Labb4 - C#

Uppgiften i den här labben är att först skapa en klass Position vi kan använda för att hantera en position i ett koordinatsystem. Efter detta skall du skapa en klass SortedPosList för att hantera en sorterad samling utav positioner.

Skapa en solution som heter Labb4, och skapa dina filer i denna.

Inlämning:

Vad som skall lämnas in:
Position.cs, SortedPosList.cs

Deadline:
Söndag 8 novmber, 23:55

Testprogram

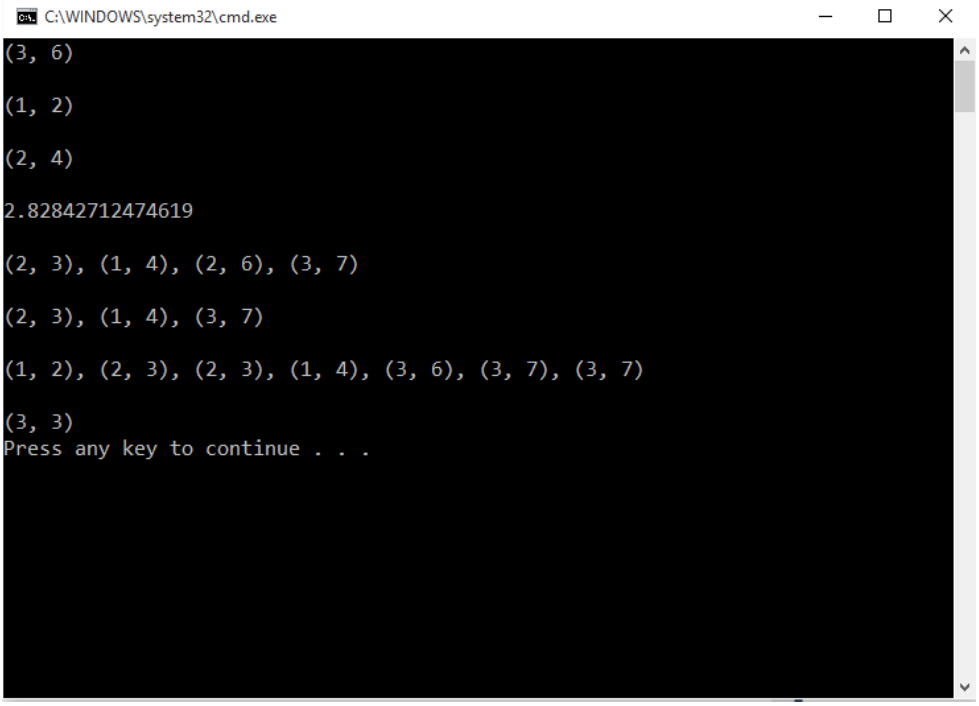
Det finns en körbar klass, Program, på kurshemsidan som ni ska använda för att testa er kod. Lägg märke till att testkoden för VG-uppgiften är bortkommenterad. För er som gör den uppgiften, se till att den bortkommenterade koden körs också.

OBS! För att testkoden ska fungera måste följande ToString metod läggas i SortedPosList:

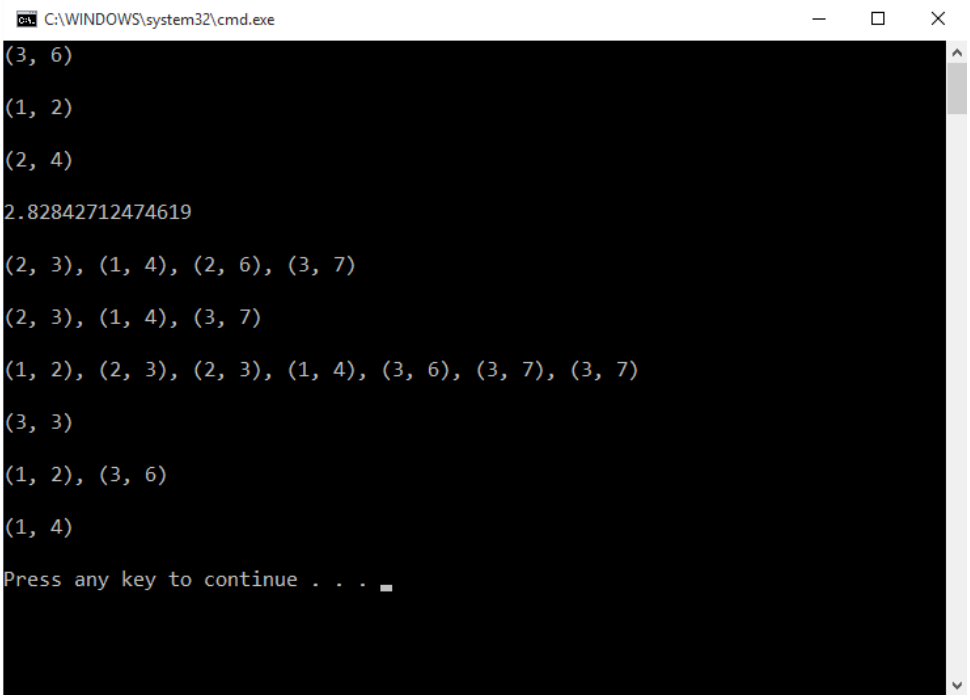
```
override
public string ToString()
{
     return string.Join(", ", positionList);
}
```

Kolla nästa sida för att se hur utskriften borde se ut. **Se till att testa din kod innan du lämnar in den!**

Så här ska utskriften se ut på G delen:



För G+VG ska utskriften se ut så här:



Uppgift 1 - Position

Denna klass skall representera en position i ett tvådimensionellt koordinatsystem, alltså ett par av en x-koordinat och en y-koordinat. Tillsammans kan man använda dessa koordinater för att hitta en specifik punkt på ett koordinatsystem med två axlar. Här finns mer information om koordinatsystem.

En position skall ha två stycken **properties**:

- X-värdet
- Y-värdet

Bägge dessa properties ska kunna förändras, men får aldrig bli negativa. Använd absolutvärde!

Klassen skall också ha ett antal metoder:

- **En konstruktor som tar emot ett x-värde och ett y-värde och lagrar dem i lämpliga properties.**
- **public double Length()**
  - Den skall returnera avståndet från origo (koordinaten (0, 0)) till den här punkten.
  - Detta kan göras mha pythagoras sats
  - Den slutliga formeln ser ut på följande sätt:
    - $\textbf{\textit{avstånd från origo}} = \sqrt{X^2 + Y^2}$
    - Där X är x-positionen och Y är y-positionen
    - Detta är några metoder som kan vara till hjälp:
      Math.Sqrt(n): returnerar en double som är den kvadratiska roten ur n
      Math.Pow(n, k): returnerar n upphöjt till k
- **public bool Equals(Position p)**
  - Skall returnera true om den givna positionen är samma som den här
    - Annars skall den returnera false
  - Två punkter räknas som lika om både X-koordinaten och Y-koordinaten är lika
- **public Position Clone()**
  - Skall returnera en ny instans av Position, som har samma X och Y-värden som den nuvarande punkten.
- **public string ToString()**
  - Skall returnera en string i följande format: (X, Y)

För att underlätta användandet utav klassen skall vissa operatorer också overloadas.

- **< och >**
  - **public static bool operator >(Position p1, Position p2)**
  - Skall jämföra två positioners avstånd från origo
    - Jämför alltså storleken på de bägge punkternas Length()
  - p1 > p2
    - Returnerar true om p1 ligger längst ifrån origo
    - Returnerar false annars

    - Om positionerna har samma längd från origo:
        - Skall x-variablen jämföras istället
- **+ och -**
    - **public static Position operator +(Position p1, Position p2)** Skall addera eller subtrahera två punkter med varandra
    - Returnerar en ny position
    - Exempel:
        - p1 = (3, 5), p2 = (1,3)
        - p1 + p2 = (3+1, 5+3) = (4, 8)
        - p2 – p1 = (1-3, 3-5) = (2, 2)
- **%**
    - **public static double operator %(Position p1, Position p2)**
    - Skall returnera en double
    - Räknar ut avståndet mellan två punkter och returnerar det
    - Även detta görs mha utav Pythagoras sats
    - Formeln ser ut på följande sätt:
        - $\textbf{\textit{avstånd}} = \sqrt{(\textbf{\textit{X}}1 - \textbf{\textit{X}}2)^2 + (\textbf{\textit{Y}}1 - \textbf{\textit{Y}}2)^2}$
    - Exempel:
        - p1 = (3,5), p2=(1,3)
        - p1 % p2 = $\sqrt{(3-1)^2 + (5-3)^2} = \sqrt{8} \approx 2.82$

Uppgift 2 - SortedPosList

Vi skall nu skapa en klass som skall fungera som en lista för just positioner. Den skall hela tiden vara **sorterad efter Length** (dvs avståndet varje position i listan har till origo). Den närmsta punkten skall ligga först och den längst bort skall ligga sist. Listan [(5,5), (3,3)] är alltså inte giltligt sorterad, men [(3,3), (5,5)] är det.

För att skapa den här skall vi ha en instansvariabel:

- En private List utav Positions
    - Detta är den bakomliggande listan som vår SortedPosList lagrar info i. Den skall endast användas inuti klassen

Vi skall också skapa ett antal metoder:

- **public int Count()**
    - Skall returnera antalet positioner vi har i vår lista
- **public void Add(Position pos)**
    - Skall lägga till en given position till vår lista
    - Den skall se till att positionen hamnar på rätt plats i listan. Kom ihåg att listan

skall vara sorterad! Denna metod ansvarar för att lägga in ***pos*** på rätt plats i den interna listan.
    - Exempel:
        - Anta att listan innehåller [(5,5), (7,9)].
        - Anta att vi vill lägga till (6,6).
        - Den nya listan blir då [(5,5), (6,6), (7,9)]. Add lade in elementet på index 1, eftersom listan alltid måste vara sorterad
- **public bool Remove(Position pos)**
    - Skall ta bort positionen ***pos*** från listan. Returnerar ***true*** om pos fanns i listan och togs bort, och ***false*** annars.
- **public SortedPosList Clone()**
    - Skall returnera en ny instans av SortedPosList, som innehåller kopior av alla punkter som finns i den här listan. (Ska alltså anropa Clone på alla punkter.
- **public SortedPosList circleContent(Position centerPos, double radius)**
    - Skall returnera en ny lista (använd Clone), som innehåller alla positioner som befinner sig inom en given cirkel
    - Som argument skickas cirkelns mitt-position och cirkelns radie
    - För att ta reda på om en punkt ligger i en cirkel: länk
    - Exempel:
        - Anta att vi har en lista som innehåller punkterna [(1,1),(2,2),(3,3)]
        - Anta att metoden anropas med denna cirkel med centerPos=(5,5) och radius=4
        - Om vi ska filtrera ut alla punkter som ligger inom cirkeln ovan, returneras en ny lista som endast innehåller punkten [(3,3)], eftersom det är den enda av ursprungspunkterna som ligger i cirkeln.

SortedPosList skall också overloada några operatorer

- []
    - Getter: ska hämta ett element på en given position i listan.
    - Setter ska **ej** finnas (eftersom detta kan förstöra listans sortering. Metoden Add bör istället alltid användas)

- **+**
    - **public static SortedPosList operator +(SortedPosList sp1, SortedPosList sp2)**
    - Skall returnera en ny lista (använd Clone), som är de två adderade listorna ihopslagna till en
        - Tänk på att den resulterande listan skall vara sorterad! (Tips: använd din egen Add-metod)

Uppgift 3 - VG

Minus av listor

Vi skall nu utöka vår SortedPosList med en ny operator **-**. Den skall ta bort alla gemensamma positioner mellan två listor från en av dem. Exempel:
A = { (3, 7), (1, 4), (2, 6), (2, 3) }
B = { (3, 7), (1, 2), (3, 6), (2, 3) }
A - B = { (1, 4), (2, 6) }
B - A = { (1, 2), (3, 6) }

(3, 7) och (2, 3) är gemensamma mellan A och B, därför tas de bort. Notera att det spelar roll vilken ordning man skriver listorna när man använder operatorn!

Operatorn ska returnera en ny lista (använd Clone).

Vanligtvis om vi skall lösa detta måste vi loopa igenom alla element i B för varje element i A. Om elementet påträffas, tar vi bort det.

Ovstående lösning är dock rätt ineffektiv, eftersom datorn måste utföra väldigt många operationer. Eftersom både A och B är sorterade, kan vi lösa detta på ett effektivare sätt, så att vi endast behöver loopa igenom A och B en gång. Ni ska implementera den på detta effektiva sätt (**Detta är krav för VG**).

Tips:
Algoritmen har två “pekare” hela tiden som pekar på varsin position i de båda listorna, dessa “pekare” är helt enkelt heltal som bara definierar en position.

Algoritmen går i steg hela tiden, där den först gör en jämförelse mellan de två listorna sedan avgör om den skall ta bort något, eller öka någon utav pekarna. En pekare kommer aldrig att minskas under algoritmen.

Den kör tills någon utav pekarna har nått slutet av sin respektive lista.

Detaljer kring hur jämförelserna och borttagningen ska göras kan ni klura ut själva.