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

ПОЛИНОМЫ

Выполнил: студент группы
3822Б1ПР2

_____ М.В.Фёдоров
Подпись

Проверила:

_____ Я.В. Силенко
Подпись

Нижний Новгород 2023

Содержание

Введение

Полином — это алгебраическое выражение, состоящее из переменных, возведенных в некоторую степень, с коэффициентами, которые могут быть как вещественными, так и комплексными числами. Полиномы являются одним из основных объектов изучения алгебры и имеют широкое применение в различных областях науки и техники.

Полиномы широко используются в различных областях науки и техники, таких как физика, химия, биология, экономика и другие. Например, в физике полиномы используются для моделирования движения тел, в химии — для описания химических реакций, в биологии — для моделирования роста организмов и т.д.

Ещё полиномы широко используются в физике для описания различных явлений и решения различных задач. Например, полиномы могут использоваться для описания движения тел в пространстве. Для этого можно записать уравнение движения в виде полинома, где переменная будет обозначать время, а коэффициенты - параметры движения, такие как масса тела, ускорение.

Еще одним примером использования полиномов в физике является моделирование электрических цепей. В этом случае полиномы используются для описания силы тока, напряжения и других параметров цепи. Для этого можно записать уравнение цепи в виде полинома, где переменная будет обозначать напряжение или силу тока, а коэффициенты - параметры цепи, такие как сопротивление, емкость.

Также полиномы могут использоваться для моделирования механических систем, например, для описания движения груза на пружине или движения тела в жидкости. В этих случаях полиномы используются для описания зависимости координаты или скорости от времени.
Кроме того, полиномы могут использоваться для решения различных задач, связанных с физикой, таких как расчет энергии, силы, момента импульса и т.д. Например, полиномы могут использоваться для расчета энергии гравитационного поля или энергии электрического поля.

Полиномы также широко используются для решения различных задач, связанных с расчетами. Например, полиномы могут использоваться для расчета стоимости товаров в магазинах, для прогнозирования цен на акции на бирже, для определения траектории движения спутников и т.д.

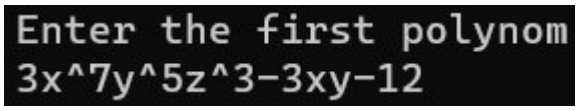
Постановка задачи

Выполнение работы предполагает написание программы для работы с полиномами, включающей в себя следующие этапы:

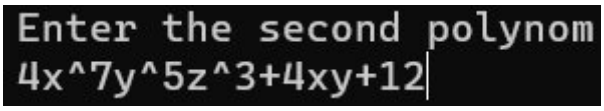
1) Формирование полинома из строки;
2) Вывод полинома в строку;
3) Сложение двух полиномов;

Руководство пользователя

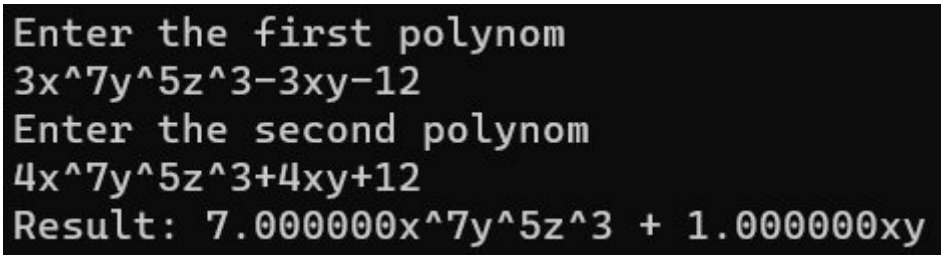
При запуске программы выводится сообщение с просьбой ввести первый полином:



После ввода первого полинома вводим второй полином:



И получаем полином, который является результатом сложения двух введённых пользователем полиномов.



Как мы видим, программа произвела вычисления и вывела правильный ответ.

Руководство программиста

Описание структуры программы

Программа состоит из следующих файлов:

1) TNode.h – содержит реализацию шаблонного класса TNode, который представляет узел для односвязного списка.
2) TList.h – содержит реализацию шаблонного класса TList, который представляет односвязный список.
3) THeadList.h – содержит реализацию шаблонного класса THeadList, который представляет список с ссылкой на первый узел.
4) TMonom.h – содержит реализацию класса TMonom, который представляет моном полинома.
5) TPolinom.h – содержит реализацию класса TPolinom, который представляет полином.

Описание структур данных

Class TNode:

Поля:

T value – значение звена;

TNode* pNext – индекс следующего звена;

Конструкторы и деструктор:

TNode();

TNode(const T& val);

TNode(const TNode<T>& node);

Class TList:

Поля:

TNode<T>* pFirst – указатель на первый элемент списка;

TNode<T>* pCurrent – указатель на текущий элемент списка;

TNode<T>* pPrevious – указатель на предыдущий элемент списка;

TNode<T>* pLast – указатель на последний элемент списка;

TNode<T>* pStop – указатель, конечная метка списка;

int length – длина списка;

Конструкторы и деструктор:

TList();

~TList();

Методы:

int GetLength() – возвращает длину списка;

bool IsEmpty() – проверка списка на пустоту;

void InsertFirst(T item) – вставляет элемент в начало списка;

void InsertCurrent(T item) – вставляет элемент перед текущим элементом списка;

void InsertLast(T item) – вставляет элемент в конец списка;

void DeleteFirst() – удаляет первый элемент списка:

void DeleteCurrent() – удаляет текущий элемент списка;

void GoNext() – переходит к следующему элементу списка;

void Reset() – устанавливает текущий элемент в начало списка;

bool IsEnd() – проверяет, является ли текущий элемент конечной меткой списка;

T GetCurrentItem() – возвращает текущий элемент списка;

void SetCurrentItem(T item) – устанавливает значение текущего элемента списка;

Class THeadList:

Поля:

TNode<T>* pHead – указатель на заголовочное звено;

Конструкторы и деструктор:

THeadList();

~THeadList();

Методы:

void InsertFirst(T item) – вставляет элемент в начало списка;

void DeleteFirst() – удаляет первый элемент списка;

Class TMonom:

Поля:

double coef – коэффициент монома;

int index – индекс монома (x, y, z);

Конструкторы и деструктор:

TMonom();

TMonom(double coef_arg, int degX, int degY, int degZ);

Методы:

void SetCoef(int cval) – устанавливает значение коэффициента монома;

double GetCoef(void) – возвращает значение индекса монома;

void SetIndex(int ival) – устанавливает значение индекса монома;

int GetIndex(void) – возвращает значение индекса монома;

Операторы:

bool operator==(const TMonom& other) – оператор сравнения равенства мономов;

bool operator>(const TMonom& other) – оператор сравнения для установления порядка мономов (больше);

bool operator<(const TMonom& other) – оператор сравнения для установления порядка мономов (меньше);

Class TPolinom:

Конструкторы и деструктор:

TPolinom();

TPolinom(TPolinom& other);

TPolinom(string str);

Методы:

void AddMonom(TMonom newMonom) – добавляет моном в полином;

TPolinom MultMonom(TMonom monom) – умножает полином на моном;

string ToString() – представляет полином в виде строки;

Операторы:

TPolinom& operator=(TPolinom& other) – оператор присваивания;

TPolinom& operator+(TPolinom& q) – оператор сложения полиномов;

TPolinom& operator*(double coef) – оператор умножения полинома на число;

bool operator==(TPolinom& other) – оператор сравнения на равенство полиномов;

Описание алгоритмов

Чтобы выполнить сложение и вычитание полиномов, нужно

1) Записать сумму/разность полиномов;
2) Если есть скобки, то раскрыть их;
3) Привести подобные мономы;
4) Записать полученный полином в порядке возрастания степеней у мономов;

Чтобы выполнить умножение полинома на число, нужно каждый моном полинома умножить на это число;

Чтобы выполнить умножение полиномов, нужно:

1) Записать произведение полиномов;
2) Умножить каждый моном первого полинома на каждый моном второго полинома;
3) Просуммировать полученные произведения мономов и привести подобные мономы;
4) Записать полученный полином в порядке возрастания степеней у мономов;

Результаты

P1 = $3x^7y^5z^3 - 3xy - 12$

P2 = $4x^7y^5z^3 + 4xy + 12$

P3 = P1 + P2 = $7x^7y^5z^3 + xy$

Результат работы программы:

```
Enter the first polynom
3x^7y^5z^3-3xy-12
Enter the second polynom
4x^7y^5z^3+4xy+12
Result: 7.000000x^7y^5z^3 + 1.000000xy
```

Заключение

В ходе данной лабораторной работы были успешно реализованы поставленные задачи, а именно формирование полинома из строки, вывод полинома в строку и сложение двух полиномов. Результаты работы программы соответствуют реальным вычислениям.

Литература

1) Правило сложения полиномов: http://www.cleverstudents.ru/expressions/addition_and_subtraction_of_polynomials.html
2) Лекции ННГУ по курсу «Алгоритмы и структуры данных»

Приложение

TNode.h

```
#pragma once

template<class T>
class TNode
{
public:
  T value;
  TNode* pNext;
  TNode() {
    value = T();
    this->pNext = nullptr;
  }
  TNode(const T& val) {
    this->value = val;
    this->pNext = nullptr;
  }
  TNode(const TNode<T>& node) {
    this->value = node.value;
    this->pNext = node.pNext;
  }
};
```

TList.h

```
#pragma once
```

```
#include "TNode.h"
#include <iostream>
using namespace std;

template<class T>
class TList
{
protected:
  TNode<T>* pFirst;
  TNode<T>* pCurrent;
  TNode<T>* pPrevious;
  TNode<T>* pLast;
  TNode<T>* pStop;
  int length;

public:
  TList();
  ~TList();
  int GetLength() { return length; }
  bool IsEmpty();

  void InsertFirst(T item);
  void InsertCurrent(T item);
  void InsertLast(T item);

  void DeleteFirst();
  void DeleteCurrent();

  void GoNext();
```

```
  void Reset();
  bool IsEnd();

  T GetCurrentItem();
  void SetCurrentItem(T item) { pCurrent->value = item; }
};

template <class T>
TList<T>::TList() : pFirst(nullptr), pCurrent(nullptr), pPrevious(nullptr), pLast(nullptr), pStop(nullptr), length(0) { }

template <class T>
TList<T>::~TList()
{
  while (!IsEmpty()) {
    DeleteFirst();
  }
}

template <class T>
bool TList<T>::IsEmpty()
{
  return pFirst == nullptr;
}
```

```
template <class T>
void TList<T>::InsertFirst(T item)
{
  TNode<T>* New_Node = new TNode<T>{ item, pFirst };
  pFirst = New_Node;
  if (length == 0)  pLast = pFirst;
  length++;
}


template <class T>
void TList<T>::InsertLast(T item)
{
  TNode<T>* New_Node = new TNode<T>(item);
  if (IsEmpty()) { pFirst = pLast = New_Node; }
  else {
    pLast->pNext = New_Node;
    pLast = New_Node;
  }
  length++;
}


template <class T>
void TList<T>::InsertCurrent(T item)
{
  if (pCurrent == nullptr  pCurrent == pFirst) {
    InsertFirst(item);
    return;
```

```
  }
  TNode<T>* New_Node = new TNode<T>{ item, pCurrent };
  pPrevious->pNext = New_Node;
  length++;
}


template <class T>
void TList<T>::DeleteFirst()
{
  if (IsEmpty()) throw runtime_error("List is empty");
  TNode<T>* temp = pFirst;
  pFirst = pFirst->pNext;
  if (pFirst == nullptr) pLast = nullptr;
  delete temp;
  length--;
}


template <class T>
void TList<T>::DeleteCurrent()
{
  if (pCurrent == nullptr) throw runtime_error("Current node is null");
  if (pCurrent == pFirst) {
    DeleteFirst();
    return;
  }
  if (pCurrent == pLast) {
    delete pCurrent;
```

```
    pPrevious->pNext = nullptr;
    pLast = pPrevious;
    pCurrent = nullptr;
  }
  else {
    pPrevious->pNext = pCurrent->pNext;
    delete pCurrent;
    pCurrent = pPrevious->pNext;
  }
}


template <class T>
T TList<T>::GetCurrentItem()
{
  if (pCurrent == nullptr) throw runtime_error("Current node is null");
  return pCurrent->value;
}


template <class T>
void TList<T>::Reset()
{
  pCurrent = pFirst;
  pPrevious = nullptr;
}


template <class T>
```

```
void TList<T>::GoNext()
{
  pPrevious = pCurrent;
  pCurrent = pCurrent->pNext;
  if (pCurrent == nullptr  pCurrent == pStop) { pLast = pPrevious; }
}



template <class T>
bool TList<T>::IsEnd()
{
  return pCurrent == pStop;
}
```

THeadList.h

```
#pragma once
#include "TList.h"
using namespace std;



template<class T>
class THeadList : public TList<T>
{
protected:
  TNode<T>* pHead;
public:
  THeadList();
  ~THeadList();
```

```
  void InsertFirst(T item);
  void DeleteFirst();

};
template<class T>
THeadList<T>::THeadList()
{
  pHead = new TNode<T>();
  if (pHead == nullptr) throw bad_alloc();
  this->pLast = pHead;
  pHead->pNext = this->pFirst;
}


template<class T>
THeadList<T>::~THeadList()
{
  delete pHead;
}


template <class T>
void THeadList<T>::InsertFirst(T item)
{
  TNode<T>* newNode = new TNode<T>{ item, nullptr };
  if (newNode == nullptr) throw bad_alloc();
  newNode->pNext = pHead->pNext;
  pHead->pNext = newNode;
  if (this->IsEmpty()) {
```

```
    this->pLast = newNode;
  }
  this->length++;
}


template <class T>
void THeadList<T>::DeleteFirst()
{
  if (this->pFirst == nullptr) throw runtime_error("List is empty");
  TNode<T>* temp = this->pFirst;
  this->pFirst = this->pFirst->pNext;
  if (this->pFirst == nullptr) {
    this->pLast = nullptr;
  }
  if (this->pCurrent == temp) {
    this->pCurrent = this->pFirst;
    this->pPrevious = nullptr;
  }
  delete temp;
  this->length--;
}
```

TMonom.h

```
#pragma once
using namespace std;

class TMonom
```

```
{
private:
  double coef;
  int index;
public:
  TMonom() {
    coef = 0;
    index = 0;
  }
  TMonom(double coef_arg, int degX, int degY, int degZ) : coef(coef_arg) {
   if (degX < 0  degY < 0  degZ < 0) throw invalid_argument("Can't work. Change dates");
    if (coef_arg == 0) throw invalid_argument("Can't work. Change dates");
    index = degX + degY * 10 + degZ * 100;
  }
  void SetCoef(int cval) { this->coef = cval; }
  double GetCoef(void) { return this->coef; }

  void SetIndex(int ival) { this->index = ival; }
  int GetIndex(void) { return this->index; }

  bool operator==(const TMonom& other) {
    return (this->coef == other.coef) && (this->index == other.index);
  }
  bool operator>(const TMonom& other) {
    return (this->index > other.index)  ((this->index == other.index) && (this->coef > other.coef));
  }
  bool operator<(const TMonom& other) {
    return (this->index < other.index)  ((this->index == other.index) && (this->coef < other.coef));
```

```
  }
};
```

TPolinom.h

```
#pragma once
#include "THeadList.h"

#include <string>
#include<sstream>
#include <string>
#include <vector>
#include <iostream>
#include <sstream>
#include <iomanip>
#include "TMonom.h"

using namespace std;

const int nonDisplayedZeros = 4;
const double EPSILON = 1e-6;



class TPolinom : public THeadList<TMonom>
{
public:
  TPolinom();
  TPolinom(TPolinom& other);
  TPolinom(string str);
```

```
  TPolinom& operator=(TPolinom& other);
  TPolinom& operator+(TPolinom& q);
  void AddMonom(TMonom newMonom);
  TPolinom MultMonom(TMonom monom);
  TPolinom& operator*(double coef);
  bool operator==(TPolinom& other);
  string ToString();
};


TPolinom::TPolinom() :THeadList<TMonom>::THeadList() {}


TPolinom::TPolinom(TPolinom& other)
{
  pHead = new TNode<TMonom>;
  TNode<TMonom>* h = other.pHead->pNext;
  while (h != other.pStop) {
    this->AddMonom(h->value);
    h = h->pNext;
  }
}


TPolinom::TPolinom(string str) {
  size_t pos = 0;
  while (pos < str.length()) {
    double coef = 0.0;
    int degX = 0, degY = 0, degZ = 0;
```

```
char var;
if (isdigit(str[pos])  str[pos] == '-'  str[pos] == '+') {
  size_t nextPos;
  coef = std::stod(str.substr(pos), &nextPos);
  pos += nextPos;
}
while (pos < str.length() && (str[pos] == 'x'  str[pos] == 'y'
  str[pos] == 'y' || str[pos] == 'z')) {
  var = tolower(str[pos]);
  pos++;
  if (pos < str.length() && str[pos] == '^') {
    pos++;
    size_t nextPos;
    int deg = std::stod(str.substr(pos), &nextPos);
    pos += nextPos;
    switch (var) {
    case 'x': degX = deg; break;
    case 'y': degY = deg; break;
    case 'z': degZ = deg; break;
    }
  }
  else {
    switch (var) {
    case 'x': degX = 1; break;
    case 'y': degY = 1; break;
    case 'z': degZ = 1; break;
    }
  }
}
```

```
    this->AddMonom(TMonom(coef, degX, degY, degZ));
  }
}


TPolinom& TPolinom::operator=(TPolinom& other)
{
  if (this != &other) {
    while (!this->IsEmpty()) {
      this->DeleteFirst();
    }
    TNode<TMonom>* current = other.pHead->pNext;
    while (current != nullptr) {
      this->AddMonom(current->value);
      current = current->pNext;
    }
  }
  return *this;
}


void TPolinom::AddMonom(TMonom m)
{
  if (m.GetCoef() == 0) throw invalid_argument("Can't work. Change dates");
  this->Reset();
  bool isAdded = false;
  while (!this->IsEnd()) {
    if (this->pCurrent->value.GetIndex() == m.GetIndex()) {
      this->pCurrent->value.SetCoef(pCurrent->value.GetCoef() + m.GetCoef());
```

```
      if (fabs(this->pCurrent->value.GetCoef()) < EPSILON) {
        this->DeleteCurrent();
      }
      isAdded = true;
      break;
    }
    this->GoNext();
  }
  if (!isAdded) this->InsertLast(m);
  pHead->pNext = this->pFirst;
}


TPolinom TPolinom::MultMonom(TMonom monom)
{
  TPolinom res(*this);
  TNode<TMonom>* current = res.pHead->pNext;
  while (current != nullptr) {
    current->value.SetCoef(current->value.GetCoef() * monom.GetCoef());
    current->value.SetIndex(current->value.GetIndex() + monom.GetIndex());
    current = current->pNext;
  }
  return res;
}


TPolinom& TPolinom::operator+(TPolinom& other)
{
  if (other.IsEmpty()) throw invalid_argument("Can't work. Change dates");
```

```
  TNode<TMonom>* current = other.pHead->pNext;
  while (current != nullptr) {
    this->AddMonom(current->value);
    current = current->pNext;
  }
  return *this;
}



TPolinom& TPolinom::operator*(double coef)
{
  if (this->IsEmpty()) throw invalid_argument("Can't work. Change dates");
  TNode<TMonom>* current = this->pHead->pNext;
  while (current != nullptr) {
    current->value.SetCoef(current->value.GetCoef()                    *
current->value.GetCoef());
    if (current->pNext == nullptr) break;
    current = current->pNext;
  }
  return *this;
}

bool TPolinom::operator==(TPolinom& other) {
  if (this->GetLength() != other.GetLength()) return false;
  TNode<TMonom>* thisCurrent = this->pHead->pNext;
  TNode<TMonom>* otherCurrent = other.pHead->pNext;
  while (thisCurrent != nullptr && otherCurrent != nullptr) {
    if (!(thisCurrent->value == otherCurrent->value)) { return false; }
    thisCurrent = thisCurrent->pNext;
    otherCurrent = otherCurrent->pNext;
```

```
  }
  return thisCurrent == otherCurrent;
}



string TPolinom::ToString() {
  string result;
  TNode<TMonom>* current = this->pHead->pNext;
  while (current != nullptr) {
    int degX = current->value.GetIndex() % 10;
    int degY = (current->value.GetIndex() / 10) % 10;
    int degZ = current->value.GetIndex() / 100;
    if (!result.empty()) result += (current->value.GetCoef() > 0) ? " + " : " - ";
    else if (current->value.GetCoef() < 0) { result += "-"; }
    result += to_string(abs(current->value.GetCoef()));
    if (degX > 0) {
      result += "x";
      if (degX > 1) {
        result += "^" + to_string(degX);
      }
    }
    if (degY > 0) {
      result += "y";
      if (degY > 1) {
        result += "^" + to_string(degY);
      }
    }
    if (degZ > 0) {
      result += "z";
```

```
      if (degZ > 1) {
        result += "^" + to_string(degZ);
      }
    }
    current = current->pNext;
  }
  return result;
}
```