Programmeren Basis: tutorial

Auteur: Nico Kuijpers
Datum: 8 april 2018

Inleiding

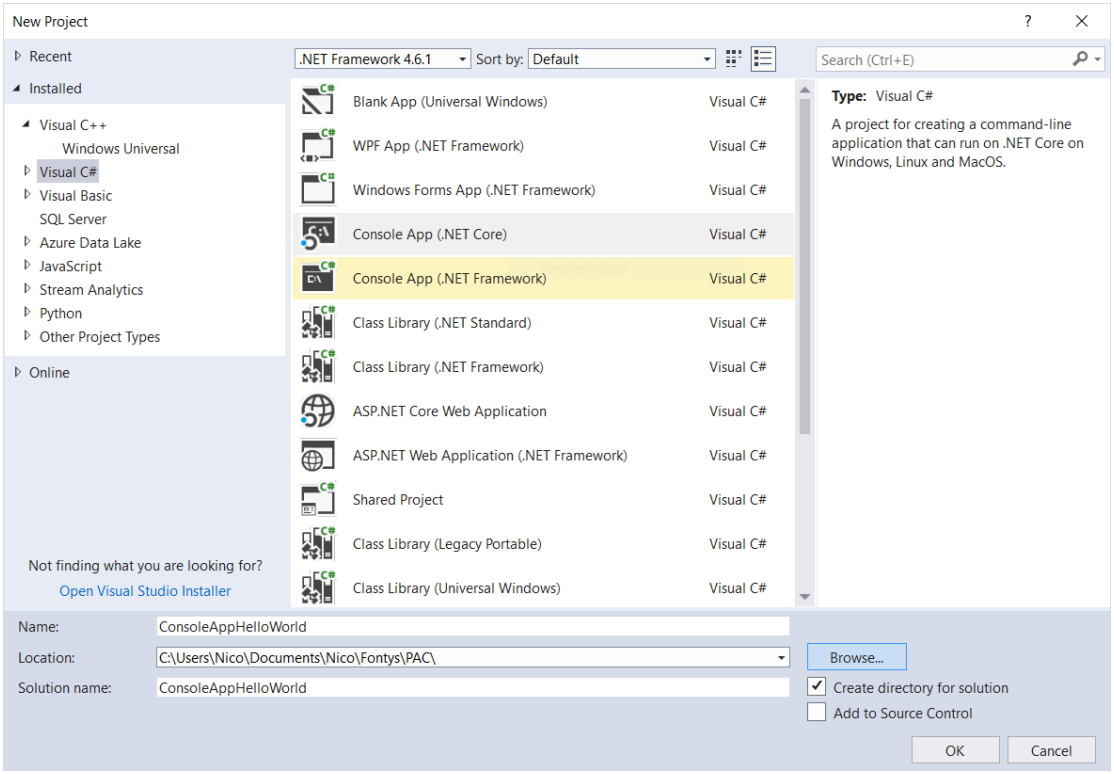
In deze tutorial maak je kennis met programmeren in de programmeertaal C# (spreek uit: See-Sharp). C# is een zogenaamde object-georiënteerde programmeertaal die veel wordt gebruikt op scholen en in het bedrijfsleven. Ook bij Fontys Hogeschool voor ICT wordt veel in C# geprogrammeerd. In deze tutorial leer je hoe je een Console applicatie kunt maken en leer je programmeren met variabelen, `if`-statement, `switch`-statement, `while`-loop, `for`-loop en methoden. Deze termen zijn nu misschien nog nieuw voor je, maar dat is geen probleem. Als je deze tutorial doorloopt zal het allemaal duidelijk worden.

Om te beginnen heb je een computer nodig met daarop een programmeeromgeving. Dat is een computerprogramma waar je nieuwe programma's mee kunt schrijven. In deze cursus maken we gebruik van de programmeeromgeving Visual Studio. We gaan ervan uit dat je deze applicatie al geïnstalleerd hebt op je laptop of desktop computer.

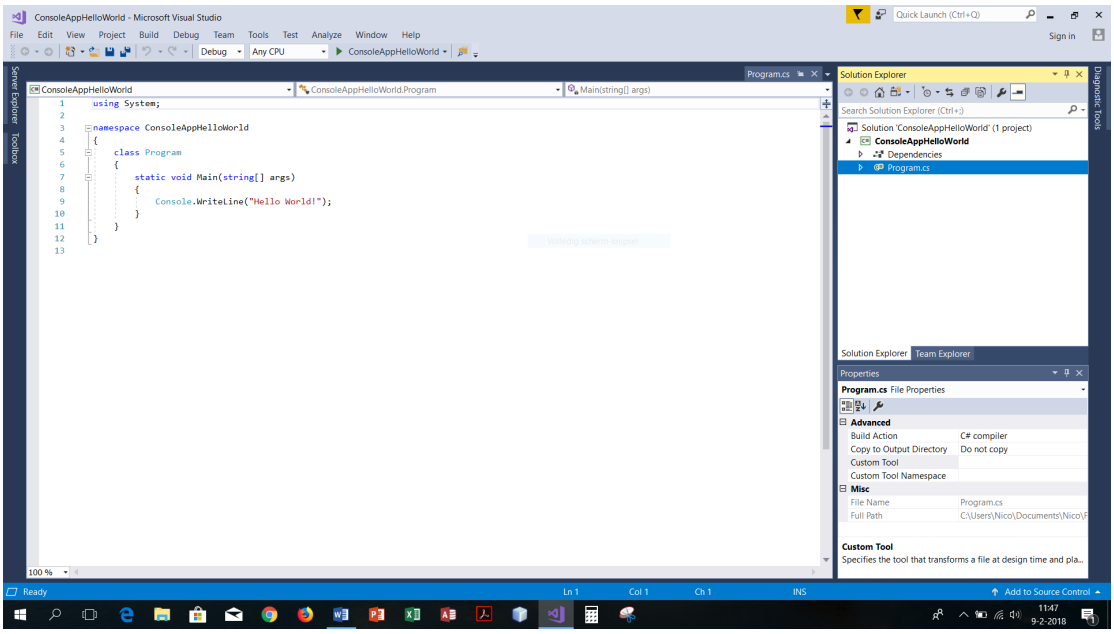
Hello World!

Bijna iedere programmeur die kennismaakt met een nieuwe programmeertaal schrijft als eerste een programma waarmee je "Hello World!" op het beeldscherm kunt schrijven. Ook wij gaan als eerste een programma schrijven maken dat de woorden "Hello World!" op het scherm laat zien. Dit doen we met een zogenaamde Console-applicatie. Voer nu de onderstaande stappen uit:

1. Start Visual Studio
2. Kies File → New → Project
3. Klik op Visual C# (zie figuur op de volgende pagina)
4. Kies Console App (.NET Core)
5. Vul bij Name in: ConsoleAppHelloWorld; de naam achter Solution Name verandert mee.
6. Wijzig de locatie naar een folder van jouw keuze.
7. Klik op OK.



In de Solution Explorer, klik op Program.cs. Dit is het programma dat er voor zorgt dat de tekst “Hello World!” op het scherm verschijnt.

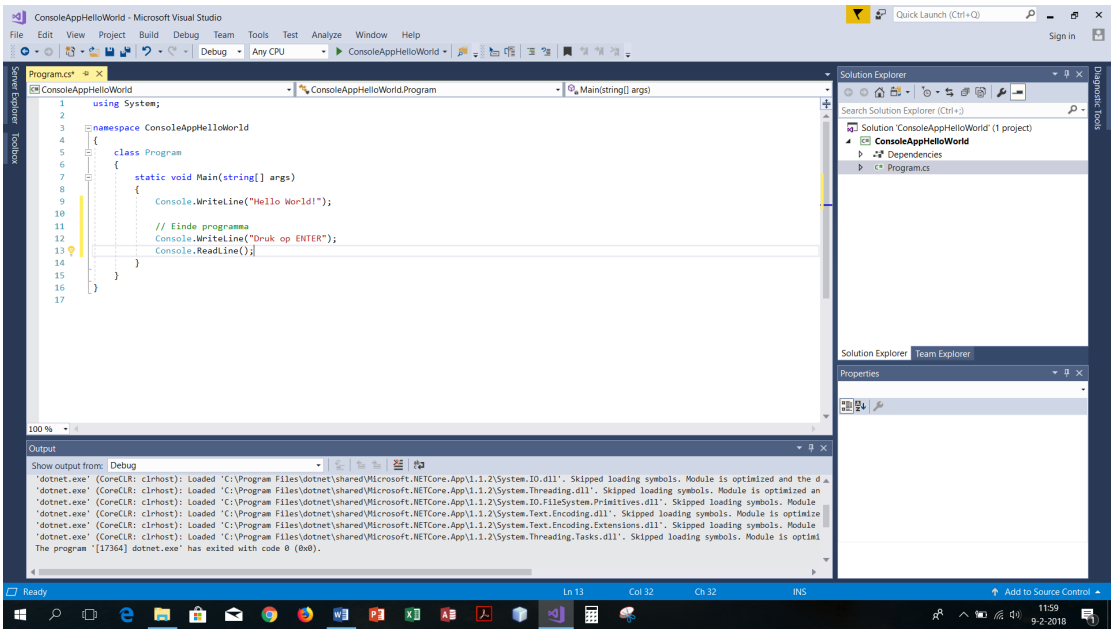


Om het programma uit te voeren (te runnen), klik op de knop met het groene driehoekje waar ConsoleAppHelloWorld bij staat. Je ziet dan een zwart scherm met de tekst “Hello World!”. Helaas verdwijnt het scherm vrijwel meteen en heb je misschien de tekst niet eens kunnen lezen. We kunnen dit probleem oplossen door de volgende programmaregels toe te voegen (zie de figuur op de volgende pagina):

```
        Console.WriteLine("Hello World!");

        // Einde programma
        Console.WriteLine("Druk op ENTER");
        Console.ReadLine();
```

De regel die begint met `//` is een commentaar-regel. Dit is uitleg voor de programmeur, maar bij het uitvoeren van het programma wordt hier niets mee gedaan. De volgende regel zorgt ervoor dat de tekst “Druk op ENTER” op het scherm komt te staan en de laatste regel zorgt ervoor dat het programma wacht totdat je op ENTER hebt gedrukt.



Als je het programma nu opnieuw uitvoert verschijnen de tekst-regels “Hello World!” en “Druk op ENTER” op het scherm. Het zwarte scherm wordt het Console genoemd en verdwijnt zodra je op ENTER hebt gedrukt.

Je hebt nu je eerste programma in C# geschreven:

```
using System;

namespace ConsoleAppHelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

            // Einde programma
            Console.WriteLine("Druk op ENTER");
            Console.ReadLine();
        }
    }
}
```

Hierboven wordt de ‘namespace’ ConsoleAppHelloWorld weergegeven. Hierin staat de C# klasse (engels: class) `Program`. De klasse `Program` begint met `class Program { ` en eindigt met de een-na-laatste `}`.

Iedere klasse in C# kan *velden* (engels: field) en *methoden* (engels: method) bevatten. Velden zijn bedoeld om informatie in op te slaan, bijvoorbeeld getallen zoals jouw leeftijd of stukjes tekst zoals jouw naam. Methoden zijn kleine stukjes programma. Binnen een methode kun je informatie opvragen die in de velden staat opgeslagen. Het is ook mogelijk om binnen een methode een andere methode "aan te roepen". Dan wordt het stukje programma uitgevoerd dat in die andere methode staat. Op die manier kun je dus een programma schrijven dat uit meerdere methoden bestaat die elkaar kunnen aanroepen.

Er is altijd een methode waar het programma begint en dat is de `Main` method. In het programma Program.cs begint deze methode met `static void Main(string[] args) {` en eindigt de methode bij de eerstvolgende `}`. Alles wat tussen `{` en `}` van de `Main` methode staat wordt uitgevoerd door de computer. In het programma Program.cs is dat bijvoorbeeld

```
Console.WriteLine("Hello World!");
```

Dit wordt een *statement* genoemd en met bovenstaand statement wordt "Hello World!" op het scherm geschreven. Binnen een methode kun je meerdere statements na elkaar plaatsen. Een enkelvoudig statement eindigt altijd met een punt-komma, maar zoals we later zullen zien geldt dit niet voor samengestelde statements waarbij meerdere en enkelvoudige of samengestelde statements tussen `{` en `}` kunnen worden geplaatst.

De betekenis van `static` is lastig uit te leggen, maar het is belangrijk om te onthouden dat bij de `Main` methode altijd `static` moet staan, anders kan het programma niet gestart worden. De betekenis van `void` wordt ook later uitgelegd.

Rekenspel

Zo, nu wordt het tijd om zelf een programma te schrijven. We beginnen met een programma waarin we de gebruiker vragen om naam en leeftijd. Vervolgens vragen we hem of haar of hij/zij zin heeft in een rekenspelletje. Als hij/zij dan met "ja" antwoordt gaan we het spelletje spelen, maar als hij/zij niet met "ja" antwoordt, dan laten we zien dat de computer goed is in rekenen.

Aan de hand van deze tutorial zul je leren wat variabelen zijn, wat een selectie-statement (`if`-statement) is en en wat een herhaling-statement (`for`-statement en `while`-statement) is. Daarnaast behandelen we ook het `switch`-statement en de aanroep van methoden. Om tekst naar het scherm te schrijven en gebruikersinvoer af te handelen maken we gebruik van de methoden `WriteLine()` en `ReadLine()` van de standaard-klasse `Console`.

Maak een nieuwe console applicatie met de naam ConsoleAppRekenspel op dezelfde manier als dat je ConsoleAppHelloWorld hebt gemaakt:

1. Kies File → New → Project
2. Klik op Visual C#
3. Kies Console App (.NET Core)
4. Vul bij Name in: ConsoleAppRekenspel
5. Wijzig de locatie naar een folder van jouw keuze.
6. Klik op OK.

Voeg nu enkele programmaregels toe binnen de `Main`-methode, zodat het onderstaande programma ontstaat:

```
using System;

namespace ConsoleAppRekenspel
{
    class Program
    {
        static void Main(string[] args)
        {
            // Begroeting
            Console.WriteLine("Hallo!");

            // Vraag naam en sla op in variabele naam van type string
            Console.WriteLine("Hoe heet jij?");
            string naam = Console.ReadLine();

            // Nog een begroeting, maar nu met de naam erbij
            Console.WriteLine("Hallo " + naam);

            // Einde programma
            Console.WriteLine("Druk op ENTER");
            Console.ReadLine();
        }
    }
}
```

Als je dit programma uitvoert verschijnt eerst “Hallo” op het scherm en zal worden gevraagd hoe je heet. Als je je naam hebt ingevoerd gevolgd door ENTER zal een tweede begroeting verschijnen, namelijk “Hallo” gevolgd door jouw naam. Het programma wordt afgesloten nadat je op ENTER hebt gedrukt.

In het programma wordt dus naar de naam van de gebruiker gevraagd. Een naam is eigenlijk een rijtje tekens (engels: characters) en wordt in de variabele `naam` van type `string` opgeslagen. Een variabele kun je zien als een ‘vakje’ waar je een ‘waarde’ in kunt bewaren. In dit geval kun je dus je eigen naam bewaren in het vakje `naam`, onder de voorwaarde is dat de variabele `naam` van het type `string` is. De programmaregel met `Console.ReadLine()` zorgt ervoor dat de variable `naam` wordt gevuld met het rijtje tekens dat door de gebruiker wordt ingetypt.

Voeg nu de onderstaande regels toe voor de regel `// Einde programma`:

```
        // Vraag leeftijd en sla op in variabele leeftijd van type int
        // Om een int te krijgen moeten we de string invoer converteren naar int
        Console.WriteLine("Hoe oud ben jij?");
        string invoer = Console.ReadLine();
        int leeftijd = Convert.ToInt32(invoer); // Converteer string naar int

        // Print leeftijd
        Console.WriteLine("Jij bent " + leeftijd + " jaar oud");
```

In deze programmaregels wordt eerst naar de leeftijd gevraagd. De leeftijd wordt in eerste instantie als een string opgeslagen in de variabele `invoer` van het type `string`. Vervolgens wordt de leeftijd ‘geconverteerd’ naar een geheel getal en opgeslagen in variabele `leeftijd` van type `int`, dat een geheel getal representeert. Het converteren gebeurt door middel van methode-aanroep `ToInt32()` van de standaard-klasse `Convert`. Tot slot wordt de leeftijd op het scherm getoond.

Het is mogelijk om een string samen te stellen met behulp van het +-teken:

```
"Jij bent " + leeftijd + " jaar oud"
```

In dit geval wordt de integer waarde van variabele leeftijd eerst geconverteerd naar een string, bijvoorbeeld de string "`18`". Vervolgens worden de drie strings "`Jij bent `", "`18`" en "` jaar oud`" aan elkaar 'geplakt' met het +-teken.

Behalve `string` en `int` zijn er ook andere typen. Enkele voorbeelden zijn `bool`, `char`, `float`, `double` en `decimal`. Het type `bool` wordt gebruikt om logische waarden in op te slaan. Er zijn slechts twee mogelijke waarden voor een `bool`, namelijk `true` (waar) en `false` (niet waar). Het type `char` wordt gebruikt om een enkel karakter in op te slaan, bijvoorbeeld de letter 'a'. De typen `float`, `double` en `decimal` worden gebruikt om reële getallen (komma getallen) in op te slaan. Een voorbeeld van een reëel getal is 3.14159265359. Het verschil tussen `float` en `double` is dat `double` nauwkeuriger is (double precision) dan `float` en daardoor meer computergeheugen gebruikt. Het is mogelijk om reële getallen volgens de 'wetenschappelijke' notatie op te slaan (bijvoorbeeld `1.7E18`) in `float` en `double`. Dit is niet mogelijk met een `decimal`, maar daar staat tegenover dat met de `decimal` preciezer kan worden gerekend. De `decimal` wordt vaak gebruikt voor berekeningen met geld.

We hebben eerder gezegd dat je een variabele kunt zien als een 'vakje' waar je 'waarden' in kunt bewaren. De waarde die in het vakje wordt gestopt kan steeds veranderen. Bijvoorbeeld in onderstaande programmaregels is de waarde van variabele `x` uiteindelijk gelijk aan 2:

```
int x; // Declareer variabele x van type int
x = 1; // Ken waarde 1 toe aan x
x = 2; // Ken waarde 1 toe aan x
```

In de bovenste regel wordt de variabele `x` *gedeclareerd*. Dat wil zeggen dat er een variabele met de naam `x` bestaat van het type `int`. Deze variabele heeft echter nog geen waarde (er zit niets in het vakje). In de volgende regel wordt de waarde `1` toegekend aan variabele `x` en in de laatste regel wordt de waarde `2` toegekend aan variabele `x`. De waarde `1` is nu niet meer opgeslagen in een 'vakje' en is dus 'vergeten'. We hadden net zo goed kunnen schrijven:

```
int x = 2;
```

Merk op dat een variabele alleen 'bekend' is in de programmeregels tussen `{` en `}` waarbinnen de variabele is gedeclareerd. Dit deel van het programma noemen we de *scope* van de variabele.

De volgende stap in ons programma is om de gebruiker te vragen of hij zin heeft in een rekenspelletje. Als hij dan met "ja" antwoordt roepen we de methode `Rekenen()` aan, maar als hij niet met "ja" antwoordt, dan roepen we de methode `AlleTafels()` aan.

Voeg nu de onderstaande regels toe voor de regel `// Einde programma`:

```
// Vraag de gebruiker of hij een spelletje wil doen
// Sla het antwoord op in variabele antwoord van type string
Console.WriteLine("Zullen we een rekenspelletje doen? (ja/nee)");
string antwoord = Console.ReadLine();
```

```
// Afhankelijk van het antwoord: rekenen of tafel printen
if (antwoord.Equals("ja"))
{
    // Aanroep van methode Rekenen
    Rekenen();
}
else
{
    // Aanroep van methode AlleTafels
    AlleTafels();
}
```

Het is nu niet mogelijk om het programma uit te voeren. Dit komt doordat de methoden `Rekenen()` en `AlleTafels()` nog niet zijn gedefinieerd. Om het programma te kunnen uitvoeren kun je de aanroep van deze methoden tijdelijk weghalen door `//` toe te voegen, bijvoorbeeld: `// AlleTafels()`. De aanroep van de methode wordt dan commentaar en is dus niet langer onderdeel van het programma.

De programmaregels met `if` en `else` worden een selectie-statement (of `if`-statement) genoemd. Er zijn twee mogelijkheden: als het antwoord gelijk is aan "ja" (`antwoord.Equals("ja")` is waar) dan worden alle programmaregels na `if (antwoord.Equals("ja"))` tussen `{` en `}` uitgevoerd. In dit geval wordt dan de methode `Rekenen()` aangeroepen. Als het antwoord niet gelijk is aan "ja" worden de programmaregels na `else` tussen `{` en `}` uitgevoerd. In dit geval wordt dan de methode `AlleTafels()` aangeroepen.

Methode `Rekenen()`
Hieronder staat de methode `Rekenen()`, deze kun je overnemen in het programma direct onder de methode `Main()`, dus na de `}` die onder de 's' van `static` staat.

```
// Methode om rekenspelletje te spelen
private static void Rekenen()
{
    // Variabele doorgaan van type bool
    // De variabele doorgaan krijgt waarde true
    bool doorgaan = true;

    // Maak een instantie van klasse Random aan
    // Deze instantie kan willekeurige getallen genereren
    Random rnd = new Random();

    // Ga door zolang doorgaan gelijk is aan true
    while (doorgaan)
    {
        // Bepaal een willekeurig getal tussen 1 en 10 (10 niet inbegrepen)
        // Sla dit getal op in variabele a van type int
        int a = rnd.Next(1, 10);

        // Bepaal nog een willekeurig getal tussen 1 en 10
        // Sla dit getal op in variabele b van type int
        int b = rnd.Next(1, 10);

        // Bereken het product a maal b
        // Sla het resultaat op in variabele product
        int product = a * b;

        // Vraag naar de uitkomst van a maal b
        // Sla de uitkomst op in variabele uitkomst van type int
```

```
            Console.WriteLine("Hoeveel is " + a + " maal " + b + "?");
            string invoer = Console.ReadLine();
            int uitkomst = Convert.ToInt32(invoer);

            // Controleer of de uitkomst gelijk is aan het product
            // Om twee waarden te vergelijken moet je twee =-tekens gebruiken
            if (uitkomst == product)
            {
                Console.WriteLine("Het antwoord " + uitkomst + " is goed!");
            }
            else
            {
                Console.WriteLine("Het antwoord " + uitkomst + " is fout!");
                Console.WriteLine("Het juiste antwoord is " + product);
            }

            // Vraag of de gebruiker nog een keer wil
            // Sla het antwoord op in variabele antwoord van type string
            Console.WriteLine("Nog een keer? (ja/nee)");
            string antwoord = Console.ReadLine();

            // Als het antwoord gelijk is aan nee, dan niet meer doorgaan
            if (antwoord.Equals("nee"))
            {
                doorgaan = false;
            }
        }
    }
```

In bovenstaande programmaregels staat een zogenaamde `while`-loop. Dit is een manier om een stukje programma steeds te herhalen. De variabele `doorgaan` van het type `bool` krijgt de waarde `true`. Vervolgens wordt steeds opnieuw gekeken of de variabele `doorgaan` gelijk is aan `true`. Als dat zo is, dan worden alle programmaregels achter `while (doorgaan)` tussen `{` en `}` uitgevoerd. In plaats van `while (doorgaan)` zou je ook `while (doorgaan == true)` kunnen schrijven, dat heeft precies hetzelfde effect. Het programma blijft doorgaan totdat de variabele `doorgaan` de waarde `false` krijgt. Dat gebeurt als de gebruiker het antwoord “nee” geeft op de vraag of hij/zij het spelletje nog een keer wil spelen. Merk op dat `Equals("nee")` case-sensitive is, dus als de gebruiker “Nee” invoert zal het programma toch doorgaan.

In het bovenstaande stukje programma zien we ook dat we een willekeurig getal tussen 1 en 10 (10 niet inbegrepen) kunnen bepalen door aanroep van de methode `rnd.Next(1,10)`. Deze methode wordt aangeroepen op de instantie `rnd` van de klasse `Random`. Voordat de methode kan worden aangeroepen moet eerst de instantie `rnd` worden aangemaakt. Dit gebeurt in de programmaregel

```
Random rnd = new Random();
```

Hierin staat: maak een nieuwe instantie aan van de klasse `Random` en onthoud een verwijzing naar deze nieuwe instantie in variabele `rnd`. Deze variabele `rnd` moet van het type `Random` zijn.

Tafel printen met `while`-loop

Hieronder staat een methode om de tafel van een opgegeven getal af te drukken met behulp van een `while`-loop. Voeg deze methode toe na de methode `Rekenen()`.

```
// Methode om tafel te printen voor een getal met while-loop
private static void TafelMetWhileLoop(int getal)
{
    int n = 1;
    while (n <= 10)
    {
        Console.WriteLine(" " + n + " maal " + getal + " is " + (n*getal));
        n = n + 1;
    }
}
```

In de `Main`-methode kun je bovenstaande methode aanroepen door de regel `AlleTafels()` te vervangen door `TafelMetWhileLoop(5)`. De `5` is de waarde voor de parameter met de naam `getal` van het type `int`. In dit voorbeeld wordt de waarde `5` toegekend aan de parameter `getal` in de aanroep van de methode `TafelMetWhileLoop()`. Tijdens het uitvoeren van de methode zal de tafel van `5` op het scherm worden getoond.

In bovenstaande programmaregels staat een `while`-loop. Met dit statement is het mogelijk om een stukje programma steeds te herhalen. In dit geval worden de programmaregels tussen `{` en `}` na `while (n <= 10)` steeds herhaald zolang de variabele `n` een waarde heeft die niet groter is dan `10`. Door tussen `{` en `}` de waarde van `n` steeds met `1` te verhogen zal `n` uiteindelijk de waarde `11` krijgen en stopt de `while`-loop. Let erop dat je geen punt-komma plaatst na `while (n <= 10)`, want anders stopt de `while`-loop nooit.

Tafel printen met `for`-loop

Hieronder staat een methode om de tafel van een opgegeven getal af te drukken met behulp van een `for`-loop in plaats van een `while`-loop. Voeg deze methode toe na de methode `TafelMetWhileLoop()`.

```
// Methode om tafel te printen voor een getal met for-loop
private static void TafelMetForLoop(int getal)
{
    for (int i = 1; i <= 10; i = i + 1)
    {
        Console.WriteLine(" " + i + " maal " + getal + " is " + (i*getal));
    }
}
```

In de `Main`-methode kun je bovenstaande methode aanroepen door de regel `TafelMetWhileLoop(5)` te vervangen door `TafelMetForLoop(5)`. Ook hier wordt de waarde `5` toegekend aan parameter `getal` van type `int`.

In bovenstaande methode zien we een zogenaamde `for`-loop. Net als een `while`-loop is ook dit een manier om een stukje programma steeds te herhalen. In het begin krijgt de integer variabele `i` de waarde `1`. Zolang de variabele `i` kleiner of gelijk is aan `10` worden de programmaregels achter `for (int i = 1; i <= 10; i = i + 1)` tussen `{` en `}` uitgevoerd. Na uitvoering van deze regels wordt variabele `i` verhoogd met `1` door middel van `i = i + 1`. Vervolgens wordt opnieuw bekeken of `i` kleiner of gelijk is aan `10`, etc. Ervaren programmeurs schrijven vaak `i++` in plaats van `i = i + 1`.

De `for`-loop is dus eigenlijk een afkorting voor de `while`-loop met teller. Dit kun je goed zien als je de programmaregels van de methoden `TafelMetWhileLoop()` en `TafelMetForLoop()` met elkaar vergelijkt.

Methode `AlleTafels()`

In plaats van alleen de tafel van `5` te laten zien, kunnen we ook alle tafels van 1 tot 10 laten zien. Voeg nu de onderstaande methode toe aan je programma:

```
// Methode om alle tafels van 1 tot 10 te printen
private static void AlleTafels()
{
    Console.WriteLine("Alle tafels van 1 tot 10");
    for (int i = 1; i <= 10; i = i + 1)
    {
        Console.WriteLine("De tafel van " + i + " is");
        TafelMetForLoop(i);
    }
}
```

In bovenstaande programmaregels wordt de methode `TafelMetForLoop()` aangeroepen binnen een `for`-loop en wordt als waarde voor de parameter de waarde van loop-variabele `i` meegegeven. Met andere woorden: de methode `TafelMetForLoop()` wordt 10 keer aangeroepen met parameter-waarde 1 tot en met 10.

In de `Main`-methode kun je bovenstaande methode aanroepen door de regel `TafelMetForLoop(5)` te vervangen door `AlleTafels()`.

Van getal naar woord met `if`-statement

In plaats van “De tafel van 5 is” zouden we ook “De tafel van vijf is” kunnen schrijven. We moeten dan de integer-waarden 1 t/m 10 omzetten naar strings “een”, “twee”, etc. We schrijven hiervoor een nieuwe methode `GetalNaarWoordMetIfString()` en vervangen de betreffende programmaregel door:

```
string woord = GetalNaarWoordMetIfStatement(i);
Console.WriteLine("De tafel van " + woord + " is");
```

Het resultaat van de methode-aanroep `GetalNaarWoordMetIfStatement()` wordt opgeslagen in variabele `woord` van type `string` en in de volgende regel gebruikt. Hieronder staat de methode:

```
// Methode om getal tussen 1 en 10 om te zetten naar woord
// In deze methode wordt een if-statement gebruikt
// Invoer: getal tussen 1 en 10
// Resultaat: string met het woord dat bij het getal hoort
private static string GetalNaarWoordMetIfStatement(int getal)
{
    string resultaat;
    if (getal == 1)
    {
        resultaat = "een";
    }
    else if (getal == 2)
    {
        resultaat = "twee";
    }
```

```
        else if (getal == 3)
        {
            resultaat = "drie";
        }
        else if (getal == 4)
        {
            resultaat = "vier";
        }
        else if (getal == 5)
        {
            resultaat = "vijf";
        }
        else if (getal == 6)
        {
            resultaat = "zes";
        }
        else if (getal == 7)
        {
            resultaat = "zeven";
        }
        else if (getal == 8)
        {
            resultaat = "acht";
        }
        else if (getal == 9)
        {
            resultaat = "negen";
        }
        else if (getal == 10)
        {
            resultaat = "tien";
        }
        else
        {
            resultaat = "onbekend";
        }

        // Geef het resultaat terug
        return resultaat;
    }
```

Net als bijvoorbeeld methode `TafelMetForLoop()` heeft ook deze methode een parameter `getal` van het type `int`. Het resultaat van deze methode is een `string`. Tot nu toe hebben we alleen methoden geschreven zonder een `return`-statement. Deze methoden geven bij de voltooiing geen waarde als resultaat terug en daarom staat er `void` voor de naam de van methode. De methode `GetalNaarWoordMetIfStatement()` geeft een `string` als resultaat. Om dit aan te geven staat er `string` voor de naam van de methode en eindigt de methode met `return resultaat;`

De juiste string wordt bepaald middels een `if`-statement. De waarde van `resultaat` is afhankelijk van van de waarde van de parameter `getal`. Zoals je kunt zien is het dus mogelijk om binnen de `else` een nieuw `if`-statement op te nemen. Dit noemen we het "nesten" van `if`-statements. Overigens is het ook mogelijk om `for`-loops of `while`-loops te "nesten" (meerdere loops in elkaar) of een `if`-statement te plaatsen binnen een loop, of een loop te plaatsen binnen een `if`-statement, etc. Op deze manier is het mogelijk om complexe programma's te schrijven.

Van getal naar woord met `switch`-statement

In plaats van zo'n uitgebreid `if`-statement wordt vaak het `switch`-statement gebruikt. Hieronder staat een methode met hetzelfde gedrag, maar dan met het `switch`-statement. Voeg deze methode ook toe aan je programma en roep deze methode aan in plaats van de vorige methode.

```
// Methode om getal tussen 1 en 10 om te zetten naar woord
// In deze methode wordt een switch-statement gebruikt
// Invoer: getal tussen 1 en 10
// Resultaat: string met het woord dat bij het getal hoort
private static string GetalNaarWoordMetSwitchStatement(int getal)
{
    string resultaat;
    switch(getal)
    {
        case 1:
            resultaat = "een";
            break;
        case 2:
            resultaat = "twee";
            break;
        case 3:
            resultaat = "drie";
            break;
        case 4:
            resultaat = "vier";
            break;
        case 5:
            resultaat = "vijf";
            break;
        case 6:
            resultaat = "zes";
            break;
        case 7:
            resultaat = "zeven";
            break;
        case 8:
            resultaat = "acht";
            break;
        case 9:
            resultaat = "negen";
            break;
        case 10:
            resultaat = "tien";
            break;
        default:
            resultaat = "onbekend";
            break;
    }

    // Geef het resultaat terug
    return resultaat;
}
```

De juiste string wordt bepaald middels een `switch`-statement en is afhankelijk van de waarde van `getal`. Let op: bij een `switch`-statement is het van belang om na elke `case` een `break` te plaatsen. Als je dit niet doet, dan wordt automatisch ook de eerstvolgende `case` uitgevoerd totdat er wel een `break` staat. In het geval de parameter `getal` een waarde heeft die niet gelijk is aan 1, 2, 3, 4, 5, 6, 7, 8, 9 of 10, zal

het resultaat gelijk worden aan “onbekend”. In dat geval worden namelijk uiteindelijk de programmaregels uitgevoerd die na `default:` staan.

Nu heb je een programma met daarin voorbeelden van het gebruik van variabelen, `if`-statement, `switch`-statement, `for`-loop, `while`-loop en methode-aanroepen. De tip is om dit programma goed te bewaren zodat je altijd een programmeervoorbeeld bij de hand hebt. Indien je twijfelt over een onderdeel van je programma, kijk dan in de uitwerking van deze tutorial of je de programmaregels op de juiste manier hebt overgenomen.