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

Хранение и обработка полиномов

Выполнил:

студент ф-та ИИТММ гр. 381808-2

Подскребко Н.В.

Нижний Новгород

2019

Содержание

Введение…………………………………………………………………………………………..…2

Постановка Задачи………………………………………………………………………………….3

Руководство Пользователя………………………………………………………………….……..3

Руководство Программиста……………………………………………………………….………4

* Описание Структуры Программы…………………………………………..…………..5
* Описание алгоритмов…………………………………………………………..…….….6

Заключение………………………………………………………………………………….….….7

Приложение…………………………………………………………………………………..…....8

**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. Но так было не всегда, стоит нам вернуться немного назад, вниз по временной петле.

Сами мат-модели всегда интересовали математиков как таковые и уже в начале двадцатого века центральной проблемой алгебры была проблема оптимизации операций над этими самыми полиномами и вычислением их значений. К тому времени алгебра полиномов уже имела значительный фундамент теоретических знаний. Первооткрывателями в этой сфере были: Гаус, Эйлер, Маклорен и Декарт. Они внесли очень значительную лепту в наши современные познания и сформулировали многие ныне общепринятые вещи.

На сегодняшний же день полиномы как хорошо изучены в математике. Фактически мат-модель формируют алгебру полиномов. Сегодня вычисление и привидение полиномов к каноническому виду очень важно для корректной работы многих алгоритмов, определяющих работу самой различной техники. Поэтому для корректного вычисления и применения операций над полиномами необходимы компьютерные технологии. Это значительно увеличит скорость обработки полиномов, что сохраняет огромное количество времени, а это очень важно.

**Постановка задачи**

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 списка);
* 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

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

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

**P = ΣCoef\*XAYBZC**

Для формирования элементарной единицы полинома – **монома** – необходимо определить два элемента. Первый – коэффициент, определяющий знак монома и величину первого сомножителя монома. Второй - набор степеней монома для каждой переменной. Этот набор задаётся используя несколько структур ***S вида***:

***struct S***

***{***

***имя переменной;***

***степень переменной;***

***}***

**Руководство пользователя**

Для использования доступен консольный проект **main** и статическая библиотека **polynomial**.

Данный проект позволяет хранить полиномы, выполнять с ними арифметические действия и вычислять их значение в какой-либо точке.

* ***Добавить***

Для добавления необходимо выбрать пункт меню 1. Полином добавляется в формате, состоящем из **имени** (название переменных, которые могут встречаться в данном полиноме)и **максимальной степени**(максимальная степень каждой переменной, должна быть степенью 10). Если формат не задан, программа его сама запросит. Задать или посмотреть текущий формат можно с помощью пунктов 6 и 7.

***Полином вводится в формате:***

**coefficient1\*x^i1\*y^i2\*…\*z^ij+coef2\*x^i1\*y^i2\*\*…\*z^ij+…**

// Дробный коэффициент пишется через запятую

* ***Удалить***

Каждому полиному присваивается номер и для удаления полинома необходимо ввести его номер

* ***Выполнить арифметическое действие***

Формат ввода:

Доступные операторы:

* + Умножение.
  + Сложение.
  + Вычитание.

**Руководство программиста**

**Описание структуры программы**

Описание должно содержать то, из каких модулей состоит программа, назначение модулей, состав модулей, какие функции и классы содержатся в модуле и алгоритм работы программы (например, ввод таких-то данных, обработка таким-то образом, вывод результата).

Проект состоит из трёх файлов:

* Source.cpp;
* Polynom.cpp;
* Polynom.h;

Так как полином удобно представить в виде нескольких мономов, класс **Polynom** состоит из списка мономов.

**Описание алгоритмов**

* Сложение/вычитание мономов

Если два монома «корректны» (используются одни и те же переменные и степени равны), то складываются (вычитаются) коэффициенты

* Сложение/вычитание полиномов

Почленно складываются мономы обоих полиномов. Если очередной моном больше последнего монома в результирующем полиноме – он вставляется перед курсором.

* Умножение мономов

Если два монома «корректны» (используются одни и те же переменные и сумма степеней меньше максимальной степени), то складываются степени и умножаются коэффициенты.

* Умножение полиномов

Последовательно умножается полином 1 на мономы полинома 2 и получившийся полином прибавляется к результату.

* Вычисление монома в точке

Возводится значения переменных в необходимую степень и умножаем на коэффициент.

* Вычисление полинома в точке

Суммируются значения всех мономов полинома.

* Быстрое возведение числа в целочисленную степень

Реализован способ рекурсивный способ возведения в степень. Число умножается само на себя, а степень удваивается, пока степень меньше или равна необходимой. После этого, число умножается на один из предыдущих результатов, а увеличивается на соответствующее значение. Время работы алгоритма O(lg(n)), где n – необходимвя степень.

**Заключение**

В проекте использованы новые (относительно других проектов) принципы построения структур хранения данных в несвязных между собой участках памяти компьютера.

Были использованы указатели в представлении отношений между данными. Создан интерфейс для работы с данными без прямого использования указателей.

**Код опубликован в личном репозитории на GitHub.**

**//** [**https://github.com/Scheusal/LabWork4.git**](https://github.com/Scheusal/LabWork4.git)

## Приложение:

#include "Polynom.h"

#include <string>

using namespace std;

int PartOfStringToInt(string str, int& pos)

{

int res;

int j = pos;

int count = 0;

char \*str2;

string tmpstr;

while ((str[j] >= '0') && (str[j] <= '9'))

{

count++;

j++;

}

tmpstr = str.substr(pos, count);

str2 = new char[tmpstr.size() + 1];

pos = j;

for (j = 0; j < tmpstr.size() + 1; j++)

str2[j] = tmpstr[j];

res = strtod(str2, NULL);

return res;

}

int Polynom::maxPower;

Polynom::Polynom()

{

this->pHead = new List;

this->pHead->monom.coef = 0;

this->pHead->monom.power = -1;

this->pHead->pNext = pHead;

}

Polynom::Polynom(char\* strch)

{

setlocale(LC\_ALL, "Russian");

string str(strch);

int size = str.size();

string tmpstr;

char now;

int i, j, count;

for (i = 0; i < size; i++)

{

now = str[i];

if (!((now == '\*') || (now == '^') || (now == '+') || (now == '-') || (now == 'x') || ((now >= '0') && (now <= '9'))) || (now == ' '))

throw"Введён недопустимый символ";

}

List\* listStart = new List;

listStart->monom.power = -1;

listStart->monom.coef = 0;

List\* currentPartOfList = listStart;

i = 0;

while (i < size)

{

j = i;

count = 0;

if ((str[j] == '-') || (str[j] == '+'))

{

j++;

count++;

}

while ((str[j] != '+') && (str[j] != '-'))

{

count++;

j++;

if (j == size)

break;

}

tmpstr = str.substr(i, count);

j = 0;

currentPartOfList->pNext = new List;

currentPartOfList = currentPartOfList->pNext;

currentPartOfList->monom.coef = 1;

if (tmpstr[j] == '-')

{

currentPartOfList->monom.coef = -1;

j++;

}

if (tmpstr[j] == '+')

j++;

if ((tmpstr[j] >= '0') && (tmpstr[j] <= '9'))

currentPartOfList->monom.coef \*= PartOfStringToInt(tmpstr, j);

else if (tmpstr[j] == 'x');

else

throw"Ошибка ввода полинома: Неверно начат полином";

if (tmpstr[j] == '\*')

j++;

currentPartOfList->monom.power = 0;

if (tmpstr[j] == 'x')

{

j++;

if (tmpstr[j] == '^')

{

j++;

if (!((tmpstr[j] >= '0') && (tmpstr[j] <= '9')))

throw "Ошибка ввода полинома: Неправильная запись монома 1";

}

if ((tmpstr[j] >= '0') && (tmpstr[j] <= '9'))

currentPartOfList->monom.power += maxPower \* maxPower\*PartOfStringToInt(tmpstr, j);

else

currentPartOfList->monom.power += maxPower \* maxPower \* 1;

}

if (j != tmpstr.size())

throw"Ошибка ввода полинома: Неправильная запись монома 4";

i += j;

}

currentPartOfList->pNext = listStart;

this->pHead = listStart;

this->SortByMaxPower();

}

Polynom::Polynom(string str)

{

setlocale(LC\_ALL, "Russian");

int size = str.size();

string tmpstr;

char now;

int i, j, count;

for (i = 0; i < size; i++)

{

now = str[i];

if (!((now == '\*') || (now == '^') || (now == '+') || (now == '-') || (now == 'x') || ((now >= '0') && (now <= '9'))) || (now == ' '))

throw"Введён недопустимый символ";

}

List\* listStart = new List;

listStart->monom.power = -1;

listStart->monom.coef = 0;

List\* currentPartOfList = listStart;

i = 0;

while (i < size)

{

j = i;

count = 0;

if ((str[j] == '-') || (str[j] == '+'))

{

j++;

count++;

}

while ((str[j] != '+') && (str[j] != '-'))

{

count++;

j++;

if (j == size)

break;

}

tmpstr = str.substr(i, count);

j = 0;

currentPartOfList->pNext = new List;

currentPartOfList = currentPartOfList->pNext;

currentPartOfList->monom.coef = 1;

if (tmpstr[j] == '-')

{

currentPartOfList->monom.coef = -1;

j++;

}

if (tmpstr[j] == '+')

j++;

if ((tmpstr[j] >= '0') && (tmpstr[j] <= '9'))

currentPartOfList->monom.coef \*= PartOfStringToInt(tmpstr, j);

else if (tmpstr[j] == 'x');

else

throw"Ошибка ввода полинома: Неверно начат полином";

if (tmpstr[j] == '\*')

j++;

currentPartOfList->monom.power = 0;

if (tmpstr[j] == 'x')

{

j++;

if (tmpstr[j] == '^')

{

j++;

if (!((tmpstr[j] >= '0') && (tmpstr[j] <= '9')))

throw "Ошибка ввода полинома: Неправильная запись монома 1";

}

if ((tmpstr[j] >= '0') && (tmpstr[j] <= '9'))

currentPartOfList->monom.power += maxPower \* maxPower\*PartOfStringToInt(tmpstr, j);

else

currentPartOfList->monom.power += maxPower \* maxPower \* 1;

}

if (j != tmpstr.size())

throw"Ошибка ввода полинома: Неправильная запись монома 4";

i += j;

}

currentPartOfList->pNext = listStart;

this->pHead = listStart;

this->SortByMaxPower();

}

Polynom::Polynom(const Polynom &polynom)

{

this->pHead = new List;

this->pHead->monom.coef = 0;

this->pHead->monom.power = -1;

List\* tmp = new List;

List\* startTmp = tmp;

List\* curentList = polynom.pHead;

curentList = curentList->pNext;

if (curentList->monom.power == -1)

{

startTmp->pNext = this->pHead;

}

while (curentList->monom.power != -1)

{

tmp->pNext = new List;

tmp = tmp->pNext;

tmp->monom.coef = curentList->monom.coef;

tmp->monom.power = curentList->monom.power;

curentList = curentList->pNext;

}

this->pHead->pNext = startTmp->pNext;

tmp->pNext = this->pHead;

}

Polynom::~Polynom()

{

List \*tmp = this->pHead;

tmp = tmp->pNext;

List \*tmpDel;

while (tmp->monom.power != -1)

{

tmpDel = tmp;

tmp = tmp->pNext;

delete tmpDel;

}

delete tmp;

}

void Polynom::SortByMaxPower()

{

int maxDegree;

int start = 1;

List\* tmp;

List\* prePointer=0;

List\* lastPointer=0;

List\* maxPointer=0;

List\* Pointer;

int count = 0;

tmp = this->pHead;

tmp = tmp->pNext;

while (tmp->monom.power != -1)

{

tmp = tmp->pNext;

count++;

}

for (int j = 0; j < count; j++)

{

tmp = this->pHead;

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

{

lastPointer = tmp;

tmp = tmp->pNext;

}

maxDegree = -1;

while (tmp->monom.power != -1)

{

if (tmp->monom.power > maxDegree)

{

maxDegree = tmp->monom.power;

maxPointer = tmp;

prePointer = lastPointer;

}

lastPointer = tmp;

tmp = tmp->pNext;

}

tmp = this->pHead;

for (int k = 0; k < start - 1; k++)

tmp = tmp->pNext;

if (tmp->pNext->monom.power != maxPointer->monom.power)

{

Pointer = tmp->pNext;

prePointer->pNext = maxPointer->pNext;

maxPointer->pNext = Pointer;

tmp->pNext = maxPointer;

}

start++;

}

}

Polynom Polynom::operator+(const Polynom &polynom)

{

Polynom tmpPolynom;

List\* result = tmpPolynom.pHead;

List\* Pol1Link = this->pHead;

Pol1Link = Pol1Link->pNext;

List\* Pol2Link = polynom.pHead;

Pol2Link = Pol2Link->pNext;

while ((Pol1Link->monom.power != -1) || (Pol2Link->monom.power != -1))

{

if (Pol1Link->monom.power < Pol2Link->monom.power)

{

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol2Link->monom.power;

result->monom.coef = Pol2Link->monom.coef;

Pol2Link = Pol2Link->pNext;

}

else if (Pol1Link->monom.power > Pol2Link->monom.power)

{

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol1Link->monom.power;

result->monom.coef = Pol1Link->monom.coef;

Pol1Link = Pol1Link->pNext;

}

else if (Pol1Link->monom.power == Pol2Link->monom.power)

{

if ((Pol1Link->monom.coef + Pol2Link->monom.coef) == 0)

{

Pol1Link = Pol1Link->pNext;

Pol2Link = Pol2Link->pNext;

continue;

}

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol1Link->monom.power;

result->monom.coef = Pol1Link->monom.coef + Pol2Link->monom.coef;

Pol1Link = Pol1Link->pNext;

Pol2Link = Pol2Link->pNext;

}

}

result->pNext = tmpPolynom.pHead;

return tmpPolynom;

}

Polynom Polynom::operator-(const Polynom &polynom)

{

Polynom tmpPolynom;

List\* result = tmpPolynom.pHead;

List\* Pol1Link = this->pHead;

Pol1Link = Pol1Link->pNext;

List\* Pol2Link = polynom.pHead;

Pol2Link = Pol2Link->pNext;

while ((Pol1Link->monom.power != -1) || (Pol2Link->monom.power != -1))

{

if (Pol1Link->monom.power < Pol2Link->monom.power)

{

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol2Link->monom.power;

result->monom.coef = Pol2Link->monom.coef;

Pol2Link = Pol2Link->pNext;

}

else if (Pol1Link->monom.power > Pol2Link->monom.power)

{

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol1Link->monom.power;

result->monom.coef = Pol1Link->monom.coef;

Pol1Link = Pol1Link->pNext;

}

else if (Pol1Link->monom.power == Pol2Link->monom.power)

{

if ((Pol1Link->monom.coef - Pol2Link->monom.coef) == 0)

{

Pol1Link = Pol1Link->pNext;

Pol2Link = Pol2Link->pNext;

continue;

}

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol1Link->monom.power;

result->monom.coef = Pol1Link->monom.coef - Pol2Link->monom.coef;

Pol1Link = Pol1Link->pNext;

Pol2Link = Pol2Link->pNext;

}

}

result->pNext = tmpPolynom.pHead;

return tmpPolynom;

}

Polynom Polynom::operator\*(const Polynom &polynom)

{

Polynom tmpPolynom;

List\* result = tmpPolynom.pHead;

List\* Pol1List = this->pHead;

Pol1List = Pol1List->pNext;

List\* Pol2List = polynom.pHead;

Pol2List = Pol2List->pNext;

while (Pol1List->monom.power != -1)

{

while (Pol2List->monom.power != -1)

{

result->pNext = new List;

result = result->pNext;

result->monom.power = Pol1List->monom.power + Pol2List->monom.power;

result->monom.coef = Pol1List->monom.coef \* Pol2List->monom.coef;

Pol2List = Pol2List->pNext;

}

Pol2List = polynom.pHead;

Pol2List = Pol2List->pNext;

Pol1List = Pol1List->pNext;

}

result->pNext = tmpPolynom.pHead;

return tmpPolynom;

/\*int tr = tmpPolynom.maxPower + tmpPolynom.maxPower;

Polynom trPolynom;

List\* ptr = trPolynom.pHead;

double pre = 0.0;

for (int i = tr; i != -1; i--)

{

do

{

if (result->monom.power == i)

{

pre += result->monom.coef;

}

result = result->pNext;

} while (result->pNext != this->pHead);

if (pre != 0.0)

{

ptr->pNext = new List;

ptr = ptr->pNext;

ptr->monom.power = i;

ptr->monom.coef = pre;

pre = 0.0;

}

}

ptr->pNext = trPolynom.pHead;

return trPolynom;\*/

}

Polynom Polynom::operator=(const Polynom &polynom)

{

this->pHead = new List;

this->pHead->monom.coef = 0;

this->pHead->monom.power = -1;

List\* tmp = new List;

List\* startTmp = tmp;

List\* curentList = polynom.pHead;

curentList = curentList->pNext;

if (curentList->monom.power == -1)

{

startTmp->pNext = this->pHead;

}

while (curentList->monom.power != -1)

{

tmp->pNext = new List;

tmp = tmp->pNext;

tmp->monom.coef = curentList->monom.coef;

tmp->monom.power = curentList->monom.power;

curentList = curentList->pNext;

}

this->pHead->pNext = startTmp->pNext;

tmp->pNext = this->pHead;

return \*this;

}

void Polynom::ShowPolynom()

{

List \*tmp = this->pHead;

tmp = this->pHead;

tmp = tmp->pNext;

if (tmp->monom.power == -1)

cout << "0";

while (tmp->monom.power != -1)

{

if (tmp->monom.coef != 1)

cout << tmp->monom.coef;

if ((tmp->monom.coef == 1) && (tmp->monom.power == 0))

cout << tmp->monom.coef;

if (tmp->monom.power / (maxPower\*maxPower) > 0)

{

cout << "x";

if (tmp->monom.power / (maxPower\*maxPower) > 1)

cout << "^" << tmp->monom.power / (maxPower\*maxPower);

}

tmp = tmp->pNext;

if (tmp->monom.coef != 0)

if (tmp->monom.coef > 0)

cout << "+";

}

cout << endl;

}

int Polynom::Calculate(int x)

{

List \*tmp = this->pHead;

int count = 0;

int res = 0;

int tmpint;

tmp = tmp->pNext;

while (tmp->monom.power != -1)

{

count = tmp->monom.coef;

tmpint = 1;

if (tmp->monom.power / (maxPower\*maxPower) != 0)

{

for (int i = 0; i < (tmp->monom.power / (maxPower\*maxPower)); i++)

tmpint \*= x;

count \*= tmpint;

}

tmpint = 1;

res += count;

tmp = tmp->pNext;

}

return res;

}

string Polynom::PolynomToString()

{

string str;

List \*tmp = this->pHead;

int count = 0;

tmp = tmp->pNext;

while (tmp->monom.power != -1)

{

tmp = tmp->pNext;

count++;

}

if (count == 0)

{

str = "0";

return str;

}

tmp = this->pHead;

tmp = tmp->pNext;

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

{

if (tmp->monom.coef != 1)

str += to\_string(tmp->monom.coef);

if ((tmp->monom.coef == 1) && (tmp->monom.power == 0))

str += to\_string(tmp->monom.coef);

if (tmp->monom.power / (maxPower\*maxPower) > 0)

{

str += "x";

if (tmp->monom.power / (maxPower\*maxPower) > 1)

{

str += "^";

str += to\_string(tmp->monom.power / (maxPower\*maxPower));

}

}

tmp = tmp->pNext;

if (tmp->monom.coef != 0)

if (tmp->monom.coef > 0)

str += "+";

}

return str;

}