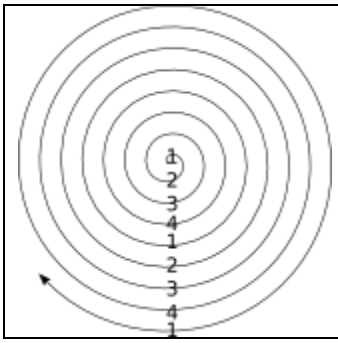
De basis van programmeren

Les 1: De basis van programmeren

Het **spiraalmodel** (zoals hieronder afgebeeld) is één van de manieren waarop programma's gemaakt worden.



1. Probleemstelling. Wat wordt er gevraagd? Wat is de bedoeling?
2. Probleemoplossing/ontwerpen. Welke methodes en technieken? Welke stappen? Welk algoritme?
3. Implementeren. Het algoritme als programmacode verwerken.
4. Controle/evaluatie.

Als je al iets weet van programmeren dan heb je nu al ideeën hebben van een programma dat je wilt schrijven. Wanneer je voor iemand anders, of een bedrijf een programma gaat ontwikkelen dan heeft die een visie. Het is jouw taak als ontwikkelaar/programmeur om deze visie om te zetten naar een werkend programma.

Neem bijvoorbeeld het bouwen van een eigen huis.

1. Je gaat bij een architect om je ideeën voor het huis te verwoorden. De architect luistert naar je ideeën en probeert deze te visualiseren. Het is hier heel belangrijk dat de architect en de klant elkaar goed begrijpen.
2. De architect maakt een bouwtekening en eventueel een schaalmodel of een 3D-model.
3. De architect gaat daarna samen met de klant de plannen doornemen. Mocht de klant dingen willen veranderen dan moet de architect opnieuw de tekeningen/3D-model aanpassen. Wanneer er een akkoord bereikt wordt kan begonnen worden met de bouwfase.
4. Het bouwen van het huis zelf gebeurd door een bouwbedrijf, niet de architect. Het is wel de taak van de architect om het werk te controleren.
5. Het kan voorkomen dat de klant tijdens de bouw toch dingen wil bijsturen (bijvoorbeeld een extra badkamer) en dan moet het proces helemaal opnieuw beginnen.

Het verschil tussen programmeren en het bouwen van een huis is dat programmacode makkelijker te wijzigen is dan het aanpassen van een gebouw. Makkelijker betekent echter niet dat het 'gemakkelijk' is om te doen.

Deelproblemen

Voor simpele problemen lijkt de oplossing soms heel eenvoudig. Het is echter heel belangrijk om te leren één probleem op te splitsen in deelproblemen. Dit maakt het eenvoudiger om een complex probleem te kunnen oplossen. Door het goed opsplitsen kunnen de verschillende deelproblemen door meerdere mensen worden uitgewerkt.

Vraag 1:

Vaak voert men zonder het te beseffen deelproblemen uit. Stel je krijgt een groepsopdracht voor het vak Informatica om een verslag te maken over een onderwerp binnen dit vakgebied.

Schrijf de stappen op om zo'n opdracht goed uit te voeren.

Voor het maken van een programma zijn er dus twee taken te onderscheiden:

1. **Analyseren** van wat men wilt maken.
2. **Ontwerpen** van een oplossing.
3. **Programmeren** van de oplossing.
4. **Testen** van de oplossing.

De taken van het analyseren van het probleem en het ontwerpen van een oplossing worden gedaan door een **systeemanalist** *(de architect)*. Het schrijven van de broncode, en dus het realiseren van het programma gebeurd door een **programmeur** *(het bouwbedrijf)*. Het controleren of het programma werkt naar behoren wordt gedaan door de klant en de systeemanalist.

Voorbereiding op coderen

Voordat er begonnen wordt met het coderen ofwel intikken van programma code, zijn er nog een aantal punten die uitgelegd moeten worden daar deze zeker zullen worden tegengekomen.

Fouten

Tijdens het schrijven van bron code kunnen er verschillende soorten fouten optreden. Er wordt een onderscheiding gemaakt tussen **Syntax fouten**, **Semantische fouten** en **Pragmatische/Logische fouten.**

Syntax fouten zijn programmeerfouten door een typefout, verkeerde datatype voor een variabele of andere verkeerd getypte coderingsfouten. Syntax fouten zijn niet toestaan. Het programma zal niet door de computer uitgevoerd worden. De programeer zal deze fouten eerst moeten verbeteren voor dat de compiler de broncode vertaald naar machine code die door de computer kan worden uitgevoerd. Enkele veel voorkomende fouten zijn eenvoudig fouten die niet lang moeten duren om te repareren.

Soorten Syntax fouten

- **Punt-Komma**
  Puntkomma fouten, deze komen vaak voor in talen afgeleid van de C - programmeertaal. C - stijl talen beëindigen een instructie regel altijd met behulp van de puntkomma. Dit vertelt de compiler dat de volgende regel een nieuwe instructie is. Het vergeten van een puntkomma veroorzaakt een syntax error, zodat de compiler niet door gaat met het vertalen van de broncode naar machine code. Programmeertalen die eisen dat puntkomma's gebruikt worden om een instructie te beëindigen zijn onder andere C#, C++, Perl en Java. Veel van deze compilers zijn in staat om te bepalen in welke regels van de broncode de vereiste puntkomma ontbreken. Voeg de puntkomma toe en compileer de broncode opnieuw. Dit lost deze fout(en) op.

- **Brackets**
  Sommige talen gebruiken openings- en sluithaken om een blok code dat bij elkaar hoort aan te duiden. Deze haken, ook wel "curly - braces " genoemd, worden gebruikt om een deel van de code te omsluiten. De blokken van code waar de haken nodig zijn, de "if" statements, de "while", de "for" loops en de "try-catch" blokken. Als de programmeur per ongeluk vergeet deze haken toe te voegen, geeft de compiler een syntax error. Talen die haken nodig om deze code blokken te omsluiten zijn bijvoorbeeld Javascript, Java, C#, C en C++.

- **Hoofdletter gevoelig (Case sensitive)**
  Veel programmeertalen zijn hoofdletter gevoelig. Dit betekent dat de variabele "the**V**ariable" niet hetzelfde is als de variabele "thevariable". Door de hoofdletter "V" in de eerste variabele en een kleiner letter "v" in de andere, creëert de compiler een geheel nieuwe variabele, en dit kan een syntax fout veroorzaken. Programmeurs plaatsen de variabele namen aan de top van de functies, zodat ze gemakkelijk kunnen verwijzen naar de juiste spelling en case lay-out. Als de programmeur per ongeluk een verkeerde schrijfwijze van een variabele gebruikt dan

treedt er een syntax fout op, in talen zoals Visual Basic, C, C++, C#, Javascript en Java.

Een tweede type programeer fout is de **Semantische fout**. Hier wordt het programma correct vertaald door de compiler en worden er geen waarschuwingen of fouten gemeld, maar wanneer het programma wordt uitgevoerd kan het vast lopen. Let op soms kan het ook wel goed gaan.

Het derde type fouten die men tegen kan komen tijdens het programmeren zijn de **Pragmatische** ook wel **Logische fouten**. Bij dit type fouten werkt het programma correct, er waren geen waarschuwingen en of foutmeldingen tijdens het compileren en het loopt ook niet vast, maar het programma doet niet wat men verwacht had dat het moest doen. Neem bijvoorbeeld een auto. Normaal verwacht men dat als men op de rem trapt de auto snelheid vermindert en stil komt te staan. Stel nu dat dat niet gebeurt, in plaats daarvan gaat de auto nog harder rijden.

Debuggen

Een groot deel van de tijd dat je aan het programmeren bent, ben je fouten in het programma aan het opsporen. Dit wordt **Debuggen** genoemd. In het Engels is een fout namelijk een **bug**. De Engelse betekenis van bug is een insect, iets dat lastig is en dat zijn programmafouten ook.

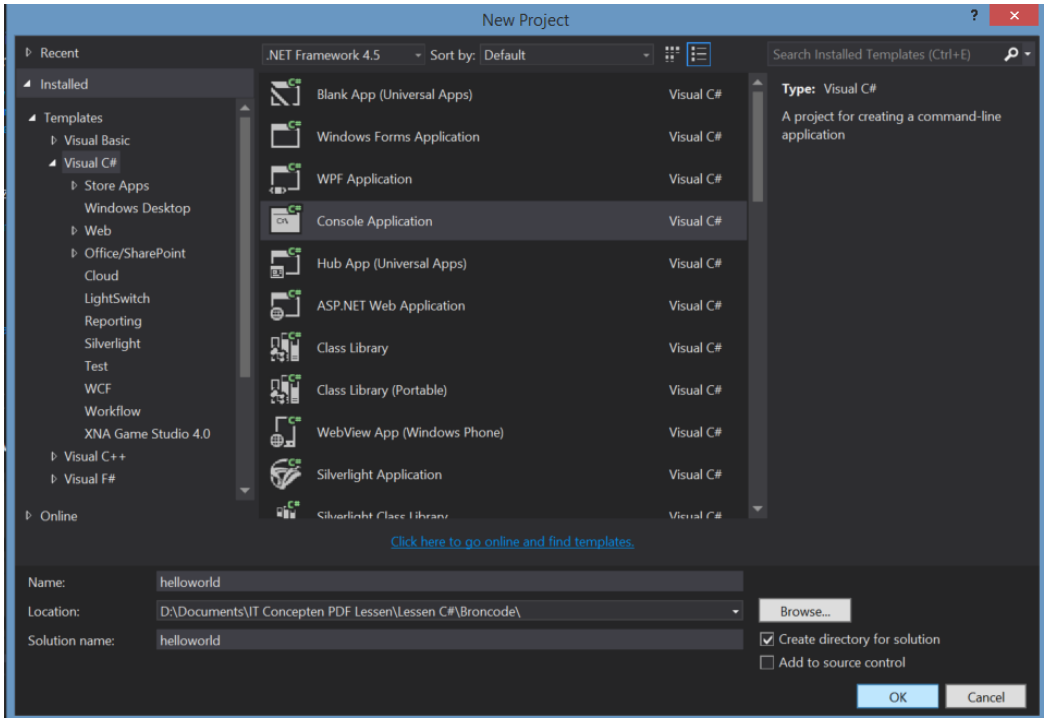
Visual Studio heeft een hulpmiddel waarmee je fouten kunt op sporen, dat is de **Debugger**. Je kunt hier mee regel voor regel door je programmacode gaan en precies bekijken wat de waarden zijn van de verschillende variabelen in het programma. Je kunt ook bepaalde punten in het programma aangeven waar met het debuggen gestart moet worden. Debuggen is vergelijkbaar met het maken van een ooggetuigenverslag, waar je stap voor stap de waarden van de variabelen in het programma registreert in een soort tabel. De debugger is natuurlijk veel handiger dan het maken van een ooggetuigenverslag.

De basis van programmeren

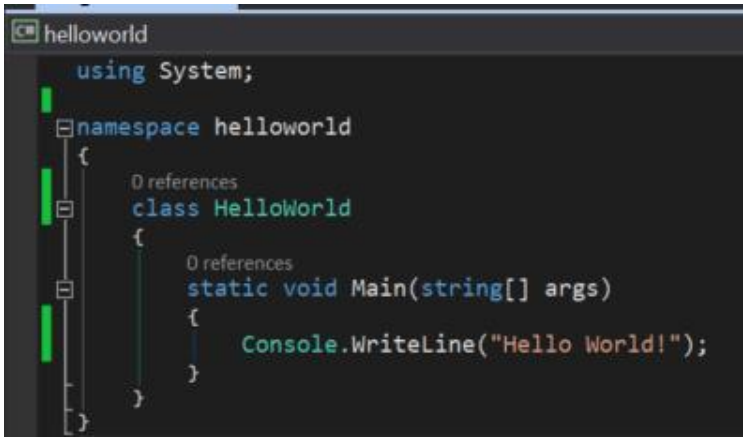
Hello World

Het meest gebruikte voorbeeld om iemand een programmeertaal te leren is het programma waarbij de computer op het scherm de tekst “Hello World!” afdrukt.

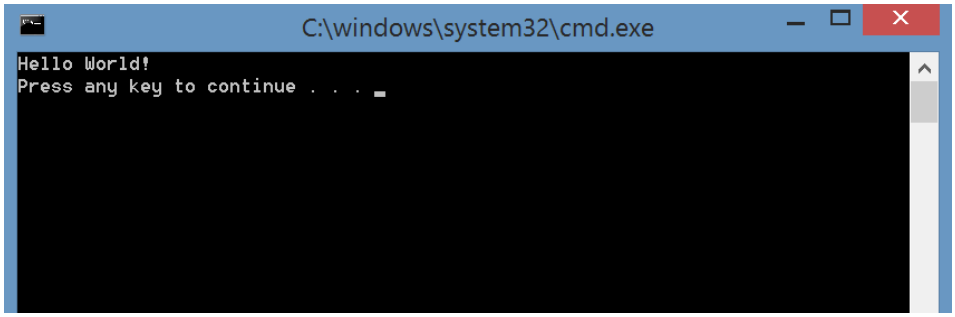
1. Start Visual Studio en maak een nieuw project. Geeft het de naam “helloworld”. Kies de plaats waar VS (Visual Studio) het project moet opslaan. Klik op de OK knop.



2. Pas de programmacode zodanig aan, dat het er precies hetzelfde uitziet als in de afbeelding hieronder:



3. Voer het programma uit door in het menu Debug – Start Without Debugging te kiezen of door de toetsen CTRL+F5 in te drukken.



4. Hierboven staat een afbeelding van de uitvoer van het programma.

De broncode begrijpen

De eerste regel van het programma, using System, komt in practisch elk C# programma voor. Het geeft toegang tot de kern functionaliteiten van de computer.

De tweede regel, namespace helloworld, is een logische verzameling van gerelateerde klassen in C#. Wat een klasse precies is zullen we later bespreken.

In C# programma's mogen geen twee klassen met dezelfde naam voorkomen en om dit tegen te gaan worden namespaces gebruikt.

Een namespace kan classes, events, exceptions, delegates en ook andere namespaces bevatten. Deze laatste noemt men een "internal namespace".

```
Namespace Parent
{
      Namespace Child
      {
       …..
       }
}
```

Het using Sleutelwoord *(Keyword)*

De eerste regel:

```
using System;
```

Het sleutelwoord using laat ons gebruik maken van de klassen die de namespace "System" staan.

Willen we echter gebruik maken van internal/child namespaces dan moet dat specifiek worden aangegeven.

```
Using System.Collections;
```

Het class Sleutelwoord

Alle C# programmas hebben minstens één class. De Main() method bevindt zich in één van deze classes.

De classes zijn een combinatie van data (fields) en functions (methods) die kunnen worden uitgevoerd op de data om zo de oplossing uit te voeren voor ons probleem.

De Main() Method

In de volgende regel definieren we de Main() method van ons programma.

```
Static void Main(string[] args)
```

De Main method is het begin punt van ons programma. Een C# programma begint de uitvoer bij de eerste regel van de Main method en eindigt met de beëindiging van de Main method.

De Main method is static gedeclareerd. Als er een attribuut static voor staat, wil dat zeggen dat we geen instantie van die klasse nodig hebben, om die functie te kunnen uitvoeren. Staat static er niet voor, moeten we eerst een instantie aanmaken.

De method is void omdat er niets terug wordt gestuurd. Main is de standaard naam voor deze method, terwijl string[] args een lijst van parameters die aan het programma kunnen worden mee gestuurd als het van de command line wordt uitgevoerd.

Afdrukken op het Console

De volgende regel drukt de tekst “Hello World!” af op het Console scherm.

```
Console.WriteLine(“Hello World!”);
```

Hier roepen we de static method WriteLine() aan die in de class Console gedefinieerd is. Deze method heeft een string als parameter en drukt deze af op het scherm van de Console.

C# gebruikt de punt (.) operator om toegang te krijgen variabelen en methodes van een klasse.

Booghaken () worden gebruikt om een method te identificeren. Een string waarde wordt geplaatst tussen dubbele aanhalingstekens (“). Elke instructieregel moet in C# eindigen met de puntkomma teken (;). Deze wordt ook wel de statment terminator genoemd.

Commentaar

Commentaar is tekst dat de programmeer toevoegt aan de broncode om uit te leggen wat er in dat deel van de code gebeurt. Commentaar wordt door het vertaalprogramma, **compiler**, genegeerd.

Er zijn drie manieren waarop commentaar in C# kan worden toegevoegd.

Op één regel:

```
// This is my main method of program
Static void Main()
{
….
}
```

Als een blok. Het blok begint met (/*) en eindigt met (*/).

```
Static void Main()
{
      /* These lines of tekst
          will be ignored by the compiler */
      …
}
```

De derde manier is als documentatie commentaar. C# kan deze gebruiken om documentatie te genereren voor classes en het programma.

```
/// These are documentation comments
```

Belangrijke punten om te onthouden

- C# is hoofdletter gevoelig.
- Whitespace (enter, tab, space) worden door de compiler genegeerd.
- Er zijn drie type commentaar, regel, blok en documentatie.
- De grenzen van een namespace, class en method worden gedefinieerd door openings- en sluit-accolade {}.

HelloWorld versie 2

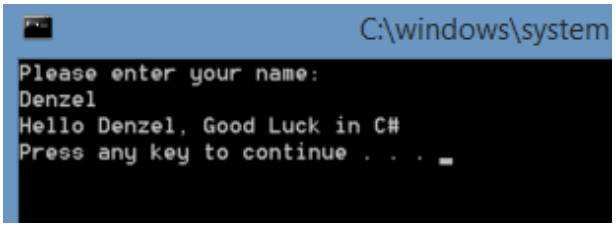
Het *helloworld* programma dat tot nu toe bekeken is, is statisch wanneer het door de computer uitgevoerd wordt.

Laten we het aanpassen en wat interactie met de gebruiker er in zetten. We willen dat het programma om de naam van de gebruiker vraagt en deze dan groet met zijn naam. De code hiervoor ziet er als volgt uit:

```
static void Main(string[] arg)
{
    Console.Write("Please enter your name: ");
    string name = Console.Readline();
    Console.WriteLine("Hello {0}, Good Luck in C#", name);
}
```

Opmerking: De code hierboven is niet van het volledige programma. Alleen de Main method. Om ruimte te besparen. Deze strategie zal vaker gebruikt worden.

Hieronder zie je hoe de uitvoer eruit ziet.



In de eerste regel is er een andere method, Write(), van de Console class gebruikt. Deze is ongeveer hetzelfde als WriteLine(). Het verschil is dat de

computer na het tonen van de tekst niet naar de volgende regel gaat in het console venster.

Op de tweede regel wordt een variabelen gedeclareerd van het type string en het wordt genoemd 'name'. In deze declaratie wordt met de ReadLine() method invoer door de gebruik van het console venster genomen.

```
string name = Console.Readline();
Console.WriteLine("Hello {0}, Good Luck in C#", name);
```

De code {0} wordt door de computer vervangen door de waarde die in de variabele name staat. Dit is ook te gebruiken als er meerdere variabelen hun waarde getoond moeten worden. Bekijk de volgende code:

```
Console.WriteLine("De leerling met de naam: {0} heeft het
                    stamboeknummer {1}.", name, stamboeknr);
```

Hier wordt {0} vervangen door de waarde van name en {1} door de waarde van stamboeknr.