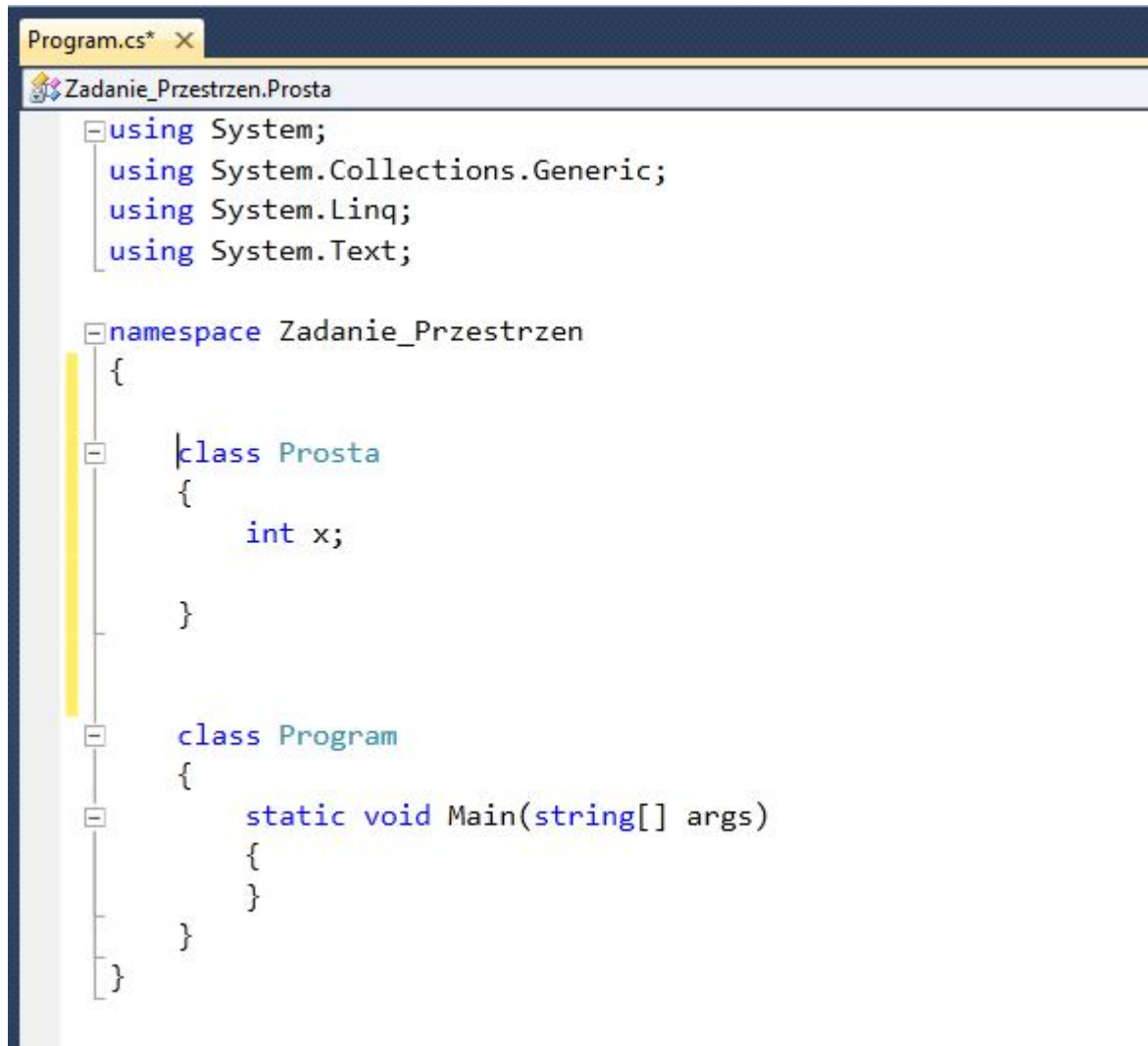


Rozpoczynamy tradycyjnie od stworzenia nowego projektu (aplikacji konsolowej).

W tym rozwiązaniu cały kod będziemy tworzyć tylko w jednym pliku Program.cs. Na razie nie będziemy wpisywać modyfikatora dostępu. W trakcie testowania danego fragmentu programu się tym zajmiemy. Czyli będzie na początek tylko tyle:



```
Program.cs* X
Zadanie_Przestrzen.Prosta
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zadanie_Przestrzen
{
    class Prosta
    {
        int x;
    }

    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Czyli zmienna x typu Integer będzie współrzędną punktu na prostej. Czy można już coś przetestować na tym etapie? Można jedynie stworzyć obiekt typu Prosta, bazując na konstruktorze domyślnym. Ale nic, poza tym się nie da zrobić. Napişmy i sprawdzimy...

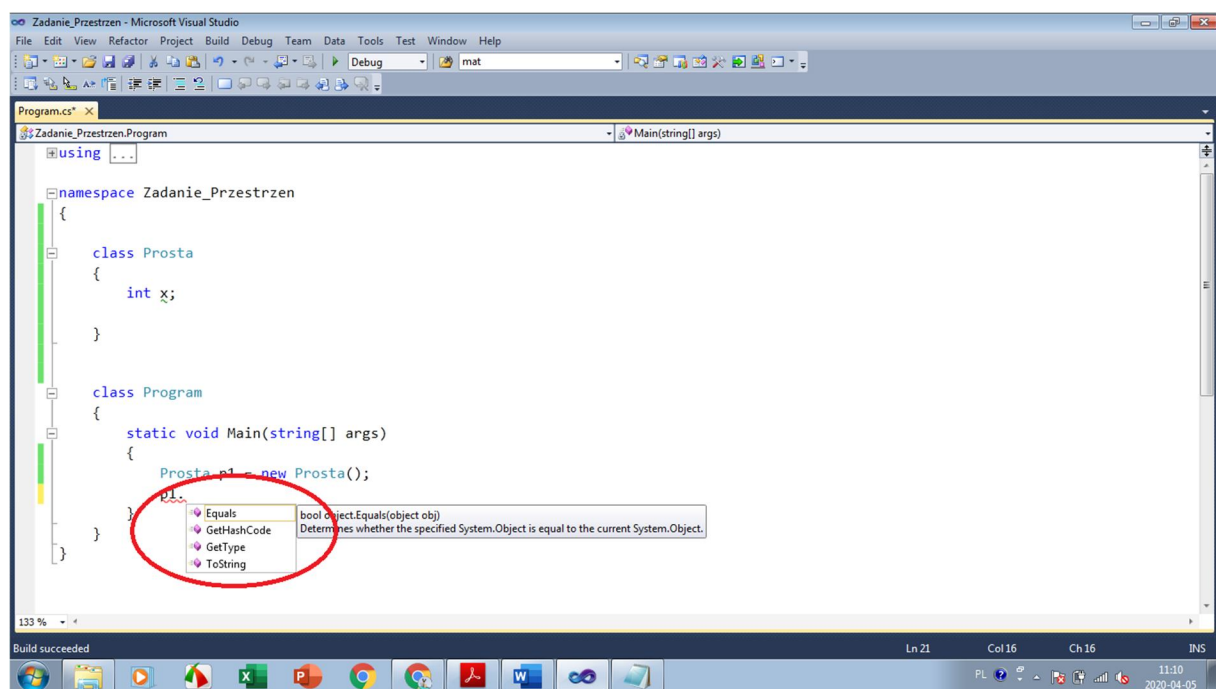
```

namespace Zadanie_Przestrzen
{
    class Prosta
    {
        int x;
    }

    class Program
    {
        static void Main(string[] args)
        {
            Prosta p1 = new Prosta();
        }
    }
}

```

Rzeczywiście program się wykonał i tyle. Czy można chociaż odczytać wartość współrzędnej punktu? Nie można...



Aby można było trzeba by stworzyć właściwość (ale tego w tym zadaniu nie będziemy robić). Oczywiście można by też dopisać modyfikator dostępu **public** przed zmienną **x**. Ale to nie tędy

droga...W treści zadania jest zapisane, aby w każdej klasie stworzyć konstruktor do inicjowania współrzędnych punktu. Tak też teraz zrobimy...

```
class Prosta
{
    int x;

    public Prosta(int wsp_x)
    {
        x = wsp_x;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Prosta p1 = new Prosta(3);
    }
}
```

Jasne jest że musieliśmy w metodzie **Main()**, gdzie tworzymy obiekt podać jakąś wartość współrzędnej. Musieliśmy też dopisać modyfikator dostępu **public** w definicji konstruktora, bo kompilator zgłaszał błąd. Jeszcze jedną rzecz mamy zrobić w klasie Prosta – utworzyć metodę wirtualną wyświetlającą odległość punktu od zera. Czyli piszemy...

```

class Prosta
{
    int x;

    public Prosta(int wsp_x)
    {
        x = wsp_x;
    }

    public virtual void Odl_zero()
    {
        Console.WriteLine( Math.Abs(x));
    }
}

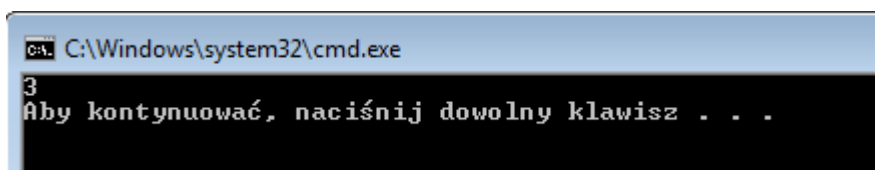
```

... i testujemy na punkcie o współrzędnej -3. Działa poprawnie.

```

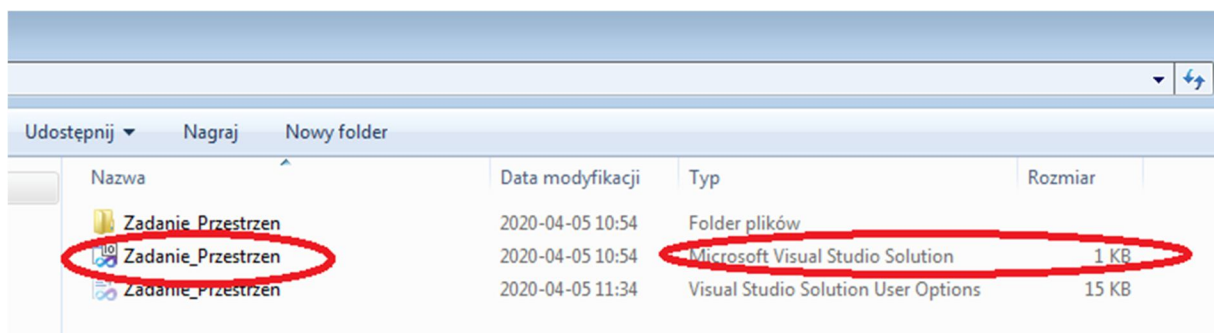
class Program
{
    static void Main(string[] args)
    {
        Prosta p1 = new Prosta(-3);
        p1.Odl_zero();
    }
}

```

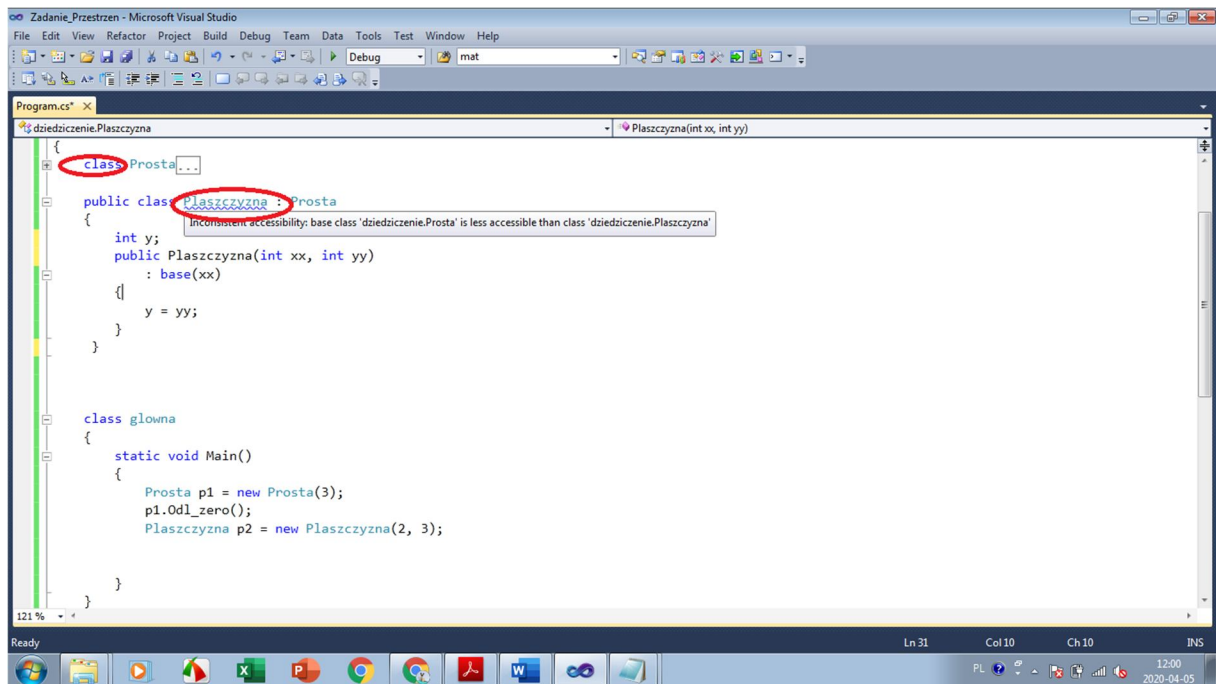


Teraz zrobimy małą przerwę na kawę, czyli Ctrl+Shift+S i zamykamy VS.

A po kawce wracamy do pracy. Odnajdujemy folder i klikamy na plik z solucją...



Mamy już klasę bazową Prosta. Teraz czas na klasę pochodną Płaszczyzna. Zdefiniujemy w niej pole (element danych) oraz konstruktor. Będzie:



Jednak przy próbie uruchomienia pojawi się błąd... Brakuje modyfikatora dostępu **public** przy definicji klasy **Prosta**. Dopiszmy go utworzymy drugi obiekt w metodzie głównej – punkt na płaszczyźnie...

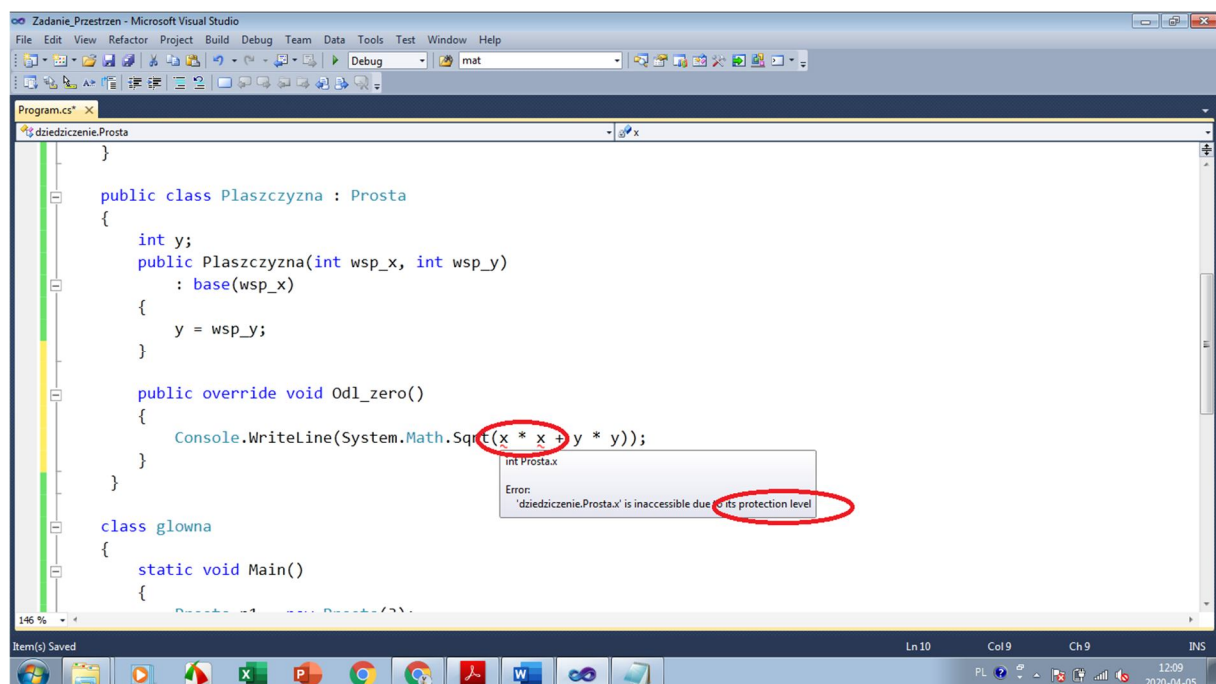
Cały kod wygląda na ten moment tak:

```
namespace dziedzenie
{
    public class Prosta
    {
        int x;
        public Prosta(int wsp_x)
        {
            x = wsp_x;
        }
        public virtual void Od1_zero()
        {
            Console.WriteLine(Math.Abs(x));
        }
    }

    public class Plaszczyna : Prosta
    {
        int y;
        public Plaszczyna(int wsp_x, int wsp_y)
            : base(wsp_x)
        {
            y = wsp_y;
        }
    }

    class glowna
    {
        static void Main()
        {
            Prosta p1 = new Prosta(3);
            p1.Od1_zero();
            Plaszczyna p2 = new Plaszczyna(2, 3);
        }
    }
}
```

Teraz w klasie **Plaszczyna** napiszemy metodę przesłaniającą metodę z klasy dziedziczonej. Będzie:



```
Program.cs - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug mat
Program.cs x
dziedzzenie.Prosta
}

public class Plaszczyna : Prosta
{
    int y;
    public Plaszczyna(int wsp_x, int wsp_y)
        : base(wsp_x)
    {
        y = wsp_y;
    }

    public override void Od1_zero()
    {
        Console.WriteLine(System.Math.Sqrt(x * x + y * y));
    }
}

class glowna
{
    static void Main()
    {
        Prosta p1 = new Prosta(3);
        p1.Od1_zero();
        Plaszczyna p2 = new Plaszczyna(2, 3);
    }
}
```

Error: 'dziedzzenie.Prosta.x' is inaccessible due to its protection level

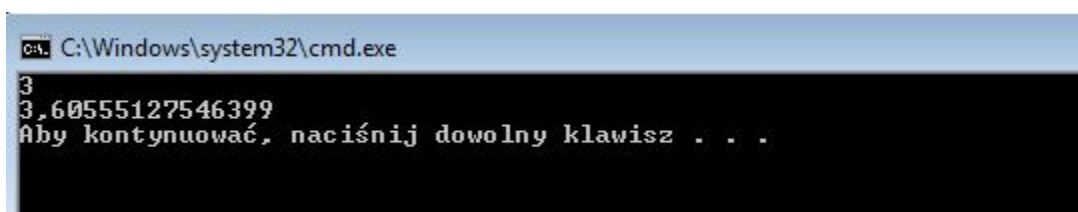
Mamy problem. Z argumentem **x** metody **Sqrt()**. Trzeba w klasie bazowej **Punkt** udostępnić zmienną **x** dla klasy pochodnej za pomocą modyfikatora dostępu **protected**.

```
public class Prosta
{
    protected int x;
    public Prosta(int wsp_x)
    {
        x = wsp_x;
    }
    public virtual void Od1_zero()
    {
        Console.WriteLine(Math.Abs(x));
    }
}
```

Ten sam problem się pojawi gdy będziemy tworzyć następną klasę pochodną, więc już teraz dopisujemy też ten modyfikator dostępu przy deklaracji zmiennej **y**. Zobaczmy czy działa metoda **Od1_zero()** dla obiektu klasy **Plaszczyzna**.

```
class glowna
{
    static void Main()
    {
        Prosta p1 = new Prosta(3);
        p1.Od1_zero();
        Plaszczyzna p2 = new Plaszczyzna(2, 3);
        p2.Od1_zero();
    }
}
```

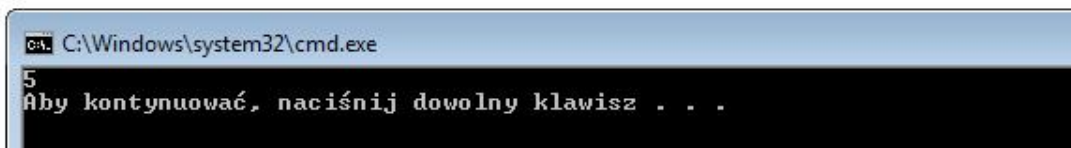
Uruchamiany program...



```
C:\Windows\system32\cmd.exe
3
3,60555127546399
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Działa. Sprawdźmy jeszcze dla punktu (3,4). Wtedy powinno wyjść 5. I tak się dzieje....

```
class glowna
{
    static void Main()
    {
        Prosta p1 = new Prosta(3);
        // p1.Odl_zero();
        Plaszczyzna p2 = new Plaszczyzna(3, 4);
        p2.Odl_zero();
    }
}
```



W analogiczny sposób trzeba utworzyć klasę **Przestrzen**. Kod będzie wyglądał tak:

```
public class Przestrzen : Plaszczyzna
{
    protected int z;

    public Przestrzen(int wsp_x, int wsp_y, int wsp_z)
        : base(wsp_x, wsp_y)
    {
        z = wsp_z;
    }

    public override void Odl_zero()
    {
        Console.WriteLine(System.Math.Sqrt(x * x + y * y + z * z));
    }
}
```

A przetestowanie w metodzie **Main()** np. tak:


```

class glowna
{
    static void Main()
    {
        // Prosta p1 = r
        // p1.Odl_zero()
        // Płaszczyzna p
        // p2.Odl_zero(),
        Przestrzen p3 = new Przestrzen(2,3,4);
        p3.Odl_zero();
    }
}

```

cmd. C:\Windows\system32\cmd.exe

5.3851648071345

Aby kontynuować, naciśnij dowolny klawisz . . .

Dla pewności sprawdzimy wynik porównując go z jakimś kalkulatorem znalezionym w sieci....

Podaj współrzędne punktów $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$.

Współrzędne punkta A

$x_1 =$ 0

$y_1 =$ 0

$z_1 =$ 0

Współrzędne punkta B

$x_2 =$ 2

$y_2 =$ 3

$z_2 =$ 4

Oblicz

Wynik obliczeń

Odległość $d(A,B)$ pomiędzy podanymi punktami wynosi:

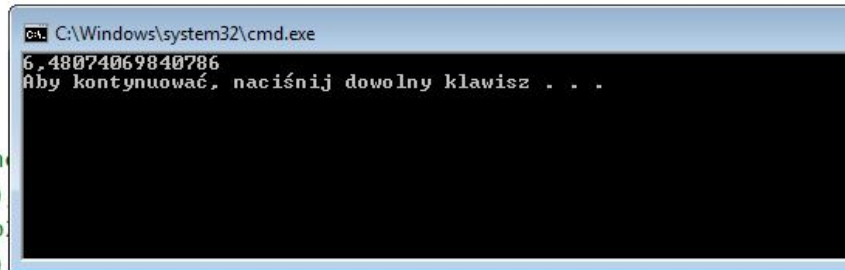
$d(A,B) =$ 5.385

Czyli jest OK. Pozostała do napisania metoda statyczna zwracająca odległość między dwoma punktami w przestrzeni. Będzie wyglądać tak:

```
public static void Odl_punktow(Przestrzen p1, Przestrzen p2)
{
    Console.WriteLine(System.Math.Sqrt(Math.Pow(p1.x - p2.x, 2) + Math.Pow(p1.y - p2.y, 2) + Math.Pow(p1.z - p2.z, 2)));
}
```

A jej wywołanie np. tak:

```
class glowna
{
    static void Main()
    {
        // Prosta p1 = n
        // p1.Odl_zero()
        // Płaszczyzna p
        // p2.Odl_zero()
        Przestrzen p3 = new Przestrzen(2,3,4);
        // p3.Odl_zero();
        Przestrzen p4 = new Przestrzen(1, -2, 0);
        Przestrzen.Odl_punktow(p3, p4);
    }
}
```



Znowu możemy się upewnić czy dobrze zapisaliśmy wzór na odległość między punktami. Będzie:

Podaj współrzędne punktów $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$.

Współrzędne punkta A

$x_1 =$	2
$y_1 =$	3
$z_1 =$	4

Współrzędne punkta B

$x_2 =$	1
$y_2 =$	-2
$z_2 =$	0

Oblicz

Wynik obliczeń

Odległość $d(A, B)$ pomiędzy podanymi punktami wynosi:

$d(A, B) =$ 6.48

Jest dobrze. Zatem koniec zadania.

