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(а): студент(ка) группы 3822Б1ФИ2
_____ / Ясакова Т.Е./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП
_____ / Кустикова В.Д./
Подпись

Нижний Новгород
2024

Содержание

Введение

Лабораторная работа направлена на изучение обработки полиномов от трёх переменных (x, y, 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.

1 Постановка задачи

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 **`TPolynom`**, который будет использоваться класс **`TMonom`** и выполнять всю работу с полиномами. Предоставить пример использования и обеспечить работоспособность тестов, покрывающих все методы классов **`TMonom`** и **`TPolynom`**.

2 Руководство пользователя

2.1 Приложение для демонстрации работы звена

1. Запустите приложение с названием sample_node.exe. В результате появится окно, показанное ниже (рис. 1). Пользователю будет предложено ввести значение для звена, далее на экран выведутся **`data`** звена и его указатель **`pNext`**.

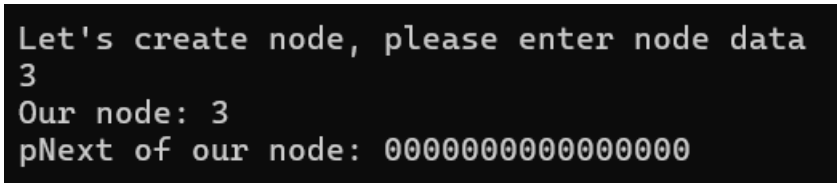

Рис. 1. Основное окно программы

2.2 Приложение для демонстрации работы списка

1. Запустите приложение с названием sample_list.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.

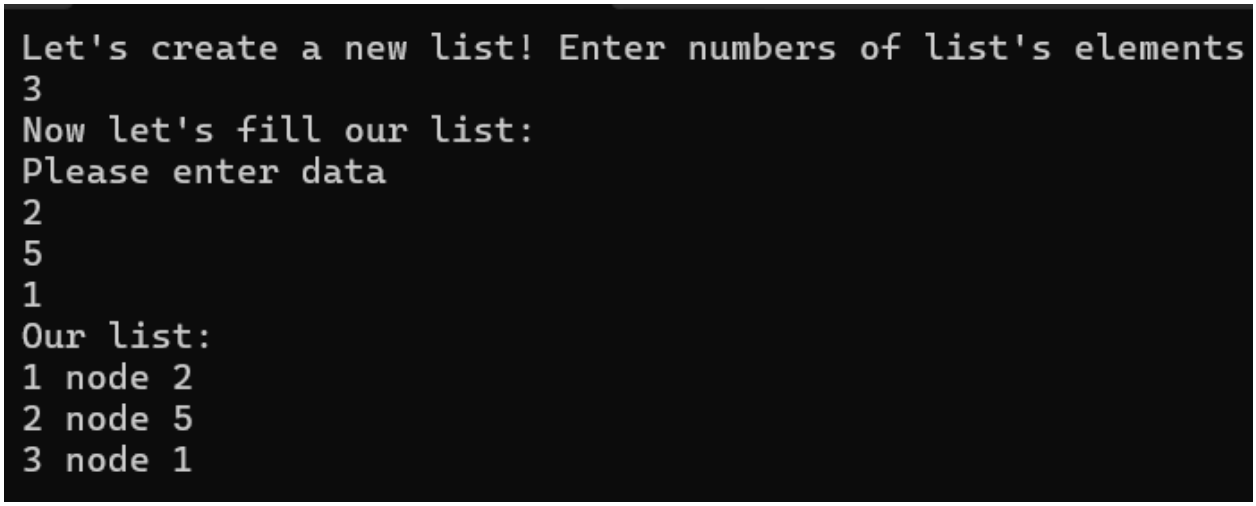

Рис. 2. Создание списка

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).

```
Let's insert something to the beginning of the list  and in the end
Please enter data to insert to the beginning
7
Please enter data to insert in the end
9
Our list:
1 node 7
2 node 2
3 node 5
4 node 1
5 node 9
```

Рис. 3. Вставка в начало и в конец списка

3. 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).

```
Let's insert something before element you want. At first, enter the element before which you want to insert
6
Error. List don't have this element. Please, enter correct element
1
Please enter data to insert
0
Our list:
1 node 7
2 node 2
3 node 5
4 node 0
5 node 1
6 node 9
```

Рис. 4. Вставка до указанного звена

4. 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).

```
Let's insert something after element you want. At first, enter the element after which you want to insert
3
Error. List don't have this element. Please, enter correct element
2
Please enter data to insert
4
Our list:
1 node 7
2 node 2
3 node 4
4 node 5
5 node 0
6 node 1
7 node 9
```

Рис. 5. Вставка после указанного звена

5. Ниже представлен результат функций удаления первого звена списка (рис. 6).

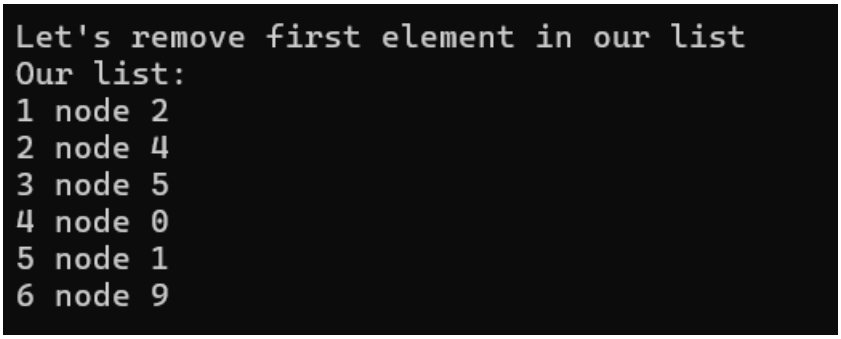

Рис. 6. Удаление первого звена списка

2.3 Приложение для демонстрации работы кольцевого списка

1. Запустите приложение с названием sample_headlist.exe. В результате появится окно, показанное ниже (рис. 7). 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.

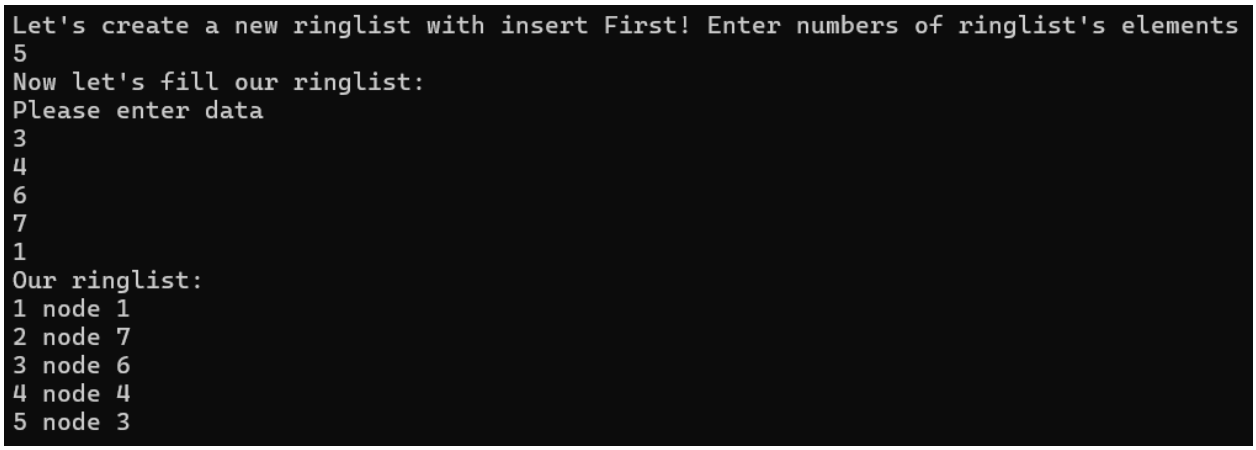

Рис. 7. Создание кольцевого списка

2. Далее представлена работа удаления «первого» звена кольцевого списка (рис. 8).

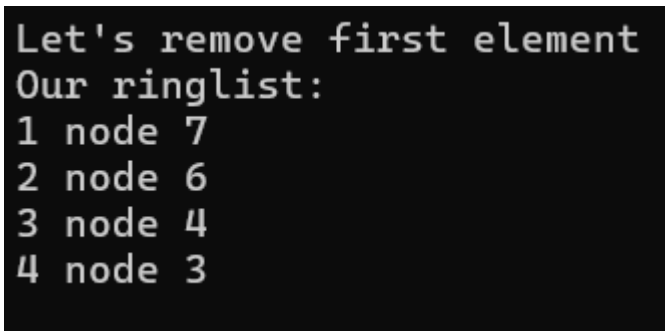

Рис. 8. Удаление первого звена в кольцевом списке

2.4 Приложения для демонстрации работы мономов

1. Запустите приложение с названием sample_monom.exe. В результате появится окно, показанное ниже (рис. 9). Пользователю будет предложено ввести коэффициенты и степени двух мономов.

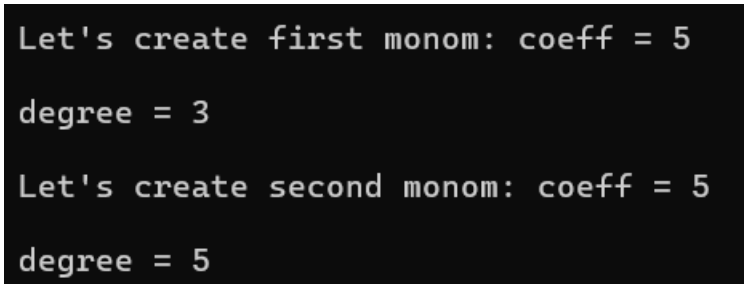

Рис. 9. Создание мономов

2. Далее будут выведены результаты сравнения введенных мономов (рис. 10).

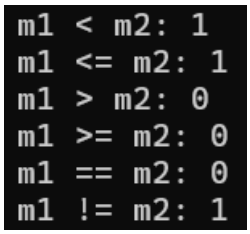

Рис. 10. Сравнение мономов

2.5 Приложение для демонстрации работы полиномов

1. Запустите приложение с названием sample_polynom.exe. В результате появится окно, показанное ниже, вам будет предложено ввести два полинома. (рис. 11).

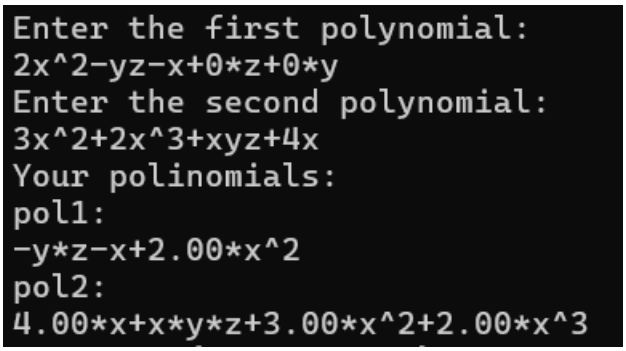

Рис. 11. Создание полиномов

2. Далее будут выведены результаты сложения, вычитания, умножения полиномов и результаты взятия производной по каждой из трех переменной первого полинома (рис. 12).

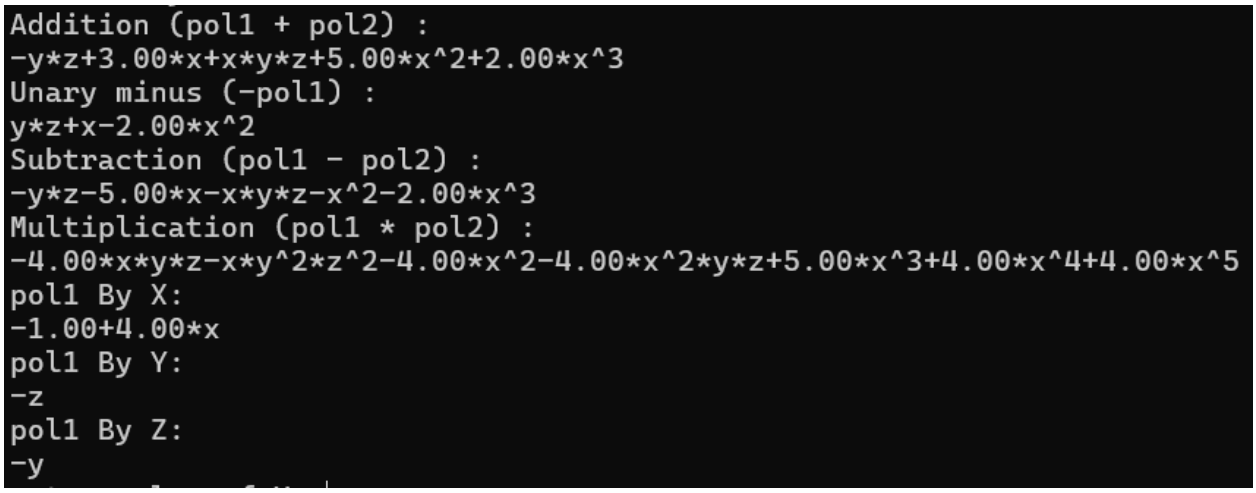

Рис. 12. Сложение, вычитание, умножение и производные полиномов

3. В конце предложат ввести значения переменных, чтобы посчитать значение первого полинома (рис. 13).

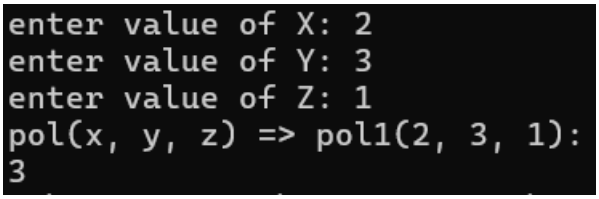

Рис. 13. Вычисление значения полинома

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 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.

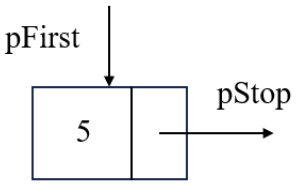

Рис. 14. Добавление в начало, если список пустой

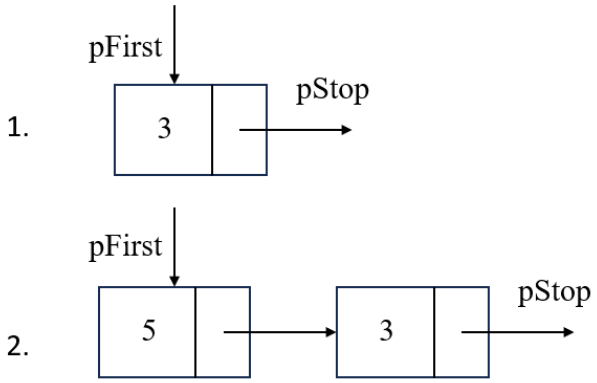

Рис. 15. Добавление в начало, если список не пустой

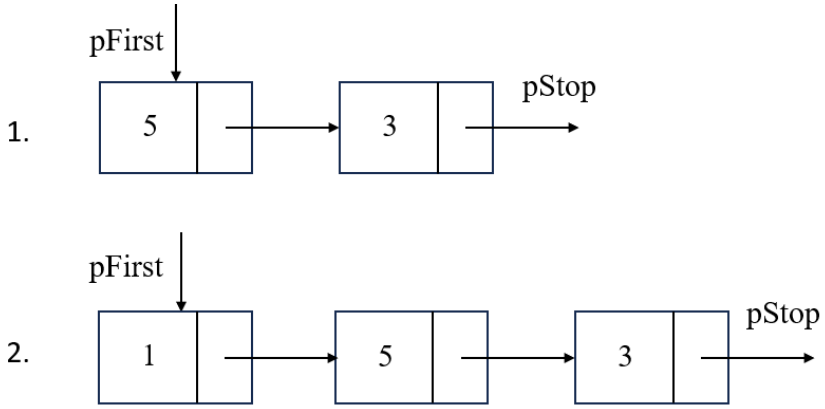

Рис. 16. Операция добавления элемента (1) в начало

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. 14).

Если список не пуст, то создаем новый элемент, с 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.

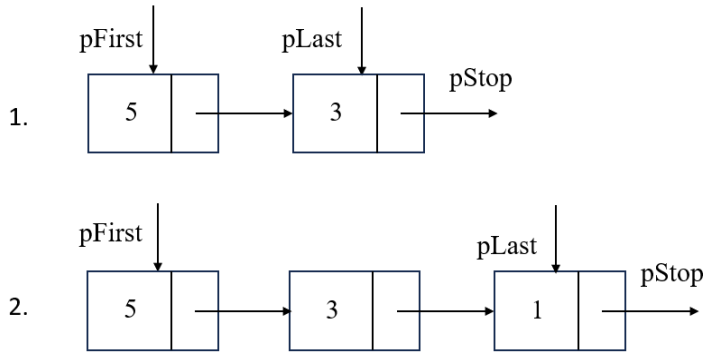

Рис. 17. Операция добавления элемента (1) в конец

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.

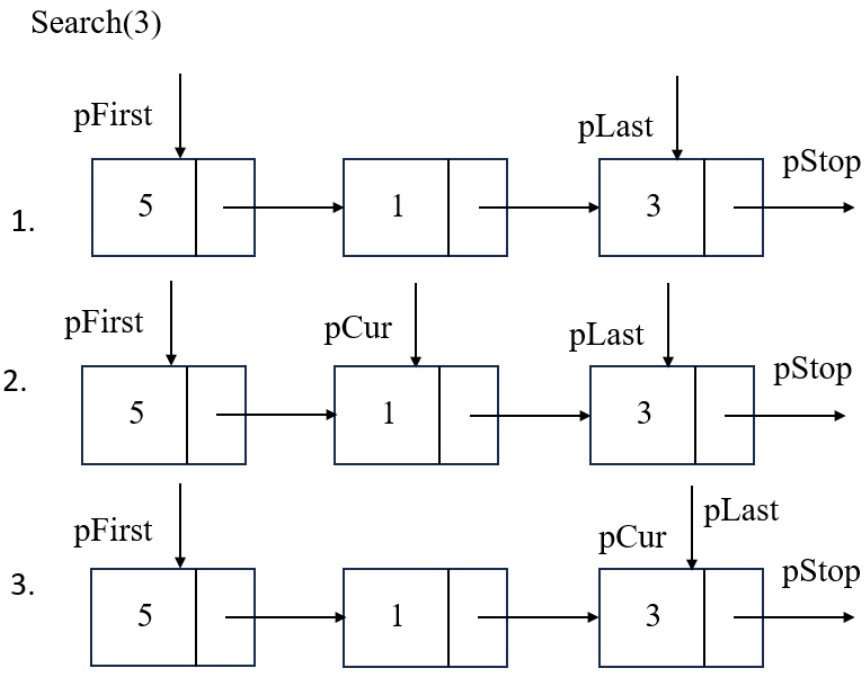

Рис. 18. Операция поиска элемента (3) в списке

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. 20):

- Находим элемент, после которого хотим выполнить операцию вставки с помощью операции поиска (рис. 18).
- 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. 20).

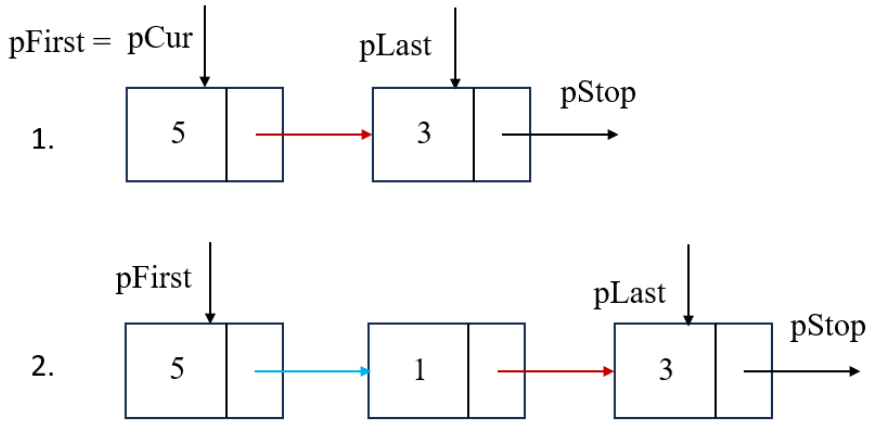

Рис. 19. Операция добавления элемента (1) после текущего. Текущий элемент - первый

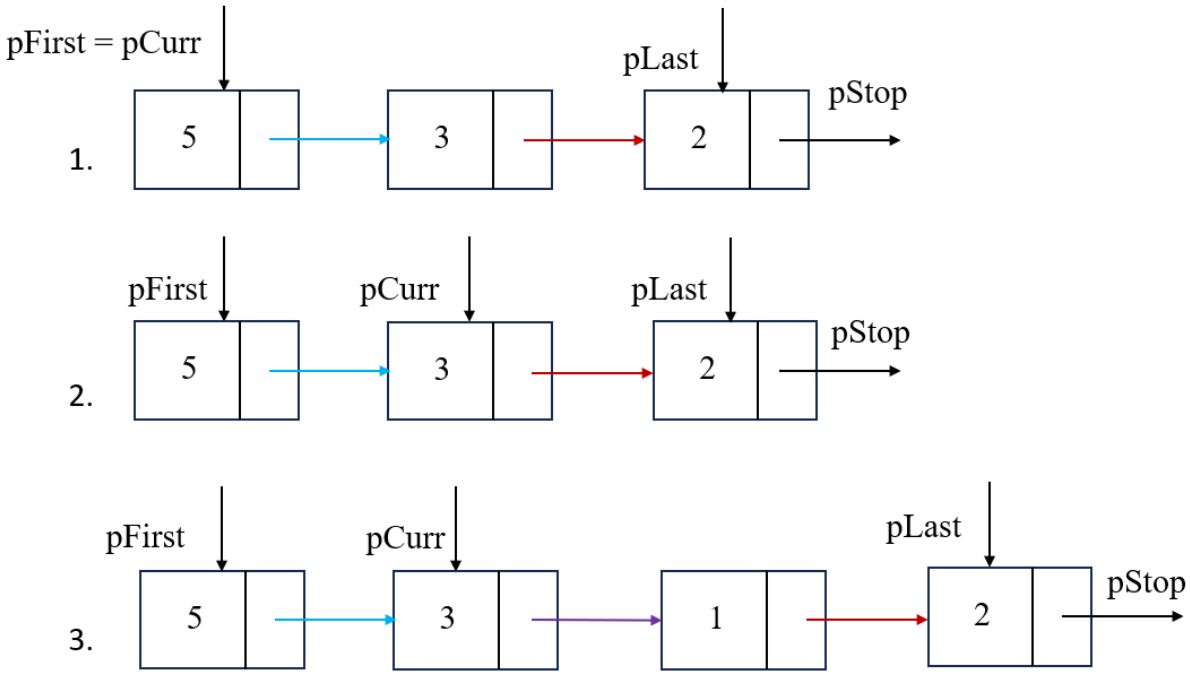

Рис. 20. Операция добавления элемента (1) после текущего (3)

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. 21):

- С помощью операции поиска (рис. 18) 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. 21). 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).

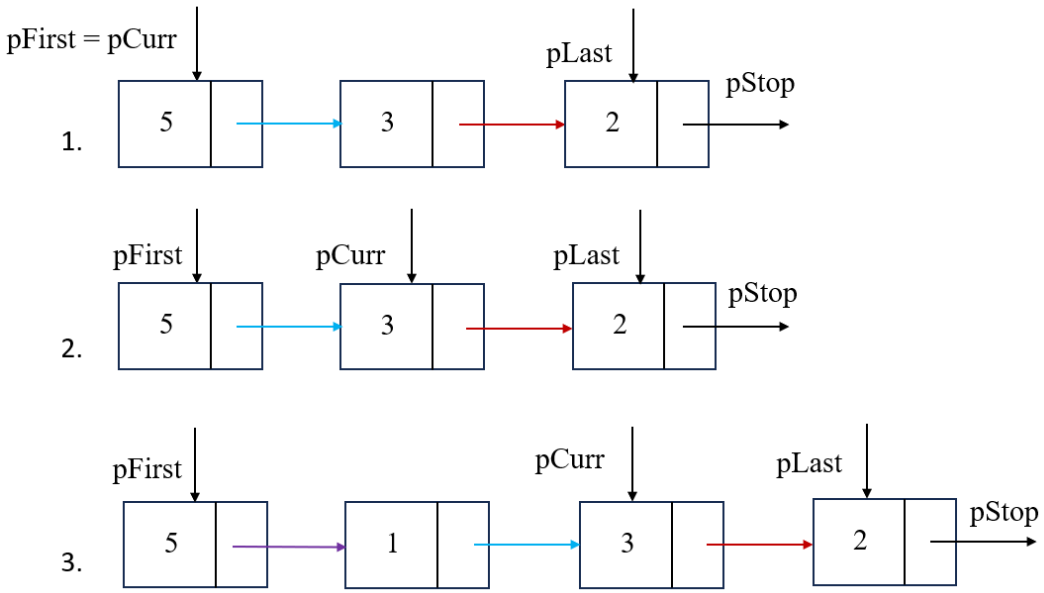

Рис. 21. Операция вставки (1) перед текущим (3)

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).

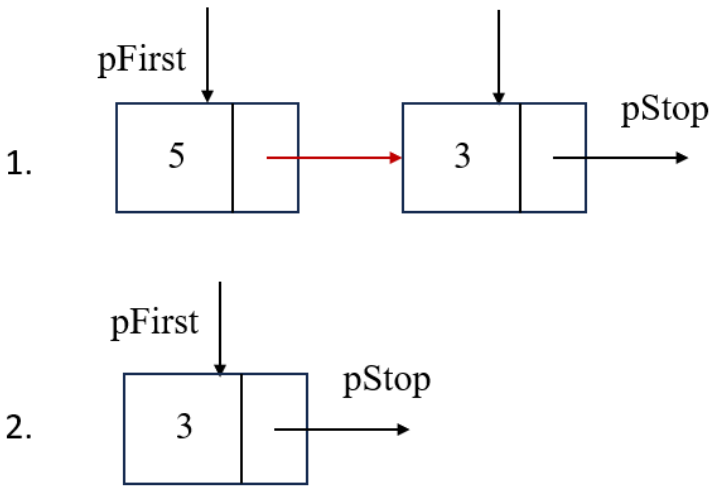

Рис. 22. Операция удаления первого элемента

Операция удаления элемента по его данным

Ищем в списке звено с нужными данными с помощью операции поиска (рис. 18) и удаляем, меняя указатели, чтобы не потерять связь.

Разберем на примере ниже (рис. 23):

- С помощью операции поиска (рис. 18) находим элемент, который хотим удалить
- Создается вспомогательное звено, с указателем на следующий (3) равным советующему у того элемента (1), который хотим удалить. 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)

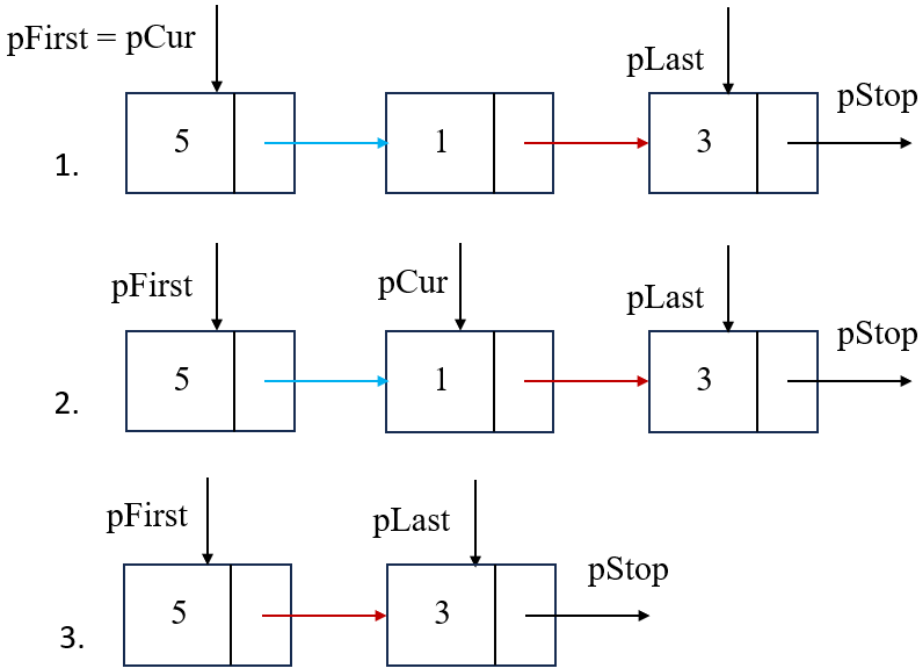

Рис. 23. Операция удаления элемента (1):

Операция получения текущего элемента

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

Пример:

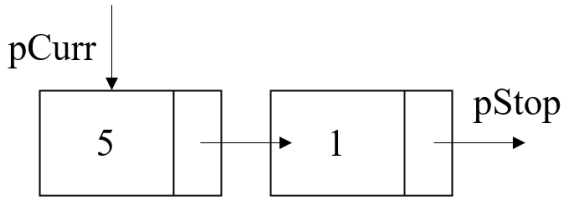

Рис. 24. Операция взятия элемента если текущий по умолчанию

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.

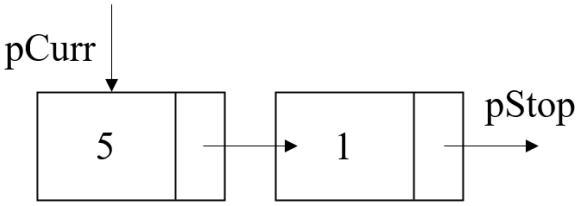

Рис. 25. Операция проверки на полноту

Результат: false

3.1.2 Кольцевой список с 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. 26):

- 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.

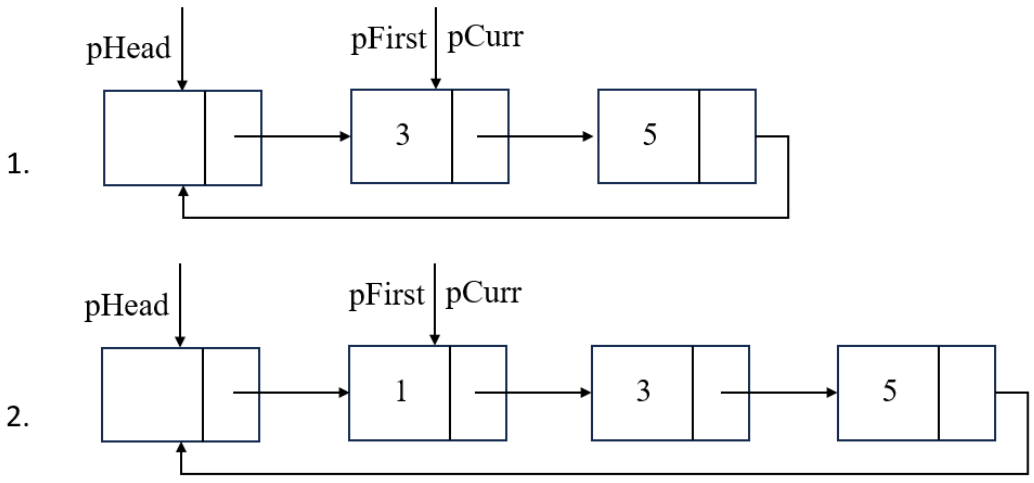

Рис. 26. Операция добавления элемента (1) в начало кольцевого списка с головой

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. 26).

Если список не пуст, то создаём новый элемент и сдвигаем указатели. Разберем на примере ниже (рис. 27):

- 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.

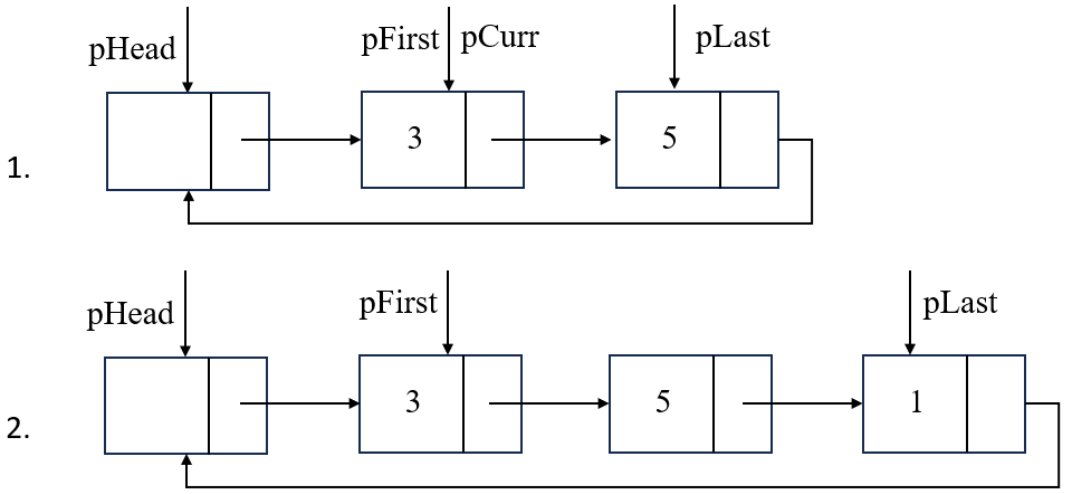

Рис. 27. Операция добавления элемента (1) в конец

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.

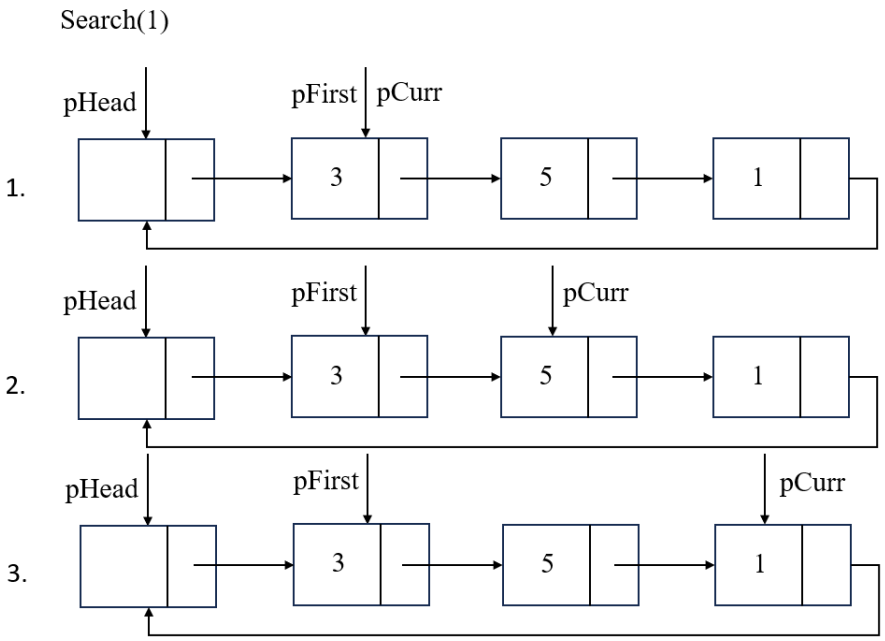

Рис. 28. Операция поиска элемента (1)

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).

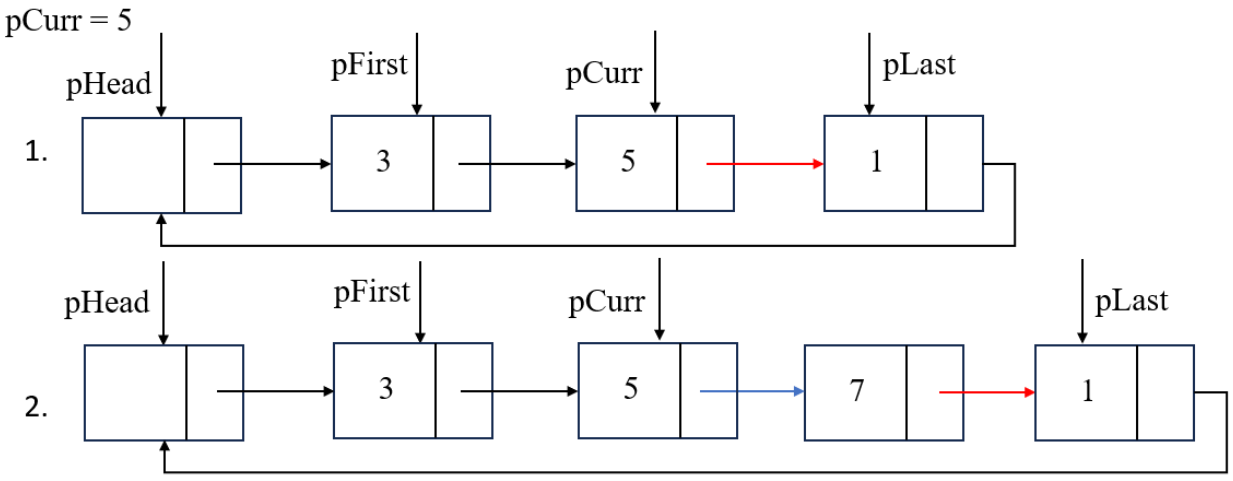

Рис. 29. Операция добавления элемента (7) после текущего (5)

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).

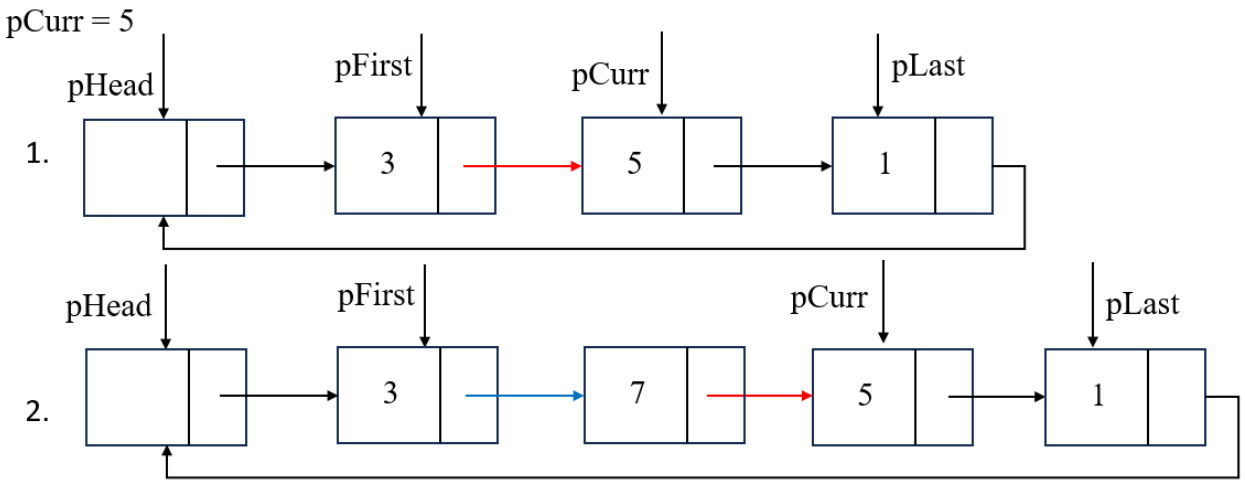

Рис. 30. Операция добавления элемента (7) перед текущим (5)

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.

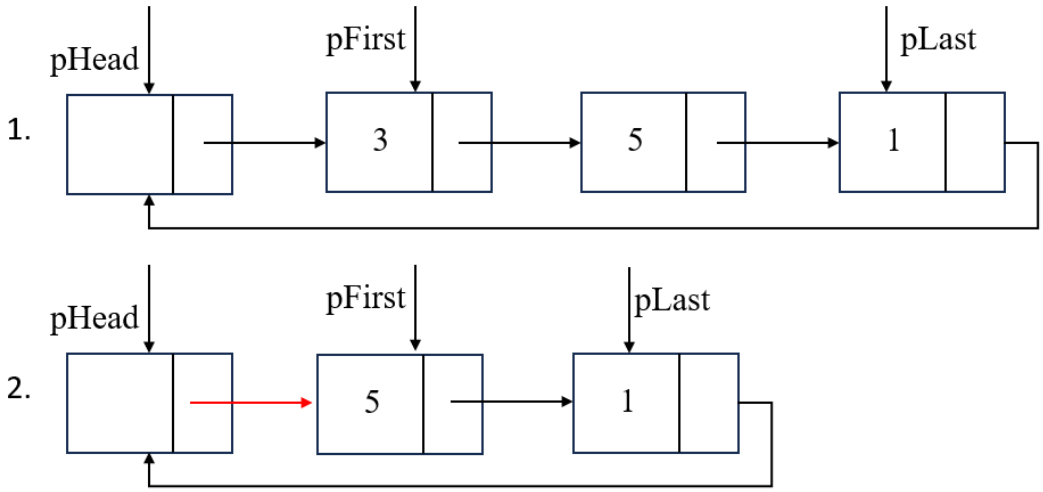

Рис. 31. Операция удаления первого элемента

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)

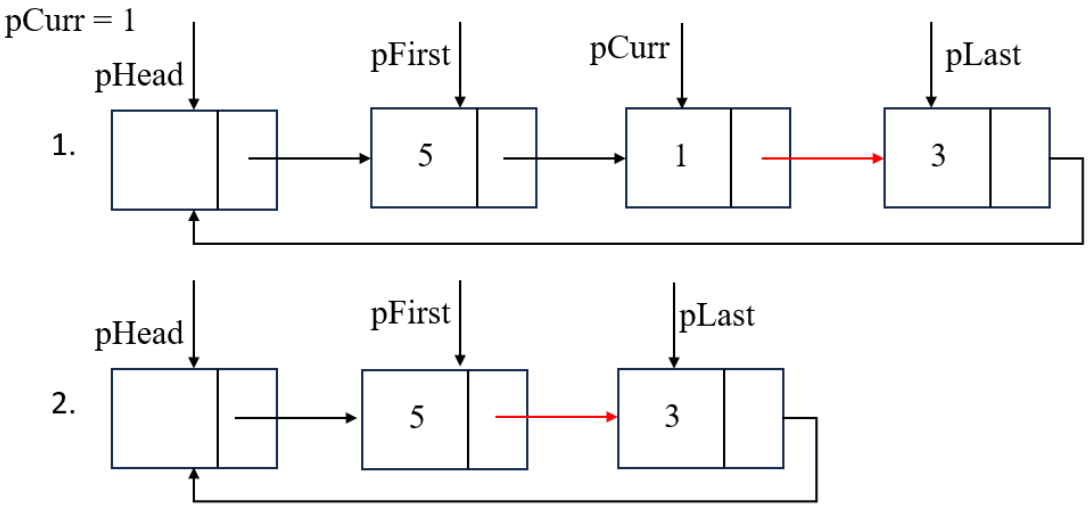

Рис. 32. Операция удаления текущего (1) элемента

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)

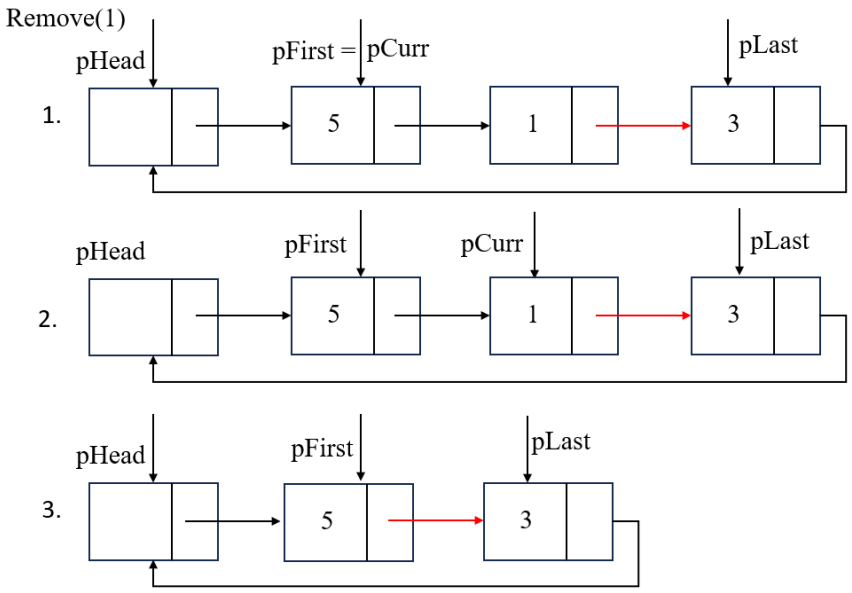

Рис. 33. Операция удаления элемента по его данным (1)

Операция получения текущего элемента

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

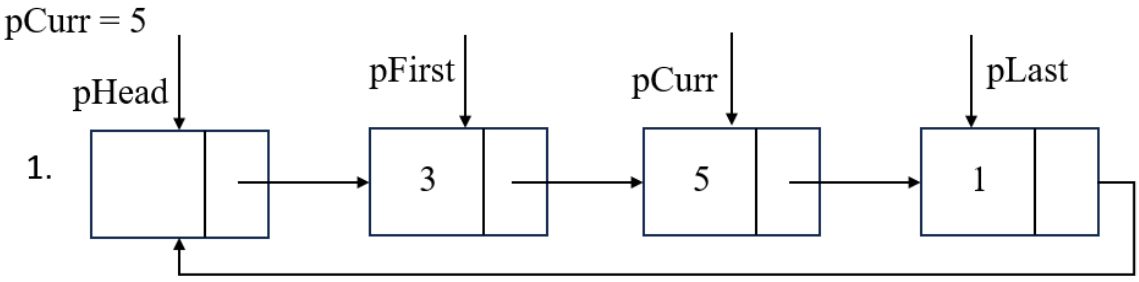

Рис. 34. Операция взятия текущего элемента

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.

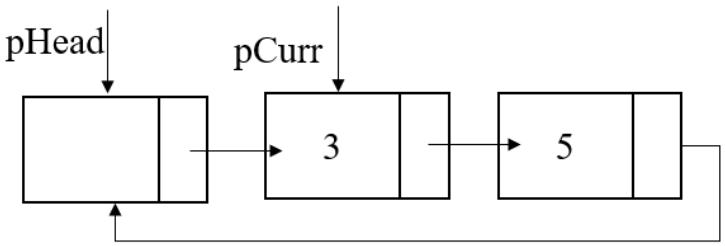

Рис. 35. Операция проверки на полноту

Результат: false

3.1.3 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.

3.1.4 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 x: +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

3.1 Описание программной реализации

3.1.1 Описание структуры TNode

```
template <typename TData>
struct TNode {
    TData data;
    TNode* pNext;
    TNode() {
        this->data = TData();
        this->pNext = nullptr;
    }
    TNode(const TData& data, TNode* pNext = nullptr) {
        this->data = data;
        this->pNext = pNext;
    }
};
```

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 TData& data, 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.

3.1.2 Описание класса TList

```
template <typename TData>
class TList {
protected:
  TNode<TData>* pFirst;
  TNode<TData>* pLast;
  TNode<TData>* pCurr;
  TNode<TData>* pPrev;
  TNode<TData>* pStop;

public:
  TList();
  TList (const TList<TData>&);
  TList (TNode<TData>*);
```

```
  virtual ~TList();
  TNode<TData>* Search (const TData&);
  virtual void InsertFirst (const TData&);
  virtual void InsertLast (const TData&);
  void InsertBefore (const TData&, const TData&);
  void InsertAfter (const TData&, const TData&);
  virtual void RemoveFirst();
  void Remove (const TData&);
  virtual void Clear();
  void Reset();
  void Next();
  TNode<TData>* GetCurr() const;
  virtual bool IsEnded() const;
  bool IsEmpty() const;
  bool IsFull() const;
  friend ostream& operator<<(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<TData>&);

Назначение: конструктор копирования.

Входные параметры: ссылка на существующий объект класса.

Выходные параметры: отсутствуют.

virtual ~TList();

Назначение: деструктор.

Входные параметры отсутствуют.

Выходные параметры: отсутствуют.

TNode<TData>* Search (const TData&);

Назначение: поиск звена с указанным значением.

Входные параметры: искомое значение.

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

virtual void InsertFirst(const TData&);

Назначение: вставляет новое звено с данными в начало списка.

Входные параметры: данные для нового звена.

Выходные параметры: отсутствуют.

```
virtual void InsertLast(const TData&);
```

Назначение: вставляет новое звено с данными в конец списка.

Входные параметры: данные для нового звена.

Выходные параметры: отсутствуют.

```
void InsertBefore(const TData&, const TData&);
```

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

Входные параметры: данные для нового звена, данные звена, перед которым будет вставлен новое звено.

Выходные параметры: отсутствуют.

```
void InsertAfter(const TData&, const TData&);
```

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

Входные параметры: данные для нового звена, данные звена, перед которым будет вставлен новое звено.

Выходные параметры: отсутствуют.

```
void Remove(const TData&);
```

Назначение: удаляет звено с определенными данными из списка.

Входные параметры: данные звена для удаления.

Выходные параметры: отсутствуют.

```
virtual void RemoveFirst();
```

Назначение: удаляет первое звено в списке.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
virtual void Clear();
```

Назначение: отчистка списка.

Входные параметры отсутствуют.

Выходные параметры отсутствуют.

```
void Reset();
```

Назначение: установка текущего звена на первый.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
void Next();
```

Назначение: переход к следующему звену.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
TNode<TData>* GetCurr() const;
```

Назначение: возвращает указатель на текущее звено.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущее звено.

```
virtual bool IsEnded() const;
```

Назначение: проверяет, завершен ли список.

Входные параметры: отсутствуют.

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

```
bool IsEmpty() const;
```

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

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

```
bool IsFull() const;
```

Назначение: проверка списка на заполненность.

Входные параметры: отсутствуют.

Выходные параметры: **true** – если список достиг максимального размера, **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**.

3.1.3 Описание класса headlist

```
template <typename TData>
class headlist : public TList<TData> {
protected:
  TNode<TData>* pHead;
public:
  headlist();
  headlist(const headlist<TData>& list);
  ~headlist();
  void RemoveFirst();
  void Clear();
  void InsertFirst(const TData& data);
  void InsertLast(const TData& data);
  bool IsEnded() const;
  const headlist<TData>& operator=(const headlist<TData>& l);
  friend ostream& operator<<(ostream& out, headlist<TData>& ringList);
};
```

Назначение: представление кольцевого списка.

Поля:

pHead – указатель на головной элемент.

Методы:

```
headlist();
```

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
headlist(const headlist<TData>& list);
```

Назначение: конструктор копирования

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

Выходные параметры: отсутствуют.

```
~headlist();
```

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
void RemoveFirst();
```

Назначение: удаляет первое звено кольцевого списка.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
void Clear();
```

Назначение: отчистка кольцевого списка.

Входные параметры отсутствуют.

Выходные параметры отсутствуют.

```
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.

```
bool IsEnded() const;
```

Назначение: проверка завершения списка.

Входные параметры: отсутствуют.

Выходные параметры: **true** – если текущий элемент указывает на конец списка, **false** – в противном случае.

```
const headlist<TData>& operator=(const headlist<TData>& l);
```

Назначение: операция присваивания.

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

Выходные параметры: ссылка на присвоенный список.

```
friend ostream& operator<<(ostream& out, headlist <TData>& headlist);
```

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет выходной поток, **headlist** – ссылка на объект типа **headlist** который будет выводиться.

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

3.1.4 Описание класса TMonom

```
class TMonom {
public:
  double coeff_;
  int degree_;
  TMonom();
  TMonom(double coeff, int degree);
  bool operator<(const TMonom&) const;
  bool operator<=(const TMonom&) const;
  bool operator>(const TMonom&) const;
  bool operator>=(const TMonom&) const;
  bool operator==(const TMonom&) const;
  bool operator!=(const TMonom&) const;
  friend ostream& operator<< (ostream& out, const TMonom& m);
     };
```

Назначение: представление монома.

Поля:

coeff_ – коэффициент монома.

degree_ – степень монома.

Методы:

```
TMonom();
```

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
TMonom(double coeff, int degree);
```

Назначение: конструктор по умолчанию.

Входные параметры: **coeff** – коэффициент монома, **degree** – степень монома.

Выходные параметры: отсутствуют.

```
bool operator<(const TMonom&) const;
```

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

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

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

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

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

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

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

```
bool operator>(const TMonom&) const;
```

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

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

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

```
bool operator>=(const TMonom&) const;
```

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

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

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

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

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

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

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

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

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

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

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

```
friend ostream& operator<<(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**.

3.1.5 Описание класса TPolynom

```
class TPolynom {
protected:
  string name;
  headlist<TMonom> monoms;
  void smash_pol(const string& name);
  bool check(const string& name);
  void InsertToSort(const TMonom& monom);
public:
```

```
  TPolynom() : monoms() {}
  TPolynom(const string& name);
  TPolynom(const headlist<TMonom>& l);
  TPolynom(const TPolynom& p);
  ~TPolynom() = default;

  TPolynom operator-() const;
  TPolynom operator+(const TPolynom& p);
  TPolynom operator-(const TPolynom& p);
  TPolynom operator*(const TPolynom& p);
  bool operator==(const TPolynom& p) const;

  string TPolynom::ToString() const;

  double operator()(double x, double y, double z) const;
  TPolynom dx() const;
  TPolynom dy() const;
  TPolynom dz() const;

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

  friend ostream& operator<< (ostream& out, TPolynom& pol);
    };
```

Назначение: работа с полиномами

Поля:

monoms – список мономов.

name – строковое представление полинома.

Методы:

void smash_pol(const string& name);

Назначение: обработка имени полинома.

Входные параметры: **name** – строка с именем полинома.

Выходные параметры: отсутствуют.

bool check(const string& name);

Назначение: проверка имени полинома.

Входные параметры отсутствуют:

Выходные параметры: **true** – если верно, **false** - иначе.

void InsertToSort(const TMonom& monom);

Назначение: вставка монома в отсортированный список мономов.

Входные параметры: **monom** – моном для вставки.

Выходные параметры: отсутствуют.

TPolynom();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют.

Выходные параметры: отсутствуют.

TPolynom(const string& name);

Назначение: конструктор с параметрами.

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

Выходные параметры: отсутствуют.

```
TPolynom(const headlist<TMonom>& l);
```

Назначение: конструктор с параметрами.

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

Выходные параметры: отсутствуют.

```
TPolynom(const TPolynom& p);
```

Назначение: конструктор копирования.

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

Выходные параметры: отсутствуют

```
~TPolynom() = default;
```

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
TPolynom operator-()const;
```

Назначение: унарный минус.

Входные параметры: отсутствуют.

Выходные параметры: полином с измененными знаками коэффициентов.

```
TPolynom operator+(const TPolynom& p);
```

Назначение: сложение полиномов.

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

Выходные параметры: сумма полиномов.

```
TPolynom operator-(const TPolynom& p);
```

Назначение: разность полиномов.

Входные параметры: **p** – полином, который будем вычитать.

Выходные параметры: разность полиномов.

```
TPolynom operator*(const TPolynom& p);
```

Назначение: умножение полиномов.

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

Выходные параметры: произведение полиномов.

```
bool operator==(const TPolynom& p) const;
```

Назначение: сравнение полиномов.

Входные параметры: **p** – полином, с которым будем сравнивать.

Выходные параметры: **true** – если полиномы равны, **false** - иначе.

String TPolynom:: ToString() const;

Назначение: получение строкового представления полинома.

Входные параметры: отсутствуют.

Выходные параметры: строка.

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

Назначение: вычисление значения полинома с данными коэффициентами.

Входные параметры: значения переменных.

Выходные параметры: полученный полином.

TPolynom dx() const;

Назначение: дифференцирование полинома по x.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по x.

TPolynom dy() const;

Назначение: дифференцирование полинома по y.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по y.

TPolynom dz() const;

Назначение: дифференцирование полинома по z.

Входные параметры отсутствуют.

Выходные параметры: дифференциал полинома по z.

friend ostream& operator<<(ostream& out, TPolynom& pol)

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет выходной поток, **pol** – ссылка на объект типа **TPolynom** который будет выводиться.

Заключение

В рамках данной лабораторной работы была разработана и реализована структура данных для работы с полиномами. Были созданы классы **`TMonom`** и **`TPolynom`**, 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 структуры TNode

```
using namespace std;
template <typename TData>
struct TNode {
    TData data;
    TNode* pNext;
    TNode() {
        this->data = TData();
        this->pNext = nullptr;
    }
    TNode(const TData& data, TNode* pNext = nullptr) {
        this->data = data;
        this->pNext = pNext;
    }
};
```

Приложение Б. Реализация класса TList

```
#include "node.h"
#include <iostream>
using namespace std;
template <typename TData>
class TList {
protected:
  TNode<TData>* pFirst;
  TNode<TData>* pLast;
  TNode<TData>* pCurr;
  TNode<TData>* pPrev;
  TNode<TData>* pStop;

public:
  TList();
  TList (const TList<TData>&);
  TList (TNode<TData>*);
  virtual ~TList();
  TNode<TData>* Search (const TData&);
  virtual void InsertFirst (const TData&);
  virtual void InsertLast (const TData&);
  void InsertBefore (const TData&, const TData&);
  void InsertAfter (const TData&, const TData&);
  virtual void RemoveFirst();
  void Remove (const TData&);
  virtual void Clear();
  void Reset();
  void Next();
  TNode<TData>* GetCurr() const;
  virtual bool IsEnded() const;
  bool IsEmpty() const;
  bool IsFull() const;

  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 <typename TData>
TList<TData>::TList() {
  pFirst = nullptr;
  pLast = nullptr;
  pCurr = nullptr;
  pPrev = nullptr;
  pStop = nullptr;
}

template <typename TData>
bool TList<TData>::IsEmpty() const {
  return pFirst == nullptr;
}

template <typename TData>
TList<TData>::TList(const TList& l) {
    if (l.IsEmpty())
      {
            pFirst=nullptr;
            pLast=nullptr;
            pCurr=nullptr;
        pPrev=nullptr;
            pStop=nullptr;
        return;
      }

      pFirst = new TNode<TData>(l.pFirst->data);
      TNode<TData>* tmp = pFirst;
      TNode<TData>* ltmp = l.pFirst->pNext;
      while(ltmp != l.pStop)
      {
            tmp->pNext = new TNode<TData>(ltmp->data);
            tmp = tmp->pNext;
            ltmp = ltmp->pNext;
      }
      pLast = tmp;
      pCurr = pFirst;
      pPrev = nullptr;
      pStop = nullptr;
}

template <typename TData>
TList<TData>::TList(TNode<TData>* pNode) {
  pFirst = pNode;
  TNode<TData>* tmp = pNode;
  while (tmp->pNext != nullptr)
    tmp = tmp->pNext;
  pLast = tmp;
  pCurr = pFirst;
  pPrev = nullptr;
  pStop = nullptr;
}

template <typename TData>
void TList<TData>::Clear() {
  if (pFirst == nullptr)
    return;
  TNode<TData>* curr = pFirst;
```

```
  TNode<TData>* next = pFirst->pNext;
  while (next != pStop) {
    delete curr;
    curr = next;
    next = curr->pNext;
  }
  delete curr;
  pCurr = nullptr;
  pFirst = nullptr;
  pPrev = nullptr;
  pLast = nullptr;
}

template <typename TData>
TList<TData>::~TList() {
  Clear();
}

template <typename TData>
bool TList<TData>::IsFull() const {
  TNode<TData>* tmp = new TNode<TData>();
  if (tmp == nullptr)
    return true;
  delete tmp;
  return false;
}

template <typename TData>
bool TList<TData>::IsEnded()const {
  return pCurr == pStop;
}

template <typename TData>
TNode<TData>* TList<TData>::Search(const TData& data) {
  if (IsEmpty()) return nullptr;

  TNode<TData>* prev = nullptr;
  TNode<TData>* curr = pFirst;

  while (curr != pStop && curr->data != data) {
    prev = curr;
    curr = curr->pNext;
  }

  if (curr == pStop) return nullptr;
  pCurr = curr;
  pPrev = prev;
  return curr;
}

template <typename TData>
void TList<TData>::InsertFirst(const TData& data) {
  TNode<TData>* new_first = new TNode<TData>(data, pFirst);
  pFirst = new_first;
  if (pLast == nullptr) {
    pLast = pFirst;
  }
  pCurr = pFirst;
  pPrev = nullptr;
}

template <typename TData>
void TList<TData>::InsertLast(const TData& data) {
```

```
  if (IsEmpty()) {
    InsertFirst(data);
    return;
  }
  TNode<TData>* new_last = new TNode<TData>(data, pStop);
  pLast->pNext = new_last;
  pPrev = pLast;
  pLast = new_last;
  pCurr = new_last;
}

template <typename TData>
void TList<TData>::InsertBefore(const TData& who, const TData& where) {
  TNode<TData>* pWhere = Search(where);
  if (pWhere == nullptr) {
    throw exception("no elements");
  }

  if (pWhere == pFirst) {
    InsertFirst(who);
    return;
  }
  TNode<TData>* new_node = new TNode<TData>(who, pWhere);
  pPrev->pNext = new_node;
  pCurr = new_node;
}

template <typename TData>
void TList<TData>::InsertAfter(const TData& who, const TData& where) {
  TNode<TData>* pWhere = Search(where);
  if (pWhere == nullptr) {
    throw exception("no elements");
  }
  if (pWhere == pLast) {
    InsertLast(who);
    return;
  }
  TNode<TData>* new_node = new TNode<TData>(who, pWhere->pNext);
  pWhere->pNext = new_node;
  pCurr = new_node;
  pPrev = pWhere;
}

template <typename TData>
void TList<TData>::RemoveFirst() {
  TNode<TData>* first = pFirst;
  pFirst = pFirst->pNext;
  delete first;
}

template <typename TData>
void TList<TData>::Remove(const TData& where)
{
      TNode<TData>* pWhere = Search(where);
      if (pWhere == nullptr) {
            throw exception("no elements");
      }

      if (pWhere == pFirst && pWhere->pNext == pStop)
      {
            Clear();
            return;
      }
```

```
      if (pWhere == pFirst)
      {
            RemoveFirst();
            pCurr = pFirst;
            return;
      }
      if (pWhere == pLast)
      {
            pPrev->pNext = pStop;
            pLast = pPrev;
            pCurr = pStop;
            delete pWhere;
            return;
      }
      pPrev->pNext = pWhere->pNext;
      pCurr = pWhere->pNext;
      delete pWhere;

}

template <typename TData>
void TList<TData>::Reset() {
  pCurr = pFirst;
  pPrev = nullptr;
}

template <typename TData>
void TList<TData>::Next() {
  if (pCurr == pStop)
    throw exception("end of the list");
  pPrev = pCurr;
  pCurr = pCurr->pNext;
}

template <typename TData>
TNode<TData>* TList<TData>::GetCurr() const {
  return pCurr;
}
```

Приложение В. Реализация класса headlist

```
#include "list.h"
#include <iostream>
using namespace std;

template <typename TData>
class headlist : public TList<TData> {
protected:
  TNode<TData>* pHead;
public:
  headlist();
  headlist(const headlist<TData>& list);
  ~headlist();
  void RemoveFirst();
  void Clear();
  void InsertFirst(const TData& data);
  void InsertLast(const TData& data);
  bool IsEnded() const;
  const headlist<TData>& operator=(const headlist<TData>& l);

  friend std::ostream& operator<<(std::ostream& out, headlist<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 <typename TData>
headlist<TData>::headlist() : TList<TData>() {
  pHead = new TNode<TData>();
  pHead->pNext = pHead;
  pStop = pHead;
}

template <typename TData>
headlist<TData>::headlist(const headlist<TData>& list) : TList<TData>(list) {
  pHead = new TNode<TData>(list.pHead->data, pFirst);
  if (!list.IsEmpty())
        pLast->pNext = pHead;
  else
        pHead->pNext = pHead;
  pStop = pHead;
}

template <typename TData>
headlist<TData>::~headlist() {
  delete pHead;
  if (pLast != nullptr)
      pLast->pNext = nullptr;
  pStop = nullptr;
}

template <typename TData>
void headlist<TData>::InsertFirst(const TData& data) {
      TList<TData>::InsertFirst(data);
      pHead->pNext = pFirst;
      pStop = pHead;
      pLast->pNext = pHead;
}

template <typename TData>
void headlist<TData>::InsertLast(const TData& data) {
  if (IsEmpty()) {
    headlist<TData>::InsertFirst(data);
    return;
  }
  TList<TData>::InsertLast(data);
}

template <typename TData>
void headlist<TData>::RemoveFirst() {
  TNode<TData>* first = pFirst;
  pFirst = pFirst->pNext;
  pHead->pNext = pFirst;
  delete first;
}

template <typename TData>
void headlist<TData>::Clear()
{
```

```
	TList<TData>::Clear();
	pHead->pNext = pHead;
}

template <typename TData>
const headlist<TData>& headlist<TData>::operator=(const headlist<TData>& l) {
	if (this == &l) return (*this);

	if (l.IsEmpty())
	{
		pFirst=nullptr;
		pLast=nullptr;
		pCurr=nullptr;
		pPrev=nullptr;
		pStop=nullptr;
		return *(this);
	}

	Clear();

	pFirst = new TNode<TData>(l.pFirst->data);
	TNode<TData>* tmp = pFirst;
	TNode<TData>* ltmp = l.pFirst->pNext;
	while(ltmp != l.pStop)
	{
		tmp->pNext = new TNode<TData>(ltmp->data);
		tmp = tmp->pNext;
		ltmp = ltmp->pNext;
	}
	pLast = tmp;
	pLast->pNext = pHead;
	pHead->pNext = pFirst;
	pCurr = pFirst;
	pPrev = nullptr;
	pStop = nullptr;

	return *(this);
}

template <typename TData>
bool headlist<TData>::IsEnded() const {
  if (IsEmpty())
    return true;
  return pCurr == pStop;
}
```

Приложение Г. Реализация класса TMonom

```
#include "tmonom.h"
using namespace std;

TMonom::TMonom() {
  coeff_ = 0;
  degree_ = -1;
}

TMonom::TMonom(double coeff, int degree) {
  if (degree > 999 || degree < 0) throw exception("BadDegree!\n");
  coeff_ = coeff;
  degree_ = degree;
}

bool TMonom::operator<(const TMonom& data) const {
```

```
  return degree_ < data.degree_;
}

bool TMonom::operator<=(const TMonom& data) const {
  return degree_ <= data.degree_;
}

bool TMonom::operator>(const TMonom& data) const {
  return degree_ > data.degree_;
}

bool TMonom::operator>=(const TMonom& data) const {
  return degree_ >= data.degree_;
}
bool TMonom::operator==(const TMonom& data) const {
  return degree_ == data.degree_;
}
bool TMonom::operator!=(const TMonom& data) const {
  return degree_ != data.degree_;
}
```

Приложение Д. Реализация класса TPolynom

```
#include "headlist.h"
#include "tpolynom.h"
using namespace std;

TPolynom::TPolynom(const string& name) : monoms() {
  this->name = name;
  bool res = check(name);
  smash_pol(name);
}

TPolynom::TPolynom(const headlist<TMonom>& list) {
  if (list.IsEmpty()) {
    name = "";
    monoms = headlist<TMonom>();
    return;
  }
  headlist<TMonom> tmp_l(list);
  tmp_l.Reset();
  while (!tmp_l.IsEnded()) {
    InsertToSort(tmp_l.GetCurr()->data);
    tmp_l.Next();
  }
  if (monoms.IsEmpty()) {
    TMonom zero(0, 0);
    monoms.InsertFirst(zero);
  }
  name = ToString();
}

TPolynom::TPolynom(const TPolynom& p) : monoms(p.monoms), name(p.name) {}

bool TPolynom::check(const string& name) {
  string correct = "0123456789xyz*^+-";
  for (char ch : name) {
    if (correct.find(ch) == string::npos) return false;
  }

  return true;
}
```

```
void TPolynom::InsertToSort(const TMonom& monom) {
  monoms.Reset();
  if (monom.coeff_ == 0 && monom.degree_ != 0) { return; }
  if (monom.coeff_ == 0 && monom.degree_ == 0 && !monoms.IsEmpty()) { return;
}

      if (monoms.IsEmpty() || monoms.GetCurr()->data > monom) {
            monoms.InsertFirst(monom);
            return;
      }

      while (!monoms.IsEnded() && monoms.GetCurr()->data < monom) {
            monoms.Next();
      }
      if (monoms.IsEnded()) {
            monoms.InsertLast(monom);
            return;
      }

      if (monoms.GetCurr()->data == monom) {
            monoms.GetCurr()->data.coeff_  =  monoms.GetCurr()->data.coeff_  +
monom.coeff_;
            if (monoms.GetCurr()->data.coeff_ == 0) {
                  monoms.Remove(monoms.GetCurr()->data);
            }
            return;
      }
      monoms.InsertBefore(monom, monoms.GetCurr()->data);
}


string TPolynom::ToString() const {
      string str;
      TPolynom new_this(*this);
      if (new_this.monoms.IsEmpty()) {
            return "0.00";
      }
      bool firstTerm = true;
      new_this.monoms.Reset();

      if (new_this.monoms.GetCurr()->data.coeff_ == 0
            &&
             new_this.monoms.GetCurr()->data.degree_ == 0
            ) {
                  return "0";
      }
      while (!new_this.monoms.IsEnded()) {
            int deg = new_this.monoms.GetCurr()->data.degree_;
            double coeff = new_this.monoms.GetCurr()->data.coeff_;
            int x = deg / 100;
            int y = (deg % 100) / 10;
            int z = deg % 10;
            if (coeff != 0) {
                  if (!firstTerm) {
                        str += ((coeff > 0) ? "+" : "-");
                  }
                  else {
                        if (coeff < 0) str += '-';
                        firstTerm = false;
                  }
                  if (abs(coeff) != 1 || deg == 0) {
                        char buf[15];
                        sprintf(buf, "%.2f", abs(coeff));
```

```
                    str += string(buf);
                }
                string mul_symbol = ((abs(coeff) == 1) ? "" : "*");
      if (x != 0) {
        str += (mul_symbol + "x") + ((x != 1) ? "^" + to_string(x) : "");
      }
      if (y != 0) {
        mul_symbol = (x == 0) ? mul_symbol : "*";
        str += (mul_symbol + "y") + ((y != 1) ? "^" + to_string(y) : "");
      }
      if (z != 0) {
        mul_symbol = (x == 0 && y == 0) ? mul_symbol : "*";
        str += (mul_symbol + "z") + ((z != 1) ? "^" + to_string(z) : "");
      }
          }
          new_this.monoms.Next();
      }
      return str;
}

TPolynom TPolynom::operator-() const {
  TPolynom negativePol(*this);

  while (!negativePol.monoms.IsEnded()) {
    negativePol.monoms.GetCurr()->data.coeff_ = negativePol.monoms.GetCurr()-
>data.coeff_ * (-1);
    negativePol.monoms.Next();
  }
  negativePol.name = negativePol.ToString();
  return negativePol;
}

TPolynom TPolynom::operator+(const TPolynom& p) {
  TPolynom res(p);

  monoms.Reset();
  while (!monoms.IsEnded()) {
    res.InsertToSort(monoms.GetCurr()->data);
    monoms.Next();
  }
  if (res.monoms.IsEmpty()) {
    TMonom zero(0, 0);
    res.monoms.InsertFirst(zero);
  }
  res.name = res.ToString();
  return res;
}

TPolynom TPolynom::operator-(const TPolynom& p) {
  TPolynom res = (*this) + (-p);
  res.name = res.ToString();
  return res;
}

TPolynom TPolynom::operator*(const TPolynom& p) {
  TPolynom res_pol;
  TPolynom tmp_p(p);

  monoms.Reset();
  while (!monoms.IsEnded()) {
        tmp_p.monoms.Reset();
    while (!tmp_p.monoms.IsEnded()) {
      TMonom mon1 = monoms.GetCurr()->data;
```

```
      TMonom mon2 = tmp_p.monoms.GetCurr()->data;
      double newCoeff = mon1.coeff_ * mon2.coeff_;
      int newDegree = mon1.degree_ + mon2.degree_;

      if (newDegree > 999) throw exception("invalid_degree");

      res_pol.InsertToSort(TMonom(newCoeff, newDegree));
      tmp_p.monoms.Next();
    }

    monoms.Next();
  }
  if (res_pol.monoms.IsEmpty()) {
    TMonom zero(0, 0);
    res_pol.monoms.InsertFirst(zero);
  }
  res_pol.name = res_pol.ToString();
  return res_pol;
}

bool TPolynom::operator==(const TPolynom& p) const {
  TPolynom new_this(*this);
  TPolynom new_p(p);

  while (!new_p.monoms.IsEnded() && !new_this.monoms.IsEnded()) {
    if   (new_p.monoms.GetCurr()->data   !=   new_this.monoms.GetCurr()->data)
return false;
    new_p.monoms.Next();
    new_this.monoms.Next();
  }
  if (new_p.monoms.IsEnded() && new_this.monoms.IsEnded()) return true;

  return false;
}

double TPolynom::operator()(double x, double y, double z) const {
  double result = 0;
  TPolynom tmp_this(*this);

  while (!tmp_this.monoms.IsEnded()) {
    double mn;
    mn = tmp_this.monoms.GetCurr()->data.coeff_;
    mn *= pow(x, tmp_this.monoms.GetCurr()->data.degree_ / 100);
    mn *= pow(y, tmp_this.monoms.GetCurr()->data.degree_ / 10 % 10);
    mn *= pow(z, tmp_this.monoms.GetCurr()->data.degree_ % 10);

    result += mn;
    tmp_this.monoms.Next();
  }
  return result;
}

TPolynom TPolynom::dx() const {
  TPolynom dx_pol;
  TPolynom tmp_this(*this);

  while (!tmp_this.monoms.IsEnded()) {
    if (tmp_this.monoms.GetCurr()->data.degree_ / 100 != 0) {
      double    newCoeff    =    tmp_this.monoms.GetCurr()->data.coeff_    *
(tmp_this.monoms.GetCurr()->data.degree_ / 100);
      int newDegree = tmp_this.monoms.GetCurr()->data.degree_ - 100;

      dx_pol.InsertToSort(TMonom(newCoeff, newDegree));
```

```
    }

    tmp_this.monoms.Next();
  }

  return dx_pol;
}

TPolynom TPolynom::dy() const {
  TPolynom dy_pol;
  TPolynom tmp_this(*this);

  while (!tmp_this.monoms.IsEnded()) {
    if (tmp_this.monoms.GetCurr()->data.degree_ / 10 % 10 != 0) {
      double     newCoeff     =     tmp_this.monoms.GetCurr()->data.coeff_     *
(tmp_this.monoms.GetCurr()->data.degree_ / 10 % 10);
      int newDegree = tmp_this.monoms.GetCurr()->data.degree_ - 10;

      dy_pol.InsertToSort(TMonom(newCoeff, newDegree));
    }

    tmp_this.monoms.Next();
  }

  return dy_pol;
}

TPolynom TPolynom::dz() const {
  TPolynom dz_pol;
  TPolynom tmp_this(*this);

  while (!tmp_this.monoms.IsEnded()) {
    if (tmp_this.monoms.GetCurr()->data.degree_  % 10 != 0) {
      double     newCoeff     =     tmp_this.monoms.GetCurr()->data.coeff_     *
(tmp_this.monoms.GetCurr()->data.degree_  % 10);
      int newDegree = tmp_this.monoms.GetCurr()->data.degree_ - 1;

      dz_pol.InsertToSort(TMonom(newCoeff, newDegree));
    }

    tmp_this.monoms.Next();
  }

  return dz_pol;
}

const TPolynom& TPolynom::operator=(const TPolynom& p) {
  if (this == &p) {
    return (*this);
  }

  name = p.name;
  monoms = p.monoms;

  return *(this);
}


void TPolynom::smash_pol(const string& name) {
      string str = name;
      while (!str.empty()) {
            int degree = 0;
            size_t j = str.find_first_of("+-", 1);
```

```
            string monom = str.substr(0, j);
            if (monom[monom.length() - 1] == '^') {
                  throw exception("Negative degree");
            }
            str.erase(0, j);

            string coefficent = monom.substr(0, monom.find_first_of("xyz"));
            TMonom tmp;
            tmp.coeff_ = (coefficent == "" || coefficent == "+") ? 1 :
(coefficent == "-") ? -1 : stod(coefficent);
            monom.erase(0, monom.find_first_of("xyz"));

            for (size_t i = 0; i < monom.size(); ++i) {
                  if (isalpha(monom[i])) {
                        int exp = 1;
                        if (monom[i + 1] == '^') {
                              size_t exp_start = i + 2;
                              while (isdigit(monom[exp_start])) {
                                    exp_start++;
                              }
                              exp = stoi(monom.substr(i + 2, exp_start - i -
2));
                        }
                        switch (monom[i]) {
                        case 'x':
                              degree += exp * 100;
                              break;
                        case 'y':
                              degree += exp * 10;
                              break;
                        case 'z':
                        degree += exp * 1;
                              break;
                        default:
                              throw ("exp");
                              break;
                        }
                  }
            }
            tmp.degree_ = degree;
            if (tmp.coeff_ != 0) {
                  this->InsertToSort(tmp);
            }
      }
  if (this->monoms.IsEmpty()) {
    TMonom zero(0, 0);
    this->monoms.InsertFirst(zero);
  }
  this->name = this->ToString();
}
```

Приложение E. sample_polynom

```
#include "list.h"
#include "tpolynom.h"
#include <iostream>
#include <string>
using namespace std;

int main() {
  try {
    cout << "Enter the first polynomial:\n";
    string P1;
```

```
    cin >> P1;

    cout << "Enter the second polynomial:\n";
    string P2;
    cin >> P2;

    TPolynom pol1 = TPolynom(P1);
    TPolynom pol2 = TPolynom(P2);

    cout << "Your polinomials:\n";
    cout << "pol1:\n";
    cout << pol1.ToString() << "\n";

    cout << "pol2:\n";
    cout << pol2.ToString() << "\n";

    cout << "Addition (pol1 + pol2) :\n";
    cout << (pol1 + pol2).ToString() << "\n";

    cout << "Unary minus (-pol1) :\n";
    cout << (-pol1).ToString() << "\n";

    cout << "Subtraction (pol1 - pol2) :\n";
    cout << (pol1 - pol2).ToString() << "\n";

    cout << "Multiplication (pol1 * pol2) :\n";
    cout << (pol1 * pol2).ToString() << "\n";

    cout << "pol1 By X:\n";
    cout << (pol1.dx()).ToString() << "\n";

    cout << "pol1 By Y:\n";
    cout << (pol1.dy()).ToString() << "\n";

    cout << "pol1 By Z:\n";
    cout << (pol1.dz()).ToString() << "\n";

    double x, y, z;
    cout << "enter value of X: ";
    cin >> x;
    cout << "enter value of Y: ";
    cin >> y;
    cout << "enter value of Z: ";
    cin >> z;

    cout << "pol(x, y, z) => pol1(" << x << ", " << y << ", " << z << "):\n";
    cout << pol1(x, y, z);
  }
  catch (const exception ex) {
    cerr << ex.what() << "\n";
  }
  return 0;
}
```