# Datastructures, Oefeningen Recursie

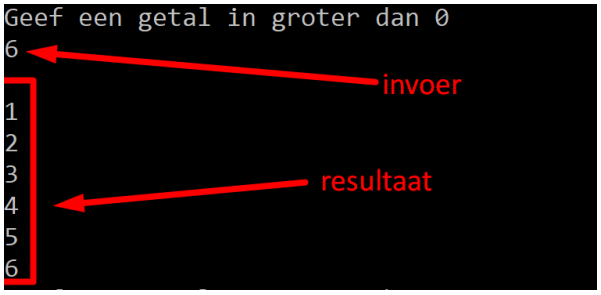
## 1. Countdown raket

- Gebruik een console project
- Maak een recursieve functie die terugtelt vanaf een gegeven aantal seconden.
- **void Countdown(int number)**
- Print de seconden via Console.WriteLine
- Print “Takeoff” wanneer de 0 seconden bereikt werd.
- Maak je functie robuust tegen ongeldige “input”.

```
Geef een getal in groter of gelijk aan 0
5
5.. 4.. 3.. 2.. 1.. Take-off !
```

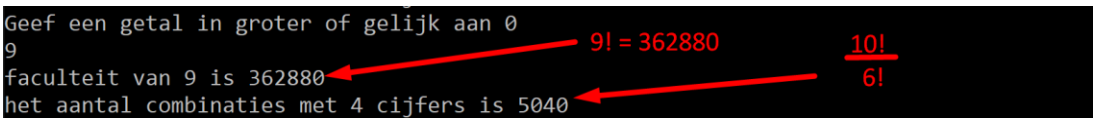
## 2. Teller

- Maak een recursieve functie die optelt tot een gegeven getal.
- **void Upcounter(int number)**
- Print de getallen in de console




## 3. Faculteit berekenen

- De faculteit wordt gebruikt bij berekeningen omtrent combinaties. Bijvoorbeeld als je wil berekenen hoeveel mogelijke getallen je kan maken met 4 verschillende cijfers tussen 0 en 9.
- De faculteit wordt genoteerd als n!
- De faculteit van bv. 6 kan je berekenen als 6! = 6 x 5 x 4 x 3 x 2 x 1
- Speciaal geval is: 0! = 1
- Faculteit berekenen van een negatief getal is niet mogelijk.
- Maak een recursieve methode om de faculteit te bereken van een bepaald getal.
- Gebruik vervolgens je methode om te berekenen hoeveel 10! / 6! bedraagt, want dat is het antwoord op de bovenste vraag.



## 4. Een string omkeren

- Maak een recursieve functie om een gegeven string om te keren
- **string Reverse (string text)**
- “Think defensive”

```
Geef een string en ik keer deze om!
aardgasreserves
Het omgekeerde van aardgasreserves is sevresersagdraa
```

## 5. Een String splitsen in afzonderlijke karakters

- Maak een functie **LineairQueueString Reverse(string tekst)**
- De gegeven tekst wordt opsplitst in afzonderlijke karakters aan de hand van recursie.
- De karakters worden tevens in een queue gepompt maar in de omkeerde volgorde (dus de laatste letter moet eerst uit de queue komen)

```
Geef een string en ik keer deze om!
secretaressedag
Het omgekeerde van secretaressedag is
gadesseraterces
```

## 6. Tel het aantal cijfers in een getal

- Maak een methode om het aantal cijfers te bepalen in een getal
- Int NumberOfDigits(int number)

```
Geef een getal in
45679223
Het aantal cijfers in 45679223 is 8
```

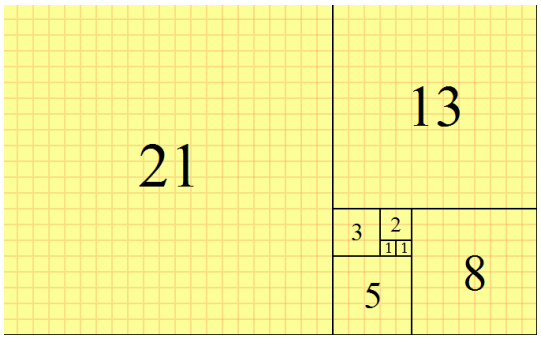
## 7. Bereken het Fibonacci getal

Maak een functie die het zoveelste getal berekent uit de Fibonacci reeks. De fibonacci reeks zie je hieronder ook afgebeeld in de figuur: 1 - 1- 2 – 3 – 5- 8 – 13 – enz… Het 6e fibonacci getal is dus 8, het 7e fibonacci getal is 13 enz…

- int FibonacciNr (int n)
- Een Fibonacci getal voor **n** kan je bepalen door **FibonacciNr (n-1) + FibonacciNr (n-2)** te berekenen. Bijvoorbeeld, het 6e fib. Getal = het 5e fib.getal + 4e fib. Getal oftewel 8 = 5 + 3
- FibonacciNr (0) geeft 0 en FibonacciNr (1) geeft de waarde 1
- Negatieve waarden voor n zijn niet mogelijk.

Maak vervolgens een 2e functie die de eerste zoveel getallen uit de Fibonacci reeks berekent en afprint in de console.

- void FibonacciSerie (int count)



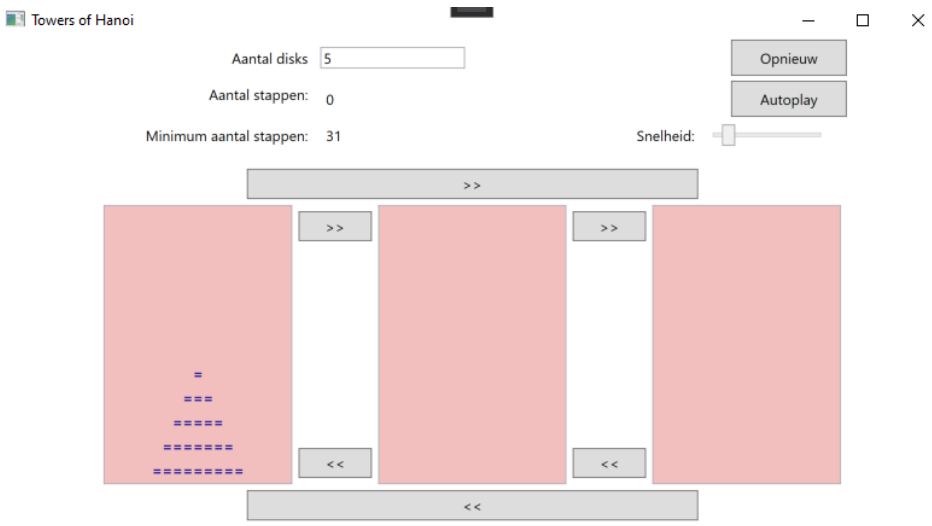

```
Het hoeveelste fibonacci number wil u berekenen ?
6
Het 6e fibonacci nummer is 8
8 - 5 - 3 - 2 - 1 - 1 - 0
```

FibonacciNr(6)

FibonacciSerie(6)

## 8. Torens van Hanoï

- Maak een WPF versie van dit spel (zie ook theorie)



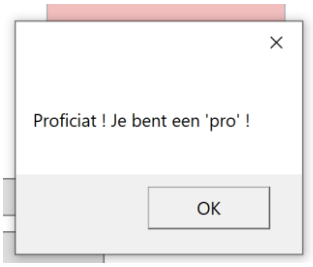

- De gebruiker kan instellen: **aantal disks** en krijgt dan als feedback het **minimum aantal benodigde stappen** en het **aantal reeds gedane stappen**.
- Met de knop '**opnieuw**' kan hij/zij het spel herstarten (alles disks komen terug op de linker toren)
- Elke toren wordt voorgesteld door een **listbox** waarin de schijven worden toegevoegd. Het probleem is dat bij een listbox de elementen standaard bovenaan verschijnen. WPF heeft echter krachtige mogelijkheden. Op deze manier kan je de alignering aanpassen zodat de elementen onderaan gealigneerd worden (en dus van onder naar boven zullen gaan):

```
<ListBox x:Name="lb2"  HorizontalContentAlignment="Center" FontWeight="Bold" Background="#FFF3BEBE" Foreground="#FF1B159B"
    <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
            <VirtualizingStackPanel VerticalAlignment="Bottom"/>
        </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
</ListBox>
```

*(HorizontalContentAligment property zorgt er voor dat de disks horizontaal in het midden worden gealigneerd, met background en foreground properties kan je desgewenst de kleuren aanpassen)*

- Maak voor het spel zelf een aparte '**TowerOfHanoi'** game **klasse** waarin je alle logica en variabelen (torens, disks, aantal moves, ..) van het spel kan onderbrengen.
- Een belangrijke vraag hierbij is: **hoe gaan we bijhouden in welke toren, welke disk ligt en wat de volgorde is van de disks in een toren**. Ga na welke datastructuur - die we reeds gezien hebben -, hiervoor ideaal geschikt is om de disks in 1 toren bij te houden !
- Denk na wat het spel allemaal moet kunnen en voorzie hiervoor de nodige methodes en properties op je game klasse.
- Tracht eerst de **manuele** versie van het spel te bouwen, waarbij de gebruiker de disks 1 voor 1 verplaatst met de knoppen.
    1. De gebruiker drukt op een knop -> click event wordt afgevuurd
    2. Elke knop geeft hiermee een move aan (bv. van toren 1 naar toren 3)
    3. Laat je game deze move uitvoeren (indien deze move is toegelaten uiteraard)

4. Toon vervolgens de huidige inhoud van elke toren op het scherm in de overeenkomstige listbox. In je datastructuur kan je best gewoon getallen bijhouden (bv. 1 = de kleinste disk, 2 de volgende enz…). Om die dan op het scherm te tonen kan je ze telkens eerst omzetten naar een string, bv. 1 wordt "=", 2 wordt "===" enz…
5. Deze stappen herhalen zich tot het spel ten einde is.




Denk goed na over:

- Waar bepaal je of een move wel is toegelaten (maw. de spelregels volgen) ?
- Waar zet je de waarde van de disk (bv. 1) om naar een visuele versie (bv. "=") ?
- Hoe bepaal je welke knop, welke move zal veroorzaken ?
- Waar en hoe bepaal je wanneer het spel ten einde is ?

- Vervolgens kan je overgaan naar de **automatische** versie. Uiteraard kan je hiervoor terug je game klasse gebruiken om het spel te spelen. Alleen gaat de gebruiker de moves nu niet uitvoeren. Je gaat eerst alle moves laten "berekenen" aan de hand van een **recursieve** methode. Dit berekenen gaat supersnel, veel te snel om tegelijkertijd alle moves te tonen aan de gebruiker. Daarom kan je dit best opsplitsen in 2 stappen wanneer de gebruiker klikt op "Autoplay":
    1. Laat eerst **alle moves berekenen** met de **recursieve functie** en houdt alle moves bij in een hiervoor **geschikte datastructuur** (die we reeds gezien hebben). Je kan elke move voorstellen adhv. een string: bv. "13" betekent een move van toren 1 naar toren 3, "32" betekent een move van Toren 3 naar 2, enz…
    2. **Speel de moves** in je datastructuur vervolgens **1 voor 1** (in de juiste volgorde uiteraard) met een kleine tussenpauze zodat de gebruiker het spel kan meevolgen. (Hiervoor kan je de timer gebruiken van de fabriek, elke tick doe je een move, op die manier kan je ook de snelheid makkelijk bepalen waarmee de moves gebeuren)

  *De recursieve methode om de moves te berekenen vind je terug in de presentatie (theorie) en bijhorend filmpje. In plaats van de moves af te printen in het debug venster moet je elke move nu dus bijhouden in een datastructuur om ze nadien te kunnen spelen.*

  Denk goed na over:

  - Waar plaats je de recursieve functie om de moves te berekenen ?
  - Waar zet je de timer met tick event ?
  - Welke datastructuur gebruik je om de moves te bewaren zodat je ze terug kan spelen nadien.