МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
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**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**Аналитические преобразования полиномов от нескольких переменных (списки)**

Выполнил:

студент группы 3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ермолаев В.А.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc153310549)

[2. Постановка задачи 4](#_Toc153310550)

[3. Руководство пользователя 5](#_Toc153310551)

[4. Руководство программиста 7](#_Toc153310552)

[4.1. Описание структуры программы 7](#_Toc153310553)

[4.2. Описание структур данных 7](#_Toc153310554)

[4.2.1. TDatValue (абстрактный класс) 7](#_Toc153310555)

[4.2.2. TRootLink (шаблонный абстрактный класс) 7](#_Toc153310556)

[4.2.3. TDatLink (шаблонный класс, наследованный от TRootLink) 7](#_Toc153310557)

[4.2.4. TDatList 8](#_Toc153310558)

[4.2.5. THeadRing (шаблонный класс, наследованный от TDatList) 8](#_Toc153310559)

[4.2.6. TMonom (наследуется от TDatValue) 8](#_Toc153310560)

[4.2.7. TPolinom (наследуется от специализации THeadRing<TMonom>) 9](#_Toc153310561)

[4.2.8. experiments.cpp 9](#_Toc153310562)

[5. Эксперименты 10](#_Toc153310563)

[6. Заключение 12](#_Toc153310564)

[7. Литература 13](#_Toc153310565)

[8. Приложение 1 14](#_Toc153310566)

[9. Приложение 2 15](#_Toc153310567)

[10. Приложение 3 16](#_Toc153310568)

[11. Приложение 4 20](#_Toc153310569)

[12. Приложение 5 21](#_Toc153310570)

[13. Приложение 6 22](#_Toc153310571)

[14. Приложение 7 24](#_Toc153310572)

[15. Приложение 8 30](#_Toc153310573)

[16. Приложение 9 35](#_Toc153310574)

[17. Приложение 10 36](#_Toc153310575)

[18. Приложение 11 38](#_Toc153310576)

[19. Приложение 12 40](#_Toc153310577)

[20. Приложение 13 41](#_Toc153310578)

[21. Приложение 14 42](#_Toc153310579)

[22. Приложение 15 43](#_Toc153310580)

[23. Приложение 16 45](#_Toc153310581)

[24. Приложение 17 46](#_Toc153310582)

1. Введение

Аналитическая обработка данных является важным аспектом использования компьютеров в современном мире. Кроме вычислительных задач, компьютеры активно применяются для решения таких задач, как компьютерное доказательство теорем, логический вывод информации и анализ текстовых данных. Одним из таких примеров является задача обработки полиномов, представленных в общей аналитической форме. Полиномы - это хорошо изученная область математики, широко используемая в различных приложениях, включая аппроксимацию экспериментальных данных и построение функциональных зависимостей.

Цель данной лабораторной работы заключается в изучении методов компьютерной обработки полиномов. Для этого проводится анализ различных способов хранения полиномов и разрабатываются соответствующие программы обработки. Главной акцент в работе делается на практическом использовании методов организации структур данных с помощью списков. В процессе выполнения лабораторной работы разрабатывается универсальное представление линейных списков и программа для работы со списками, которая может быть применена и в других областях приложений.

1. Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение полиномов и выполнение основных операций над ними:

1. сложение/вычитание;
2. умножение
3. копирование/присваивание;
4. сравнение.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

Выполнение работы предполагает решение следующих задач:

1. Реализация односвязных списков:
   1. Реализация класса TDatValue;
   2. Реализация класса TRootLink;
   3. Реализация класса TDatLink;
   4. Реализация класса TDatList;
   5. Реализация класса THeadRing.
2. Реализация класса TMonom.
3. Реализация класса TPolinom.
4. Обеспечение работоспособности тестов и примера использования.
5. Реализация визуального интерфейса на основе Visual C++ WinForms.
6. Руководство пользователя

При запуске программы необходимо ввести два полинома в текстовые первого и второго блока соответствующе (также присутствуют полиномы по умолчанию). После чего выбрать операцию над введенными полиномами.

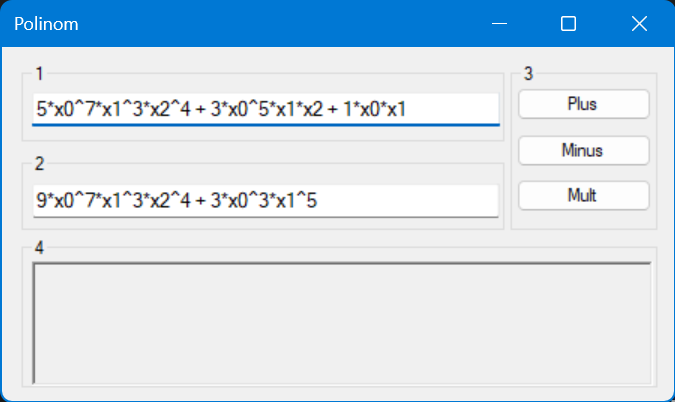


Рис. 1 – Исходное состояние

После выбора операции будет выведен результат.

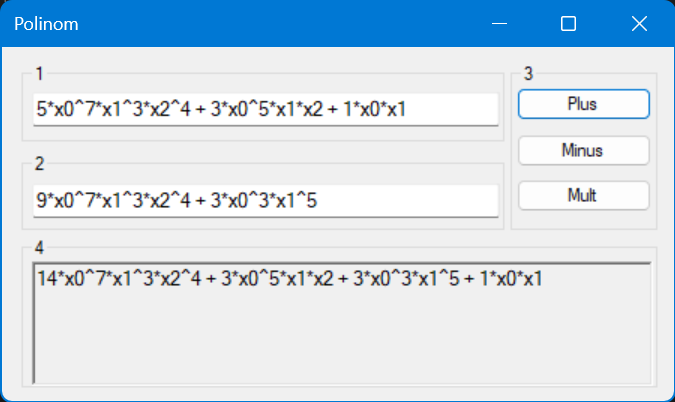


Рис. 2 –Сумма полиномов

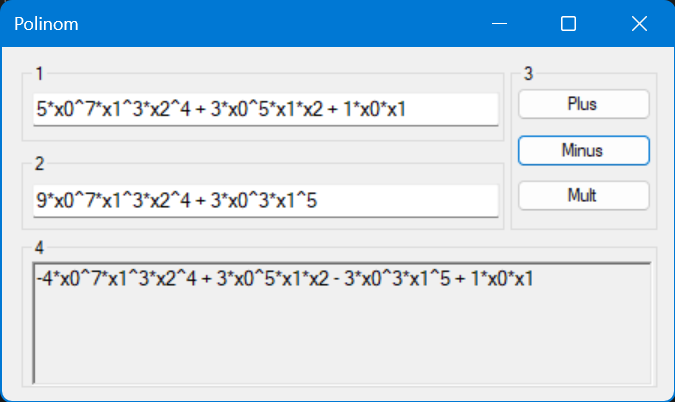


Рис. 3 – Вычитание полиномов

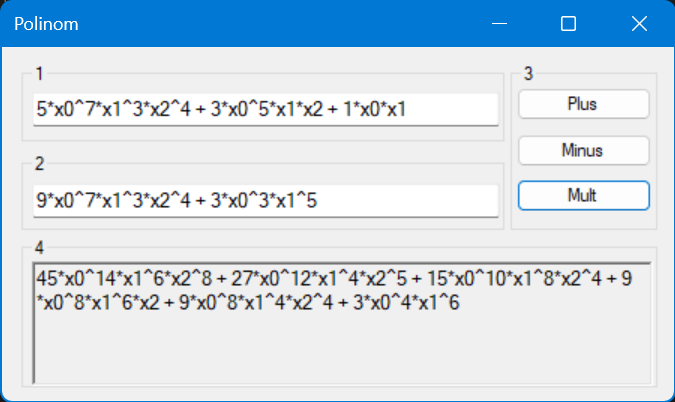


Рис. 4 – Умножение полиномов

В случае возникновения ошибки во время исполнения, будет вызвано окно-сообщения с ошибкой

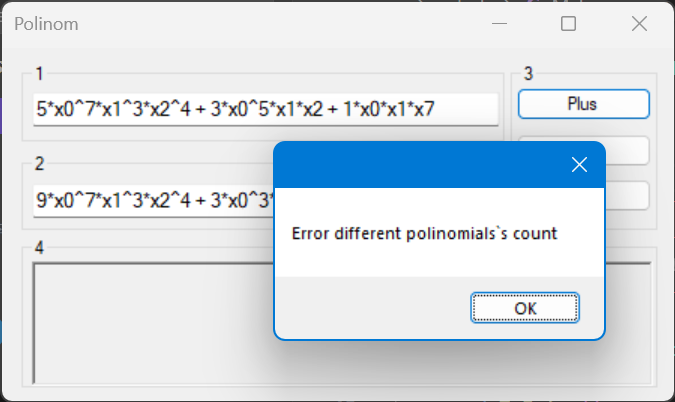


Рис. 5 – Сообщение об ошибке

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Polinom, содержащий реализацию классов TDatValue, TRootLink, TDatLink, TDalList, THeadRing, TMonom, TPolinom (файлы в директориях ./include/ и ./src/).
* Модуль Polinom\_test. Набор тестов для класса TPolinom. Включает в себя файл ./test/test\_polinom.cpp. Реализованы они с помощью использования фреймворка Google Test.
* Примеры использования классов (файлы в директории ./samples/).
* Графическое приложение Polinom\_gui (файлы в директории ./gui/)

## Описание структур данных

### TDatValue (абстрактный класс)

virtual TDatValue\* GetCopy(void) const = 0 – чисто виртуальный метод создания копии

~TDatValue(void) - деструктор

### TRootLink (шаблонный абстрактный класс)

protected:

TRootLink<T>\* pNext - указатель на следующее звено

public:

TRootLink(TRootLink<T>\* pN = NULL) - конструктор

TRootLink<T>\* GetNextLink(void) – получение следующей ссылки

void SetNextLink(TRootLink<T>\* pLink) – задать следующую ссылку

void InsNextLink(TRootLink<T>\* pLink) – вставить ссылку

virtual void SetDatValue(T\* pVal) = 0 – чисто виртуальный метод задания значения

virtual T\* GetDatValue(void) = 0 – чисто виртуальный метода получения значения

virtual TRootLink<T>\* GetCopy(TRootLink<T>\* next = NULL) = 0 – чисто виртуальный метод получения копии

friend class TDatList

### TDatLink (шаблонный класс, наследованный от TRootLink)

protected:

T\* pValue - указатель на объект значения

public:

TDatLink(T\* pVal = NULL, TRootLink<T>\* pN = NULL) :

TRootLink<T>(pN) – конструктор с делегацией

void SetDatValue(T\* pVal) override – задать ссылку на значение

T\* GetDatValue(void) override – получить ссылку на значение

TDatLink<T>\* GetNextDatLink(void) – получить ссылку на следующий элемент TDatLink

TDatLink<T>\* GetCopy(TRootLink<T>\* next = NULL) override – получить копию

friend class TDatList

}

### TDatList

protected:

TDatLink<T>\* pFirst - первое звено

TDatLink<T>\* pLast - последнее звено

TDatLink<T>\* pCurrLink - текущее звено

TDatLink<T>\* pPrevLink - звено перед текущим

TDatLink<T>\* pStop - значение указателя, означающего конец списка

int CurrPos - номер текущего звена (нумерация от 0)

int ListLen - количество звеньев в списке

void DelLink(const TDatLink<T>\* pLink) - удаление звена

public:

TDatList() - конструктор

TDatList(const TDatList<T>& q) – конструктор копирования

~TDatList(void) - деструктор

T\* GetDatValue(const int pos) const - получить значение

T\* GetDatValue(void) const – получить значение

virtual int IsEmpty(void) const – проверка списка на пустоту

int GetLength(void) const – получить длину списка

TDatLink<T>\* GetFirst(void) const – получить ссылку на первое звено

TDatLink<T>\* GetLast(void) const – получить ссылку на последнее звено

int SetCurrentPos(const int pos) - установить текущее звено

int GetCurrentPos(void) const - получить номер текущего звена

virtual int Reset(void) - установить на начало списка

virtual int IsListEnded(void) const - проверка на окончание списка

virtual int GoNext(void) - сдвиг вправо текущего звена

virtual void InsFirst(T\* pVal = NULL) – вставить звено перед первым

virtual void InsLast(T\* pVal = NULL) - вставить звено последним

virtual void InsCurrent(T\* pVal = NULL) – вставить звено перед текущим

virtual void DelFirst(void) - удалить первое звено

virtual void DelCurrent(void) - удалить текущее звено

virtual void DelList(void) - удалить весь список

TDatList<T>& operator=(const TDatList<T>& q) – оператор присваивания

### THeadRing (шаблонный класс, наследованный от TDatList)

protected:

TDatLink<T>\* pHead - заголовок, pFirst - звено за pHead

public:

THeadRing(void) - конструктор

THeadRing(const THeadRing& q) – конструктор копирования

~THeadRing(void) – деструктор

virtual void InsFirst(T\* pVal = NULL) - после заголовка

virtual void DelFirst(void) - удалить первое звено

int GoNext(void) override - сдвиг вправо текущего звена

void InsCurrent(T\* pVal) override – вставить звено перед текущим

void InsLast(T\* pVal) override - вставить звено последним

void DelList(void) override - удалить весь список

THeadRing& operator=(const THeadRing& q); – оператор присваивания

### TMonom (наследуется от TDatValue)

protected:

double Coeff - коэффициент монома

int CountDeg – количество переменных

int\* Deg - массив степеней

public:

TMonom(const double cval = 1, const int ival = 0, const int\* deg = NULL) – конструктор

TMonom(const TMonom& m) – конструктор копирования

~TMonom(void) - деструктор

virtual TMonom\* GetCopy(void) const override - изготовить копию

void SetCoeff(const double cval) – задать коэффициент

double GetCoeff(void) const – получить коэффициент

void SetCountDeg(const int ival) – задать количество переменных

int GetCountDeg(void) const – получить количество переменных

void SetDeg(const int\* deg) – задать переменные

int\* GetDeg(void) – получить переменные

std::string ToString(void) –получить моном в виде строки

TMonom operator+(const TMonom& mon) – оператор сложения

TMonom& operator+=(const TMonom& mon) – оператор сложения

TMonom operator-(const TMonom& mon) – оператор вычитания

TMonom& operator-=(const TMonom& mon) – оператор вычитания

TMonom operator\*(const TMonom& mon) – оператор умножения

TMonom& operator\*=(const TMonom& mon) - оператор умножения

TMonom& operator=(const TMonom& tm) - оператор присваивания

bool operator==(const TMonom& tm) const - оператор сравнения равенства

int operator<(const TMonom& mon) - оператор сравнения меньше

int operator<(const TMonom& mon) const - оператор сравнения меньше

int operator>(const TMonom& mon) – оператор сравнения больше

int operator>(const TMonom& mon) const - оператор сравнения больше

friend class TPolinom

friend std::istream& operator>>(std::istream& istr, TMonom& mon) – оператор ввода

friend std::ostream& operator<<(std::ostream& ostr, const TMonom& mon) – оператор вывода

### TPolinom (наследуется от специализации THeadRing<TMonom>)

protected:

int countVar - количество переменных в каждом мономе

public:

TPolinom(const int countVar = 3, const int\*\* monoms = NULL, const int km = 0) – конструктор полинома из массива «коэффициент-индекс»

TPolinom(const TPolinom& q) - конструктор копирования

const TMonom\* GetMonom(void) const – получить текущий моном

const TMonom\* GetMonom(int pos) const – получить моном по индексу

void AddMonom(const TMonom& nM) – добавить моном в полином

TPolinom MultMonom(const TMonom monom) – умножить полином на моном

TPolinom AddPolinom(const TPolinom& other) – прибавить полином

TPolinom operator+(const TPolinom& q) – оператор сложения

TPolinom& operator+=(const TPolinom& q) – оператор сложения

TPolinom operator-(const TPolinom& q) – оператор вычитания

TPolinom& operator-=(const TPolinom& q) – оператор вычитания

TPolinom operator\*(const double coef) const - оператор умножения

TPolinom operator\*(const double coef) – оператор умножения

TPolinom operator\*(const TMonom& coef) – оператор умножения

TPolinom operator\*(const TPolinom& other) – оператор умножения

TPolinom& operator=(const TPolinom& q) - оператор присваивания

bool operator==(const TPolinom& other) const – оператор сравнения равно

std::string ToString(void) – получить полином в виде строки

friend std::ostream& operator<<(std::ostream& ostr, TPolinom& q) – оператор вывода

### experiments.cpp

enum operations {

PLUS,

MINUS,

MULTIPLY,

} - перечисление констант для запуска тестов производительности

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 11
2. Процессор: Intel(R) Core™ i3-1115G4- CPU @ 3.00 GHz
3. Версия Visual Studio: 2022

В таблицах ниже под временем работы подразумевается усредненное время работы, основанное на 10 последовательных запусках на различных данных, полученных генерацией псевдослучайных чисел.

Таблица 1 – Измерение работы оператора сложения

|  |  |
| --- | --- |
| Количество переменных и размер полинома | Время работы оператора сложения (в млс) |
| 100 | 2.5 |
| 500 | 30.2 |
| 1 000 | 50.8 |
| 2 000 | 192.1 |
| 3 000 | 422.6 |
| 5 000 | 917.3 |

Рис. 6 – График времени работы оператора сложения

Сложность оператора сложения примерно соответствует сложности .

Таблица 2 – Измерение времени работы оператора умножения

|  |  |
| --- | --- |
| Количество переменных и размер полинома | Время работы оператора умножения (в млс) |
| 20 | 4.6 |
| 30 | 20.5 |
| 40 | 27.7 |
| 50 | 82.1 |
| 75 | 237.1 |
| 100 | 936.5 |
| 150 | 6 554.4 |
| 200 | 25 643.0 |

Рис. 7– График времени работы оператора умножения

Сложность оператора умножения примерно соответствует экспоненциальной сложности.

1. Заключение

В заключении можно отметить, что проведенный анализ различных способов хранения полиномов показал, что наиболее эффективным является хранение полиномов в виде односвязного списка. Этот способ позволяет легко выполнять операции над полиномами, такие как сложение, вычитание и умножение.

Таким образом, цель лабораторной работы достигнута. В ходе лабораторной работы изучены методы компьютерной обработки полиномов, и разработаны соответствующие программы обработки. Кроме того, в процессе выполнения лабораторной работы разработан универсальный способ представления линейных связных списков и программа для работы со списками, которая может быть применена и в других областях приложений.

В заключение можно также отметить, что полученные в ходе лабораторной работы результаты могут быть использованы для дальнейшего развития методов компьютерной обработки полиномов. Например, можно рассмотреть возможность разработки более эффективных алгоритмов обработки полиномов, а также разработки программ обработки полиномов, оптимизированных для работы на различных типах вычислительных устройств.

1. Литература
   * + 1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.
       2. Сысоев А.В. Полиномы / Сысоев А.В. [Электронный ресурс] // cloud.unn.ru : [сайт]. — URL: [https://cloud.unn.ru/s/x33MEa9on8HgNgw](https://cloud.unn.ru/s/x33MEa9on8HgNgw%20) (дата обращения: 10.12.2023).).
2. Приложение 1

DatLink.h

#ifndef \_POLINOM\_LIB\_TDATLINK\_

#define \_POLINOM\_LIB\_TDATLINK\_

#include <cstddef>

#include "RootLink.h"

#include "DatValue.h"

template<class T>

class TDatLink : public TRootLink<T>{

protected:

T\* pValue; // указатель на объект значения

public:

TDatLink(T\* pVal = NULL, TRootLink<T>\* pN = NULL) :

TRootLink<T>(pN) {

pValue = pVal;

}

void SetDatValue(T\* pVal) override { this->pValue = pVal; }

T\* GetDatValue(void) override { return this->pValue; }

TDatLink<T>\* GetNextDatLink(void) { return (TDatLink<T>\*)this->pNext; }

TDatLink<T>\* GetCopy(TRootLink<T>\* next = NULL) override {

if (this->pValue != NULL)

return new TDatLink<T>(new T(\*this->pValue), next);

return new TDatLink<T>(NULL, next);

}

template<class U>

friend class TDatList;

};

#endif

1. Приложение 2

DatList.h

#ifndef \_POLINOM\_LIB\_TDATLIST\_

#define \_POLINOM\_LIB\_TDATLIST\_

#include <exception>

#include "DatLink.h"

#include "RootLink.h"

template<class T>

class TDatList {

protected:

TDatLink<T>\* pFirst; // первое звено

TDatLink<T>\* pLast; // последнее звено

TDatLink<T>\* pCurrLink; // текущее звено

TDatLink<T>\* pPrevLink; // звено перед текущим

TDatLink<T>\* pStop; // значение указателя, означающего конец списка

int CurrPos; // номер текущего звена (нумерация от 0)

int ListLen; // количество звеньев в списке

protected: // методы

//TDatLink\* GetLink(T\* pVal = NULL, TDatLink\* pLink = NULL);

void DelLink(const TDatLink<T>\* pLink); // удаление звена

public:

TDatList();

TDatList(const TDatList<T>& q);

~TDatList(void) { DelList(); }

// доступ

T\* GetDatValue(const int pos) const; // значение

T\* GetDatValue(void) const { return GetDatValue(CurrPos); } // значение

virtual int IsEmpty(void) const { return pFirst == pStop; } // список пуст ?

int GetLength(void) const { return ListLen; } // к-во звеньев

TDatLink<T>\* GetFirst(void) const { return pFirst; }

TDatLink<T>\* GetLast(void) const { return pLast; }

// навигация

int SetCurrentPos(const int pos); // установить текущее звено

int GetCurrentPos(void) const { return CurrPos; } // получить номер тек. звена

virtual int Reset(void); // установить на начало списка

virtual int IsListEnded(void) const; // список завершен ?

virtual int GoNext(void); // сдвиг вправо текущего звена

// (=1 после применения GoNext для последнего звена списка)

// вставка звеньев

virtual void InsFirst(T\* pVal = NULL); // перед первым

virtual void InsLast(T\* pVal = NULL); // вставить последним

virtual void InsCurrent(T\* pVal = NULL); // перед текущим

// удаление звеньев

virtual void DelFirst(void); // удалить первое звено

virtual void DelCurrent(void); // удалить текущее звено

virtual void DelList(void); // удалить весь список

TDatList<T>& operator=(const TDatList<T>& q);

};

#include "../src/DatList.hpp"

#endif

1. Приложение 3

DatList.hpp

//TDatLink\* TDatList::GetLink(T\* pVal, TDatLink\* pLink)

//{

// return TDatLink\*();

//}

template<class T>

void TDatList<T>::DelLink(const TDatLink<T>\* pLink)

{

TDatLink<T>\* ptr = pFirst;

TDatLink<T>\* pre\_ptr = NULL;

while (ptr->pNext != pStop && ptr != pLink)

pre\_ptr = ptr;

ptr = ptr->GetNextDatLink();

if (ptr->pNext == pStop && ptr != pLink) {

throw std::exception("Element not found");

}

if (pre\_ptr != NULL) {

pre\_ptr->pNext = ptr->GetNextLink();

}

delete ptr;

ListLen--;

}

template<class T>

TDatList<T>::TDatList()

{

pFirst = NULL;

pLast = NULL;

pPrevLink = NULL;

pCurrLink = NULL;

pStop = new TDatLink<T>();

CurrPos = NULL;

ListLen = 0;

}

template<class T>

TDatList<T>::TDatList(const TDatList& q)

{

ListLen = q.ListLen;

pFirst = q.pFirst->GetCopy(q.pFirst->GetNextDatLink());

pStop = q.pStop->GetCopy();

pCurrLink = pFirst;

for (size\_t i = 0; i < ListLen - 1; i++) {

pPrevLink = pCurrLink;

pCurrLink = pCurrLink->GetNextDatLink();

pCurrLink = pCurrLink->GetCopy(pCurrLink->GetNextDatLink());

pPrevLink->SetNextLink(pCurrLink);

}

pLast = pCurrLink;

pLast->SetNextLink(pStop);

SetCurrentPos(q.CurrPos);

}

template<class T>

T\* TDatList<T>::GetDatValue(int pos) const

{

if (pos >= ListLen)

return NULL;

if (pos == CurrPos)

return pCurrLink->GetDatValue();

TDatLink<T>\* ptr = pFirst;

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

ptr = ptr->GetNextDatLink();

return ptr->GetDatValue();

}

template<class T>

int TDatList<T>::SetCurrentPos(int pos)

{

if (pos < 0 || pos >= ListLen)

throw std::exception("Wrong index");

pCurrLink = pFirst;

pPrevLink = NULL;

CurrPos = 0;

while (CurrPos < pos) {

pPrevLink = pCurrLink;

pCurrLink = pCurrLink->GetNextDatLink();

CurrPos++;

}

return CurrPos;

}

template<class T>

int TDatList<T>::Reset(void)

{

pCurrLink = pFirst;

pPrevLink = NULL;

CurrPos = 0;

return CurrPos;

}

template<class T>

int TDatList<T>::IsListEnded(void) const

{

return pCurrLink->pNext == pStop;

}

template<class T>

int TDatList<T>::GoNext(void)

{

if (IsListEnded()) {

throw std::exception("List is ended");

}

pPrevLink = pCurrLink;

pCurrLink = pCurrLink->GetNextDatLink();

CurrPos = ++CurrPos % ListLen;

return CurrPos;

}

template<class T>

void TDatList<T>::InsFirst(T\* pVal)

{

if (pFirst == NULL && ListLen == 0) {

pFirst = new TDatLink<T>(pVal, pStop);

pLast = pCurrLink = pFirst;

}

else {

TDatLink<T>\* tmp = pFirst;

pFirst = new TDatLink<T>(pVal);

pFirst->pNext = tmp;

SetCurrentPos(CurrPos);

}

ListLen++;

}

template<class T>

void TDatList<T>::InsLast(T\* pVal)

{

TDatLink<T>\* tmp = new TDatLink<T>(pVal, pStop);

if (pFirst == NULL && ListLen == 0) {

pFirst = tmp;

pLast = pCurrLink = pFirst;

}

else {

pLast->InsNextLink(tmp);

pLast = tmp;

}

ListLen++;

}

template<class T>

void TDatList<T>::InsCurrent(T\* pVal)

{

TDatLink<T>\* tmp = new TDatLink<T>(pVal);

if (pFirst == NULL && ListLen == 0) {

pFirst = new TDatLink<T>(pVal, pStop);

pLast = pCurrLink = pFirst;

}

else {

if (pCurrLink->pNext == pStop) {

pLast = tmp;

}

pCurrLink->InsNextLink(tmp);

}

ListLen++;

}

template<class T>

void TDatList<T>::DelFirst(void)

{

if (pFirst != NULL) {

TDatLink<T>\* prev = pFirst;

pFirst = (TDatLink<T>\*)pFirst->GetNextLink();

if (pCurrLink == prev)

pCurrLink = pFirst;

delete prev;

prev = NULL;

ListLen--;

}

}

template<class T>

void TDatList<T>::DelCurrent(void)

{

if (pCurrLink != NULL) {

TDatLink<T>\* prev = pCurrLink;

pFirst = (TDatLink<T>\*)pCurrLink->GetNextLink();

delete prev;

ListLen--;

}

}

template<class T>

void TDatList<T>::DelList(void)

{

TDatLink<T>\* prev = pFirst;

while (ListLen > 0) {

DelFirst();

}

if (pStop != NULL)

{

delete pStop;

pStop = NULL;

}

ListLen = 0;

}

template<class T>

TDatList<T>& TDatList<T>::operator=(const TDatList& q)

{

if (this == &q) {

return \*this;

}

if (pFirst != NULL)

DelList();

if (q.pFirst == NULL)

return \*this;

ListLen = q.ListLen;

pFirst = q.pFirst->GetCopy(q.pFirst->GetNextDatLink());

pStop = q.pStop->GetCopy();

pCurrLink = pFirst;

for (size\_t i = 0; i < ListLen - 1; i++) {

pPrevLink = pCurrLink;

pCurrLink = pCurrLink->GetNextDatLink();

pCurrLink = pCurrLink->GetCopy(pCurrLink->GetNextDatLink());

pPrevLink->SetNextLink(pCurrLink);

}

pLast = pCurrLink;

pLast->SetNextLink(pStop);

SetCurrentPos(q.CurrPos);

return \*this;

}

1. Приложение 4

DatValue.h

#ifndef \_POLINOM\_LIB\_TDATVALUE\_

#define \_POLINOM\_LIB\_TDATVALUE\_

class TDatValue {

public:

virtual TDatValue\* GetCopy(void) const = 0; // создание копии

~TDatValue(void) {}

};

#endif

1. Приложение 5

HeadRing.h

#ifndef \_POLINOM\_LIB\_THEADRING\_

#define \_POLINOM\_LIB\_THEADRING\_

#include "DatList.h"

template<class T>

class THeadRing : public TDatList<T> {

protected:

TDatLink<T>\* pHead; // заголовок, pFirst - звено за pHead

public:

THeadRing(void);

THeadRing(const THeadRing& q);

~THeadRing(void);

// вставка звеньев

virtual void InsFirst(T\* pVal = NULL); // после заголовка

// удаление звеньев

virtual void DelFirst(void); // удалить первое звено

int GoNext(void) override;

void InsCurrent(T\* pVal) override;

void InsLast(T\* pVal) override;

void DelList(void) override;

THeadRing& operator=(const THeadRing& q);

};

#include "../src/HeadRing.hpp"

#endif

1. Приложение 6

HeadRing.hpp

template<class T>

THeadRing<T>::THeadRing()

{

pHead = new TDatLink<T>();

}

template<class T>

THeadRing<T>::THeadRing(const THeadRing<T>& q) : TDatList<T>(q)

{

this->pHead = q.pHead->GetCopy();

this->pHead->SetNextLink(this->pFirst);

this->pLast->SetNextLink(this->pHead);

}

template<class T>

THeadRing<T>::~THeadRing()

{

delete this->pHead;

}

template<class T>

void THeadRing<T>::InsFirst(T\* pVal)

{

TDatList<T>::InsFirst(pVal);

this->pHead->SetNextLink(this->pFirst);

this->pLast->SetNextLink(this->pHead);

}

template<class T>

void THeadRing<T>::DelFirst(void)

{

TDatList<T>::DelFirst();

this->pHead->SetNextLink(this->pFirst);

this->pLast->SetNextLink(this->pHead);

}

template<class T>

int THeadRing<T>::GoNext(void)

{

if (this->pCurrLink->GetNextDatLink() == this->pHead) {

this->pCurrLink = this->pHead;

}

return TDatList<T>::GoNext();

}

template<class T>

void THeadRing<T>::InsCurrent(T\* pVal)

{

if (this->pCurrLink != NULL && this->pCurrLink->GetNextDatLink() == pHead) {

this->pCurrLink->InsNextLink(new TDatLink<T>(pVal));

this->ListLen++;

}

else

TDatList<T>::InsCurrent(pVal);

if (this->ListLen == 1)

this->pHead->SetNextLink(this->pFirst);

}

template<class T>

void THeadRing<T>::InsLast(T\* pVal)

{

TDatList<T>::InsLast(pVal);

this->pLast->SetNextLink(this->pHead);

if (this->ListLen == 1)

this->pHead->SetNextLink(this->pFirst);

}

template<class T>

void THeadRing<T>::DelList(void)

{

TDatList<T>::DelList();

if (pHead != NULL) {

delete this->pHead;

this->pHead = NULL;

}

}

template<class T>

THeadRing<T>& THeadRing<T>::operator=(const THeadRing<T>& q)

{

DelList();

TDatList<T>::operator=(q);

this->pHead = q.pHead->GetCopy();

if (q.pFirst != NULL) {

this->pHead->SetNextLink(this->pFirst);

this->pLast->SetNextLink(this->pHead);

}

return \*this;

}

1. Приложение 7

Monom.h

#ifndef \_POLINOM\_LIB\_TMONOM\_

#define \_POLINOM\_LIB\_TMONOM\_

#include <iostream>

#include <string>

#include "DatValue.h"

class TMonom : public TDatValue {

protected:

double Coeff; // коэффициент монома

int CountDeg; // индекс (свертка степеней)

int\* Deg;

public:

TMonom(const double cval = 1, const int ival = 0, const int\* deg = NULL);

TMonom(const TMonom& m);

~TMonom(void);

virtual TMonom\* GetCopy(void) const

{

return new TMonom(\*this);

} // изготовить копию

void SetCoeff(const double cval) { Coeff = cval; }

double GetCoeff(void) const { return Coeff; }

void SetCountDeg(const int ival);

int GetCountDeg(void) const { return CountDeg; }

void SetDeg(const int\* deg);

int\* GetDeg(void) { return Deg; }

std::string ToString(void);

TMonom operator+(const TMonom& mon);

TMonom& operator+=(const TMonom& mon);

TMonom operator-(const TMonom& mon);

TMonom& operator-=(const TMonom& mon);

TMonom operator\*(const TMonom& mon);

TMonom& operator\*=(const TMonom& mon);

TMonom& operator=(const TMonom& tm);

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

int operator<(const TMonom& mon);

int operator<(const TMonom& mon) const;

int operator>(const TMonom& mon);

int operator>(const TMonom& mon) const;

friend class TPolinom;

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

friend std::ostream& operator<<(std::ostream& ostr, const TMonom& mon);

};

TMonom::TMonom(const double cval, const int ival, const int\* deg)

{

Coeff = cval; CountDeg = ival;

if (deg != NULL) {

Deg = new int[ival];

std::copy(deg, deg + ival, Deg);

}

else {

Deg = NULL;

}

}

TMonom::TMonom(const TMonom& m)

{

this->Coeff = m.Coeff;

this->CountDeg = m.CountDeg;

this->Deg = new int[this->CountDeg];

std::copy(m.Deg, m.Deg + m.CountDeg, this->Deg);

}

TMonom::~TMonom()

{

if (Deg != nullptr)

delete[] Deg;

CountDeg = 0;

Coeff = 0;

}

void TMonom::SetCountDeg(int \_count)

{

if (\_count < 0)

throw std::exception("Error monom`s count");

else if (\_count == 0)

{

if (Deg != 0)

delete[] Deg;

Deg = 0;

CountDeg = 0;

}

else if (\_count == CountDeg) {

return;

}

else

{

int\* tmp = new int[CountDeg];

std::copy(Deg, Deg + CountDeg, tmp);

if (Deg != NULL)

delete[] Deg;

Deg = new int[\_count] {};

int i = 0;

if (CountDeg >= \_count)

std::copy(tmp, tmp + \_count, Deg);

else

std::copy(tmp, tmp + CountDeg, Deg);

CountDeg = \_count;

if (tmp != NULL)

delete[] tmp;

}

}

void TMonom::SetDeg(const int\* \_deg)

{

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

{

if (\_deg[i] >= 0)

Deg[i] = \_deg[i];

else

throw std::exception("Error degree");

}

}

TMonom TMonom::operator+(const TMonom& mon)

{

TMonom tmp(mon);

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

for (int i = 0; i < mon.CountDeg; i++)

if (Deg[i] != mon.Deg[i])

throw std::exception("Error different degree");

tmp.Coeff = Coeff + mon.Coeff;

return tmp;

}

TMonom& TMonom::operator+=(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

for (int i = 0; i < mon.CountDeg; i++)

if (Deg[i] != mon.Deg[i])

throw std::exception("Error different degree");

Coeff += mon.Coeff;

return \*this;

}

TMonom TMonom::operator-(const TMonom& mon)

{

TMonom tmp(mon);

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

for (int i = 0; i < mon.CountDeg; i++)

if (Deg[i] != mon.Deg[i])

throw std::exception("Error different degree");

tmp.Coeff = Coeff - mon.Coeff;

return tmp;

}

TMonom& TMonom::operator-=(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

for (int i = 0; i < mon.CountDeg; i++)

if (Deg[i] != mon.Deg[i])

throw std::exception("Error different degree");

Coeff -= mon.Coeff;

return \*this;

}

TMonom TMonom::operator\*(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

TMonom tmp(mon);

tmp.Coeff = Coeff \* mon.Coeff;

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

tmp.Deg[i] = Deg[i] + mon.Deg[i];

return tmp;

}

TMonom& TMonom::operator\*=(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw std::exception("Error different monoms`s count");

Coeff = Coeff \* mon.Coeff;

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

Deg[i] += mon.Deg[i];

return \*this;

}

std::istream& operator>>(std::istream& istr, TMonom& mon)

{

istr >> mon.Coeff;

istr >> mon.CountDeg;

for (int i = 0; i < mon.CountDeg; i++)

istr >> mon.Deg[i];

return istr;

}

std::ostream& operator<<(std::ostream& ostr, const TMonom& mon)

{

double coef = mon.GetCoeff();

if (coef == (int)coef)

ostr << std::to\_string((int)coef);

else

ostr << std::to\_string(round(coef \* 100) / 100);

if (coef != 0)

{

for (int i = 0; i < mon.CountDeg; i++)

if (mon.Deg[i] != 0)

{

ostr << "\*x" + std::to\_string(i);

if (mon.Deg[i] != 1)

ostr << "^" + std::to\_string(mon.Deg[i]);

}

}

return ostr;

}

TMonom& TMonom::operator=(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw std::exception("Error leng not equal");

Coeff = mon.Coeff;

if (CountDeg != mon.CountDeg) {

CountDeg = mon.CountDeg;

if (Deg == NULL)

delete[] Deg;

Deg = new int[CountDeg];

}

std::copy(mon.Deg, mon.Deg + mon.CountDeg, Deg);

return \*this;

}

int TMonom::operator<(const TMonom& mon)

{

if (CountDeg != mon.CountDeg)

throw false;

if (\*this == mon)

return false;

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

{

if (Deg[i] == mon.Deg[i])

continue;

else if (Deg[i] < mon.Deg[i])

return true;

else

return false;

}

if (Coeff >= mon.Coeff)

return false;

return true;

}

int TMonom::operator<(const TMonom& mon) const

{

if (\*this == NULL)

return true;

if (mon == NULL)

return false;

if (CountDeg != mon.CountDeg)

throw false;

if (\*this == mon)

return false;

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

{

if (Deg[i] == mon.Deg[i])

continue;

else if (Deg[i] < mon.Deg[i])

return true;

else

return false;

}

if (Coeff > mon.Coeff)

return false;

return true;

}

std::string TMonom::ToString()

{

std::string res;

double coef = GetCoeff();

if (coef == (int)coef)

res += std::to\_string((int)coef);

else

res += std::to\_string(round(coef \* 100000) / 100000);

if (coef != 0)

{

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

if (Deg[i] != 0)

{

res += "\*x" + std::to\_string(i);

if (Deg[i] != 1)

res += "^" + std::to\_string(Deg[i]);

}

}

return res;

}

int TMonom::operator>(const TMonom& mon) {

return !(\*this < mon);

}

int TMonom::operator>(const TMonom& mon) const {

return !(\*this < mon);

}

bool TMonom::operator==(const TMonom& mon) const

{

if (Coeff != mon.Coeff)

return false;

if (CountDeg != mon.CountDeg)

return false;

if (!std::equal(Deg, Deg + CountDeg, mon.Deg))

return false;

return true;

}

#endif

1. Приложение 8

Polinom.h

#ifndef \_POLINOM\_LIB\_TPOLIMOM\_

#define \_POLINOM\_LIB\_TPOLINOM\_

#include "HeadRing.h"

#include "Monom.h"

class TPolinom : public THeadRing<TMonom> {

protected:

int countVar; //количество переменных в каждом мономе

public:

TPolinom(const int countVar = 3, const int\*\* monoms = NULL, const int km = 0); // конструктор

// полинома из массива «коэффициент-индекс»

TPolinom(const TPolinom& q); // конструктор копирования

const TMonom\* GetMonom(void) const { return GetDatValue(); }

const TMonom\* GetMonom(int pos) const { return GetDatValue(pos); }

void AddMonom(const TMonom& nM);

TPolinom MultMonom(const TMonom monom);

TPolinom AddPolinom(const TPolinom& other);

TPolinom operator+(const TPolinom& q);

TPolinom& operator+=(const TPolinom& q);

TPolinom operator-(const TPolinom& q);

TPolinom& operator-=(const TPolinom& q);

TPolinom operator\*(const double coef) const;

TPolinom operator\*(const double coef);

TPolinom operator\*(const TMonom& coef);

TPolinom operator\*(const TPolinom& other);

TPolinom& operator=(const TPolinom& q); // присваивание

bool operator==(const TPolinom& other) const;

std::string ToString(void);

friend std::ostream& operator<<(std::ostream& ostr, TPolinom& q);

};

TPolinom::TPolinom(const int cVar, const int\*\* monoms, const int len)

{

countVar = cVar;

if (monoms != NULL && len != 0) {

for (size\_t i = 0; i < len; i++) {

InsCurrent(new TMonom(monoms[i][0], countVar, monoms[i] + 1));

}

}

}

TPolinom::TPolinom(const TPolinom& q) : THeadRing(q)

{

countVar = q.countVar;

}

TPolinom TPolinom::operator\*(const double coef) const

{

TPolinom res(countVar);

TDatLink<TMonom>\* tmp1 = pFirst;

while (tmp1 != pHead)

{

TMonom k = \*tmp1->GetDatValue();

k.SetCoeff(k.GetCoeff() \* coef);

res.AddMonom(k);

tmp1 = tmp1->GetNextDatLink();

}

return res;

}

TPolinom TPolinom::operator\*(const double coef)

{

TPolinom res(countVar);

TDatLink<TMonom>\* tmp1 = pFirst;

while (tmp1 != pHead)

{

TMonom k = \*tmp1->GetDatValue();

k.SetCoeff(k.GetCoeff() \* coef);

res.AddMonom(k);

tmp1 = tmp1->GetNextDatLink();

}

return res;

}

TPolinom TPolinom::operator\*(const TMonom& coef)

{

if (this->countVar != coef.CountDeg)

throw std::exception("Error different polinomial`s count");

TPolinom res(countVar);

TDatLink<TMonom>\* tmp1 = pFirst;

while (tmp1 != pHead)

{

TMonom k = (\*tmp1->GetDatValue()) \* coef;

res.AddMonom(k);

tmp1 = tmp1->GetNextDatLink();

}

return res;

}

inline TPolinom TPolinom::operator\*(const TPolinom& other)

{

if (this->countVar != other.countVar)

throw std::exception("Error different polinomial`s count");

TPolinom res(countVar);

TDatLink<TMonom>\* tmp1 = pFirst;

TDatLink<TMonom>\* tmp2 = other.pFirst;

while (tmp1 != pHead)

{

while (tmp2 != other.pHead)

{

TMonom k = (\*tmp1->GetDatValue()) \* (\*tmp2->GetDatValue());

res.AddMonom(k);

tmp2 = tmp2->GetNextDatLink();

}

tmp1 = tmp1->GetNextDatLink();

tmp2 = other.pFirst;

}

return res;

}

TPolinom& TPolinom::operator=(const TPolinom& q)

{

if (this == &q)

return \*this;

else

{

DelList();

countVar = q.countVar;

THeadRing::operator=(q);

return \*this;

}

}

bool TPolinom::operator==(const TPolinom& other) const

{

if (this->countVar != other.countVar)

return false;

if (this->ListLen != other.ListLen)

return false;

TDatLink<TMonom>\* tmp1 = pFirst;

TDatLink<TMonom>\* tmp2 = other.pFirst;

for (size\_t i = 0; i < ListLen; i++)

{

if (!(\*tmp1->GetDatValue() == (\*tmp2->GetDatValue())))

return false;

tmp1 = tmp1->GetNextDatLink();

tmp2 = tmp2->GetNextDatLink();

}

return true;

}

void TPolinom::AddMonom(const TMonom& nM)

{

if (this->countVar != nM.GetCountDeg())

throw std::exception("Error different polinomial`s count");

if (nM.GetCoeff() == 0)

return;

if (pFirst == NULL)

InsFirst(nM.GetCopy());

else if (std::equal(nM.Deg, nM.Deg + nM.CountDeg,

pFirst->GetDatValue()->Deg))

{

pFirst->GetDatValue()->operator+=(nM);

if (pFirst->GetDatValue()->GetCoeff() == 0)

DelFirst();

}

else {

int i = 0;

while (ListLen > 0 && i <= ListLen && \*GetMonom() > nM &&

!(std::equal(nM.Deg, nM.Deg + nM.CountDeg, GetMonom()->Deg)))

{

i++;

if (i == ListLen) {

InsLast(nM.GetCopy());

return;

}

GoNext();

}

if (std::equal(nM.Deg, nM.Deg + nM.CountDeg,

GetMonom()->Deg)) {

pCurrLink->GetDatValue()->operator+=(nM);

if (pCurrLink->GetDatValue()->GetCoeff() == 0)

DelCurrent();

}

else {

if (GetCurrentPos() == 0) {

InsFirst(nM.GetCopy());

}

else if (GetCurrentPos() > 0) {

SetCurrentPos(GetCurrentPos() - 1);

InsCurrent(nM.GetCopy());

}

}

}

}

TPolinom TPolinom::MultMonom(const TMonom monom)

{

return \*this \* monom;

}

TPolinom TPolinom::AddPolinom(const TPolinom& other)

{

return \*this + other;

}

std::string TPolinom::ToString()

{

std::string res = "";

TDatLink<TMonom>\* tmp = pFirst;

if (ListLen == 0) {

return "0";

}

if (tmp != NULL)

{

if (tmp->GetDatValue() == NULL) {

res += "0";

}

else {

res += ((TMonom\*)tmp->GetDatValue())->ToString();

}

tmp = tmp->GetNextDatLink();

}

while (tmp != NULL && tmp != pHead)

{

if (((TMonom\*)tmp->GetDatValue())->GetCoeff() > 0) {

res += " + " + ((TMonom\*)tmp->GetDatValue())->ToString();

}

else if (((TMonom\*)tmp->GetDatValue())->GetCoeff() < 0) {

tmp->GetDatValue()->SetCoeff(-tmp->GetDatValue()->GetCoeff());

res += " - " + ((TMonom\*)tmp->GetDatValue())->ToString();

tmp->GetDatValue()->SetCoeff(-tmp->GetDatValue()->GetCoeff());

}

tmp = tmp->GetNextDatLink();

}

return res;

}

TPolinom TPolinom::operator+(const TPolinom& q)

{

if (countVar != q.countVar)

throw std::exception("Error different polinomials`s count");

if (\*this == q \* -1.0) {

return TPolinom(countVar);

}

TPolinom res(\*this);

TDatLink<TMonom>\* tmp2 = q.pFirst;

for (size\_t i = 0; i < q.ListLen; i++)

{

res.AddMonom(\*tmp2->GetDatValue());

tmp2 = tmp2->GetNextDatLink();

}

return res;

}

TPolinom& TPolinom::operator+=(const TPolinom& q)

{

\*this = \*this + q;

return \*this;

}

TPolinom TPolinom::operator-(const TPolinom& q)

{

if (countVar != q.countVar)

throw std::exception("Error different polinomials`s count");

TPolinom res(\*this);

TDatLink<TMonom>\* tmp2 = q.pFirst;

for (size\_t i = 0; i < q.ListLen; i++)

{

TMonom tmp = \*tmp2->GetDatValue();

tmp.SetCoeff(-tmp.GetCoeff());

res.AddMonom(tmp);

tmp2 = tmp2->GetNextDatLink();

}

return res;

}

TPolinom& TPolinom::operator-=(const TPolinom& q)

{

\*this = \*this - q;

return \*this;

}

std::ostream& operator<<(std::ostream& ostr, TPolinom& q)

{

return ostr << q.ToString();

}

#endif

1. Приложение 9

RootLink.h

#ifndef \_POLINOM\_LIB\_TROOTLINK\_

#define \_POLINOM\_LIB\_TROOTLINK\_

#include "DatValue.h"

#include "DatList.h"

template<class T>

class TRootLink {

protected:

TRootLink<T>\* pNext; // указатель на следующее звено

public:

TRootLink(TRootLink<T>\* pN = NULL) { pNext = pN; }

TRootLink<T>\* GetNextLink(void) { return pNext; }

void SetNextLink(TRootLink<T>\* pLink) { pNext = pLink; }

void InsNextLink(TRootLink<T>\* pLink) {

TRootLink\* p = pNext; pNext = pLink;

if (pLink != NULL) pLink->pNext = p;

}

virtual void SetDatValue(T\* pVal) = 0;

virtual T\* GetDatValue(void) = 0;

virtual TRootLink<T>\* GetCopy(TRootLink<T>\* next = NULL) = 0;

template<class U>

friend class TDatList;

};

#endif#

1. Приложение 10

Polinom\_experiments.cpp

#include <iostream>

#include <chrono>

#include <random>

#include "../include/Polinom.h"

enum operations {

PLUS,

MINUS,

MULTIPLY,

DIVIDE,

COPY,

COMPARE

};

auto average\_test(size\_t size, operations op, size\_t iterations = 10) {

int max\_random = 10000;

int min\_random = -10000;

long long average\_time = 0;

for (size\_t i = 0; i < iterations; i++) {

TPolinom param1(size), param2(size);

for (size\_t i1 = 0; i1 < size; i1++) {

double c1 = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

double c2 = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

std::vector<int> deg;

for (size\_t j1 = 0; j1 < size; j1++) {

deg.push\_back(std::rand() % static\_cast<int>(max\_random - min\_random + 1));

}

param1.AddMonom(TMonom(c1, (int)size, °[0]));

param2.AddMonom(TMonom(c2, (int)size, °[0]));

}

std::chrono::steady\_clock::time\_point begin, end;

switch (op)

{

case PLUS:

begin = std::chrono::steady\_clock::now();

param1 + param2;

end = std::chrono::steady\_clock::now();

break;

case MINUS:

begin = std::chrono::steady\_clock::now();

param1 - param2;

end = std::chrono::steady\_clock::now();

break;

case COPY:

begin = std::chrono::steady\_clock::now();

param1 = param2;

end = std::chrono::steady\_clock::now();

break;

case COMPARE:

begin = std::chrono::steady\_clock::now();

param1 == param2;

end = std::chrono::steady\_clock::now();

break;

case MULTIPLY:

begin = std::chrono::steady\_clock::now();

param1 \* param2;

end = std::chrono::steady\_clock::now();

break;

default:

break;

}

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

average\_time += elapsed\_ms.count();

std::cout << i + 1 << '\t' << elapsed\_ms.count() << std::endl;

}

std::cout << "Average time: " << average\_time / static\_cast<double>(iterations) << std::endl;

return average\_time / static\_cast<double>(iterations);

}

int main(int argc, char\*\* arhv)

{

auto begin = std::chrono::steady\_clock::now();

average\_test(150, MULTIPLY);

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "Execution time: " << elapsed\_ms.count();

}

1. Приложение 11

Polinom\_sample\_parse.cpp

#include <vector>

#include <iostream>

#include <map>

#include <set>

#include <regex>

#include "Polinom.h"

std::vector<std::string> find\_str(std::string s, std::string del) {

std::vector<std::string> res;

int end = s.find(del);

while (end != -1) {

res.push\_back(s.substr(0, end));

s.erase(s.begin(), s.begin() + end + del.size());

end = s.find(del);

}

res.push\_back(s.substr(0, end));

return res;

}

void parsePol(TPolinom& pol1, TPolinom& pol2) {

std::vector<std::string> s\_inp;

s\_inp.push\_back("5\*x0^7\*x1^3\*x2^4 + 3\*x0^5\*x1\*x2 + 1\*x0\*x1");

//s\_inp.push\_back("9\*x0^7\*x1^3\*x2^4 + 3\*x0^3\*x1^5");

s\_inp.push\_back("-5\*x0^7\*x1^3\*x2^4 - 3\*x0^5\*x1\*x2 - 1\*x0\*x1");

for (auto it = s\_inp.begin(); it < s\_inp.end(); it++) {

std::set<std::string> params;

std::vector<std::pair<double, std::map<std::string, int>>> monoms;

std::string str = std::regex\_replace(\*it, std::regex("\\ \*"), "");

str = std::regex\_replace(str, std::regex("\\-"), "+-");

auto v\_monoms = find\_str(str, "+");

for (auto monom : v\_monoms) {

if (monom != "") {

auto comp = find\_str(monom, "\*");

std::pair<double, std::map<std::string, int>> p\_monom;

p\_monom.first = std::stod(comp.at(0));

for (auto it = ++comp.begin(); it < comp.end(); it++) {

std::string str = \*it;

int deg;

auto tmp = find\_str(str, "^");

if (tmp.size() == 1)

deg = 1;

else

deg = std::stoi(tmp.at(1));

params.insert(tmp.at(0));

p\_monom.second[tmp.at(0)] = deg;

}

monoms.push\_back(p\_monom);

}

}

TPolinom tmp(params.size());

for (auto pr : monoms) {

for (auto param : params) {

if (pr.second.find(param) == pr.second.end()) {

pr.second[param] = 0;

}

}

std::vector<int> deg;

for (auto it = pr.second.begin(); it != pr.second.end(); ++it) {

deg.push\_back(it->second);

}

tmp.AddMonom(TMonom(pr.first, params.size(), °[0]));

}

if (it == s\_inp.begin())

pol1 = tmp;

else

pol2 = tmp;

}

}

int main() {

TPolinom pol1, pol2;

parsePol(pol1, pol2);

std::cout << pol1 << std::endl << pol2 << std::endl;

std::cout << (pol1 + pol2).ToString();

}

1. Приложение 12

Polinom\_sample\_tdatlist.cpp

#include <iostream>

#include "DatList.h"

#include "Monom.h"

int main(int argc, char\*\* argv) {

TDatList<TMonom> list;

list.InsFirst(new TMonom());

list.InsCurrent(new TMonom());

list.GoNext();

}

1. Приложение 13

Polinom\_sample\_theadring.cpp

#include <iostream>

#include "HeadRing.h"

#include "Monom.h"

int main(int argc, char\*\* argv) {

THeadRing<TMonom> list;

list.InsFirst(new TMonom());

list.InsFirst(new TMonom());

list.InsCurrent(new TMonom());

list.GoNext();

list.GoNext();

list.GoNext();

list.DelFirst();

list.GoNext();

list.GoNext();

list.GoNext();

}

1. Приложение 14

test\_main.cpp

#include <gtest.h>

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

1. Приложение 15

test\_polinom.cpp

#include <gtest.h>

#include "Polinom.h"

using namespace std;

TEST(TPolinom, test1)

{

TPolinom P, Q, res;

int deg1[] = { 5, 2, 5 };

int deg2[] = { 4, 3, 3 };

int deg3[] = { 3, 5, 1 };

int deg4[] = { 3, 2, 6 };

int deg5[] = { 2, 1, 8 };

P.AddMonom(TMonom(3, 3, deg1));

P.AddMonom(TMonom(-5, 3, deg2));

P.AddMonom(TMonom(7, 3, deg3));

Q.AddMonom(TMonom(4, 3, deg4));

Q.AddMonom(TMonom(-6, 3, deg5));

res.AddMonom(TMonom(3, 3, deg1));

res.AddMonom(TMonom(-5, 3, deg2));

res.AddMonom(TMonom(7, 3, deg3));

res.AddMonom(TMonom(4, 3, deg4));

res.AddMonom(TMonom(-6, 3, deg5));

ASSERT\_EQ(res, P + Q);

}

TEST(TPolinom, test2)

{

TPolinom P, Q, res;

int deg1[] = {5, 2, 5};

int deg2[] = {4, 3, 3};

int deg3[] = {3, 5, 1};

int deg4[] = {7, 2, 6};

int deg5[] = {6, 1, 8};

P.AddMonom(TMonom(3, 3, deg1));

P.AddMonom(TMonom(-5, 3, deg2));

P.AddMonom(TMonom(7, 3, deg3));

Q.AddMonom(TMonom(4, 3, deg4));

Q.AddMonom(TMonom(-6, 3, deg5));

res.AddMonom(TMonom(3, 3, deg1));

res.AddMonom(TMonom(-5, 3, deg2));

res.AddMonom(TMonom(7, 3, deg3));

res.AddMonom(TMonom(4, 3, deg4));

res.AddMonom(TMonom(-6, 3, deg5));

ASSERT\_EQ(res, P + Q);

}

TEST(TPolinom, test3)

{

TPolinom P, Q, res;

int deg1[] = {5, 2, 5};

int deg2[] = {4, 3, 3};

int deg3[] = {3, 5, 1};

P.AddMonom(TMonom(3, 3, deg1));

P.AddMonom(TMonom(-5, 3, deg2));

P.AddMonom(TMonom(7, 3, deg3));

Q.AddMonom(TMonom(4, 3, deg1));

Q.AddMonom(TMonom(-6, 3, deg2));

res.AddMonom(TMonom(3, 3, deg1));

res.AddMonom(TMonom(-5, 3, deg2));

res.AddMonom(TMonom(7, 3, deg3));

res.AddMonom(TMonom(4, 3, deg1));

res.AddMonom(TMonom(-6, 3, deg2));

ASSERT\_EQ(res, P + Q);

}

TEST(TPolinom, test4)

{

TPolinom P, Q, res;

int deg1[] = {5, 2, 5};

int deg2[] = {4, 3, 3};

int deg3[] = {3, 5, 1};

int deg4[] = {6, 2, 6};

int deg5[] = {2, 1, 8};

P.AddMonom(TMonom(3, 3, deg1));

P.AddMonom(TMonom(-5, 3, deg2));

P.AddMonom(TMonom(7, 3, deg3));

Q.AddMonom(TMonom(4, 3, deg1));

Q.AddMonom(TMonom(-6, 3, deg2));

res.AddMonom(TMonom(3, 3, deg1));

res.AddMonom(TMonom(-5, 3, deg2));

res.AddMonom(TMonom(7, 3, deg3));

res.AddMonom(TMonom(4, 3, deg1));

res.AddMonom(TMonom(-6, 3, deg2));

ASSERT\_EQ(res, P + Q);

}

TEST(TPolinom, test5)

{

TPolinom P, Q, res;

int deg1[] = {5, 2, 5};

int deg2[] = {4, 3, 3};

int deg3[] = {7, 5, 1};

P.AddMonom(TMonom(3, 3, deg1));

P.AddMonom(TMonom(-5, 3, deg2));

P.AddMonom(TMonom(7, 3, deg3));

Q = P \* -1.0;

ASSERT\_EQ(res, P + Q);

}

1. Приложение 16

MyForm.cpp

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CppWinForm1::MyForm form;

Application::Run(% form);

}

1. Приложение 17

MyForm.h

#pragma once

#include <msclr\marshal\_cppstd.h>

#include <regex>

#include <map>

#include <set>

#include "Polinom.h"

namespace CppWinForm1 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

}

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::GroupBox^ groupBox1;

private: System::Windows::Forms::GroupBox^ groupBox2;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::GroupBox^ groupBox3;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::GroupBox^ groupBox4;

private: System::Windows::Forms::RichTextBox^ richTextBox1;

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());

this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());

this->button3 = (gcnew System::Windows::Forms::Button());

this->button2 = (gcnew System::Windows::Forms::Button());

this->button1 = (gcnew System::Windows::Forms::Button());

this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());

this->richTextBox1 = (gcnew System::Windows::Forms::RichTextBox());

this->groupBox1->SuspendLayout();

this->groupBox2->SuspendLayout();

this->groupBox3->SuspendLayout();

this->groupBox4->SuspendLayout();

this->SuspendLayout();

//

// textBox1

//

this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->textBox1->Location = System::Drawing::Point(11, 29);

this->textBox1->Name = L"textBox1";

this->textBox1->Size = System::Drawing::Size(466, 30);

this->textBox1->TabIndex = 1;

this->textBox1->Text = L"5\*x0^7\*x1^3\*x2^4 + 3\*x0^5\*x1\*x2 + 1\*x0\*x1";

//

// groupBox1

//

this->groupBox1->Controls->Add(this->textBox1);

this->groupBox1->Location = System::Drawing::Point(20, 17);

this->groupBox1->Name = L"groupBox1";

this->groupBox1->Size = System::Drawing::Size(483, 81);

this->groupBox1->TabIndex = 4;

this->groupBox1->TabStop = false;

this->groupBox1->Text = L"1";

//

// groupBox2

//

this->groupBox2->Controls->Add(this->textBox2);

this->groupBox2->Location = System::Drawing::Point(20, 109);

this->groupBox2->Name = L"groupBox2";

this->groupBox2->Size = System::Drawing::Size(483, 80);

this->groupBox2->TabIndex = 5;

this->groupBox2->TabStop = false;

this->groupBox2->Text = L"2";

//

// textBox2

//

this->textBox2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->textBox2->Location = System::Drawing::Point(11, 31);

this->textBox2->Name = L"textBox2";

this->textBox2->Size = System::Drawing::Size(466, 30);

this->textBox2->TabIndex = 1;

this->textBox2->Text = L"9\*x0^7\*x1^3\*x2^4 + 3\*x0^3\*x1^5";

//

// groupBox3

//

this->groupBox3->Controls->Add(this->button3);

this->groupBox3->Controls->Add(this->button2);

this->groupBox3->Controls->Add(this->button1);

this->groupBox3->Location = System::Drawing::Point(509, 17);

this->groupBox3->Name = L"groupBox3";

this->groupBox3->Size = System::Drawing::Size(147, 172);

this->groupBox3->TabIndex = 6;

this->groupBox3->TabStop = false;

this->groupBox3->Text = L"3";

//

// button3

//

this->button3->Location = System::Drawing::Point(6, 119);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(135, 34);

this->button3->TabIndex = 2;

this->button3->Text = L"Mult";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// button2

//

this->button2->Location = System::Drawing::Point(6, 73);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(135, 34);

this->button2->TabIndex = 1;

this->button2->Text = L"Minus";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// button1

//

this->button1->Location = System::Drawing::Point(6, 25);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(135, 34);

this->button1->TabIndex = 0;

this->button1->Text = L"Plus";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// groupBox4

//

this->groupBox4->Controls->Add(this->richTextBox1);

this->groupBox4->Location = System::Drawing::Point(20, 195);

this->groupBox4->Name = L"groupBox4";

this->groupBox4->Size = System::Drawing::Size(636, 156);

this->groupBox4->TabIndex = 7;

this->groupBox4->TabStop = false;

this->groupBox4->Text = L"4";

//

// richTextBox1

//

this->richTextBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->richTextBox1->Location = System::Drawing::Point(11, 25);

this->richTextBox1->Name = L"richTextBox1";

this->richTextBox1->ReadOnly = true;

this->richTextBox1->Size = System::Drawing::Size(619, 125);

this->richTextBox1->TabIndex = 0;

this->richTextBox1->Text = L"";

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(672, 363);

this->Controls->Add(this->groupBox4);

this->Controls->Add(this->groupBox3);

this->Controls->Add(this->groupBox2);

this->Controls->Add(this->groupBox1);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;

this->Name = L"MyForm";

this->Text = L"Polinom";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

this->groupBox1->ResumeLayout(false);

this->groupBox1->PerformLayout();

this->groupBox2->ResumeLayout(false);

this->groupBox2->PerformLayout();

this->groupBox3->ResumeLayout(false);

this->groupBox4->ResumeLayout(false);

this->ResumeLayout(false);

}

#pragma endregion

private: std::vector<std::string> find\_str(std::string s, std::string del) {

std::vector<std::string> res;

int end = s.find(del);

while (end != -1) {

res.push\_back(s.substr(0, end));

s.erase(s.begin(), s.begin() + end + del.size());

end = s.find(del);

}

res.push\_back(s.substr(0, end));

return res;

}

private: void parsePol(TPolinom& pol1, TPolinom& pol2) {

msclr::interop::marshal\_context context;

std::vector<std::string> s\_inp;

s\_inp.push\_back(context.marshal\_as<std::string>(textBox1->Text));

s\_inp.push\_back(context.marshal\_as<std::string>(textBox2->Text));

for (auto it = s\_inp.begin(); it < s\_inp.end(); it++) {

std::set<std::string> params;

std::vector<std::pair<double, std::map<std::string, int>>> monoms;

std::string str = std::regex\_replace(\*it, std::regex("\\ \*"), "");

str = std::regex\_replace(str, std::regex("\\-"), "+-");

auto v\_monoms = find\_str(str, "+");

for (auto monom : v\_monoms) {

if (monom != "") {

auto comp = find\_str(monom, "\*");

std::pair<double, std::map<std::string, int>> p\_monom;

p\_monom.first = std::stod(comp.at(0));

for (auto it = ++comp.begin(); it < comp.end(); it++) {

std::string str = \*it;

int deg;

auto tmp = find\_str(str, "^");

if (tmp.size() == 1)

deg = 1;

else

deg = std::stoi(tmp.at(1));

params.insert(tmp.at(0));

p\_monom.second[tmp.at(0)] = deg;

}

monoms.push\_back(p\_monom);

}

}

TPolinom tmp(params.size());

for (auto pr : monoms) {

for (auto param : params) {

if (pr.second.find(param) == pr.second.end()) {

pr.second[param] = 0;

}

}

std::vector<int> deg;

for (auto it = pr.second.begin(); it != pr.second.end(); ++it) {

deg.push\_back(it->second);

}

tmp.AddMonom(TMonom(pr.first, params.size(), °[0]));

}

if (it == s\_inp.begin())

pol1 = tmp;

else

pol2 = tmp;

}

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

TPolinom pol1, pol2, res;

parsePol(pol1, pol2);

res = pol1 + pol2;

richTextBox1->Text = gcnew String(res.ToString().c\_str());

}

catch (std::exception e) {

MessageBox::Show(gcnew String(e.what()));

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

TPolinom pol1, pol2, res;

parsePol(pol1, pol2);

res = pol1 - pol2;

richTextBox1->Text = gcnew String(res.ToString().c\_str());

}

catch (std::exception e) {

MessageBox::Show(gcnew String(e.what()));

}

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

TPolinom pol1, pol2, res;

parsePol(pol1, pol2);

res = pol1 \* pol2;

richTextBox1->Text = gcnew String(res.ToString().c\_str());

}

catch (std::exception e) {

MessageBox::Show(gcnew String(e.what()));

}

}

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

};

}