Lista jednokierunkowa cykliczna, składa się z początku, końca oraz dodatkowego pola mówiącego o jej długości

```
public List()
    {
      head = null;
      tail = null;
      lenght = 0;
    }
```

Inicjalizowanego za pomocą metody GetLenght()

```
public void GetLenght()
    {
      Element current = head;
      while (current != null)
      {
        lenght = lenght + 1;
        current = current.next;
      }
    }
```

Element Listy opisuje klasa Element, której konstruktor wywoływany jest z argumentem wartości współczynnika, wykładnika oraz wskaźnikiem na następny element. Jeśli następny element nie istnieje, pozostaje ze wskaźnikiem na null

```
public Element(int fac, int i)
    {
      factor = fac;
      index = i;
      next = null;
    }
```

Dwie podstawowe operacje: dodawanie do listy nowych elementów

```
public void Add(int factor, int index)
    {
      Element el = new Element(factor, index);
      if(head == null)
      {
        head = tail = el;
      }
      else
      {
        tail.next = el;
        tail = el;
      }
    }
```

oraz usuwanie istniejących. Dla obu przypadków, szukamy po wartości współczynnika oraz wykładnika.

```
public bool Remove(int factor, int index)
    {
        Element current = head;
        Element before = null;


        if (head == null)
        {
            return false;
        }


        while (current != null && current.factor != factor)
        {
            before = current;
            current = current.next;
        }


        if (current == null)
        {
            return false;
        }


        if(current == head)
        {
            head = current.next;
            return true;
        }


        if(current.next == null)
        {
            tail = before;
            before.next = null;
            return true;
        }


        before.next = current.next;
        return true;
    }
```

Klasa PolynomialsOperations służy do operacji na wielomianach, czyli dodawania, odejmowania oraz mnożenia wielomianów. Do prawidłowej obsługi wszystkich tych operacji potrzebowaliśmy metod pomocniczych – metody obliczeniowej dla dodawania i odejmowania, metody sortującej wielomian według wysokości wykładnika oraz sklejającej dwa wielomiany w jedno działanie / wyrażenie.

W metodzie sumującej wykorzystaliśmy wszystkie metody pomocnicze. Najpierw łączymy w jedno wyrażenie oba wielomiany, następnie je sortujemy według wykładnika, obliczamy te same indeksy,

inaczej mówiąc skracamy, następnie wynik sklejamy z pierwszym argumentem 0^{-1} , następnie taki wynik trafia już bezpośrednio na konsolę.

```
public List Addition(List Px, List Qx)
{
   List Wx = GlueExpressions(Px, Qx);
   List firstArgument = new List();
   firstArgument.Add(0, -1);
   var sortedExpressions = SortAscending(Wx);
   var shortedExpressions = CalculateTheSameIndexes(sortedExpressions);
   var result = GlueExpressions(firstArgument, shortedExpressions);
   return result;
}
```

W odejmowaniu robimy dokładnie to samo – z tym wyjątkiem, że zanim przejdziemy do tych operacji drugi wielomian otrzymuje odwrócone znaki współczynników, czyli tak jak byśmy odejmowali nawias.

```
public List Substraction(List Px, List Qx)
{
   List firstArgument = new List();
   firstArgument.Add(0, -1);


   Element el = Qx.head;
   List QxCopy = new List();
   while(el != null)
   {
      Element newEl = new Element(el.factor * -1, el.index);
      QxCopy.Add(newEl.factor, newEl.index);
      el = el.next;
   }
   List Wx = GlueExpressions(Px, QxCopy);
   var sortedExpressions = SortAscending(Wx);
   var shortedExpressions = CalculateTheSameIndexes(sortedExpressions);
   var result = GlueExpressions(firstArgument, shortedExpressions);
   return result;
}
```

Mnożenie jest nieco odrębne, chociaż na koniec korzystamy również z metod pomocniczych, między innymi aby skrócić, dobrze uszeregować wielomian oraz dodać pierwszy argument 0^{-1}. Wszystko co dzieje się w mnożeniu można opisać w dwóch pętlach - pierwsza pętla, iteruje się przez pierwszy wielomian, mnożąc każdy współczynnik ze współczynnikami drugiego wielomianu oraz dodając do siebie wykładniki.

```
public List Multiplication(List Px, List Qx)
{
   List firstArgument = new List();
```

```
        firstArgument.Add(0, -1);


        List multiplyResult = new List();
        Element pxEl = Px.head;
        Element qxEl = Qx.head;
        Px.GetLenght();
        Qx.GetLenght();
        for(int i = 0; i < Px.lenght; i++)
        {
           if(pxEl.factor != 0)
           {
              for(int j = 0; j < Qx.lenght; j++)
              {
                 if(qxEl.factor != 0)
                 {
                    var factor = pxEl.factor * qxEl.factor;
                    var index = pxEl.index + qxEl.index;
                    multiplyResult.Add(factor, index);

                 }
                 qxEl = qxEl.next;
              }


           }
           pxEl = pxEl.next;
           qxEl = Qx.head;
        }
        var sortedResult = SortAscending(multiplyResult);
        var shortedExpressions = CalculateTheSameIndexes(sortedResult);
        var result = GlueExpressions(firstArgument, shortedExpressions);
        return result;
     }
```

Sortowanie:

```
private List SortAscending(List Wx)
     {
        List Vx = new List();
        Wx.GetLenght();
        for (int i = 0; i < Wx.lenght; i++)
        {
           Element el = Wx.head;
           Element max = null;
           while (el != null && el.next != null)
           {
              max = el;
              var current = el.next;
              if (max.index < current.index)
```

```
            {
                max = current;
            }
            el = el.next;
        }
        if(max != null)
        {
            Wx.Remove(max.factor, max.index);
            Vx.Add(max.factor, max.index);
        }

    }


    return Vx;
}
```

Sklejanie wielomianów:

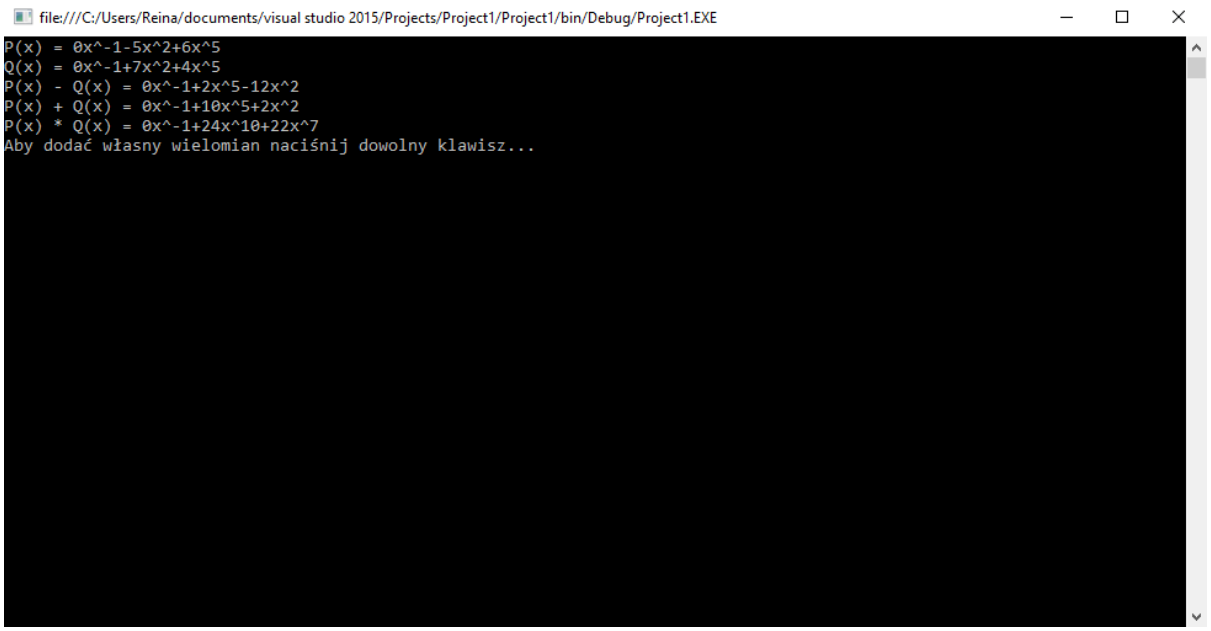
```
private static List GlueExpressions(List Px, List Qx)
    {
        List Wx = new List();
        Element el = Px.head;


        while (el != null)
        {
            Wx.Add(el.factor, el.index);
            el = el.next;
        }
        el = Qx.head;
        while (el != null)
        {
            if (el.factor == 0 && el.index == -1)
            {
                el = el.next;
            }
            Wx.Add(el.factor, el.index);
            el = el.next;
        }


        return Wx;
    }
```

Obliczanie tych samych wykładników:

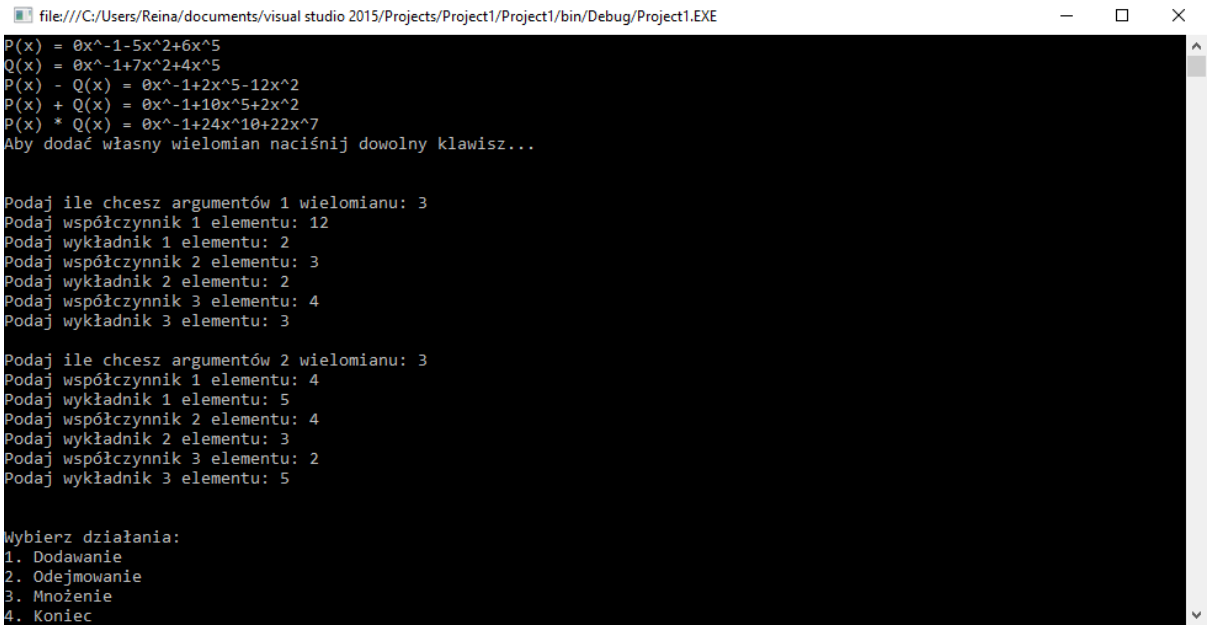
```
private List CalculateTheSameIndexes(List Vx)
    {
```

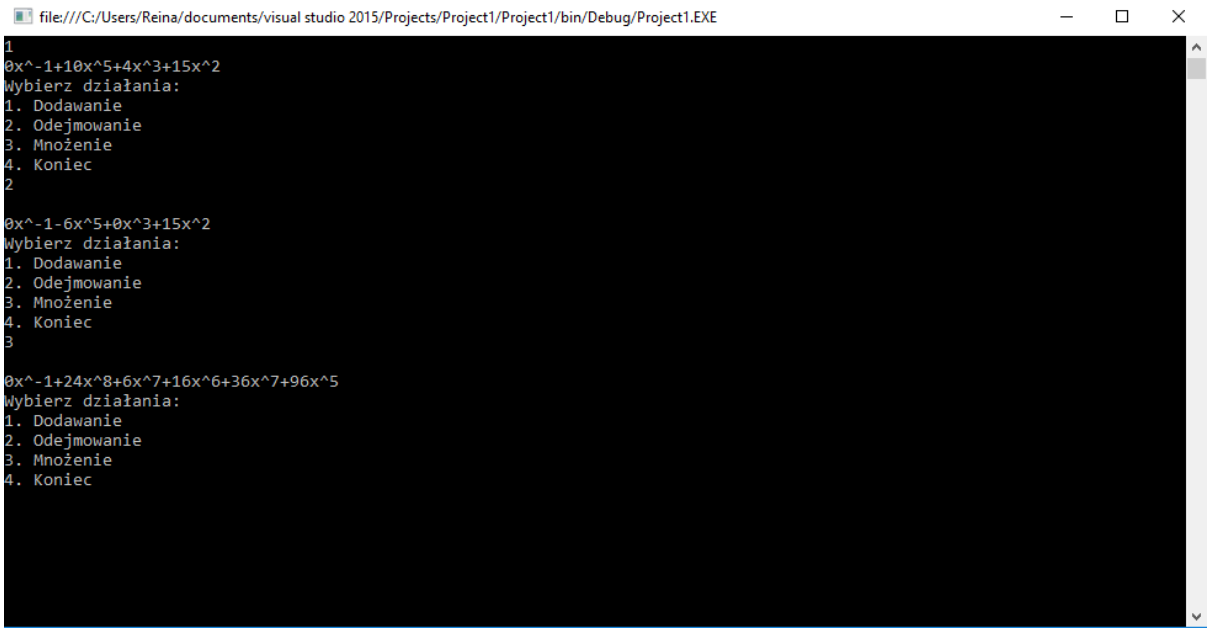
```
        Element el = Vx.head;
        Element newEl = null;
        int newFactor = 0;
        List Calculated = new List();
        Vx.GetLenght();
        var lenght = Vx.lenght / 2;
        for(;;)
        {
            newFactor = 0;
            while (el.index == el.next.index)
            {

                newFactor += el.factor;
                el = el.next;


                if (el.next == null)
                {
                    break;
                }

            }
            newFactor += el.factor;
            newEl = new Element(newFactor, el.index);
            Calculated.Add(newEl.factor, newEl.index);
            if (el.next != null)
            {
                el = el.next;
            }
            else
            {
                break;
            }

        }
        return Calculated;
    }
```

Wyniki - przykładowe wielomiany



Wyniki – pobieranie wielomianu z konsoli




file:///C:/Users/Reina/documents/visual studio 2015/Projects/Project1/Project1/bin/Debug/Project1.EXE
1
0x^-1+10x^5+4x^3+15x^2
Wybierz działania:
1. Dodawanie
2. Odejmowanie
3. Mnożenie
4. Koniec
2
0x^-1-6x^5+0x^3+15x^2
Wybierz działania:
1. Dodawanie
2. Odejmowanie
3. Mnożenie
4. Koniec
3
0x^-1+24x^8+6x^7+16x^6+36x^7+96x^5
Wybierz działania:
1. Dodawanie
2. Odejmowanie
3. Mnożenie
4. Koniec