Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Арифметические операции с полиномами

Выполнил:

студент ИИТММ ПМИ гр. 381703-1

Лаптев В. В.

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В. Д.

Нижний Новгород

2019 г.

Содержание

Введение3

Постановка задачи4

Руководство пользователя5

Руководство программиста8

Описание структур данных8

Описание алгоритмов10

Описание компонентов программного комплекса.14

Заключение.15

Список литературы.16

Приложение17

**Введение**

При решении различных прикладных задач нередко возникают такие объекты как полиномы. При достаточно большом количестве слагаемых выполнение «вручную» простых арифметических операций над ними становиться затруднительным. Поэтому возникает необходимость разработки программы, которая способна выполнять элементарные арифметические операции с полиномами и быстро вычислять результат той или иной операции.

**Постановка задачи**

Требуется разработать программу, выполняющую арифметические операции с полиномами трех переменных (x, y и z): сложение, вычитание, умножение на константу, умножение двух полиномов. Считается, что полином составлен из мономов от трех переменных с ограничением на степень каждой переменой от 0 до 9. Коэффициенты полинома - вещественные числа. Работоспособность программы необходимо проверить с помощью Google Test-ов. Кроме того, необходимо разработать пользовательское консольное приложение.

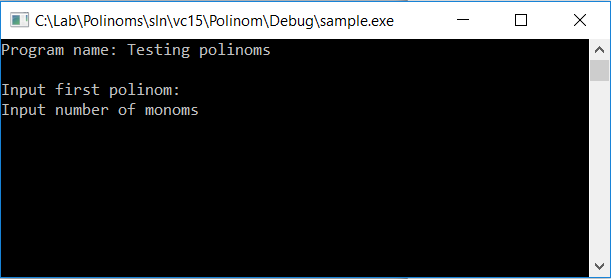
Особенности реализации:

1. В качестве структуры хранения полинома использовать список мономов.
2. Элементы списка хранить упорядоченными.
3. Степень полинома хранить в "свернутом" виде, т. е. степень должна быть представлена как трехзначное число
4. При умножении и сложение (вычитание) необходимо следить, чтобы в итоговом полиноме были приведены подобные слагаемые и не хранилось мономов с нулевым коэффициентом

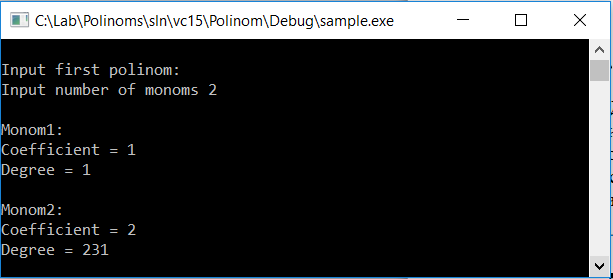
**Руководство пользователя**

Для начала работы программы нужно запустить на выполнение файл sample.exe.

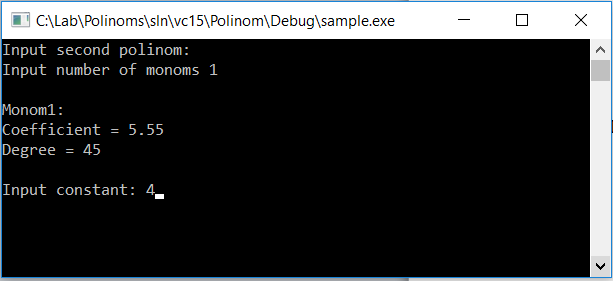
В появившемся окне необходимо ввести первый полином.



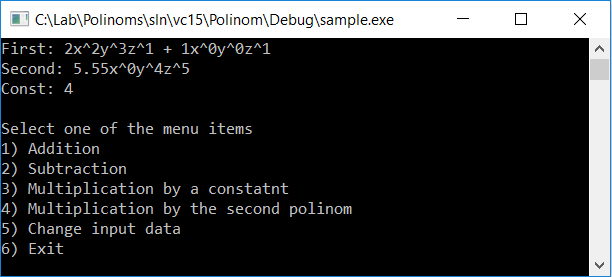
Для этого нужно указать число мономов, а затем выполнить их последовательный ввод. Ввод монома осуществляется посредством ввода его вещественного коэффициента и степени в виде целого, однозначного/двухзначного/трёхзначного числа.



Затем аналогичным образом необходимо ввести второй полином и указать вещественную константу.



После этого на экран будут выведены данные, введённые пользователем и меню, состоящее из 6 пунктов.



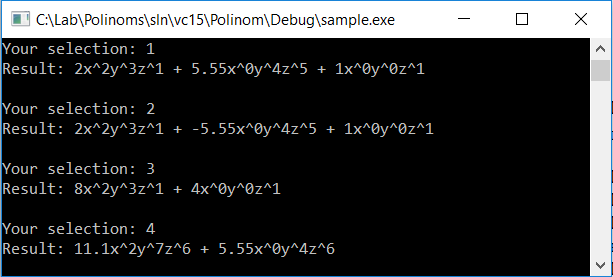
Пользователю предлагается выбрать один из пунктов меню, введя соответствующее числовое значение.

Пункт\_1: на экран будет выведен результат суммы первого и второго полиномов.

Пункт\_2: на экран будет выведен результат разности первого и второго полиномов.

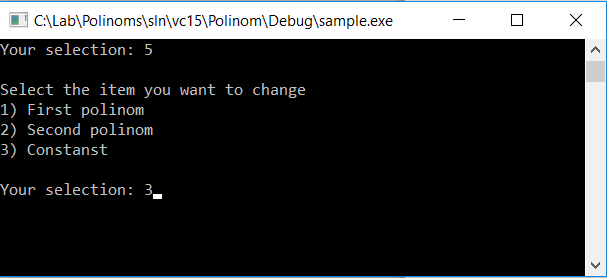
Пункт\_3: на экран будет выведен результат умножение первого полинома на константу.

Пункт\_4: на экран будет выведен результат умножение первого и второго полиномов.



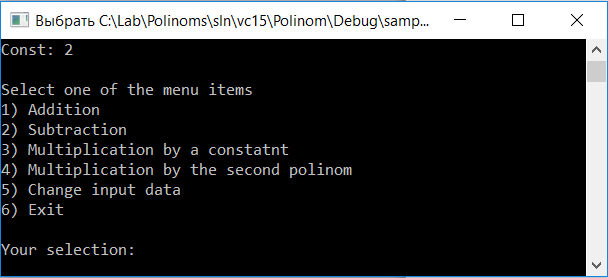
Пункт\_5: позволяет изменить введённые пользователем данные.

При выборе данного пункта на консоль будет выведено дополнительное меню.



Пользователю необходимо выбрать данные, которые он собирается изменить, выбрав один из пунктов дополнительного меню. После этого пользователь должен выполнить процедуру ввода соответствующих данных.

По окончанию ввода произойдёт выход из дополнительного меню, вывод на консоль изменённых данных и главного меню.



Пункт\_6: выход из программы.

**Руководство программиста**

**Описание структур данных**

1. *сlass Node* – структура данных, которая выступает в качестве звена другой структуры – *class*’a *List*. Данная структура реализована с использованием шаблона.

template <typename T>

class Node

{

public:

Node<T>\* prev; //указатель на предыдущее звено

Node<T>\* next; //указатель на следующее звено

T data; //данные

Node(); //конструктор по умолчанию

Node(T); //конструктор с параметром

~Node(); //деструктор

};

1. *сlass List* – структура, предназначенная для хранения однотипных данных, причём звенья хранятся в упорядоченном виде. Данная структура реализована с использованием шаблона.

template <typename T>

class List

{

protected:

Node<T>\* head; //первый элемент списка

Node<T>\* end; //последний элемент списка

public:

List(); //конструктор по умолчанию

List(Node<T>\*); //конструктор с параметром

List(List<T>&); //конструктор с параметром

~List(); //деструктор

bool IsEmpty(); //проверка списка на пустоту

void Clear(); //очистка списка

void Delete(Node<T>\*); //удаление звена из списка

void Insert(T); //вставка звена в список

List<T> Merge(List<T>); //объединение списков

List<T>& operator= (List<T>&); //присваивание

bool operator== (const List<T>&) const; //сравнение списков

virtual void Print(); //печать списка

};

1. *сlass Monom* – структура данных предназначенная для представления монома, причём степень монома представлена в виде трёхзначного числа, где число сотен – это степень при переменной “x”, число десятков - степень при переменной “y”, число единиц - степень при переменной “z”. Таким образом, поле deg не может быть больше 999.

class Monom

{

public:

double coef; //коэффициент монома

int deg; //степень монома

Monom(); //конструктор по умолчанию

Monom(double, int); //конструктор с параметром

~Monom(); //деструктор

int RetSum(); //вернуть сумму цифр степени

bool operator>= (Monom&); //сравнение

bool operator< (Monom&); //сравнение

bool operator!= (const Monom&)const; //сравнение

friend std::ostream& operator<< (std::ostream& os, const Monom& v); //перегрузка потокового вывода

};

1. *сlass Polinom* – структура данных, построенная на базе структуры *List* с конкретным типовым параметром – *Monom* (является наследником класса *List*), предназначена для представления полиномов.

class Polinom : public List<Monom>

{

public:

Polinom(); //конструктор по умолчанию

~Polinom(); //деструктор

void ToStandartView(); //приведение подобных и удаление мономов с нулевым коэффициентом

Polinom& operator= (Polinom&); //присваивание

Polinom& operator= (List<Monom>&); //присваивание

Polinom operator+ (Polinom&); //сложение полиномов

Polinom operator- (Polinom&); //разность полиномов

Polinom operator\* (double); //умножение полинома на константу

Polinom operator\* (Monom&); //умножение полинома на моном

Polinom operator\* (Polinom&); //умножение полиномов

void Print(); //печать полинома

};

**Описание алгоритмов**

*1. Алгоритм слияния двух списков*

Вход: Список

1. Создаём пустой результирующий список, создаём фиктивную голову (fk\_head), устанавливаем «указатель на последний элемент» (last) на фиктивную голову, устанавливаем два указателя (cur1 и cur2) на начала двух списков
2. Если списки пустые, то возвращаем пустой список
3. Пока оба указателя cur1 или cur2 не нулевые
4. Сравниваем данные звеньев на которые указывают cur1 и cur2 и запоминаем, кто указывает на бо̀льшие (b\_dat), а кто на меньшие (s\_dat) данные
5. Тот указатель, который указывает на большие данные (cur1 или cur2) переставляем на следующее звено списка
6. Копируем звено по указателю s\_dat
7. Вставляем копию в конец результирующего списка, устанавливаем last на копию
8. Запоминаем какой из указателей (cur1 или cur2) ненулевой в no\_empty
9. Пока no\_empty не нулевой
10. Копируем звено по указателю no\_empty
11. Вставляем копию в конец результирующего списка, устанавливаем last на копию
12. no\_empty переставляем на следующее звено списка
13. Устанавливаем голову (как «следующее звено» фиктивной головы) и конец (как last) результирующего списка

Выход: Результирующий список

2. *Алгоритм вставки звена в список*

Вход: Данные

1. Создаём звено с входными данными
2. Устанавливаем указатель cur на начало списка
3. Если список пуст, то устанавливаем голову и конец списка на созданное звено
4. Иначе:
5. Если данные головы списка меньше входных данных, то вставляем созданное звено перед головой и устанавливаем голову списка на созданное звено
6. Пока cur не нулевой и не была произведена вставка
7. Если данные звена, на которое указывает cur меньше входных данных, то устанавливаем созданное звено перед звеном cur
8. Иначе: Переставляем указатель cur на следующее звено списка
9. Если вставка не была произведена, то вставляем созданное звено в конец списка и устанавливаем конец списка на созданное звено

Выход: Нет возвращаемого результата

3. *Алгоритм приведения полинома к «стандартному виду»*

Вход: Нет входных данных

1. Устанавливаем указатель cur на первый моном
2. Пока cur не нулевой
3. Если коэффициент текущего монома (монома на который указывает cur) равен нулю, то удалить моном
4. Иначе:
5. Если есть следующий моном и степени текущего монома и следующего совпадают, то сложим их коэффициенты и запишем результат в коэффициент текущего монома
6. Иначе: Переставляем cur на следующее звено

Выход: Нет возвращаемого результата

*4. Алгоритм сложения двух полиномов*

Вход: Ссылка на полином (второй операнд)

1. Создаём пустой результирующий полином (res)
2. Вызываем метод «Слияние двух списков», результат которого запоминаем в res
3. Приводим res к стандартному виду

Выход: Результирующий полином

*5. Алгоритм умножения полинома на константу*

Вход: Константа

1. Создаём пустой результирующий полином (res)
2. Если константа нулевая возвращаем пустой полином
3. Копируем исходный полином в результирующий полином
4. Устанавливаем указатель (cur) на начало полинома
5. Пока cur не нулевой
6. Умножаем коэффициент каждого монома на входную константу
7. Переставляем cur на следующее звено

Выход: Результирующий полином

*6. Алгоритм вычитания двух полиномов*

Вход: Ссылка на полином (второй операнд)

1. Создаём пустой результирующий полином (res)
2. Входной полином умножаем на -1 и складываем с исходным полиномом, при этом запоминаем полученный результат в res

Выход: Результирующий полином

*7. Алгоритм умножения полинома на моном*

Вход: Моном

1. Создаём пустой результирующий полином, создаём фиктивную голову (fk\_head), устанавливаем «указатель на последний элемент» (last) на фиктивную голову, устанавливаем указатель cur на первый моном
2. Если коэффициент входного монома равен нулю, то возвращаем пустой полином
3. Пока cur не нулевой
4. Создаём моном
5. Коэффициент созданного монома устанавливаем как произведение коэффициентов текущего и входного мономов
6. Степень созданного монома устанавливаем как сумму степеней текущего и входного мономов
7. Если степень хотя бы одной из переменных стала больше 9, то выбрасываем исключение
8. Устанавливаем созданный моном в конец результирующего полинома и переставляем на него указатель last
9. Устанавливаем голову (как «следующее звено» фиктивной головы) и конец (как last) результирующего полинома.

Выход; Результирующий полином

*8. Алгоритм умножения двух полиномов*

Вход: Ссылка на полином (второй операнд)

1. Создаём пустой результирующий полином (res)
2. Устанавливаем указатель (cur) на начало входного полинома
3. Пока cur не нулевой
4. В res запоминаем результат слияния res и исходного полинома, умноженного на текущий моном (моном по указателю cur)
5. Переставляем cur на следующее звено
6. Приводим res к стандартному виду

Выход: Результирующий полином

*Примечание: реализацию вышеизложенных алгоритмов смотри в Приложении.*

**Описание компонентов программного комплекса**

Программа состоит из следующих основных файлов:

1. list.h – содержит интерфейс и реализацию методов шаблонов классов Node и List
2. polinom.h – содержит интерфейсы классов Monom и Polinom
3. polinom.cpp – содержит реализацию методов классов Monom и Polinom
4. test\_list.cpp – содержит набор юнит-тестов для классов Node и List
5. test\_polinom.cpp – содержит набор юнит-тестов для классов Monom и Polinom
6. main.cpp – содержит пользовательское приложение
7. test\_main.cpp – обеспечивает запуск google tests

**Заключение**

В данной лабораторной работе реализован программный комплекс, способный выполнять элементарные арифметические операции над полиномами трёх переменных, причём степень каждой переменной является целым числом от 0 до 9.

Достигнутые результаты:

1. Реализация требуемых арифметических операций (сложение, вычитание, умножение полиномов, умножение полинома на константу)
2. Корректное выполнение арифметических операций
3. Реализация тестов для проверки корректности работы программы
4. Диалоговое пользовательское приложение

**Литература**

1. Гергель В.П. Рабочие материалы преподавателя по общему курсу “Методы программирования” – Нижний Новгород – 2002. – 100 с.
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. М.: МЦНМО, 1999.-960 с., 263 ил.

**Приложение**

**Реализация методов класса List:**

1. *Слияние двух списков*

template< typename T>

List<T> List<T>::Merge(List<T> lst)

{

List<T>res;

Node<T>\*cur1 = head;

Node<T>\*cur2 = lst.head;

Node<T>\*last;

Node<T>\* b\_dat, \*s\_dat;

Node<T>\* no\_empty;

Node<T> fk\_head;

last = &(fk\_head);

if (cur1 == NULL && cur2 == NULL)

{

return(res);

}

while (cur1 != NULL && cur2 != NULL)

{

Node<T>\* temp = new Node<T>;

if (cur1->data >= cur2->data)

{

b\_dat = cur1;

s\_dat = cur2;

cur1 = cur1->next;

}

else

{

b\_dat = cur2;

s\_dat = cur1;

cur2 = cur2->next;

}

temp->data = b\_dat->data;

last->next = temp;

temp->prev = last;

last = temp;

}

if (cur1 == NULL)

{

no\_empty = cur2;

}

else

{

no\_empty = cur1;

}

while (no\_empty != NULL)

{

Node<T>\* temp = new Node<T>;

temp->data = no\_empty->data;

last->next = temp;

temp->prev = last;

last = temp;

no\_empty = no\_empty->next;

}

res.head = fk\_head.next;

res.head->prev = NULL;

res.end = last;

return(res);

}

1. *Вставка звена в список*

template<typename T>

void List<T>::Insert(T \_data)

{

Node<T>\* temp = new Node<T>;

temp->data = \_data;

Node<T>\* cur = head;

bool insrt = false;

if (cur == NULL)

{

head = end = temp;

}

else

{

if (cur->data < \_data)

{

temp->next = cur;

cur->prev = temp;

head = temp;

cur = cur->next;

insrt = true;

}

while (cur != NULL && insrt == false)

{

if (cur->data < \_data)

{

temp->next = cur;

temp->prev = cur->prev;

cur->prev = temp;

cur = temp->prev;

cur->next = temp;

cur = temp->next;

insrt = true;

}

else

{

cur = cur->next;

}

}

if (!insrt)

{

temp->prev = end;

end->next = temp;

end = temp;

}

}

}

**Реализация методов класса Polinom:**

1. *Приведение полинома к стандартному виду*

void Polinom::ToStandartView()

{

Node<Monom>\* cur = head;

Node<Monom>\* temp;

double eps = 0.00000000001;

while (cur != NULL)

{

if (abs(cur->data.coef) < eps)

{

temp = cur->next;

Delete(cur);

cur = temp;

}

else

{

if (cur->next != NULL && cur->next->data.deg == cur->data.deg)

{

cur->data.coef += cur->next->data.coef;

Delete(cur->next);

}

else

{

cur = cur->next;

}

}

}

}

1. *Сложение двух полиномов*

Polinom Polinom::operator+ (Polinom& pol)

{

Polinom res;

res = Merge(pol);

res.ToStandartView();

return(res);

}

1. *Умножение полинома на константу*

Polinom Polinom::operator\* (double \_const)

{

Polinom res;

Node<Monom>\* cur;

if (\_const == 0.0)

return (res);

res = (\*this);

cur = res.head;

while (cur != NULL)

{

cur->data.coef \*= \_const;

cur = cur->next;

}

return(res);

}

1. *Вычитание двух полиномов*

Polinom Polinom::operator- (Polinom& pol)

{

Polinom res;

res = (\*this) + pol \* (-1);

return(res);

}

1. *Умножение полинома на моном*

Polinom Polinom::operator\* (Monom& mon)

{

Polinom res;

Node<Monom>\* cur = head;

Node<Monom>\* last;

Node<Monom> fk\_head;

last = &fk\_head;

double eps = 0.00000000001;

if (abs(mon.coef) < eps)

return(res);

while (cur != NULL)

{

Node<Monom>\* temp = new Node<Monom>;

temp->data.coef = mon.coef \* cur->data.coef;

temp->data.deg = mon.deg + cur->data.deg;

if (temp->data.RetSum() != (mon.RetSum() + cur->data.RetSum()))

throw "Error: degree of x/y/z more than 9";

temp->prev = last;

last->next = temp;

last = temp;

cur = cur->next;

}

res.head = fk\_head.next;

res.head->prev = NULL;

res.end = last;

return(res);

}

1. *Произведение двух полиномов*

Polinom Polinom::operator\* (Polinom& pol)

{

Polinom res;

Node<Monom>\* cur = pol.head;

while (cur != NULL)

{

res = res.Merge((\*this) \* cur->data);

cur = cur->next;

}

res.ToStandartView();

return (res);

}