**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №6

«Полиномы»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-24б |  | преподаватель каф. ИУ5 |
| Байдаков В.М. |  |  |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2024 г.\

**Постановка задачи**

Написать программу ввода и оперирования полиномами, состоящими из термов. Для этого разработать классы Term и Polynomial, описание которых представлено ниже.

**Ввод полинома[​](https://iu5edu.ru/wiki/cpp2/docs/labs/lab6/" \l "%D0%B2%D0%B2%D0%BE%D0%B4-%D0%BF%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B0" \o "Прямая ссылка на Ввод полинома)**

* Термы полинома могут вводиться в любом порядке.
* Во вводимом терме может присутствовать отрицательные коэффициенты, например, -1.
* Терм (член полинома одного порядка) может складываться с другим термом. Например: 3x^2 – x^2, -3x^2 + x^2.
* Пробелы при вводе могут появляться где угодно

**Разработка алгоритма**

Класс Term:

int mult - множитель члена терма

int exp - степень члена терма

Методы класса Term:

Term Term(int mult = 0, int exp = 0) - конструктор с параметрами для множителя и степени

Term operator+(const Term &f, const Term &s) - сложение термов, учитывающее одинаковые степени

std::istream& operator>>(std::istream &in, Term &term) - ввод терма из потока

std::ostream& operator<<(std::ostream &out, const Term &term) - вывод терма в поток

Класс Polynomial:

Term\* poly - массив термов, составляющих полином

unsigned degree - количество термов в полиноме

bool order\_ - порядок следования термов (0 - убывающий, 1 - возрастающий)

Методы класса Polynomial:

Polynomial Polynomial(int mult\_ = 0, int exp\_ = 0) - конструктор с параметрами для начального множителя и степени

Polynomial& operator+=(const Polynomial &pol) - сложение полиномов с присваиванием

Polynomial& operator\*=(const Polynomial &pol) - умножение полиномов с присваиванием

Polynomial Polynomial(const Polynomial &pol) - конструктор копирования

Polynomial& operator=(const Polynomial &pol) - оператор присваивания

std::ostream& operator<<(std::ostream &out, const Polynomial &pol) - вывод полинома в поток

std::istream& operator>>(std::istream &in, Polynomial &pol) - ввод полинома из потока

Дружественные функции:

operator+(const Polynomial &first, const Polynomial &second) - сложение полиномов

operator\*(const Polynomial &first, const Polynomial &second) - умножение полиномов

**Текст программы**

**Polynom.h**

#define MAX\_INPUT 50

#define TERM\_INPUT 64

#define POLYNOM\_INPUT 512

#include <iostream>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

class Polynomial;

class Term

{

  friend Polynomial;

  int mult;

  int exp;

public:

  Term(int mult = 0, int exp = 0) : mult(mult), exp(exp) {}

  friend Term operator+(const Term &f, const Term &s);

  friend std::istream &operator>>(std::istream &in, Term &term);

  friend std::ostream &operator<<(std::ostream &out, const Term &term);

  friend Polynomial operator+(const Polynomial &first, const Polynomial &second);

  friend Polynomial operator\*(const Polynomial &first, const Polynomial &second);

  friend std::ostream &operator<<(std::ostream &out, const Polynomial &pol);

  friend std::istream &operator>>(std::istream &in, Polynomial &pol);

};

class Polynomial

{

  Term \*poly;

  unsigned degree;

  bool order\_ = 0; // 0- по убыванию, 1- по возрастанию

public:

  Polynomial(int mult\_ = 0, int exp\_ = 0)

  {

    degree = 1;

    poly = new Term[degree];

    poly[0] = Term(mult\_, exp\_);

  }

  friend Polynomial operator+(const Polynomial &first, const Polynomial &second);

  friend Polynomial operator\*(const Polynomial &first, const Polynomial &second);

  Polynomial &operator+=(const Polynomial &pol);

  Polynomial &operator\*=(const Polynomial &pol);

  Polynomial(const Polynomial &pol);

  Polynomial &operator=(const Polynomial &pol);

  friend std::ostream &operator<<(std::ostream &out, const Polynomial &pol);

  friend std::istream &operator>>(std::istream &in, Polynomial &pol);

};

**Main.cpp**

#include "polynom.h"

#include <iostream>

int main()

{

  Term a;

  Term c;

  std::cin >> c;

  std::cin >> a;

  std::cout << "\n a+c:" << a + c;

  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  Polynomial d(3, 2);

  Polynomial e(213, 34);

  Polynomial f;

  f += e + d;

  std::cout << '\n'

            << f;

  f \*= e;

  std::cout << '\n'

            << f;

  std::cout << "\nenter polynom:";

  Polynomial r;

  std::cin >> r;

  std::cout << r;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.23)

set(project "lab6")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

        main.cpp)

add\_subdirectory(polynom)

set(${project}\_SOURCE\_LIST

        ${${project}\_SOURCES})

add\_executable(${project}

        ${${project}\_SOURCE\_LIST})

target\_link\_libraries(${project} polynom)

target\_include\_directories(

        ${PROJECT\_NAME} PUBLIC

        ${CMAKE\_SOURCE\_DIR}/polynom

)

**Polynom.cpp**

#include "polynom.h"

Term operator+(const Term &f, const Term &s)

{

  if (s.mult == 0)

  {

    return f;

  }

  else if (f.mult == 0)

  {

    return s;

  }

  if (f.exp == s.exp)

  {

    return Term(f.mult + s.mult, f.exp);

  }

  else

  {

    std::cerr << " can't slozhit' terms, first term returned ";

    return f;

  }

}

std::istream &operator>>(std::istream &in, Term &term)

{

  char input[TERM\_INPUT];

  in.getline(input, TERM\_INPUT);

  char finalws[TERM\_INPUT];

  int finalIndex = 0;

  for (size\_t i = 0; input[i] != '\0'; ++i)

  {

    if (input[i] != ' ')

    {

      finalws[finalIndex++] = input[i];

    }

  }

  finalws[finalIndex] = '\0';

  term.mult = 0;

  term.exp = 0;

  unsigned len = strlen(finalws);

  int xIndex = -1;

  int stIndex = -1;

  for (size\_t i = 0; i < len; ++i)

  {

    if (finalws[i] == 'x')

    {

      xIndex = i;

      break;

    }

  }

  if (xIndex != -1)

  {

    for (size\_t i = xIndex + 1; i < len; ++i)

    {

      if (finalws[i] == '^')

      {

        stIndex = i;

        break;

      }

    }

  }

  if (xIndex == -1)

  {

    term.mult = atoi(finalws);

  }

  else if (xIndex == 0 || (xIndex == 1 && (finalws[0] == '-' || finalws[0] == '+')))

  {

    term.mult = (finalws[0] == '-') ? -1 : 1;

  }

  else

  {

    char coeffPart[TERM\_INPUT];

    strncpy(coeffPart, finalws, xIndex);

    coeffPart[xIndex] = '\0';

    term.mult = atoi(coeffPart);

  }

  if (xIndex != -1 && stIndex == -1)

  {

    term.exp = 1;

  }

  else if (stIndex != -1)

  {

    char expPart[TERM\_INPUT];

    strcpy(expPart, finalws + stIndex + 1);

    term.exp = atoi(expPart);

  }

  return in;

}

std::ostream &operator<<(std::ostream &out, const Term &term)

{

  if (term.exp == 0)

  {

    out << term.mult;

  }

  else

  {

    if (term.mult == -1)

    {

      out << "-x";

    }

    else if (term.mult < 0)

    {

      out << term.mult << "x";

    }

    else if (term.mult == 1)

    {

      out << "x";

    }

    else

    {

      out << term.mult << "x";

    }

    if (term.exp != 1)

    {

      out << "^" << term.exp;

    }

  }

  return out;

}

Polynomial operator+(const Polynomial &first, const Polynomial &second)

{

  Polynomial final;

  final.poly = new Term[first.degree + second.degree];

  int i1 = 0, i2 = 0, i = 0;

  while (i1 < first.degree && i2 < second.degree)

  {

    if (first.poly[i1].exp < second.poly[i2].exp)

    {

      final.poly[i++] = first.poly[i1++];

    }

    else if (first.poly[i1].exp == second.poly[i2].exp)

    {

      final.poly[i++] = first.poly[i1++] + second.poly[i2++];

    }

    else

    {

      final.poly[i++] = second.poly[i2++];

    }

  }

  while (i1 < first.degree)

  {

    final.poly[i++] = first.poly[i1++];

  }

  while (i2 < second.degree)

  {

    final.poly[i++] = second.poly[i2++];

  }

  final.degree = i;

  return final;

}

Polynomial operator\*(const Polynomial &first, const Polynomial &second)

{

  Polynomial final;

  for (int i = 0; i < first.degree; i++)

  {

    for (int j = 0; j < second.degree; j++)

    {

      final += Polynomial(first.poly[i].mult \* second.poly[j].mult, first.poly[i].exp + second.poly[j].exp);

    }

  }

  return final;

}

Polynomial &Polynomial::operator+=(const Polynomial &pol)

{

  Polynomial final = pol + \*this;

  \*this = final;

  return \*this;

}

Polynomial &Polynomial::operator\*=(const Polynomial &pol)

{

  Polynomial final = pol \* \*this;

  \*this = final;

  return \*this;

}

Polynomial::Polynomial(const Polynomial &pol)

{

  degree = pol.degree;

  poly = new Term[degree];

  for (size\_t i = 0; i < degree; i++)

  {

    poly[i] = pol.poly[i];

  }

}

Polynomial &Polynomial::operator=(const Polynomial &pol)

{

  degree = pol.degree;

  delete[] poly;

  poly = new Term[degree];

  for (int i = 0; i < degree; i++)

  {

    poly[i] = pol.poly[i];

  }

  return \*this;

}

std::ostream &operator<<(std::ostream &out, const Polynomial &pol)

{

  bool first = true;

  if (pol.order\_)

  {

    for (int i = 0; i < pol.degree; ++i)

    {

      if (pol.poly[i].mult != 0)

      {

        if (!first && pol.poly[i].mult > 0)

          out << " + ";

        out << pol.poly[i];

        first = false;

      }

    }

  }

  else

  {

    for (int i = pol.degree - 1; i >= 0; --i)

    {

      if (pol.poly[i].mult != 0)

      {

        if (!first && pol.poly[i].mult > 0)

          out << " + ";

        out << pol.poly[i];

        first = false;

      }

    }

  }

  if (first)

    out << "0";

  return out;

}

std::istream &operator>>(std::istream &in, Polynomial &pol)

  {

    char input[POLYNOM\_INPUT];

    in.getline(input, POLYNOM\_INPUT); // ввод полинома

    char finalws[POLYNOM\_INPUT];

    unsigned finalIndex = 0;

    for (size\_t i = 0; input[i] != '\0'; ++i) // удаление пробелов

    {

      if (input[i] != ' ')

      {

        finalws[finalIndex++] = input[i];

      }

    }

    finalws[finalIndex] = '\0';

    int startIndex = 0;

    int termIndex = 0;

    char termString[TERM\_INPUT]; // текущий терм для обработки

    char coeffPart[TERM\_INPUT];  // его коэффицент

    char expPart[TERM\_INPUT];    // его степень

    Term term;

    for (unsigned i = 0; i < strlen(finalws) + 1; ++i)

    {

      if (finalws[i] == '+' || finalws[i] == '-' || finalws[i] == '\0') // ищем либо знак первого терма, либо, при отсутствии знака у первого терма ищем знак у второго и кидаем все что от 0 до i

      {

        strncpy(termString, finalws + startIndex, i - startIndex);

        termString[i - startIndex] = '\0';

        // далее обрабатываем терм

        int xIndex = -1;

        int expIndex = -1;

        for (int j = 0; termString[j] != '\0'; ++j)

        {

          if (termString[j] == 'x')

          {

            xIndex = j;

            break;

          }

        }

        term.exp = 0;

        term.mult = 0;

        if (xIndex == -1) // х не найден соответственно степень нулевая и искать ее нет смысла

        {

          term.mult = atoi(termString);

        }

        else

        {

          if (xIndex == 0 || (xIndex == 1 && (termString[0] == '-' || termString[0] == '+'))) // если нет коэффицента

          {

            term.mult = (termString[0] == '-') ? -1 : 1;

          }

          else

          {

            strncpy(coeffPart, termString, xIndex);

            coeffPart[xIndex] = '\0';

            term.mult = atoi(coeffPart);

          }

          for (int j = xIndex + 1; termString[j] != '\0'; ++j) // поиск степени

          {

            if (termString[j] == '^')

            {

              expIndex = j;

              break;

            }

          }

          if (expIndex == -1)

          {

            term.exp = 1;

          }

          else

          {

            strncpy(expPart, termString + expIndex + 1, strlen(termString) - expIndex - 1);

            expPart[strlen(termString) - expIndex - 1] = '\0';

            term.exp = atoi(expPart);

          }

        }

        pol += Polynomial(term.mult, term.exp);

        startIndex = i;

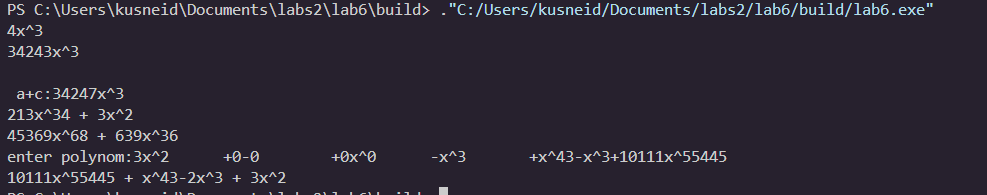
      }

    }

    return in;

  }

**Анализ результатов**

****