

1.0.0

H2: Variabelen en datatypes

De syntaxis van C#

✓ Kennisclip voor deze inhoud

Statements en de C# syntax

Om een uitvoerbaar C#-programma te maken moeten we bepaalde regels respecteren. We noemen deze regels de **syntaxis** of *syntax* van de taal.

Een C#-programma bestaat uit een opeenvolging van instructies ook wel **statements** genoemd. Met de uitzondering van **blokken** (ook *blocks*, afgebakend door `{ en }`) eindigen deze steeds met een **puntkomma** (`;`)

De volgorde van de woorden is niet vrijblijvend en moet aan grammaticale regels voldoen. Enkel indien alle statements correct zijn zal het programma gecompileerd worden naar een werkend en uitvoerbaar programma (zoals in een vorige sectie besproken).

Enkele belangrijke regels van C#:


- **Hoofdletter-gevoelig:** C# is hoofdlettergevoelig. Dat wil zeggen dat hoofdletter `T` en kleine letter `t` totaal verschillende zaken zijn voor C#. `Reinhardt` en `reinhardt` zijn dus ook niet hetzelfde.
- **Statements afsluiten met puntkomma:** Ieder C# statement dat geen block is, 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.
- **Blocks:** Een block begint met `{`, eindigt met `}` en bevat daartussen verdere statements (ook geneste blocks zijn toegelaten).
- **Witruimtes:** Spaties, tabs en newlines worden door de C# compiler genegeerd. Je kan ze dus gebruiken om de layout van je code 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:** met behulp van `//` voor een enkele lijn en `/* */` voor meerdere lijnen commentaar. Commentaar zal door de compiler genegeerd worden.


Keywords

Er zijn binnen C# dan ook 80 woorden, zogenaamde **reserved keywords** die deel van de taal zelf uitmaken. In deze cursus zullen we stelselmatig deze keywords leren kennen en gebruiken op een correcte manier om zo werkende code te maken.

Deze keywords zijn:

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

 De keywords in vet zijn keywords die we dit semester zullen kennen. Die in cursief in het tweede semester. De overige zal je zelf moeten leren.

 Indien je deze tabel in pdf bekijkt zal deze om zeep zijn. Onze gitbook gnomes proberen dit op te lossen maar voorlopig vinden ze helaas geen oplossing, waarvoor onze excuses.

Variabelen, identifiers en naamgeving

We hebben variabelen nodig om data een naam te geven, zodat we er later naar kunnen verwijzen. Wanneer we een statement schrijven dat bijvoorbeeld input van de gebruiker moet vragen, dan willen we die input ook bewaren. Zo kunnen we verderop in het programma iets met deze data doen. We doen hetzelfde in ons hoofd wanneer we bijvoorbeeld zeggen: "tel 3 en 4 op en vermenigvuldig dat resultaat met 5". In die zin is "dat resultaat" een manier om te verwijzen naar wat we eerder gedaan hebben. Met andere woorden, een variabele. Vervolgens zullen we de inhoud van die variabele vermenigvuldigen met 5.

Wanneer we een variabele aanmaken, zal deze moeten voldoen aan enkele afspraken. Zo moeten we minstens 2 zaken aangeven:

- Het type van de variabele: het **datatype** dat aangeeft wat voor data we wensen op te slaan (tekst, getal, afbeelding, etc.).
- De naam van de variabele: de **identificer** waarmee we snel aan de variabele-waarde kunnen.

De verschillende datatypes bespreken we in een volgend [hoofdstuk](#).

Regels voor identifiers

De code die we gaan schrijven moet voldoen aan een hoop regels. Wanneer we in onze code zelf namen (**identifiers**) moeten geven aan **variabelen** (en later ook methoden, objecten, etc.) dan moeten we een aantal regels volgen:

- **Hoofdlettergevoelig**: de identifiers `tim` en `Tim` zijn verschillend, zoals reeds vermeld.
- **Geen keywords**: identifiers mogen geen gereserveerde C# keywords zijn. De keywords van hierboven mogen dus niet. Varianten waarbij de hoofdletters anders zijn mogen wel, bijvoorbeeld: `gOTO` en `stRING` mogen dus wel, maar niet `goto` of `string` daar beide een gereserveerd keyword zijn maar dankzij de hoofdlettergevoelig-regel is dit dus toegelaten. `INT` mag ook, maar niet `int`.
- **Eerste karakter-regel**: het eerste karakter van de identifier mag enkel zijn:
 - kleine of grote letter
 - liggend streepje (`_`)
- **Alle andere karakters-regels**: de overige karakters mogen enkel zijn:
 - kleine of grote letter
 - liggend streepje
 - een cijfer (`0` tot en met `9`)
- **Lengte**: Een legale identifier mag zo lang zijn als je wenst, maar je houdt het best leesbaar.

Enkele voorbeelden

Enkele voorbeelden van toegelaten en niet toegelaten identifiers:

identificer	toegelaten?	uitleg indien niet toegelaten
werknemer	ja	
kerst2018	ja	
pippo de clown	neen	geen spaties toegestaan
4dPlaats	neen	mag niet starten met getal
_ILOVE2019	ja	
Tor+Bjorn	neen	enkel cijfers, letters en liggen streepjes toegestaan
ALLCAPSMAN	ja	
B_A_L	ja	
class	neen	gereserveerd keyword
WriteLine	ja	

Naamgeving afspraken

Er zijn geen vaste afspraken over hoe je je variabelen moet noemen toch hanteren we enkele **coding guidelines** die doorheen je opleiding moeten gevolgd worden. Naarmate we meer C# leren zullen er extra guidelines bijkomen (zie [deze pagina met richtlijnen](#)).

- **Duidelijke naam:** de identificer moet duidelijk maken waarvoor de identificer dient. Schrijf dus liever `gewicht` of `leeftijd` in plaats van `a` of `meuh`.
- **Camel casing:** gebruik camel casing indien je meerdere woorden in je identificer wenst te gebruiken. Camel casing wil zeggen dat ieder nieuw woord terug met een hoofdletter begint. Een goed voorbeeld kan dus zijn `leeftijdTimDams` of `aantalLeerlingenKlas1EA`. Merk op dat we liefst het eerste woord met kleine letter starten. Uiteraard zijn er geen spaties toegelaten.
- **Pascal casing:** Zoals camel casing, maar ook de eerste letter is een hoofdletter. Gebruik dit voor namespaces, klassen en voor methodes waarbij je `public` zet.

Commentaar

Soms wil je misschien extra opmerkingen bij je code zetten. Als je dat gewoon zou doen (bv `Dit deel zal alles verwijderen`), dan zal je compiler niet begrijpen wat die zin doet. Hij verwacht namelijk C# en geen Nederlandstalige zin. Om dit op te lossen kan je in je code op twee manieren aangeven dat een stuk tekst gewoon commentaar is en mag genegeerd worden door de compiler:

Enkele lijn commentaar

Eén lijn commentaar geef je aan door de lijn te starten met twee voorwaartse slashes `//`. Uiteraard mag je ook meerdere lijnen op deze manier in commentaar zetten. Zo wordt dit ook vaak gebruikt om tijdelijk een stuk code "uit te schakelen". Ook mogen we commentaar *achter* een stuk C# code plaatsen (zie voorbeeld hieronder).

```
//De start van het programma
int getal=3;
//Nu gaan we rekenen
int result = getal * 5;
// result= 3*5;
Console.WriteLine(result); //We tonen resultaat op scherm: 15
```

Blok commentaar

We kunnen een stuk tekst als commentaar aangeven door voor de tekst `/*` te plaatsen en `*/` achteraan. Een voorbeeld:

```
/*
    Veel commentaar.
    Een heel verhaal
    Mooi he.
    Is dit een haiku?
*/
int leeftijd= 0;
leeftijd++;
```

Datatypes

✓ Kennisclip voor deze inhoud

Een essentieel onderdeel van C# is kennis van datatypes. Een **datatype** is, zoals de naam het zegt, een soort waartoe bepaalde gegevens kunnen behoren. Wanneer je data wenst te bewaren in je applicatie dan zal je je moeten afvragen wat voor soort data het is. Gaat het om een getal, een geheel getal, een kommagetal, een stuk tekst of misschien een binaire reeks? Een variabele met een bepaald datatype kan een bepaald soort data bewaren en dit zal afhankelijk hiervan een bepaalde hoeveelheid computergeheugen vereisen.

Er zijn tal basistypes in C# gedeclareerd (zogenaamde **built-in datatypes**). Dit semester leren we werken met datatypes voor:

- Gehele getallen: `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`
- Kommagetallen: `double`, `float`, `decimal`
- Tekst: `char`, `string`
- Booleans: `bool`

i Het datatype `string` heb je al in actie gezien in het vorig hoofdstuk. Je hebt toen al een variabele aangemaakt van het type `string` door de zin `string result;`.

Verderop koppelden we de naam `result` dan aan het resultaat van een actie, namelijk inlezen van tekst via `Console.ReadLine` (eerst wordt dat resultaat uitgerekend, dan pas wordt het aan de naam gekoppeld):

```
result = Console.ReadLine();
```

Basistypen voor getallen

C# heeft een hoop datatypes gedefinieerd om te werken met getallen zoals wij ze kennen, gehele en kommagetallen. Intern zullen deze getallen steeds binair bewaard worden, maar dat merken we zelden tijdens het programmeren.

De basistypen van C# om getallen in op te slaan zijn:

- Voor gehele getallen: `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`
- Voor kommagetallen: `double`, `float`, `decimal`

Deze datatypes hebben allemaal een bepaald bereik, wat een rechtstreeks gevolg is van de hoeveelheid geheugen die ze innemen.

Gehele getallen

Voor de gehele getallen:

Type	Geheugen	Bereik	Meer info
<code>sbyte</code>	8 bits	-128 tot 127	info
<code>byte</code>	8 bits	0 tot 255	info
<code>short</code>	16 bits	-32768 tot 32767	info
<code>ushort</code>	16 bits	0 tot 65535	info
<code>int</code>	32 bits	-2 147 483 648 tot 2 147 483 647	info
<code>uint</code>	32 bits	0 tot 4294967295	info
<code>long</code>	64 bits	-9 223 372 036 854 775 808 tot 9 223 372 036 854 775 807	info
<code>ulong</code>	64 bits	0 tot 18 446 744 073 709 551 615	info
<code>char</code>	16 bits	0 tot 65535	info

Enkele opmerkingen bij deze tabel:


- De **s** vooraan `sbyte` types staat voor **signed** : m.a.w. 1 bit wordt gebruikt om het + of - teken te bewaren.
- De **u** vooraan `ushort`, `uint` en `ulong` staat voor **unsigned**. Het omgekeerde van signed dus. Kwestie van het ingewikkeld te maken. Deze twee datatypes hebben dus geen teken en zijn **altijd positief**.
- `char` heeft dezelfde kenmerken als `ushort`, maar dient voor iets anders. `ushort` gebruik je echt om getallen binnen dat bereik voor te stellen. `char` bewaart karakters, d.w.z. symbolen zoals letters. Een beetje vereenvoudigd kan je stellen dat er een tabel is waarin elk symbool gekoppeld is aan een getal, zodat een reeks nulletjes en eentjes ook een letter kan voorstellen.

Kommagetallen

Voor de kommagetallen zijn er maar 3 mogelijkheden. Ieder datatype heeft een voordeel tegenover de 2 andere, dit voordeel staat vet in de tabel:

Type	Geheugen	Bereik	Precisie	
<code>float</code>	32 bits	$\pm 1.5 \cdot 10^{-45}$ to $\pm 3.4 \cdot 10^{38}$	6 to 9 digits	info
<code>double</code>	64 bits	$\pm 5.0 \cdot 10^{-324}$ to $\pm 1.7 \cdot 10^{308}$	15 to 17 digits	info
<code>decimal</code>	128 bits	$\pm 1.0 \cdot 10^{-28}$ to $\pm 7.9228 \cdot 10^{28}$	28 to 29 digits	info

Zoals je ziet moet je bij kommagetallen een afweging maken tussen 3 even belangrijke criteria. Heb je zeer grote precisie (veel cijfers na de komma) nodig, dan ga je voor een `decimal`. Wil je vooral erg grote of erg kleine getallen (met meer kans op afrondingen), dan kies je voor `double`.

 Bij twijfel opteren we meestal voor kommagetallen om het **double** datatype te gebruiken. Bij gehele getallen kiezen we meestal voor **int**.

Boolean datatype

Het `bool` (**boolean**) is het eenvoudigste datatype van C#. Het kan maar 2 mogelijke waarden bevatten: `true` of `false`. 1 of 0 met andere woorden. In het Nederlands meestal uitgedrukt als "waar" en "niet waar".

We zullen het `bool` datatype erg veel nodig hebben wanneer we met **beslissingen** zullen werken, specifiek de **if statements** die afhankelijk van de uitslag van een `bool` bepaalde code wel of niet zullen doen uitvoeren.

Tekst/String datatype

We besteden verderop een heel apart hoofdstuk aan tonen hoe je tekst en enkele karakters kan bewaren in variabelen. De basis:

- Tekst kan bewaard worden in het `string` datatype
 - Letterlijke tekst schrijf je tussen dubbele quotes, bijvoorbeeld `"Hallo Wereld!"`
- Een enkel karakter wordt bewaard in het `char` datatype, dat we ook hierboven al even hebben zien passeren.
 - Letterlijke karakter schrijf je tussen enkele quotes, bijvoorbeeld `'à'`

Meer info vind je later in [dit hoofdstuk](#).

Variabelen

✓ Kennisclip voor deze inhoud

Variabelen

In het vorige hoofdstuk zagen we dat er verschillende datatypes bestaan. Deze types hebben we nodig om **variabelen** aan te maken. Een variabele is een koppeling van een naam aan gegevens. In C# heeft elke variabele ook een type.

Een variabele wordt bijgehouden in het geheugen van je machine, maar in een programmeertaal als C# vragen we ons niet af waar in het geheugen. In plaats daarvan gebruiken we de naam van de variabele, de zogenaamde **identifier**, om de gekoppelde gegevens op te vragen.

De naam (identifier) van de variabele moet voldoen aan de identifier regels onder "Inleiding -> Afspraken code".

Variabelen aanmaken en gebruiken

Om een variabele te maken moeten we deze **declareren**, door een type en naam te geven. Vanaf dan zal de computer een hoeveelheid geheugen voor je reserveren. Hiervoor dien je op te geven:

1. Het **datatype** (bv `int`, `double`).
2. Een **identifier** zodat de variabele uniek kan geïdentificeerd worden ([volgens de naamgevingsregel van C#](#)).

Een variabele declaratie heeft als syntax:

```
datatype identifier;
```

Bijvoorbeeld: `int leeftijd;`

Je mag ook meerdere variabelen van het zelfde datatype in 1 enkele declaratie aanmaken door deze met komma's te scheiden:

```
datatype identifier1, identifier2, identifier3;
```

Bijvoorbeeld `string voornaam, achternaam, adres;`

Indien je reeds weet wat de beginwaarde moet zijn van de variabele dan mag je de variabele ook reeds deze waarde toekennen bij het aanmaken. Dit noemen we de **initialisatie** van de variabele.

```
int mijnLeeftijd = 37;
```

Eens een variabele is geïnitieerd, kunnen we deze (op de meeste plaatsen) gebruiken alsof we de gekoppelde waarde rechtstreeks gebruiken.

Waarden toekennen aan variabelen

Een initialisatie is een speciaal geval van een **toekenning**. Een toekenning houdt in dat je de waarde die bij een bepaalde naam hoort instelt. In C# mag dit ook indien de variabele al een waarde heeft.

Met de **toekennings-operator (=)** kan je een waarde toekennen aan een variabele. Hierbij kan je zowel een letterlijke waarde toekennen oftewel het resultaat van een berekening (een "expressie").

Je kan ook een waarde uit een variabele uitlezen en toewijzen aan een andere variabele:

```
int eenAndereLeeftijd = mijnLeeftijd;
```

Literal toewijzen

Literals (of "letterlijke waarden") zijn expliciet uitgeschreven waarden in je code. Als je in je code expliciet de waarde 4 wilt toekennen aan een variabele dan is het getal 4 in je code een zogenaamde literal. Wanneer we echter data bijvoorbeeld eerst uitlezen of berekenen (via bijvoorbeeld invoer van de gebruiker of als resultaat van een berekening) en het resultaat hiervan toekennen aan een variabele dan is dit geen literal.

Voorbeelden van een literal toekennen:

```
int temperatuurGisteren = 20;  
int temperatuurVandaag = 25;
```

Het is belangrijk dat het type van de literal overeenstemt met dat van de variabele waaraan je deze zal toewijzen. Een `string` literal stel je voor met aanhalingstekens. Volgende code zal dan ook een compiler-fout generen, daar je een `string` literal aan een `int`-variabele wil toewijzen, en vice versa.

```
string eenTekst;  
int eenGetal;  
  
eenTekst = 4;  
eenGetal = "4";
```

Als je bovenstaande probeert te compileren dan krijg je volgende error-boodschappen:

Error List		
Entire Solution ▾		<div>✖ 2 Errors</div> <div>⚠ 0 Warnings</div>
	Code	Description
✖	CS0029	Cannot implicitly convert type 'int' to 'string'
✖	CS0029	Cannot implicitly convert type 'string' to 'int'

Literal bepaalt het datatype

De manier waarop je een literal schrijft in je code zal bepalen wat het datatype van de literal is:

- **Gehele getallen** worden standaard als `int` beschouwd, vb: `125`.
- **Kommagetallen** (met punt `.`) worden standaard als `double` beschouwd, vb: `12.5`.
- Via een **suffix** na het getal kan je aangeven als het om andere types gaat:
 - `U` of `u` voor `uint`, vb: `125U`.
 - `L` of `l` voor `long`, vb: `125L`.
 - `UL` of `ul` voor `ulong`, vb: `125ul`.
 - `F` of `f` voor `float`, vb: `12.5f`.
 - `M` of `m` voor `decimal`, vb: `12.5M`.
- Voor **bool** (zie verder) is dit enkel `true` of `false`.
- Voor **char** (zie verder) wordt dit aangeduid met een enkele apostrof voor en na de literal, vb: `'q'`.
- Voor **string** (zie verder) wordt dit aangeduid met aanhalingsteken voor en na de literal, vb: `"pikachu"`.

De overige types `sbyte`, `short` en `ushort` hebben geen literal aanduiding. Er wordt vanuit gegaan wanneer je een literal probeert toe te wijzen aan een van deze types dat dit zonder problemen zal gaan (ze worden impliciet geconverteerd).

Nieuwe waarden overschrijven oude waarden

Wanneer je een reeds gedeclareerde variabele een **nieuwe waarde toekent** dan zal de oude waarde in die variabele onherroepelijk verloren zijn. Probeer dus altijd goed op te letten of je de oude waarde nog nodig hebt of niet. Wil je de oude waarde ook nog bewaren dan zal je een nieuwe, extra variabele moeten aanmaken en daarin de nieuwe waarde moeten bewaren:

```
int temperatuurGisteren = 20;
temperatuurGisteren = 25;
```

In dit voorbeeld zal er dus voor gezorgd worden dat de oude waarde van `temperatuurGisteren`, `20` , overschreven zal worden met `25` .

Volgende code toont hoe je bijvoorbeeld eerst de vorige waarde kunt bewaren en dan overschrijven:

```
int temperatuurGisteren= 20;
//Doe vanalles
//...
//1 dag later
int temperatuurEerGisteren= temperatuurGisteren; //Vorige temperatuur in eergisteren bewaren
temperatuurGisteren = 25; //temperatuur nu overschrijven
```

We hebben dus aan het einde van het programma zowel de temperatuur van eergisteren, `20` , als die van vandaag, `25` .

Expressies en operators

✓ Kennisclip voor deze inhoud

Expressies en operators

Een bewerking die een waarde kan opleveren is een **expressie**. Anders gezegd: een expressie in C# is iets dat je kan toekennen aan een variabele. Bijvoorbeeld `3+2`, `Console.ReadLine()`, of `7`. Als je kan zeggen dat een bewerking een duidelijk resultaat heeft, is het waarschijnlijk een expressie.

Expressie-resultaat toewijzen

Meestal zal je expressies schrijven waarin je bewerkingen op en met variabelen uitvoert. Vervolgens zal je het resultaat van die expressie willen bewaren voor verder gebruik in je code.

Voorbeeld van **expressie**-resultaat toekennen:

```
int temperatuursVerschil = temperatuurGisteren - temperatuurVandaag;
```

Hierbij zal de temperatuur uit de rechtse 2 variabelen worden uitgelezen, van elkaar wordt afgetrokken en vervolgens bewaard worden in `temperatuursVerschil`.

De voorgaande code kan ook langer geschreven worden als:

```
int tussenResultaat = temperatuurGisteren - temperatuurVandaag;  
int temperatuursVerschil = tussenResultaat;
```

Een ander voorbeeld van een **expressie**-resultaat toewijzen maar nu met literals (stel dat we `temperatuursVerschil` reeds hebben gedeclareerd eerder):

```
temperatuursVerschil = 21 - 25;
```

Uiteraard mag je ook combinaties van literals en variabelen gebruiken in je expressies:

```
int breedte = 15;  
int hoogte = 20 * breedte;
```

Operators

Operators in C# zijn de welgekende 'wiskundige bewerkingen' zoals optellen (+), aftrekken (-), vermenigvuldigen (*) en delen (/). Deze volgen de wiskundige regels van **volgorde van berekeningen**:

1. **Haakjes**
2. **Vermenigvuldigen, delen**: * (vermenigvuldigen), / (delen)
3. **Optellen en aftrekken**: + en -
(etc.)

Net zoals in de wiskunde kan je in C# met behulp van de haakjes verplichten het deel tussen de haakjes eerst te doen, ongeacht de andere operators en hun volgorde van berekeningen:

```
3+5*2 => zal 13 als resultaat geven  
(3+5)*2 => zal 16 geven
```

Je kan nu complexe berekeningen doen door literals, operators en variabelen samen te voegen. Bijvoorbeeld om te weten hoe zwaar je je op Mars zou voelen:

```
double gewichtOpAarde = 80.3; //kg  
double zwaartekrachtAarde = 9.81; //m/s2  
double zwaartekrachtMars = 3.711; //m/s2  
  
double gewichtOpMars = (gewichtOpAarde/zwaartekrachtAarde) * zwaartekrachtMars; //kg  
Console.WriteLine("Op Mars voelt het alsof je " + gewichtOpMars + " kg weegt.");
```

Automatische berekening van datatypes



De types die je in je berekeningen gebruikt bepalen ook het type van het resultaat. Als je bijvoorbeeld twee `int` variabelen of literals optelt zal het resultaat terug een `int` geven.

```
int result = 3+4;
```


Je kan echter geen kommagetallen aan `int` toewijzen. Als je dus twee `double` variabelen deelt is het resultaat terug een `double` en zal deze lijn een fout geven daar je probeert een `double` aan een `int` toe te wijzen:

```
int otherResult = 3.1/45.2;
```

Wat als je een `int` door een `int` deelt? Het resultaat is terug een `int`. Je bent gewoon alle informatie na de komma kwijt. Kijk maar:

```
int getal1 = 9;
int getal2 = 2;
int result = getal1 / getal2;
Console.WriteLine(result);
```

Er zal `4` op het scherm verschijnen! (niet `4.5` daar dat geen `int` is).

Wat als je datatypes mengt? Als je een berekening doet met een `int` en een `double` dan zal C# het meest algemene datatype kiezen. In dit geval een `double`. Volgende code zal dus werken:

```
double result = 3/5.6;
```

Volgende niet:

```
int result = 3/5.6;
```

Wil je dus het probleem oplossen om 9 te delen door 2 dan zal je minstens 1 van de 2 literals of variabelen door een `double` moeten omzetten. Het voorbeeld van hierboven herschrijven we dan naar:

```
int getal1 = 9;
double getal2 = 2.0;
double result = getal1/getal2;
Console.WriteLine(result);
```

En nu krijgen we wel `4.5`.

En complexer?

Het kan subtiel worden in grotere berekeningen.

Stel dat ik afsprek dat je van mij de helft van m'n salaris krijgt. Ik verdien (fictief) 10000 euro per maand. Ik gebruik volgende formule:

```
double helft = 10000.0 * (1/2);
```

Hoeveel krijg je van me? **0.0 euro!**

De volgorde van berekeningen zal eerst het gedeelte tussen de haakjes doen: 1 delen door 2 geeft 0, daar we een `int` door een `int` delen en dus terug een `int` als resultaat krijgen. Vervolgens zullen we deze 0 vermenigvuldigen met `10000.0`. We vermenigvuldigen weliswaar een `double` (het salaris) met een `int`, maar die `int` is reeds 0 en we krijgen dus `0.0` als resultaat.

Wil je het dus eerlijk spelen dan zal je de formule moeten aanpassen naar:

```
double helft = 10000.0 * (1.0/2); // of 1/2.0 of zonder haakjes
```

Nu krijgt het gedeelte tussen de haakjes een `double` als resultaat, namelijk `0.5` dat we dan kunnen vermenigvuldigen met het salaris om `5000.0` te krijgen.

Oefeningen

Al deze oefeningen maak je in een klasse `VariabelenEnDatatypes`. In de oefeningen van hoofdstuk 1 heb je gezien hoe je een nieuwe klasse maakt.

Oefening: H2-optellen

Leerdoelen

- gebruik van variabelen om input via `Console.ReadLine` op te slaan
- berekeningen met de opgeslagen data uitvoeren
- het resultaat dat werd opgeslagen in een variabele via `Console.WriteLine` te tonen

Functionele analyse

Een applicatie vraagt je twee getallen in te voeren. Na de invoer van het tweede getal worden beide getallen bij elkaar opgeteld. Het resultaat wordt uiteindelijk weergegeven.

Technische analyse

Noem de methode voor deze oefening `Optellen`.

1. De vraag wordt gesteld om een getal in te typen en daarna op enter/return te drukken.
2. Er wordt gevraagd een tweede getal in te typen en dan op enter/return te drukken.
3. De twee getallen worden opgeteld.
4. Het resultaat wordt weergegeven.

voorbeeldinteractie(s)

```
Wat is het eerste getal?  
> 1  
Wat is het tweede getal?  
> 4  
De som is 5.
```

Technische hulp

Programmaverloop

Lees de gebruikersinvoer van de console en sla dit op in een variabele voor wat het eerste getal betreft. Herhaal dit voor het tweede getal. Tel de twee getallen samen en bewaar deze in een derde variabele. Uiteindelijk geef je dan de inhoud van deze derde variabele weer in de console.

Let op: met `Console.ReadLine()` lees je **tekst** in, dus waarden die je kan toekennen aan variabelen van type `string`. Om een getal in te lezen, vervang je `Console.ReadLine()` door `Convert.ToInt32(Console.ReadLine())`. De werking hiervan zie je later.

Testscenario's

- Voer tekst in.
- Voer een getal met 100 cijfers in.
- Voer geen getal in.

Oefening: H2-verbruik-wagen

Leerdoelen

- gebruik van variabelen om input via `Console.ReadLine` op te slaan
- berekeningen met de opgeslagen data uitvoeren
- het resultaat dat werd opgeslagen in een variabele via `Console.WriteLine` te tonen

Functionele analyse

Een applicatie zal voor jou het gemiddelde verbruik van een wagen berekenen.

Hiervoor worden volgende vragen gesteld:

1. Hoeveel liter is er nog aanwezig in de benzinetank.
2. Hoeveel liter zit er nog in de benzinetank na de rit.
3. Ook de kilometerstand van bij de aanvang van de rit wordt gevraagd en ook deze nadat de rit werd uitgevoerd.

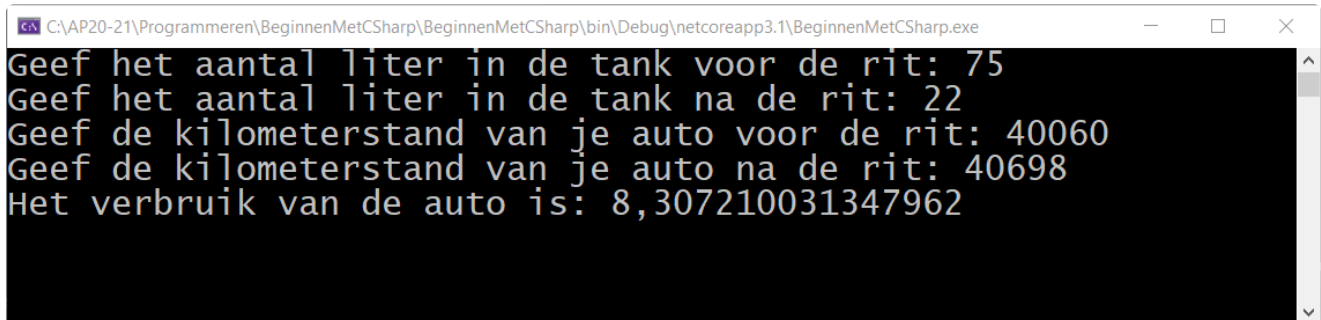
Op basis van deze parameters wordt het gemiddelde verbruik berekend en weergegeven.

Technische analyse

Noem de methode voor deze oefening `VerbruikWagen`.

1. De vraag wordt gesteld om het aantal liter, aanwezig in de benzinetank, op te geven.
2. Daarna wordt gevraagd om ook het aantal liter op te geven na de rit.
3. De kilometerstand van de aanvang van de rit wordt gevraagd.
4. Uiteindelijk ook de kilometerstand na het beëindigen van de rit wordt opgevraagd.

voorbeeldinteractie(s)

A screenshot of a Windows console window. The title bar shows the file path: C:\AP20-21\Programmeren\BeginnenMetCSharp\BeginnenMetCSharp\bin\Debug\netcoreapp3.1\BeginnenMetCSharp.exe. The console output is as follows:

```
Geef het aantal liter in de tank voor de rit: 75
Geef het aantal liter in de tank na de rit: 22
Geef de kilometerstand van je auto voor de rit: 40060
Geef de kilometerstand van je auto na de rit: 40698
Het verbruik van de auto is: 8,307210031347962
```

Technische hulp

Programmaverloop

Lees de gebruikersinvoer van de console en slaag dit op in variabelen.

Zorg ervoor dat je het juiste gegevenstype kiest voor de verschillende variabelen.

Nadien voer je de berekening uit om op basis van de ingevoerde gegevens het gemiddeld verbruik te berekenen $(100 * (\text{aantalLiterinTankVoorRit} - \text{aantalLiterinTankNaRit}) / (\text{kilometerstandNaRit} - \text{kilometerstandVoorRit}))$

Uiteindelijk geef je dan het resultaat weer in de console.

Testscenario's

- Voer tekst in.
- Voer een getal met 100 cijfers in.
- Voer geen getal in.

Oefening: H2-beetje-wiskunde

Leerdoelen

- expressies schrijven
- voorrang van operatoren
- effect van operaties naargelang datatype begrijpen

Functionele analyse

Je schrijft een programma dat de rol vervult van een rekenmachine. Het voert volgende berekeningen uit:

- $-1 + 4 * 6$
- $(35 + 5) * 7$
- $14 + -4 * 6 / 11$
- $2 + 15 / 6 * 1 - 7 * 2$

Technische analyse

Noem de methode voor deze oefening `BeetjewisKunde`.

voorbeeldinteractie(s)

```
23
280
12
-10
```

Technische hulp

Programmaverloop

Eerst wordt een resultaat berekend, daarna wordt het geprint.

Testscenario's

- Test uit met getallen van het type `int`.
- Test uit met getallen van het type `float`.

Ondersteunend materiaal

[Hier](#) vind je een tabel terug die uitlegt welke operaties voorrang hebben.

Oefening: H2-gemiddelde

Leerdoelen

- expressies schrijven
- voorrang van operatoren
- effect van operaties naargelang datatype begrijpen

Functionele analyse

Je schrijft een programma dat het gemiddelde van 18, 11 en 8 berekent, d.w.z. deze drie getallen optelt en de som deelt door drie.

Technische analyse

Noem de methode voor deze oefening `Gemiddelde`.

voorbeeldinteractie(s)

```
12
```

Technische hulp

Programmaverloop

Eerst wordt het resultaat berekend, daarna wordt het geprint.

Testscenario's

- Test uit met getallen van het type `int`.
- Test uit met getallen van het type `float`.

Oefening: H2-maaltafels

Leerdoelen

- de console leegmaken
- werken met wiskundige operatoren
- interactie met de gebruiker

Functionele analyse

Je schrijft een programma dat de tafel van vermenigvuldiging voor 411 geeft. Dit programma wacht steeds tot de gebruiker op ENTER duwt voor het het volgend resultaat toont. Verder maakt het steeds het scherm leeg voor het een nieuw resultaat toont. Zie "programmaverloop".

Technische analyse

Noem de methode voor deze oefening `Maaltafels`. Je kent nog geen lusstructuren, dus probeer deze niet te gebruiken, zelfs als je ze al ergens anders bent tegengekomen. Schrijf gewoon tien instructies.

voorbeeldinteractie(s)

```
1 * 411 is 411.
```

```
2 * 411 is 822.
```

(enzovoort)

```
10 * 411 is 4110.
```

Technische hulp

Programmaverloop

Voor elk resultaat wordt het scherm eerst leeggemaakt. Daarna pas wordt het resultaat getoond. Wanneer de gebruiker op ENTER duwt, wordt deze handeling herhaald voor het volgende resultaat (of eindigt het programma, na het tiende resultaat). Het scherm leegmaken doe je met `Console.Clear()`. Plaats 411 ook in een variabele.

Testscenario's

- Test uit zoals gegeven.
- Test uit voor 511. Je zou maar één teken in je code moeten aanpassen als je de instructies hebt gevolgd.

Oefening: H2-ruimte

Leerdoelen

- werken met kommagetallen

Functionele analyse

Je massa is overal dezelfde en wordt uitgedrukt in kilogram. Je gewicht daarentegen is afhankelijk van de zwaartekracht van de plek waar je bent en wordt uitgedrukt in Newton. Je hebt dus een ander gewicht op andere planeten. Zo is je gewicht veel groter op Jupiter dan op Mars, omdat Jupiter meer zwaartekracht uitoefent dan Mars. Schrijf een programma dat je gewicht op aarde omzet naar je gewicht op een ander hemellichaam. Je krijgt volgende omzettingstabel:

- Mercurius: 0.38 (een persoon van 100kg voelt zich alsof hij 38kg weegt)
- Venus: 0.91
- Aarde: 1.00 (een persoon van 100kg voelt zich alsof hij 100kg weegt)
- Mars: 0.38
- Jupiter: 2.34
- Saturnus: 1.06
- Uranus: 0.92
- Neptunus: 1.19
- Pluto: 0.06

Technische analyse

Noem de methode voor deze oefening `Ruimte`.

voorbeeldinteractie(s)

```
Op Mercurius voel je je alsof je 26.22kg weegt.
Op Venus voel je je alsof je 62.79kg weegt.
Op Aarde voel je je alsof je 69kg weegt.
Op Mars voel je je alsof je 26.22kg weegt.
Op Jupiter voel je je alsof je 161.46kg weegt.
Op Saturnus voel je je alsof je 73.14kg weegt.
Op Uranus voel je je alsof je 63.48kg weegt.
Op Neptunus voel je je alsof je 82.11kg weegt.
Op Pluto voel je je alsof je 4.14kg weegt.
```

Technische hulp

Programmaverloop

Plaats je gewicht in een variabele. Kies zelf een geschikt type.

Testscenario's

- Test uit voor je eigen gewicht.
- Test uit voor het gewicht van een persoon met een massa van 100kg.