Zie Scherp Scherper - 2e editie

Object georiënteerd programmeren met C#, van beginner naar gevorderde

Tim Dams

Zie Scherp Scherper - 2e editie

Object georiënteerd programmeren met C#, van beginner naar gevorderde

Tim Dams

ISBN 9789464651560

Inhoudsopgave

Welkom

Zo, je hebt besloten om C# te leren? Je bent hier aan het juiste adres. Dit boek wordt gebruikt als handboek binnen de opleidingen professionele bachelor elektronica-ict en toegepaste informatica van de AP Hogeschool. Het is gedurende vele jaren gegroeid tot een tweedelige reeks (1 per semester), genaamd “Zie Scherp” en “Zie Scherper”. Dit boek is de combinatie van die twee delen.

Eerst zullen we de fundering leggen en zaken behandelen zoals variabelen, loops methoden en arrays. Vervolgens zal (hopelijk) de mystieke, maar oh zo belangrijke, wereld van het *Object georiënteerd programmeren* uit de doeken gedaan worden.

Je vraagt je misschien af hoe up-to-date dit boek is? Wel, het is origineel samengesteld tijdens de lockdowns in 2020, dus… mmm, het jaar 2020 als kwaliteitslabel gebruiken is een beetje zoals zeggen dat je wijn maakt met rioolwater. Toen eind 2021 een nieuwe versie van Visual Studio verscheen werd het tijd om dit boek grondig te updaten. De versie die je nu in handen hebt werd geüpdatet in de zomer van 2022, een veel fijner jaar dan 2020 ;)

Net zoals spreektalen, evolueert ook de programmeertaal C# constant. Terwijl ik dit schrijf zijn we aan versie 10.0 van C# en staat versie 11 in de startblokken. Bij iedere nieuwe C#-versie worden bepaalde concepten plots veel eenvoudiger of zelfs gewoon overbodig. Een goed programmeur moet natuurlijk zowel met de oude als de nieuwe constructies kunnen werken. Ik heb getracht een gezonde mix tussen oud en nieuw te zoeken, waarbij de nadruk ligt op maximale bruikbaarheid in je verdere professionele carrière. Je zal hier dus geen stoere, state-of-the-art C# innovaties terugvinden die enkel in heel specifieke projecten bruikbaar zijn. Integendeel, ik hoop dat als je aan het laatste hoofdstuk bent, je een zodanige basis hebt, dat je ook zonder problemen in andere ‘zustertalen’ durft te duiken (zoals Java, C en C++, maar ook zelfs Python of JavaScript).

Dit boek ambieert niet om de volledige C#-taal en alles dat daar rond hangt aan te leren. Het boek daarentegen is gericht op eender wie die interesse heeft in de wondere wereld van programmeren, maar mogelijk nog nooit één letter code effectief heeft geprogrammeerd. Bepaalde concepten die ik te gecompliceerd acht voor een beginnende programmeur werden dan ook uit deze cursus gelaten. Beschouw wat je gaat lezen dus maar als een gatewaydrug naar meer C#, meer programmeertalen en vooral meer (programmeer)plezier! U weze gewaarschuwd.

Veel lees-en programmeerplezier,

Tim Dams
Zomer 2022

Over de bronnen

Dit boek is het resultaat van bijna een decennium C# doceren aan de AP Hogeschool (eerst nog Hogeschool Antwerpen, dan Artesis Hogeschool, dan Artesis Plantijn Hogeschool…). De eerste schrijfsels verschenen op een eigen gehoste blog ("Code van 1001 Nacht", die ondertussen ter ziele is gegaan) en vervolgens kreeg deze een iets strakkere, eenduidige vorm als gitbook cursus. Deze cursus, alsook een hele resem oefeningen en andere nuttige extra's kan je terugvinden op **ziescherp.be**. De inhoud van die cursus loopt integraal gelijk aan die van dit boek. Uiteraard is de kans bestaande dat er in de online versie ondertussen weer wat minder schrijffoutjes staan.

Waarom deze korte historiek? Wel, de kans is bestaande dat er hier en daar flarden tekst, code voorbeelden, of oefeningen niet origineel de mijne zijn. Ik heb getracht zo goed mogelijk aan te geven wat van waar komt, maar als ik toch iets vergeten ben, aarzel dan niet om me er op te wijzen.

Benodigdheden

Alle codevoorbeelden in deze cursus kan je zelf (na)maken met de gratis **Visual Studio 2022 Community** editie die je kan downloaden op **visualstudio.microsoft.com**.

Dankwoord

Aardig wat mensen - grotendeels mijn eerstejaars studenten van de professionele bachelor Elektronica-ICT en Toegepaste Informatica van de AP Hogeschool - hebben me met deze cursus geholpen. Hen allemaal afzonderlijk bedanken zou me een extra pagina kosten, en ik heb de meeste al nadrukkelijk bedankt in de vorige editie van dit boek.

Een speciale dank nogmaals aan Maarten Wachters die de originele pixel-art van me maakte waar ik vervolgens enkele varianten op heb gemaakt.

Ook een bos bloemen voor collega's Olga Coutrin en Walter Van Hoof om de ondankbare taak op zich te nemen mijn vele dt-fouten uit de vorige editie te halen op nog geen week voor de deadline. Bedankt!

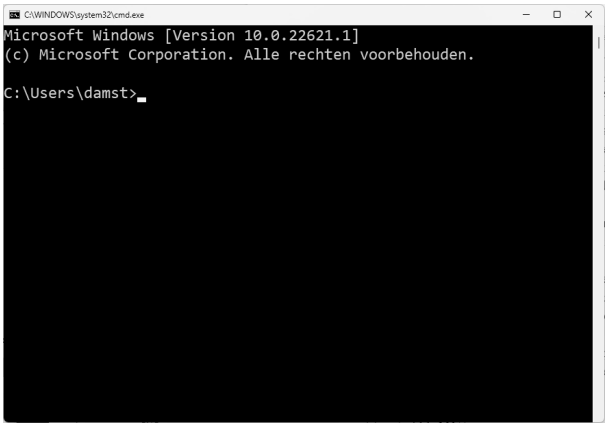
De trainers van Multimedi BV. die dit handboek ook gebruiken wil ik expliciet bedanken voor hun nuttige feedback op de eerste versie van dit boek, alsook om mij een extra reden te geven om dit boek in de eerste plaats uit te brengen.

1. De eerste stappen

Wel, wel, wie we hier hebben?! Iemand die de edele kunst van het programmeren wil leren? Dan ben je op de juiste plaats gekomen. Je gelooft het misschien niet, maar reeds aan het einde van dit hoofdstuk zal je je eerste eigen computer-applicaties kunnen maken. De weg naar eeuwige roem, glorie, véél vloeken en code herbruiken ligt voor je. Ben je er klaar voor?

De eerste stappen zijn nooit eenvoudig. We gaan proberen het aantal dure woorden, vreemde afkortingen en ingewikkelde schema's tot een minimum te houden. Maar toch, als je een nieuwe kunst wil leren zal je je handen (én toetsenbord) vuil moeten maken. Wat er ook gebeurt de komende hoofdstukken: blijf volhouden. Leren programmeren is een beetje als een berg leren beklimmen waarvan je nooit de top lijkt te kunnen bereiken. Wat ook zo is. Er is geen "top", en dat is net het mooie van dit alles. Er valt altijd iets nieuws te leren! De zaken waar je de komende pagina's op gaat vloeken zullen over enkele hoofdstukken al kinderspel lijken. Hou dus vol, blijf oefenen, vloek gerust af en toe en vooral: geniet van nieuwe dingen ontdekken!

Voor we verder gaan wil ik je wel even waarschuwen. Dit boek gaat uit van geen enkele kennis van programmeren, laat staan C#. Om die reden heb ik er dan ook voor gekozen om te beginnen bij het prille begin. Verwacht echter niet dat je aan het einde van dit boek vervolgens supercoole grafische applicaties of games zult kunnen maken. Het is zelfs zo dat we hoegenaamd geen woord gaan reppen over "windows applicaties", met knoppen en menu's enz. Alles dat in dit boek gemaakt wordt zal uitgevoerd "in de console", die oeroude DOS-schermen (ook wel een *shell* genoemd) die je nu nog vaak in films ziet wanneer hackers proberen in een erg beveiligd systeem in te breken. Waarom kies ik voor deze aanpak? Omdat de ervaring leert dat je hierdoor je kan focussen op de essentie van het probleem, en niet afgeleid wordt door existentiële vragen zoals "moet ik deze knop 3 pixel opschuiven?" of "ga ik voor een rode rand of een geel gearceerde?".



De "console". Qua zwarte inkt-verspilling zal deze afbeelding de hoofdprijs winnen!

1.1 Wat is programmeren?

Je hoort de termen geregeld: softwareontwikkelaar, programmeur, app-developer, enz. Allen zijn beroepen die in essentie kunnen herleid worden tot hetzelfde: programmeren. Programmeurs hebben geleerd hoe ze computers opdrachten kunnen geven (**programmeren**) zodat deze hopelijk doen wat je ze vraagt.

In de 21e eeuw is de term *computer* natuurlijk erg breed. Quasi ieder apparaat dat op elektriciteit werkt bevat tegenwoordig een computertje. Gaande van slimme lampen, tot de servers die het Internet draaiende houden of de smartwatch aan je pols. Zelfs aardig wat ijskasten en wasmachines beginnen (kleine) computers te bevatten.

Het probleem van computers, ongeacht hun grootte of kracht, is dat het in essentie ongelooflijk domme dingen zijn. Ze zullen altijd **exact** doen wat jij hen vertelt dat ze moeten doen. Als je hen dus de opdracht geeft om te ontploffen, schrik dan niet dat je even later naar de 112 kunt bellen.

Programmeren houdt in dat je leert praten met die domme computers zodat ze doen wat jij wilt dat ze doen.

Het algoritme

> First, solve the problem. Then, write the code.

Deze quote van John Johnson wordt door veel beginnende programmeurs soms met een scheef hoofd aanhoort. “Ik wil gewoon code schrijven!” Het is een mythe dat programmeurs constant code schrijven. Integendeel, een goed programmeur zal veel meer tijd in de “voorbereiding” tot code schrijven steken: het maken van een goed **algoritme** na een grondige **analyse van het probleem** .

Het algoritme is de essentie van een computerprogramma en kan je beschouwen als het recept dat je aan de computer gaat geven zodat deze jouw probleem op de juiste manier zal oplossen. **Het algoritme bestaat uit een reeks instructies** die de computer moet uitvoeren telkens jouw programma wordt uitgevoerd.

Het algoritme van een programma moet je zelf verzinnen. De volgorde waarin de instructies worden uitgevoerd zijn echter zeer belangrijk. Dit is exact hetzelfde als in het echte leven: een algoritme om je fiets op te pompen kan zijn:

```
1 Haal dop van het ventiel.
2 Plaats pomp op ventiel.
3 Begin te pompen.
```

Eender welke andere volgorde van bovenstaande algoritme zal vreemde (en soms fatale) fouten geven.

Wil je dus leren programmeren, dan zal je logisch moeten leren denken en een analytische geest hebben. Als je eerst tegen een bal trapt voor je kijkt waar de goal staat dan zal de edele kunst van het programmeren voor jou een…speciale aangelegenheid worden.

Vanaf nu ben je trouwens gemachtigd om naar de nieuwsdiensten te mailen telkens ze foutief het woord “logaritme” gebruiken in plaats van “algoritme”. Het woord logaritme is iets wat bij sommige nachtmerries uit de lessen wiskunde opwekt en heeft hoegenaamd niets met programmeren te maken. Uiteraard kan het wel zijn dat je ooit een algoritme moet schrijven om een logaritme te berekenen. Hopelijk moet een journalist nooit voorgaande zin in een nieuwsbericht gebruiken.

Programmeertaal

Om een algoritme te schrijven dat onze computer begrijpt dienen we een programmeertaal te gebruiken. Computers hebben hun eigen taaltje dat programmeurs moeten kennen voor ze hun algoritme aan de computer kunnen *voeden*. Er zijn tal van computertalen, de ene al wat obscuurder dan de andere. Maar wat al deze talen gelijk hebben is dat ze meestal:

- **ondubbelzinnig** zijn: iedere opdracht of woord kan door de computer maar op exact één manier geïnterpreteerd worden. Dit in tegenstelling tot bijvoorbeeld het Nederlands waar "wat een koele kikker" zowel een letterlijke, als een figuurlijke betekenis heeft die niets met elkaar te maken heeft.
- bestaan uit **woordenschat**: net zoals het Nederlands heeft ook iedere programmeertaal een , meestal beperkte, lijst woorden die je kan gebruiken. Je gaat ook niet in het Nederlands zelf woorden verzinnen in de hoop dat je partner je kan begrijpen.
- bestaan uit **grammaticaregels**: Enkel Yoda mag Engels in een verkeerde volgorde gebruiken. Iedereen anders houdt zich best aan de grammatica-afspraken die een taal heeft. "bal rood is" lijkt nog begrijpbaar, maar als we zeggen "bal rood jongen is gooit veel"?

De C# taal

Net zoals er ontelbare spreektalen in de wereld zijn, zijn er ook vele programmeertalen. **C#** (spreek uit *'siesjarp'*) is er één van de vele. C# is een taal die deel uitmaakt van de .NET (spreek uit *'dotnet'*) omgeving die meer dan 20 jaar geleden door Microsoft werd ontwikkeld. Het fijne van C# is dat deze een zogenaamde **hogere programmeertaal** is. Hoe "hoger" de programmeertaal, hoe leesbaarder deze wordt voor leken omdat hogere programmeertalen dichter bij onze eigen taal aanleunen.

De geschiedenis van de hele .NET-wereld vertellen zou een boek op zich betekenen en gaan we hier niet doen. Het is nuttig om weten dat er een gigantische bron aan informatie over .NET en C# online te vinden is, beginnende met **docs.microsoft.com/en-us/dotnet/csharp/getting-started**.

Het fijne van leren programmeren is dat je binnenkort op een bepaald punt gaat komen waarbij de keuze van programmeertaal er minder toe doet. Vergelijk het met het leren van het Frans. Van zodra je Frans onder knie hebt is het veel eenvoudiger om vervolgens Italiaans of Spaans te leren. Zo ook met programmeertalen. De C# taal bijvoorbeeld lijkt bijvoorbeeld als twee druppels water op Java, alsook op de talen waar ze van afstamt, C en C++.

Zelfs JavaScript, Python en veel andere moderne talen zullen weinig geheimen voor jou hebben wanneer je aan het einde van dit boek bent.

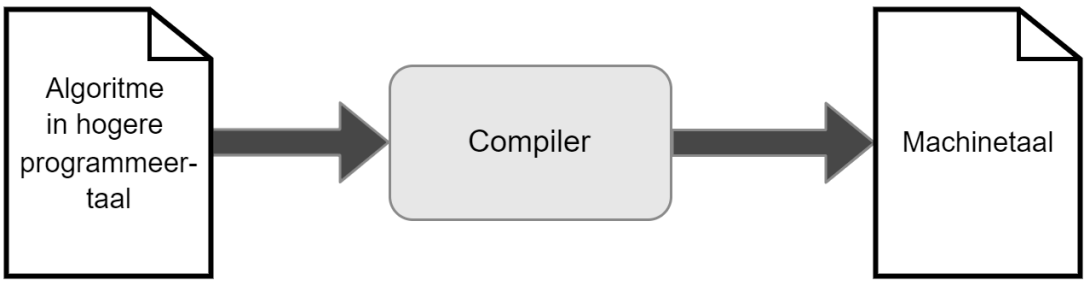
Anders Hejlsberg

Deze Deen krijgt een eigen sectie in dit boek. Waarom? Hij is niemand minder dan de "uitvinder" van C#. Anders Hejlsberg heeft een stevig palmares inzake programmeertalen verzinnen. Voor hij C# boven het doopvont hield bij Microsoft, schreef hij ook al Turbo Pascal én was hij de *chief architect* van Delphi. Je zou denken dat hij na 3 programmeertalen wel op z'n lauweren zou rusten, maar zo werkt Anders niet. In 2012 begon hij te werken aan een JavaScript alternatief, wat uiteindelijk het immens populaire TypeScript werd. Dit allemaal om maar te zeggen dat als je één poster in je slaapkamer moet ophangen, het die van Anders zou moeten zijn.

De compiler

Rechtstreeks onze algoritmen tegen de computer vertellen vereist dat we machinetaal kunnen. Deze is echter zo complex dat we tientallen lijnen machinetaal nodig hebben om nog maar gewoon 1 letter op het scherm te krijgen. Er werden daarom dus hogere programmeertalen ontwikkeld die aangenamer zijn dan deze zogenaamde machinetalen om met computers te praten.

Uiteraard hebben we een vertaler nodig die onze code zal vertalen naar de machinetaal van het apparaat waarop ons programma moet draaien. Deze vertaler is de **compiler** die aardig wat complex werk op zich neemt, maar dus in essentie onze code gebruiksklaar maakt voor de computer.



Vereenvoudigd compiler overzicht.

Merk op dat we hier veel details van de compiler achterwege laten. De compiler is een uitermate complex element, maar in deze fase van je (prille) programmeursleven hoeven we enkel de kern van de compiler te begrijpen: **het omzetten van C# code naar een uitvoerbaar bestand geschreven in machinetaal**.

Microsoft .NET

Bij de geboorte van .NET in 2000 zat ook de taal C#.

.NET is een zogenaamd **framework**. Dit framework bestaat uit een grote groep van bibliotheken (*class libraries*) en een *virtual execution system* genaamd de **Common Language Runtime (CLR)**. De CLR zal ervoor zorgen dat C#, of andere .NET talen (F#, VB.NET, enz.), kunnen samenwerken met de vele bibliotheken.

Om een uitvoerbaar bestand te maken (**executable**, vandaar de extensie .exe bij uitvoerbare programma's in Windows) zal de broncode die je hebt geschreven in C# worden omgezet naar **Intermediate Language** (IL) code. Op zich is deze IL code nog niet uitvoerbaar, maar dat is niet ons probleem. Wanneer een gebruiker een in IL geschreven bestand wil uitvoeren dan zal, achter de schermen, de CLR deze code ogenblikkelijk naar machine code omzetten (**Just-In-Time** of JIT compilatie) en uitvoeren. De gebruiker zal dus nooit dit proces opmerken (tenzij er geen .NET framework werd geïnstalleerd op het systeem).

Over nummeringen en naamgevingen van .NET en C#

Microsoft heeft er een handje van weg om hun producten ingewikkelde volgnummers-of letters te geven, denk maar aan Windows 10 die de opvolger was van Windows 8 (dat had trouwens een erg goede reden; zoek maar eens op), of Windows 7 dat Windows Vista opvolgde. Het helpt ook niet dat ze geregeld hun producten een nieuwe naam geven. Zo was het binnen .NET tot voor kort erg ingewikkeld om te weten welke versie nu eigenlijk de welke was. Microsoft heeft gelukkig recent de naamgevingen herschikt én hernoemt in de hoop het allemaal wat duidelijker te maken. Laten we daarom even kort te bespreken waar we nu zitten.

.NET 6 (framework)

Telkens er een nieuwe .NET framework werd *gereleased* verscheen er ook een bijhorende nieuwe versie van Visual Studio. Vroeger had je verschillende frameworks binnen de .NET familie zoals *.NET Framework*, *".NET Standard"*, *.NET Core* enz. die allemaal net niet dezelfde doeleinden hadden wat het erg verwarrend maakte. Om dit te vereenvoudigen bestaat sinds 2020 enkel nog .NET gevolgd door een nummer.

Zo had je in 2020 .NET 5 en verschijnt eind 2022 .NET 7. Dit boek maakt gebruikt van **.NET 6** dat verscheen samen met Visual Studio 2022…in november 2021. Je moet er maar aan uit kunnen.

C# 10

De C# taal is eigenlijk nog het eenvoudigst qua nummering. Om de zoveel tijd krijgt C# een update met een nieuwe reeks taal-eigenschappen die je kan, maar niet hoeft te gebruiken. Momenteel zitten we aan **C# 10** dat werd uitgebracht samen met .NET 6.

Eind 2022 komt .NET 7 uit en dus ook alweer een nieuwe versie van C#, namelijk versie 11. De kans is dus groot dat voorgaande zin alweer gedateerd is tegen dat je hem leest. De vernieuwingen in C# zijn niet altijd belangrijk voor beginnende programmeurs. In dit boek hebben we getracht de belangrijkste én meest begrijpbare nieuwe features uit de taal te gebruiken waar relevant, maar over het algemeen gezien mag je stellen dat dit boek tot en met versie 7.3 de belangrijkste zaken zal behandelen.

Volgende youtube-video van *Johnny does DOTNET* geeft een erg goed historisch overzicht van de naamsveranderingen: **youtube.com/watch?v=O4Qcg5Uon4g**.

Je vraagt je misschien af waarom dit allemaal verteld wordt? Waarom wordt deze geschiedenisles gegeven? De reden is heel eenvoudig. Je gaat zeker geregeld zaken op het internet willen opzoeken tijdens het (leren) programmeren en zal dan ook vaker op artikels stuiten met de oude(re) naamgeving en dan mogelijks niet kunnen volgen.

Recent heeft Microsoft besloten om veel sneller nieuwe updates voor .NET en dus ook C# en Visual Studio uit te brengen, en dat allemaal opensource (je kan zelfs meehelpen via **github.com/dotnet**). Dit heeft als voordeel dat bugs sneller opgelost worden én dat we sneller toegang krijgen tot de nieuwste features. Het nadeel is echter dat informatie zoals in dit boek van de één op de andere dag out-dated kan zijn.

1.2 Kennismaken met C# en Visual Studio

We gaan in dit boek leren programmeren met Microsoft Visual Studio 2022, een softwarepakket waar ook een gratis community versie voor bestaat. Microsoft Visual Studio (vanaf nu **VS**) is een pakket dat een groot deel van de tools samenvoegt die een programmeur nodig heeft (debugger, code editor, compiler, etc).

VS is een zogenaamde **IDE ("Integrated Development Environment")** en is op maat gemaakt om in C# geschreven applicaties te ontwikkelen. Je bent echter verre van verplicht om enkel C# applicaties in VS te ontwikkelen, je kan gerust VB.NET, TypeScript, Python en andere talen gebruiken. Ook vice versa ben je niet verplicht om VS te gebruiken om te ontwikkelen. Je kan zelfs in notepad code schrijven en vervolgens compileren (zie hierna). Er bestaan zelfs online C# programmeer omgevingen, zoals **dotnetfiddle.net**.

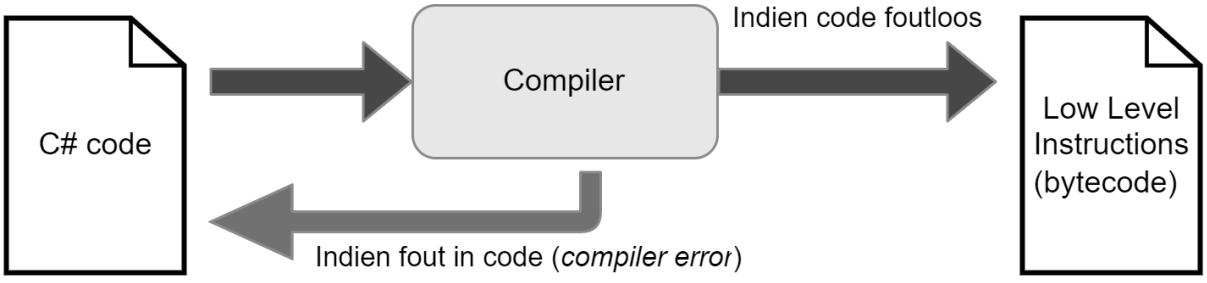
In dit boek zullen we steeds werken met Visual Studio. Niet met *Visual Studio Code*. Visual Studio code is een zogenaamde lightweight versie van VS die echter zeker ook z'n voordelen heeft (makkelijk uitbreidbaar, snel, compact, etc). Visual Studio vindt dankzij VS Code eindelijk ook z'n weg op andere platformen dan enkel die van Microsoft. Je kan de laatste versie ervan downloaden op: **code.visualstudio.com**.

De compiler en Visual Studio

Zoals gezegd: jouw taak als programmeur is algoritmes in C# taal uitschrijven. We zouden dit in een eenvoudige tekstverwerker kunnen doen, maar dan maken we het onszelf lastig. Net zoals je tekst in notepad kunt schrijven, is het handiger dit bijvoorbeeld in tekstverwerker zoals Word te doen: je krijgt een spellingchecker en allerlei handige extra's.

Ook voor het schrijven van computer code is het handiger om een IDE te gebruiken, een omgeving die ons zal helpen foutloze C# code te schrijven.

Het hart van Visual Studio bestaat uit de compiler die we hiervoor besproken hebben. De compiler zal je C# code omzetten naar de IL-code zodat jij (of anderen) je applicatie op een computer (of ander apparaat) kunnen gebruiken. Zolang je C# code niet exact voldoet aan de C# syntax en grammatica zal de compiler het vertikken een uitvoerbaar bestand voor je te genereren.

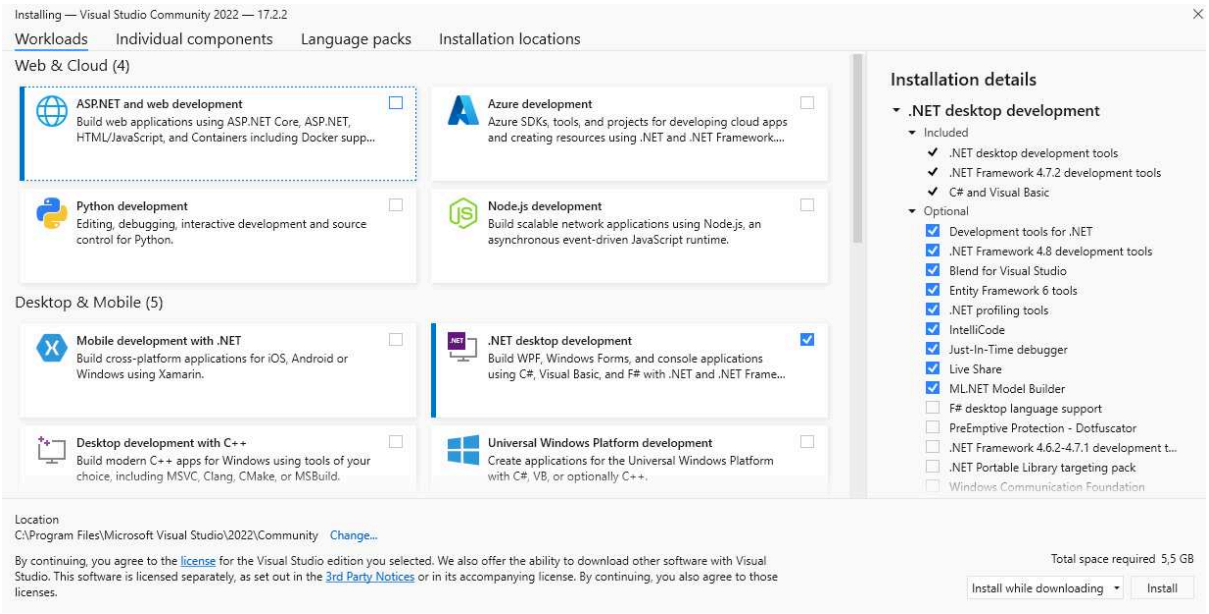


Vereenvoudigd compiler overzicht.

Visual Studio Installeren

In dit boek zullen de voorbeelden steeds met de **Community** editie van VS gemaakt zijn. Je kan deze gratis downloaden en installeren via **visualstudio.microsoft.com/vs**.

Het is belangrijk bij de installatie dat je zeker de **.NET desktop development** workload kiest. Uiteraard ben je vrij om meerdere zaken te installeren.



In dit boek zullen we enkel met de .NET desktop development workload werken.

Visual studio opstarten

Als alles goed is geïnstalleerd kan je Visual Studio starten via het start-menu van Windows.

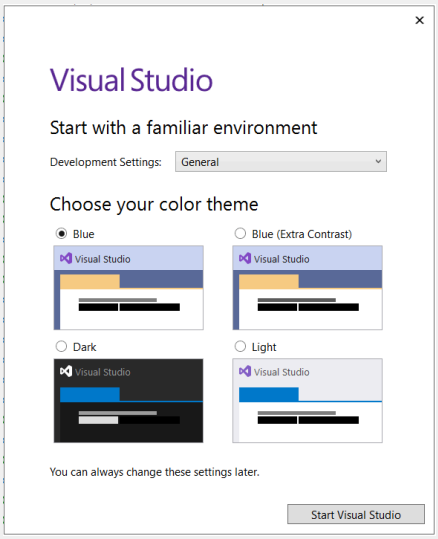
Allereerste keer opstarten

De allereerste keer dat je VS opstart krijg je 2 extra schermen te zien:

- Het “sign in” scherm mag je overslaan (kies “Not now, maybe later”).
- Op het volgende scherm kies je best als “Development settings” voor **Visual C#**. Vervolgens kan je je kleurenthema kiezen. Dit heeft geen invloed op de manier van werken.

Dark is uiteraard het coolste thema om in te coderen. Je voelt je ogenblikkelijk Neo uit The Matrix. Het nadeel van dit thema is dat het veel meer inkt verbruikt indien je screenshots in een boek zoals dit wilt plaatsen. De keuze voor Development Setting kan je naar "Visual C#" veranderen, maar General is even goed (je zal geen verschil merken in eerste instantie).

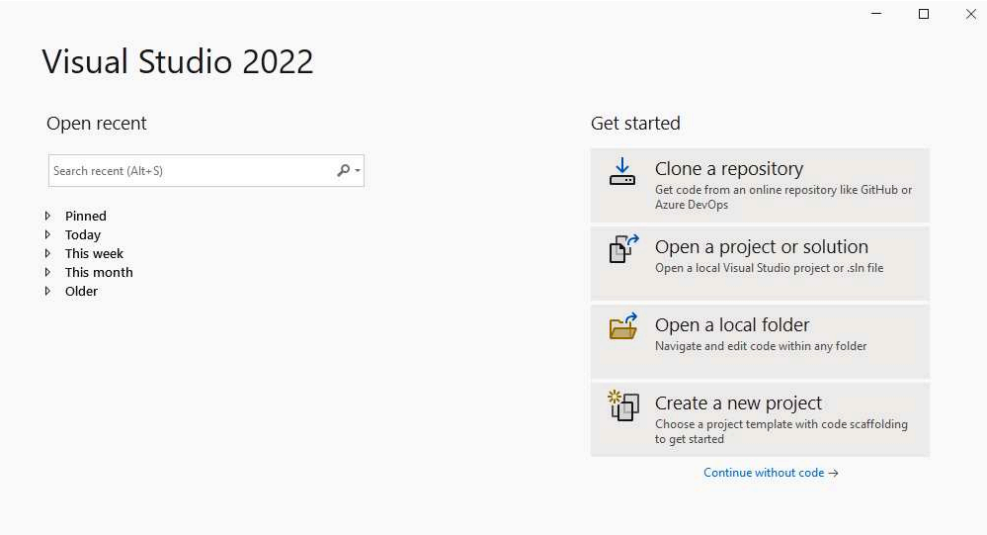
Je kan dit achteraf nog aanpassen in VS via "Tools" in de menubalk, dan "Import and Export Settings" en kiezen voor "Import and Export Settings Wizard".



Je kan dit nadien ook altijd nog aanpassen. En zelfs personaliseren tot de vreemdste kleur- en lettertypecombinaties.

Project keuze

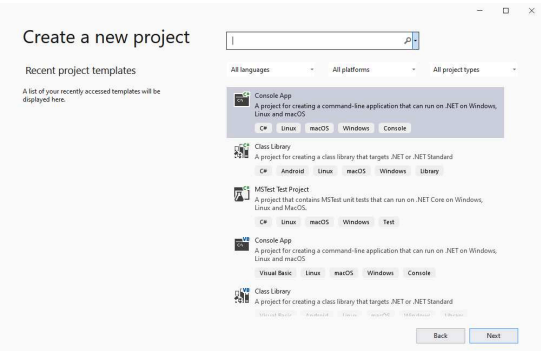
Na het opstarten van VS krijg je het startvenster te zien van waaruit je verschillende dingen kan doen. Van zodra je projecten gaat aanmaken zullen deze in de toekomst ook op dit scherm getoond worden zodat je snel naar een voorgaand project kunt gaan.



Het startscherm van Visual Studio.

Een nieuw project aanmaken

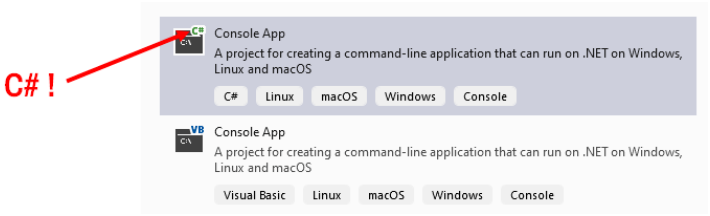
We zullen nu een nieuw project aanmaken, kies hiervoor “Create a new project”.



Kies je projecttype.

Het “New Project” venster dat nu verschijnt geeft je hopelijk al een glimp van de veelzijdigheid van VS. In het rechterdeel zie je bijvoorbeeld alle Project Types staan. M.a.w. dit zijn alle soorten programma’s die je kan maken in VS. Naargelang de geïnstalleerde opties en bibliotheken zal deze lijst groter of kleiner zijn.

In dit boek zullen we altijd het Project Type **Console App** gebruiken (ZONDER .NET Framework achteraan). Je vindt deze normaal bovenaan de lijst terug, maar kunt deze ook via het zoekveld bovenaan terugvinden door te zoeken op, je raadt het nooit: console. **Let er op dat je een klein groen C# icoontje ziet staan bij het zwarte icoon van de Console app**. Ook andere talen ondersteunen console applicaties, maar wij gaan natuurlijk met C# aan het werk.



Kies voor C#, niet Visual Basic (VB). Dank bij voorbaat!

Een console applicatie is een programma dat alle uitvoer naar een zogenaamde *console* stuurt, een shell. Je kan met andere woorden enkel tekst (UNICODE) als uitvoer genereren en dus geen multimedia elementen zoals afbeeldingen, geluid, enz.

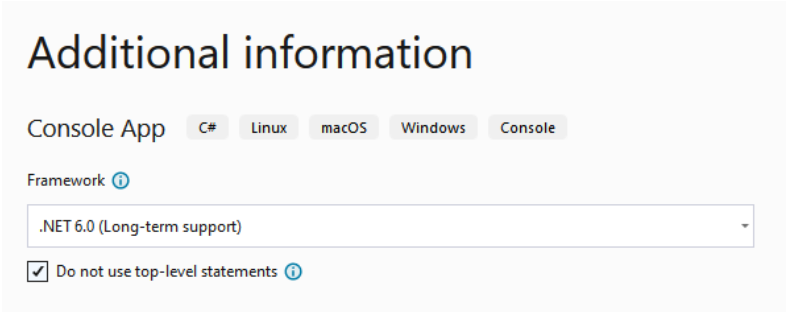
Kies dit type en klik 'Next'.

Op het volgende scherm kan je een naam ingeven voor je project alsook de locatie op de harde schijf waar het project dient opgeslagen te worden. **Onthoud waar je je project aanmaakt zodat je dit later terugvindt**.

Het "Solution name" tekstveld blijf je af. Hier zal automatisch dezelfde tekst komen als die dat je in het "Project name" tekstveld invult.

Geef je projectnamen ogenblikkelijk duidelijke namen zodat je niet opgezadeld geraakt met projecten zoals Project201, enz. waarvan je niet meer weet welke belangrijk zijn en welke niet.

Geef je project de naam "MyFirstProject" en kies een goede locatie (ik raad je aan dit steeds in Dropbox of Onedrive te doen). **We raden aan om de checkbox ("Place solution and project in the same directory") onderaan niét aan te vinken**. In de toekomst zal het nuttig zijn dat je meer dan 1 project per solution zal kunnen hebben. Lig er nog niet van wakker. Klik op next en kies als Target Framework de meest recente versie. **Duidt hier zeker de checkbox aan met "Do not use-top level statements"!!!** Klik nu op Create.



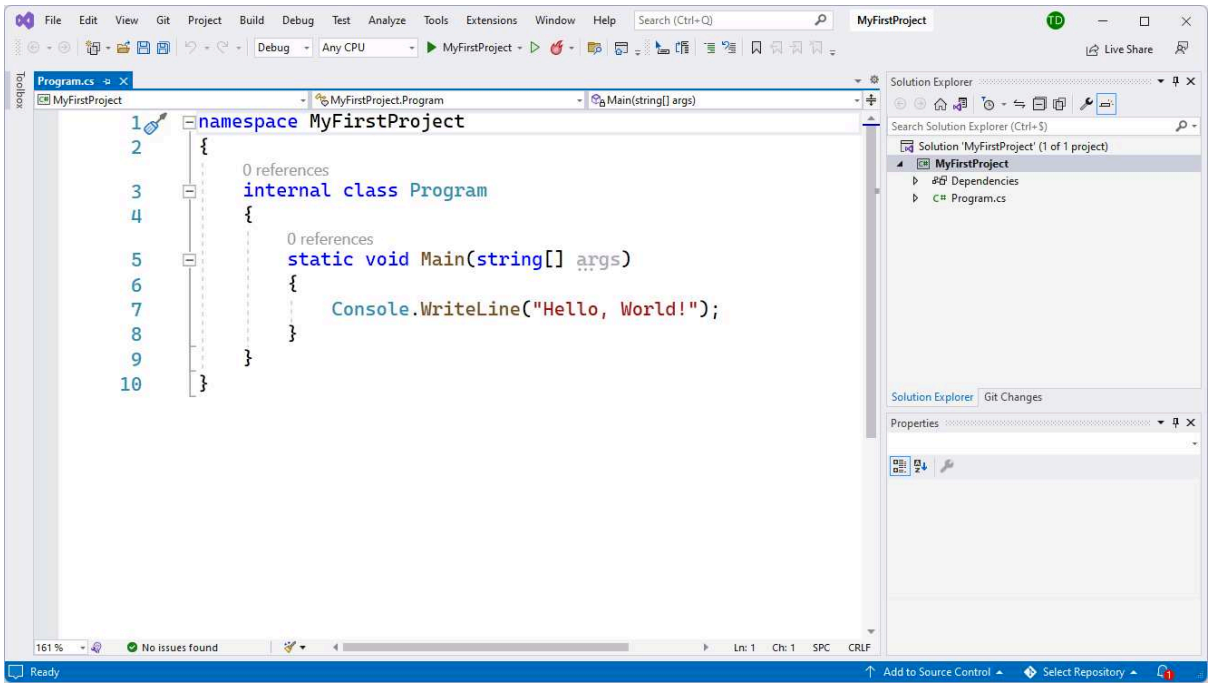
De auteur van dit boek kan fier melden dat die checkbox er staat mede dankzij zijn gezaag op github.com/dotnet/docs/issues/2742.

VS heeft nu reeds een aantal bestanden aangemaakt die je nodig hebt om een 'Console Applicatie' te maken.

IDE Layout

Wanneer je VS opstart zal je mogelijk overweldigd worden door de hoeveelheid menu's, knopjes, schermen, enz. Dit is normaal voor een IDE: deze wil zoveel mogelijk mogelijkheden aanbieden aan de gebruiker. Vergelijk dit met Word: afhankelijk van wat je gaat doen gebruikt iedere gebruiker andere zaken van Word. De makers van Word kunnen dus niet bepaalde zaken weglaten, ze moeten net zoveel mogelijk aanbieden.

We zullen nu eerst eens bekijken wat we allemaal zien in VS na het aanmaken van een nieuw programma.



VS IDE overzicht.

- Je kan meerdere bestanden tegelijkertijd openen in VS. Ieder bestand zal z'n eigen **tab** krijgen. De actieve tab is het bestand wiens inhoud je in het hoofdgedeelte eronder te zien krijgt. Merk op dat enkel open bestanden een tab krijgen. Je kan deze tabbladen ook "lostrekken" om bijvoorbeeld enkel dat tabblad op een ander scherm te plaatsen.
- De "**solution explorer**" aan de rechterzijde toont alle bestanden en elementen die tot het huidige project behoren. Als we dus later nieuwe bestanden toevoegen, dan kan je die hier zien (en openen). Verwijder hier géén bestanden zonder dat je zeker weet wat je aan het doen bent!
- Het **properties** venster (eigenschappen) rechts onderaan is een belangrijk venster. Hier komen alle eigenschappen van het huidige geselecteerde element. Selecteer bijvoorbeeld maar eens Program.cs in de Solution Explorer en merk op dat er allerlei eigenschappen getoond worden. Onderaan het Properties venster wordt steeds meer informatie getoond over de huidig geselecteerde eigenschap.

Indien je een nieuw project hebt aangemaakt en de code die je te zien krijgt lijkt in de verste verte niet op de code die je hierboven ziet dan heb je vermoedelijk een verkeerd projecttype aangemaakt (of je hebt de "Do not use top-level statements" checkbox niet aangeduidt). De meest gemaakte fout in deze fase is dat je een Visual Basic (VB) Console applicatie hebt gekozen en niet een C# versie.

Layout kapot/kwijt/vreemd?

De layout van VS kan je volledig naar je hand zetten. Je kan ieder (deel-)venster en tab verzetten, verankeren en zelfs verplaatsen naar een ander bureaublad. Experimenteer hier gerust mee en besef dat je steeds alles kan herstellen. Het gebeurt namelijk al eens dat je layout een beetje om zeep is:

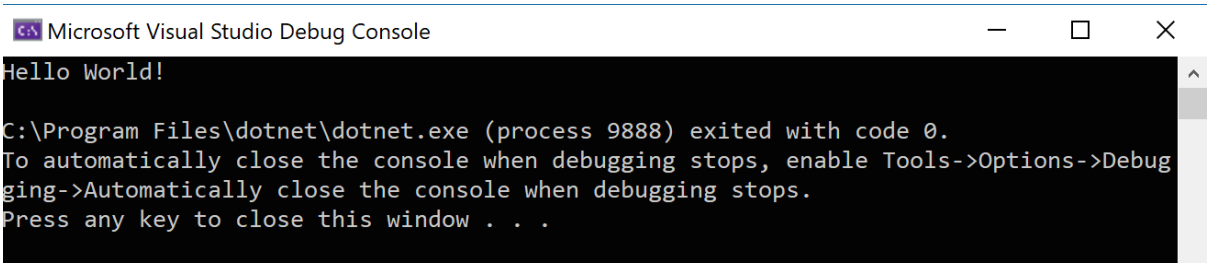
- Om eenvoudig een venster terug te krijgen, bijvoorbeeld het properties window of de solution explorer: klik bovenaan in de menubalk op "View" en kies dan het gewenste venster (soms staat dit in een submenu).
- Je kan ook altijd je layout in z'n geheel **resetten**: ga naar "Window" en kies "Reset window layout".

Je programma starten

De code in Program.cs die VS voor je heeft gemaakt is reeds een werkend, maar weinig nuttig, programma. Je kan de code compileren en uitvoeren door op de groene driehoek bovenaan te klikken:

Het programma uitvoeren.

Als alles goed gaat krijg je nu "Hello World!" te zien en wat extra informatie omtrent het programma dat net werd uitgevoerd:



Uitvoer van het programma.

Veel doet je programma nog niet natuurlijk, dus sluit dit venster maar terug af door een willekeurige toets in te drukken.

Is dit alles?

Nee hoor. Visual Studio is lekker groot, maar laat je dat niet afschrikken. Net zoals voor het eerst op een nieuwe reisbbestemming komen, kan deze in het begin overweldigend zijn, tot je weet waar het zwembad en de pingpongtafel staat en je van daaruit je weg stilletjes aan leert kennen.

1.3 Console-applicaties

Een console-applicatie is een programma dat zijn in- en uitvoer via een klassiek commando/shell-scherm toont. Een console-applicatie draait in dezelfde omgeving als wanneer we in Windows een command-prompt openen (via Start-> Uitvoeren-> `cmd` [enter]). We zullen in dit boek enkel console-applicaties leren maken. Grafische frontends zoals WPF komen in dit boek niet aan bod.

In en uit - ReadLine en WriteLine

Een programma zonder invoer van de gebruiker is niet erg boeiend. De meeste programma's die we leren schrijven vereisen dan ook "input" (**IN**). We moeten echter ook zaken aan de gebruiker kunnen tonen (bijvoorbeeld de uitkomst van een berekening, een foutboodschap, enz.) wat dus vereist dat er "output" (**UIT**) naar het scherm kan gestuurd worden.

In het begin zullen al je applicaties deze opbouw hebben.

Console-applicaties maken in C# vereist dat je minstens twee belangrijke C# methoden leert gebruiken:

- Met behulp van **`Console.ReadLine()`** kunnen we input van de gebruiker inlezen en in ons programma verwerken.
- Via **`Console.WriteLine()`** kunnen we tekst op het scherm tonen.

Je eerste console programma

Sluit het eerder gemaakte "MyFirstProject" project af en herstart Visual Studio. Maak nu een nieuw console-project aan (noem het Demo1) en open het Program.cs bestand (indien het nog niet open is). **Veeg de code die hier reeds staat niet weg!**

Voeg onder de lijn `Console.WriteLine("Hello World!");` volgende code toe (vergeet de puntkomma niet):

```
Console.WriteLine("Hoi, ik ben het!");
```

Zodat je dus volgende code krijgt:

```
1  namespace Demo1
2  {
3      internal class Program
4      {
5          static void Main(string[] args)
6          {
7              Console.WriteLine("Hello World!");
8              Console.WriteLine("Hoi, ik ben het");
9          }
10     }
11 }
```

Compileer deze code en voer ze uit: **druk hiervoor weer op het groene driehoekje bovenaan**. Of via het menu Debug en dan Start Debugging.

Moet ik niets bewaren?

Neen. Telkens je op de groene "build en run" knop duwt worden al je aanpassingen automatisch bewaard. Trouwens: **Kies nooit voor "save as…"!** want dan bestaat de kans dat je project niet meer compileert. Dit zal aardig wat problemen in je project voorkomen, geloof me maar.

Analyse van de code

We gaan nu iedere lijn code kort bespreken. Sommige lijnen code zullen lange tijd niet belangrijk zijn. Onthoud nu alvast dat: **alle belangrijke code staat tussen de accolades onder de lijn `static void Main(string[] args)`**!

Laat je niet afschrikken door wat er nu volgt. We gooien je even in het diepe gedeelte van het zwembad maar zullen je er op tijd uithalen zodat we vervolgens terug in het babybadje rustig op de glijbaan kunnen gaan spelen en C# op een trager tempo kunnen ontdekken.

- **`namespace Demo1`**: Dit is de unieke naam waarbinnen we ons programma zullen plaatsen, en het is niet toevallig de naam van je project. Verander dit nooit tenzij je weet wat je aan het doen bent. We bespreken namespaces in hoofdstuk 10.
- **`internal class Program`**: Hier start je echte programma. Alle code binnen deze Program accolades zullen gecompileerd worden naar een uitvoerbaar bestand. Vanaf hoofdstuk 9 zal deze lijn geen geheimen meer hebben voor je.
- **`static void Main(string[] args)`**: Het startpunt van iedere console-applicatie. Wat hier gemaakt wordt is een **methode** genaamd `Main`. Je programma kan meerdere methoden (of functies) bevatten, maar enkel degene genaamd `Main` zal door de compiler als het startpunt van het programma gemaakt worden. Deze lijn gaan we in hoofdstuk 7 en 8 uit de doeken doen.
- **`Console.WriteLine("Hello World!");`**: Dit is een **statement** dat de `WriteLine`-methode aanroept van de `Console`-bibliotheek. Het zal alle tekst die tussen de aanhalingstekens staat op het scherm tonen.
- **`Console.WriteLine("Hoi ik ben het");`**: en ook deze lijn zorgt ervoor dat er tekst op het scherm komt wanneer het programma zal uitgevoerd worden.
- **Accolades**: vervolgens moet voor iedere openende accolade eerder in de code nu ook een bijhorende sluitende volgen. We gebruiken accolades om de *scope* aan te duiden, iets dat we in hoofdstuk 5 geregeld zullen nodig hebben.

Net zoals een recept, zal ook in C# code van boven naar onder worden uitgevoerd. In het geval van ons eerste programma zal het programma (voor ons) starten bij de lijn `Console.WriteLine("Hello World!");` en dan verder blijven werken tot het aan het einde van de `Main` komt, het gebied dat wordt afgebakend door de accolade van lijn 9. Kortom, van zodra lijn 9 wordt bereikt is dat het signaal voor de computer om je applicatie af te sluiten.

Jawadde…Wat was dit allemaal?! We hebben al aardig wat vreemde code zien passeren en het is niet meer dan normaal dat je nu denkt “dit ga ik nooit kunnen”. Wees echter niet bevreesd: je zal sneller dan je denkt bovenstaande code als ‘kinderspel’ gaan bekijken. Een tip nodig? Test en experimenteer met wat je al kunt!

Laat deze info rustig inzinken en onthoud alvast volgende belangrijke zaken:

- Al je eigen code komt momenteel enkel tussen de `Main` accolades.
- Eindig iedere lijn code daar met een puntkomma (;).

De oerman verschijnt wanneer we een stevige stap gezet hebben en je mogelijk even onder de indruk bent van al die nieuwe informatie. Hij zal proberen informatie nog eens vanuit een ander standpunt toe te lichten en te herhalen waarom deze nieuwe kennis zo belangrijk is.

“Hello world” op het scherm laten verschijnen wanneer je een nieuwe programmeertaal leert is ondertussen een traditie bij programmeurs. Er is zelfs een website die dit verzamelt namelijk **helloworldcollection.de**. Deze site toont in honderden programmeertalen hoe je “Hello world” moet programmeren.

WriteLine: Tekst op het scherm

De `WriteLine`-methode is een veelgebruikte methode in Console-applicaties. Het zorgt ervoor dat we tekst op het scherm kunnen tonen.

Voeg volgende lijn toe na de vorige WriteLine-lijn in je project:

```
1 Console.WriteLine("Wie ben jij?!");
```

De WriteLine methode zal alle tekst tonen die tussen de aanhalingstekens (`" "`) staat tussen de haakjes van de methode. **De aanhalingstekens aan het begin en einde van de tekst zijn uiterst belangrijk! Alsook het puntkomma helemaal achteraan**.

Je code binnen de `Main` accolades zou nu moeten zijn:

```
1 Console.WriteLine("Hello World!");
2 Console.WriteLine("Hoi, ik ben het");
3 Console.WriteLine("Wie ben jij?!");
```

Kan je voorspellen wat de uitvoer zal zijn? Test het eens!

We tonen niet telkens de volledige broncode. Als we dat blijven doen dan wordt dit boek dubbel zo dik. We tonen daarom meestal enkel de code die binnen de `Main` (of later ook elders) moet komen.

ReadLine: Input van de gebruiker verwerken

Met de Console kan je met een handvol methoden reeds een aantal interessante dingen doen.

Zo kan je bijvoorbeeld input van de gebruiker inlezen en bewaren in een variabele als volgt:

```
1 string result;
2 result = Console.ReadLine();
```

Wat gebeurt er hier juist?

De eerste lijn code:

- Concreet zeggen we hiermee aan de compiler: maak in het geheugen een plekje vrij waar enkel data van het type string in mag bewaard worden (wat deze zin exact betekent komt later. Onthoud nu dat geheugen van het type `string` enkel “tekst” kan bevatten).
- Noem deze geheugenplek `result` zodat we deze later makkelijk kunnen in en uitlezen.

Tweede lijn code:

- Vervolgens roepen we de `ReadLine` methode aan. Deze methode zal de invoer van de gebruiker van het toetsenbord uitlezen tot de gebruiker op enter drukt.
- Het resultaat van de ingevoerde tekst wordt bewaard in de variabele `result`.

Merk op dat de toekenning in C# van rechts naar links gebeurt.Vandaar dat `result` dus links van de toekenning (=) staat en de waarde krijgt van het gedeelte rechts ervan.

Je programma zou nu moeten zijn:

```
1  Console.WriteLine("Hello World!");
2  Console.WriteLine("Hoi, ik ben het!");
3  Console.WriteLine("Wie ben jij?!");
4  string result;
5  result = Console.ReadLine();
```

Start nogmaals je programma. Je zal merken dat je programma nu een cursor toont en wacht op invoer nadat het de eerste 3 lijnen tekst op het scherm heeft gezet. Je kan nu eender wat intypen en van zodra je op enter duwt gaat het programma verder. Maar aangezien lijn 5 de laatste lijn van ons algoritme is, zal je programma hierna afsluiten (en we hebben dus de gebruiker voor niets iets laten invoeren).

Input gebruiker gebruiken

Een variabele is een geheugenplekje (met een naam) waar we zaken in kunnen bewaren. In het volgende hoofdstuk gaan we zo vaak het woord variabele vertellen dat je oren en ogen er van gaan bloeden, dus trek je nu nog niet te veel aan van dit woord. We kunnen nu invoer van de gebruiker, die we hebben bewaard in de variabele `result`, gebruiken en tonen op het scherm.

```
1  Console.WriteLine("Dag");
2  Console.WriteLine(result);
3  Console.WriteLine("hoe gaat het met je?");
```

In de tweede lijn hier gebruiken we de variabele `result` (waar de invoer van de gebruiker in bewaard wordt) als parameter in de `WriteLine`-methode.

Met andere woorden: de `WriteLine` methode zal op het scherm tonen wat de gebruiker even daarvoor heeft ingevoerd.

Je volledige programma ziet er dus nu zo uit:

```
1 Console.WriteLine("Hello World!");
2 Console.WriteLine("Hoi, ik ben het!");
3 Console.WriteLine("Wie ben jij?!");
4 string result;
5 result = Console.ReadLine();
6 Console.WriteLine("Dag ");
7 Console.WriteLine(result);
8 Console.WriteLine("hoe gaat het met je?");
```

Test het programma en voer je naam in wanneer de cursor knippert.

Voorbeelduitvoer (lijn 3 is wat de gebruiker heeft ingetypt)

```
1 Hoi, ik ben het!
2 Wie ben jij?!
3 tim [enter]
4 Dag
5 tim
6 hoe gaat het met je?
```

Aanhalingsteken of niet?

Wanneer je de inhoud van een variabele wil gebruiken in een methode zoals `WriteLine()` dan plaats je deze zonder aanhalingsteken!
Bekijk zelf eens wat het verschil wordt wanneer je volgende lijn code `Console.Write(result);` vervangt door `Console.Write("result");`.

De uitvoer wordt dan (merk het verschil op op lijn 5):

```
1 Hoi, ik ben het!
2 Wie ben jij?!
3 tim [enter]
4 Dag
5 result
6 hoe gaat het met je?
```

Write en WriteLine

Naast `WriteLine` bestaat er ook `Write`.

De `WriteLine`-methode zal steeds een line break (een 'enter') aan het einde van de lijn zetten zodat de cursor naar de volgende lijn springt.

De `Write`-methode daarentegen zal geen enter aan het einde van de lijn toevoegen. Als je dus vervolgens iets toevoegt (met een volgende `Write` of `WriteLine`) **dan zal dit aan dezelfde lijn toegevoegd worden**.

Vervang daarom eens in de laatste 3 lijnen code in je project `WriteLine` door `Write`:

```
1  Console.Write("Dag");
2  Console.Write(result);
3  Console.Write("hoe gaat het met je?");
```

Voer je programma uit en test het resultaat. Je krijgt nu:

```
1  Hoi, ik ben het!
2  Wie ben jij?!
3  tim [enter]
4  Dagtimhoe gaat het met je?
```

Wat is er "verkeerd" gelopen? Al je tekst van de laatste lijn plakt zo dicht bij elkaar? Inderdaad, we zijn spaties vergeten toe te voegen. Spaties zijn ook tekens die op scherm moeten komen (ook al zien we ze niet) en je dient dus binnen de aanhalingstekens spaties toe te voegen. Namelijk:

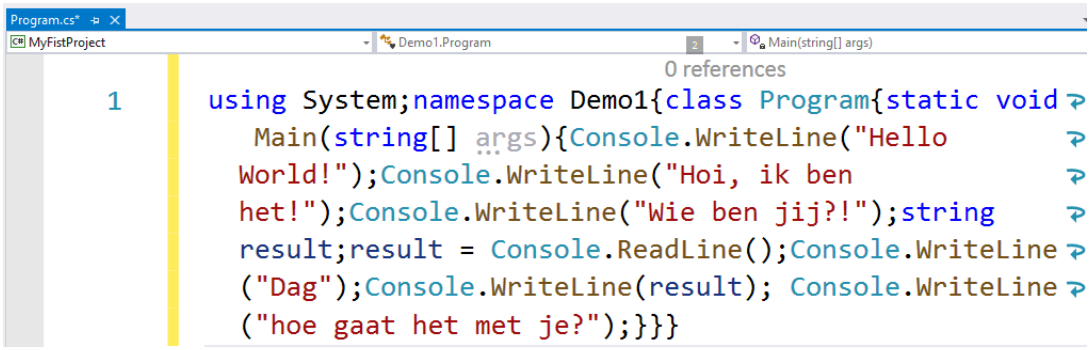
```
1  Console.Write("Dag ");
2  Console.Write(result);
3  Console.Write(" hoe gaat het met je?");
```

Je uitvoer wordt nu:

```
1  Hoi, ik ben het!
2  Wie ben jij?!
3  tim [enter]
4  Dag tim hoe gaat het met je?
```

Witregels in C#

C# trekt zich niets aan van **witregels die niét binnen aanhalingstekens staan** (zowel spaties, enters en tabs worden genegeerd). Met andere woorden: je kan het voorgaande programma perfect in één lange lijn code typen, zonder enters. Dit is echter niet aangeraden want het maakt je code een pak onleesbaarder.



Voorgaande programma in exact 1 lijn. Cool? Ja, in sommige kringen. Dom en onleesbaar? Ook ja.

Opletten met spaties

Let goed op hoe je spaties gebruikt bij `WriteLine`. **Indien je spaties buiten de aanhalingstekens plaatst dan heeft dit geen effect.**

Hier een fout gebruik van spaties (de code zal werken maar je spaties worden genegeerd):

```
1 //we visualiseren de spaties even als liggende streepjes in volgende voorbeeld
2 Console.Write("Dag"_);
3 Console.Write(result_);
4 Console.Write("hoe gaat het met je?");
```

En een correct gebruik:

```
1 Console.Write("Dag_");
2 Console.Write(result);
3 Console.Write("_hoe gaat het met je?");
```

Zinnen aan elkaar plakken

We kunnen dit allemaal nog een pak korter tonen zonder dat de code onleesbaar wordt. De plus-operator (+) in C# kan je namelijk gebruiken om tekst aan elkaar te plakken. De laatste 3 lijnen code kunnen dan korter geschreven worden als volgt:

```
Console.WriteLine("Dag " + result + " hoe gaat het met je?");
```

Merk op dat `result` dus NIET tussen aanhalingstekens staat, in tegenstelling tot de andere stukken van de zin. Waarom is dit? Aanhalingstekens in C# duiden aan dat een stuk tekst moet beschouwd worden als tekst van het type `string`. Als je geen aanhalingsteken gebruikt dan zal C# de tekst beschouwen als een variabele met die naam.

Bekijk zelf eens wat het verschil wordt wanneer je volgende lijn code:

```
Console.WriteLine("Dag "+ result + " hoe gaat het met je?");
```

Vervangt door:

```
Console.Write("Dag "+ "result" + " hoe gaat het met je?");
```

Meer input vragen

Als je meerdere inputs van de gebruiker wenst te bewaren dan zal je meerdere geheugenplekken (variabelen) nodig hebben. Bijvoorbeeld:

```
1 Console.WriteLine("Geef leeftijd");
2 string leeftijd; //eerste variabele aanmaken
3 leeftijd = Console.ReadLine();
4 Console.WriteLine("Geef adres");
5 string adres; //tweede variabele aanmaken
6 adres = Console.ReadLine();
```

Je mag echter ook de variabelen al vroeger aanmaken. In C# zet men de geheugenplek creatie zo dicht mogelijk bij de code waar je die plek gebruikt (zoals vorig voorbeeld), maar dat is geen verplichting. Dit mag dus ook:

```
1 string leeftijd; //eerste variabele aanmaken
2 string adres; //tweede variabele aanmaken
3 Console.WriteLine("Geef leeftijd");
4 leeftijd = Console.ReadLine();
5 Console.WriteLine("Geef adres");
6 adres = Console.ReadLine();
```

Je zal vaak `Console.WriteLine` moeten schrijven als je dit boek volgt. We hebben echter goed nieuws voor je: er zit een ingebouwde "snippet" in VS om sneller `Console.WriteLine` op het scherm te toveren. We gaan je niet langer in spanning houden…of toch… nog even. Ben je benieuwd? Spannend he!

Hier gaan we: `cw [tab] [tab]`

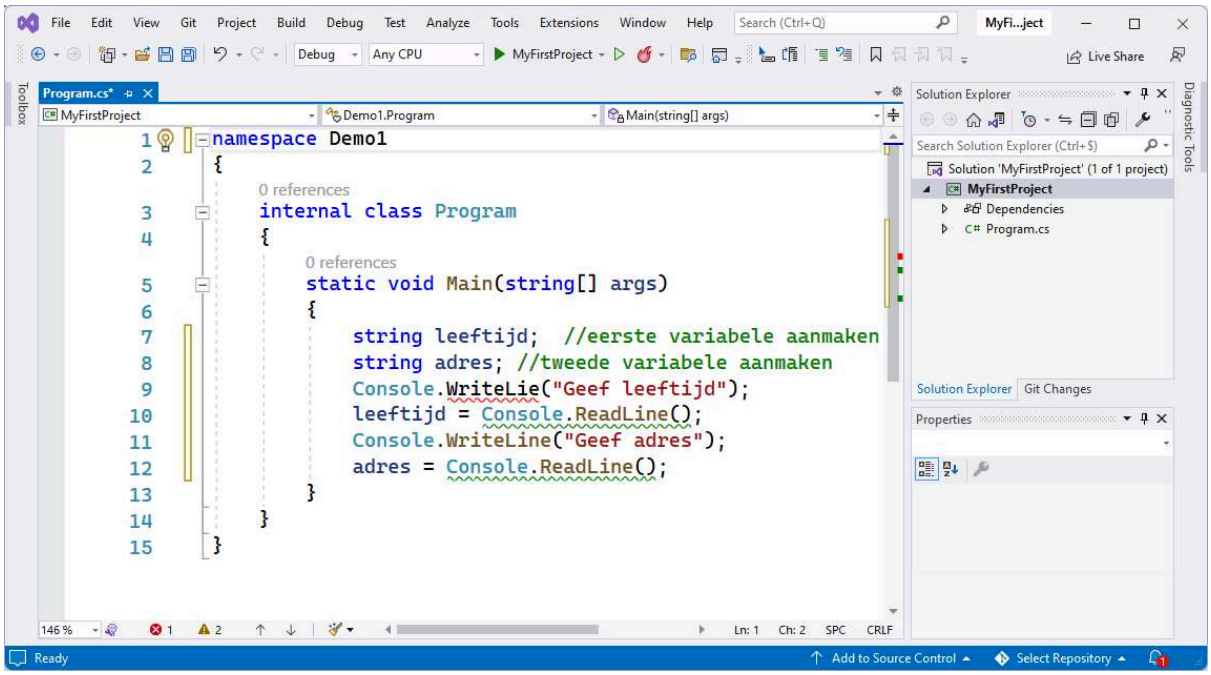
Als je dus `cw` schrijft en dan twee maal op de tab-toets van je toetsenbord duwt verschijnt daar *automagisch* een verse lijn met `Console.WriteLine();`.

1.4 Fouten oplossen

Je code zal pas compileren indien deze foutloos is geschreven. Herinner je dat computers uiterst dom zijn en dus vereisen dat je code 100% foutloos is qua woordenschat en grammatica.

Zolang er dus fouten in je code staan moet je deze eerst oplossen voor je verder kan. Gelukkig helpt VS je daarmee op 2 manieren:

- Fouten in code worden met een rode squiggly onderlijnd.
- Onderaan zie je in de statusbalk of je fouten hebt.



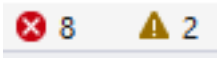
Zie je de fout?

Laat je trouwens niet afschrikken door de gigantische reeks fouten die soms plots op je scherm verschijnen. VS begint al enthousiast fouten te zoeken terwijl je mogelijk nog volop aan het typen bent.

Als je plots veel fouten krijgt, kijk dan altijd vlak boven de plek waar de fouten verschijnen. Heel vaak zit daar de echte fout:en meestal is dat gewoon het ontbreken van een kommapunt aan het einde van een statement.

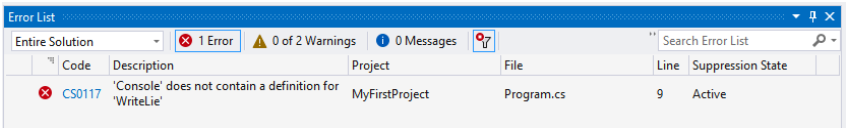
Fouten sneller vinden

Uiteraard ga je vaak code hebben die meerdere schermen omvat. Je kan via de error-list snel naar al je fouten gaan. Open deze door op het error-icoontje onderaan te klikken:



So many errors?!

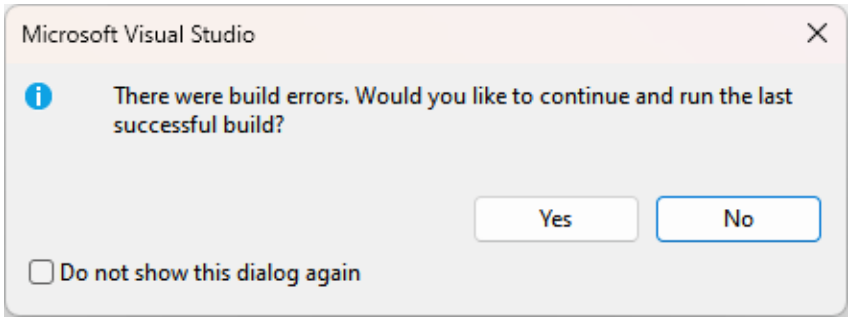
Dit zal de "error list" openen (**een schermdeel van VS dat we aanraden om altijd open te laten én dus niet weg te klikken**). Warnings kunnen we (voorlopig) meestal negeren en deze 'filter' hoef je dus niet aan te zetten.



De error list.

In de error list kan je nu op iedere error klikken om ogenblikkelijk naar de correcte lijn te gaan.

Zou je toch willen compileren en je hebt nog fouten dan zal VS je proberen tegen te houden. **Lees nu onmiddellijk wat de voorman hierover te vertellen heeft**.



OPLETTEN!

Opletten aub : Indien je op de groene start knop duwt en bovenstaande waarschuwing krijgt **KLIK DAN NOOIT OP YES EN DUID NOOIT DE CHECKBOX AAN!**

Lees de boodschap eens goed na: wat denk je dat er gebeurt als je op 'yes' duwt? Inderdaad, VS zal de laatste werkende versie uitvoeren en dus niet de code die je nu hebt staan waarin nog fouten staan.

De voorman verschijnt wanneer er iets beschreven wordt waar véél fouten op gemaakt worden, zelfs bij ervaren programmeurs. Opletten geblazen dus.

Fouten oplossen met lampje

Wanneer je je cursor op een lijn met een fout zet dan zal je soms vooraan een geel error-lampje zien verschijnen (dit duurt soms even):

Lampje: de brenger der oplossingen…In tegenstelling tot Clippy de Office assistent uit de jaren '90….

Je kan hier op klikken en heel vaak krijg je dan ineens een mogelijke oplossing. **Wees steeds kritisch** hierover want VS is niet alwetend en kan niet altijd raden wat je bedoelt. Neem dus het voorstel niet zomaar over zonder goed na te denken of het dat was wat je bedoelde.

Warnings kan je over het algemeen, zeker in het begin, negeren. Bekijk ze gewoon af en toe, wie weet bevatten ze nuttige informatie om je code te verbeteren.

Meest voorkomende fouten

De meest voorkomende fouten die je als beginnende C# programmeur maakt zijn:

- **Puntkomma** vergeten.
- **Schrijffouten** in je code, bijvoorbeeld `RaedLine` i.p.v. `ReadLine`.
- Geen rekening gehouden met **hoofdletter gevoeligheid van C#**, bijvoorbeeld `Readline` i.p.v. `ReadLine` (zie verder).
- Per ongeluk **accolades verwijderd**.
- Code geschreven op plekken waar dat niet mag (je mag momenteel enkel binnen de accolades van `Main` schrijven).

1.5 Kleuren in console

Je kan in console-applicaties zelf bepalen in welke kleur nieuwe tekst op het scherm verschijnt. Je kan zowel de kleur van het lettertype instellen (via `ForegroundColor`) als de achtergrondkleur (`BackgroundColor`).

Je kan met de volgende expressies de console-kleur veranderen, bijvoorbeeld de achtergrond in blauw en de letters in groen:

```
1 Console.BackgroundColor = ConsoleColor.Blue;
2 Console.ForegroundColor = ConsoleColor.Green;
```

Vanaf dan zal alle tekst die je hierna met `WriteLine` en `Write` naar het scherm stuurt met deze kleuren werken. Merk op dat we **bestaande tekst op het scherm niét van kleur kunnen veranderen zonder deze eerst te verwijderen en dan opnieuw, met andere kleurinstellingen, naar het scherm te sturen**.

Alle kleuren die beschikbaar zijn staan beschreven in `ConsoleColor` deze zijn: Black, DarkBlue, DarkGreen, DarkCyan, DarkRed, DarkMagenta, DarkYellow, Gray, DarkGray, Blue, Green, Cyan, Red, Magenta, Yellow.

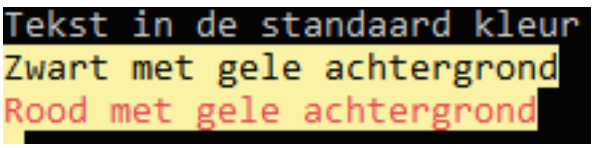
Wens je dus de kleur Red dan zal je deze moeten aanroepen door er `ConsoleColor.` voor te zetten: `ConsoleColor.Red`.

Waarom is dit? `ConsoleColor` is een zogenaamd `enum`-type, een concept dat we verderop in hoofdstuk 5 zullen bespreken.

Een voorbeeld:

```
1 Console.WriteLine("Tekst in de standaard kleur");
2 Console.BackgroundColor = ConsoleColor.Yellow;
3 Console.ForegroundColor = ConsoleColor.Black;
4 Console.WriteLine("Zwart met gele achtergrond");
5 Console.ForegroundColor = ConsoleColor.Red;
6 Console.WriteLine("Rood met gele achtergrond");
```

Als je deze code uitvoert krijg je als resultaat:



Resultaat voorgaande code.

Kleur in console gebruiken is nuttig om je gebruikers een minder eentonig en meer informatieve applicatie aan te bieden. Je zou bijvoorbeeld alle foutmeldingen in het rood kunnen laten verschijnen. Let er wel op dat je applicatie geen aartslelijk, op psychedelische producten geschreven, programma lijkt.
Hou er ook rekening mee dat niet iedereen (alle) kleuren kan zien. In de vorige editie van dit boek gebruikte ik rode letters op een groene achtergrond…Dat resulteerde in onleesbare tekst voor mensen met Daltonisme.

Kleur resetten

Soms wil je terug de originele applicatie-kleuren hebben. Je zou manueel dit kunnen instellen, maar wat als de gebruiker slechtziend is en in z'n besturingssysteem andere kleuren als standaard heeft ingesteld?!

De veiligste manier is daarom de kleuren te resetten door de `Console.ResetColor()` methode aan te roepen zoals volgend voorbeeld toont:

```
1 Console.ForegroundColor = ConsoleColor.Red;
2 Console.WriteLine("Error!!!! Contacteer de helpdesk");
3 Console.ResetColor();
4 Console.WriteLine("Het programma sluit nu af");
```

1.6 Oefeningen

We sluiten ieder hoofdstuk af met een kleine selectie oefeningen. In de mate van het mogelijk zijn ze gerangschikt van eenvoudig tot iets pittiger. Bekijk zeker **ziescherp.be** waar je een grote hoeveelheid extra oefeningen zal terugvinden, alsook de oplossingen voor de meeste oefeningen.

Visitekaartje

Schrijf een programma om de volgende zaken te tonen op afzonderlijke regels

- Naam: *voornaam achternaam*
- Adres: *straat en gemeente*
- Hobby: *hobby*
- Waarom wil je leren programmeren? *Argumentatie*

Wat cursief staat moet vervangen worden door je eigen gepaste waarden

Zorg ervoor dat de titel van ieder element in een andere kleur getoond wordt.

Communicatiefouten

Schrijf een applicatie met behulp van `ReadLine()` en `WriteLine()`-methoden waarbij de computer aan de gebruiker om zijn of haar favoriete kleur, eten, auto, film en boek vraagt. Het programma zal de antwoorden echter door elkaar halen waardoor de computer vervolgens toont:

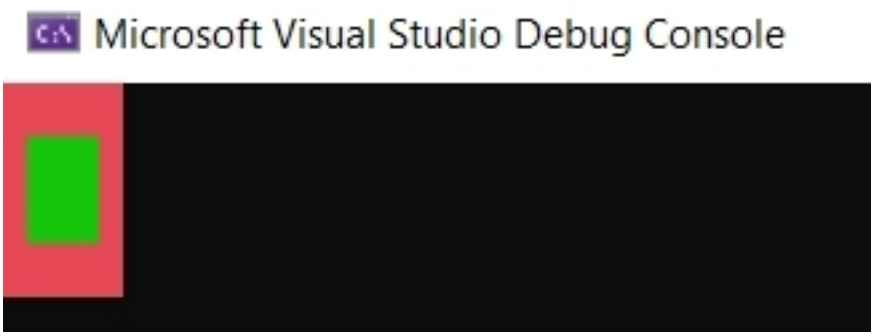
```
Je favoriete kleur is [eten]. Je eet graag [auto].
Je lievelingsfilm is [boek] en je favoriete boek is [kleur].
```

Waarbij tussen de rechte haakjes steeds de invoer komt die de gebruiker eerder opgaf voor de bijhorende vraag.

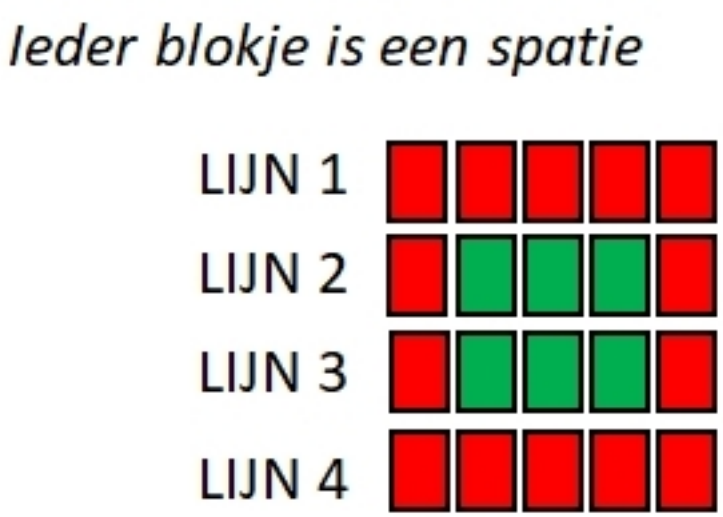
Maak het programma “grappig” door de antwoorden op de verkeerde plek te gebruiken, bijvoorbeeld: “Zo, je favoriete kleur is The Lord of the Rings?!”.

Tekening

Kan je volgende afbeeldingen namaken in de console?



Volgende tekening toont een schematische weergave:



Je kan een gekleurd vakje 'tekenen' door de `BackGroundColor` van de console in te stellen en dan één of meerdere **spaties** naar het scherm te sturen met `Write` of `WriteLine`.

Muziek*

Met de `Console.Beep()` methode kan je muziek maken. Volgende voorbeeld toont bijvoorbeeld hoe je do-re-mi-fa-sol-la-si-do afspeelt:

```
1  Console.Beep(264, 1000);
2  Console.Beep(297, 1000);
3  Console.Beep(330, 1000);
4  Console.Beep(352, 1000);
5  Console.Beep(396, 1000);
6  Console.Beep(440, 1000);
7  Console.Beep(495, 1000);
8  Console.Beep(528, 1000);
```

Je geeft aan `Beep` 2 getallen mee (*argumenten*):

1. De frequentie van de toon die moet afgespeeld worden. Bijvoorbeeld 264 (in Hertz).
2. De duur dat de toon moet afgespeeld worden in milliseconden. Als je dus 1000 meegeeft zal de toon gedurende 1000 ms, oftewel 1 seconde, afgespeeld worden.

Open 1 van de eerder gemaakte oefeningen en zorg ervoor dat bij het opstarten ervan er een kort, door jezelf gecomponeerd, introliedje wordt afgespeeld.

2. De basisconcepten van C#

Om een werkend C#-programma te maken moeten we de C#-taal beheersen. Net zoals iedere taal, bestaat ook C# uit enerzijds grammatica, in de vorm van de **C# syntax** en anderzijds woordenschat in de vorm van de te gebruiken gereserveerde **keywords**.

Een C#-programma bestaat uit een opeenvolging van instructies ook wel **statements** genoemd. **Deze eindigen steeds met een puntkomma (`;`)** (zoals ook in het Nederlands een zin eindigt met een punt). Ieder statement kan je vergelijken als één lijn in ons recept, het algoritme.

De volgorde van de woorden (keywords, variabelen, enz.) zijn niet vrijblijvend en moeten aan (grammaticale) regels voldoen. Enkel indien alle statements correct zijn zal het programma gecompileerd worden naar een werkend en uitvoerbaar programma (zoals in het vorige hoofdstuk besproken).

Enkele belangrijke regels van C#:

- **Hoofdlettergevoelig**: C# is hoofdlettergevoelig. Dat wil zeggen dat hoofdletter `R` en kleine letter `r` totaal verschillende zaken zijn voor C#. `Reinhardt` en `reinhardt` zijn dus ook niet hetzelfde.
- **Statements afsluiten met puntkomma**: Iedere C# statement wordt afgesloten moet een puntkomma (`;`). Doe je dat niet dan zal C# denken dat de regel gewoon op de volgende lijn doorloopt en deze als één (fout) geheel proberen te compileren.
- **Witruimtes**: Spaties, tabs en enters worden door de C# compiler genegeerd. Je kan ze dus gebruiken om de layout van je code (*bladspiegel* zeg maar) te verbeteren. De enige plek waar witruimtes wél een verschil geven is tussen aanhalingstekens `" "` die we later (bij string) zullen leren gebruiken.
- **Commentaar toevoegen kan**: door `//` voor een enkele lijn te zetten zal deze lijn genegeerd worden door de compiler. Je kan ook meerdere lijnen code in commentaar zetten door er `/*` voor en `*/` achter te zetten. Voorts zijn er 2 handige knoppen die toelaten om een heel blok code in één keer van commentaar te voorzien of uit commentaar te halen (zie verder).
- **Van boven naar onder**: je code wordt van boven naar onder uitgevoerd en zal enkel naar andere plaatsen springen als je daar expliciet in je code om vraagt (bijvoorbeeld met behulp van loops (hoofdstuk 6) of methoden (hoofdstuk 7)).
- **Je code begint altijd in de `Main`-methode!!!**

2.1 Keywords: de woordenschat

C# bestaat zoals gezegd niet enkel uit grammaticale regels. Grammatica zonder woordenschat is nutteloos. Er zijn binnen C# dan ook momenteel 80 woorden, zogenaamde **reserved keywords** die de woordenschat voorstellen. Het spreekt voor zich dat deze keywords een eenduidige, specifieke betekenis hebben en dan ook enkel voor dat doel gebruikt kunnen worden.
In dit boek zullen we stelselmatig deze keywords leren kennen en gebruiken op een correcte manier om zo werkende code te maken.

Deze keywords zijn:

abstract	*as*	*base*	**bool**
break	**byte**	**case**	*catch*
char	checked	*class*	**const**
continue	**decimal**	*default*	delegate
do	**double**	**else**	**enum**
event	explicit	extern	**false**
finally	fixed	**float**	**for**
foreach	goto	**if**	implicit
in	**int**	*interface*	internal
is	lock	**long**	**namespace**
new	*null*	*object*	*operator*
out	*override*	params	*private*
protected	*public*	readonly	**ref**
return	**sbyte**	*sealed*	**short**
sizeof	stackalloc	*static*	**string**
struct	**switch**	*this*	*throw*
true	*try*	typeof	**uint**
ulong	unchecked	unsafe	**ushort**
using	using static	*virtual*	**void**
volatile	**while**		

De keywords in vet zijn keywords die we in het eerste deel van dit boek zullen bekijken (hoofdstuk 1 tot en met 8). Die in cursief in het tweede deel (9 en verder). De overige zal je zelf moeten ontdekken (of mogelijk zelfs nooit in je carrière gebruiken vanwege hun soms obscure nut).

C# is een levende taal. Soms verschijnen er dan ook nieuwe keywords. De afspraak is echter dat de lijst hierboven niet verandert. Nieuwe keywords maken deel uit van de *contextual keywords* en zullen nooit gereserveerde keywords worden. We zullen enkele van deze "nieuwere" keywords tegenkomen waaronder: `get`, `set`, `value` en `var`.

Aandacht, aandacht! Step away from the keyboard! I repeat. Step away from the keyboard. Hierbij wil ik u attent maken op een belangrijke, onbeschreven, wet voor C# programmeurs: "**NEVER EVER USE `goto`**"

Het moet hier alvast even uit m'n systeem. `goto` is weliswaar een officieel C# keyword, toch zal je het in dit boek **nooit** zien terugkomen in code. Je kan alle problemen in je algoritmes oplossen zonder ooit `goto` nodig te hebben.

Voel je toch de drang: **don't!** Simpelweg, don't. Het is het niet waard. Geloof me.

NEVER USE GOTO.

Enneuh, ik hou je in't oog hoor!