;

while(!Q.monoms->Is\_Ended()) {

TMonom mQ = Q.monoms->getCurrent()->data;

if (R.monoms->IsEmpty()) {

TMonom m = mThis \* mQ;

R.monoms = new TRingList<TMonom>();

R.monoms->insert\_first(m);

Q.monoms->next();

}

else {

TMonom result = mThis \* mQ;

Node<TMonom>\* pNode = R.monoms->search(result);

R.monoms->reset();

if (pNode != nullptr) {

pNode->data = pNode->data + result;

Q.monoms->next();

}

else {

R.monoms->insertion\_sort(result);

Q.monoms->next();

}

}

}

Q.monoms->reset();

this->monoms->next();

}

this->monoms->reset();

return TPolynom(R.monoms);

}

const TPolynom& TPolynom::operator=(const TPolynom& Q) {

polynom = Q.GetPolynomString();

int size = polynom.size();

if (size == 0) {

monoms = new TRingList<TMonom>();

return \*this;

}

TMonom mQ = Q.monoms->getCurrent()->data;

monoms = new TRingList<TMonom>(new Node<TMonom>(mQ));

Q.monoms->next();

Node<TMonom>\* tmp\_this = monoms->getCurrent();

while (!Q.monoms->Is\_Ended()) {

TMonom mQ = Q.monoms->getCurrent()->data;

monoms->insert\_last(mQ);

Q.monoms->next();

}

return \*this;

}

bool TPolynom::operator==(const TPolynom& Q)const {

if (monoms->GetSize() != Q.monoms->GetSize()) {

return false;

}

while (!Q.monoms->Is\_Ended()) {

TMonom mThis = this->monoms->getCurrent()->data;

TMonom mQ = Q.monoms->getCurrent()->data;

if (mThis == mQ && mThis.coeff == mQ.coeff) {

monoms->next();

Q.monoms->next();

}

else{

monoms->reset();

Q.monoms->reset();

return false;

}

}

monoms->reset();

Q.monoms->reset();

return true;

}

double TPolynom::operator()(double x, double y, double z) {

ArithmeticExpression my\_polynom(polynom);

vector<string> operands = my\_polynom.GetOperands();

map<string, double> values;

if (values\_xyz.size() == 0) {

ArithmeticExpression my\_polynom(polynom);

vector<string> my\_operands = my\_polynom.GetOperands();

auto it\_begin{ my\_operands.begin() };

auto it\_end{ my\_operands.end() };

while (it\_begin != it\_end) {

if (\*it\_begin == "x") {

values\_xyz.insert({ \*it\_begin, x });

}

else if (\*it\_begin == "y") {

values\_xyz.insert({ \*it\_begin, y });

}

else if (\*it\_begin == "z") {

values\_xyz.insert({ \*it\_begin, z });

}

it\_begin++;

}

values = my\_polynom.SetOperands\_v2(operands, values\_xyz);

}

else {

values = my\_polynom.SetOperands\_v2(operands, values\_xyz);

}

double res = my\_polynom.Calculate(values);

return res;

}

TPolynom TPolynom::differentiate\_by\_x()const {

TPolynom R(\*this);

int degree\_x = 0;

int degree\_y = 0;

int degree\_z = 0;

int new\_wrap\_degree = 0;

while (!R.monoms->Is\_Ended()) {

TMonom mR = R.monoms->getCurrent()->data;

degree\_x = mR.wrap\_degree / 100;

if (degree\_x == 0) {

R.monoms->remove(mR);

}

else {

mR.coeff= mR.coeff \* degree\_x;

degree\_x--;

degree\_y = (mR.wrap\_degree % 100) / 10;

degree\_z = (mR.wrap\_degree % 10);

mR.wrap\_degree= degree\_x \* 100 + degree\_y \* 10 + degree\_z;

R.monoms->getCurrent()->data = mR;

R.monoms->next();

}

}

R.monoms->reset();

return TPolynom(R.monoms);

}

TPolynom TPolynom::differentiate\_by\_y()const {

TPolynom R(\*this);

int degree\_x = 0;

int degree\_y = 0;

int degree\_z = 0;

int new\_wrap\_degree = 0;

while (!R.monoms->Is\_Ended()) {

TMonom mR = R.monoms->getCurrent()->data;

degree\_y = (mR.wrap\_degree % 100) / 10;

if (degree\_y == 0) {

R.monoms->remove(mR);

}

else {

mR.coeff= mR.coeff \* degree\_y;

degree\_y--;

degree\_x = mR.wrap\_degree / 100;

degree\_z = (mR.wrap\_degree % 10);

mR.wrap\_degree= degree\_x \* 100 + degree\_y \* 10 + degree\_z;

R.monoms->getCurrent()->data = mR;

R.monoms->next();

}

}

R.monoms->reset();

return TPolynom(R.monoms);

}

TPolynom TPolynom::differentiate\_by\_z()const {

TPolynom R(\*this);

int degree\_x = 0;

int degree\_y = 0;

int degree\_z = 0;

int new\_wrap\_degree = 0;

while (!R.monoms->Is\_Ended()) {

TMonom mR = R.monoms->getCurrent()->data;

degree\_z = (mR.wrap\_degree % 10);

if (degree\_z == 0) {

R.monoms->remove(mR);

}

else {

mR.coeff= mR.coeff \* degree\_z;

degree\_z--;

degree\_x = mR.wrap\_degree / 100;

degree\_y = (mR.wrap\_degree % 100) / 10;

new\_wrap\_degree = degree\_x \* 100 + degree\_y \* 10 + degree\_z;

mR.wrap\_degree=new\_wrap\_degree;

R.monoms->getCurrent()->data = mR;

R.monoms->next();

}

}

R.monoms->reset();

return TPolynom(R.monoms);

}

istream& operator>>(istream& istr, TPolynom& Q) {

int number = 1;

int i = 0;

double coeff = 0;

int wrap\_degree = 0;

int count = 0;

int d\_x = 0; int d\_y = 0; int d\_z = 0;

TRingList<TMonom>\* new\_list\_monoms = nullptr;

cout << "Welcome to the polynomial builder!" << endl;

cout << "To create a polynomial you will need to enter " << endl;

cout << "corresponding coefficients of the monomial and degree ";

cout << "variables." << endl;

cout << endl;

cout << "Specify the number of monomials to create the polynomial: ";

do {

cin >> count;

} while (count <= 0);

cout << endl;

cout << "Let`s get started!" << endl;

cout << "| "<< endl;

cout << "| " << endl;

cout << "| Constraints: " << endl;

cout << "| 1)The coefficient must not be equal to 0. " << endl;

cout << "| 2)The degree of the variable is 0 <= x,y,z <= 9"<< endl;

while (i < count) {

cout << "| " << endl;

cout << "| Set the coefficient of the monomial! " << endl;

do {

cout << "| coeff = ";

cin >> coeff;

cout << "| " << endl;

cout << "| " << endl;

if (coeff == 0) {

cout << "| Wrong coeff! Try again. " << endl;

}

} while (coeff == 0);

cout << "| Determine the degrees of the variables x, y, z! "<< endl;

cout << "| " << endl;

do {

cout << "| Degree x = ";

cin >> d\_x;

cout << "| " << endl;

cout << "| " << endl;

cout << "| Degree y = ";

cin >> d\_y;

cout << "| " << endl;

cout << "| " << endl;

cout << "| Degree z = ";

cin >> d\_z;

cout << "| " << endl;

cout << "| " << endl;

if ((d\_x < 0 || d\_x > 9) || (d\_y < 0

|| d\_y > 9) || (d\_z < 0 || d\_z > 9)) {

cout << "| Wrong degrees! Try again. " << endl;

cout << "| " << endl;

}

} while

(

(d\_x < 0 || d\_x > 9) || (d\_y < 0 || d\_y > 9) || (d\_z < 0 || d\_z > 9)

);

wrap\_degree = d\_x \* 100 + d\_y \* 10 + d\_z;

cout << "| " << endl;

cout << "| Data monom " << number << ":" << " " << endl;

cout << "| " << endl;

cout << "| coeff = " << coeff << endl;;

cout << "| wrap\_degree = " << wrap\_degree << endl;;

cout << "| " << endl;

cout << "| " << endl;

cout << "| " << endl;

cout << "==================================================================" << endl;

if (new\_list\_monoms == nullptr) {

TMonom m(coeff, wrap\_degree);

new\_list\_monoms = new TRingList<TMonom>();

new\_list\_monoms->insert\_first(m);

}

else {

TMonom m(coeff, wrap\_degree);

new\_list\_monoms->insertion\_sort(m);

}

number++;

i++;

}

TPolynom new\_polynom(new\_list\_monoms);

Q = new\_polynom;

cout << "FLAG END" << endl;

return istr;

}

ostream& operator<<(ostream& ostr, const TPolynom& Q) {

cout << Q.GetPolynomString() << endl;

cout << endl;

return ostr;

}

Пример:

int main() {

cout << "The polynomials P and Q:" << endl << endl;

TPolynom P("2\*x^3\*y^3\*z+y^2\*z^2-7\*x");

TPolynom Q("2\*y^3\*z^3-5\*z^3\*x^3+10\*y");

TPolynom U\_P("");

cout << "P(x,y,z) = " << P << endl;

cout << "Q(x,y,z) = " << Q << endl;

cout << "Getting copy polynomial P..." << endl;

TPolynom P\_copy(P);

cout << "P\_copy(x,y,z) = " << P\_copy << endl << endl;

cout << "Check polynomials P and Q on equality" << endl << endl;

if (P == Q) {

cout << "P equal Q!" << endl;

}

else {

cout << "P not equal Q!" << endl;

}

cout << endl;

cout << "Check polynomials P and P\_copy on equality" << endl << endl;

if (P == P\_copy) {

cout << "P equal P\_copy!" << endl;

}

else {

cout << "P not equal P\_copy!" << endl;

}

cout << endl;

cout << endl;

cout << "Before we start performing operations on polynomials," << endl;

cout << "create your own polynomial: " << endl;

cin >> U\_P;

cout << "User polynom: " << endl;

cout << "U\_P(x,y,z) = ";

cout << U\_P;

TPolynom res1; TPolynom res2;

TPolynom res3;

TPolynom res4; TPolynom res5;

TPolynom res6; TPolynom res7;

cout << endl;

cout << "Let`s start!" << endl;

cout << endl;

cout << "Operation +:" << endl;

res1 = P + U\_P;

cout << res1;

cout << endl;

cout << "Operation -:" << endl;

res2 = P - Q;

cout << res2;

cout << endl;

cout << "Operation \*:" << endl;

res3 = P \* Q;

cout << res3;

cout << endl;

cout << "For your polynom set point which you want calculate value polynom: " << endl;

double x; double y; double z;

cout << "x = ";

cin >> x;

cout << "y = ";

cin >> y;

cout << "z = ";

cin >> z;

cout << "U\_P(" << x << "," << y << "," << z << ")" << " = " << U\_P(x, y, z) << endl;//

cout << "Time for differentiation!" << endl;

cout << "Your polynom(x,y,z) = " << U\_P;

cout << "Operation differentiation by x:" << endl;

res4 = U\_P.differentiate\_by\_x();

cout << res4;

cout << endl;

cout << "Operation differentiation by y:" << endl;

res6 = U\_P.differentiate\_by\_y();

cout << res6;

cout << endl;

cout << "Operation differentiation by z:" << endl;

res7 = U\_P.differentiate\_by\_z();

cout << res7;

cout << endl;

return 0;

}