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 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Чистов А.Д./

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc164027825)

[1 Постановка задачи 4](#_Toc164027826)

[2 Руководство пользователя 5](#_Toc164027827)

[2.1 Приложение для демонстрации работы полиномов 5](#_Toc164027828)

[3 Руководство программиста 7](#_Toc164027829)

[3.1 Описание алгоритмов 7](#_Toc164027830)

[3.1.1 Линейный односвязный список 7](#_Toc164027831)

[3.1.2 Кольцевой список с головой 8](#_Toc164027832)

[3.1.3 Полином 9](#_Toc164027833)

[3.2 Описание программной реализации 10](#_Toc164027834)

[3.2.1 Описание класса TNode 10](#_Toc164027835)

[3.2.2 Описание класса TList 11](#_Toc164027836)

[3.2.3 Описание класса THeadRingList 14](#_Toc164027837)

[3.2.4 Описание класса TMonom 15](#_Toc164027838)

[3.2.5 Описание класса TPolynom 17](#_Toc164027839)

[Заключение 21](#_Toc164027840)

[Литература 22](#_Toc164027841)

[Приложения 23](#_Toc164027842)

[Приложение А. Реализация класса TNode 23](#_Toc164027843)

[Приложение Б. Реализация класса TList 23](#_Toc164027844)

[Приложение В. Реализация класса THeadRingList 27](#_Toc164027845)

[Приложение Г. Реализация класса TMonom 28](#_Toc164027846)

[Приложение Д. Реализация класса TPolynom 28](#_Toc164027847)

# Введение

Полином — это алгебраическое выражение, состоящее из суммы или разности членов, каждый из которых является произведением переменной (или переменных) на некоторую степень.

Полиномы широко используются в математике и её приложениях. Они играют важную роль в алгебре, анализе, теории чисел, физике, инженерии и других областях. Например, они часто используются при дифференцировании и интегрировании функций. Поскольку производная или интеграл полинома легко вычисляется, это делает этот тип алгебраических выражений удобным инструментом при анализе функций. Таким образом, полиномы представляет собой мощный и универсальный инструмент в математике и науке.

# Постановка задачи

Цель – реализовать классы THeadRingList и TPolynom

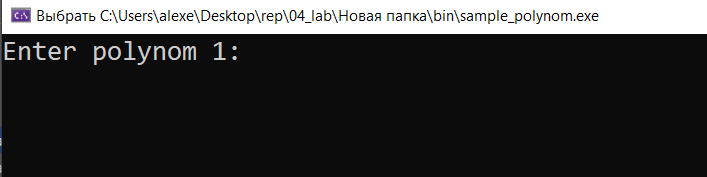
Задачи:

1. Исследовать тематическую литературу.
2. Реализовать класс THeadRingList.
3. Реализовать класс TPolynom.
4. Провести тестирование разработанных классов для проверки их корректной работы.
5. Сделать выводы о проделанной работе.

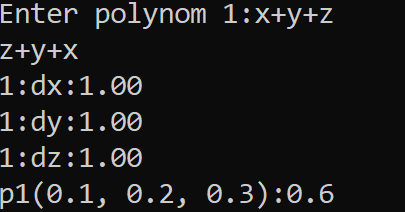
# Руководство пользователя

## Приложение для демонстрации работы полиномов

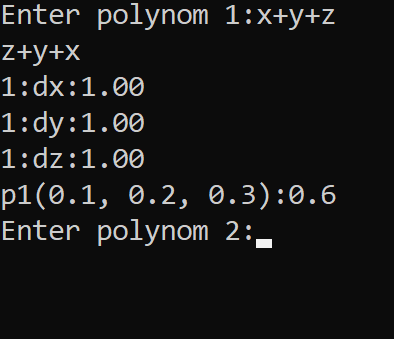
1. Запустите приложение с названием sample\_polynom.exe. В результате появится окно, показанное ниже, где вам нужно будет ввести первый полином (Рис. 1).



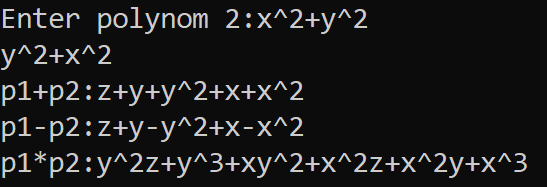
1. Основное окно программы
2. Далее вам будет представлены производные по каждой из трех переменных x, y, z, а также значение полинома в точке с координатами (0.1,0.2,0.3) (Рис. 2).



1. Результат работы программы
2. Далее вам необходимо будет ввести второй полином (Рис. 3).



1. Ввод второго полинома
2. Далее вам будет представлен результат в виде суммы, разности и произведения двух полиномов (Рис. 4).



1. Результат работы программы

# Руководство программиста

## Описание алгоритмов

### Линейный односвязный список

Связный список — базовая динамическая структура данных в информатике, состоящая из узлов, содержащих данные и ссылки на следующий и/или предыдущий узел списка.

Данный класс поддерживает следующие операции:

* **Операция добавления в начало**

Операция добавления элемента реализуется при помощи указателя на первый элемент. Если структура хранения пуста, то мы просто создаем новый элемент, иначе создаём новый элемент и сдвигаем указатель на начало.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 |  |

* **Операция добавления в конец**

Операция добавления элемента реализуется при помощи указателя на последний элемент. Если структура хранения пуста, то мы просто создаем новый элемент, иначе создаём новый элемент и сдвигаем указатель на конец.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 |  |  |

Операция добавления элемента (1) в конец:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 |  |

* **Операция добавления после текущего**

Операция добавления элемента реализуется при помощи указателя на текущий элемент.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 |  |

* **Операция поиска.**

Операция поиска ищет элемент в списке. Возвращает нужный элемент, если он есть.

* **Операция удаления элемента**

Операция удаления элемента реализуется при помощи функции поиска

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  |  |

* **Операция получения текущего элемента.**

Операция взятия элемента с вершины реализуется указателя на текущий элемент.

* **Операция вставки в сортированный список**

Операция вставляет элемент в отсортированный список.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 |  |

* **Операция сортировки.**

Операция сортировки позволяет сортировать список.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 |  |

### Кольцевой список с головой

Кольцевые списки могут быть представлены как круг, где последний элемент связан с первым, образуя замкнутую структуру данных. Такой подход обеспечивает удобство в работе с элементами списка и позволяет производить циклические операции без необходимости проверки на достижение конца списка.

Данный класс поддерживает все операции, которые присутствуют в обычном списке.

### Полином

Полином представляет собой кольцевой список мономов, который получается в результате преобразования входной строки.

Данный класс поддерживает следующие операции:

* **Операция суммирования полиномов**

Операция суммирования полиномов согласно математическим правилам

Пример:

Первый полином: x^3+y^2+y+x+1

Второй полином x^3+y+z+1

Результат: 2x^3+y^2+2y+x+2+z

* **Операция вычитания полиномов**

Операция вычитания полиномов согласно математическим правилам

Пример:

Первый полином: 2x^3+y^2+y+x

Второй полином x^3-2y^2+z

Результат: x^3+3y^2+y^1+x-z

* **Операция произведения полиномов**

Операция произведения полиномов согласно математическим правилам

Пример:

Первый полином: x^2+y

Второй полином x^3+x

Результат: x^5+x^3+x^3\*y+x\*y

* **Операция дифференцирования полиномов**

Операция дифференцирования полинома согласно математическим правилам по переменным x, y или z.

Пример:

Первый полином: z^3+z^2+y+x+1

Результат после дифференцирования функции по z: 3z^2+2z

## Описание программной реализации

### Описание класса TNode

template <typename T>

struct TNode {

T data;

TNode<T>\* pNext;

TNode() : data(), pNext(nullptr) {};

TNode(const T& data) : data(data), pNext(nullptr) {};

TNode(TNode<T>\* \_pNext) : data(), pNext(\_pNext) {};

TNode(const T& data, TNode<T>\* \_pNext) : data(data), pNext(\_pNext) {};

};

Поля:

* data – данные, хранящиеся в звене.
* pNext – указатель на следующее звено.

Методы:

TNode();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

TNode(const T& data);

Назначение: конструктор с параметром.

Входные параметры: data – данные для хранения в звене.

Выходные параметры: отсутствуют.

TNode(TNode<T>\* \_pNext);

Назначение: конструктор с параметром.

Входные параметры: \_pNext – указатель на следующее звено.

Выходные параметры: отсутствуют.

TNode(const T& data, TNode<T>\* \_pNext);

Назначение: конструктор с параметрами.

Входные параметры: data – данные для хранения в звене, \_pNext – указатель на следующее звено.

Выходные параметры: отсутствуют.

### Описание класса TList

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pCurr;

TNode<T>\* pStop;

TNode<T>\* pLast;

public:

TList();

TList(TNode<T>\* \_pFirst);

TList(const TList<T>& list);

virtual ~TList();

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

virtual void remove(const T& data);

virtual void insert\_before(const T& data, const T& nextdata);

void insert\_last(const T& data);

void insert\_after(const T& data, const T& beforedata);

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

TNode<T>\* GetCurrent() const;

void clear();

int GetSize() const;

bool IsEmpty() const;

bool IsFull() const;

bool IsEnded()const;

void next();

void reset();

void insert\_sort(const T& data);

void sort();

};

Поля:

* pFirst– указатель на первый элемент.
* pStop – указатель на конец списка.
* pCurr – указатель на текущий элемент.
* pPrev – указатель на предыдущий элемент.
* pLast – указатель на последний элемент.

Методы:

TList();

Назначение: создание пустого списка.

Входные параметры: отсутствуют.

Выходные параметры: новый объект класса TList.

TList(TNode<T>\* \_pFirst);

Назначение: создание списка с заданным начальным узлом

Входные параметры: \_pFirst – указатель на первый узел списка

Выходные параметры: новый объект класса TList.

TList(const TList<T>& list);

Назначение: создание копии существующего списка.

Входные параметры: list – существующий список для копирования.

Выходные параметры: новый объект класса TList, являющийся копией списка list.

virtual ~TList();

Назначение: освобождение памяти списка при удалении объекта.

Входные параметры: отсутствуют.

Выходные параметры: освобожденная память объекта класса TList.

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

Назначение: вставляет новый узел с данными в начало списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

virtual void remove(const T& data);

Назначение: удаляет узел с определенными данными из списка.

Входные параметры: data – данные узла для удаления.

Выходные параметры: отсутствуют.

void insert\_before(const T& data, const T& nextdata);

Назначение: вставляет новый узел с данными перед узлом с определенными данными.

Входные параметры: data – данные для нового узла, nextdata – данные узла, перед которым будет вставлен новый узел.

Выходные параметры: отсутствуют.

void insert\_last (const T& data);

Назначение: вставляет новый узел с данными в конец списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

void insert\_after (const T& data, const T& beforedata);

Назначение: вставляет новый узел с данными после узла с определенными данными.

Входные параметры: data – данные для нового узла, beforedata – данные узла, после которого будет вставлен новый узел.

Выходные параметры: отсутствуют.

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

Назначение: поиск узла с указанным значением.

Входные параметры: data – искомое значение.

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

TNode<T>\* GetCurrent();

Назначение: возвращает указатель на текущий узел.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущий узел.

void clear();

Назначение: очищает список, освобождает выделенную память.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

int GetSize() const;

Назначение: возвращает текущий размер списка.

Входные параметры: отсутствуют.

Выходные параметры: размер списка (целочисленное значение).

bool IsEmpty() const;

Назначение: проверяет, пуст ли список.

Входные параметры: отсутствуют.

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

bool IsFull() const;

Назначение: проверяет, полон ли список.

Входные параметры: отсутствуют.

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

bool IsEnded() const;

Назначение: проверяет, достигли ли конца списка.

Входные параметры: отсутствуют.

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

void next();

Назначение: переход к следующему узлу.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

virtual void reset();

Назначение: установка текущего узла как первого.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void insert\_sort(const T& data);

Назначение: вставляет новый узел с данными в отсортированный список

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void sort();

Назначение: сортировка элементов списка

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

### Описание класса THeadRingList

template <typename T>

class THeadRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

THeadRingList();

THeadRingList(const THeadRingList& ringL);

virtual ~THeadRingList();

void insert\_first(const T& data);

void insert\_before(const T& who, const T& where);

};

Поля:

pHead – указатель на головной элемент.

Методы:

THeadRingList ();

Назначение: конструктор без параметров, создает пустой кольцевой список.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

THeadRingList (const TRingList<T>& rlist);

Назначение: конструктор копирования, создает копию существующего кольцевого списка.

Входные параметры: rlist – ссылка на существующий кольцевой список.

Выходные параметры: отсутствуют.

virtual ~ THeadRingList ();

Назначение: виртуальный деструктор, освобождает выделенную память при уничтожении объектов производных классов.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void insert\_first (const T& data);

Назначение: вставляет новый узел с данными в начало списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

void insert\_before(const T& data, const T& nextdata);

Назначение: вставляет новый узел с данными перед узлом с определенными данными.

Входные параметры: data – данные для нового узла, nextdata – данные узла, перед которым будет вставлен новый узел.

Выходные параметры: отсутствуют.

### Описание класса TMonom

class TMonom {

public:

double coeff;

int degree;

TMonom();

TMonom(const TMonom& monom);

TMonom(double \_coeff, int \_degree);

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

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

bool operator <(const TMonom& data)const;

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

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

TMonom operator+(const TMonom& monom)const;

};

Назначение: представление монома.

Поля:

coeff – коэффициент монома.

degree – степень монома.

Методы:

Tmonom();

Назначение: конструктор по умолчанию, инициализирует объект TMonom с коэффициентом и степенью равными нулю.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TMonom(const TMonom& monom);

Назначение: конструктор копирования, создает копию существующего TMonom.

Входные параметры: monom – ссылка на существующий объект TMonom.

Выходные параметры: отсутствуют.

TMonom(double \_coeff, int \_degree);

Назначение: создает объект TMonom с указанным коэффициентом и степенью.

Входные параметры: \_coeff – коэффициент, \_degree - степень.

Выходные параметры: отсутствуют.

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

Назначение: перегруженный оператор "равно". Проверяет равенство коэффициента и степени двух объектов TMonom.

Входные параметры: data – ссылка на объект TMonom для сравнения.

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

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

Назначение: перегруженный оператор "не равно". Проверяет неравенство коэффициента и степени двух объектов TMonom.

Входные параметры: data – ссылка на объект TMonom для сравнения.

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

bool operator < (const TMonom& data) const;

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

Входные параметры: data – ссылка на объект TMonom для сравнения.

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

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

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

Входные параметры: data – ссылка на объект TMonom для сравнения.

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

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

Назначение: произведение мономов: коэффиценты умножаются, а степени складываются

Входные параметры: monom – ссылка на объект TMonom

Выходные параметры: моном, который получился в результате операции

TMonom operator+(const TMonom& monom)const;

Назначение: произведение мономов: коэффиценты складываются, а степени не изменяются

Входные параметры: monom – ссылка на объект TMonom

Выходные параметры: моном, который получился в результате операции

### Описание класса TPolynom

class TPolynom {

private:

string name;

THeadRingList<TMonom>\* monoms;

void ParseMonoms();

void conversion(string& str) const;

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const THeadRingList<TMonom>\* m);

TPolynom(const TPolynom& p);

~TPolynom();

TPolynom operator +(const TPolynom& p);

TPolynom operator -(const TPolynom& p);

TPolynom operator-() const;

TPolynom operator \*(const TPolynom& p);

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

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

TPolynom dx() const;

TPolynom dy() const;

TPolynom dz() const;

string ToString()const;

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

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

friend ostream& operator<<(ostream& out, const TPolynom& p);

};

Назначение: представление полинома.

Поля:

name – строка полинома.

monoms – кольцевой линейный односвязный список.

Методы:

Tpolynom();

Назначение: конструктор по умолчанию, создает объект TPolynom с пустым списком мономов и пустым именем.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TPolynom(const string& name);

Назначение: создает объект TPolynom с указанным именем.

Входные параметры: name – строка, используемая в качестве имени полинома.

Выходные параметры: отсутствуют.

TPolynom(const THeadRingList<TMonom>\* m);

Назначение: создает объект TPolynom на основе существующего кольцевого списка мономов.

Входные параметры: THeadRingList<TMonom>\* m – ссылка на кольцевой список мономов.

Выходные параметры: отсутствуют.

TPolynom(const TPolynom& p);

Назначение: конструктор копирования, создает копию существующего объекта TPolynom.

Входные параметры: p – ссылка на существующий объект TPolynom.

Выходные параметры: отсутствуют.

~Tpolynom();

Назначение: деструктор, освобождает память при уничтожении объекта TPolynom.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TPolynom operator+(const TPolynom& p);

Назначение: перегруженный оператор сложения полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом сложения полиномов.

TPolynom operator-(const TPolynom& p);

Назначение: перегруженный оператор вычитания полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом вычитания полиномов.

TPolynom operator-();

Назначение: получение полинома с противоположными по знаку коэффицентами

Входные параметры: отсутствуют.

Выходные параметры: объект TPolynom с противоположными знаками

TPolynom operator\*(const TPolynom& p);

Назначение: перегруженный оператор умножения полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом умножения полиномов.

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

Назначение: перегруженный оператор присваивания.

Входные параметры: p – ссылка на объект TPolynom, который присваивается текущему объекту.

Выходные параметры: копия объекта TPolynom после присваивания.

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

Назначение: вычисляет значение полинома для заданных значений переменных x, y и z.

Входные параметры: x, y, z

Выходные параметры: значение полинома с заданными значениями переменных.

TPolynom dx() const;

Назначение: возвращает производную по переменной «x» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «x».

TPolynom dy() const;

Назначение: возвращает производную по переменной «y» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «y».

TPolynom dz() const;

Назначение: возвращает производную по переменной «z» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «z».

string ToString()const;

Назначение: возвращает строковое представление полинома.

Входные параметры: отсутствуют.

Выходные параметры: строковое представление полинома.

void ParseMonoms();

Назначение: разбирает строку, представляющую полином, и создает соответствующий список мономов.

Входные параметры: s – строка, представляющая полином.

Выходные параметры: отсутствуют.

friend ostream& operator<<(ostream& os, const TPolynom& p);

Назначение: оператор вывода для класса TPolynom.

Входные параметры:

os – ссылка на объект типа ostream, который представляет выходной поток.

p – ссылка на объект типа TPolynom который будет выводиться.

Выходные параметры: ссылка на объект типа ostream.

void conversion(string& str) const;

Назначение: удаляет из строки пробелы

Входные параметры: s – строка, представляющая полином.

Выходные параметры: отсутствуют.

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

Назначение: перегруженный оператор "равно". Проверяет равенство коэффициента и степени двух объектов TPolynom.

Входные параметры: p – ссылка на объект TPolynom для сравнения.

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

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

Назначение: перегруженный оператор "не равно". Проверяет равенство коэффициента и степени двух объектов TPolynom.

Входные параметры: p – ссылка на объект TPolynom для сравнения.

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

# Заключение

В результате данной лабораторной работе удалось изучить и реализовать алгоритм обработки полиномов на основе связных списков. Была освоена работа с мономами и полиномами, а также разработали функционал для выполнения операций над ними: сложение, вычитание, умножение, и вычисление значений при заданных переменных. Это обеспечивает удобный и эффективный способ решения математических задач, требующих работы с полиномами.

# Литература

1. Кольцевой односвязный список [[Структуры данных в C# | Кольцевой односвязный список (metanit.com)](https://metanit.com/sharp/algoritm/2.7.php)].
2. Лекция «Списковые структуры хранения» Сысоев А. В. [ <https://cloud.unn.ru/s/6g44ey6HFB4ncDy>].

# Приложения

## Приложение А. Реализация класса TNode

template <typename T>

struct TNode {

T data;

TNode<T>\* pNext;

TNode() : data(), pNext(nullptr) {};

TNode(const T& data) : data(data), pNext(nullptr) {};

TNode(TNode<T>\* \_pNext) : data(), pNext(\_pNext) {};

TNode(const T& data, TNode<T>\* \_pNext) : data(data), pNext(\_pNext) {};

};

## Приложение Б. Реализация класса TList

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pCurr;

TNode<T>\* pStop;

TNode<T>\* pLast;

public:

TList();

TList(TNode<T>\* \_pFirst);

TList(const TList<T>& list);

virtual ~TList();

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

virtual void remove(const T& data);

virtual void insert\_before(const T& data, const T& nextdata);

void insert\_last(const T& data);

void insert\_after(const T& data, const T& beforedata);

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

TNode<T>\* GetCurrent() const;

void clear();

int GetSize() const;

bool IsEmpty() const;

bool IsFull() const;

bool IsEnded()const;

void next();

void reset();

void insert\_sort(const T& data);

void sort();

};

template <typename T>

TList<T>::TList() {

pFirst = nullptr;

pLast = nullptr;

pCurr = nullptr;

pStop = nullptr;

}

template <typename T>

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

if (l.IsEmpty())

{

pFirst = nullptr;

pLast = nullptr;

pCurr = nullptr;

pStop = nullptr;

return;

}

pFirst = new TNode<T>(l.pFirst->data);

TNode<T>\* tmp = pFirst;

TNode<T>\* ltmp = l.pFirst->pNext;

while (ltmp != l.pStop)

{

tmp->pNext = new TNode<T>(ltmp->data);

tmp = tmp->pNext;

ltmp = ltmp->pNext;

}

pLast = tmp;

pCurr = pFirst;

pStop = nullptr;

}

template <typename T>

TList<T>::TList(TNode<T>\* pNode) {

pFirst = pNode;

TNode<T>\* tmp = pNode;

while (tmp->pNext != nullptr)

tmp = tmp->pNext;

pLast = tmp;

pCurr = pFirst;

pStop = nullptr;

}

template <typename T>

void TList<T>::clear() {

if (pFirst == nullptr) return;

TNode<T>\* curr = pFirst;

while (curr != pStop) {

TNode<T>\* next = curr->pNext;

delete curr;

curr = next;

}

pFirst = nullptr;

pLast = nullptr;

}

template <typename T>

TList<T>::~TList() {

clear();

}

template <typename T>

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

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

if (tmp == nullptr)

return true;

delete tmp;

return false;

}

template <typename T>

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

return (pFirst == nullptr);

}

template <typename T>

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

return pCurr == pStop;

}

template <typename T>

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

TNode<T>\* curr = pFirst;

while (curr != pStop && curr->data != data) {

curr = curr->pNext;

}

if (curr == pStop) {

throw ("Element not found!");

}

return curr;

}

template <typename T>

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

TNode<T>\* new\_first = new TNode<T>(data, pFirst);

pFirst = new\_first;

if (pLast == nullptr) {

pLast = pFirst;

}

pCurr = pFirst;

}

template <typename T>

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

if (IsEmpty()) {

insert\_first(data);

return;

}

TNode<T>\* new\_last = new TNode<T>(data, pStop);

pLast->pNext = new\_last;

pLast = new\_last;

pCurr = new\_last;

}

template <typename T>

void TList<T>::insert\_before(const T& who, const T& where) {

TNode<T>\* pWhere = search(where);

if (pWhere == pFirst) {

insert\_first(who);

return;

}

TNode<T>\* pPrev = pFirst;

while (pPrev->pNext != pWhere) {

pPrev = pPrev->pNext;

}

TNode<T>\* new\_node = new TNode<T>(who, pWhere);

pPrev->pNext = new\_node;

}

template <typename T>

void TList<T>::insert\_after(const T& who, const T& where) {

TNode<T>\* pWhere = search(where);

if (pWhere == pLast) {

insert\_last(who);

return;

}

TNode<T>\* new\_node = new TNode<T>(who, pWhere->pNext);

pWhere->pNext = new\_node;

}

template <typename T>

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

if (pFirst == nullptr) throw "List is empty!";

TNode<T>\* tmp = pFirst;

TNode<T>\* pPrev = nullptr;

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

{

pPrev = tmp;

tmp = tmp->pNext;

}

if (tmp == pFirst){

pFirst = pFirst->pNext;

delete tmp;

return;

}

if (tmp == pStop)throw "Data not found!";

pPrev->pNext = tmp->pNext;

delete tmp;

}

template <typename T>

void TList<T>::reset() {

pCurr = pFirst;

}

template <typename T>

void TList<T>::next() {

if (pCurr == pStop) throw("Lis is ended");

pCurr = pCurr->pNext;

}

template<typename T>

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

if (pFirst == nullptr) return 0;

int size = 0;

TNode<T>\* tmp = pFirst;

while (tmp != pStop) {

size++;

tmp = tmp->pNext;

}

return size;

}

template<typename T>

TNode<T>\* TList<T>::GetCurrent()const {

return pCurr;

}

template <typename T>

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

if (IsEmpty() || data < pFirst->data) {

insert\_first(data);

return;

}

TNode<T>\* tmp = pFirst;

while (tmp->pNext != pStop && tmp->pNext->data <= data) {

tmp = tmp->pNext;

}

if (tmp->data == data) {

tmp->data = tmp->data + data;

return;

}

insert\_after(data, tmp->data);

}

template <typename T>

void TList<T>::sort() {

if (IsEmpty() || GetSize() == 1) {

return;

}

TNode<T>\* current = pFirst;

do {

bool swapped = false;

TNode<T>\* prev = nullptr;

TNode<T>\* next = current->pNext;

while (next != nullptr && next != pStop) {

if (current->data > next->data) {

swap(current->data, next->data);

swapped = true;

}

prev = current;

current = next;

next = next->pNext;

}

if (!swapped) {break;}

current = pFirst;

} while (1);

pCurr = pFirst;

}

## Приложение В. Реализация класса THeadRingList

template <typename T>

class THeadRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

THeadRingList();

THeadRingList(const THeadRingList& ringL);

virtual ~THeadRingList();

void insert\_first(const T& data);

void insert\_before(const T& who, const T& where);

};

template <typename T>

THeadRingList<T>::THeadRingList() : TList<T>() {

pHead = new TNode<T>();

pStop = pHead;

}

template <typename T>

THeadRingList<T>::THeadRingList(const THeadRingList<T>& ringL) : TList<T>(ringL) {

pHead = new TNode<T>(ringL.pHead->data, pFirst);

if (!ringL.IsEmpty()) {

pLast->pNext = pHead;

}

pStop = pHead;

}

template <typename T>

THeadRingList<T>::~THeadRingList() {

delete pHead;

}

template <typename T>

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

TList<T>::insert\_first(data);

pHead->pNext = pFirst;

pStop = pHead;

pLast->pNext = pHead;

}

template <typename T>

void THeadRingList<T>::insert\_before(const T& who, const T& where) {

TNode<T>\* pWhere = search(where);

if (pWhere == pFirst) {

insert\_first(who);

return;

}

TNode<T>\* pPrev = pFirst;

while (pPrev->pNext != pWhere) {

pPrev = pPrev->pNext;

}

TNode<T>\* new\_node = new TNode<T>(who, pWhere);

pPrev->pNext = new\_node;

}

## Приложение Г. Реализация класса TMonom

class TMonom {

public:

double coeff;

int degree;

TMonom();

TMonom(const TMonom& monom);

TMonom(double \_coeff, int \_degree);

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

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

bool operator <(const TMonom& data)const;

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

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

TMonom operator+(const TMonom& monom)const;

};

## Приложение Д. Реализация класса TPolynom

class TPolynom {

private:

string name;

THeadRingList<TMonom>\* monoms;

void ParseMonoms();

void conversion(string& str) const;

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const THeadRingList<TMonom>\* m);

TPolynom(const TPolynom& p);

~TPolynom();

TPolynom operator +(const TPolynom& p);

TPolynom operator -(const TPolynom& p);

TPolynom operator-() const;

TPolynom operator \*(const TPolynom& p);

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

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

TPolynom dx() const;

TPolynom dy() const;

TPolynom dz() const;

string ToString()const;

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

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

friend ostream& operator<<(ostream& out, const TPolynom& p);

};