Общее задание:

- *Составить на языке C# описание классов для указанных объектов.*
- *В среде Visual Studio разработать консольную программу, иллюстрирующую использование объектов заданных классов.*

Индивидуальное задание:

Многочлен вида P(x,sin(x),cos(x)) на основе связного списка (сложение и умножение).

Описание работы программы:

В результате запуска программы всплывает окошко с выбором трех действий: сложить многочлены, перемножить многочлены и завершить программу.

Выбирая первое или второе действие, вы далее вводите ключевые числа для правильной работы программы. Одним из первых чисел является число элементарных многочленов вида P(sin(x),cos(x),x), затем поочередно вводятся значение коэффициента перед элементарным многочленом, далее степени x, sin и cos. После ввода первого многочлена требуется ввести то же самое для второго, т.е. элементарные многочлены, их число и ключевые цифры.

После ввода исходных данным программа вычисляет сумму или произведение этих самых многочленов.

Программа циклична и может выполнятся сколько угодно, если пользователь сам не решит покинуть консоль, нажав соответствующую кнопку.

Алгоритмы выполнения основных операций на псевдокоде:

- *Суммирование многочленов:*

P1, P2 - многочлены

Складываем элементарные многочлены многочленов P1\P2 , если такое возможно.

Объявляем результирующий многочлен -- Res

ЦИКЛ по списку P1:

Объявление флажка f “В результирующем многочлене есть слагаемое с такими степенями”, опускаем его.

ЦИКЛ по списку P2:

ЕСЛИ степени x, sin(x),cos(x) элементарного многочлена P1 совпадает со степенями x, sin(x), cos(x) слагаемого P2

ТО коэффициент слагаемого nNew есть сумма коэффициентов слагаемого P1 и P2

Добавляем nNew в результирующий многочлен.

Поднять флаг f

Переставить указатель слагаемого P2 на следующее

ЕСЛИ флаг f не поднят

ТО добавляем к Res текущее слагаемое P1

Переставляем указатель слагаемого P2 на следующий элемент

КЦ

ЦИКЛ по списку P2:

Объявление флажка f “В результирующем многочлене есть слагаемое с такими степенями”, опускаем его.

ЦИКЛ по списку P1:

ЕСЛИ степени x, sin(x), cos(x), слагаемого P1 совпадает со степенями x, sin(x),cos(x) слагаемого P2

ТО поднимаем флаг f

Переставляем указатель слагаемого P1 на следующее

ЕСЛИ флаг f не поднят

ТО добавляем к Res текущее слагаемое P2

Переставляем указатель слагаемого P2 на следующее

ВЫВОД: Res

- *Умножение многочленов:*

P1, P2 - многочлены

Складываем элементарные многочлены многочленов P1\P2 , если такое возможно.

Объявляем результирующий многочлен -- Res

ЦИКЛ по списку P1

Установка флага f “В Res есть слагаемое с такими степенями”, опускаем его

ЦИКЛ по списку P2

Перемножаем соответствующие многочлены и заносим результат в nNew.

Добавляем nNew в Res

Поднимаем флаг f

ЕСЛИ флаг f не поднят

ТО добавим к Res текущее слагаемое P1

Переставляем указатель слагаемого P2 на следующее;

Переставляем указатель слагаемого P1 на следующий

ЦИКЛ по списку P2

Установка флага f “В Res есть слагаемое с такими степенями”, опускаем его.

ЦИКЛ по списку P1

nNew добавим в Res

Поднимем флаг f

ЕСЛИ флаг не поднят

ТО добавить к Res текущее слагаемое P2

Переставляем указатель слагаемого P1 на следующее;

Переставим указатель слагаемого P2 на следующее

ВЫВОД: Res

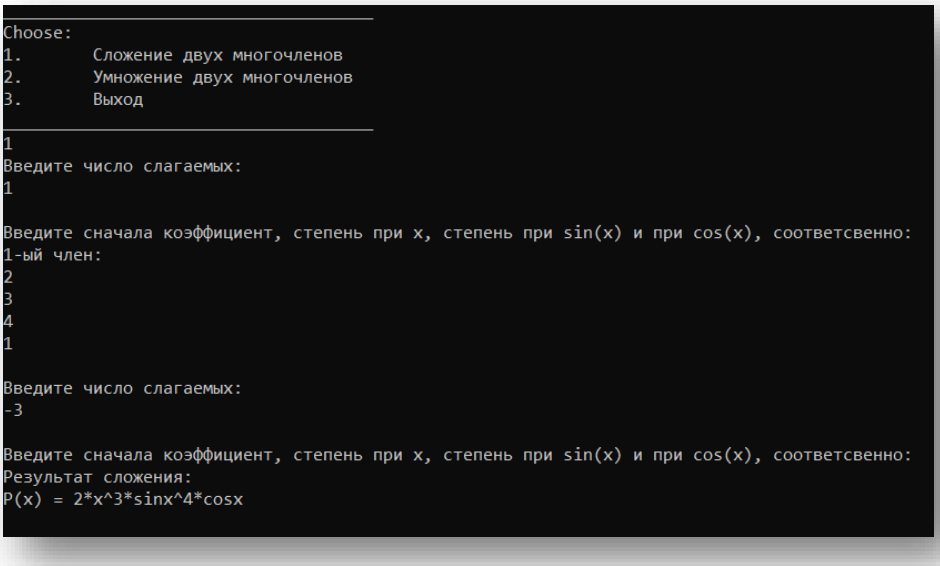
Тесты:

Номер теста, N	*Входные данные*	*Выходные данные*
1	2*x^3*sinx*cosx^4 + -3*x^3*sinx*cosx^4	-1*x^3*sinx*cosx^4
2	-1*x^2*sinx^3*cosx^3 + 3*x*sinx^4*cosx + -3*x*sinx^4*cosx	-1*x^2*sinx^3*cosx^3+
3	(3*x^2*sinx^2*cosx^2 + 1*cosx^3) * 3*x^5	9*x^7*sinx^2*cosx^2+3*x^ 5*cosx^3
4	2*sinx*cosx^3 * (- 3*x^2*sinx*cosx^3+3*x^2* sinx*cosx^3)	0
5	(1*x*sinx – 1*cosx)	1*x^3*sinx^3+++-1*cosx^3

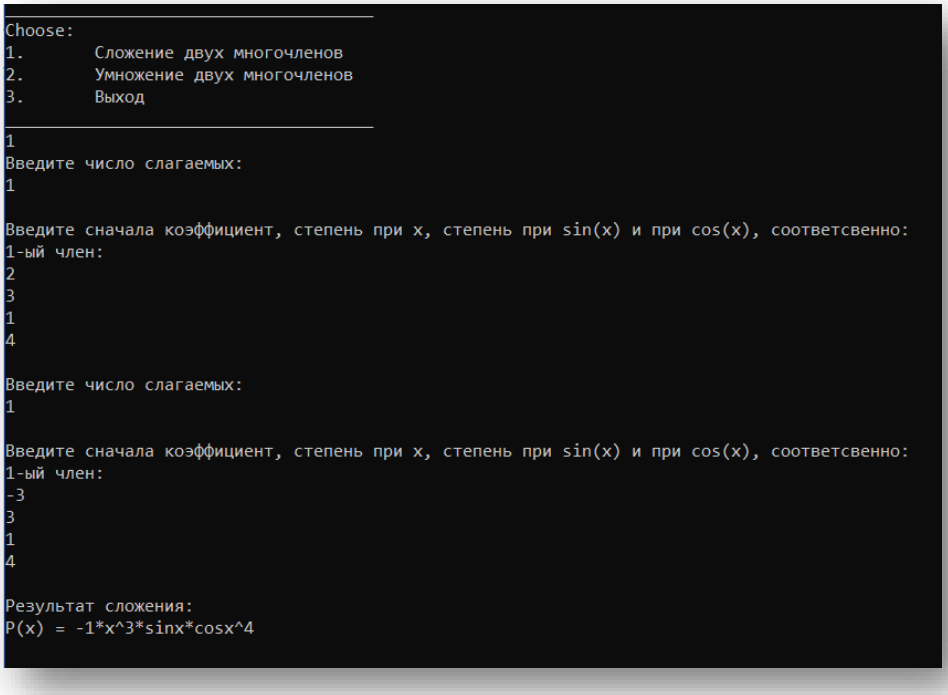
	* (x^2*sinx^2 +x*sinx*cosx + cosx^2)	

Распечатки работы программы:

Тест -- 1



Тест -- 2



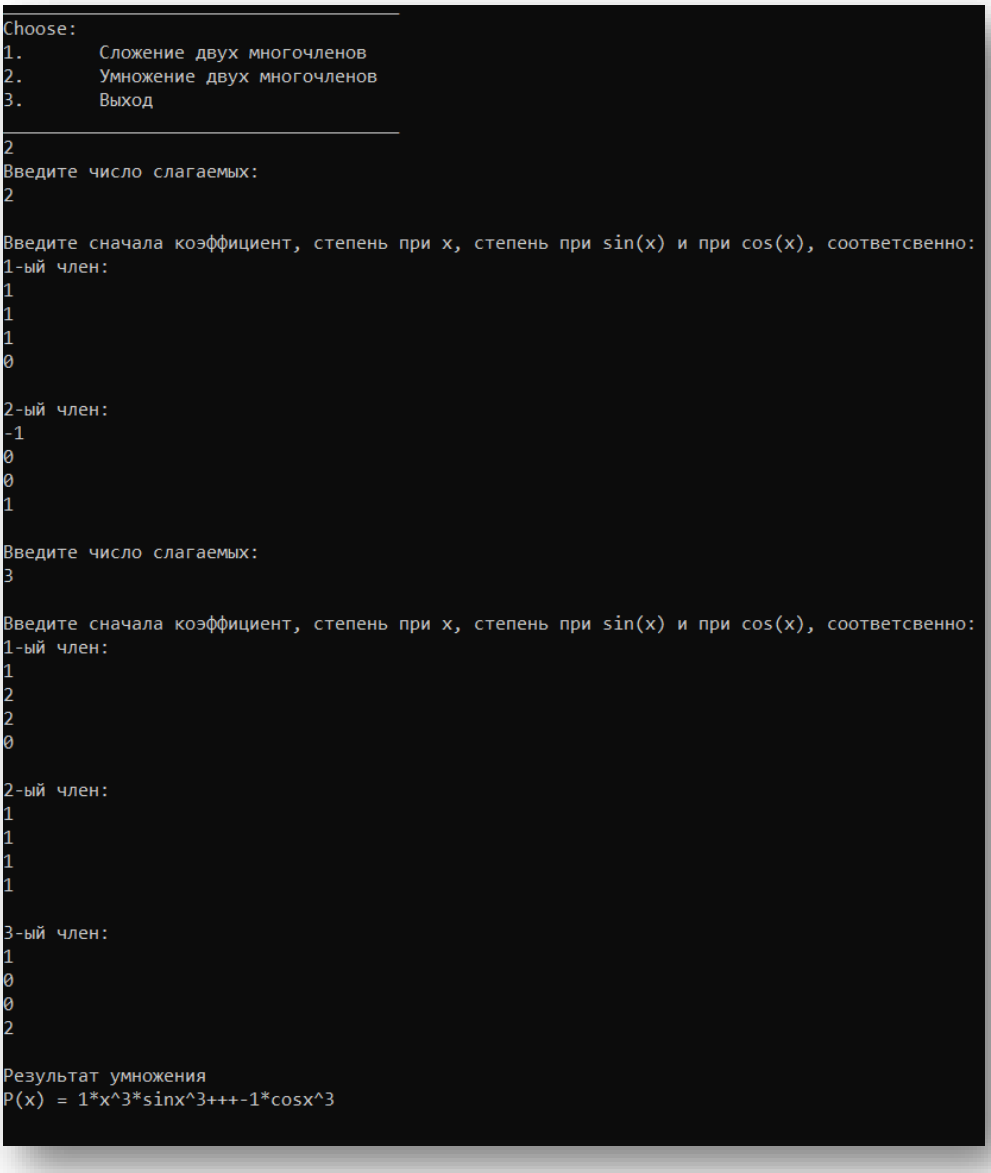
Тест -- 3

```
_____________________________________
Choose:
1.      Сложение двух многочленов
2.      Умножение двух многочленов
3.      Выход
_____________________________________
1
Введите число слагаемых:
2

Введите сначала коэффициент, степень при x, степень при sin(x) и при cos(x), соответсвенно:
1-ый член:
-1
2
3
3

2-ый член:
3
1
4
1

Введите число слагаемых:
1

Введите сначала коэффициент, степень при x, степень при sin(x) и при cos(x), соответсвенно:
1-ый член:
-3
1
4
1

Результат сложения:
P(x) = -1*x^2*sinx^3*cosx^3+
```

Тест -- 4

```
_____________________________________
Choose:
1.      Сложение двух многочленов
2.      Умножение двух многочленов
3.      Выход
_____________________________________
2
Введите число слагаемых:
1

Введите сначала коэффициент, степень при x, степень при sin(x) и при cos(x), соответсвенно:
1-ый член:
2
0
1
3

Введите число слагаемых:
2

Введите сначала коэффициент, степень при x, степень при sin(x) и при cos(x), соответсвенно:
1-ый член:
-3
2
1
3

2-ый член:
3
2
1
3

Результат умножения
P(x) = 0
```

Тест -- 5



Листинг программы:

- *class Program.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab5
        {
                class Program
        {
            static void Main(string[] args)
            {
```

```
            string choice;
            for (; ; )
            {
                Console.WriteLine("_____________________________________");
                Console.WriteLine("Choose: ");
                Console.WriteLine("1.\t Сложение двух многочленов");
                Console.WriteLine("2.\t Умножение двух многочленов");
                Console.WriteLine("3.\t Выход");
                Console.WriteLine("_____________________________________");
                choice = Console.ReadLine();
                PolyFunc PFunc = new PolyFunc();
                switch (choice)
                {
                    case "1":
                        Polynomial P1 = new Polynomial();
                        P1.Input();
                        Polynomial P2 = new Polynomial();
                        P2.Input();
                        Console.WriteLine("Результат сложения: ");
                        PFunc.Sum(P1, P2).Output();
                        P1.Clean();
                        P2.Clean();
                        break;
                    case "2":
                        Polynomial P3 = new Polynomial();
                        P3.Input();
                        Polynomial P4 = new Polynomial();
                        P4.Input();
                        Console.WriteLine("Результат умножения");
                        PFunc.Multyply(P3, P4).Output();
                        P3.Clean();
                        P4.Clean();
                        break;
                    case "3":
                        return;
                }
            }
        }
    }
    }
```

- *class NODES.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab5
    {
            class NODE
    {
```

```
        private double Value;
        private int Pow_x;
        private int Pow_sin_x;
        private int Pow_cos_x;
        private NODE next;
        public NODE() { }
        public NODE(double val, int px, int psinx, int pcosx)
        {
            Value = val;
            Pow_x = px;
            Pow_sin_x = psinx;
            Pow_cos_x = pcosx;
        }
        public double Get_value()
        {
            return Value;
        }
        public void Set_value(double val)
        {
            Value = val;
        }
        public int Get_pow_x()
        {
            return Pow_x;
        }
        public int Get_pow_cos_x()
        {
            return Pow_cos_x;
        }
        public int Get_pow_sin_x()
        {
            return Pow_sin_x;
        }
        public NODE Get_next()
        {
            return this.next;
        }
        public void Set_next(NODE node)
        {
            this.next = node;
        }
    }
    }
```

- *class POLYNOMIALS.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab5
```

```
        {
                class Polynomial
        {
            private NODE first;
            private NODE last;
            public void Add(NODE node)
            {
                if (this.first == null)
                {
                    first = node;
                }
                else
                {
                    this.last.Set_next(node);
                }
                last = node;
            }
            public void Input()
            {
                double Value;
                int Pow_x;
                int Pow_sin_x;
                int Pow_cos_x;
                Console.WriteLine("Введите число слагаемых: ");
                int n = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine();
                Console.WriteLine("Введите сначала коэффициент, степень при x, степень
при sin(x) и при cos(x), соответсвенно: ");
                for (int i = 0; i < n; i++)
                {
                    Console.WriteLine((i + 1) + "-ый член: ");
                    Value = Convert.ToDouble(Console.ReadLine());
                    Pow_x = Convert.ToInt32(Console.ReadLine());
                    Pow_sin_x = Convert.ToInt32(Console.ReadLine());
                    Pow_cos_x = Convert.ToInt32(Console.ReadLine());
                    NODE node = new NODE(Value, Pow_x, Pow_sin_x, Pow_cos_x);
                    this.Add(node);
                    Console.WriteLine();
                }
            }
            public void Output()
            {
                NODE p = new NODE();
                p = this.first;
                bool f = false;
                Console.Write("P(x) = ");
                while (p != null)
                {
                    if (p.Get_value() != 0)
                    {
                        f = true;
```

```
                Console.Write(p.Get_value());
                if (p.Get_pow_x() == 1)
                {
                    Console.Write("*x");
                }
                else if (p.Get_pow_x() == 0)
                {
                    Console.Write("");
                }
                else
                {
                    Console.Write("*x^" + p.Get_pow_x());
                }
                if (p.Get_pow_sin_x() == 1)
                {
                    Console.Write("*sinx");
                }
                else if (p.Get_pow_sin_x() == 0)
                {
                    Console.Write("");
                }
                else
                {
                    Console.Write("*sinx^" + p.Get_pow_sin_x());
                }
                if (p.Get_pow_cos_x() == 1)
                {
                    Console.Write("*cosx");
                }
                else if (p.Get_pow_cos_x() == 0)
                {
                    Console.Write("");
                }
                else
                {
                    Console.Write("*cosx^" + p.Get_pow_cos_x());
                }
            }
            p = p.Get_next();
            if (p != null)
            {
                Console.Write("+");
            }
        }
        if (!f)
        {
            Console.WriteLine("0");
        }
        Console.WriteLine("\n");
    }
    public void Delete_node(NODE node)
```

```
        {
            NODE prev;
            prev = first;
            while (prev.Get_next() != node)
            {
                prev = prev.Get_next();
            }
            if (node == this.first)
            {
                first = node.Get_next();
                node.Set_next(null);
            }
            else if (node == this.last)
            {
                last = prev;
                last.Set_next(null);
            }
            else
            {
                prev.Set_next(node.Get_next());
            }
        }
        public NODE Get_first()
        {
            return first;
        }
        public void Balance()
        {
            NODE n1 = this.Get_first();
            NODE n2;
            while (n1 != null)
            {
                n2 = n1.Get_next();
                while (n2 != null)
                {
                    if (n1.Get_pow_x() == n2.Get_pow_x() && n1.Get_pow_sin_x() ==
n2.Get_pow_sin_x()
                        && n1.Get_pow_cos_x() == n2.Get_pow_cos_x())
                    {
                        n1.Set_value(n1.Get_value() + n2.Get_value());
                        this.Delete_node(n2);
                    }
                    n2 = n2.Get_next();
                }
                n1 = n1.Get_next();
            }
        }
        public void Clean()
        {
            first = null;
            last = null;
```

```
            }
        }
        }
```

- *class FUNCTIONS.cs*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab5
        {
                class PolyFunc
        {
            public PolyFunc() { }
            public Polynomial Sum(Polynomial P1, Polynomial P2)
            {
                P1.Balance();
                P2.Balance();
                Polynomial Res = new Polynomial();
                NODE n1 = P1.Get_first();
                NODE n2;
                bool f;
                while (n1 != null)
                {
                    n2 = P2.Get_first();
                    f = false;
                    while (n2 != null)
                    {
                        if (n1.Get_pow_x() == n2.Get_pow_x() && n1.Get_pow_sin_x() ==
n2.Get_pow_sin_x()
                            && n1.Get_pow_cos_x() == n2.Get_pow_cos_x())
                        {
                            NODE n_new = new NODE((n1.Get_value() + n2.Get_value()),
n1.Get_pow_x(), n1.Get_pow_sin_x(), n1.Get_pow_cos_x());
                            Res.Add(n_new);
                            f = true;
                            break;
                        }
                        n2 = n2.Get_next();
                    }
                    if (!f)
                    {
                        NODE n_new = new NODE(n1.Get_value(), n1.Get_pow_x(),
n1.Get_pow_sin_x(), n1.Get_pow_cos_x());
                        Res.Add(n_new);
                    }
                    n1 = n1.Get_next();
                }
                n2 = P2.Get_first();
                while (n2 != null)
```

```
                {
                    n1 = P1.Get_first();
                    f = false;
                    while (n1 != null)
                    {
                        if (n1.Get_pow_x() == n2.Get_pow_x() && n1.Get_pow_sin_x() ==
n2.Get_pow_sin_x()
                            && n1.Get_pow_cos_x() == n2.Get_pow_cos_x())
                        {
                            f = true;
                            break;
                        }
                        n1 = n1.Get_next();
                    }
                    if (!f)
                    {
                        NODE n_new = new NODE(n2.Get_value(), n2.Get_pow_x(),
n2.Get_pow_sin_x(), n2.Get_pow_cos_x());
                        Res.Add(n_new);
                    }
                    n2 = n2.Get_next();
                }
                return Res;
            }
            public Polynomial Multyply(Polynomial P1, Polynomial P2)
            {
                Polynomial Res = new Polynomial();
                NODE n1 = P1.Get_first();
                NODE n2;
                bool f;
                while (n1 != null)
                {
                    n2 = P2.Get_first();
                    f = false;
                    while (n2 != null)
                    {
                        NODE nNew = new NODE(n1.Get_value() * n2.Get_value(),
n1.Get_pow_x() + n2.Get_pow_x(), n1.Get_pow_sin_x() + n2.Get_pow_sin_x(),
n1.Get_pow_cos_x() + n2.Get_pow_cos_x());
                        Res.Add(nNew);
                        f = true;
                        n2 = n2.Get_next();
                    }
                    if (!f)
                    {
                        NODE newNode = new NODE(n1.Get_value(), n1.Get_pow_x(),
n1.Get_pow_sin_x(), n1.Get_pow_cos_x());
                        Res.Add(newNode);
                    }
                    n1 = n1.Get_next();
                }
```

```
                n2 = P2.Get_first();
                while (n2 != null)
                {
                    n1 = P1.Get_first();
                    f = false;
                    while (n1 != null)
                    {
                        f = true;
                        n1 = n1.Get_next();
                    }
                    if (!f)
                    {
                        NODE newNode = new NODE(n2.Get_value(), n2.Get_pow_x(),
n2.Get_pow_sin_x(), n2.Get_pow_cos_x());
                        Res.Add(newNode);
                    }
                    n2 = n2.Get_next();
                }
                Res.Balance();
                return Res;
            }
        }
        }
```