C# Taal begrippen

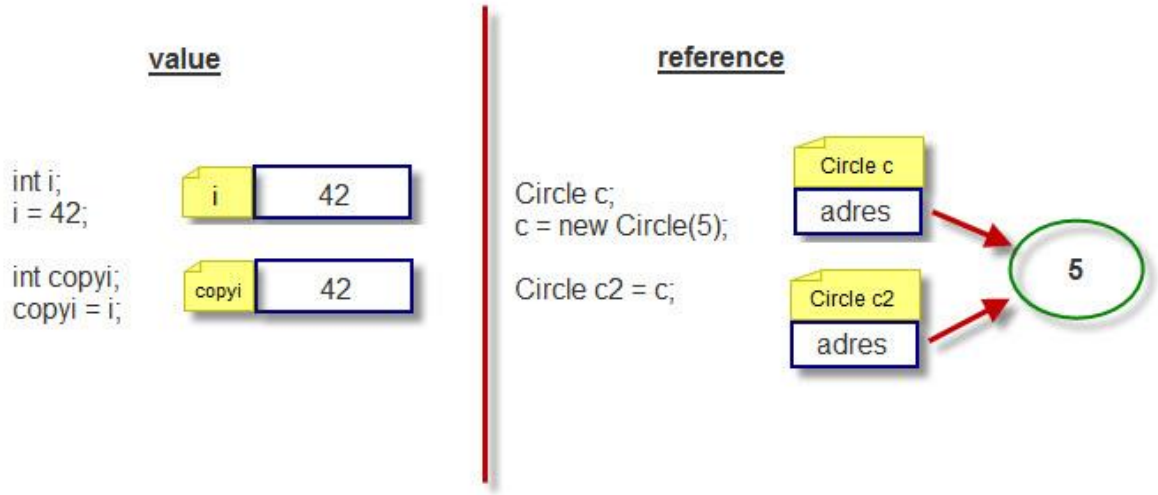
Les 2: C# Taal begrippen

Basis Data Types

Er zijn twee typen in C#:

- Value Types
  - Impliciete data types, structs en enumeraties
- Reference Types
  - Objects en Delegates (afgevaardigden)

Value types worden doorgestuurd als een exacte kopie naar de methode terwijl bij **Reference types** er verwezen wordt naar de plaats waar de waarde is opgeslagen. Zeg maar het adres van waar de waarde staat wordt dan meegegeven aan de method.



In de afbeelding hierboven wordt in de linker helft het value datatype geïllustreerd en in de rechte het reverence datatype. Bij de value datatype bevat de variabele de waarde. Bij de reverence type wordt er verwezen naar de locatie waar de waarde staat. Er zijn bij het reference voorbeeld twee verschillende variabelen die naar de zelfde waarde verwijzen terwijl er in het voorbeeld van de value type er twee verschillende geheugen ruimtes worden gereserveerd om een waarde te bewaren.

C# Taal begrippen

De onderstaande tabel geeft een overzicht van de verschillende datatypen in C#.

C# type	.NET Type	Size in bytes	Description
Integral Types			
byte	Byte	1	May contain integers from 0-255
sbyte	SByte	1	Signed byute from -128 to 127
short	Int16	2	Ranges from -32,768 to 32,767
ushort	UInt16	2	Unsigned, ranges from 0 to 65,535
int(default)	Int32	4	Ranges from -2,147,483,648 to 2,147,483,647
uint	UInt32	4	Unsigned, ranges from 0 to 4,294,967,295
long	Int64	8	Ranges from -9,223,372,036,854 to 9,223,372,036,854,775,807
ulong	UInt64	8	Ranges from 0 to 18,446,744,073,709,551,615
Floating Point types			
float	Single	4	Ranges from $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 digits precision. Requires the suffix 'f' or 'F'
double (default)	Double	8	Ranges from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15-16 digits Precision
Other types			
bool	Boolean	1	Contains either true or false
char	Char	2	Contains any single Unicode character enclosed in single quotation mark such as 'c'
decimal	Decimal	12	Ranges from 1.0×10^{-28} to 7.9×10^{28} with 28-29 digits precision. Required the suffix 'm' or 'M'

Impliciete data types worden in de taal gerepresenteerd als sleutelwoorden (*keywords*) zoals in de tabel op de vorige pagina.

Impliciete data types worden bewaard op de **stack**, terwijl door de gebruiker gedefinieerde referentie types op de **heap** bewaard worden.

De stack

Dit is een stuk geheugen door de processor wordt ondersteund en de grootte ervan wordt bepaald tijdens het compileren. Stack maakt gebruik van het FIFO (First In First Out) principe.

De heap

Deze bestaat uit geheugen dat beschikbaar is voor het programma tijdens diens uitvoering. Referentie type variabelen maken gebruik van de heap om dynamisch geheugen toe te kennen.

Variabelen

Tijdens de uitvoer van een programma wordt data tijdelijk bewaard in geheugen. Een variabele is de naam die gegeven wordt aan een geheugen locatie die de data bewaart. Dus elke variabele heeft een data type en een waarde.

In C# worden variabelen zo gedeclareerd:

```
<data type> <variabele naam>;
```

Bijvoorbeeld:

```
int i;
```

De regel hier boven zal een ruimte van 4 bytes in het geheugen reserveren om een integer type waarde op te slaan.

Je kunt een variabelen initialiseren terwijl je het aan het declareren bent en je kunt meerdere variabelen declareren/initialiseren van het zelfde type in 1 instructieregel (statement).

```
bool isReady = true;
float percentage = 87.88, average = 43.9;
char digit = '7';
```

In C# (net als andere moderne talen), moet je de variabelen eerst declareren voor dat je ze gebruikt.

Ook is er het concept, "Definite Assignment" in C#. Dit zegt dat "lokale variabelen" (variabelen gedefinieerd in een method), eerst moeten worden geïnitialiseerd voor ze gebruikt worden.

Het volgende programma fragment zal niet gecompileerd worden:

```
Static void Main();
{
    int leeftijd;
```

```
    // leeftijd = 18;
    Console.WriteLine(leeftijd);        //error
}
```

Haal echter de commentaar regel weg bij regel 2 en het programma zal dan wel compileren.

Variabelen in C# kunnen alleen waarden van hun eigen datatype opslaan. Je kunt dus geen *char* waarde in een variabelen stoppen die als *int* is gedeclareerd.

Declare, Instantia en Initialize

Declare (declareren), Instantiate (instantiëren) en Initialize (initialiseren), zijn drie termen die veel gebruikt worden en te maken hebben met variabelen, objecten en klasse.

Een variabele bestaat uit twee delen. De **naam** van de variabele en de **waarde**. De naam van de variabele is wat je ***declareert (declare)*** dat het is. De waarde is wat je er aan ***toewijst (assign)***.

Alle variabelen krijgen een initiële waarde wanneer de variabele gedeclareerd wordt. Dus alle variabelen zijn **geïnitialiseerd (initialized)**.

Objecten worden geïnstantieerd. Mensen worden geboren. Objecten worden geïnstantieerd. Een baby is een instantie van een mens, een object is een instantie van een klasse.

De handeling het creëren van een instantie van een klasse, wordt instantiatie genoemd. Dus declareren, initialiseren, en instantiëren komen op de volgende manier bij elkaar:

```
MyClass myClassyReference = new MyClass();
```

Scopes

Het **bereik (scope)** van een variabele bepaalt de zichtbaarheid ervan in de rest van het programma. Wanneer een variabele gedefinieerd is in een methode dan is deze bereikbaar verder in de methode. Gaat het programma over naar een andere methode dan is dezelfde variabele niet bereikbaar in de nieuwe methode.

Er zijn andere mogelijkheden voor het bereik van een variabele. Bijvoorbeeld kan een variabele in een lus of een andere codestructuur worden

gedeclareerd en alleen zichtbaar voor de code binnen deze structuur. Een bredere bereikbare variabele kan worden gedeclareerd op klasse-niveau, zodat het gebruikt kan worden door elke methode in de klasse. In feite, is de bereikbaarheid van een variabele altijd de volledige omvang van het codeblok waarin het gedeclareerd is.

Er zijn sleutelwoorden, **access modifiers**, waar mee de bereikbaarheid van variabelen en objecten beïnvloed kan worden. Denk aan:

- **public**,
  Er zijn geen beperkingen op de toegang tot de openbare leden.

- **private**,
  De toegang is beperkt tot binnen de klasse definitie.

- **protected**,
  De toegang is beperkt tot binnen de klasse definitie en elke klasse die erft van de klasse.

- **internal**,
  De toegang is uitsluitend beperkt tot de klassen gedefinieerd in het huidige projectverzameling.

- **protected interna**l,
  De toegang is beperkt tot het huidige verzameling en typen die afgeleid zijn van de klasse waar ze toe behoren. Alle leden in de huidige project en alle leden in afgeleide klasse kunnen toegang krijgen tot de variabelen.

Method-Level Scope

Variabelen die in een codeblok van een methode gedeclareerd zijn, zijn beschikbaar voor gebruik door een ander deel van de methode, met inbegrip van geneste code blokken. Het volgende voorbeeld demonstreert dit door het creëren van een variabele in de methode, het instellen van de waarde en het gebruik ervan binnen de reikwijdte van een 'if' statement.

```
namespace ConsoleScope_1
{
    class Program
    {
        static void Main(string[] args)
        {
            int score;                                      // Declared at method-level
            score = 100;                                    // Used at method-level

            if (score >= 50)
                Console.WriteLine("Good score : {0}", score); // Used in nested scope
            else
                Console.WriteLine("Poor score : {0}", score); // Used in nested scope
        }
    }
}
```

Nested Scope

Variabelen die in een geneste scope aangegeven zijn, zijn niet zichtbaar buiten hun codeblok. De volgende code zal niet compileren, omdat de variabele die wordt geprobeerd om te gebruiken op methode niveau, gedeclareerd is in de dieper genest omvang van de if-statement.

```
        static void Main(string[] args)
    {
        int score = 100;

        if (score >= 50)
            string message = "Good score";            // Declared in if statement
        else
            string message = "Poor score";            // Declared in if statement

        Console.WriteLine(message);                   // Variable unavailable
    }
}
```

Opmerking: Om dit voorbeeld te laten werken, zou de variabele gedeclareerd moeten worden voordat de ‘if-statement’ en een waarde toegekend worden binnen de if statement.

Constants

Constanten ofwel **constants,** zijn variabelen wiens waarde nooit verandert. Constanten moeten meteen geïnitialiseerd worden wanneer ze gedeclareerd worden.

```
const double PI = 3.1415;
```

Het volgende voorbeeld is dus fout daar de variabele niet wordt geïnitaliseerd.

```
const int WAARDE;
```

Het conventioneel (gebruikelijk) om constante variabelen altijd met hoofdletters te schrijven.

Benaming conventies voor variabelen en methods

Microsoft suggereert om de **Camel Notatie** (eerste letter in kleine letters) te schrijven voor variabelen en de **Pascal Notatie** (eerste letter in hoofdletters) voor methods. Elk woord na het eerste woord bij zowel variabelen als methods moet beginnen met een hoofdletter.

Voorbeelden van variabelen met de Camel Notatie:

```
salaris, totaleSalaris
```

Voorbeelden van method met de Pascal Notatie:

```
GetSalaris(), GetTotaleSalaris()
```

Reken (Arithmetic) operatoren

Operand	Omschrijving
+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen
%	Restwaarde of modulo
++	Verhogen met 1
--	Verlagen met 1

Het onderstaande programma gebruikt deze operatoren als voorbeeld.

```
using System;
nameSpace Operatorentest
{
   Class RekenOpertatoren
   {
       // Dit programma toont het gebruik van de reken operatoren
       // + - * / % ++ --
       static void Main()
       {
            // resultaten van optellen, aftrekken
            // vermenigvuldigen en de modulus operatoren
            int som=0, verschil=0, product=0, modulo=0;
            float quotient=o;                  // resultaat van deling
            int num1 = 10, num2 = 2;       // operator variabelen

            som = num1 + num2;
            verschil = num1 – num2;
            product = num1 * num2;
            quotient = num1 / num2;

            // restwaarde van 3/2
            modulo = 3 % num2;

            Console.WriteLine("num1 = {0}, num2 = {1}", num1, num2);
            Console.WriteLine();

            Console.WriteLine("Som van {0} en {1} is", num1, num2, som);
            Console.WriteLine("Verschil van {0} en {1} is", num1, num2, verschil);
            Console.WriteLine("Produt van {0} en {1} is", num1, num2, product);
            Console.WriteLine("Quotient van {0} en {1} is", num1, num2, quotient);
            Console.WriteLine();
            Console.WriteLine("Restwaarde van 3 gedeeld door {0} is {1}", num2, modulo);
            num1++;
            num2--;
            Console.WriteLine("num1 = {0}, num2 = {1}", num1, num2);
       }
   }
}
```

Opmerkingen:

```
Console.WriteLine("num1 = {0}, num2 = {1}", num1, num2);
```

In de bovenstaande regel worden {0), {1} en {2} vervangen door de waarden in de variabelen num1, num2 en som.

De opdracht num1++; heeft dezelfde betekenis als:

```
num1 = num1 + 1;
```

en

```
Num1 += 1;
```

Prefix en Postfix notatie

De operatoren ++ en – kunnen als **prefix** en als **postfix** operatoren gebruikt worden. Als prefix:

```
num1 = 3;
num2 = ++num1;       // num1 = 4, num2 = 4
```

De compiler zal eerst num1 met 1 verhogen en daarna deze toekennen aan num2. Terwijl bij de postfix vorm:

```
Num2 = num1++;       // num1 = 4, num2 = 3
```

De compiler zal de waarde van num1 toekennen aan num2 en daarna num1 met 1 verhogen.

Toekennings (Assignment) operatoren

Operand	**Omschrijving**
=	Simpele toekenning
+=	Optel toekenning
-=	Aftrek toekenning
*=	Vermenigvuldig toekenning
/=	Delen toekenning
%=	Modulo toekenning

De (=) operator wordt gebruikt om een waarde toe te kennen aan een object.

```
bool isBetaald = false;
```

In de regel hier boven wordt de waarde false aan de variabele isBetaald van het booleaanse type gegeven.

Soms wordt **casting** gebruikt om van type te veranderen (type conversion). Bijvoorbeeld om een waarde van een variabele van het type double toe te kennen aan een variabele van het type int.

```
double dubbeleWaarde = 4.67;
// intWaarde zal worden 4
int intWaarde = (int) doubbeleWaarde;
```

Er is bij casting wel het gevaar van verlies van precisie.

Relationele (Relationship) operatoren

Operand	Omschrijving
==	Gelijk
!=	Niet gelijk
>	Groter dan
<	Kleiner dan
>=	Groter dan of gelijk
<=	Kleiner dan of gelijk

Relationele operatoren worden gebruikt om te vergelijken in conditionele instructies.

Relationele operatoren geven altijd een Booleaanse instructie, dus altijd een resultaat van waar of niet waar (true or false).

```
int num1 = 5, num2 = 6;

num1 == num2   // false
num1 !=  num2   // true
num1 >   num2   // false
num1 <   num2   // true
num1 <= num2   // true
num1 >= num2   // false
```

Alleen gelijk soortige datatypes kunnen met elkaar worden vergeleken.

```
int i = 1;
bool b = true;
```

Je kunt dus niet i en b met elkaar vergelijken of ze gelijk zijn (i==b). Dit zal een syntax fout tot gevolgen hebben.

Logische (logica land Bitwise) operatoren

Operand	Omschrijving
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
!	Bitwise NOT
&&	"Logical" of "short circuit" AND
||	"Logical" of "short circuit"

The operatoren &, | en ^ worden zelden gebruikt.

The logische operatoeren '&&'en '||' worden gecombineerd om verschillende vergelijkingen te kunnen maken.

```
int i = 6, j = 12;
bool eersteVar = i>3 && j<10;

// eersteVar wordt false

Bool tweedeVar = i>3 || j<10;

// tweedeVar wordt true
```

In de eerste vergelijking: i>3 && j<10 zal alleen waar zijn als beide condities i>3 en j<10 waar zijn.

De tweede vergelijking: i>3 || j<10 zal waar zijn wanneer één van de vergelijkingen waar is.

Je kunt beide, && en||, in een enkele instructie gebruiken zoals:

```
bool eersteVar = (i>3 && j<10) || (i<7 && j>10)    // eersteVar is waar
```

In de instructie hierboven zijn booghaakjes gebruikt om de condities te groeperen en ambiguïteit (dubbelzinnigheid) te voorkomen.

Andere operatoren

In de tabel hieronder zie je enkele andere operatoren die in C# aanwezig zijn:

Operand	Omschrijving
<<	Links verschuiven bitwise operator
>>	Rechts verschuiven bitwise operator
.	Member access for objects
[]	Index operator gebruikt bij arrays en collections
()	Cast operator
?:	Ternary operator

Prioriteiten van operatoren

Operatoren worden niet gelijk behandeld, er is een prioriteiten volgorde.

```
Int i =2 + 3 * 6;
// i wordt 20 niet 30
```

3 zal vermenigvuldigd worden met 6 en het resultaat zal worden opgeteld met 2.

In principe dezelfde regels al bij het rekenen. Voor meer in formatie over prioriteiten volgorde, raadpleeg de MSDN documentatie of de .NET Framework documentatie.