**Оглавление**

1. [.Введение 3](#_TOC_250005)
2. [.Постановка задачи 4](#_TOC_250004)
3. [.Структура проекта 7](#_TOC_250003)
4. [.Руководство пользователя 11](#_TOC_250002)
5. [.Заключение 15](#_TOC_250001)
6. [.Список литературы 16](#_TOC_250000)
7. .Приложение 17

# 1.Введение

### Сложные математические модели всегда интересовали профессиональных математиков. В начале двадцатого века центральной проблемой алгебры была проблема алгебры полиномов, а точнее оптимизация операций над ними и вычисление их значений. К этому времени алгебра полиномов уже имела значительный «багаж» теоретических знаний. Первооткрывателями в этой сфере были: Карл Фридрих Гаус и его основная теорема алгебры; Маклорен, Эйлер, Декарт. Все они внесли непомерный вклад в развитие алгебры полиномов и алгебры в целом.

### На сегодняшний день полиномы как формальный объект хорошо изучены в математике. Фактически математическая модель формируют *алгебру полиномов*. Сегодня вычисление и привидение полиномов к каноническому виду очень важно для корректной работы многих алгоритмов, определяющих работу станков, военной и гражданской техники. Поэтому для корректного вычисления и применения операций над полиномами необходимы компьютерные технологии. Это значительно увеличит скорость обработки полиномов, что сохраняет наиважнейший ресурс процесса – время. Плюс ко всему не будем забывать, что компьютер точнее в вычислениях, чем человек.

### И так, важность такой математической структуры как полиномы, неоспорима. Теперь необходимо переложить всю сложность операций над полиномами на машину, а именно на компьютер. Для этого необходимо выбрать структуру хранения для данной алгебраической структуры. Воспользуемся уже частично известной нам структурой – циклическим списком. Циклическая структура хранения списка отличается от линейной тем, что указатель последнего элемента не нулевой и указывает начало списка.

### Рассмотрение в рамках лабораторной работы задачи обработки полиномов обеспечивает начальное ознакомление с организацией аналитических вычислений на компьютере; основные учебно-методические цели работы состоят в следующем:

### практическое освоение основных принципов построения структур хранения данных в неодносвязных областях памяти (представление основных отношений в структурах данных при помощи адресных указателей);

### начальное знакомство с элементами теории списковых структур хранения данных (на примере линейного и циклического списков);

### получение практических навыков использования системы динамически- распределяемой памяти

### Результатом данной лабораторной работы должна стать программа, проводящая элементарные преобразования (сложение, вычитание, умножение на полином, умножение на число) над полиномами. При этом, в соответствии с требованиями, будет организован графический интерфейс для интуитивного управления данной программой.

Под ***полиномом*** (***многочленом****)* понимается выражение из нескольких ***термов***, соединенных знаками сложения или вычитания.

***Терм*** включает ***коэффициент*** и ***моном*,** содержащий одну или несколько переменных, каждая из которых может иметь *степень:*

*P =* Σ*Coef\*XAYBZC*

### Для формирования элементарной единицы полинома – монома – необходимо определить два элемента.

### Первым из них является коэффициент определяющий знак монома и величину первого сомножителя монома.

### Каждый моном, имеющий разный набор степеней, характеризуется величиной, называемой свёрткой. При этом, для нашего примера свёртка будет равна:

### Si=Ai\*maxSt2+Bi\*maxSt+Ci ( где maxSt - максимальная степень). Данная формула однозначно определяет набор степеней монома и после применения элементарных операций.

### Как уже было сказано, используем как структуру хранения циклический список. Первым (главным) элементом данной структуры сделаем first - элемент с коэффициентом -1 и со свёрткой равной нулю. Для обеспечение нашей структуры хранения мы создадим цикл, поставив указатель с первого элемента на самого себя. Покажем это графически:

#### -1 4

#### First 0 4

#### . . . 7

#### 5

### Вот и всё. Основные идеи и задачи данного проекта сформулированы.

### При выполнении работы можно использовать следующее понимание полинома. Полином состоит из мономов. Каждый моном характеризуется коэффициентом KOF и степенями переменных A, B, C: KOF\*x^Ay^Bz^C. Величину степеней переменных можно ограничить значением 9. При таком предположении максимально возможное количество мономов в полиноме - 1000. Полином разрежен, если по сравнению с максимально возможным количеством мономов он имеет относительно небольшое количество отличных от нуля коэффициентов.

### При манипуляции разреженными полиномами эффективной структурой хранения являются списки. При этом каждое звено списка хранит моном с отличным от нуля коэффициентом.

### Структура хранения полиномов должна обеспечивать эффективное выполнение операций. Если мономы в списке упорядочить по степеням переменных, то можно предложить эффективный алгоритм сложения полиномов, основанный на идее алгоритма слияния двух упорядоченных массивов.

### В результате операций (например, сложения) может быть получен полином, у которого нет отличных от нуля коэффициентов. Структура хранения такого нулевого полинома не должна отличаться от структуры хранения прочих полиномов.

### Для эффективного использования оперативной памяти реализуем операцию сложения таким образом, чтобы результат операции был получен на месте одного из слагаемых. При этом неизбежны вставки и удаления звеньев в списке. Для унификации вставок и удалений звеньев в начало, середину и конец списка рекомендуется использовать кольцевые списки.

### Таким образом, с учетом вышеизложенного, рекомендуется следующая структура хранения полиномов. Полиномы упорядочиваются по убыванию степеней мономов. Для определения старшинства мономов вводится следующее правило. Во-первых, фиксируется старшинство переменных. Будем считать, что x - самая старшая переменная, затем следует y, затем z. Для каждого монома определим его "свернутую степень" (индекс). Для монома x^Ay^Bz^C. индекс равен A\*102+B\*101+C\*100 (по условию задачи A, B и C не выше 9). Старшим считается моном с большей свернутой степенью. Например, x^3y старше xy^7z^6, так как 310 больше 176.

### Полином представляется кольцевым списком. Каждое звено списка соответствует моному с ненулевым коэффициентом. Звено состоит из трех полей и включает поле для хранения коэффициента, поле для хранения свернутой степени и поле для хранения указателя на звено следующего по порядку монома. Следующим после последнего звена является:

### -нулевой полином.



#### 0

#### -1

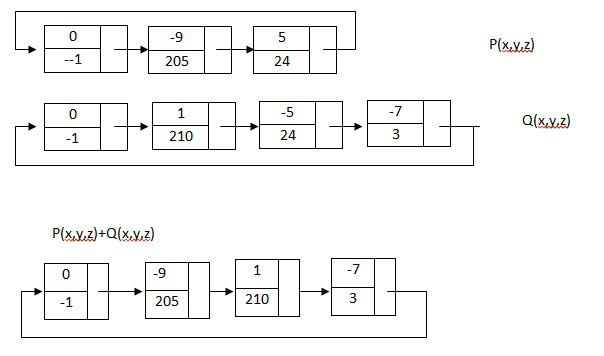
### Например, имеется два полинома: P(x,y,z)=5y^2z^4-9x^2z^5 и Q(x,y,z)=-7z^3+x^2y-5y^2z^4.

### В результате получим:

### Сложение: -9x^2z^5-7z^3+x^2y Вычитание: 10y^2z^4-9x^2z^5+7z^3-x^2y

### Умножение: 9x^4y^1z^5+5x^2y^3z^4+45x^2y^2z^9+63x^2z^8-25y^4z^8-35y^2z^7 Структура хранения полиномов должна обеспечивать эффективное выполнение операций над ними.

### Графическим представлением нашей структуры может служить объединение двух циклических списков, ну или же сумма двух циклических списков.



### Пример структуры хранения P(x,y,z) и Q(x,y,z) и их сумма. Аналогично получим и при вычитании и при произведении.

# 3. Структура проекта

## Проект состоит из трёх классов и формы. Между данными классами установлены связи, обеспечивающие актуальность выбранной структуры. Данный проект будет составлен с помощью известных нами средств связи классов, главным образом инкапсуляции. Структура проекта выглядит следующим образом:

Класс Monom

Класс Polynom

Класс Polynom

**Опишем классы и рассмотрим основные функции и методы: Класс Monom:**

class monom

{

private:

double coinf;

int svert;

monom\* Next;

public:

monom (double \_coinf=0, int \_svert=-1) //конструктор иницилизации

monom (monom & tmp) //конструктор копирования

monom operator=(monom & tmp) //оператор присваивания

void SetCoinf(double tmp) //метод задания коэфицента

void SetSvert(int tmp) //метод задания свёртки

void SetNext(monom\* tmp) //метод задания указателя на следующий элемент

double GetCoinf() // получить коэфицент

int GetSvert() // получить свёртку

monom\* GetNext() // получить следующий элемент

~monom () //диструктор

};

### Хранения полинома

### Как говорилось выше, полином состоит из мономов:

monom(int \_coif=0,int \_svert=-1)//структура хранения полинома

{

coif=\_coif; svert=\_svert;

Next=this;

};

### Сам полином хранится в виде списка мономов:

CircleList()// создана вершина полинома, с которой начнется добавление мономов

{

monom \*tmp=new monom();

first=tmp;

};

**Класс CircleList:**

class CircleList

{

private:

monom\* first;

public:

CircleList() // конструктор иницилизатор

void addmonom(monom \* contr) // добавление элемента к циклическому списку

CircleList ( CircleList const& tmp ) // конструктор копирования

~CircleList() // Диструктор

CircleList & operator=(CircleList & tmp) //оператор присваивания

CircleList operator+(CircleList copy)// оператор суммирования двух циклических списков

CircleList operator\*(double k) // оператор умножения на число

CircleList multy(CircleList tmp, int maxSt) // метод умножения списка на список

CircleList operator-(CircleList & tmp) // оператор вычитания списков

string getpolynom(int maxSt) // методы вывода циклического списка

void addmonom(int a, int s) // метод добавления монома в список при получении коэфицента и свёртки

};

### Реализация алгоритмов обработки данных

void addmonom(monom\* copytmp)// метод добавления мономов в список, с помощью одновременного обхода обоих списков

{

monom\* tmp(copytmp);

monom\* prev=first;

monom\* cur=first->getnext(); monom\* copy=new monom();

while (cur->getsvert()>tmp->getsvert())//цикл сравнений, который контролирует старшинство индексов(сверток)

{

prev=cur;

cur=cur->getnext();

};

равны

if( cur->getsvert()==tmp->getsvert())//сложение мономов, если индексы

{

cur->setkof(tmp->getkof()+cur->getkof());

if(cur->getkof()==0)

{

prev->setnext(cur->getnext()); cur->setnext(NULL);

delete cur;

}

}

изменений указателей

Else // добавление в список новых мономов,с помощью

{

copy=new monom(\*tmp); copy->setnext(cur); prev->setnext(copy);

};

string getpolynom(int maxst)//перевод полинома в строку но используемый в классе

Circlelist

{

string st=""; monom \*cur; int s;

cur=first->getnext();

while(cur!=first)

{

string stmonom=tost(cur->getkof());//tost-метод перевода числа в строку

if(stmonom[0]!='-')

stmonom='+'+stmonom;

if((s=cur->getsvert()/maxst/maxst)!=0)

stmonom=stmonom+"x^"+tost(s);

if((s=cur->getsvert()/maxst%maxst)!=0)

stmonom=stmonom+"y^"+tost(s);

if((s=cur->getsvert()%maxst)!=0) stmonom=stmonom+"z^"+tost(s);

st=st+stmonom; cur=cur->getnext();

};

return st;

};

**Класс Polynom:**

class Polynom

{

private:

int maxSt; CircleList\* Mylist;

void DelenieNaElm( string st, string \* w, int & k, string zn) // закрыт метод деления строки на элементы

void getMonom (string str, int & a, int & n, int & m, int & k) //метод получения числовых значений коэфицента и степеней

void get\_MyList(string str) // получить полином формируемый из ЦС

public:

Polynom ( string st="", int \_maxSt=10 // конструктор иницилизации

Polynom ( Polynom const & tmp ) // конструктор копирования

~Polynom() // Диструктор

Polynom & operator=(Polynom & tmp) // Оператор присваивания

Polynom operator+(Polynom & tmp) //Оператор сложения

Polynom operator\*(int k) // оператор умножения полинома на число

Polynom operator\*(Polynom & tmp) // оператор умножения полинома на полином

string getPolynom() // функция преобразования полинома в строк

};

Polynom(string st="",int \_maxst=100 )//пользовательский класс для визуализации структуры хранения

{

maxst=\_maxst;

mylist=new CircleList();

if(st.length()>0)

getmylist(st);// метод, получающий строку. Сначала разбиваем полином на мономы, а далее идёт обработка скрытым методом getmonom(string str,int &a,int &n,int &m,int

&k),проверяя каждый символ строки и сохроняя коэфицент монома и набор его степеней.

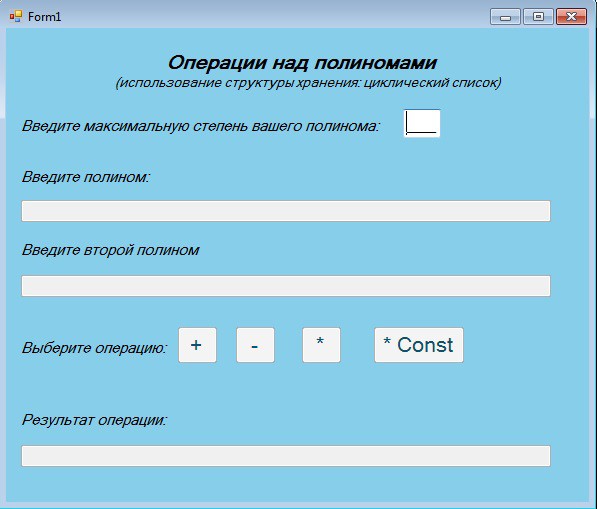
}

# 4.Руководство пользователя

## Как и всегда, сделаем ярлык на рабочем столе для удобства пользователя, и для уменьшения поиска программы в системе данных.

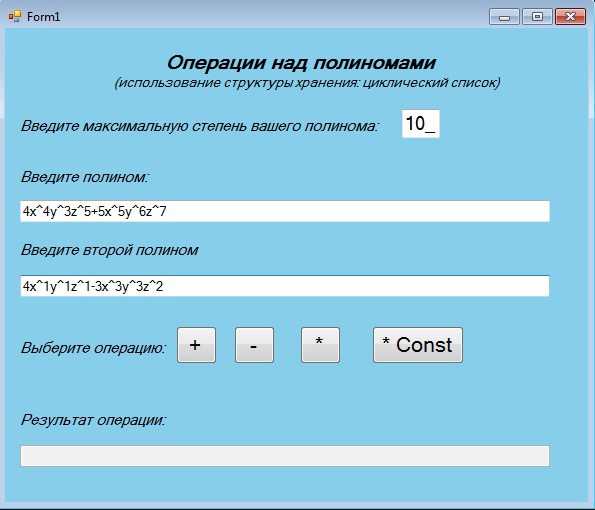
## Затем, кликнув по иконке мышкой (дважды), приступим к работе.

## Должно появиться окно, вида:



### И для начала введём максимальную степень нашего полинома. Заметим, что пока не будет введена максимальная степень остальные поля будут заблокированы. Всё поддерживается програмно, на основе методов Windows Forms.

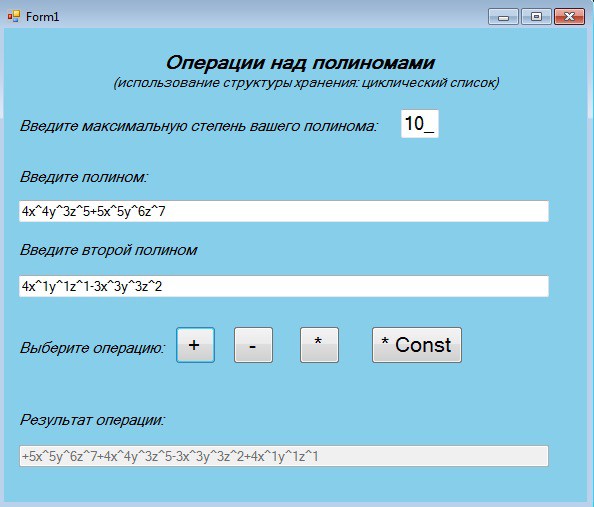
### После введения максимальной степени и разблокировки полей вводим в поля соответственно первый и второй полиномы.



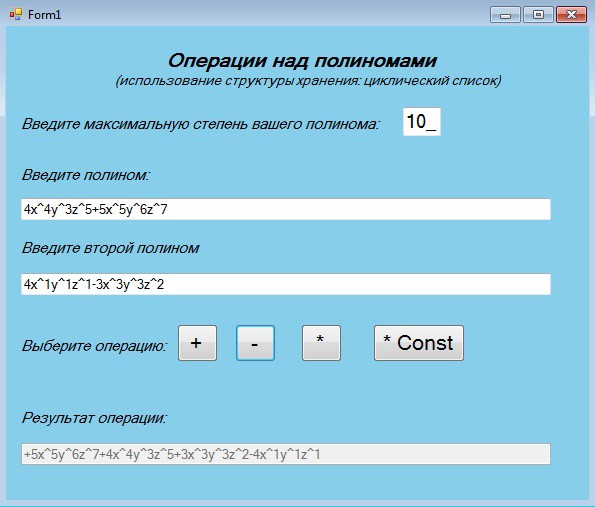
### Далее используя соответствующие кнопки сложения, вычитания, умножение полиномов и умножения полинома на число мы можем рассчитать, а результат представить в поле

### «Результат операции».

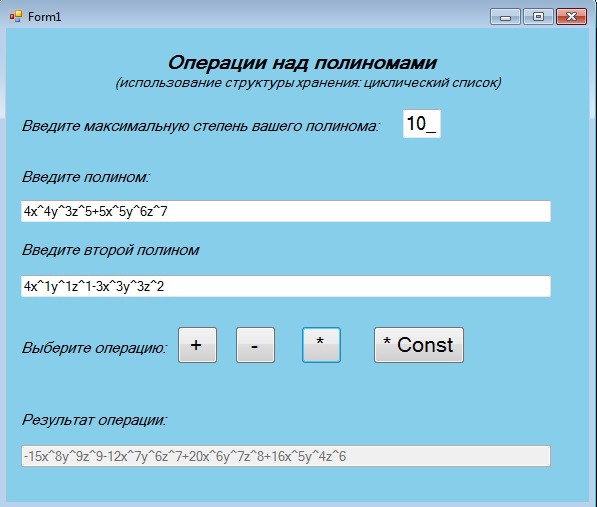
### И так проверим работоспособность каждой кнопки: Сумма:



### Справедливость вычислений очевидна. Вычитание:

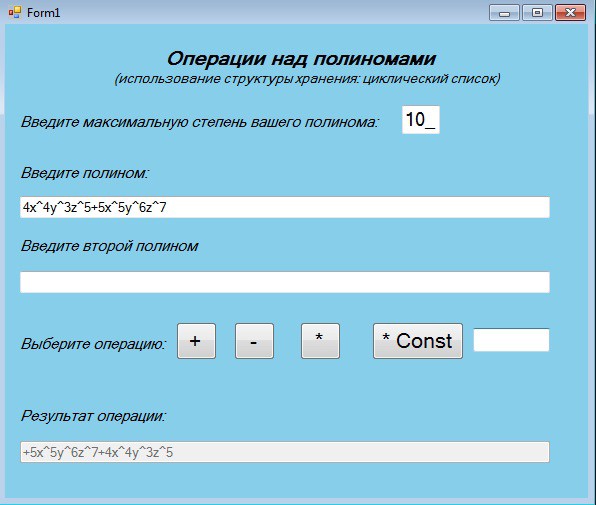


### Умножение полинома на полином:

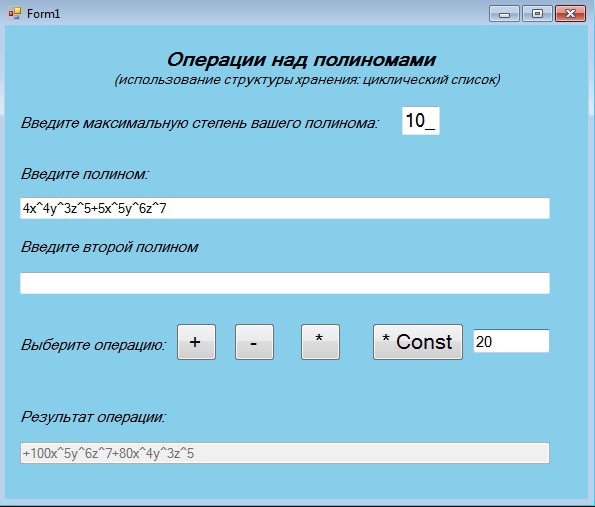


### Умножение полиномов тоже верно, что так же можно рассчитать.

### Немного интереснее сделано умножение на число. При нажатии на кнопку \*Const появится отдельное окно с инструкцией. Вводим коэффициент на который будет умножен наш первый введённый полином.



### Заметьте. Как только вы введёте число в появившемся окне, то в течении секунды ваш полином будет умножен на это число.



### Ну вот и всё. Для нового начала использования данной программы достаточно удалить старую максимальную степень и ввести новую.

# 5.Заключение

### Нам удалось освоить и представить в нашем проекте новые для нас принципы построения структур хранения данных в несвязных между собой участках памяти

### компьютера. Научились использовать указатели в представлении отношений между данными.

### Познакомились с теорией циклических списковых структур хранения данных.

### Получили практический навык использования системы динамически-распределяемой памяти.

### Получили навык работы с новой версией компилятора winforms и соответственно с основными атрибутами данной среды.

### Наши теоретические представления о результатах работы приложения оказались верны и результаты примера совпали нашими расчетами.

# Список литературы

### Б. Страуструп Язык программирования С++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008 г. – 1104 с.: ил.

### Р.Лафоре. Объектно-ориентированное программирование в языке СИ++.-М: «ПИТЕР», 2004 г.-922 с.

### <http://algolist.manual.ru/maths/misc/revpn.php>

**7.Приложение Класс Monom:**

#include "stdafx.h"

class monom

{

private:

double coinf; int svert; monom\* Next;

public:

monom (double \_coinf=0, int \_svert=-1) //конструктор иницилизации

{

svert=\_svert; coinf=\_coinf;

Next=this;

}

monom (monom & tmp) //конструктор копирования

{

coinf=tmp.coinf; svert=tmp.svert; Next=0;

}

monom operator=(monom & tmp) //оператор присваивания

{

coinf=tmp.coinf;

svert=tmp.svert;

return \*this;

}

void SetCoinf(double tmp) //метод задания коэфицента

{

coinf=tmp;

}

void SetSvert(int tmp) //метод задания свёртки

{

svert=tmp;

}

void SetNext(monom\* tmp) //метод задания указателя на следующий элемент

{

Next=tmp;

}

double GetCoinf() // получить коэфицент

{ return coinf;}

int GetSvert() // получить свёртку

{return svert;}

monom\* GetNext() // получить следующий элемент

{ return Next;}

~monom () //диструктор

{

Next=0;

delete Next;

}

};

**Класс CircleList:**

#include "stdafx.h"

#include <iostream>

#include "monom.cpp"

#include <string>

using namespace std;

class CircleList

{

private:

monom\* first;

public:

CircleList() // конструктор иницилизатор

{

monom \*tmp;

tmp=new monom(); first=tmp;

}

void addmonom(monom \* contr) // добавление элемента к циклическому списку

{

monom \*tmp; tmp=new monom;

\*tmp=\*contr;

monom\* prev=first;

monom\* cur=first->GetNext();

while (cur->GetSvert()>tmp->GetSvert())

{

prev = cur;

cur=cur->GetNext();

}

if (cur->GetSvert()==tmp->GetSvert())

{

cur->SetCoinf(cur->GetCoinf()+tmp->GetCoinf());

if (cur->GetCoinf()==0)

{

prev->SetNext(cur->GetNext()); cur->SetNext(NULL);

delete cur;

}

}

else

{

}

}

tmp->SetNext(cur); prev->SetNext(tmp);

CircleList ( CircleList const& tmp ) // конструктор копирования

{

monom \*tmp1; tmp1=new monom(); first=tmp1;

monom \* cur= tmp.first->GetNext();

while (cur!=tmp.first)

{

monom \* newcur (cur);

addmonom(newcur); //this->addmonom(newcur);

cur=cur->GetNext();

}

}

~CircleList() // Диструктор

{

monom \* cur=first->GetNext(); first->SetNext(NULL);

//delete cur;

first=NULL;

}

CircleList & operator=(CircleList & tmp) //оператор присваивания

{

monom \* prev=first;

monom \* cur = first->GetNext();

monom \* tmpcur=tmp.first->GetNext();

while ((cur!=first) && (tmpcur!=tmp.first))

{

\*cur=\*tmpcur; cur=cur->GetNext();

tmpcur=tmpcur->GetNext();

}

if (cur==first)

{

while (tmpcur!=tmp.first)

{

}}

else

{

monom copy= \*tmpcur; //may be point

addmonom(&copy);

tmpcur=tmpcur->GetNext();

if (cur!=first)

{

prev->SetNext(first); //????

while (cur!=first)

{

prev=cur;

cur=cur->GetNext(); prev->SetNext(NULL); delete prev;

}

}

}

return \*this;

}

CircleList operator+(CircleList copy) // оператор суммирования двух циклических списков

{

CircleList tmp=copy;

monom \* cur=first->GetNext();

while (cur!=first)

{

monom copymonom=\*cur; tmp.addmonom(&copymonom); cur=cur->GetNext();

}

return tmp;

}

CircleList operator\*(double k) // оператор умножения на число

{

if (k!=0)

{

CircleList copy=\*this;

monom \* cur=copy.first->GetNext();

while (cur!=copy.first)

{

}

return copy;

}

else

cur->SetCoinf(cur->GetCoinf()\*k); cur=cur->GetNext();

{CircleList copy;

return copy;}

}

CircleList multy(CircleList tmp, int maxSt) // метод умножения списка на список

{

CircleList rez;

int maxSv=(maxSt\*maxSt+maxSt)\*maxSt+maxSt;

monom \* cur=first->GetNext();

while (cur!=first)

{

monom \* curtmp=tmp.first->GetNext();

while (curtmp!=tmp.first)

{

monom rezmonom ((cur->GetCoinf())\*(curtmp->GetCoinf()), cur-

>GetSvert()+curtmp->GetSvert());

if

(((rezmonom.GetSvert()/(maxSt\*maxSt))>=maxSt)||((rezmonom.GetSvert()/maxSt%maxSt)>=maxSt)||(( rezmonom.GetSvert()%maxSt)>=maxSt))

{

}

else

{

}

}

//delete rez;

CircleList rez;

return rez;

rez.addmonom(&rezmonom); curtmp=curtmp->GetNext();

cur=cur->GetNext();

}

return rez;

}

CircleList operator-(CircleList & tmp) // оператор вычитания списков

{

CircleList rez=tmp\*(-1);

return \*this+rez;

}

string getpolynom(int maxSt) // методы вывода циклического списка

{

string st="";

monom \* cur = first->GetNext();

while (cur != first)

{

char buffer[100];

string stmonom = itoa(cur->GetCoinf(), buffer, 10);

if (stmonom[0]!='-') //||(stmonom[0]!='+'))

stmonom='+'+stmonom;

\_itoa\_s((cur->GetSvert()/(maxSt\*maxSt)), buffer, 10);

stmonom = stmonom+ "x^" + buffer;

\_itoa\_s((cur->GetSvert()/maxSt%maxSt), buffer, 10); stmonom = stmonom+ "y^" + buffer;

\_itoa\_s((cur->GetSvert() % maxSt), buffer, 10); stmonom = stmonom+ "z^" + buffer;

st=st + stmonom; cur=cur->GetNext();

}

return st;

}

void addmonom(int a, int s) // метод добавления монома в список при получении коэфицента и свёртки

{

monom tmp(a,s); addmonom(&tmp);

}

};

**Класс Polynom:**

#include "stdafx.h"

#include "CircleList.cpp"

using namespace std;

class Polynom

{

private:

int maxSt;

CircleList\* Mylist;

void DelenieNaElm( string st, string \* w, int & k, string zn) // закрыт метод деления строки на элементы

{

k=0;

string out=st+"$"; string cur=""; w[k]="";

for (int i=0; i<out.length(); i++)

{

if ((zn.find(out[i])==-1)||(i==0))

{

if ((out[i]!=' ')&&(out[i]!='$'))

{

w[k]=w[k]+out[i];

}

else

{

}

}

k++;

}

}

if (zn.find(out[i])!=-1)

{

k++;

w[k]=w[k]+out[i];

}

void getMonom (string str, int & a, int & n, int & m, int & k) //метод получения числовых значений коэфицента и степеней

{

string st="";

st=str[0];

for (int i=0; i<str.length(); i++)

{

switch (str[i])

{

case 'x':

{

string nstr="";

if (str[i+1]!='^')

n=1;

else

{

(i++); i++;

while

((str.length()>i)&&(str[i]>='0')&&(str[i]<='9'))

{

nstr+=str[i++];

}

i--;

} break; case 'y':

{

}

n=atoi(nstr.c\_str()); // может лучше atof???

string nstr="";

if (str[i+1]!='^')

m=1;

else

{

(i++); i++;

while

((str.length()>i)&&(str[i]>='0')&&(str[i]<='9'))

{

nstr+=str[i++];

}

i--;

} break; case 'z':

}

m=atoi(nstr.c\_str()); // может лучше atof???

{

string nstr="";

if (str[i+1]!='^')

k=1;

else

{

(i++); i++;

while

((str.length()>i)&&(str[i]>='0')&&(str[i]<='9'))

{

nstr+=str[i++];

}

i--;

} break;

case '+':

{

}

k=atoi(nstr.c\_str()); // может лучше atof???

if ((str[i+1]=='x')||(str[i+1]=='y')||(str[i+1]=='z'))

{

}

else

{

st=st+"1";

while ((++i<=str.length())&&(str[i]>='0')&&(str[i]<='9'))

{

st+=str[i];

}

}

a=atof(st.c\_str()); i--;

} break;

case '-':

{

if ((str[i+1]=='x')||(str[i+1]=='y')||(str[i+1]=='z'))

{

}

else

{

st=st+"1";

while ((++i<=str.length())&&(str[i]>='0')&&(str[i]<='9'))

{

st+=str[i];

}

}

a=atof(st.c\_str()); i--;

} break;

}

}

}

void get\_MyList(string str) // получить полином формируемый из ЦС

{

if ((str[0]!='-') || (str[0]!='+')) str="+"+str;

string \* w;

w=new string [str.length()/2]; int dl=0, a=0, n=0, m=0, k=0; string zn="+-=\*/";

DelenieNaElm( str, w, dl, zn);

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

{

getMonom(w[i],a,n,m,k);

Mylist->addmonom(a,n\*maxSt\*maxSt+m\*maxSt+k);

}

}

public:

Polynom ( string st="", int \_maxSt=10) // конструктор иницилизации

{

maxSt=\_maxSt+1; Mylist=new CircleList(); if (st.length()>0)

{

get\_MyList(st);

}

}

Polynom ( Polynom const & tmp ) // конструктор копирования

{

maxSt=tmp.maxSt; Mylist=new CircleList();

\*Mylist=\*tmp.Mylist;

}

~Polynom() // Диструктор

{

delete Mylist;

maxSt=0; //i am not trust

}

Polynom & operator=(Polynom & tmp) // Оператор присваивания

{

maxSt=tmp.maxSt;

\*Mylist=\*tmp.Mylist;

return \*this;

}

Polynom operator+(Polynom & tmp) //Оператор сложения

{

Polynom rez; rez.maxSt=maxSt;

\*rez.Mylist=\*Mylist+(\*tmp.Mylist);

return rez;

}

Polynom operator-(Polynom & tmp) //Оператор вычитания

{

Polynom rez; rez.maxSt=maxSt;

\*rez.Mylist=\*Mylist-(\*tmp.Mylist);

return rez;

}

Polynom operator\*(int k) // оператор умножения полинома на число

{

Polynom rez; rez.maxSt=maxSt;

\*rez.Mylist=(\*Mylist)\*k; //?????

return rez;

}

Polynom operator\*(Polynom & tmp) // оператор умножения полинома на полином

{

Polynom rez; rez.maxSt=maxSt;

\*rez.Mylist=(\*Mylist).multy(\*tmp.Mylist,maxSt);

return rez;

}

string getPolynom() // функция преобразования полинома в строку

{

return Mylist->getpolynom(maxSt);

}