Welkom!

Ik ben
Filip Geens

Email : filip.geens@live.com

Programmeren in C#

C# als programmeertaal

Programmatie

Welke stappen volgen we bij het ontwerp van een programma?

- Een goede beschrijving van het probleem
- Data? Inkomende – Uitgaande
- Fun: het ontwerp van de code.
- Bewijs leveren dat je programma werkt! => Testing!

Wat is C#?

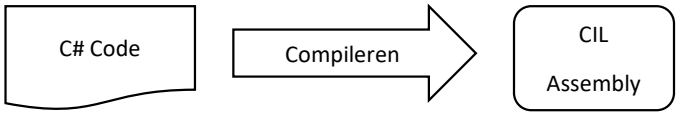
- Het is een object georiënteerde taal
- Toont gelijkenissen met C, C++ en Java
- Wordt gecompileerd en niet geïnterpreteerd
- Speciaal ontworpen om in een .Net omgeving te werken

.Net platform

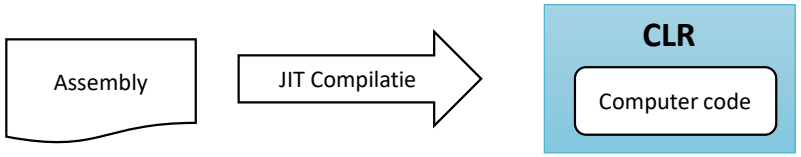
- CLI : Common Language Infrastructure
  - Taal neutraal framework
  - Bevat:
    - CTS -> Common Type System
    - Metadata
    - CLS -> Common Language Specification
    - VES -> Virtual Execution System
- CLR: Common Language Runtime
  - Virtuele machine waarin de code wordt uitgevoerd
  - JIT compiling
  - Zorgt voor Type mapping, GC (garbage collector), …

Compilatie in .Net

- Compilatie naar een tussentaal : CIL (Common Intermediate Language)



- Resultaat wordt opgeslagen in een Assembly
  - Dit is meestal een DLL of een EXE
  - Er kunnen ook resources in opgeslagen worden zoals foto's, tekst, geluid…
  - Uitwisselbaar tussen andere assemblies binnen het framework
- JIT compiling door de CLR
  - Omzetten naar lokale machinetaal op het moment dat dit deel wordt gebruikt



Coderen in C#

- Naamgeving
  - Goede afspraken zijn nodig om de leesbaarheid te verhogen
  - C# is hoofdletter gevoelig. Dit is niet zo bij alle talen (VB.Net)
    - Let op wanneer je code uitwisselt met een Visual Basic applicatie!
  - Bij conventie zijn volgende schrijfregels gangbaar in C# code:
    - Parameters, lokale variabelen en private velden : **camelCase**. -> myVar
    - Andere identifiers: **PascalCase** -> MyFunction();
- Keywords
  - Dit zijn benamingen die een speciale betekenis hebben voor de compiler
    - Bijv class, enum, int, decimal, …
  - Deze kunnen (*best**) niet gebruikt worden als keyword

Opmerkingen toevoegen in code

- Wordt gebruikt om de leesbaarheid te verhogen
- We kunnen moeilijk verstaanbare code toelichten
- 2 manieren:
  - Lijn: //
    - De rest van de lijn wordt beschouwd als commentaar:

```
int myVar = 0; //Dit is mijn variable
```

  - Blok: /* */
    - De tekst binnen de commentaar markering wordt beschouwd als een opmerking
      - Gebruikt om meerdere lijnen toe te voegen
      - We kunnen ook opmerkingen tussen de code plaatsen:

```
int myVar /* De waarde moet standaard negatief zijn !! */ = -1;
```

  - Commentaar wordt genegeerd door de compiler
- Overdrijf ook niet met opmerkingen!

Types

- Voorgedefinieerde Types
  - Gekend door de compiler / CLR
  - Getallen, tekst, datum, logische types
    - Bijv : int, long, float, string, bool,…
- Zelf gecreëerde Types
  - In C# is het zeer makkelijk om zelf types te creëren
  - Met behulp van class, struct, interface

Value en Reference Types

- Structureel verschil, zowel intern als in het gebruik
- Value types worden gestockeerd in de **Stack**
  - De Stack werkt sequentieel en is netjes geordend
- Reference types worden opgeslagen in de **Heap**
  - De Heap is het ongeordende geheugen en wordt beheerd door de GC (garbage collector)

Value types

- De inhoud van een value type wordt beschouwd als een **waarde**
  - Bij het kopiëren van een value type wordt de **inhoud** gekopieerd

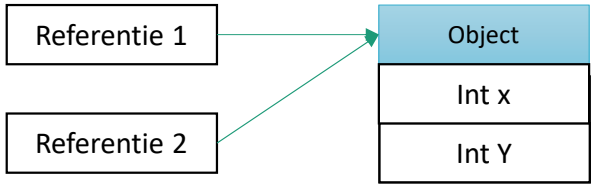
```
int a = 1;
int b = a;
a = 3;
Console.WriteLine(“b = “+b);

output=> b = 1
```

- We kunnen een value type maken door een ‘struct’ te declareren

```
public struct PointTo {
    public int x;
    public int y;
    public PointTo(int xPos,int yPos) { x = xPos; y = yPos;  }
 }
```

Reference types

- Heeft 2 delen: een object en een referentie naar dat object
- Het object bevat de waarde, het gedeclareerde type bevat een referentie naar het object.



- Het is mogelijk dat er meerdere referenties wijzen naar dezelfde waarde(n)
- Een referentie kan ook leeg zijn => null
- Als er geen enkele referentie meer naar het object wijst wordt deze verwijderd door de GC

Een reference type declareren

- We kunnen een reference declareren door ‘class’ te gebruiken ipv ‘struct’

```
public class PointTo {
   public int x;
   public int y;
   public PointTo(int xPos,int yPos) {
     x = xPos;
     y = yPos;
   }
}
```

- Ook interface, delegate, string, object, … zijn reference types

Labo : Value types contra Reference types

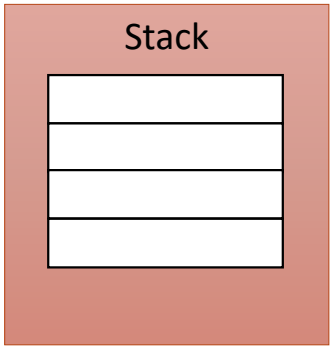
- Maak een nieuw C# console project aan
- Creëer een **value type** waarin we de hoogste en de laagste temperatuur kunnen weergeven, genaamd **ValueTemp**
- Creëer ook een **reference type** met hetzelfde doel: **RefTemp**
- Declareer 3 temp instanties van elk type en wijs het 2de aan het eerste toe.
- Stel het 2de Type gelijk aan het eerste en laat het derde leeg (niet toewijzen)
- Wijzig de temperatuur van het eerste van elk type.
- Druk het resultaat af

Wat gebeurt er binnen in de CLR ?

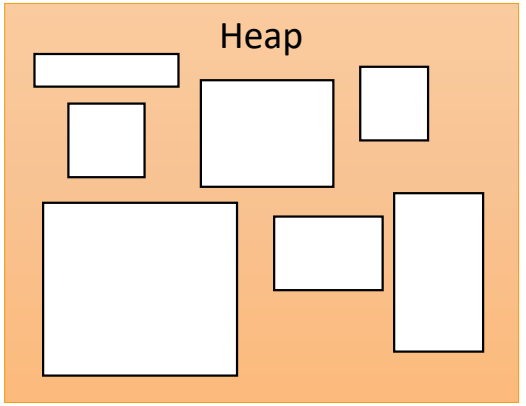
- Hoe werkt de Stack en de Heap?
  - We duiken dieper in de werking

Verschil tussen Stack en Heap

- De Stack:
  - Is sequentieel opgebouwd
  - Elke thread heeft zijn eigen stack
  - De stack is zelf onderhoudend
    - Wanneer een element niet meer wordt gebruikt dan wordt het verwijderd

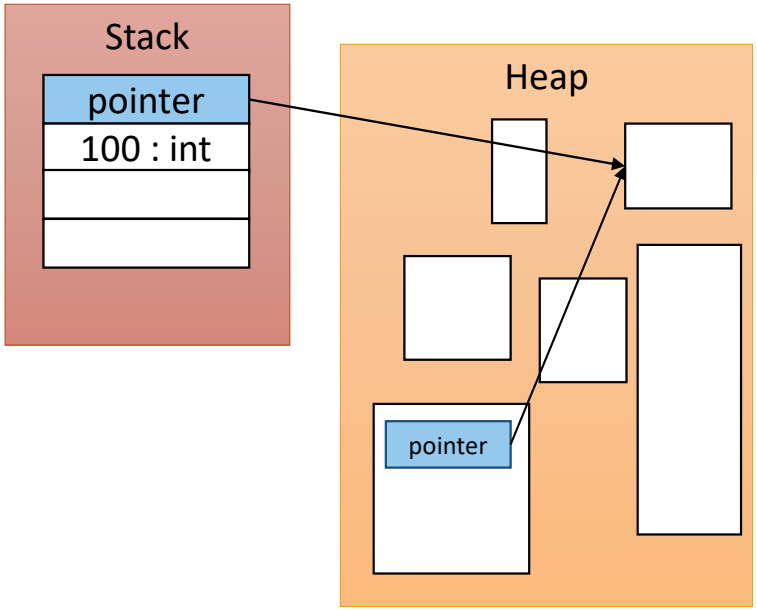


- De Heap
  - Is niet sequentieel
  - De heap wordt gedeeld!
  - Wanneer een object niet meer is gebruikt, dient deze verwijderd te worden door de eigenaar
    - In .Net is dit gelukkig de taak van de garbage collector



Wat gaat waar ?

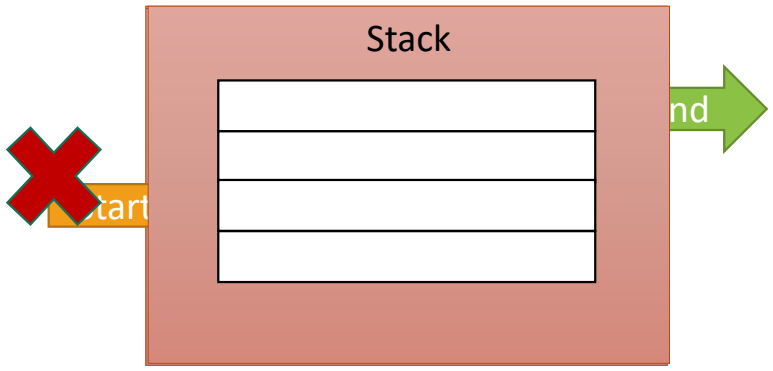
- Stack:
  - Alle value types worden in de Stack geplaatst
  - De waarde zelf komt terecht in de Stack
- Heap:
  - Wanneer we een reference type creëren wordt er in de Stack een pointer aangemaakt en in de Heap het eigenlijke object
  - In de Heap wordt de waarde opgeslagen en in de Stack een verwijzing naar de positie in de Heap
  - We kunnen meerdere pointers naar hetzelfde object laten wijzen, ook in de Heap zelf!



En nu met code …

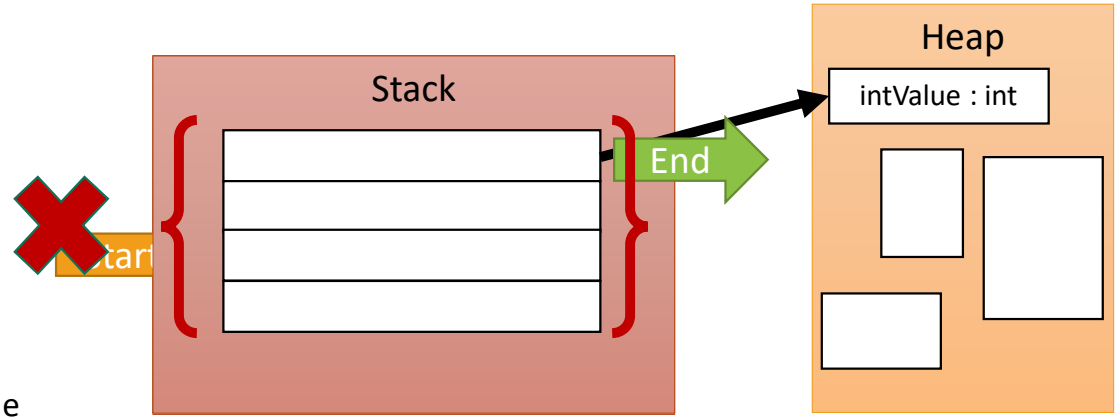
```
public int AddNumbers(int a,int b) {
   int result;
   result = a + b;
   return result;
}
```

1. De variabelen worden op de stack geplaatst
2. De lokale variabele wordt op de stack geplaatst
3. Nadat de berekening is gemaakt wordt het resultaat teruggegeven
4. De functie variabelen worden van de stack verwijderd



En wat als we een reference type gebruiken?

```
public class CustomInt {
    public int intValue;
}
public CustomInt AddNumbers(int a,int b) {
    CustomInt result=new CustomInt();
    result.intValue = a + b;
    return result;
}
```



1. De variabelen worden op de stack geplaatst
2. CustomInt is een reference type en wordt op de Heap geplaatst. Daarna wordt een pointer naar de Heap locatie op de stack geplaatst
3. Nadat de berekening is gemaakt, wordt het resultaat teruggegeven
4. De functie variabelen worden van de stack verwijderd
5. De GC verwijdert het element van de Heap als alle referenties ernaar verwijderd zijn

Predefined Value types van C#

- Numeriek:
  - Integers:
    - Signed : sbyte, short, int, long
    - Unsigned: byte, ushort, uint, ulong
  - Reals:
    - float, double
    - Decimal
  - Logisch:
    - bool
  - Character:
    - char

Basis types in .Net

Natuurlijke getallen en reële getallen.

Integers

Signed = kan negatieve getallen bevatten

C#	Systeem type	Suffix	Lengte
sbyte	SByte		8 bits
short	Int16		16 bits
int	Int32		32 bits
Long	Int64	L	64 bits

Unsigned = kan **geen** negatieve getallen bevatten

C#	Systeem type	Suffix	Lengte
byte	Byte		8 bits
ushort	UInt16		16 bits
uint	UInt32	U	32 bits
ulong	UInt64	UL	64 bits

- Elk C# type correspondeert met een .Net systeemtype
- De capaciteit is afhankelijk van de bit lengte en of het getal negatief kan zijn
- We kunnen een suffix gebruiken om een waarde toe te kennen

```
var num = 5L;
Console.WriteLine(num.GetType());        => System.Int64
```

Werken met integers

- Binair ? Hexadecimaal? Wasda?
  - 1+1 = 10 ? => Yep!
  - Als je de machten van 2 kent dan ken je het binair talstelsel!

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	128	64	32	16	8	4	2	1	
byte a = 44;	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	
byte a = 15;	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	
byte a = byte.MaxValue;	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	= 255
sbyte sb = sbyte.MaxValue;	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	= 127
Wat als? … **sb+=1**; ?	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	= **-128**

  - Als je binair kent, ken je ook hexadecimaal => 0b1111 =0xF, 0001 1010 = 1A

    (mits wat oefening dan toch …)

Operatoren en integers

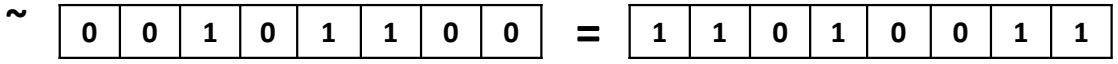
- rekenkundige operatoren: **+, -, *, /, %**
  - % = modulus (de rest van een deling)
    - Console.WriteLn(20%8); => 4
- Increment en decrement operatoren : **++ , --**
  - Int i=0;
    - Console.WriteLn(i++); => 0
    - Console.WriteLn(++i); => 1
- **checked** operator

```
int x = int.MaxValue;
int a = x * 2;                  => -2
int b =checked( x * 2);         => OverflowException
```

- Er zijn **geen** rekenkundige operatoren voor 8 & 16 bit integers !

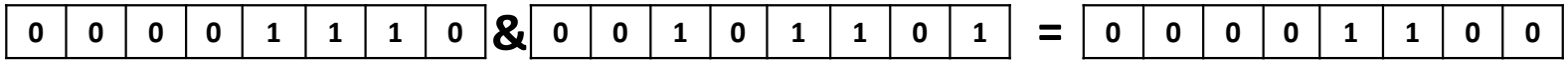
Bitwise operators

- Compliment: ~



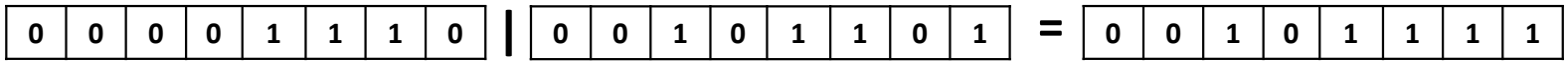
~44 = 211

- AND: &



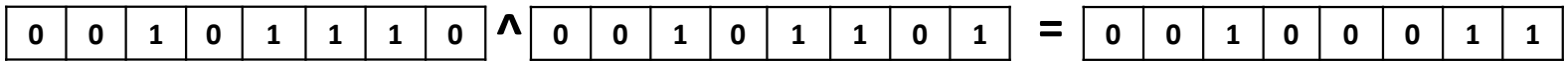
14 & 45 = 12

- OR: |



14 | 45 = 47

- XOR: ^



14 ^ 45 = 35

Conversie van integers

- Impliciete conversie:
  - Kan enkel als elke mogelijke waarde ook mogelijk is in de ontvanger:
    - int x=105001;
    - long y = x; => OK
    - short = x; => ERROR
- Expliciete conversie:
  - Tussen types waar een overflow kan optreden of de waarden anders geïnterpreteerd kunnen worden:
    - Int x=500;
    - short y = (short) y; => OK, maar op eigen risico !
- Automatische conversie van 8 & 16 bit types bij rekenkundige bewerkingen naar Int32 (int)

```
short x =1;
short y = 2;
short res = x + y;          => Compile error -> short res = (short) (x + y);
```

Reële getallen

- Door het systeem gekende reële getallen:

C#	Systeem type	Suffix	Lengte	Precisie
float	Single	F	32	6-9 digits
double	Double	D	64	15-17 digits
decimal	Decimal	M	128	28-29 digits

- Het verschil tussen float/double en decimal:
  - float en double worden gebruikt voor wetenschappelijk berekeningen
  - decimal wordt vooral gebruikt voor financiële berekeningen.
  - Omdat **double binair** werkt en **decimal** met het **tientallig** stelsel kunnen er **afrondingsfouten** optreden!
  - Berekeningen met decimal zijn ± **10x trager** dan met double !

Werken met reële getallen

- Declaratie:
  - Alle declaraties met een punt worden automatisch omgezet naar een double:

    ```
    double x = 1.5;
    ```

  - Daarom moeten bij de declaratie van float en decimal de suffixes worden gebruikt:

    ```
    float f = 1.5F;
    Decimal = 1.5M;
    ```

- Conversies:
  - Impliciet tussen van float naar double
  - decimal is steeds expliciet
  - Van int naar float en double kunnen er soms precisiefouten optreden

Specifieke waarden bij reële getallen

- NaN (Not a Number)
  - Vb: bij deling van 0 door 0
  - Opgepast NaN ≠ NaN !

```
double d = 0.0 / 0.0;
Console.WriteLine("d == NaN ?" + (d==double.NaN));          => False
Console.WriteLine("d is NaN ?" + (double.IsNaN(d)));        => True
```

- Infinity
  - Negatief en positief:

```
Console.WriteLine(1.0 / 0.0);          => ∞
Console.WriteLine(-1.0 / 0.0);         => - ∞
```

QUIZ

Wat is de output van volgende code?

```
Console.WriteLine(2L.GetType());
```

Vraag:

a) int
b) Sytem.Int64
c) long
d) Error

Labo

Werken met getallen

Creëer een functie om °C om te zetten:

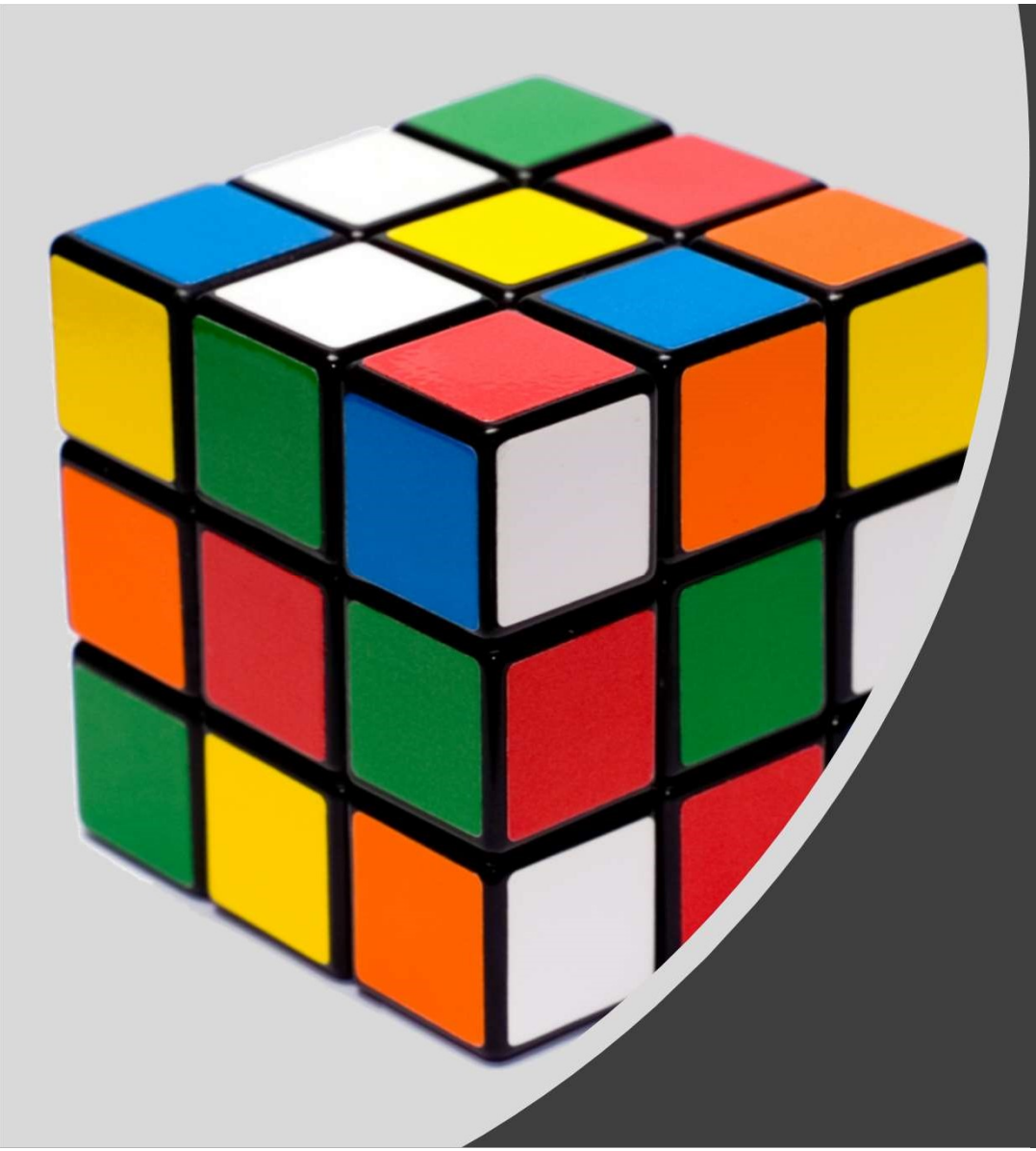
- Oefening :
  - Functionaliteit:
    - Schrijf een functie die graden Celsius omzet naar graden Fahrenheit en graden Kelvin.
      - Formule:
        - Kelvin: °C + 273
        - Fahrenheit: °C x 18 / 10 + 32
  - Test cases:
    - Input:
      1. 0°C
      2. 100°C
      3. 28°C
    - Output:
      1. 32°F en 273K
      2. 212°F en 373K
      3. 82°F en 301K

Maak een functie om te testen op veelvouden:

- Oefening :
  - Functionaliteit:
    - Maak een functie om te testen of getal ‘x’ een veelvoud is van ‘y’.
    - Vergelijk x en y en schrijf of x al dan niet een veelvoud is van y in een mooie zin.
  - Test cases:
    - Input:
      1. x:20 , y:10
      2. x:20 , y:7
    - Output:
      1. 20 is een veelvoud van 10
      2. 20 is geen veelvoud van 7

Creëer een conversie functie van kilometer naar mijl en omgekeerd:

- Oefening :
  - Functionaliteit:
    - Schrijf de functionaliteit die een afstand in kilometer omzet naar mijlen en omgekeerd.
      - Formule:
        - Km -> mijl : x 1,60934
        - Mijl - > kilometer: x 0,621371
  - Test cases:
    - Input:
      1. 1.5km
      2. 6,3 km
      3. 10 mijl
    - Output:
      1. 0.932057 mijl
      2. 3.91464 mijl
      3. 16.0934 km



Basis types in .Net

Logische operatoren

&

Karakters

Boolean

- Test op ‘true’ en ‘false’
  - Value types worden beoordeeld op de inhoud
  - Reference types worden standaard beoordeeld op hun reference en niet op de inhoud! (Dit kan enkel met ‘operator overloading’ waar we later op terugkomen)

- Testen op gelijkheid met ==, !=, <, >, >=, <=
  - Opgepast met reële getallen te vergelijken. Deze kunnen afrondingsfouten bevatten

- Conditionele operatoren : && (AND) en || (OR)
  - && en || zijn short-circuit, & en | niet. Daarom gebruiken we altijd && en || !

Karakters

- Standaard werkt .Net met Unicode (UTF-16) encoding
- Een ‘char’ is dus **2 bytes** groot.
- Initialiseren met ‘ ‘:

```
char a = 'a’;
```

- Speciale karakters met een escape karakter: ‘\ ‘

```
char newLine = '\n';
char singleQuote = '\‘’;
```

- We kunnen elk Unicode karakter voorstellen met hun hexadecimaal equivalent:

```
char phi = '\u0278';
```

Basis types in .Net

Reference types:

String

&

Object

Predefined reference types in .Net

- string
- object

string

- Hoewel een string een reference type is, heeft dit type ook kenmerken van een value type:
  - De toekenning kan gebeuren door waarden en zonder een new:

```
string tekst = "Hallo iedereen!";
```

  - Een copy kopieert de inhoud en niet de referentie!

```
string copy = tekst; => copy krijgt de waarde "Hallo iedereen!";
```

  - Een logische vergelijking vergelijkt de waarden en niet de referentie:

```
string tekst = "Hallo iedereen!";
string copy = "Hallo iedereen!";
Console.WriteLine(tekst==copy);          => True
```

- MAAR een string kan ook null zijn !

```
string empty = null;
```

- Alle escape karakters van een char zijn ook geldig in een string:

```
string tekst = "Hallo \"iedereen\"!";    => Hallo “iedereen”!
```

Het gebruik van een string object

- Concatenatie:
  - Met een + operator: `string concat = "hallo " + tekst;`
    - De plus operator laat ook toe dat andere objecten gebruikt kunnen worden

      In dit geval wordt de standaard 'ToString()' functie aangeroepen:

      ```
      string concat = "hallo alle " + 5;
      ```
- Interpolatie:
  - Interpolatie laat het toe om andere expressies te gebruiken binnen de string
  - Dit kan indien we de waarde laten voorafgaan door het $ teken:

    ```
    short x =5;
    string tekst =  $"hallo alle {x}" ;
    ```
- Gebruik van 'literal value' @ :
  - Laat geen escape karakters toe: `string path = @"c:\documenten\tmp";`
  - Laat toe om meerdere lijnen te gebruiken:

    ```
    string path =  @"dit is een string
                     met meerdere lijnen" ;
    ```

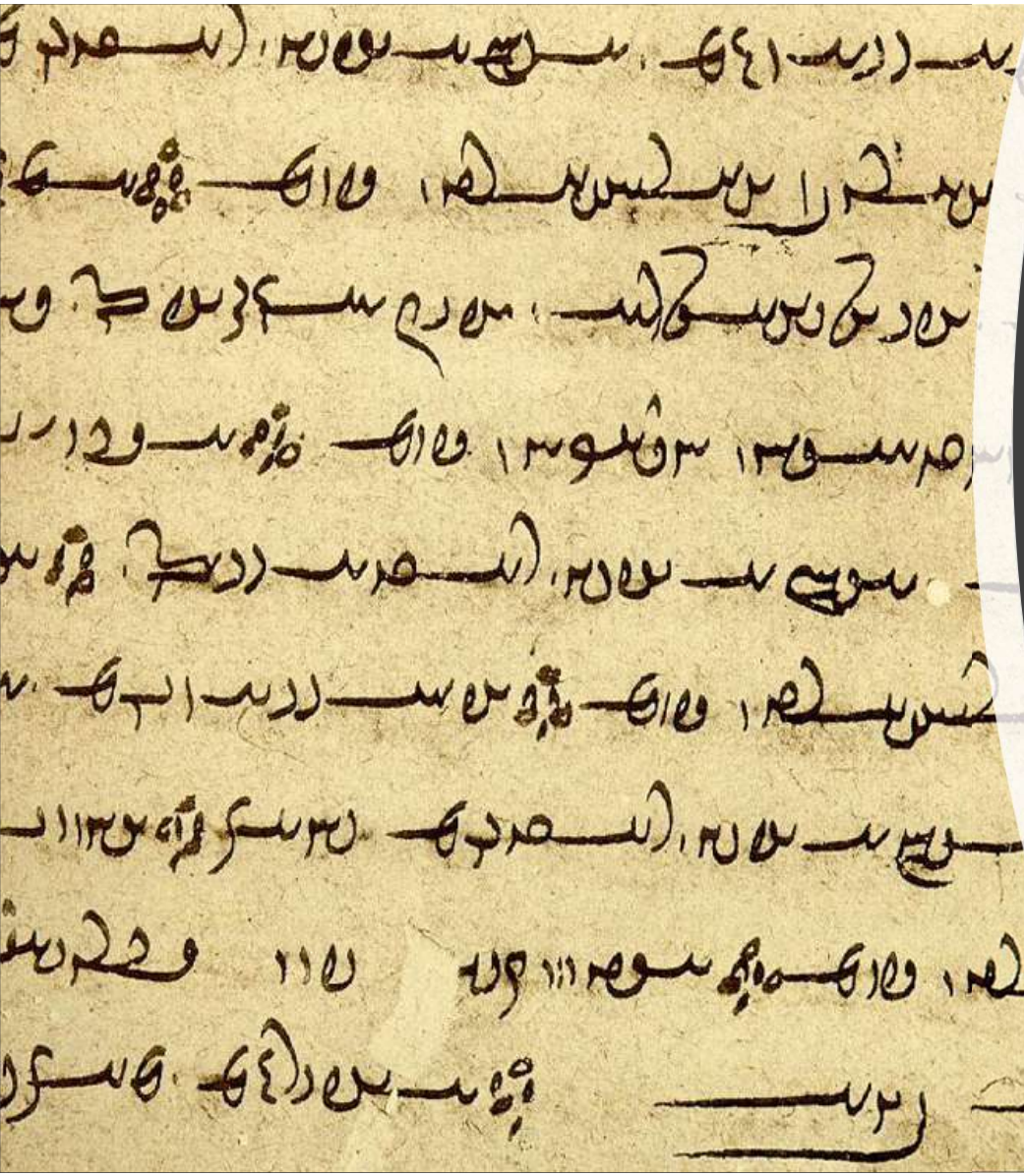
String functies

- String functies
  - Het string object bevat verschillende functies om de string te manipuleren:

    Bijvoorbeeld:

```
string tekst =  $"hallo allemaal!" ;
string subString = tekst.Substring(6,4);       => "alle"
```

- Bekijk de mogelijkheden van het string type in de online help van .Net
  - Ga op de ‘string’ declaratie staan en druk ‘F1’. Visual studio opent de browser en gaat naar



Labo

Werken met tekst

Verwijderen van een karakter in een string:

- Functionaliteit:
  - Maak een functie die een string aanvaardt en één karakter op een gegeven plaats verwijdert
- Test cases:
  - Input:
    1. SyntraAB, 6
    2. snoep, 1
    3. geven, 2
    4. fout , 8
  - Output:
    1. SyntraB
    2. soep
    3. geen
    4. fout

Basis types in .Net

Objecten:

Classes & structs

Objecten ontwerpen

- Leren denken in objecten
  - Door problemen in kleinere beheersbare stukken te hakken, maken we het geheel veel overzichtelijker.
  - Elk object behoort zijn specifieke bevoegdheden en data te bezitten. All round objecten moeten vermeden worden!
  - Objecten vormen de bouwstenen voor verdere ontwikkelingen.
  - Objecten komen ook in ons dagelijks leven voor: meubels, een tafel is een meubel, een stoel is een meubel,… Ze zijn verschillend maar hebben gezamelijke kenmerken en behoren tot eenzelfde groep.
- Creatieve luiheid
  - Door elk onderdeel af te bakenen in gebruik wordt het makkelijker deze onderdelen later opnieuw te gebruiken.
  - Hergebruik (zonder de oorspronkelijke werking aan te tasten) zorgt er ook voor dat dezelfde fouten niet meerdere malen opduiken binnen een programma.

Klassen

- Classes zijn de meest gangbare vorm om referentie types te creëren
- De bouwstenen om een klasse te creëren zijn:
  - Class attributen:
    - public, (protected, private), internal, abstract, sealed, static en partial
  - Class keyword
  - De klasse naam
  - Binnen de haakjes:
    - Constructors
    - Functies
    - Properties
    - Fields
    - Finalizers
    - Overloaded operators
    - Nested types

```
public class DemoClass {
    int myVar;
    public int MyVar {
           get { return myVar; }
           set { myVar = value; }
    }
    public void DoSomething() {
      // No!
    }
  }
}
```

Bouwstenen van een klasse

Constructors

- Een **constructor** heeft het uitzicht van een functie maar moet dezelfde naam hebben van een klasse en kan geen return type specifiëren

```
public class DemoClass {
   int myVal;
   public DemoClass(int startVal) { myVal = startVal; }
}
```

- Er kunnen meerdere constructors gemaakt worden binnen 1 class met verschillende parameters.
- Indien er geen constructor wordt aangemaakt, wordt door de compiler een impliciete lege constructor voorzien.
- Constructors kunnen elkaar aanroepen door het **this** keyword te gebruiken => `public DemoClass() : this(5) { }`
- Niet-publieke constructors zijn mogelijk.

Statische constructors

- Een statische constructor wordt maar 1x uitgevoerd per TYPE

  => Constructor = 1x per instantie

- Triggers:
  - Een eerste instantie wordt aangemaakt van dit type.
  - Er wordt een eerste maal een statische functie van dit type aangeroepen.
- Een statische constructor wordt uitgevoerd vlak voor het eerste gebruik van dit type.
- Een statische constructor mag **geen** parameters hebben en geen access modifier

```
public class DemoClass {
    static DemoClass() { }
 }
```

- Opgepast: Als er een error optreedt tijdens de uitvoering wordt het type onbruikbaar !!!

Destructor

- In uitzonderlijke omstandigheden kunnen we een functie toevoegen die wordt aangeroepen wanneer het object wordt vernietigd door de garbage collector.
- De functie opbouw is: ~ + klasse naam +(). De deconstructor kan geen parameters hebben en heeft geen attributen!

```
public class DemoClass {
   public DemoClass()  { }
   ~DemoClass() { }
}
```

- Niet verwarren met de nieuwe C# 7 functionaliteit ‘Deconstruct(…)’

Fields

- Dit zijn variabelen die behoren aan de globale scope van de class

```
public class DemoClass {
    int myVar;
    public string myText = "hallo";
    readonly int doNotTouch;
    public DemoClass(int startVal) { doNotTouch = startVal; }
}
```

- Field initialisatie:
  - Optioneel
  - Gebeurt onmiddellijk na het creëren van het object nog voor de constructor wordt aangeroepen.
- Readonly fields:
  - Waarden mogen enkel toegekend worden door de constructor of in de declaratie.
  - Kunnen nadien niet meer worden gewijzigd.

Properties

- Lijken op fields in gebruik.
- Worden gespecifieerd door **get** en **set**.
- In de set kunnen waarden worden toegekend door de impliciete parameter **value**

```
int x;
public int X { get { return x; } set { x = value; } }
```

- Het is mogelijk om de get en set anders toegankelijk te maken:

```
public int X { get { return x; } private set { x = value; } }
```

=> In dit geval kan het veld niet extern aangepast worden!

- Impliciete declaratie van properties:

```
public int X { get; set; } = 100;
```

=> Sinds C#6 is het ook mogelijk om impliciete properties te initialiseren

Het gebruik van properties

- Properties laten toe om volledige controle te krijgen over de in- en uitgaande data van een field.

```
public string Text {
    get { if(txt == null) { return string.Empty; } return txt; }
    set { if(!string.IsNullOrEmpty(value)) { txt = value; } }
    }
```

- Toegangscontrole door get of setters niet te implementeren of door restricties op de toegang te plaatsen.

```
public int X { get { return x; } }
public int X {  get { return x; }  internal set { x = value; } }
```

- De mogelijkheid om bewerkingen uit te voeren op een property.

```
public int HeigtInCm { get { return mmHeigt * 10; } }
```

Methodes

- Methode:
  - Kan input ontvangen van de aanroeper door parameters te specifiëren.
  - Voert een reeks van acties uit => statements.
  - Kan output teruggeven indien gespecifieerd:
    - Door een return statement.
    - Door bepaalde parameters te markeren als out of ref.
- Een functie is uniek door de functienaam + parameters.
  - Dit zorgt ervoor dat een functie met dezelfde naam en verschillende parameters toegelaten is => Dit noemt men '**overloading methods**'.
- Speciale functies:
  - Expression bodied: Een methode die een enkele expressie bevat:

```
public int InlineMethod(int x) => x+1;
```

Labo

Werken met objecten

Maak 1 of meerdere klassen:

- Functionaliteit:
  - We willen een weerstation maken. Hier hebben we volgende gegevens nodig:
    - Minimum temperatuur
    - Maximum temperatuur
    - Gemiddelde temperatuur.
    - Temperatuur per uur.
    - Windrichting
    - Windsnelheid
    - Pluviometer
    - Luchtvochtigheidsmeter
  - Uiteraard hebben we ook de juiste tijd en datum nodig.
    - **Extra:** Het zou ook handig zijn moesten we de weerkundige gegevens kunnen bijhouden zodat we historische gegevens kunnen bekijken. Later kunnen we deze gegevens in een database bijhouden.

Ontwerp een console helper klasse:

- Functionaliteit:
  - Maak een klasse die het toelaat om de nodige functionaliteiten toe te voegen aan een standaard console applicatie.
  - De klasse wordt aangemaakt in de Main functie van de console applicatie en wordt uitgevoerd door de ‘Run’ functie.
  - Er moet een cursor getoond worden: bijvoorbeeld : ‘cmd>’. Nadat een commando is uitgevoerd, moet de cursor opnieuw geplaatst worden op een nieuwe lijn.
  - De klasse moet standaard functies kunnen uitvoeren die via de console worden ingevoerd en omgezet naar commando en parameters.
    - Bijvoorbeeld:
      - cursor cli> vervangt de huidige cursor met ‘cli>’
      - Exit verlaat het programma
      - Clear wist het console scherm
      - Color verander de console kleur (voor en of achtergrond)
  - Commando’s zijn niet hoofdlettergevoelig, parameters wel.