Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

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

Выполнил:

студент ф-та ПРИН гр. 381908-4

Бутеску.В.В

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г

Нижний Новгород

2021 г.

Содержание

[Введение 3](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962758)

[Постановка задачи 4](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962759)

[Руководство программиста 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962761)

[Описание структур данных 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962762)

[Описание алгоритмов 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962763)

[Описание структуры программы 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962764)

[Заключение 7](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962765)

[Литература 8](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962766)

[Приложения 9](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962767)

# Введение

Наряду с привычным вычислительным применением компьютеры широко используются и для аналитической обработки данных. Среди примеров таких приложений – компьютерное доказательство теорем, логический вывод, анализ текстовой информации и многое другое. Среди таких примеров и задача обработки полиномов, задаваемых в общей аналитической форме. Полиномы являются хорошо изученной областью математики (алгебра полиномов), которая широко используется в приложениях (аппроксимация экспериментальных данных, построение функциональных зависимостей и т.п.).

Лабораторная работа направлена на изучение методов компьютерной обработки полиномов. С этой целью в лабораторной работе изучаются различные варианты структуры хранения и разрабатываются программы для обработки полиномов. Основной учебной целью работы является практическое освоение методов организации структур хранения данных с помощью списков. В ходе выполнения лабораторной работы разрабатывается общая форма представления линейных списков, разрабатываются программы работы со списками, которые могут быть использованы и в других областях приложений.

# Постановка задачи

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

• ввод полинома

• организация хранения полинома

• удаление введенного ранее полинома;

• копирование полинома;

• сложение двух полиномов;

• вычисление значения полинома при заданных значениях переменных;

• вывод.

Состав реализуемых операций над полиномами может быть расширен при постановке задания лабораторной работы. Предполагается, что в качестве структуры хранения будут использоваться списки. В качестве дополнительной цели в лабораторной работе ставится также задача разработки некоторого общего представления списков и операций по их обработке. В числе операций над списками должны быть реализованы следующие действия:

• поддержка понятия текущего звена;

• вставка звеньев в начало, после текущей позиции и в конец списков;

• удаление звеньев в начале и в текущей позиции списков;

• организация последовательного доступа к звеньям списка (итератор).

В ходе выполнения лабораторной работы должно быть выполнено сопоставление разработанных средств работы со списками с возможностями работы со списками в библиотеке STL. Важной частью лабораторной работы должна являться разработка диалоговой управляющей программы с наглядным визуальным интерфейсом, который обеспечивает возможности создавать полиномы, выполнять реализованные операции обработки полиномов, демонстрировать вид имеющихся полиномов.

Для работы со списками предлагается реализовать следующие операции:

• методы получения параметров состояния списка (проверка на пустоту, получение текущего количества звеньев);

• метод доступа к значению первого, текущего или последнего звена; • методы навигации по списку (итератор);

• методы вставки перед первым, после текущего и последнего звеньев;

• методы удаления первого и текущего звена.

Состав операций может быть расширен при разработке спецификации класса для списков. Расширение списковой структуры хранения для представления полиномов (звенозаголовок, цикличная структура списка) должно разрабатываться как производный класс от базового класса поддержки списков.

**Руководство пользователя**

Пользователю нужно запустить файл main.exe.

Откроется консольное приложение для тестирования очереди.

Программа заполнит очередь числами и выведет их в консоль, так же выведет копирования очереди

Для повторного выполнения потребуется перезапустить программу.

# Руководство программиста

## **Описание структуры программы**

Реализует операции:

template <class T>

class TListElem

{

protected:

T data;

TListElem\* next = 0;

TListElem\* prev = 0;

public:

TListElem();

TListElem(T \_data);

~TListElem();

T GetData();

TListElem\* GetNext();

TListElem\* GetPrev();

void SetData(T \_data);

void SetNext(TListElem\* \_next);

void SetPrev(TListElem\* \_prev);

virtual TListElem<T>\* Clone();

template <class T1>

friend ostream& operator<< (ostream& ostr, const TListElem<T1>& A);

template <class T1>

friend istream& operator>> (istream& istr, TListElem<T1> A);

};

template <class T>

class TList

{

protected:

TListElem<T>\* root = 0;

TListElem<T>\* end = 0;

int count;

public:

TList();

TList(TList<T>& \_v);

~TList();

virtual TList<T>& operator =(TList<T>& \_v);

void InsFirst(T d);

void InsLast(T d);

void Ins(TListElem<T>\* e, T d);

bool IsFull(void) const;

bool IsEmpty(void) const;

TListElem<T>\* GetFirst();

TListElem<T>\* GetLast();

void DelFirst();

void DelLast();

void Del(TListElem<T>\* e);

int GetCount();

template <class T1>

friend ostream& operator<< (ostream& ostr, const TList<T1>& A);

template <class T1>

friend istream& operator>> (istream& istr, TList<T1>& A);

};

class TMonomData

{

public:

double\* data;

double K;

int dim;

TMonomData(int t);

TMonomData(double\* \_data = 0, int \_dim = 0, double \_K = 0);

TMonomData(TMonomData& \_v);

~TMonomData();

};

class TPolinomial : public TList<TMonomData>

{

public:

TPolinomial();

TPolinomial(TPolinomial& \_v);

TPolinomial& operator+= (TMonom& \_v);

TPolinomial operator\* (TPolinomial& \_v);

TPolinomial operator+ (TPolinomial& \_v);

TPolinomial operator- (TPolinomial& \_v);

};

# 

# Заключение

Этап 1. Реализация программ поддержки очереди.

Этап 2. Реализация системы имитации обслуживания заданий.

Этап 3. Выполнение дополнительных заданий лабораторной работы.

# Литература

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
9. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
10. Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.

# Приложения

TList.h

#ifndef \_LIST\_

#define \_LIST\_

#include <iostream>

using namespace std;

template <class T>

class TListElem

{

protected:

T data;

TListElem\* next = 0;

TListElem\* prev = 0;

public:

TListElem();

TListElem(T \_data);

~TListElem();

T GetData();

TListElem\* GetNext();

TListElem\* GetPrev();

void SetData(T \_data);

void SetNext(TListElem\* \_next);

void SetPrev(TListElem\* \_prev);

virtual TListElem<T>\* Clone();

template <class T1>

friend ostream& operator<< (ostream& ostr, const TListElem<T1>& A);

template <class T1>

friend istream& operator>> (istream& istr, TListElem<T1> A);

};

template <class T>

class TList

{

protected:

TListElem<T>\* root = 0;

TListElem<T>\* end = 0;

int count;

public:

TList();

TList(TList<T>& \_v);

~TList();

virtual TList<T>& operator =(TList<T>& \_v);

void InsFirst(T d);

void InsLast(T d);

void Ins(TListElem<T>\* e, T d);

bool IsFull(void) const;

bool IsEmpty(void) const;

TListElem<T>\* GetFirst();

TListElem<T>\* GetLast();

void DelFirst();

void DelLast();

void Del(TListElem<T>\* e);

int GetCount();

template <class T1>

friend ostream& operator<< (ostream& ostr, const TList<T1>& A);

template <class T1>

friend istream& operator>> (istream& istr, TList<T1>& A);

};

template<class T1>

inline ostream& operator<<(ostream& ostr, const TListElem<T1>& A)

{

return ostr << A.data;

}

template<class T1>

inline istream& operator>>(istream& istr, TListElem<T1> A)

{

return istr >> A.data;

}

template <class T1>

ostream& operator<< (ostream& ostr, const TList<T1>& A)

{

TListElem<T1>\* i = A.root;

while (i != 0)

{

ostr << \*i << endl;

i = i->GetNext();

}

return ostr;

}

template <class T1>

istream& operator >> (istream& istr, TList<T1>& A) {

int count;

istr >> count;

for (int i = 0; i < A.count; i++) {

T1 d;

istr >> d;

A.InsLast(d);

}

return istr;

}

template<class T>

inline TList<T>::TList()

{

root = 0;

end = 0;

count = 0;

}

template <class T>

TList<T>::TList(TList<T>& \_v)

{

count = \_v.count;

TListElem<T>\* i = \_v.root;

TListElem<T>\* j = this->root;

TListElem<T>\* p = 0;

root = 0;

end = 0;

while (i != 0)

{

j = i->Clone();

j->SetNext(0);

if (p != 0)

{

p->SetNext(j);

j->SetPrev(p);

}

p = j;

if (root == 0)

{

root = j;

}

end = j;

i = i->GetNext();

}

}

template <class T>

TList<T>::~TList()

{

if (this->root != 0)

{

TListElem<T>\* i = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

p = i;

i = i->GetNext();

delete p;

}

this->root = 0;

this->end = 0;

count = 0;

}

}

template <class T>

TList<T>& TList<T>::operator =(TList<T>& \_v)

{

if (this == &\_v)

return \*this;

if (this->root != 0)

{

TListElem<T>\* i = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

p = i;

i = i->GetNext();

delete p;

}

this->root = 0;

this->end = 0;

count = 0;

}

count = \_v.count;

TListElem<T>\* i = \_v.root;

TListElem<T>\* j = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

j = i->Clone();

j->SetNext(0);

if (p != 0)

{

p->SetNext(j);

j->SetPrev(p);

}

p = j;

if (root == 0)

{

root = j;

}

end = j;

i = i->GetNext();

}

return \*this;

}

template<class T>

inline void TList<T>::InsFirst(T d)

{

if (this->IsFull())

{

throw "Error";

}

TListElem<T>\* temp = d.Clone();

temp->SetNext(root);

root = temp;

if (end == 0)

{

end = temp;

}

count++;

}

template<class T>

inline void TList<T>::InsLast(T d)

{

if (this->IsFull())

{

throw "Error";

}

if (end == 0)

{

end = d.Clone();

}

else

{

TListElem<T>\* temp = d.Clone();

end->SetNext(temp);

end = temp;

}

if (root == 0)

{

root = end;

}

count++;

}

template<class T>

inline void TList<T>::Ins(TListElem<T>\* e, T d)

{

TListElem<T>\* temp = d.Clone();

temp->SetNext(e->GetNext());

temp->SetPrev(e);

e->GetNext()->SetPrev(temp);

e->SetNext(temp);

count++;

}

template<class T>

inline bool TList<T>::IsFull(void) const

{

try

{

TListElem<T>\* temp = new TListElem<T>(1);

delete temp;

return false;

}

catch (...)

{

return true;

}

}

template<class T>

inline bool TList<T>::IsEmpty(void) const

{

return count == 0;

}

template<class T>

inline TListElem<T>\* TList<T>::GetFirst()

{

return root;

}

template<class T>

inline TListElem<T>\* TList<T>::GetLast()

{

return end;

}

template<class T>

inline void TList<T>::DelFirst()

{

TListElem<T>\* i = this->root;

root = root->GetNext();

count--;

delete i;

}

template<class T>

inline void TList<T>::DelLast()

{

TListElem<T>\* i = this->end;

end = root->GetPrev();

count--;

delete i;

}

template<class T>

inline void TList<T>::Del(TListElem<T>\* e)

{

e->GetPrev()->SetNext(e->GetNext());

e->GetNext()->SetPrev(e->GetPrev());

count--;

delete e;

}

template<class T>

inline int TList<T>::GetCount()

{

return count;

}

template<class T>

inline TListElem<T>::TListElem()

{

data = 0;

next = 0;

prev = 0;

}

template<class T>

inline TListElem<T>::TListElem(T \_data)

{

data = \_data;

next = 0;

prev = 0;

}

template<class T>

inline TListElem<T>::~TListElem()

{

next = 0;

prev = 0;

}

template<class T>

inline T TListElem<T>::GetData()

{

return data;

}

template<class T>

inline TListElem<T>\* TListElem<T>::GetNext()

{

return next;

}

template<class T>

inline TListElem<T>\* TListElem<T>::GetPrev()

{

return prev;

}

template<class T>

inline void TListElem<T>::SetData(T \_data)

{

data = \_data;

}

template<class T>

inline void TListElem<T>::SetNext(TListElem\* \_next)

{

next = \_next;

}

template<class T>

inline void TListElem<T>::SetPrev(TListElem\* \_prev)

{

prev = \_prev;

}

template<class T>

inline TListElem<T>\* TListElem<T>::Clone()

{

TListElem<T>\* res = new TListElem<T>(\*this);

return res;

}

#endif

TPolinom.h

#include "Polinom.h"

TPolinomial::TPolinomial() : TList<TMonomData>()

{

}

TPolinomial::TPolinomial(TPolinomial& \_v) : TList<TMonomData>(\_v)

{

}

TPolinomial& TPolinomial::operator+=(TMonom& \_v)

{

if (this->root == 0)

{

this->root = \_v.Clone();

this->end = this->root;

}

else

{

TMonom\* temp = static\_cast<TMonom\*>(this->root);

while (temp != 0)

{

if (\*temp == \_v)

{

temp->SetK(temp->GetK() +\_v.GetK());

}

else

{

TMonom\* temp2 = \_v.Clone();

if (\*temp < \_v)

{

temp2->SetPrev(temp->GetPrev());

temp2->SetNext(temp);

if (temp->GetPrev() != 0)

{

temp->GetPrev()->SetNext(temp2);

}

else

{

this->root = temp2;

}

temp->SetPrev(temp2);

return \*this;

}

else

{

if (temp->GetNext() == 0)

{

temp2->SetPrev(temp);

temp->SetNext(temp2);

this->end = temp2;

return \*this;

}

else

{

temp = static\_cast<TMonom\*>(temp->GetNext());

}

}

}

}

}

return \*this;

}

TPolinomial TPolinomial::operator\*(TPolinomial& \_v)

{

TPolinomial res;

TMonom\* temp1 = static\_cast<TMonom\*>(root);

if (root == 0 && \_v.root)

{

throw logic\_error ("Error");

}

while (temp1 != 0)

{

TMonom\* temp2 = static\_cast<TMonom\*>(\_v.root);

while (temp2 != 0)

{

TMonom temp3;

temp3 = (static\_cast<TMonom&>(\*temp2)) \* (static\_cast<TMonom&>(\*temp1));

res += temp3;

temp2 = static\_cast<TMonom\*>(temp2->GetNext());

}

temp1 = static\_cast<TMonom\*>(temp1->GetNext());

}

return res;

}

TPolinomial TPolinomial::operator+(TPolinomial& \_v)

{

if (\_v.root == 0 || root == 0)

{

throw logic\_error("Error");

}

TPolinomial res;

TMonom\* temp = static\_cast<TMonom\*>(\_v.root);

TMonom\* temp2 = static\_cast<TMonom\*>(root);

TMonom\* r = 0;

while (temp != 0 && temp2 != 0)

{

TMonom temp3;

if (\*temp == \*temp2)

{

temp3 = ((\*temp) + (\*temp2))[0];

temp = static\_cast<TMonom\*>(temp->GetNext());

temp2 = static\_cast<TMonom\*>(temp2->GetNext());

}

else if (\*temp < \*temp2)

{

temp3 = \*temp2;

temp2 = static\_cast<TMonom\*>(temp2->GetNext());

}

else

{

temp3 = \*temp;

temp = static\_cast<TMonom\*>(temp->GetNext());

}

if (r == 0)

{

res += temp3;

r = static\_cast<TMonom\*>(res.root);

}

else

{

r->SetNext(temp3.Clone());

r->GetNext()->SetPrev(static\_cast<TMonom\*>(r));

r = static\_cast<TMonom\*>(r->GetNext());

}

}

if (temp != 0)

{

while (temp != 0)

{

r->SetNext(new TMonom(\*temp));

r->GetNext()->SetPrev(r);

r = static\_cast<TMonom\*>(r->GetNext());

temp = static\_cast<TMonom\*>(temp->GetNext());

}

}

else if (temp2 != 0)

{

while (temp2 != 0)

{

r->SetNext(new TMonom(\*temp2));

r->GetNext()->SetPrev(r);

r = static\_cast<TMonom\*>(r->GetNext());

temp = static\_cast<TMonom\*>(temp2->GetNext());

}

}

return res;

}

TPolinomial TPolinomial::operator-(TPolinomial& \_v)

{

{

if (\_v.root == 0 || root == 0)

{

throw logic\_error("Error");

}

TPolinomial res;

TMonom\* temp = static\_cast<TMonom\*>(\_v.root);

TMonom\* temp2 = static\_cast<TMonom\*>(root);

TMonom\* t = 0;

while (temp != 0 && temp2 != 0)

{

TMonom temp3;

if (\*temp == \*temp2)

{

temp3 = ((\*temp) - (\*temp2))[0];

temp = static\_cast<TMonom\*>(temp->GetNext());

temp2 = static\_cast<TMonom\*>(temp2->GetNext());

}

else if (\*temp > \*temp2)

{

temp3 = \*temp;

temp = static\_cast<TMonom\*>(temp->GetNext());

}

else

{

temp3 = \*temp2;

temp2 = static\_cast<TMonom\*>(temp2->GetNext());

}

if (t == 0)

{

res += temp3;

t = static\_cast<TMonom\*>(res.root);

}

else

{

t->SetNext(temp3.Clone());

t->GetNext()->SetPrev(static\_cast<TMonom\*>(t));

t = static\_cast<TMonom\*>(t->GetNext());

}

}

if (temp != 0)

{

while (temp != 0)

{

t->SetNext(new TMonom(\*temp));

t->GetNext()->SetPrev(t);

t = static\_cast<TMonom\*>(t->GetNext());

temp = static\_cast<TMonom\*>(temp->GetNext());

}

}

else if (temp2 != 0)

{

while (temp2 != 0)

{

t->SetNext(new TMonom(\*temp2));

t->GetNext()->SetPrev(t);

t = static\_cast<TMonom\*>(t->GetNext());

temp = static\_cast<TMonom\*>(temp2->GetNext());

}

}

return res;

}

}

Tmonom.h

#ifndef \_TMONOM\_

#define \_TMONOM\_

#include "TList.h"

class TMonomData

{

public:

double\* data;

double K;

int dim;

TMonomData(int t);

TMonomData(double\* \_data = 0, int \_dim = 0, double \_K = 0);

TMonomData(TMonomData& \_v);

~TMonomData();

};

class TMonom : public TListElem<TMonomData>

{

protected:

public:

TMonom(TMonomData \_data);

TMonom(double\* \_data = 0, int \_dim = 0, double \_K = 0);

TMonom(TMonom& \_v);

bool operator == (const TMonom& \_v);

bool operator != (const TMonom& \_v);

TMonomData& operator \*();

TMonomData& operator \*(int);

double GetK();

void SetK(double \_K);

int GetDim();

void SetDim(int \_dim);

virtual TMonom\* Clone();

double& operator[] (int i);

TMonom& operator= (const TMonom& \_v);

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

TMonom operator/ (const TMonom& \_v);

TMonom\* operator+ (const TMonom& \_v);

TMonom\* operator- (const TMonom& \_v);

bool operator> (const TMonom& \_v);

bool operator< (const TMonom& \_v);

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

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

};

TMonom::TMonom(TMonomData \_data) : TListElem<TMonomData>(\_data)

{

}

TMonom::TMonom(double\* \_data, int \_dim, double \_K) : TListElem<TMonomData>()

{

if (\_dim < 0)

{

throw logic\_error("Error");

}

data.dim = \_dim;

data.K = \_K;

if (data.dim > 0)

{

data.data = new double[data.dim];

for (int i = 0; i < data.dim; i++)

{

data.data[i] = \_data[i];

}

}

}

TMonom::TMonom(TMonom& \_v) : TListElem<TMonomData>(\_v)

{

prev = 0;

next = 0;

}

bool TMonom::operator==(const TMonom& \_v)

{

bool res = true;

if (data.dim != \_v.data.dim)

{

res = false;

}

if (data.dim > 0)

{

for (int i = 0; i < data.dim; i++)

{

if (data.data[i] != \_v.data.data[i])

{

res = false;

break;

}

}

return res;

}

else

{

return data.K == \_v.data.K;

}

}

bool TMonom::operator!=(const TMonom& \_v)

{

return !this->operator==(\_v);

}

TMonomData& TMonom::operator\*()

{

return data;

}

TMonomData& TMonom::operator\*(int)

{

return data;

}

double TMonom::GetK()

{

return data.K;

}

void TMonom::SetK(double \_K)

{

data.K = \_K;

}

int TMonom::GetDim()

{

return data.dim;

}

void TMonom::SetDim(int \_dim)

{

if (data.dim == \_dim)

{

return;

}

if (\_dim < 0)

{

throw logic\_error("Error");

}

else if (\_dim == 0)

{

if (data.dim > 0)

{

delete[] data.data;

}

data.dim = 0;

data.data = 0;

}

else

{

if (data.dim > 0)

{

delete[] data.data;

}

data.dim = \_dim;

data.data = new double[data.dim];

for (int i = 0; i < data.dim; i++)

{

data.data[i] = 0;

}

}

}

double& TMonom::operator[](int i)

{

return data.data[i];

}

TMonom& TMonom::operator=(const TMonom& \_v)

{

if (this == &\_v)

{

return \*this;

}

next = 0;

prev = 0;

data.K = \_v.data.K;

if (data.dim == \_v.data.dim)

{

for (int i = 0; i < data.dim; i++)

{

data.data[i] = \_v.data.data[i];

}

}

else if (\_v.data.dim == 0)

{

if (data.dim > 0)

{

delete[] data.data;

}

data.dim = 0;

data.data = 0;

}

else

{

if (data.dim > 0)

{

delete[] data.data;

}

data.dim = \_v.data.dim;

data.data = new double[data.dim];

for (int i = 0; i < data.dim; i++)

{

data.data[i] = \_v.data.data[i];

}

}

return \*this;

}

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

{

if (data.dim != \_v.data.dim)

{

throw logic\_error("Error");

}

TMonom temp(\*this);

temp.data.K = data.K \* \_v.data.K;

for (int i = 0; i < data.dim; i++)

{

temp.data.data[i] = this->data.data[i] + \_v.data.data[i];

}

return temp;

}

TMonom TMonom::operator/(const TMonom& \_v)

{

if (data.dim != \_v.data.dim)

{

throw logic\_error("Error");

}

TMonom temp(\*this);

temp.data.K = data.K / \_v.data.K;

for (int i = 0; i < data.dim; i++)

{

temp.data.data[i] = this->data.data[i] - \_v.data.data[i];

}

return temp;

}

TMonom\* TMonom::operator+(const TMonom& \_v)

{

if (data.dim != \_v.data.dim)

{

throw logic\_error("Error");

}

TMonom\* temp;

if (\*this == \_v)

{

temp = new TMonom(\*this);

temp->data.K = this->data.K + \_v.data.K;

}

else

{

temp = new TMonom[2];

if (\*this > \_v)

{

temp[0] = \*this;

temp[1] = \_v;

}

else

{

temp[0] = \_v;

temp[1] = \*this;

}

}

return temp;

}

TMonom\* TMonom::operator-(const TMonom& \_v)

{

if (data.dim != \_v.data.dim)

{

throw logic\_error("Error");

}

TMonom\* temp = 0;

if (\*this == \_v)

{

temp = new TMonom(\*this);

temp->data.K = this->data.K - \_v.data.K;

}

else

{

temp = new TMonom[2];

if (\*this < \_v)

{

temp[0] = \*this;

temp[1] = \_v;

}

else

{

temp[0] = \_v;

temp[1] = \*this;

}

}

return temp;

}

bool TMonom::operator>(const TMonom& \_v)

{

bool temp = true;

if (data.dim != \_v.data.dim)

{

temp = false;

}

else if (data.dim > 0)

{

for (int i = 0; i < data.dim; i++)

{

if (data.data[i] > \_v.data.data[i])

{

temp = true;

}

else if (data.data[i] < \_v.data.data[i])

{

temp = false;

}

else

{

continue;

}

}

temp = false;

}

else

{

return data.K > \_v.data.K;

}

return temp;

}

bool TMonom::operator<(const TMonom& \_v)

{

if (data.dim != \_v.data.dim)

{

return false;

}

if (data.dim > 0)

{

for (int i = 0; i < data.dim; i++)

{

if (data.data[i] < \_v.data.data[i])

{

return true;

}

else if (data.data[i] > \_v.data.data[i])

{

return false;

}

else

{

continue;

}

}

return false;

}

else

{

return data.K < \_v.data.K;

}

}

istream& operator>>(istream& istr, TMonom A)

{

int dim = 0;

if (A.data.dim != 0)

{

dim = A.data.dim;

}

else

{

istr >> dim;

A.SetDim(dim);

}

istr >> A.data.K;

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

{

istr >> A[i];

}

return istr;

}

ostream& operator<<(ostream& ostr, const TMonom& A)

{

ostr << A.data.K;

for (int i = 0; i < A.data.dim; i++)

{

ostr << "\* x[" << i << "]^" << A.data.data[i];

}

ostr << " ";

return ostr;

}

TMonomData::TMonomData(int t)

{

if (t < 0)

{

throw logic\_error("Error");

}

else if ( t == 0)

{

dim = 0;

data = 0;

}

else

{

data = new double[dim];

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

{

data[i] = 0;

}

}

}

TMonomData::TMonomData(double\* \_data, int \_dim, double \_K)

{

if (\_dim < 0)

{

throw logic\_error("Error");

}

dim = \_dim;

K = \_K;

if (dim > 0)

{

data = new double[dim];

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

{

data[i] = 0;

}

}

}

TMonomData::TMonomData(TMonomData& \_v)

{

dim = \_v.dim;

K = \_v.K;

if (dim > 0)

{

data = new double[dim];

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

{

data[i] = \_v.data[i];

}

}

}

TMonomData::~TMonomData()

{

if (data != 0)

{

delete[] data;

data = 0;

dim = 0;

K = 0;

}

}

TMonom\* TMonom::Clone()

{

TMonom\* res = new TMonom(\*this);

return res;

}