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:

**«Полиномы»**

**Выполнил:** студентка группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Резанцева А. А./

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc164113924)

[1 Постановка задачи 4](#_Toc164113925)

[2 Руководство пользователя 5](#_Toc164113926)

[2.1 Приложение для демонстрации работы звена 5](#_Toc164113927)

[2.2 Приложение для демонстрации работы списка 5](#_Toc164113928)

[2.3 Приложение для демонстрации работы кольцевого списка 8](#_Toc164113929)

[2.4 Приложения для демонстрации работы мономов 8](#_Toc164113930)

[2.5 Приложение для демонстрации работы полиномов 10](#_Toc164113931)

[3 Руководство программиста 13](#_Toc164113932)

[3.1 Описание алгоритмов 13](#_Toc164113933)

[3.1.1 Список 13](#_Toc164113934)

[3.1.2 Кольцевой список c головой 16](#_Toc164113935)

[3.1.3 Моном 19](#_Toc164113936)

[3.1.4 Полином 20](#_Toc164113937)

[3.2 Описание программной реализации 21](#_Toc164113938)

[3.2.1 Описание класса TNode 21](#_Toc164113939)

[3.2.2 Описание класса TList 22](#_Toc164113940)

[3.2.3 Описание класса TRingList 27](#_Toc164113941)

[3.2.4 Описание класса TMonom 29](#_Toc164113942)

[3.2.5 Описание класса TPolynom 32](#_Toc164113943)

[Заключение 37](#_Toc164113944)

[Литература 39](#_Toc164113945)

[Приложения 40](#_Toc164113946)

[Приложение А. Реализация класса TNode 40](#_Toc164113947)

[Приложение Б. Реализация класса TList 40](#_Toc164113948)

[Приложение В. Реализация класса TRingList 47](#_Toc164113949)

[Приложение Г. Реализация класса TMonom 50](#_Toc164113950)

[Приложение Д. Реализация класса TPolinom 54](#_Toc164113951)

[Приложение Е. Sample\_tpolinom 61](#_Toc164113952)

# Введение

Лабораторная работа направлена на изучение обработки полиномов от трёх переменных (х, у, z). Полиномы могут быть использованы для решения многих задач математического анализа, теории вероятностей, линейной алгебры и других областей математики

В данной лабораторной работе студенты будут изучать основные принципы работы алгоритма обработки полиномов и реализовывать его на практике. Это позволит им лучше понять принципы работы связного списка и освоить навыки работы с алгоритмами обработки полиномов.

# Постановка задачи

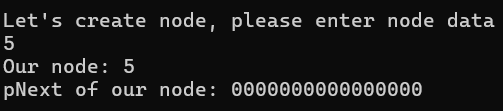
**Цель:**

Цель лабораторной работы - научиться представлять полиномы в виде связных списков, где каждый узел списка содержит моном. Такое представление позволяет эффективно решать задачи сложения, вычитания, умножения и вычисления значений полиномов. Программа должна выполнять операции над полиномами, такие как сложение, вычитание и умножение на число. Для этого нужно реализовать класс TMonom, который позволит хранить в звеньях списка мономы полинома, и класс TPolinom, который будет использоваться класс TMonom и выполнять всю работу с полиномами. Предоставить пример использования и обеспечить работоспособность тестов, покрывающих все методы классов TMonom и TPolinom.

# Руководство пользователя

## Приложение для демонстрации работы звена

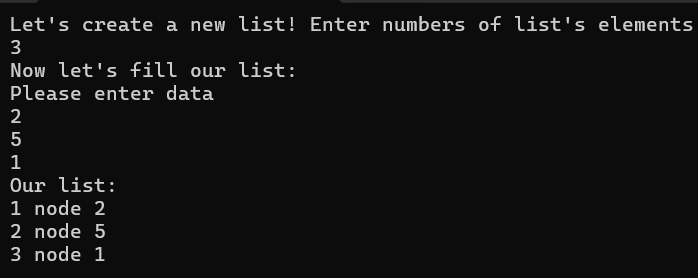
1. Запустите приложение с названием sample\_tnode.exe. В результате появится окно, показанное ниже (рис. 1). Сначала пользователю будет предложено ввести значение для звена, а затем показаны data звена и его указатель pNext.



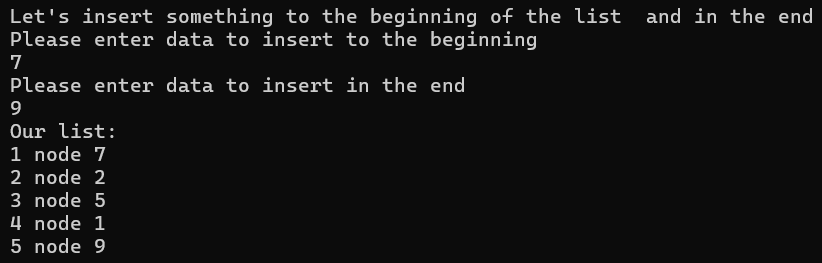
1. Основное окно программы

## Приложение для демонстрации работы списка

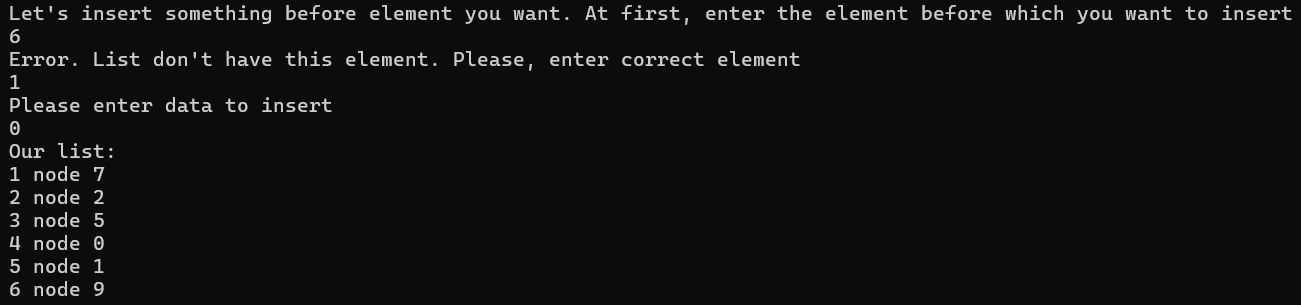
* + - 1. Запустите приложение с названием sample\_tlist.exe. В результате появится окно, показанное ниже. (рис. 2). Пользователю будет предложено ввести количество элементов списка и попросят ввести данные для звеньев списка. После введения данных будет выведен список пользователя.



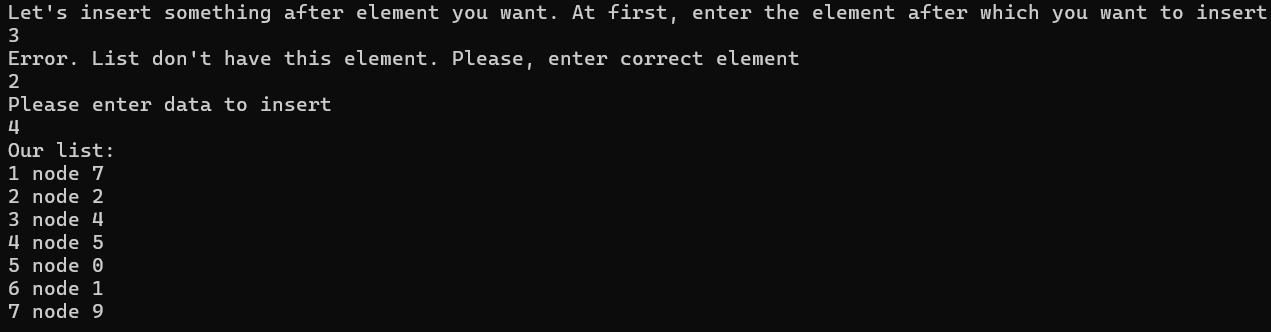
1. Создание списка
2. Далее пользователю предложат воспользоваться вставкой в начало и в конец. Программа попросит ввести данные для вставки в начало и в конец. После выполнения функций будет выведен измененный список (рис. 3).



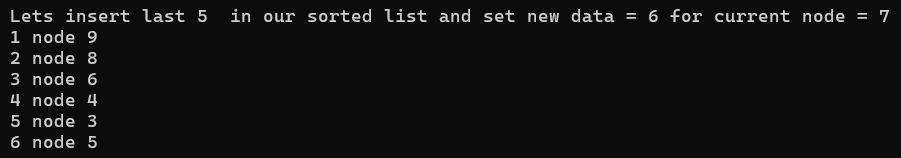
1. Вставка в начало и в конец списка
2. Далее показана работа функций вставки перед указанным звеном. Пользователя просят ввести сначала данные звена, перед которым будет выполнена вставка. Если будут введены данные, которых нет в списке, то попросят ввести еще раз. Далее пользователь введет данные звена, которые будут вставлены. После будет показан измененный список. (рис. 4).



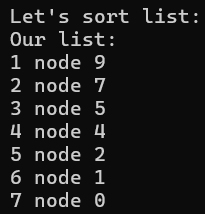
1. Вставка до указанного звена
2. Далее показана работа функций вставки после указанного звена. Пользователя просят ввести сначала данные звена, после которого будет выполнена вставка. Если будут введены данные, которых нет в списке, то попросят ввести еще раз. Далее пользователь введет данные звена, которые будут вставлены. После будет показан измененный список (рис. 5).



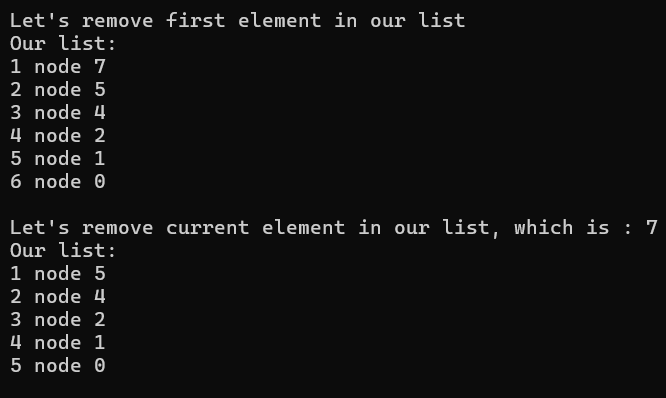
1. Вставка после указанного звена
2. Ниже предствален результат работы сортировки списка (рис. 6).



1. Вставка в конец списка, изменение указателя на текущее звено, вставка после текущего звена
2. Далее показан пример работы функции вставки после текущего звена списка (рис. 7).



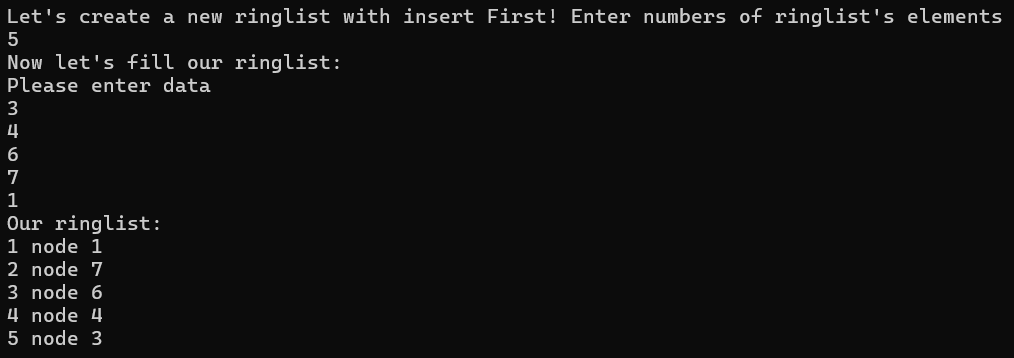
1. Сортировка списка
2. Ниже представлен результат функций удаления текущего и первого звена списка (рис. 8).



1. Удаление первого и текущего звена списка

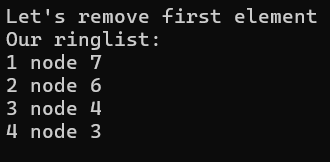
## Приложение для демонстрации работы кольцевого списка

* + - 1. Запустите приложение с названием sample\_tringlist.exe. В результате появится окно, показанное ниже (рис. 9). Пользователю будет предложено создать кольцевой список с помощью функции вставки в начало. Для начала попросят ввести количество элементов в списке и потом сами данные. После будет выведен кольцевой список.



1. Создание кольцевого списка

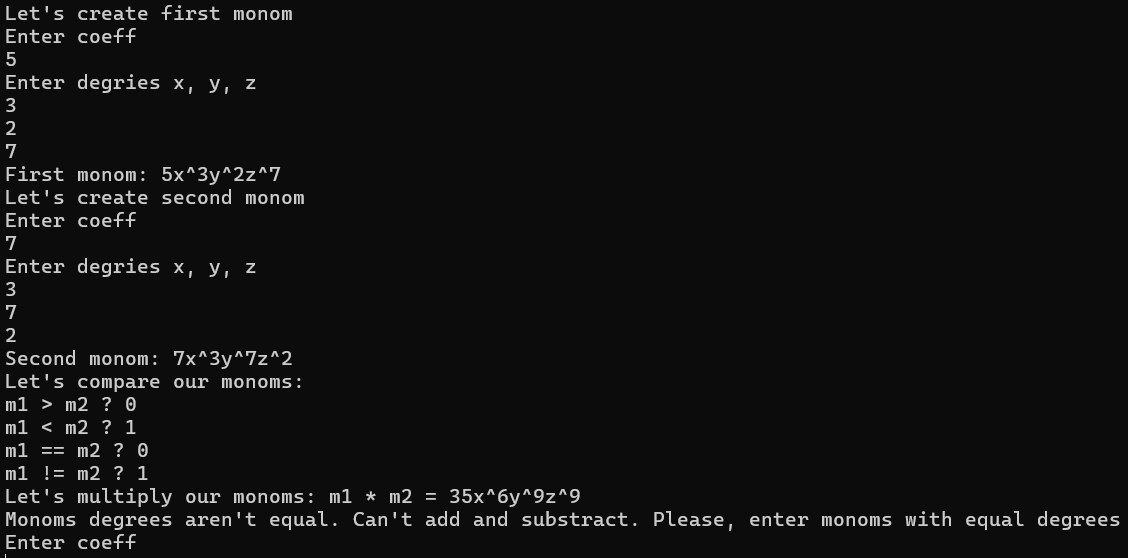
2. Далее представлена работа удаления «первого» звена кольцевого списка (рис. 10).



1. Удаление первого звена в кольцевом списке

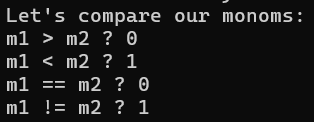
## Приложения для демонстрации работы мономов

* + - 1. Запустите приложение с названием sample\_tmonom.exe. В результате появится окно, показанное ниже (рис. 11). Пользователю будет предложено ввести коэффициенты и степени двух мономов. После этого пользовательские мономы будут выведены на экран.



1. Создание мономов

2. Далее будут выведены результаты сравнения введенных мономов (рис. 12).



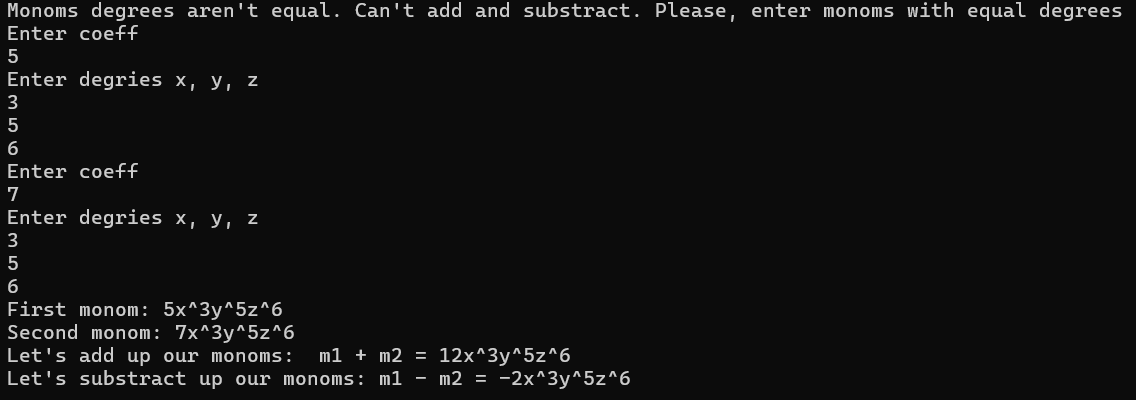
1. Сравнение мономов

3. Далее будет выведен результат умножения мономов (рис. 13).



1. Умножение мономов

4. Далее, если степени мономов не равны, то программа попросит ввести новые мономы с равными степенями, чтобы выполнить операции сложения и вычитания. Результаты операций будут выведены на экран (рис. 14).



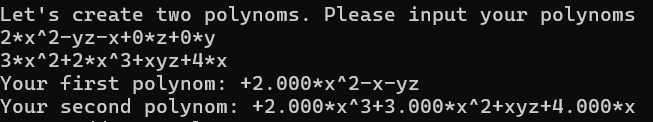
1. Сложение и вычитание мономов

## Приложение для демонстрации работы полиномов

1. Запустите приложение с названием sample\_tpolynom.exe. В результате появится окно, показанное ниже, вам будет предложено ввести два полинома.

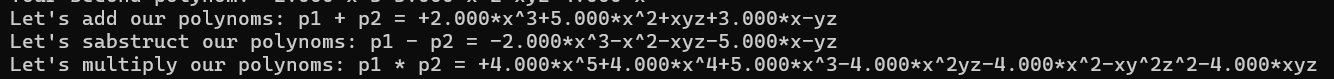


(рис. 15).

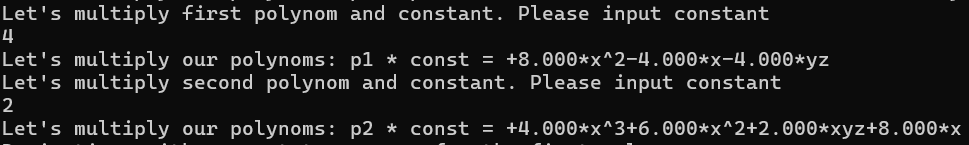




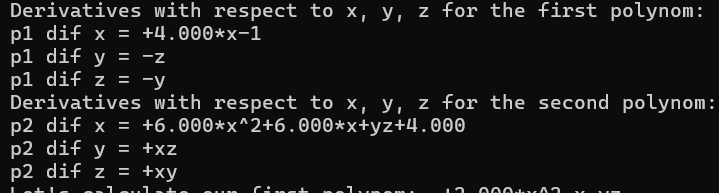
1. Создание полиномов
2. Далее будут выведены результаты сложения, вычитания и умножения полиномов   
   (рис. 16).



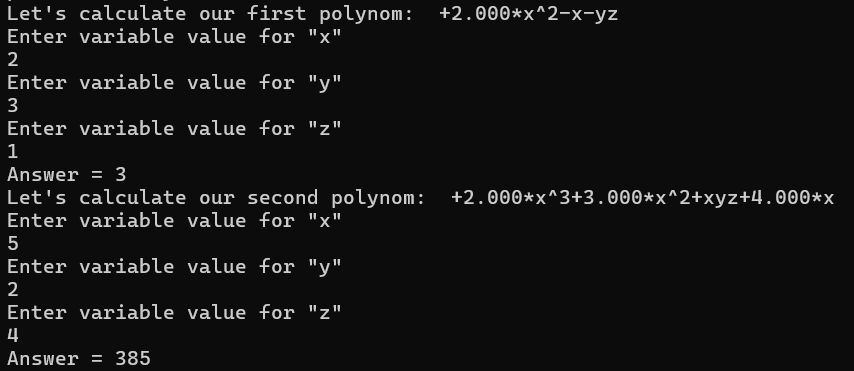
1. Сложение, вычитание и умножение полиномов
2. После пользователю предложат ввести константы, на которые будут умножены введенные полиномы (рис. 17).



1. Умножение полинома на число
2. Далее будут выведены результаты взятия производной по каждой из трех переменной полиномов (рис. 18).



1. Взятие производной по x,y,z
2. В конце предложат ввести значения переменных, чтобы посчитать значения полиномов (рис. 19).



1. Вычисление значения полинома

# Руководство программиста

## Описание алгоритмов

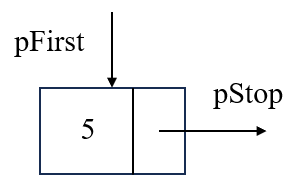
### Список

Список – это динамическая структура данных, которая состоит из звеньев, содержащих данные и ключ-указатель. Структура данных поддерживает операции такие как: вставка в начало, конец списка, вставка до и после элемента. Также присутствуют операции поиска, удаления из списка, проверка на пустоту, полноту.

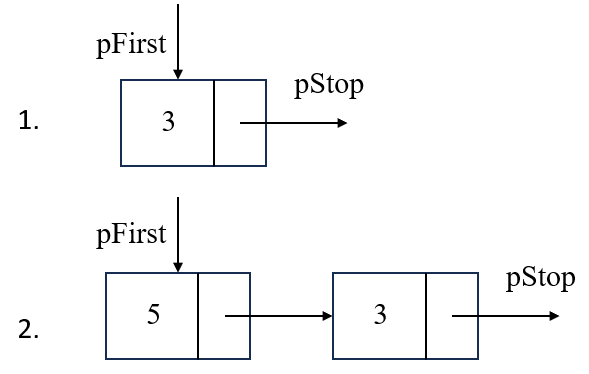
**Примечание:** По умолчанию, первый элемент является текущим, в ходе работы со списком указатель на текущий может изменяться.

**Операция добавления в начало**

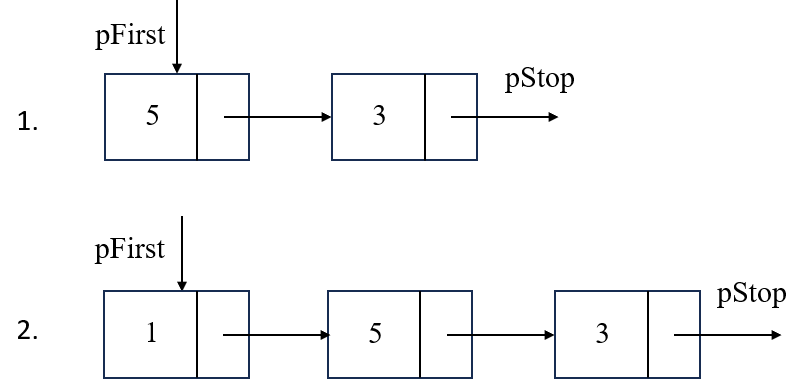
Операция добавления элемента реализуется при помощи указателя на первый элемент. Если структура хранения пуста, то создаем новый элемент, иначе создаём новый элемент и переопределяем указатель на начало, он будет указывать на вставленный элемент. Новый элемент будет указывать на следующий, который раньше был первым.



1. Добавление в начало, если список пустой



1. Добавление в начало, если список не пустой



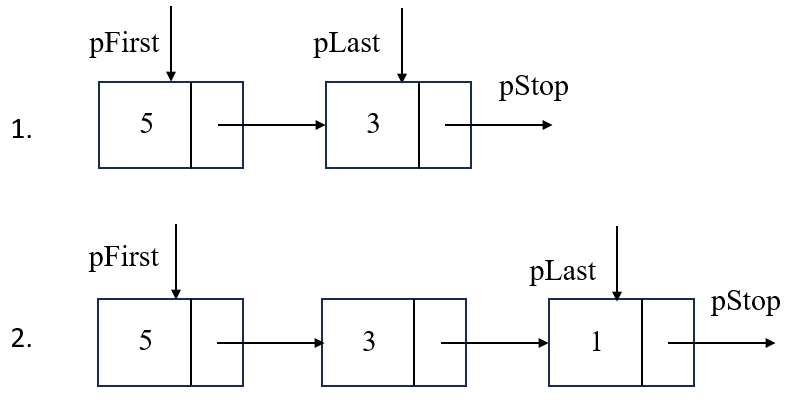
1. Операция добавления элемента (1) в начало

**Операция добавления в конец**

Операция добавления элемента реализуется при помощи указателя на последний элемент.

Если структура хранения пуста, то создаем новый элемент с помощью вставки в начало (рис. 20).

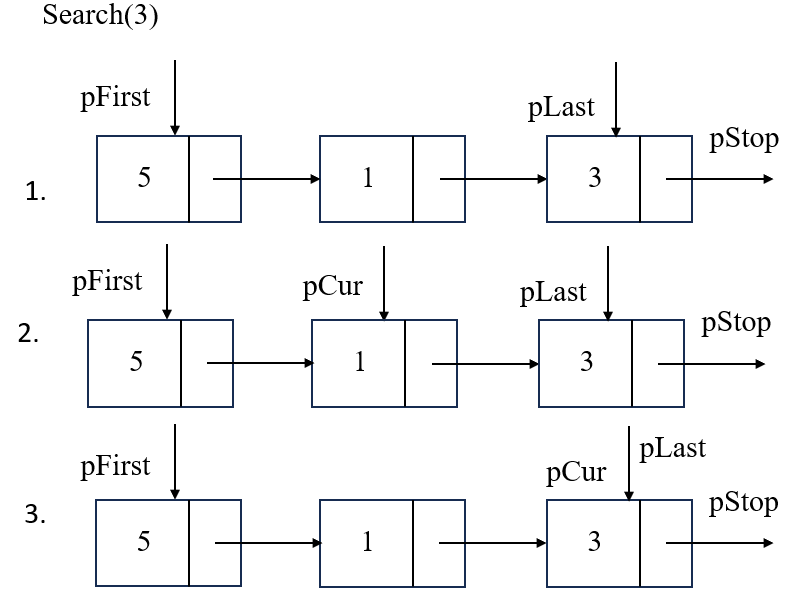
Если список не пуст, то создаем новый элемент, c указателем на pStop. У последнего элемента в списке, на который указывает pLast, меняем указатель на следующий элемент, чтобы он указывал на новый. Переопределяем указатель pLast на вставленный элемент.



1. Операция добавления элемента (1) в конец

**Операция поиска**

Операция поиска ищет элемент в списке. Проходимся от начала списка, сравнивая data звеньев с указанными пользователем. Если data не равна, идем к следующему звену, меня указатель на текущий pCur, найдя нужное звено, возвращаем указатель на найденный элемент.



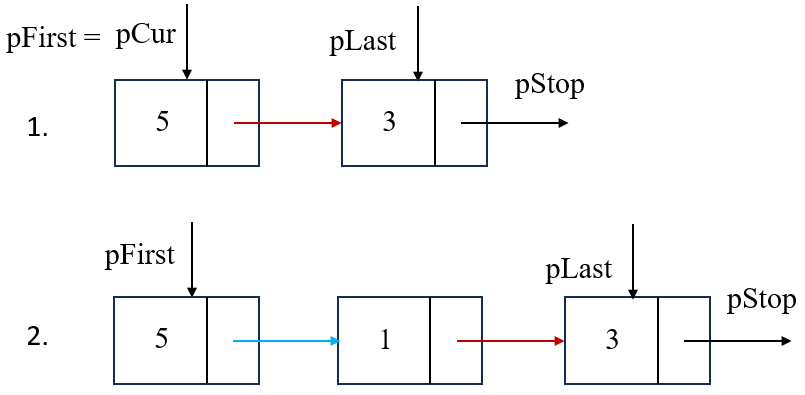
1. Операция поиска элемента (3) в списке

**Операция добавления после**

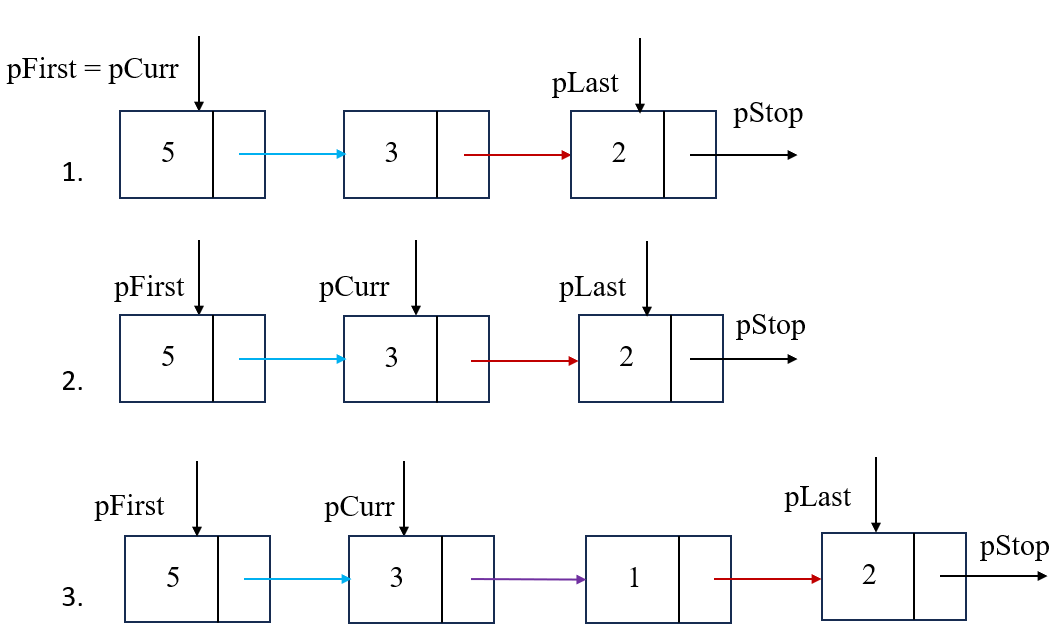
Операция добавления элемента реализуется при помощи указателя на текущий элемент.

Находим элемент, после которого хотим вставить, создаем новое звено и сдвигаем указатели. Разберем на примере ниже (рис. 26):

* Находим элемент, после которого хотим выполнить операцию вставки с помощью операции поиска (рис. 24).
* У нового звена указатель на следующий элемент равен советующему указателю элемента, после которого вставляем (красная стрелка на рис. 26).
* После меняется указатель на следующий элемент у элемента, после которого мы вставляем, он должен указывать на новый, вставленный элемент (фиолетовая стрелка на рис. 26).



1. Операция добавления элемента (1) после текущего. Текщий элемент - первый



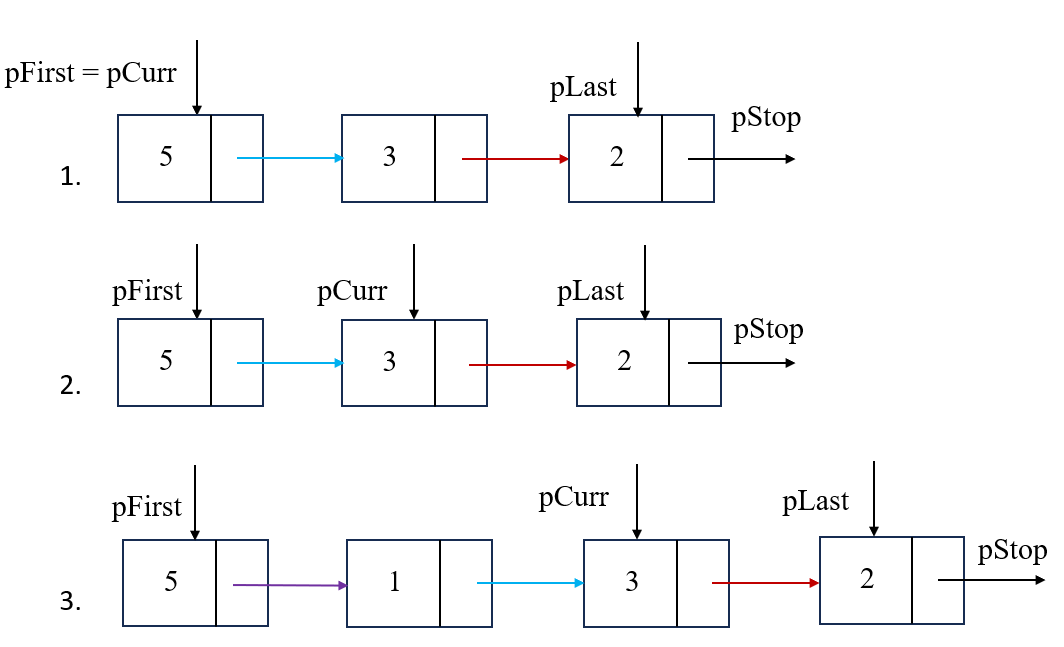
1. Операция добавления элемента (1) после текущего (3)

**Операция добавления перед**

Операция добавления элемента реализуется при помощи указателя на текущий элемент.

Находим элемент, перед которым хотим вставить, создаем новое звено и сдвигаем указатели. Разберем на примере ниже (рис. 27):

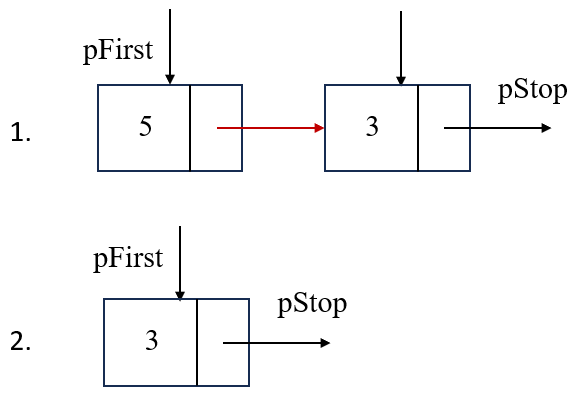
* С помощью операции поиска (рис. 24) находим нужный элемент, перед которым хотим выполнить операцию вставки.
* Создаем новый элемент (1) с указателем на следующий равный указателю на элемент (3), перед которым вставляем (голубая стрелка на рис. 27). Таким образом, мы не разрываем связь в списке.
* Далее переопределяем указатель у элемента (5), который раньше стоял перед найденным (3), он должен указывать на новый (1) (фиолетовая стрелка на рис. 27).



1. Операция вставки (1) перед текущим (3)

**Операция удаления первого элемента**

Операция удаления элемента реализуется при помощи указателя на первый элемент. Удалив первый элемент, следующий за ним, становится первым, то есть на него указывает pFirst (указатель на первый элемент в списке).



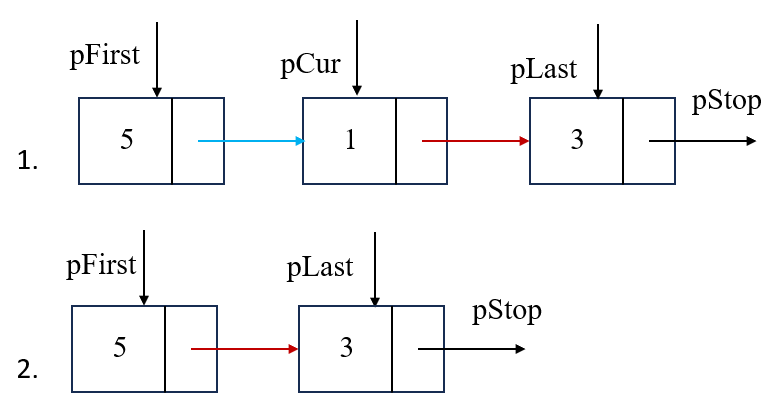
1. Операция удаления первого элемента

**Операция удаления текущего элемента**

Операция удаления элемента реализуется при помощи указателя на текущий элемент.

Разберем на примере ниже (рис. 29):

* Создается вспомогательное звено, с указателем на следующий (3) равным советующему у того элемента (1), который хотим удалить (красная стрелка на рис. 29). Таким образом, мы не потеряем связь, удалив элемент
* Удаляем нужный элемент и его указатель на следующий
* У элемента, который стоял перед удаленным (5), переопределяем указатель на следующий, теперь он равен указателю на следующий у вспомогательного звена, теперь (5) указывает на элемент (3), на который раньше указывал удаленный (1)



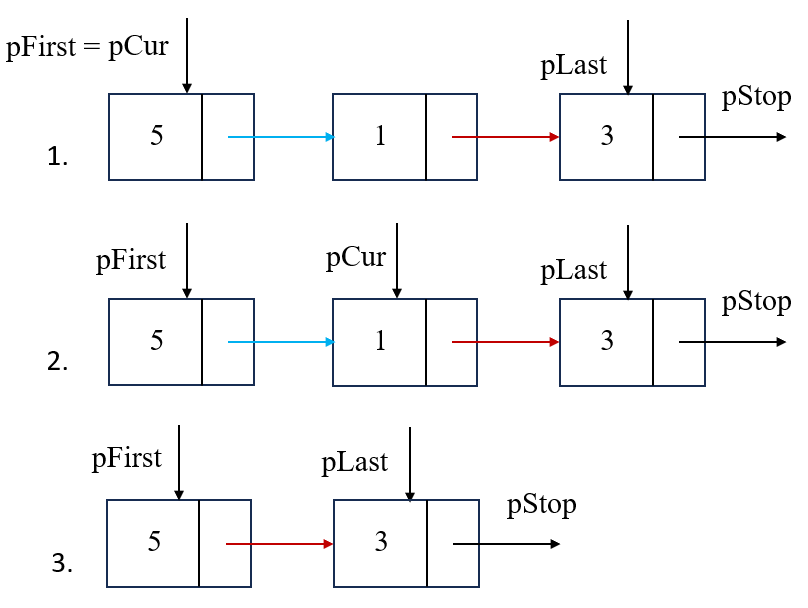
1. Операция удаления текущего элемента (1)

**Операция удаления элемента по его данным**

Ищем в списке звено с нужными данными с помощью операции поиска (рис. 24) и удаляем, меняя указатели, чтобы не потерять связь.

Разберем на примере ниже (рис. 30):

* С помощью операции поиска (рис. 24) находим элемент, который хотим удалить
* Создается вспомогательное звено, с указателем на следующий (3) равным советующему у того элемента (1), который хотим удалить (красная стрелка на рис. 29). Таким образом, мы не потеряем связь, удалив элемент
* Удаляем нужный элемент и его указатель на следующий
* У элемента, который стоял перед удаленным (5), переопределяем указатель на следующий, теперь он равен указателю на следующий у вспомогательного звена, теперь (5) указывает на элемент (3), на который раньше указывал удаленный (1)

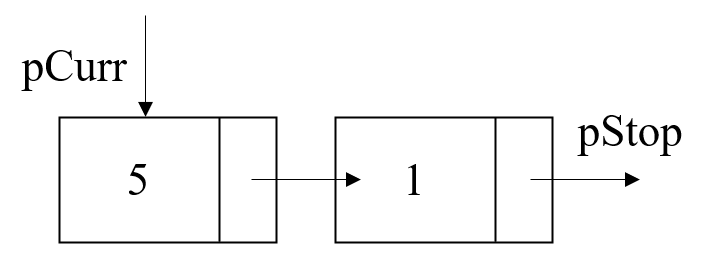


1. Операция удаления элемента (1):

**Операция получения текущего элемента**

Операция получения текущего элемента реализуется при помощи указателя на текущий элемент. Возвращает указатель на текущий.

Пример:

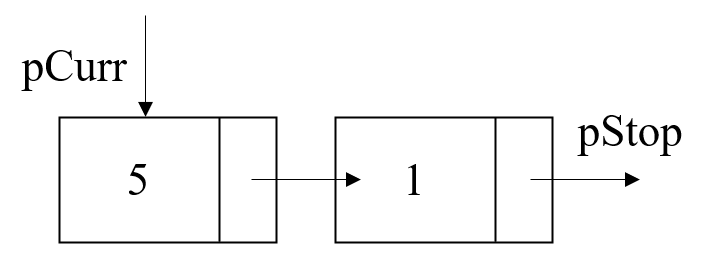


1. Операция взятия элемента если текущий по умолчанию

Результат: Указатель на 5 элемент

**Операция проверки на пустоту**

Операция проверки на полноту проверяет, есть ли хотя бы один элемент в списке. Также реализуется при помощи указателя на первый элемент.



1. Операция проверки на полноту

Результат: false

**Операция сортировки**

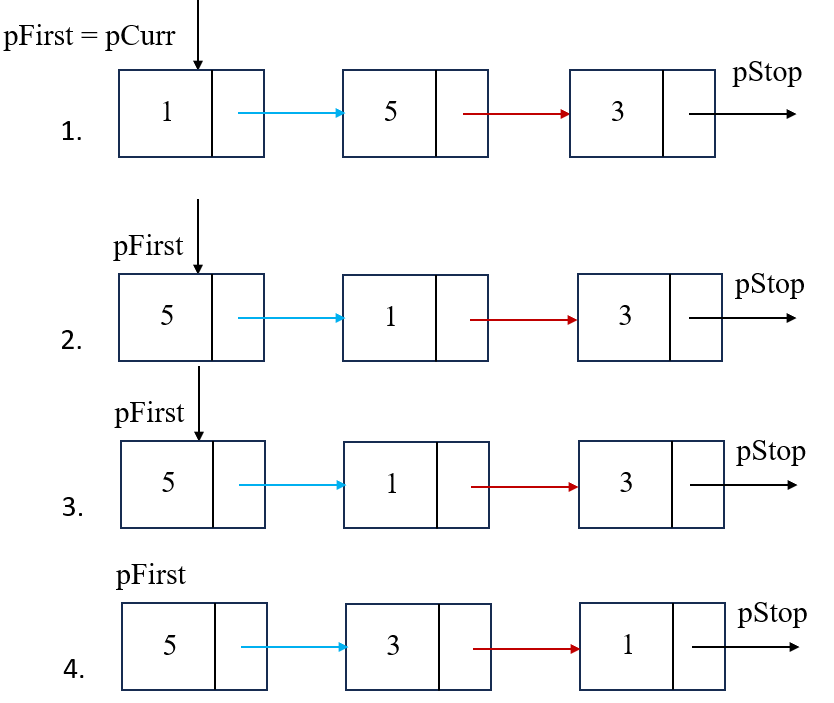
Операция сортировки позволяет сортировать список.

Разберем на примере ниже (рис. 33):

* Начиная с первого звена, сравниваем data со следующими звеньями.
* На первом шаге сравнили (1) и (5) и на втором поменяли data местами, сохранив указатели звеньев.
* На третьем шаге сравнили (1) и (3) и на четвертом поменяли их data местами.

Далее происходит аналогичное сравнение для звеньев (5) и (3).

Таким образом, получили список, отсортированный по убыванию.



1. Сортировка по убыванию

### Кольцевой список c головой

У кольцевого односвязного списка в отличии от односвязного списка, есть указатель на фиктивную голову. Это позволяет облегчить некоторые операции.

Операции, доступные с данной структурой хранения, следующие: добавление элемента, удаление элемента, взять текущий элемент, проверка на пустоту, сортировка, отчистка списка.

**Примечание:** По умолчанию, первый элемент является текущим, в ходе работы со списком указатель на текущий может изменяться. Столбцы с элементом 0 – это фиктивная голова, для наглядности.

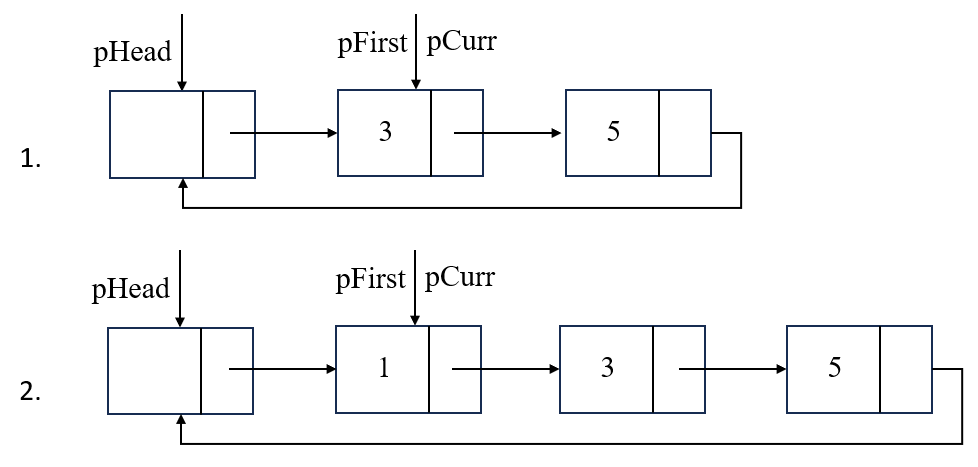
**Операция добавления в начало**

Операция добавления элемента реализуется при помощи указателя на первый элемент.

Если структура хранения пуста, то создаем новый элемент.

Если список не пуст, создаём новый элемент и сдвигаем указатели. Разберем на примере ниже (рис. 34):

* Создаем новое звено (1), которое указывает на первый элемент (3) в списке.
* Ставим указатель pFirst на новый элемент.
* У звена, являющегося фиктивной головой с указателем pHead, меняем указатель на следующий так, чтобы он указывал на вставленный элемент.



1. Операция добавления элемента (1) в начало кольцевого списка с головой

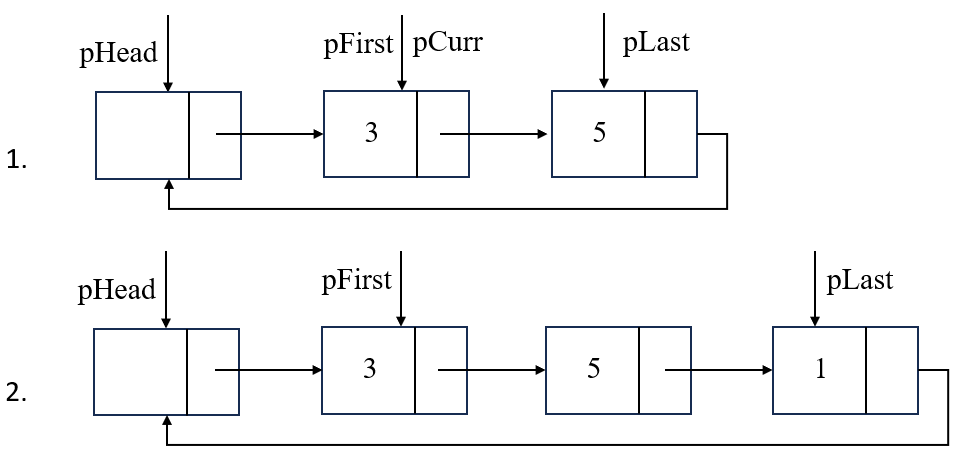
**Операция добавления в конец**

Операция добавления элемента реализуется при помощи указателя на последний элемент.

Если структура хранения пуста, то используем операцию вставки в начало (рис. 34).

Если список не пуст, то создаём новый элемент и сдвигаем указатели. Разберем на примере ниже (рис. 35):

* Создаем новое звено (1), которое указывает на фиктивную голову.
* У последнего элемента (5) с указателем pLast, меняем указатель на следующий так, чтобы он указывал на вставленный (1).
* На новое звено (1) в конце ставим указатель pLast, последнего звена.

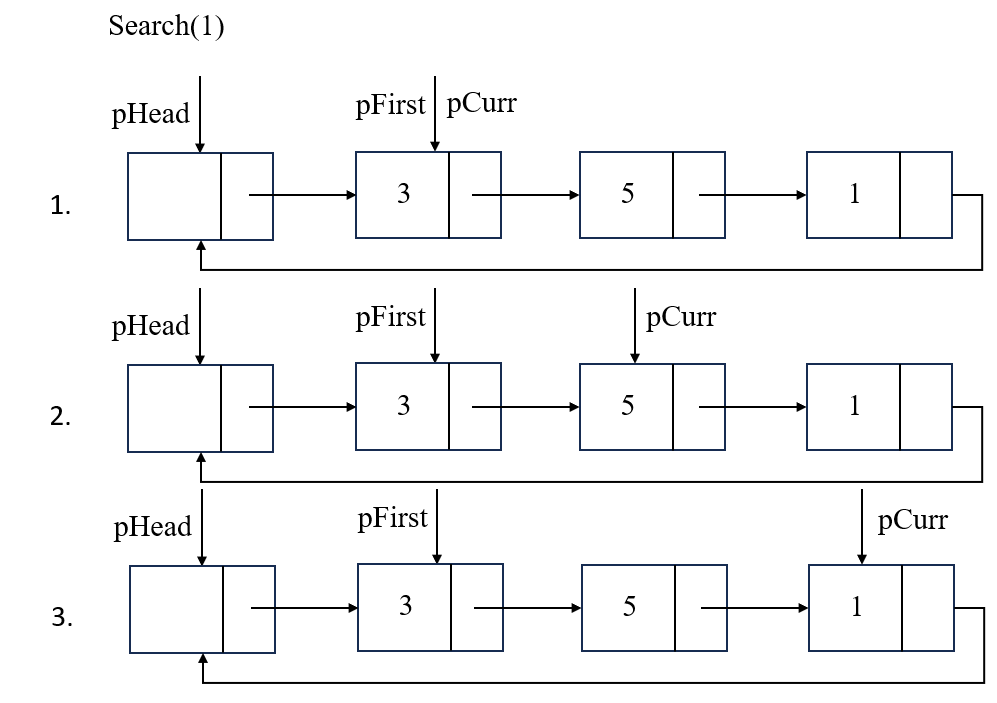


1. Операция добавления элемента (1) в конец

**Операция поиска**

Операция поиска ищет элемент в списке.

Проходимся от начала списка, сравнивая data звеньев с указанными пользователем. Если data не равны, идем к следующему звену, меня указатель на текущий pCurr, найдя нужное звено, возвращаем указатель на найденный элемент.



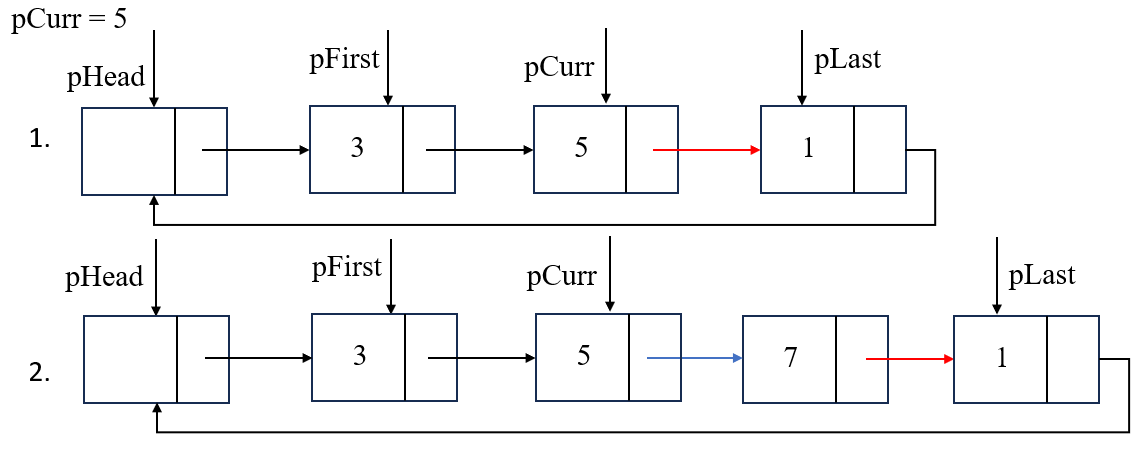
1. Операция поиска элемента (1)

**Операция добавления после текущего**

Операция добавления элемента реализуется при помощи указателя на текущий элемент.

Разберем на примере ниже (рис. 37):

* Создаем новое звено (7), которое будет указывать на звено, следующее за текущим (красная стрелка на рис. 37 ) .
* У текущего элемента (5) переопределяем указатель на следующий так, чтобы он указывал на вставленный (7) (синяя стрелка на рис. 37).



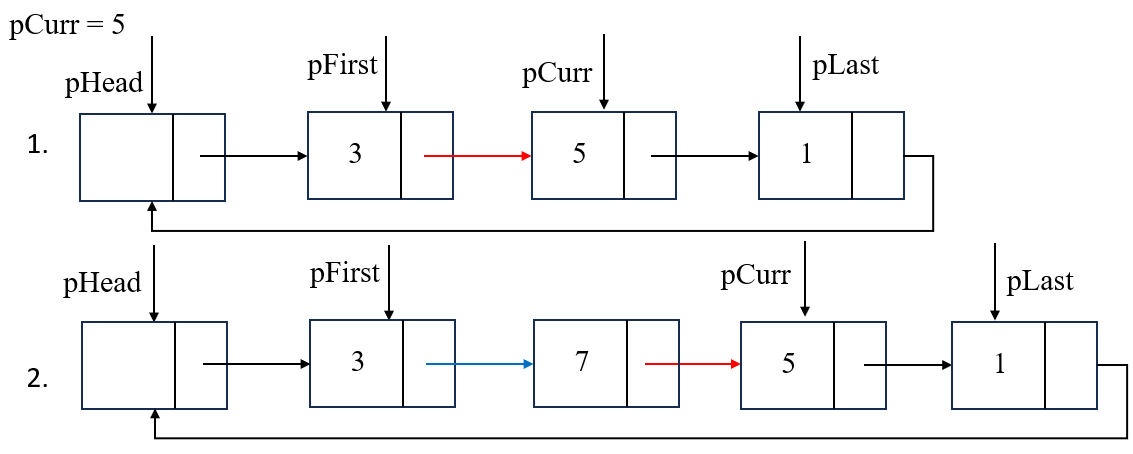
1. Операция добавления элемента (7) после текущего (5)

**Операция добавления перед текущим**

Операция добавления элемента реализуется при помощи указателя на текущий элемент. Имеем указатель на текущий, создаем новое звено и сдвигаем указатели текущего и нового.

Разберем на примере ниже (рис. 38):

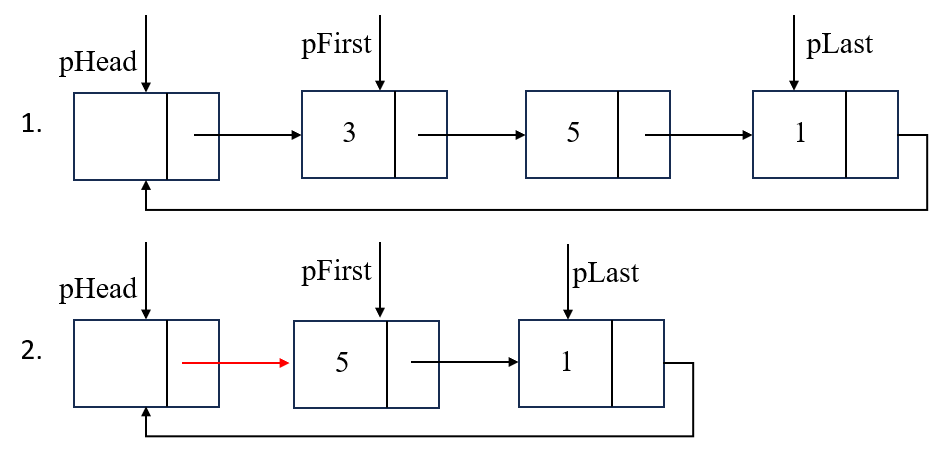
* Создаем новое звено (7), с указателем на следующий равный указателю на элемент (5), перед которым вставляем (красная стрелка на рис. 38). Этот указатель берем у элемента, стоящего перед текущим. Таким образом, мы не разрываем связь в списке.
* Далее переопределяем указатель у элемента (3), который раньше стоял перед текущим (5), он должен указывать на новый (7) (синяя стрелка на рис. 38).



1. Операция добавления элемента (7) перед текущим (5)

**Операция удаления первого элемента**

Операция удаления элемента реализуется при помощи указателя на первый элемент pFirst. Удалив первый элемент, следующий за ним, становится первым. На него теперь будет указывать pFirst, и фиктивная голова будет своим указателем на следующий указывать на новый первый элемент.



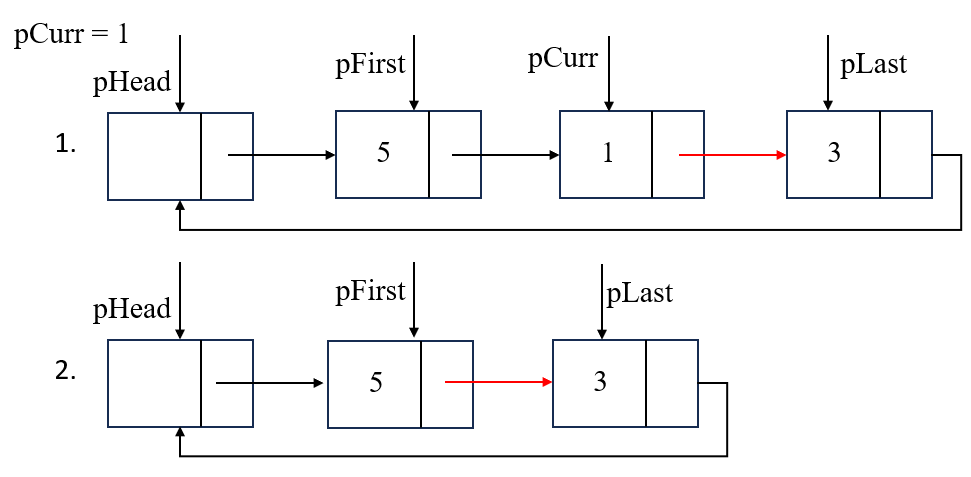
1. Операция удаления первого элемента

**Операция удаления текущего элемента**

Операция удаления элемента реализуется при помощи указателя на текущий элемент.

Разберем на примере ниже (рис. 40):

* Создается вспомогательное звено, с указателем на следующий (3) равным советующему у того элемента (1), который хотим удалить (красная стрелка на рис. 40). Таким образом, мы не потеряем связь, удалив элемент.
* Удаляем нужный элемент и его указатель на следующий
* У элемента, который стоял перед удаленным (5), переопределяем указатель на следующий, теперь он равен указателю на следующий у вспомогательного звена, теперь (5) указывает на элемент (3), на который раньше указывал удаленный (1)



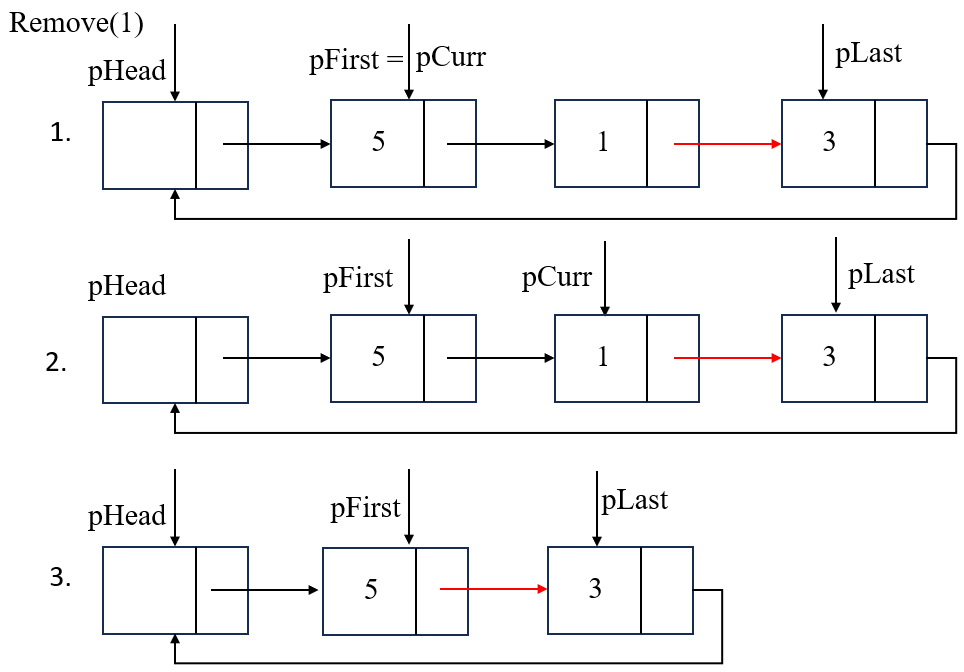
1. Операция удаления текущего (1) элемента

**Операция удаления элемента по его данным**

Ищем в списке звено с нужными данными с помощью операции поиска (рис. 24) и удаляем, меняя указатели, чтобы не потерять связь.

Разберем на примере ниже (рис. 41):

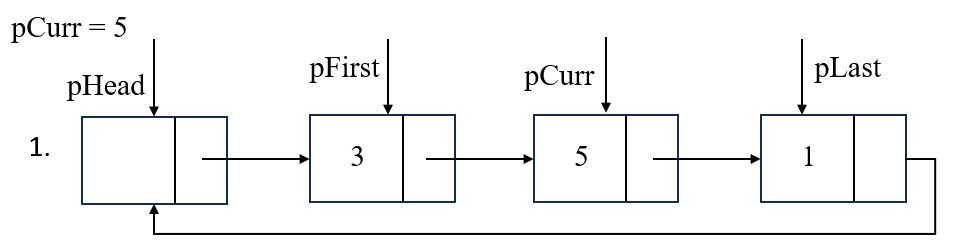
* С помощью операции поиска (рис. 24) находим элемент, который хотим удалить
* Создается вспомогательное звено, с указателем на следующий (3) равным советующему у того элемента (1), который хотим удалить (красная стрелка на рис. 41). Таким образом, мы не потеряем связь, удалив элемент
* Удаляем нужный элемент и его указатель на следующий
* У элемента, который стоял перед удаленным (5), переопределяем указатель на следующий, теперь он равен указателю на следующий у вспомогательного звена, теперь (5) указывает на элемент (3), на который раньше указывал удаленный (1)



1. Операция удаления элемента по его данным (1)

**Операция получения текущего элемента**

Операция взятия элемента текущего элемента также реализуется с помощью указателя на текущий элемент. Возвращает указатель на текущий

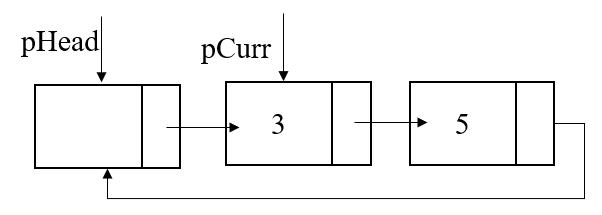


1. Операция взятия текущего элемента

Результат: указатель на (5)

**Операция проверки на пустоту.**

Операция проверки на полноту проверяет, есть ли хотя бы один элемент в списке. Также реализуется при помощи указателя на первый элемент.



1. Операция проверки на полноту

Результат: false

**Операция сортировки**

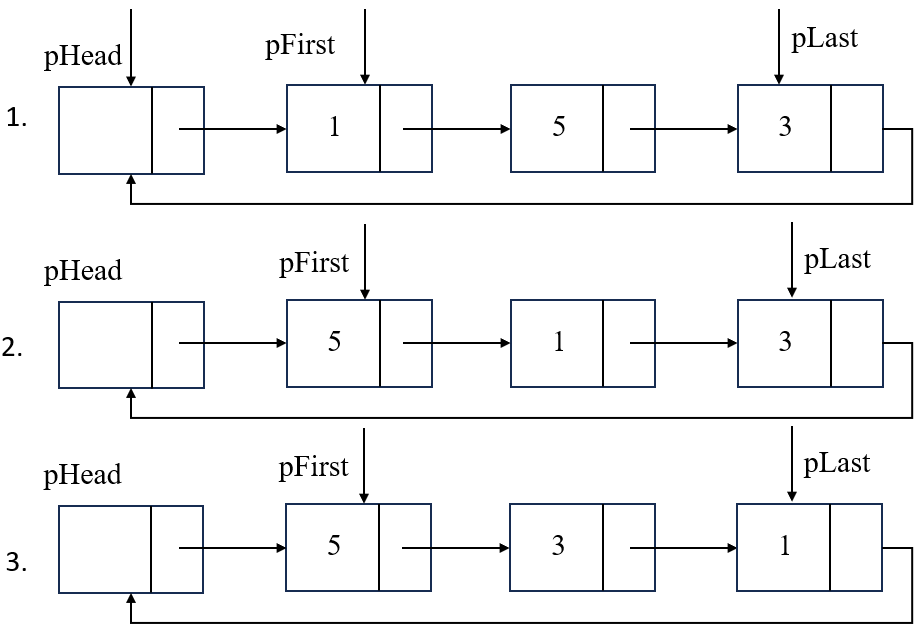
Операция сортировки позволяет сортировать список.

Разберем на примере ниже (рис. 44):

* Начиная с первого звена, пропуская фиктивную голову, сравниваем data со следующими звеньями.
* На первом шаге сравнили (1) и (5) и на втором поменяли data местами, сохранив указатели звеньев.
* На третьем шаге сравнили (1) и (3) и поменяли их data местами.

Далее происходит аналогичное сравнение для звеньев (5) и (3).

Таким образом, получили список, отсортированный по убыванию.



1. Сортировка по убыванию

### Моном

Моном представляет собой структуру данных и имеет такие поля, как коэффициент и свертку степеней. Структура данных поддерживает операции сравнения. В свертке степеней разряд сотен– это степень x, десяток – степень y, единиц – степень z.

**Операция сравнения больше**

Функция сравнивает два монома по степеням и возвращает true или false.

Пример 1: 2\*x^3\*y\*z > 2\*x^2\*y^2\*z

Функция вернет true, так как свертка степеней первого монома равна 311, а свертка степеней второго 221.

Пример 2: 2\*x^3\*y\*z > 2\*x^4\*y^2\*z

Функция вернет false, так как свертка степеней первого монома равна 311, а свертка степеней второго 421.

**Операция сравнения меньше**

Функция сравнивает два монома по степеням и возвращает true или false.

Пример 1: 2\*x^3\*y\*z < 2\*x^2\*y^2\*z

Функция вернет false, так как свертка степеней первого монома равна 311, а свертка степеней второго 221.

Пример 2: 2\*x^3\*y\*z < 2\*x^4\*y^2\*z

Функция вернет true, так как свертка степеней первого монома равна 311, а свертка степеней второго 421.

**Операция сравнения на равенство**

Функция сравнивает по степеням на равенство два монома.

Пример1: 2\*x^3\*y\*z == 2\*x^2\*y^2\*z

Функция вернет false, так как свертка степеней первого монома равна 311, а свертка степеней второго 221.

Пример 2: 2\*x^4\*y^2\*z == 2\*x^4\*y^2\*z

Функция вернет true, так как свертка степеней первого монома равна 421, и свертка степеней второго 421.

**Операция сравнения на неравенство**

Функция сравнивает по степеням на неравенство два монома.

Пример 1: 2\*x^3\*y\*z != 2\*x^2\*y^2\*z

Функция вернет true, так как свертка степеней первого монома равна 311, а свертка степеней второго 221.

Пример 2: 2\*x^4\*y^2\*z != 2\*x^4\*y^2\*z

Функция вернет false, так как свертка степеней первого монома равна 421, и свертка степеней второго 421.

**Операция сложения мономов**

Если правый и левый мономы имеют одинаковую свертку степеней, то создается результирующий моном со степенью мономов, которые складываем, и с коэффициентом, равным сумме коэффициентов левого и правого мономов.

Пример: (x^2y)+(3x^2y)

Результат: 4x^2y

**Операция умножения мономов**

Если сумма свертки степеней левого и правого мономов не меньше 0 и не больше 999, то создается результирующий моном со степенью, равной сумме сверток степеней мономов. Коэффициент результирующего монома равен произведению коэффициентов левого и правого мономов.

Пример: (8y^2)\*( 5y^2z)

Результат: 40y^4z

**Операция вычитания мономов**

Если правый и левый мономы имеют одинаковую свертку степеней, то создается результирующий моном со степенью мономов, которые складываем, и с коэффициентом, равным разности коэффициентов левого и правого мономов.

Пример: (x^2y)-(3x^2y)

Результат: -2x^2y

**Операция дифференцирования мономов**

Для дифференцирования по x, разряд сотен в свертке монома уменьшается на 1 и коэффициент монома умножается на первоначальную степень x.

Для дифференцирования по y, разряд десяток в свертке монома уменьшается на 1 и коэффициент монома умножается на первоначальную степень y.

Для дифференцирования по z, разряд единиц в свертке монома уменьшается на 1 и коэффициент монома умножается на первоначальную степень x.

Пример:

x^4y^5z^3

Результат дифференцирования по х: 4x^3y^5z^3

Результат дифференцирования по y: 5x^4y^4z^3

Результат дифференцирования по z: 3 x^4y^5z^2

### Полином

Программа предоставляет возможности для работы с полиномами на базе мономов: суммирование, разность, произведение, дифференцирование полиномов.

Алгоритм на входе требует строку, которая представляет некоторый полином. Алгоритм допускает наличия трёх независимых переменных, положительные целые степени независимых переменных и вещественные коэффициенты.

**Операция суммирования полиномов**

Полиномы разбиваются на мономы, они сравниваются и создают результирующий полином, путем создания упорядоченного списка мономов. Если свертки мономов равны, то складываются их коэффициенты.

Пример:

(+2.0\*x^2-yz-x) + (+3.0\*x^2+2.0\*x^3+xyz+4.0\*x)

Результат:

+2.0\*x^3+5.0\*x^2+xyz+3.0\*x-yz

**Операция вычитания полиномов**

Используется операция суммирования полиномов, левого полинома и правого полинома, умноженного на -1.

Пример:

(+x^4+3.0\*x+yz+3.0\*y^3z+x^2yz^5) - (+x^3+2.0\*x-yz)

Результат:

+x^4-x^3+x^2yz^5+x+3.0\*y^3z+2.0\*yz

**Операция произведения полиномов**

Мономы умножаются каждый с каждым, степени складываются, коэффициенты умножаются, создавая таким образом новый полином, путем создания упорядоченного списка мономов.

Пример:

(-2x^2 + 3x\*y\*z + 1) \* (3x^2+1)

Результат:

-6x^4 + x^2 + 9x^2yz + 3xyz+ 3x\*y\*z + 2

**Операция дифференцирования полиномов**

Операция дифференцирования полинома согласно математическим правилам. Возможно дифференцирование по независимым переменным x, y или z. Полином разбивается на мономы и к каждому моному применяется операция дифференцирования по одной из трех переменных.

Пример:

+x^4+3.0\*x+yz+3.0\*y^3z+x^2yz^5

Результат дифференцирования по х: +4.0\*x^3+2.0\*xyz^5+3.0

Результат дифференцирования по y: +x^2z^5+9.0\*y^2z+z

Результат дифференцирования по z: +5.0\*x^2yz^4+3.0\*y^3+y

## Описание программной реализации

### Описание класса TNode

template<class TData>

class TNode {

public:

TData data;

TNode\* pNext;

TNode();

TNode(const TData& data\_, const TNode\* pNext\_ = nullptr);

TNode(const TNode& Node);

};

Назначение: представление звена списка.

Поля:

**data**– указатель на массив типа **TData**.

**pNext** – указатель на следующий элемент.

Методы:

TNode();

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: новое звено с инициализированными значениями.

TNode(const T& data\_);

Назначение: инициализация значения **data** звена.

Входные параметры: **data\_** – значение **data**.

Выходные параметры: новое звено с инициализированным значением **data**.

TNode(**const TData& data\_, const TNode\* pNext\_ = nullptr**);

Назначение: инициализация значения data и указателя на следующее звено.

Входные параметры: **data\_** – значение **data**, **pNext\_** – указатель на следующее звено.

Выходные параметры: новое звено с инициализированными значениями **data** и указателем на следующее звено.

~TNode();

Назначение: освобождение выделенной памяти.

### Описание класса TList

template<class TData>

class TList {

protected:

TNode<TData>\* pFirst = nullptr;

TNode<TData>\* pLast = nullptr;

TNode<TData>\* pStop = nullptr;

TNode<TData>\* pCurr = nullptr;

TNode<TData>\* pPrev = nullptr;

public:

TList();

TList(const TNode<TData>\* pFirst\_);

TList(const TList& List);

virtual void Clear();

~TList();

bool is\_Full() const;

void Next();

void Sort();

virtual bool is\_Empty() const;

TNode<TData>\* Search(const TData& data\_);

virtual void InsertFirst(const TData& data\_);

void InsertLast(const TData& data\_);

void InsertBefore(const TData& data\_, const TData& NextData);

void InsertAfter(const TData& data\_, const TData& PrevData);

void Reset();

TNode<TData>\* GetCurrent() const;

TData getCurrData() const;

void setCurrData(const TData& \_data);

const TList<TData>& operator=(const TList<TData>& pList) ;

void InsertBeforeCurrent(const TData& data\_);

void InsertAfterCurrent(const TData& data\_);

void Remove(const TData& data\_);

void RemoveFirst();

void RemoveCurrent();

bool is\_End() const;

**friend std::ostream& operator<<(std::ostream& out, const TList<TData>& list);**

};

Назначение: представление списка.

Поля:

pFirst – указатель на первый элемент списка.

pLast – указатель на последний элемент списка.

pStop – указатель на конец списка.

pCurr – указатель на текущий элемент списка (по умолчанию равен указателю на первый элемент).

pPrev – указатель на предыдущий элемент списка.

Методы:

TList();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

TList(const TList& List);

Назначение: конструктор копирования.

Входные параметры:

List – список, на основе которого создаем новый список.

Выходные параметры: отсутствуют.

TList(const TNode<TData>\* pFirst\_);

Назначение: конструктор с параметрами.

Входные параметры:

node – звено, на основе которого создаем новый список.

Выходные параметры: отсутствуют.

virtual void Clear();

Назначение: отчистка списка.

Входные параметры отсутствуют.

Выходные параметры отсутствуют.

virtual ~TList();

Назначение: деструктор.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

bool is\_Full() const;

Назначение: проверка списка на полноту.

Входные параметры отсутствуют:

Выходные параметры: true – если список полон, false – в противном случае.

void Next();

Назначение: переход к следующему звену.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void Sort();

Назначение: сортировка списка по убыванию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

TNode<TData>\* Search(const TData& data\_);

Назначение: поиск звена с указанным значением.

Входные параметры: **data\_** – искомое значение.

Выходные параметры: указатель на звено с заданным значением, либо **nullptr**.

virtual void InsertFirst(const TData& data\_);

Назначение: вставляет новое звено с данными в начало списка.

Входные параметры: **data** – данные для нового звена.

Выходные параметры: отсутствуют.

void InsertLast(const TData& data\_);

Назначение: вставляет новое звено с данными в конец списка.

Входные параметры: **data\_** – данные для нового звена.

Выходные параметры: отсутствуют.

void InsertBefore(const TData& data\_, const TData& NextData);

Назначение: вставляет новое звено с данными перед звеном с определенными данными.

Входные параметры: **data\_** – данные для нового звена, NextData – данные звена, перед которым будет вставлен новое звено.

Выходные параметры: отсутствуют.

void InsertAfter(const TData& data\_, const TData& PrevData);

Назначение: вставляет новое звено с данными перед звеном с определенными данными.

Входные параметры: **data\_** – данные для нового звена, NextData – данные звена, перед которым будет вставлен новое звено.

Выходные параметры: отсутствуют.

virtual bool is\_Empty() const;

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: true – если список пуст, false – в противном случае.

void Reset();

Назначение: установка текущего звена на первый.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TNode<TData>\* GetCurrent() const;

Назначение: возвращает указатель на текущее звено.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущее звено.

TData getCurrData() const;

Назначение: возвращает данные текущего звена.

Входные параметры: отсутствуют.

Выходные параметры: данные текущего звена.

void setCurrData(const TData& \_data);

Назначение: устанавливает новые данные для текущего звена.

Входные параметры: новая \_data.

Выходные параметры: отсутствуют.

const TList<TData>& operator=(const TList<TData>& pList);

Назначение: операция присваивания.

Входные параметры: pList – список, на основе которого создаем новый список.

Выходные параметры: ссылка на присвоенный список.

void InsertBeforeCurrent(const TData& data\_);

Назначение: вставляет новое звено перед текущим звеном.

Входные параметры: data\_ - данные нового звена.

Выходные параметры: отсутствуют.

void InsertAfterCurrent(const TData& data\_);

Назначение: вставляет новое звено после текущего звена.

Входные параметры: data\_ - данные нового звена.

Выходные параметры: отсутствуют.

void Remove(const TData& data\_);

Назначение: удаляет звено с определенными данными из списка.

Входные параметры: **data\_** – данные звена для удаления.

Выходные параметры: отсутствуют.

void RemoveFirst();

Назначение: удаляет первое звено в списке.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void RemoveCurrent();

Назначение: удаляет текущее звено в списке.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

bool is\_End() const;

Назначение: проверяет, завершен ли список.

Входные параметры: отсутствуют.

Выходные параметры: true – если достигли, false – в противном случае.

**friend std::ostream& operator<<(std::ostream& out, const TList<TData>& list);**

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, list – ссылка на объект типа TList который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

### Описание класса TRingList

template <class TData>

class TRingList : public TList<TData> {

protected:

TNode<TData>\* pHead = nullptr;

public:

TRingList();

TRingList(const TNode<TData>\* pFirst);

TRingList(const TRingList<TData>& ringList);

void InsertFirst(const TData& data\_) override;

void DeleteFirst();

void Clear() override;

bool is\_Empty() const override;

const TRingList<TData>& operator=(const TRingList<TData>& ringList);

friend std::ostream& operator<<(std::ostream& out, TRingList<TData>& ringList);

};

Назначение: представление кольцевого списка.

Поля:

pHead – указатель на головной элемент.

Методы:

TRingList();

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TRingList(const TNode<TData>\* pFirst);

Назначение: конструктор с параметром.

Входные параметры: **pFirst** – указатель на начальное звено.

Выходные параметры: отсутствуют.

TRingList(const TRingList<TData>& ringList);

Назначение: конструктор копирования

Входные параметры: ringList – ссылка на существующий кольцевой список, на основе которого будет создан новый.

Выходные параметры: отсутствуют.

void InsertFirst(const TData& data\_) override;

Назначение: вставляет новое звено с данными в начало списка.

Входные параметры: **data\_** – данные для нового звена.

Выходные параметры: отсутствуют.

void DeleteFirst();

Назначение: удаляет первое звено кольцевого списка.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void Clear() override;

Назначение: отчистка кольцевого списка.

Входные параметры отсутствуют.

Выходные параметры отсутствуют.

bool is\_Empty() const override;

Назначение: проверка кольцевого списка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: true – если список пуст, false – в противном случае.

const TRingList<TData>& operator=(const TRingList<TData>& ringList);

Назначение: операция присваивания.

Входные параметры: ringList – ссылка на кольцевой список, на основе которого создаем новый.

Выходные параметры: ссылка на присвоенный список.

friend std::ostream& operator<<(std::ostream& out, TRingList<TData>& ringList);

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, ringList – ссылка на объект типа TRingList который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

### **Описание класса TMonom**

class TMonom {

private:

double Coeff;

int Degree;

public:

TMonom(double c = 1, int d = 0);

TMonom(const TMonom& m);

TMonom& operator=(const TMonom& m);

std::string getStringMonom() const;

void SetCoeff(double cm);

void SetDegree(int d);

double Pow(double a, double degree);

double GetCoeff();

int GetDegree();

double calculateMonom(std::map<char, double>& variables);

bool operator<(const TMonom& m) const;

bool operator>(const TMonom& m) const;

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

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

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

TMonom operator+(const TMonom& m)const;

TMonom operator-(const TMonom& m)const;

TMonom monom\_defX();

TMonom monom\_defY();

TMonom monom\_defZ();

friend std::ostream& operator<<(std::ostream& out, TMonom& m);

friend std::istream& operator>>(std::istream& istr, TMonom& m);

};

Назначение: представление монома.

Поля:

Coeff – коэффициент монома.

Degree – степень монома.

Методы:

TMonom(double c = 1, int d = 0);

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TMonom(const TMonom& m);

Назначение: конструктор копирования.

Входные параметры: **m** – ссылка на существующий объект **TMonom**.

Выходные параметры: отсутствуют.

TMonom& operator=(const TMonom& m);

Назначение: конструктор копирования.

Входные параметры: **m** – ссылка на существующий объект TMonom.

Выходные параметры: отсутствуют.

std::string getStringMonom() const;

Назначение: получение строкового представления монома.

Входные параметры: отсутствуют.

Выходные параметры: моном в виде строки.

void SetCoeff(double cm);

Назначение: устанавливает значение коэффициента.

Входные параметры: cm – новое значение коэффициента.

Выходные параметры: отсутствуют.

void SetDegree(int d);

Назначение: устанавливает значение степени.

Входные параметры: d – новое значение степени.

Выходные параметры: отсутствуют.

double Pow(double a, double degree);

Назначение: возведение в степень.

Входные параметры: a – число, которое нужно возвести в степень degree, degree –значение степени, в которую нужно возвести.

Выходные параметры: a в степени degree.

double GetCoeff();

Назначение: возвращает значение коэффициента.

Входные параметры: отсутствуют.

Выходные параметры: значение коэффициента.

int GetDegree();

Назначение: возвращает значение степени.

Входные параметры: отсутствуют.

Выходные параметры: значение степени.

double calculateMonom(std::map<char, double>& variables);

Назначение: вычисление монома, используя данные из map.

Входные параметры: variables – словарь, хранящий уникальные переменные и их значения.

Выходные параметры: значение монома.

bool operator<(const TMonom& m) const;

Назначение: перегруженный оператор "меньше". Сравнивает два объекта **TMonom** по убыванию степени.

Входные параметры: m – ссылка на объект **TMonom** для сравнения.

Выходные параметры: true, если текущий объект меньше, иначе false.

bool operator>(const TMonom& m) const;

Назначение: перегруженный оператор "больше". Сравнивает два объекта **TMonom** по убыванию степени.

Входные параметры: m – ссылка на объект **TMonom** для сравнения.

Выходные параметры: true, если текущий объект больше, иначе false.

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

Назначение: операция равенства. Проверяет равенство коэффициента и степени двух объектов **TMonom**.

Входные параметры: m – ссылка на объект **TMonom** для сравнения.

Выходные параметры: true, если объекты равны, иначе false.

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

Назначение: операция неравенства. Проверяет неравенство коэффициента и степени двух объектов **TMonom**.

Входные параметры: m – ссылка на объект **TMonom** для сравнения.

Выходные параметры: true, если объекты не равны, иначе false.

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

Назначение: умножение мономов.

Входные параметры: m – моном, на который будем умножать.

Выходные параметры: произведение мономов.

TMonom operator+(const TMonom& m)const;

Назначение: сложение мономов.

Входные параметры: m – моном, с которым будем складывать.

Выходные параметры: сумма мономов.

TMonom operator-(const TMonom& m)const;

Назначение: разность мономов.

Входные параметры: m – моном, который будем вычитать.

Выходные параметры: разность мономов.

TMonom monom\_defX();

Назначение: дифференцирование монома по переменной x.

Входные параметры: отсутствуют.

Выходные параметры: дифференциал монома по x.

TMonom monom\_defY();

Назначение: дифференцирование монома по переменной y.

Входные параметры: отсутствуют.

Выходные параметры: дифференциал монома по y.

TMonom monom\_defZ();

Назначение: дифференцирование монома по переменной z.

Входные параметры: отсутствуют.

Выходные параметры: дифференциал монома по z.

friend std::ostream& operator<<(std::ostream& out, TMonom& m);

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, m – ссылка на объект типа TMonom который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

friend std::istream& operator>>(std::istream& istr, TMonom& m);

Назначение: оператор ввода.

Входные параметры: istr – ссылка на объект типа istream, который представляет

входной поток, m – ссылка на объект типа TMonom, который будет заполняться значениями из входного потока.

Выходные параметры: ссылка на объект типа istream.

### Описание класса TPolynom

class TPolinom {

private:

TRingList<TMonom> monoms;

std::string formula;

std::map<char, double> uniques;

void sumEqual();

void deleteZero();

void Sort();

bool isMonomOperator(char op) const;

bool isSeparator(char op) const;

bool isOperator(char op) const;

bool isVariable(char var) const;

void fillUniques(const std::string& f);

void getListOfMonoms(const std::string& input);

double getCoeff(const std::string& input, int& pos);

void updateDegree(int& degree, char variable, int degreeOfVar);

void fillVariables();

double getValue(const char& variable);

public:

TPolinom();

TPolinom(const string& stringPolynom);

TPolinom(const TRingList<TMonom>& m);

TPolinom(const TPolinom& p);

~TPolinom();

std::string getStringMonoms() const;

std::string getFormula() const;

TPolinom operator+(const TPolinom& p);

TPolinom operator-(const TPolinom& p);

TPolinom operator\*(const TPolinom& p);

TPolinom operator\*(const double c);

const TPolinom& operator=(const TPolinom& p);

void GetNormalView();

TPolinom defX() const; //производная по x

TPolinom defY() const; //производная по y

TPolinom defZ() const; //производная по z

double calculate();

friend std::ostream& operator<<(std::ostream& out,TPolinom& p);

};

Назначение: работа с полиномами

Поля:

monoms – список мономов.

formula – формула, строковое представление полинома.

unique – словарь, хранящий уникальные переменные и их значения.

Методы:

void sumEqual();

Назначение: суммирование одинаковых по степеням мономов.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void deleteZero();

Назначение: удаление мономов с нулевым коэффициентом.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

void Sort();

Назначение: сортировка полинома по убыванию степени мономов.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

bool isMonomOperator(char op) const;

Назначение: проверка на умножение или возведение в степень.

Входные параметры: op – символ из строки формулы полинома.

Выходные параметры: true – если встретили \* или ^, false - иначе.

bool isSeparator(char op) const;

Назначение: проверка на знак плюс или минус, которые являются разделителями мономов.

Входные параметры: op – символ из строки формулы полинома.

Выходные параметры: true – если встретили + или -, false - иначе.

bool isOperator(char op) const;

Назначение: проверка на знак плюс, минус, умножить, возвести в степень.

Входные параметры: op – символ из строки формулы полинома.

Выходные параметры: true – если встретили +, -, \*, ^, false - иначе.

bool isVariable(char var) const;

Назначение: проверка на переменную.

Входные параметры: var – символ из строки формулы полинома.

Выходные параметры: true – если встретили x, y или z, false - иначе.

void fillUniques(const std::string& f);

Назначение: подсчет уникальных переменных в полиноме.

Входные параметры: f – формула полинома.

Выходные параметры: отсутствуют.

void getListOfMonoms(const std::string& input);

Назначение: представление мономов в виде кольцевого списка.

Входные параметры: input – формула полинома.

Выходные параметры: отсутствуют.

double getCoeff(const std::string& input, int& pos);

Назначение: возвращает значение коэффициента.

Входные параметры: input – формула полинома, pos – позиция коэффициента в строке.

Выходные параметры: значение коэффициента.

void updateDegree(int& degree, char variable, int degreeOfVar);

Назначение: изменяет значение степени.

Входные параметры: degree – ссылка на переменную, содержащую текущее значение степени полинома, variable – символ, указывающий на переменную, degreeOfVar – новое значение степени.

Выходные параметры: значение коэффициента.

void fillVariables();

Назначение: заполнение значений уникальных переменных x,y,z.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

double getValue(const char& variable);

Назначение: запрос у пользователя значений уникальных переменных x,y,z.

Входные параметры: variable - переменная, значение которой нужно присвоить.

Выходные параметры: значение переменой.

TPolinom();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

TPolinom(const string& stringPolynom);

Назначение: конструктор с параметрами.

Входные параметры: stringPolynom – строка, на основе которого создаем новый полином.

Выходные параметры: отсутствуют.

TPolinom(const TRingList<TMonom>& m);

Назначение: конструктор с параметрами.

Входные параметры: m – кольцевой список мономов, на основе которого создаем новый полином.

Выходные параметры: отсутствуют.

TPolinom(const TPolinom& p);

Назначение: конструктор копирования.

Входные параметры: p – полином, на основе которого создаем новый полином.

Выходные параметры: отсутствуют

~TPolinom();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

std::string getStringMonoms() const;

Назначение: получение строкового представления кольцевого списка мономов.

Входные параметры: отсутствуют.

Выходные параметры: строка мономов.

TPolinom operator+(const TPolinom& p);

Назначение: сложение полиномов.

Входные параметры: p – полином, с которым будем складывать.

Выходные параметры: сумма полиномов.

TPolinom operator-(const TPolinom& p);

Назначение: разность полиномов.

Входные параметры: p – полином, который будем вычитать.

Выходные параметры: разность полиномов.

TPolinom operator\*(const TPolinom& p);

Назначение: умножение полиномов.

Входные параметры: p – полином, на который будем умножать.

Выходные параметры: произведение полиномов.

TPolinom operator\*(const double c);

Назначение: умножение полинома на число.

Входные параметры: c – число, на которое будем умножать полином.

Выходные параметры: произведение полинома и числа.

const TPolinom& operator=(const TPolinom& p);

Назначение: операция присваивания.

Входные параметры: p – полином, на основе которого создаем новый полином.

Выходные параметры: ссылка на присвоенный полином.

void GetNormalView();

Назначение: выполнение операций сортировки, удаления нулевых мономов, сложения одинаковых мономов и получение строкового отображения мономов .

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

std::string getFormula() const;

Назначение: получение формулы полинома.

Входные параметры: отсутствуют.

Выходные параметры: строковое представление полинома.

TPolinom defX() const;

Назначение: дифференцирование полинома по х.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по х.

TPolinom defY() const;

Назначение: дифференцирование полинома по y.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по y.

TPolinom defZ() const;

Назначение: дифференцирование полинома по z.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по z.

double calculate();

Назначение: вычисление значения полинома.

Входные параметры отсутствуют.

Выходные параметры: результат вычисления.

friend std::ostream& operator<<(std::ostream& out, TPolinom& p)

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, p – ссылка на объект типа TPolinom который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

# Заключение

В рамках данной лабораторной работы была разработана и реализована структура данных для работы с полиномами. Были созданы классы TMonom и TPolinom, предоставляющие функционал для работы с мономами и полиномами соответственно.

Класс TMonom содержит информацию о коэффициенте и степени монома, а также определены операторы сравнения для мономов по степени.

Класс TPolynom осуществляет работу с полиномами через кольцевой список мономов. Реализованы операторы сложения, вычитания и умножения полиномов, а также оператор присваивания. Класс также предоставляет функционал для вычисления значения полинома для заданных значений переменных, а также позволяет дифференцировать полиномы по трем переменным по отдельности.

Таким образом, результатом выполнения лабораторной работы стала реализация структуры данных для работы с полиномами, позволяющей удобно и эффективно выполнять различные операции над ними, а также проводить анализ и вычисления, необходимые в контексте математических вычислений.

# Литература

1. Связный список [https://ru.wikipedia.org/wiki/Связный\_список].
2. Полином [https://ru.wikipedia.org/wiki/Многочлен]
3. Линейные списки: эффективное и удобное хранение данных [https://nauchniestati.ru/spravka/hranenie-dannyh-s-ispolzovaniem-linejnyh-spiskov/?ysclid=ltwvp5uwda145425180]

# Приложения

## Приложение А. Реализация класса TNode

#ifndef \_\_TNODE\_H\_\_

#define \_\_TNODE\_H\_\_

#include <iostream>

using namespace std;

template<class TData>

class TNode {

public:

TData data;

TNode\* pNext;

TNode();

TNode(const TData& data\_, const TNode\* pNext\_ = nullptr);

TNode(const TNode& Node);

};

template<class TData>

TNode<TData>::TNode() {

data = NULL;

pNext = nullptr;

}

template<class TData>

TNode<TData>::TNode(const TData& data\_, const TNode\* pNext\_) {

data = data\_;

pNext = (TNode\*)pNext\_;

}

template<class TData>

TNode<TData>::TNode(const TNode& Node) {

data = Node.data;

pNext = Node.pNext;

}

## Приложение Б. Реализация класса TList

#ifndef \_\_TLIST\_H\_\_

#define \_\_TLIST\_H\_\_

#include <iostream>

#include "tnode.h"

using namespace std;

template<class TData>

class TList {

protected:

TNode<TData>\* pFirst = nullptr;

TNode<TData>\* pLast = nullptr;

TNode<TData>\* pStop = nullptr;

TNode<TData>\* pCurr = nullptr;

TNode<TData>\* pPrev = nullptr;

public:

TList();

TList(const TNode<TData>\* pFirst\_);

TList(const TList& pList);

virtual void Clear();

~TList();

bool is\_Full() const;

void Next();

void Sort();

virtual bool is\_Empty() const;

TNode<TData>\* Search(const TData& data\_);

//INSERT

virtual void InsertFirst(const TData& data\_);

void InsertLast(const TData& data\_);

void InsertBefore(const TData& data\_, const TData& NextData);

void InsertAfter(const TData& data\_, const TData& PrevData);

void Reset();

TNode<TData>\* GetCurrent() const;

TData getCurrData() const;

void setCurrData(const TData& \_data);

const TList<TData>& operator=(const TList<TData>& pList);

void InsertBeforeCurrent(const TData& data\_); //вставить перед текущим

void InsertAfterCurrent(const TData& data\_); //вставить после текущего

void Remove(const TData& data\_); //удаление по ключу

void RemoveFirst();

void RemoveCurrent();

bool is\_End() const; //список завершен?

friend std::ostream& operator<<(std::ostream& out, const TList<TData>& list) {

TNode<TData>\* tmp = list.pFirst;

int num = 1;

while (tmp != nullptr) {

out << num << " node " << tmp->data << std::endl;

tmp = tmp->pNext;

num++;

}

return out;

}

};

template <class TData> TList<TData> ::TList() {

pFirst = pLast = pCurr = pStop = pPrev = nullptr;

Reset();

}

template <class TData> TList<TData> ::TList(const TNode<TData>\* pFirst\_) {

pFirst = (TNode<TData>\*)pFirst\_;

pCurr = pFirst;

while (pCurr->pNext != nullptr)

{

Next();

}

pLast = pCurr;

pCurr = pFirst;

pLast->pNext = pStop;

pStop = pPrev = nullptr;

}

template <class TData> TList<TData> ::TList(const TList& pList) {

if (pList.pFirst == nullptr)

{

return;

}

TNode<TData>\* oldCurrent = pList.pFirst;

pFirst = new TNode<TData>(\*pList.pFirst);

TNode<TData>\* newCurrent = pFirst;

oldCurrent = oldCurrent->pNext;

while (oldCurrent != pList.pStop)

{

newCurrent->pNext = new TNode<TData>(oldCurrent->data);

newCurrent = newCurrent->pNext;

oldCurrent = oldCurrent->pNext;

}

pLast = newCurrent;

pLast->pNext = pStop;

pCurr = pFirst;

pStop = nullptr;

}

template <class TData> const TList<TData>& TList<TData>::operator=(const TList<TData>& pList) {

if (this == &pList)

{

return \*this;

}

if (pList.pFirst == nullptr)

{

pFirst = pStop = pCurr = pPrev = pLast = nullptr;

return;

}

TNode<TData>\* oldCurrent = pList.pFirst;

pFirst = new TNode<TData>(\*pList.pFirst);

TNode<TData>\* newCurrent = pFirst;

oldCurrent = oldCurrent->pNext;

while (oldCurrent != pList.pStop)

{

newCurrent->pNext = new TNode<TData>(oldCurrent->data);

newCurrent = newCurrent->pNext;

oldCurrent = oldCurrent->pNext;

}

pLast = newCurrent;

pCurr = pFirst;

pStop = nullptr;

return \*this;

}

template <class TData> void TList<TData>::Clear() {

if (pFirst == nullptr)

{

return;

}

pCurr = pFirst;

while (pCurr != pStop) {

TNode<TData>\* tmp = pCurr;

pCurr = pCurr->pNext;

delete tmp;

}

pCurr = pStop;

pLast = pStop;

pFirst = pStop;

pPrev = pStop;

}

template <class TData> TList<TData> ::~TList() {

Clear();

}

//sort по убыванию

template <class TData> void TList<TData>::Sort() {

TNode<TData>\* elem1 = pFirst;

while (elem1->pNext != pStop)

{

TNode<TData>\* elem2 = elem1->pNext;

while (elem2 != pStop)

{

if (elem1->data < elem2->data)

{

TData tmp = elem1->data;

elem1->data = elem2->data;

elem2->data = tmp;

}

elem2 = elem2->pNext;

}

elem1 = elem1->pNext;

}

pPrev = pLast;

pCurr = pFirst;

}

template <class TData> void TList<TData> ::Next() {

if (pCurr == pStop)

{

return;

}

pPrev = pCurr;

pCurr = pCurr->pNext;

}

template <class TData> void TList<TData> ::Reset() {

if (is\_Empty()) {

pCurr = pStop;

}

else {

pCurr = pFirst;

pPrev = pStop;

}

return;

}

template <class TData> bool TList<TData> ::is\_Full() const {

try

{

TNode<TData>\* tmp = new TNode<TData>();

if (tmp == nullptr)

return true;

else

{

delete tmp;

return false;

}

}

catch (...)

{

return true;

}

return true;

}

template <class TData> bool TList<TData> ::is\_Empty() const {

return pFirst == pStop;

}

template <class TData> bool TList<TData> ::is\_End() const {

if (pCurr == pStop)

return true;

else

return false;

}

template <class TData> TNode<TData>\* TList<TData> ::Search(const TData& data\_) {

TNode<TData>\* tmp = pFirst;

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

tmp = tmp->pNext;

}

if (tmp == pStop) {

return nullptr;

}

return tmp;

}

template <class TData> void TList<TData>::Remove(const TData& data\_) {

// Поиск звена с данными для удаления

pCurr = Search(data\_);

if (pCurr == nullptr) {

throw "Error. List does't have node with this data";

}

if (pCurr == pFirst) {

pFirst = pFirst->pNext;

}

else {

pPrev->pNext = pCurr->pNext;

}

if (pCurr == pLast) {

pLast = pPrev;

}

delete pCurr;

}

template <class TData> void TList<TData>::RemoveFirst()

{

if (pFirst != pStop) {

TNode<TData>\* tmp = pFirst;

pFirst = pFirst->pNext;

pCurr = pFirst;

delete tmp;

}

}

template <class TData> void TList<TData>::RemoveCurrent() {

if(pCurr != pStop)

if (pCurr == pFirst) {

RemoveFirst();

}

else {

TNode<TData>\* tmp = pCurr;

pPrev->pNext = pCurr->pNext;

pCurr = pCurr->pNext;

if (pPrev->pNext == pStop)

pLast = pPrev;

pPrev->pNext = pCurr;

delete tmp;

}

}

template <class TData> void TList<TData>::InsertFirst(const TData& data\_) {

if (is\_Full()) {

throw "Error. Memory is not enough";

}

TNode<TData>\* newNode = new TNode<TData>(data\_, nullptr);

if (pFirst != pStop)

{

TNode<TData>\* oldPFirst = pFirst;

pFirst = newNode;

pFirst->pNext = oldPFirst;

}

else {

pFirst = newNode;

}

pCurr = pFirst;

if (pLast == pStop) {

pLast = pFirst;

}

}

template <class TData> void TList<TData>::InsertLast(const TData& data\_) {

if (is\_Empty())

{

InsertFirst(data\_);

pLast = pFirst;

return;

}

TNode<TData>\* newNode = new TNode<TData>(data\_, pStop);

pLast->pNext = newNode;

pLast = newNode;

pCurr = pLast;

}

template <class TData> void TList<TData>::InsertBefore(const TData& data\_, const TData& NextData) {

if (is\_Empty()) {

throw "Error. List is empty.";

}

pCurr = pFirst;

while ((pCurr != pStop) && (pCurr->data != NextData))

{

pPrev = pCurr;

Next();

}

if ((pCurr == pStop) && (pCurr->data != NextData))

{

throw "Error. Key not found.";

}

TNode<TData>\* newNode = new TNode<TData>(data\_, pCurr);

// list have only 1 node

if (pPrev == pStop) {

pFirst = newNode;

}

else {

newNode->pNext = pPrev->pNext;

pPrev->pNext = newNode;

}

pCurr = pFirst;

}

template <class TData> void TList<TData>::InsertAfter(const TData& data\_, const TData& PrevData) {

if (is\_Empty())

throw "Error. List is empty";

TNode<TData>\* prNode = Search(PrevData);

if (prNode == pStop)

{

throw "Error. Can't insert after";

}

TNode<TData>\* tmp = new TNode<TData>(data\_, prNode->pNext);

tmp->pNext = prNode->pNext;

prNode->pNext = tmp;

}

// Set & Get

template <class TData> TNode<TData>\* TList<TData>::GetCurrent() const //возвращает текущее звено

{

return pCurr;

}

template <class TData> TData TList<TData>::getCurrData() const {

if (pCurr != pStop)

return pCurr->data;

else

return NULL;

}

template <class TData> void TList<TData>::setCurrData(const TData& \_data)

{

pCurr->data = \_data;

}

//pCur на вставленный элемент

template <class TData> void TList<TData>::InsertBeforeCurrent(const TData& data\_) //вставить перед текущим

{

if (is\_Empty()) {

InsertFirst(data\_);

}

else if (is\_End()) {

InsertLast(data\_);

}

else {

TNode<TData>\* newNode = new TNode<TData>(data\_, pCurr); //pCur->pNext;

if (newNode == nullptr) {

throw "Error. List doesn't have memory";

}

else {

pCurr = newNode;

pPrev->pNext = newNode;

}

}

}

//pCur на вставленный элемент

template <class TData> void TList<TData>::InsertAfterCurrent(const TData& data\_) //вставить после текущего

{

if (is\_Empty()) {

InsertFirst(data\_);

}

else if (is\_End()) {

InsertLast(data\_);

}

else {

TNode<TData>\* newNode = new TNode<TData>(data\_);

if (newNode == nullptr) {

throw "Error. List doesn't have memory";

}

else {

newNode->pNext = pCurr->pNext;

pCurr->pNext = newNode;

pCurr = newNode;

pPrev = pPrev->pNext;

}

}

}

#endif

## Приложение В. Реализация класса TRingList

#ifndef \_\_TRINGLIST\_H\_\_

#define \_\_TRINGLIST\_H\_\_

#include <iostream>

#include "tlist.h"

using namespace std;

template <class TData>

class TRingList : public TList<TData> {

protected:

TNode<TData>\* pHead = nullptr;

public:

TRingList();

TRingList(const TNode<TData>\* pFirst);

TRingList(const TRingList<TData>& ringList);

void InsertFirst(const TData& data\_) override; //вставить после pHead

void DeleteFirst();

void Clear() override;

bool is\_Empty() const override;

const TRingList<TData>& operator=(const TRingList<TData>& ringList); // TODO: const

friend std::ostream& operator<<(std::ostream& out, TRingList<TData>& ringList) {

TNode<TData>\* tmp = ringList.pFirst;

int num = 1;

while (tmp != ringList.pHead) {

out << num << " node " << tmp->data << std::endl;

tmp = tmp->pNext;

num++;

}

return out;

}

};

template <class TData> void TRingList<TData>::Clear() {

if (pFirst == nullptr)

{

return;

}

pCurr = pFirst;

while (pCurr != pHead) {

TNode<TData>\* tmp = pCurr;

pCurr = pCurr->pNext;

delete tmp;

}

pCurr = pStop;

pLast = pStop;

pFirst = pStop;

pPrev = pStop;

pHead->pNext = pHead;

}

template <class TData> bool TRingList<TData>::is\_Empty() const {

return (pHead->pNext == pHead);

}

template <class TData> TRingList<TData>::TRingList() {

this->pHead = new TNode<TData>();

this->pHead->pNext = pHead;

this->pCurr = this->pPrev = this->pLast = this->pFirst = nullptr;

this->pStop = this->pHead;

}

template <class TData> TRingList<TData>::TRingList(const TNode<TData>\* \_pFirst) {

pHead = new TNode<TData>();

pHead->pNext = pHead;

if (\_pFirst != nullptr) {

pFirst = (TNode<TData>\*)\_pFirst;

pHead->pNext = pFirst;

pCurr = pFirst;

pStop= pHead;

TNode<TData>\* tmp = pCurr;

while (tmp->pNext != nullptr) {

tmp = tmp->pNext;

}

tmp->pNext = pHead;

pLast = tmp;

}

}

template <class TData> TRingList<TData>::TRingList(const TRingList<TData>& ringList) {

pHead = new TNode<TData>();

pFirst = new TNode<TData>(\*ringList.pFirst);

pHead->pNext = pFirst;

// Set new pStop

pStop = pHead;

pCurr = pFirst;

TNode<TData>\* tmp(ringList.GetCurrent());

while (tmp->pNext != ringList.pStop) {

pCurr->pNext = new TNode<TData>(\*tmp->pNext);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pCurr->pNext = pStop;

pLast = pCurr;

TList<TData>::Reset();

}

template <class TData> const TRingList<TData>& TRingList<TData>::operator=(const TRingList<TData>& ringList) {

if (this == &ringList)

{

return \*this;

}

pHead = new TNode<TData>();

pFirst = new TNode<TData>(\*ringList.pFirst);

pHead->pNext = pFirst;

// Set new pStop

pStop = pHead;

pCurr = pFirst;

TNode<TData>\* tmp = ringList.GetCurrent();

while (tmp->pNext != ringList.pStop) {

pCurr->pNext = new TNode<TData>(\*tmp->pNext);

pCurr = pCurr->pNext;

tmp = tmp->pNext;

}

pCurr->pNext = pStop;

pLast = pCurr;

TList<TData>::Reset();

return \*this;

}

template <class TData> void TRingList<TData>::InsertFirst(const TData& data\_) {

if (is\_Full()) {

throw "Error. Memory is not enough";

}

TNode<TData>\* newNode = new TNode<TData>(data\_, nullptr);

if (pFirst != nullptr)

{

TNode<TData>\* oldPFirst = pFirst;

pFirst = newNode;

pFirst->pNext = oldPFirst;

}

else {

pFirst = newNode;

pFirst->pNext = pStop;

pLast = pFirst;

}

pCurr = pFirst;

pHead->pNext = pFirst;

}

template <class TData> void TRingList<TData>::DeleteFirst() {

TList<TData>::RemoveFirst();

pHead->pNext = this->pFirst;

}

**#endif**

## Приложение Г. Реализация класса TMonom

#include "tmonom.h"

#include <iostream>

#include <string>

#include <sstream>

#include <iomanip>

TMonom::TMonom(const TMonom& m) {

Coeff = m.Coeff;

Degree = m.Degree;

}

TMonom::TMonom( double c, int d) {

if ((d > 999) || (d < 0))

throw "Wrong degree";

Coeff = c;

Degree = d;

}

TMonom& TMonom::operator=(const TMonom& m) {

Coeff = m.Coeff;

Degree = m.Degree;

return \*this;

}

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

if (Degree < m.Degree)

return true;

if (Degree == m.Degree)

{

if (Coeff < m.Coeff)

return true;

else

return false;

}

else

return false;

}

bool TMonom::operator>(const TMonom& m) const {

if (Degree > m.Degree)

return true;

if (Degree == m.Degree)

{

if (Coeff > m.Coeff)

return true;

else

return false;

}

else

return false;

}

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

return Degree == m.Degree;

}

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

return !(\*this == m);

}

void TMonom::SetCoeff(double cm) {

Coeff = cm;

}

void TMonom::SetDegree(int d){

if ((d > 999) || (d < 0))

throw "Wrong degree";

Degree = d;

}

double TMonom::GetCoeff() {

return Coeff;

}

int TMonom::GetDegree() {

return Degree;

}

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

if ((Degree + m.Degree) <= 999 && (Degree + m.Degree) >= 0) {

return TMonom(Coeff \* m.Coeff, Degree + m.Degree);

}

else {

throw "Degree limit!";

}

}

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

TMonom result;

if ((\*this) == m) {

result.SetCoeff((\*this).Coeff + m.Coeff);

result.Degree = m.Degree;

}

else {

throw "Error. Can't sum monoms";

}

return result;

}

TMonom TMonom::operator-(const TMonom& m)const {

TMonom result;

if ((\*this) == m) {

result.SetCoeff((\*this).Coeff - m.Coeff);

result.Degree = m.Degree;

}

else {

throw "Error. Can't sum monoms";

}

return result;

}

TMonom TMonom::monom\_defX() {

if (Degree - 100 == 0)

{

return TMonom(Coeff, 0);

}

Coeff = Coeff \* (Degree / 100);

Degree = Degree - 100;

return TMonom(Coeff, Degree);

}

TMonom TMonom::monom\_defY() {

int y = (Degree % 100) / 10;

if (y == 0) {

return TMonom(0, 0);

}

if (y == 1)

{

return TMonom(Coeff, Degree - 10);

}

Coeff = Coeff \* y;

Degree = Degree - 10;

return TMonom(Coeff, Degree);

}

TMonom TMonom::monom\_defZ() {

int z = Degree % 10;

if (z == 1)

{

return TMonom(Coeff, Degree - 1);

}

if (z == 0)

{

return TMonom(0, 0);

}

Coeff = Coeff \* z;

Degree = Degree - 1;

return TMonom(Coeff, Degree);

}

std::ostream& operator<<(std::ostream& out, TMonom& m) {

int x, y, z;

x = m.Degree / 100;

y = (m.Degree % 100) / 10;

z = m.Degree % 10;

if (m.Coeff != 1 || m.Degree == 0)

out << m.Coeff;

if (x > 0)

out << "x^" << x;

if (y > 0)

out << "y^" << y;

if (z > 0)

out << "z^" << z;

return out;

}

std::istream& operator>>(std::istream& istr, TMonom& m) {

int x, y, z;

std::cout << "Enter coeff\n";

istr >> m.Coeff;

std::cout << "Enter degries x, y, z \n";

istr >> x >> y >> z;

m.Degree = x \* 100 + y \* 10 + z;

return istr;

}

double TMonom::Pow(double a, double degree) {

double result = 1.0;

for (int i = 0; i < degree; i++)

{

result \*= a;

}

return result;

}

std::string TMonom::getStringMonom() const {

//форматирование цифр после запятой коэффициента

std::ostringstream oss;

oss << std::fixed << std::setprecision(1) << Coeff; //выводит без запятой либо сделать 1-2 знака после запятой и + "\*"

std::string stringMonom;

if (Coeff >= 0) {

if (Coeff == 1.0) {

stringMonom += "+";

}

else {

stringMonom = "+" + oss.str();

}

}

else {

if (Coeff == -1.0) {

stringMonom += "-";

}

else {

stringMonom = oss.str();

}

}

if (Degree > 0 && Coeff != 1.0 && Coeff != -1.0) {

stringMonom += "\*";

}

int x, y, z;

x = Degree / 100;

y = (Degree % 100) / 10;

z = Degree % 10;

if (x > 1)

stringMonom += "x^" + std::to\_string(x);

else if (x == 1) {

stringMonom += "x";

}

if (y > 1 )

stringMonom += "y^" + std::to\_string(y);

else if (y == 1) {

stringMonom += "y";

}

if (z > 1)

stringMonom += "z^" + std::to\_string(z);

else if (z == 1) {

stringMonom += "z";

}

return stringMonom;

}

double TMonom::calculateMonom(std::map<char, double>& variables) {

double result = Coeff;

int st = 100;

if (Degree / 100 != 0)

{

result \*= Pow(variables['x'], Degree / 100);

}

if ((Degree % 100) / 10 != 0)

{

result \*= Pow(variables['y'], (Degree % 100) / 10);

}

if (Degree % 10 != 0) {

result \*= Pow(variables['z'], Degree % 10);

}

return result;

}

## Приложение Д. Реализация класса TPolinom

#include "tpolynom.h"

TPolinom::TPolinom() {

}

TPolinom::TPolinom(const string& stringPolynom) {

getListOfMonoms(stringPolynom);

monoms.Sort();

deleteZero();

monoms.Reset();

formula = getStringMonoms();

}

TPolinom::TPolinom(const TRingList<TMonom>& m) {

monoms = m;

monoms.Sort();

deleteZero();

monoms.Reset();

formula = getStringMonoms();

fillUniques(formula);

}

TPolinom::TPolinom(const TPolinom& p) {

formula = p.formula;

uniques = p.uniques;

TRingList <TMonom> tmpPolynom = p.monoms;

if (!p.monoms.is\_Empty()) {

while (!tmpPolynom.is\_End()) {

monoms.InsertLast(tmpPolynom.getCurrData());

tmpPolynom.Next();

}

}

monoms.Reset();

}

double TPolinom::getCoeff(const std::string& input, int& pos)

{

std::string coeff = "";

while (pos < input.size() && !isMonomOperator(input[pos]) && !isVariable(input[pos]))

{

coeff += input[pos];

pos++;

}

if (coeff == "+" || coeff == "") {

coeff = "1.0";

}

if (coeff == "-") {

coeff = "-1.0";

}

return std::stod(coeff);

}

void TPolinom::updateDegree(int& degree, char variable, int degreeOfVar)

{

switch (variable) {

case 'x':

degree += 100 \* degreeOfVar;

break;

case 'y':

degree += 10 \* degreeOfVar;

break;

case 'z':

degree += degreeOfVar;

break;

default:

break;

}

}

void TPolinom::getListOfMonoms(const std::string& input) {

double currentCoef = 1.0;

int currentDegree = 0;

int index = 0;

while (index < input.size()) {

currentCoef = getCoeff(input, index);

while (!isSeparator(input[index]) && index < input.size()) {

if (isMonomOperator(input[index])) {

if (input[index] == '\*') {

index++;

}

else {

updateDegree(currentDegree, input[index - 1], input[index + 1] - '0');

index += 2;

}

}

else if (isVariable(input[index])) {

if (index == input.size() - 1 || input[index + 1] != '^') {

updateDegree(currentDegree, input[index], 1);

}

uniques[input[index]] = INFINITY;

index++;

}

else {

throw "Uncorrect polynom";

}

}

TMonom monom(currentCoef, currentDegree);

monoms.InsertLast(monom);

currentDegree = 0;

}

}

void TPolinom::fillUniques(const std::string& f)

{

int index = 0;

while (index < f.size())

{

if (isVariable(f[index]))

{

uniques[f[index]]++;

uniques[f[index]] = INFINITY;

}

index++;

}

}

void TPolinom::fillVariables() {

for (auto& uniques : uniques)

{

if (uniques.second == INFINITY) {

uniques.second = getValue(uniques.first);

}

}

}

double TPolinom::getValue(const char& variable) {

std::cout << "Enter variable value for \"" << variable << "\"" << std::endl;

double value = INFINITY;

std::cin >> value;

return value;

}

std::string TPolinom::getFormula() const {

return formula;

}

double TPolinom::calculate()

{

fillVariables();

double result = 0.0;

while( !monoms.is\_Empty() && !monoms.is\_End())

{

result += monoms.GetCurrent()->data.calculateMonom(uniques);

monoms.Next();

}

return result;

}

bool TPolinom::isVariable(char var) const

{

return (var == 'x' || var == 'y' || var == 'z');

}

bool TPolinom::isMonomOperator(char op) const {

return (op == '\*' || op == '^');

}

bool TPolinom::isSeparator(char op) const {

return (op == '+' || op == '-');

}

bool TPolinom::isOperator(char op) const {

return (op == '+' || op == '-' || op == '\*' || op == '^');

}

TPolinom::~TPolinom()

{

formula = "";

}

void TPolinom::Sort() {

monoms.Sort();

deleteZero();

formula = getStringMonoms();

fillUniques(formula);

}

void TPolinom::sumEqual() {

monoms.Sort();

while (!monoms.is\_Empty() && !monoms.is\_End())

{

TNode<TMonom>\* i = monoms.GetCurrent();

TNode<TMonom>\* j = i->pNext;

if (j != monoms.GetPStop() && i->data.GetDegree() == j->data.GetDegree())

{

double coef1 = i->data.GetCoeff();

double coef2 = j->data.GetCoeff();

i->data.SetCoeff(coef1 + coef2);

i->pNext = j->pNext;

delete j;

}

else

{

monoms.Next();

}

}

deleteZero();

formula = getStringMonoms();

}

void TPolinom::deleteZero() {

monoms.Reset();

TNode<TMonom>\* current = monoms.GetCurrent();

while (!monoms.is\_Empty() && !monoms.is\_End())

{

if (monoms.GetCurrent()->data.GetCoeff() == 0.0)

{

monoms.RemoveCurrent();

}

else {

monoms.Next();

}

}

monoms.Reset();

}

std::string TPolinom::getStringMonoms() const {

std::string stringMonoms = "";

TRingList<TMonom> tmp(monoms);

while (!tmp.is\_Empty() && !tmp.is\_End())

{

stringMonoms += tmp.getCurrData().getStringMonom();

tmp.Next();

}

return stringMonoms;

}

TPolinom TPolinom::operator+(const TPolinom& p) {

TPolinom copy(p);

TPolinom resultPolynom;

monoms.Reset();

if (copy.monoms.is\_Empty() && monoms.is\_Empty())

{

return resultPolynom;

}

while (!copy.monoms.is\_End() && !monoms.is\_End())

{

if (copy.monoms.GetCurrent()->data.GetDegree() < monoms.GetCurrent()->data.GetDegree())

{

resultPolynom.monoms.InsertLast(monoms.GetCurrent()->data);

monoms.Next();

}

else if (copy.monoms.GetCurrent()->data.GetDegree() > monoms.GetCurrent()->data.GetDegree())

{

resultPolynom.monoms.InsertLast(copy.monoms.GetCurrent()->data);

copy.monoms.Next();

}

else

{

if (copy.monoms.GetCurrent()->data.GetDegree() == monoms.GetCurrent()->data.GetDegree())

{

if (copy.monoms.GetCurrent()->data.GetCoeff() + monoms.GetCurrent()->data.GetCoeff() != 0)

{

resultPolynom.monoms.InsertLast(copy.monoms.GetCurrent()->data + monoms.GetCurrent()->data);

}

}

copy.monoms.Next();

monoms.Next();

}

}

while (!copy.monoms.is\_End())

{

resultPolynom.monoms.InsertLast(copy.monoms.GetCurrent()->data);

copy.monoms.Next();

}

while (!monoms.is\_End())

{

resultPolynom.monoms.InsertLast(monoms.GetCurrent()->data);

monoms.Next();

}

resultPolynom.Sort();

resultPolynom.formula = resultPolynom.getStringMonoms();

return resultPolynom;

}

const TPolinom& TPolinom::operator=(const TPolinom& p) {

if (this == &p)

{

return \*this;

}

formula = p.formula;

uniques = p.uniques;

TPolinom copy(p);

copy.monoms.Reset();

monoms.Clear();

monoms.Reset();

while (!copy.monoms.is\_End())

{

monoms.InsertLast(copy.monoms.GetCurrent()->data);

copy.monoms.Next();

}

monoms.Reset();

return \*this;

}

TPolinom TPolinom::operator-(const TPolinom& p) {

TPolinom copy(p);

TPolinom result = (\*this) + copy \* (-1.0);

result.formula = result.getStringMonoms();

return result;

}

TPolinom TPolinom::operator\*(const double c) {

TPolinom copy(\*this);

TPolinom resultPolymom;

if (copy.monoms.is\_Empty())

{

return resultPolymom;

}

copy.monoms.Reset();

while (!copy.monoms.is\_End())

{

copy.monoms.GetCurrent()->data.SetCoeff(copy.monoms.GetCurrent()->data.GetCoeff() \* c);

if (copy.monoms.GetCurrent()->data.GetCoeff() != 0)

{

resultPolymom.monoms.InsertLast(copy.monoms.GetCurrent()->data);

}

copy.monoms.Next();

}

resultPolymom.monoms.Reset();

resultPolymom.formula = resultPolymom.getStringMonoms();

return resultPolymom;

}

TPolinom TPolinom::operator\*(const TPolinom& p) {

TPolinom multiplyP(p);

TRingList<TMonom> resultMonoms;

monoms.Reset();

multiplyP.monoms.Reset();

if (multiplyP.monoms.is\_Empty() && monoms.is\_Empty())

{

return TPolinom(resultMonoms);

}

while (!multiplyP.monoms.is\_End())

{

while (!monoms.is\_End())

{

if (multiplyP.monoms.GetCurrent()->data.GetCoeff() \* monoms.GetCurrent()->data.GetCoeff() != 0)

{

resultMonoms.InsertLast(multiplyP.monoms.getCurrData() \* monoms.getCurrData());

}

monoms.Next();

}

monoms.Reset();

multiplyP.monoms.Next();

}

monoms.Reset();

resultMonoms.Reset();

TPolinom resultPolynom(resultMonoms);

resultPolynom.Sort();

resultPolynom.sumEqual();

resultPolynom.formula = resultPolynom.getStringMonoms();

return resultPolynom;

}

TPolinom TPolinom::defX() const {

TRingList<TMonom> copy(monoms);

TRingList<TMonom> resultMonoms;

TRingList<TMonom> tmpMonoms;

if (copy.is\_Empty())

{

return TPolinom(resultMonoms);

}

while (!copy.is\_End())

{

if (copy.GetCurrent()->data.GetDegree() / 100 > 0)

{

tmpMonoms.InsertLast(copy.GetCurrent()->data.monom\_defX());

if (tmpMonoms.GetCurrent() != 0)

{

resultMonoms.InsertLast(tmpMonoms.GetCurrent()->data);

resultMonoms.Next();

}

tmpMonoms.Next();

}

copy.Next();

}

resultMonoms.Reset();

return TPolinom(resultMonoms);

}

TPolinom TPolinom::defY() const {

TRingList<TMonom> copy(monoms);

TRingList<TMonom> resultMonoms;

TRingList<TMonom> tmpMonoms;

if (copy.is\_Empty())

{

return TPolinom(resultMonoms);

}

while (!copy.is\_End())

{

if (copy.GetCurrent()->data.GetDegree() % 100 / 10 > 0)

{

tmpMonoms.InsertLast(copy.GetCurrent()->data.monom\_defY());

if (tmpMonoms.GetCurrent() != 0)

{

resultMonoms.InsertLast(tmpMonoms.GetCurrent()->data);

resultMonoms.Next();

}

tmpMonoms.Next();

}

copy.Next();

}

resultMonoms.Reset();

return TPolinom(resultMonoms);

}

TPolinom TPolinom::defZ() const {

TRingList<TMonom> copy(monoms);

TRingList<TMonom> resultMonoms;

TRingList<TMonom> tmpMonoms;

if (copy.is\_Empty())

{

return TPolinom(resultMonoms);

}

while (!copy.is\_End())

{

if (copy.GetCurrent()->data.GetDegree() % 10 > 0)

{

tmpMonoms.InsertLast(copy.GetCurrent()->data.monom\_defZ());

if (tmpMonoms.GetCurrent() != 0)

{

resultMonoms.InsertLast(tmpMonoms.GetCurrent()->data);

resultMonoms.Next();

}

tmpMonoms.Next();

}

copy.Next();

}

resultMonoms.Reset();

return TPolinom(resultMonoms);

}

std::ostream& operator<<(std::ostream& out, TPolinom& p) {

if (p.monoms.is\_Empty())

return out << "0" << std::endl;

p.getFormula();

out << p.getFormula();

return out;

}

## Приложение Е. Sample\_tpolinom

#include <iostream>

#include "tpolynom.h"

int main()

{

std::string sp1, sp2;

double c;

std::cout << "Let's create two polynoms. Please input your polynoms" << std::endl;

std::cin >> sp1;

std::cin >> sp2;

TPolinom polynom1(sp1);

TPolinom polynom2(sp2);

TPolinom res;

std::cout << "Your first polynom: " << polynom1 << std::endl;

std::cout << "Your second polynom: " << polynom2 << std::endl;

std::cout << "Let's add our polynoms: p1 + p2 = " << polynom1 + polynom2 << std::endl;

std::cout << "Let's sabstruct our polynoms: p1 - p2 = " << polynom1 - polynom2 << std::endl;

std::cout << "Let's multiply our polynoms: p1 \* p2 = " << polynom1 \* polynom2 << std::endl;

std::cout << "Let's multiply first polynom and constant. Please input constant" << std::endl;

std::cin >> c;

res = polynom1 \* c;

std::cout << "Let's multiply our polynoms: p1 \* const = " << res << std::endl;

std::cout << "Let's multiply second polynom and constant. Please input constant" << std::endl;

std::cin >> c;

res = polynom2 \* c;

std::cout << "Let's multiply our polynoms: p2 \* const = " << res << std::endl;

std::cout << "Derivatives with respect to x, y, z for the first polynom: " << std::endl;

std::cout << "p1 dif x = " << polynom1.defX() << std::endl;

std::cout << "p1 dif y = " << polynom1.defY() << std::endl;

std::cout << "p1 dif z = " << polynom1.defZ() << std::endl;

std::cout << "Derivatives with respect to x, y, z for the second polynom: " << std::endl;

std::cout << "p2 dif x = " << polynom2.defX() << std::endl;

std::cout << "p2 dif y = " << polynom2.defY() << std::endl;

std::cout << "p2 dif z = " << polynom2.defZ() << std::endl;

std::cout << "Let's calculate our first polynom: " << polynom1 << std::endl;

std::cout << "Answer = " << polynom1.calculate() << std::endl;

std::cout << "Let's calculate our second polynom: " << polynom2 << std::endl;

std::cout << "Answer = " << polynom2.calculate() << std::endl;

return 0;

}