

1.0.0

H4: Beslissingen

Beslissingen intro

✓ Inleiding

Nu we de elementaire zaken van C# en Visual Studio kennen is het tijd om onze programma's wat interessanter te maken. De ontwikkelde programma's tot nog toe waren steevast lineair van opbouw, ze werden lijn per lijn uitgevoerd zonder de mogelijkheid om de *flow* van het programma aan te passen. Het programma doorliep de lijnen braaf na elkaar en wanneer deze aan het einde kwam sloot het programma zich af.

Onze programma's waren met andere woorden niet meer dan een eenvoudige oplistingen van opdrachten. Je kan het vergelijken met een lijst die je vertelt over hoe je een brood moet kopen:

1. Neem geld uit spaarpot
2. Wandel naar de bakker om de hoek
3. Vraag om een brood
4. Krijg het brood
5. Betaal het geld aan de bakker
6. Keer huiswaarts
7. Smullen maar

Alhoewel dit algoritme redelijk duidelijk en goed zal werken, zal de realiteit echter zelden zo rechtlijnig zijn. Een beter algoritme (dat foutgevoeliger is én interactiever voor de eindgebruiker) zal afhankelijk van de omstandigheden (bakker gesloten, geen geld meer, etc.) mogelijke andere stappen ondernemen. **Het programma zal beslissingen nemen op basis van aanwezige informatie.**

In een typisch project, bijvoorbeeld een bestelsysteem, kunnen we beslissingen gebruiken voor tal van zaken:

- als we de voorraad van producten gaan bijhouden en we willen reageren wanneer de voorraad op is
- als we op bepaalde dagen van de week kortingen op bepaalde producten willen geven in plaats van de gebruiker deze zelf te laten kiezen
- als we spaarpunten willen introduceren en de gebruiker deze kan inwisselen

Enkelvoudige booleaanse expressies



Relationele operators

Om beslissingen te kunnen nemen in C# moeten we kunnen nagaan of een bepaalde uitspraak waar of niet waar is. Anders gezegd: we moeten stukjes code kunnen schrijven waarvan we achteraf kunnen zeggen of ze "waar" of "niet waar" zijn. Zo'n stukjes code noemen we **booleaanse expressies**. De meest typische booleaanse expressies schrijf je met de **relationele operatoren**. Deze stellen vergelijkingen tussen stukjes data voor. Gelukkig ken je deze al uit het lager onderwijs en moet je alleen de notatie hieronder leren:

C#-syntax	Betekenis
>	groter dan
<	kleiner dan
==	gelijk aan
!=	niet gelijk aan
<=	kleiner dan of gelijk aan
>=	groter dan of gelijk aan

⚠ "Gelijk aan" noteren we met twee symbolen, omdat één symbool gebruikt wordt voor een toekenning.

Deze operatoren leveren je altijd één van twee mogelijkheden als uitkomst: `true` of `false`. Je kan dit ook uittesten: `Console.WriteLine(4 < 7);` of `Console.WriteLine(4 < 2);` toont je het verwachte resultaat.

Er bestaan nog simpelere booleaanse expressies dan die met de relationele operatoren: `true` en `false` zelf zijn hun eigen resultaat en zijn dus technisch gesproken ook booleaanse expressies. Je mag dus bv. ook `true` schrijven in plaats van `4 > 2`.

If, else, else if

✓ Kennisclip

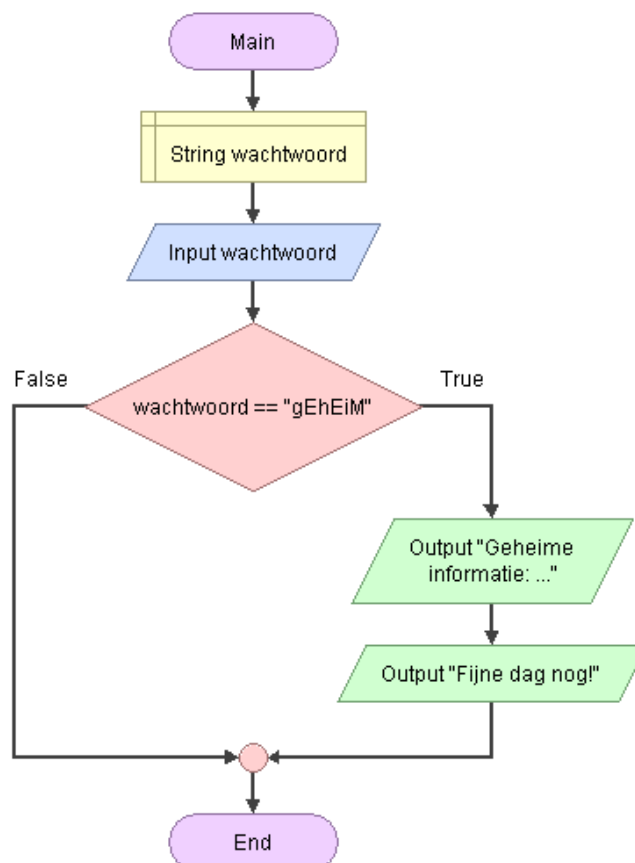
In dit deel zullen we bekijken hoe we ons programma dynamischer kunnen maken met behulp van het `if`-statement, al dan niet uitgebreid met `else` en `else if`.

If

De `if` is een van de elementairste constructies in een programmeertaal. Vrijwel elke programmeertaal bezit deze constructie. We zullen de werking ervan eerst wat algemener bekijken, zodat je het concept goed begrijpt. Daarna zullen we inzoomen op de syntax in C#.

Het basisidee is als volgt: een `if`-constructie bevat code die **enkel** uitvoert als een booleaanse expressie `true` oplevert. Met andere woorden: als een voorwaarde naar keuze **waar** is. We noemen dergelijke code **conditionele code**, want een conditie is een voorwaarde.

We zullen dit tonen met een Flowgorithm programma. Dit is een beperkte, maar heel visuele programmeertaal. In een Flowgorithm programma start je bij `Main` en volg je steeds de pijlen.



De gele box stelt een declaratie voor. Ook in Flowgorithm moet je variabelen declareren. Het blauw parallellogram: de gebruiker geeft iets in (de waarde van `wachtwoord`). De rode ruit bevat een booleaanse expressie. Dan is er een vertakking, met daarop de waarden `True` en `False`. **We volgen de pijl met daarop de uitkomst van de booleaanse expressie.** Als we dus het juiste wachtwoord intypen, krijgen we de geheime info, anders gebeurt er niets.


Het `if`-statement stemt overeen met alles tussen de rode ruit en het rode bolletje. Dus **als aan een bepaalde voorwaarde voldaan is, voeren we afgebakende code uit.**

Uit dit programma kan je dan ook volgende C#-code afleiden:

```
public static void Main(string[] args)
{
    string wachtwoord;

    wachtwoord = Console.ReadLine()
    if (wachtwoord == "gEhEiM")
    {
        Console.WriteLine("Geheime informatie: ...");
        Console.WriteLine("Fijne dag nog!");
    }
}
```

In de gegenereerde code stemt de rode ruit dus overeen met de haakjes meteen na `if` en stemt de tak `True` overeen met de accolades.

 Flowgorithm is vrij te downloaden, dus als je moeite hebt met deze concepten, wordt aangeraden hier wat mee te experimenteren.

Veelgemaakte if-fouten

Er zijn enkele veelgemaakte fouten waar je op moet letten:

Appelen en peren vergelijken

De types in je booleaanse expressie moeten steeds vergelijkbaar zijn. Volgende code is dus fout: `if ("4" > 3)` daar we hier een `string` met een `int` vergelijken.

Accolades vergeten

Als je geen accolades schrijft, verandert de werking van `if`. Het gevolg zal zijn dat enkel het eerste statement na de `if` zal uitgevoerd worden indien `true`. Gebruiken we de `if` van daarnet maar zonder accolades dan zal het tweede statement altijd uitgevoerd worden ongeacht de `if`:

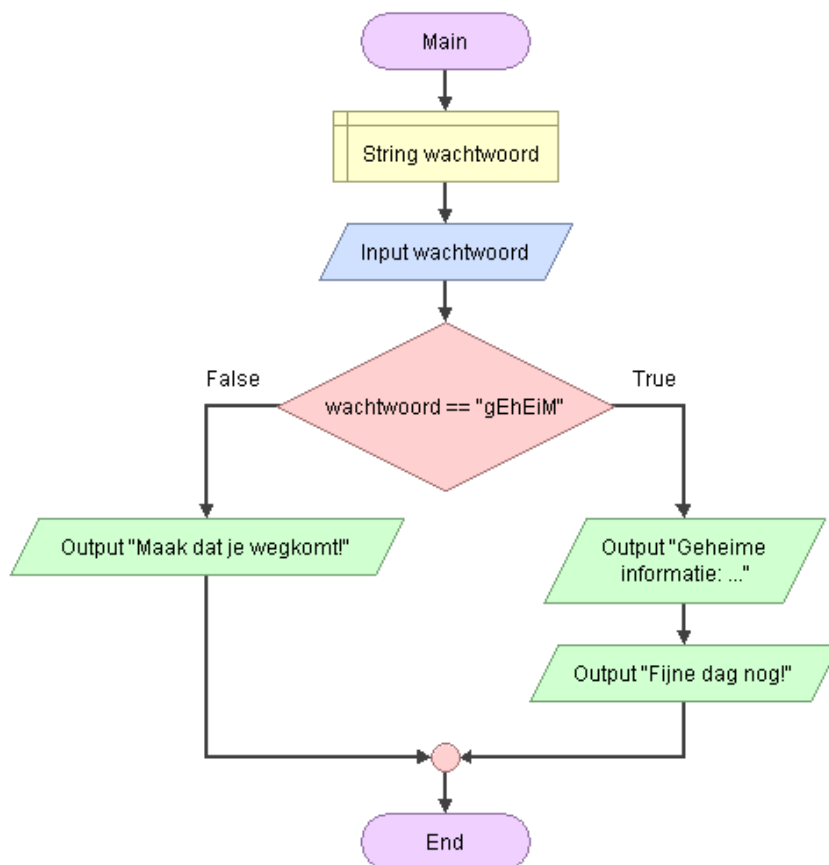
```
if (wachtwoord == "gEhEiM")  
    Console.WriteLine ("Geheime informatie: ...");  
    Console.WriteLine ("Fijne dag nog!");
```

i Voor ons is het simpel: we schrijven **if** **altijd** met accolades.

If/else

Het eerdere voorbeeld toont dat we soms actie willen ondernemen als aan een voorwaarde voldaan is, maar heel vaak willen we een **andere** actie ondernemen als aan diezelfde voorwaarde **niet** voldaan is. In dat geval maken we gebruik van de `if ... else ...`.

We gaan terug naar onze login. Een fout wachtwoord ingeven heeft bepaalde gevolgen. Dit kan bijvoorbeeld een alarm doen afgaan, omdat indringers soms foute wachtwoorden ingeven tot ze toevallig het juiste vinden. We stellen dit hier voor door een actie toe te voegen als het wachtwoord **niet** klopt.



De overeenkomstige C# code:

```

public static void Main(string[] args)
{
    string wachtwoord;

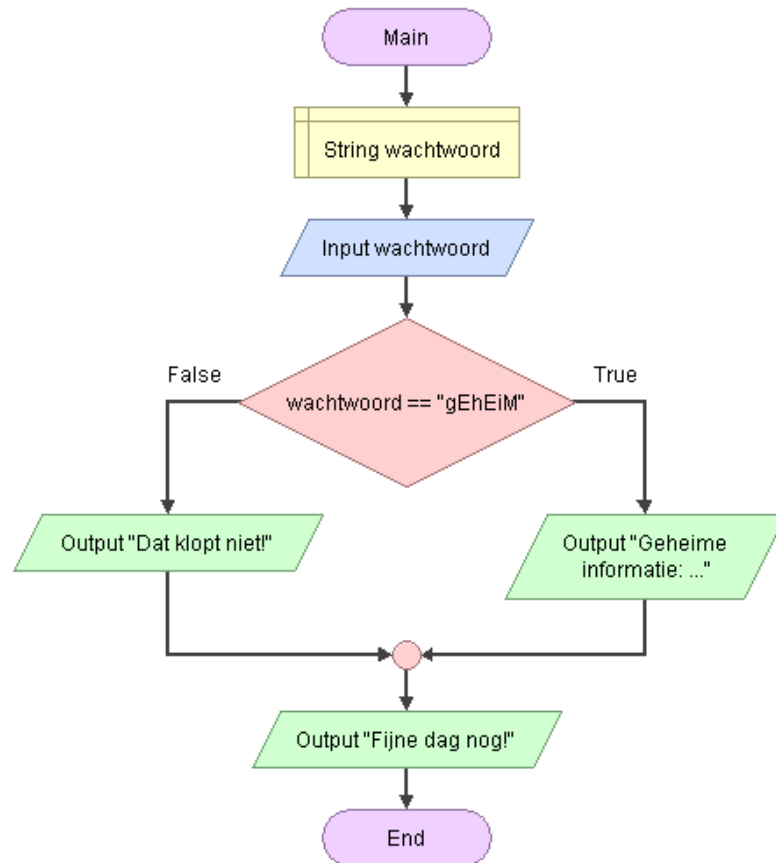
    wachtwoord = Console.ReadLine()
    if (wachtwoord == "gEhEiM")
    {
        Console.WriteLine("Geheime informatie: ...");
        Console.WriteLine("Fijne dag nog!");
    }
    else
    {
        Console.WriteLine("Maak dat je wegkomt!");
    }
}

```

In bovenstaande code stemt de rode ruit dus nog steeds overeen met de haakjes meteen na `if`, stemt de tak `True` overeen met de accolades vlak na de ronde haakjes en stemt `else { ... }` overeen met de tak `False`. Anders gezegd: **als aan een bepaalde voorwaarde voldaan is, voeren we het eerste blok afgebakende code uit, anders voeren we het tweede blok afgebakende code uit.**

Code voor beide gevallen

Als we iets altijd willen doen, hoort dat niet in een vertakking. Dan zetten we het in de flowchart na het rode bolletje. In ons programma zetten we het dan na de volledige `if ... else ...`:



Code die in beide gevallen moet uitvoeren, zetten we achter het rode bolletje.

De gegenereerde code is dan:

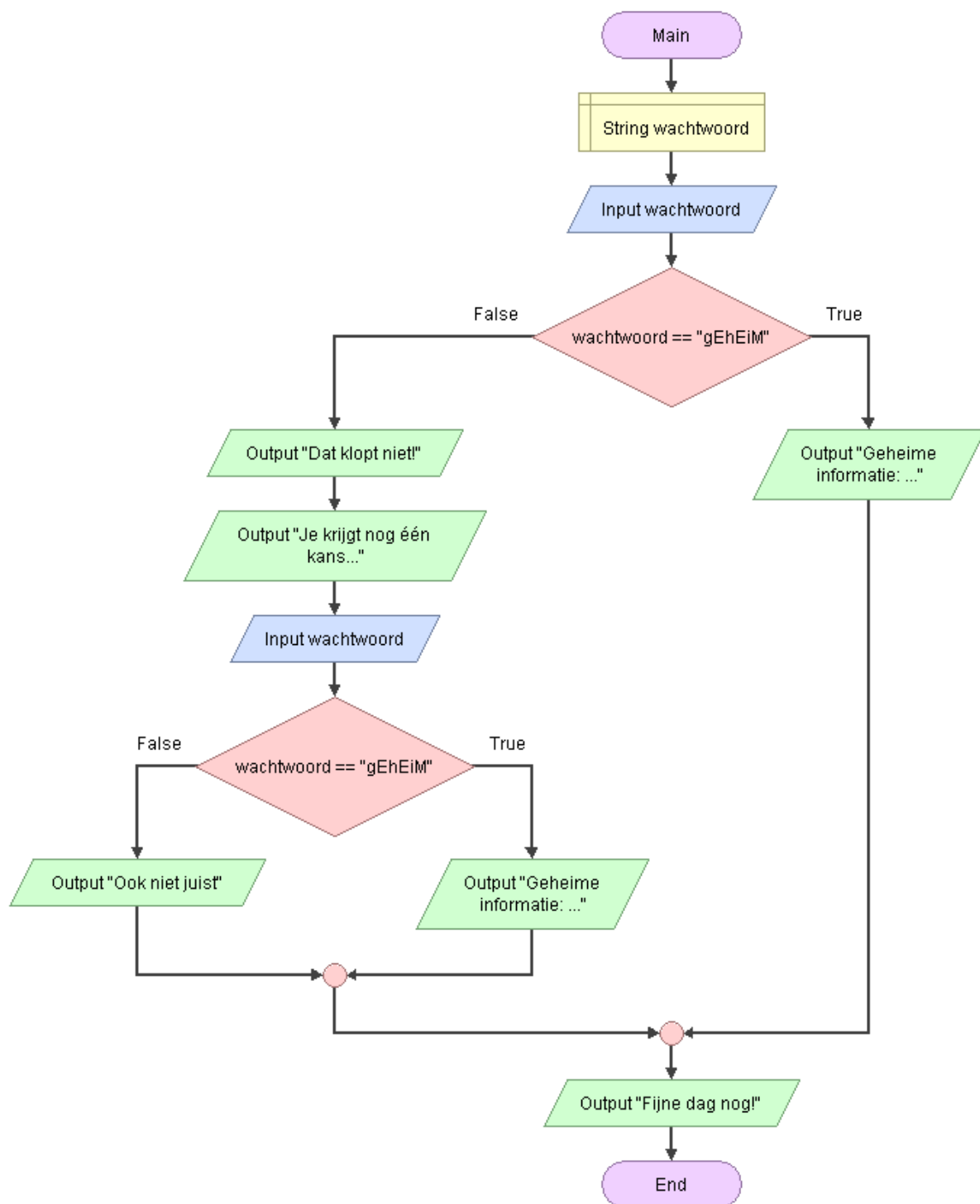
```

public static void Main(string[] args)
{
    string wachtwoord;

    wachtwoord = Console.ReadLine()
    if (wachtwoord == "gEhEiM")
    {
        Console.WriteLine("Geheime informatie: ...");
    }
    else
    {
        Console.WriteLine("Dat klopt niet!");
    }
    Console.WriteLine("Fijne dag nog!");
}
  
```

Nesting

Het is perfect mogelijk om bepaalde controles enkel te doen als eerdere controles wel of niet gelukt zijn. Dit kan er zo uitzien:



In de gegenereerde code leidt dit tot **geneste conditionele code**: een `if` binnenin een grotere `if` of `else`. In dikt geval gaat het om een `if` in een `else`. Je mag zo diep nesten als je maar wil. Er is geen technische limiet, maar je code zal wel onleesbaar worden als je overdrijft.

```

public static void Main(string[] args)
{
    string wachtwoord;

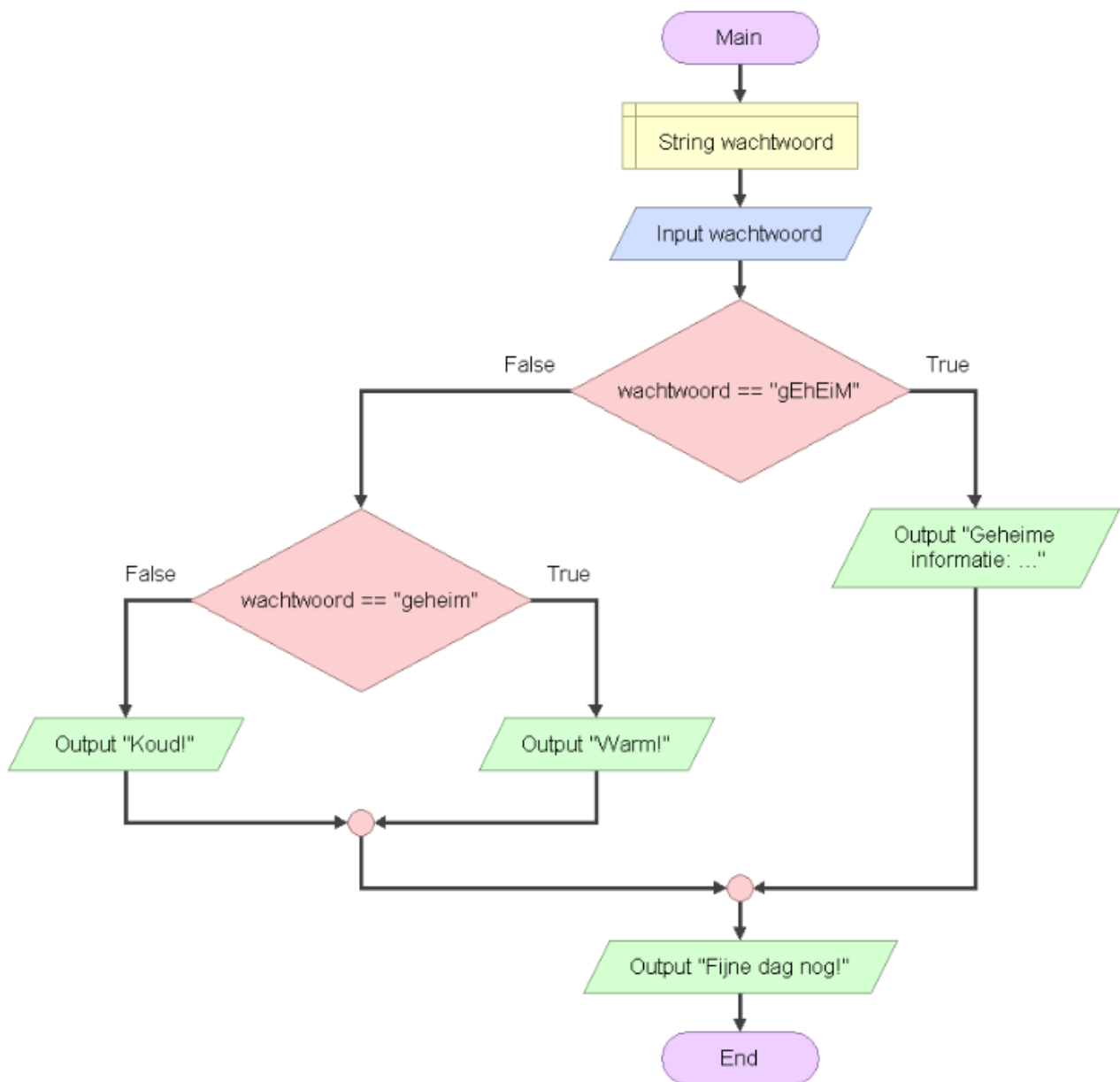
    wachtwoord = Console.ReadLine()
    if (wachtwoord == "gEhEiM")
    {
        Console.WriteLine("Geheime informatie: ...");
    }
    else
    {
        Console.WriteLine("Dat klopt niet!");
        Console.WriteLine("Je krijgt nog één kans...");
        wachtwoord = Console.ReadLine()
        if (wachtwoord == "gEhEiM")
        {
            Console.WriteLine("Geheime informatie: ...");
        }
        else
        {
            Console.WriteLine("Ook niet juist");
        }
    }
    Console.WriteLine("Fijne dag nog!");
}

```

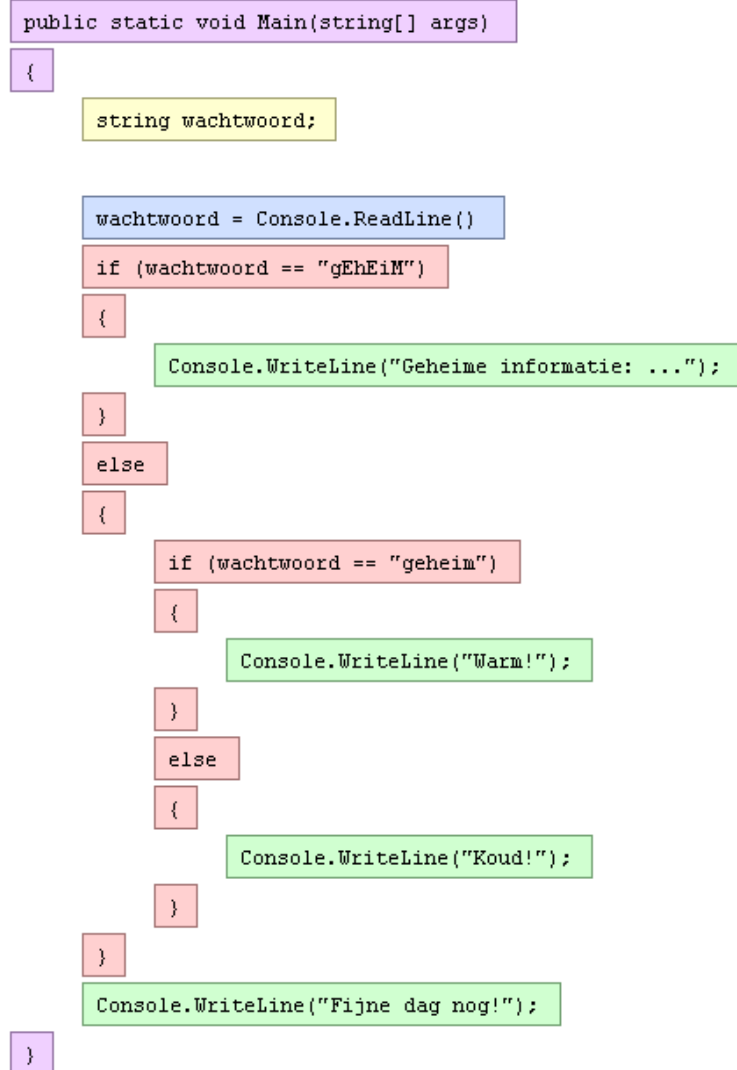
else if

Nesten van conditionele code levert soms code op die moeilijk te ontrafelen is. Het gebeurt vaak dat we een `else` nodig hebben met meteen daarin terug een `if`, om zo verschillende gevallen af te handelen.

We kunnen een programma maken dat dit demonstreert: Er is niet aan de hoofdvoorwaarde voldaan, maar er is wel aan een andere voorwaarde voldaan:



De gegenereerde code voor dit fragment is technisch juist:



Maar Flowgorithm is geen menselijke programmeur. Een menselijke programmeur zou volgende voorstelling gebruiken, die hetzelfde doet, maar makkelijker leesbaar is:

```
public static void Main(string[] args) {
    string wachtwoord;
    wachtwoord = Console.ReadLine();
    if (wachtwoord == "gEhEiM") {
        Console.WriteLine("Geheime informatie: ...");
    }
    else if (wachtwoord == "geheim") {
        Console.WriteLine("Warm!");
    }
    else {
        Console.WriteLine("Koud!");
    }
    Console.WriteLine("Fijne dag nog!");
}
```

Samengestelde booleaanse expressies

✓ Kennisclip 1

✓ Kennisclip 2

Logische operatoren

De logische EN, OF en NIET-operatoren die je misschien kent uit de lessen technologie of elektronica kan je ook gebruiken in C#:

C#-syntax	Betekenis
&&	en-operator
	of-operator
!	niet-operator

Je kan de &&-operator tussen twee booleaanse expressies zetten om een nieuwe, grote expressie te schrijven die afhangt van de kleinere expressies aan beide kanten. Idem voor de ||-operator. Je kan de niet-operator voor een booleaanse expressie zetten om het resultaat hiervan om te draaien. Bijvoorbeeld:

```
!(0==2) //zal true geven
```

Hieronder krijg je nog eens het overzicht van de werking van de AND en OR operatoren:

AND	niet waar	waar
niet waar	niet waar	niet waar
waar	niet waar	waar

OR	niet waar	waar
niet waar	niet waar	waar
waar	waar	waar

Test jezelf

Wat zal de uitkomst zijn van volgende expressies? (tip: het zal steeds `true` of `false` zijn, niets anders)

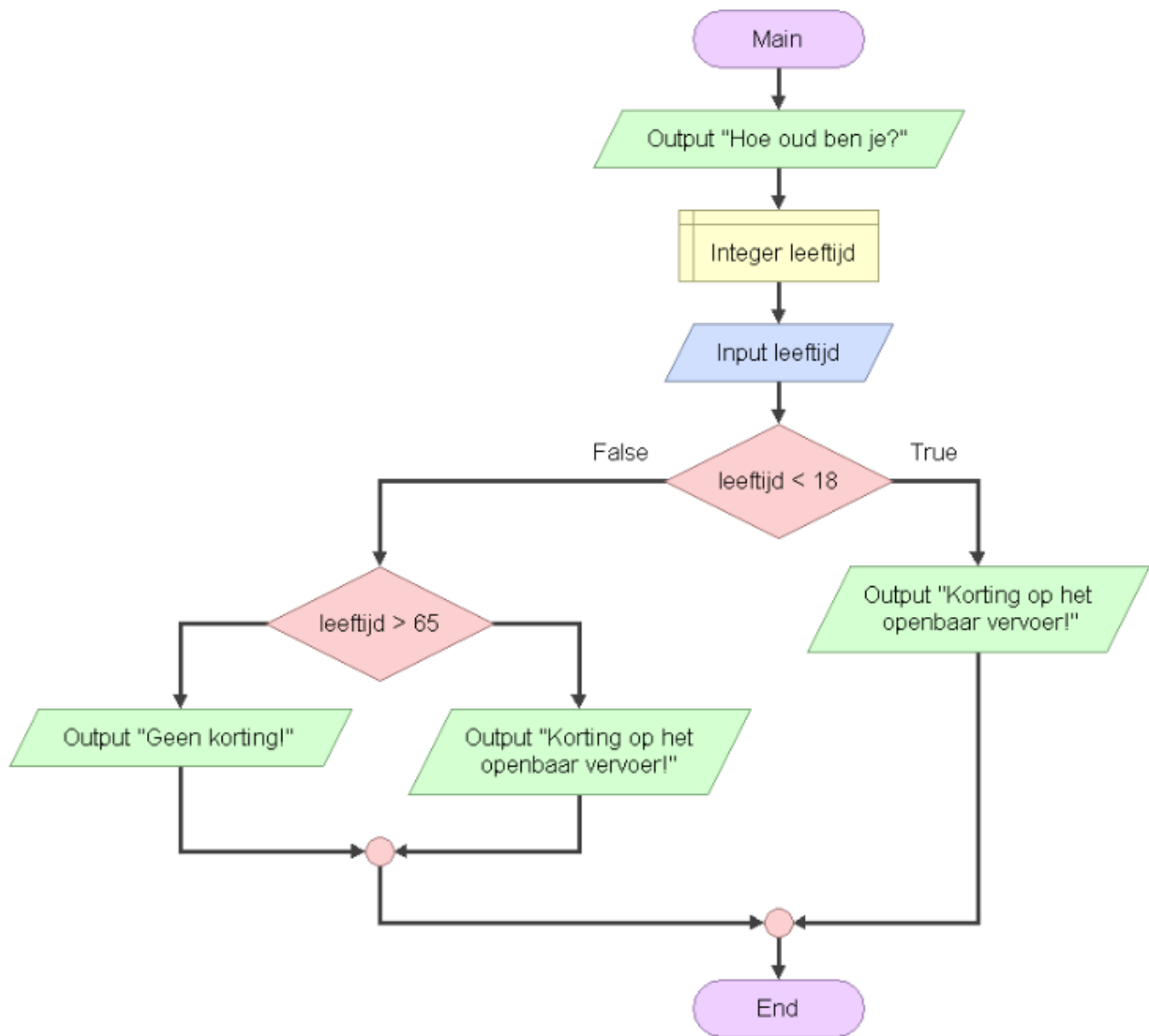
- `3>2`
- `4!=4`
- `4<5 && 4<3`
- `"a"=="a" || 4>=3`
- `(3==3 && 2<1) || 5!=4`
- `!(4<=3)`
- `true || false`
- `!true && false`

Booleaanse expressies algemeen

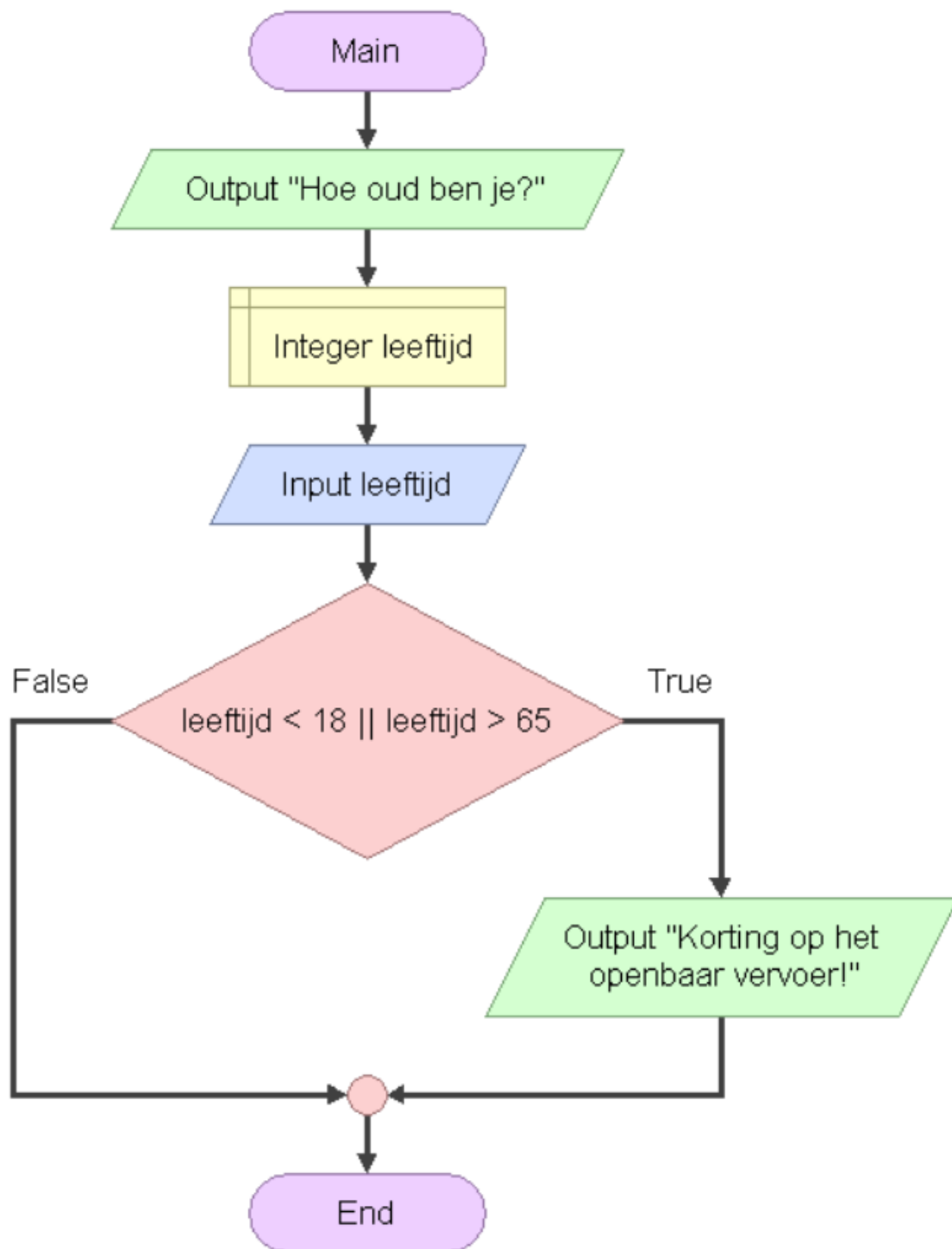
Je bent niet beperkt tot het gebruik van expressies met één relationele operator in de voorwaarde van een `if`. Je mag om het even welk type booleaanse expressie gebruiken. Anders gezegd: als het gedeelte tussen haakjes uiteindelijk maar `true` of `false` oplevert, is het goed. We kunnen dus ook gebruik maken van de logische operatoren `&&`, `||`, `!` in onze voorwaarden.

Goed gebruik van booleaanse expressies kan onze programma's drastisch vereenvoudigen. We tonen dit met een voorbeeld.

Volgend programma bepaalt of je korting krijgt op het openbaar vervoer. Volg zelf de flowchart om de logica te snappen:



Maar we kunnen dit eenvoudiger zo schrijven:



Deze code zegt dat je korting krijgt als je jonger bent dan 18 **of** als je ouder bent dan 65. Ze doet hetzelfde als de eerste versie, maar het diagram is eenvoudiger en de gegenereerde code is dat ook. Ook via de andere operatoren `&&` en `!` kan je gelijkaardige vereenvoudigingen verkrijgen.

Scope van variabelen

✓ Kennisclip voor deze inhoud

Scope van variabelen

De locatie waar je een variabele aanmaakt bepaalt de **scope**, oftewel de gegarandeerde levensduur, van de naam van de variabele. Ruwweg: als je binnen hetzelfde stel accolades werkt waarin een variabele is gedeclareerd, dan is die naam **in scope** en kan je hem gebruiken. Indien je de variabele dus buiten die accolades nodig hebt, dan heb je een probleem: de variabele is enkel bereikbaar binnen de accolades vanaf het punt in de code waarin het werd aangemaakt.

Zeker wanneer je begint met `if`, loops, methoden, etc. zal de scope belangrijk zijn: deze code-constructies gebruiken steeds accolades om codeblocks aan te tonen. Een variabele die je dus binnen een if-blok aanmaakt zal enkel binnen dit blok bestaan, niet erbuiten.

```
if( something == true)
{
    int getal = 0 ; //Start scope getal
    getal = 6;
} // einde scope getal

Console.WriteLine(getal); // zal niet werken daar de scope van getal al gedaan was
```

Wil je dus `getal` ook nog buiten de `if` gebruiken zal je je code moeten herschrijven zodat `getal` voor de `if` wordt aangemaakt:

```
int getal = 0 ; //Start scope getal
if( something == true)
{
    getal = 6;
}

Console.WriteLine(getal);
```

Variabelen met zelfde naam

Zolang je in de scope van een variabele bent kan je geen nieuwe variabele met dezelfde naam aanmaken:

Volgende code is dus niet toegestaan:

```
{  
    int getal = 0;  
    {  
        int getal = 5; //Deze lijn is niet toegestaan  
    }  
}
```

Je krijgt de error: A local variable named 'getal' cannot be declared in this scope because it would give a different meaning to 'getal', which is already used in a 'parent or current' scope to denote something else

Enkel de tweede variabele een andere naam geven is toegestaan in het voorgaande geval.

Dit is wel geldig, daar de scope van de eerste variabele afgesloten wordt door de accolades:

```
{  
    int getal = 0 ;  
    //....  
}  
//Verder in code  
{  
    int getal = 5;  
}
```

Oefeningen

Al deze oefeningen maak je in een klasse `Beslissingen`

Oefening: H4-Schoenenverkoper

Leerdoelen

- flowchart omzetten naar conditionele code
- Eenvoudige conditie
- Uitbreiding: samengestelde booleaanse expressie

Functionele analyse

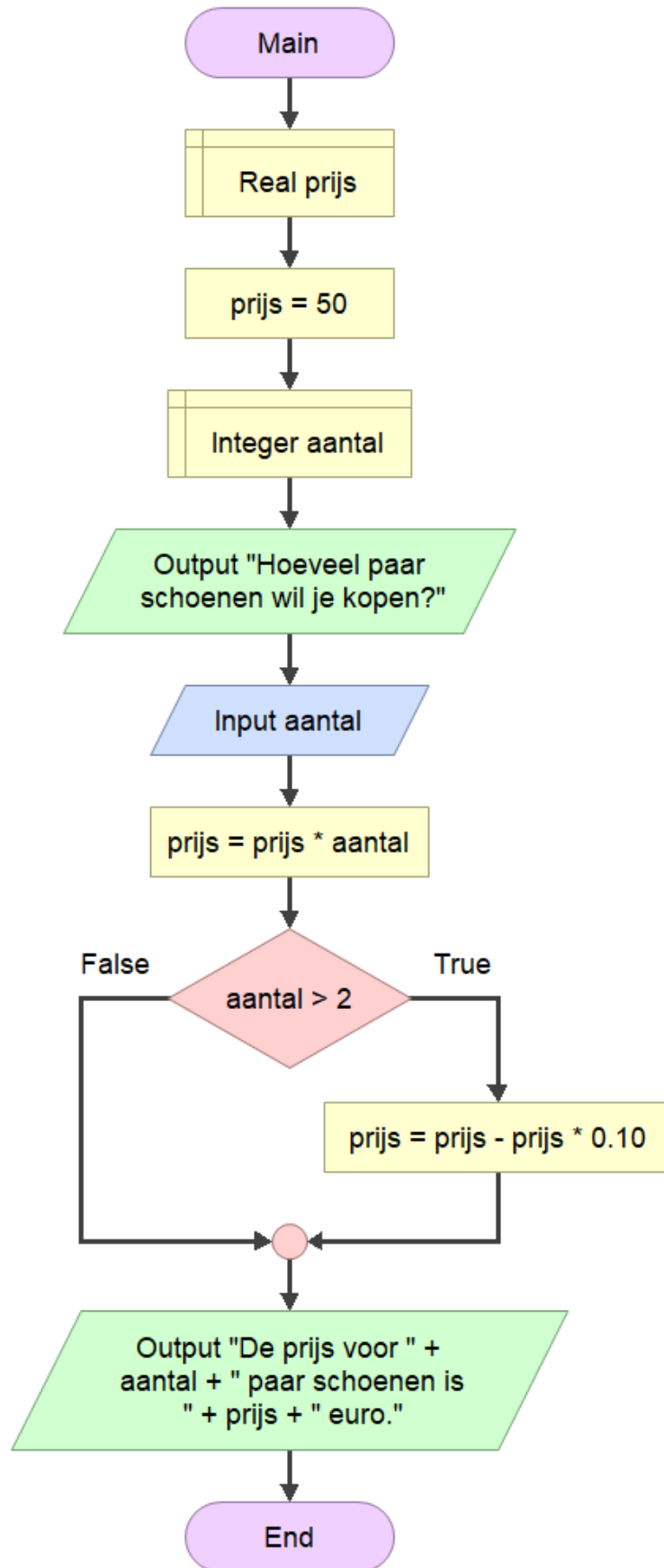
Basis: Maak een programma dat aan de gebruiker vraagt hoeveel paar schoenen hij/zij wenst te kopen. Ieder paar schoenen kost normaal 50 euro. Indien de gebruiker 2 paar of meer koopt, is er 10% korting. Toon aan de gebruiker de totale prijs.

Uitbreiding: Breid in een tweede stap je programma uit zodat de korting enkel geldt als de gebruiker een klantenkaart heeft.

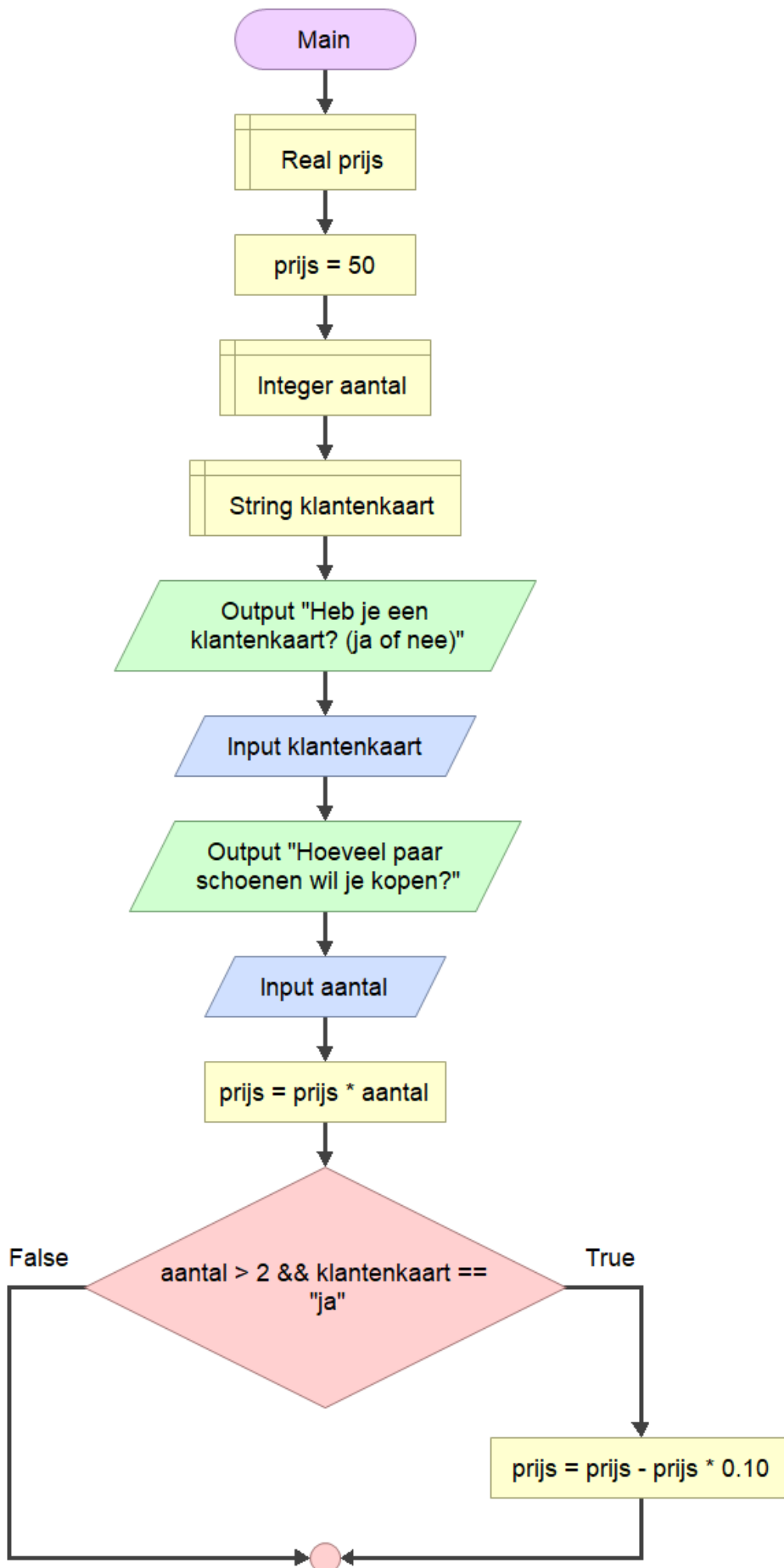
Technische analyse

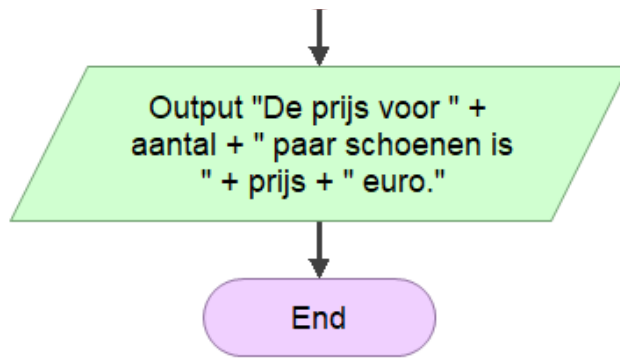
Maak een methode met de naam `SchoenenVerkoper`. Maak gebruik van een `if`. Zet volgende flowchart om in code. Een `Real` in Flowgorithm stemt overeen met een `double` in C#.

Basis:



Uitbreiding:





Voorbeeldinteractie(s)

Basis:

```
Hoeveel paar schoenen wil je kopen?  
1  
De prijs voor 1 paar schoenen is 50 euro.
```

```
Hoeveel paar schoenen wil je kopen?  
4  
De prijs voor 4 paar schoenen is 180 euro.
```

Uitbreiding:

```
Heb je een klantenkaart? (ja of nee)  
nee  
Hoeveel paar schoenen wil je kopen?  
3  
De prijs voor 3 paar schoenen is 150 euro.
```

Oefening: H4-EvenOneven

Leerdoelen

- flowchart omzetten naar conditionele code
- tweevoudige conditie
- gebruik modulo

Functionele analyse

Maak een programma dat aan de gebruiker een geheel getal vraagt. Het programma geeft terug of het ingegeven getal even of oneven is.

Technische analyse

Maak een methode met de naam `EvenOneven`.

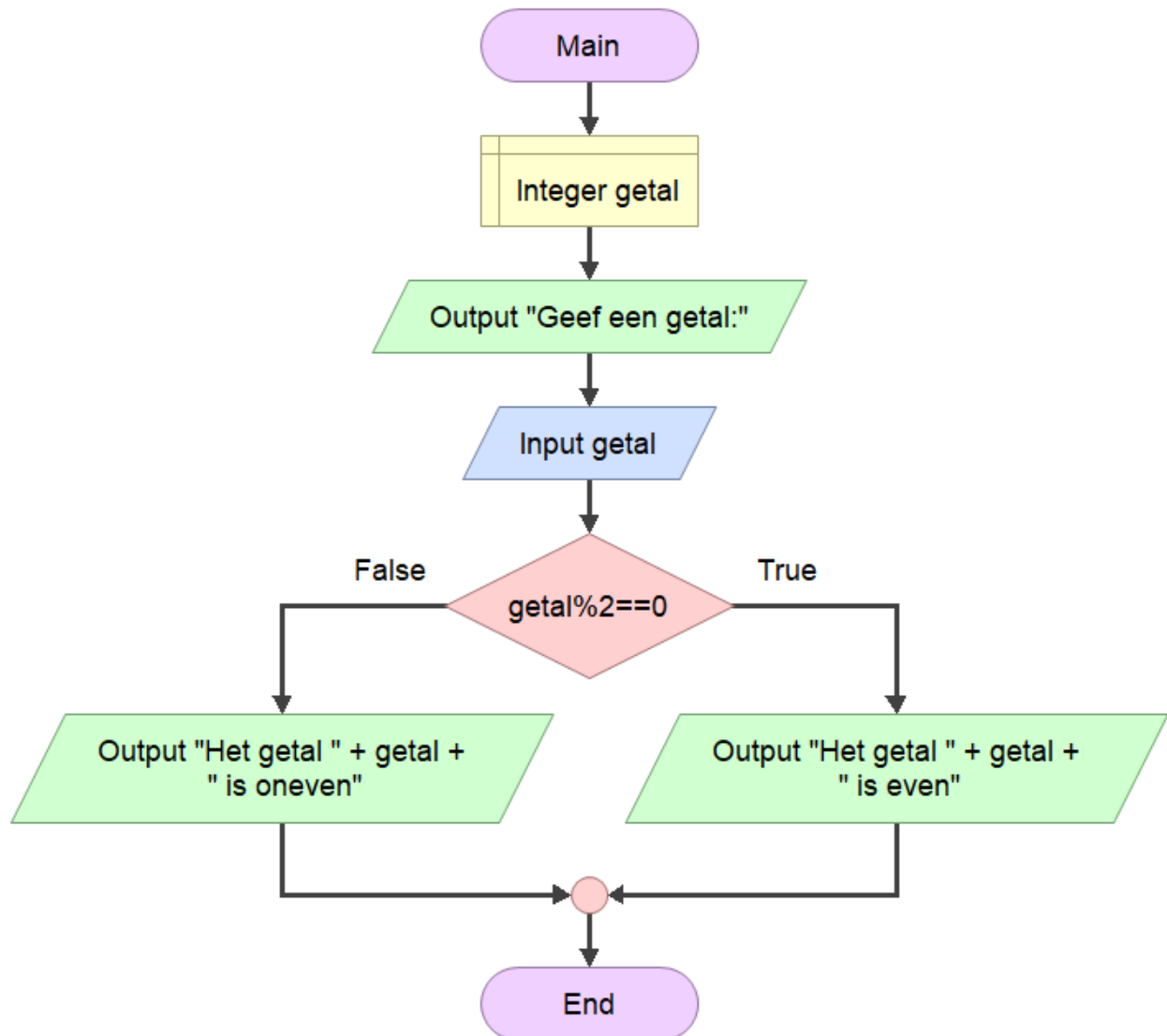
Een getal is even als de rest van de deling door 2 nul is. Hiervoor kan je de modulo operator gebruiken.
Het teken voor de modulo is het % -teken.

Voorbeelden:

$7\%2$ geeft 1 \Rightarrow 7 is oneven

$8\%2$ geeft 0 \Rightarrow 8 is even

Zet volgende flowchart om in code:



Voorbeeldinteractie(s)

```
Geef een getal:
5
Het getal 5 is oneven
```

```
Geef een getal:
18
Het getal 18 is even
```


Oefening: H4-PositiefNegatiefNul

Leerdoelen

- flowchart omzetten naar conditionele code
- meervoudige conditie

Functionele analyse

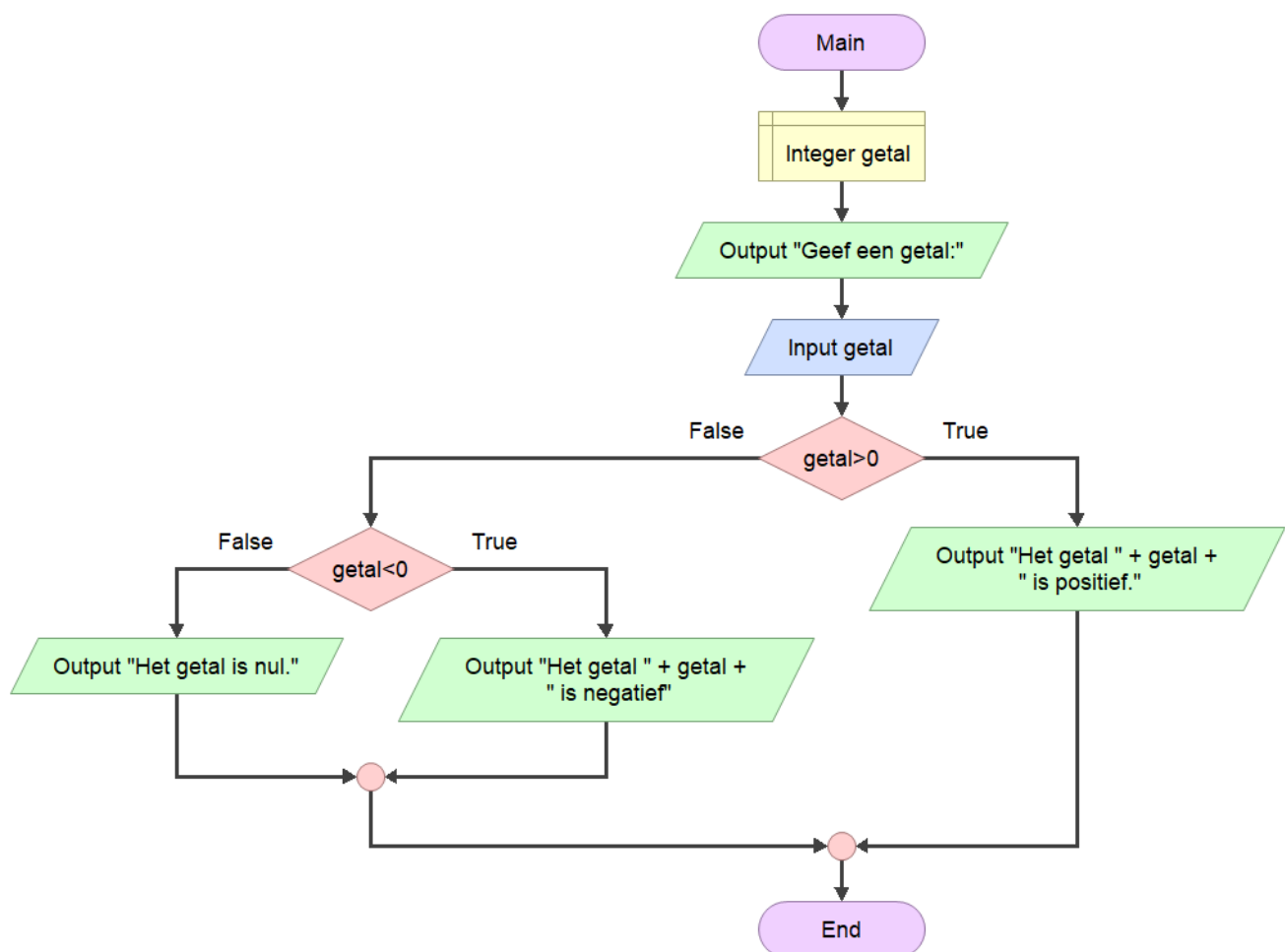
Maak een programma dat aan de gebruiker een geheel getal vraagt. Het programma geeft terug of het ingegeven getal positief, negatief of 0 is.

Technische analyse

Maak een methode met de naam `PositiefNegatiefNul`.

Maak gebruik van een `if - else if - else`.

Zet volgende flowchart om in code:



Voorbeeldinteractie(s)

```
Geef een getal:  
17  
Het getal 17 is positief.
```

```
Geef een getal:  
-5  
Het getal -5 is negatief
```

```
Geef een getal:  
0  
Het getal is nul.
```

Oefening: H4-BMIBerekenaar

Leerdoelen

- meervoudige conditie
- gebruik van `else if`

Functionele analyse

Maak een programma om de BMI van de gebruiker te berekenen. ([Meer info over BMI](#)) De BMI wordt berekend aan de hand van de lengte en het gewicht van de persoon.

De formule is: $BMI = \text{gewicht} / \text{lengte}^2$.

Je toont niet enkel de BMI maar ook een beoordeling over de BMI:

- BMI lager dan 18,5 => ondergewicht
- BMI vanaf 18,5 maar lager dan 25 => normaal gewicht
- BMI vanaf 25 maar lager dan 30 => overgewicht
- BMI vanaf 30 maar lager dan 40 => zwaarlijvig
- BMI hoger dan 40 => ernstige obesitas

Technische analyse

Maak een methode met de naam `BMIBerekenaar`.

Maak gebruik van een `if` – `else if` – `else if` ...

Voorbeeldinteractie

```
Wat is je gewicht?  
75  
Wat is je lengte in meter?  
1,79  
Je hebt een BMI van 23,4. Je hebt een normaal gewicht.
```

Oefening: H4-GrootsteVanDrie

Leerdoelen

- meervoudige conditie
- samengestelde booleaanse expressie
- gebruik van `else if`

Functionele analyse

Maak een programma om van 3 ingegeven getallen, het grootste te bepalen.

Technische analyse

Maak een methode met de naam `GrootsteVanDrie`.

Maak gebruik van een `if` – `else if` - `else`

Voorbeeldinteractie

```
Geef het eerste getal in: 98  
Geef het tweede getal in: 13  
Geef het derde getal in: 77  
Het eerste getal (98) is het grootste.
```

Oefening: H4-Examens

Leerdoelen

- conditie
- samengestelde booleaanse expressie

Functionele analyse

Maak een programma waarmee je aan de gebruiker het resultaat van 3 examens opvraagt. De opgevraagde resultaten zijn de behaalde punten op 100. Om te slagen moet de student een gemiddelde van meer dan 50% hebben én maximaal 1 onvoldoende.

Technische analyse

Maak een methode met de naam `Examens`.

Maak gebruik van een `if – else`.

Voorbeeldinteractie

```
Geef het resultaat in van het eerste examen: 60
Geef het resultaat in van het tweede examen: 35
Geef het resultaat in van het derde examen: 52
Je bent niet geslaagd!
```

Oefening: H4-Wet van Ohm

Leerdoelen

- conditionele berekeningen

Functionele analyse

De Wet van Ohm houdt in dat een elektrische stroom (voorgesteld als I) gelijk is aan een spanningsverschil (U) gedeeld door een weerstand (R), dus $I = U / R$.

Vraag aan de gebruiker wat hij wenst te berekenen: Spanning, Weerstand of Stroomsterkte. Vraag vervolgens de twee andere waarden. Als dus de gebruiker "Spanning" kiest, vraag je aan de gebruiker de waarden van de stroomsterkte en de weerstand. Bereken m.b.v. de Wet van Ohm de gewenste waarde en toon aan de gebruiker.

Technische analyse

Maak een nieuwe methode genaamd `WetVanOhm`.

Denk eraan dat de gegeven formule wiskundig gedefinieerd is. Houd rekening met het feit dat deze drie maten uitgedrukt kunnen worden in kommagetallen.

Voorbeeldinteractie(s)

```
Wat wil je berekenen:spanning, weerstand of stroomsterkte?  
spanning  
Wat is de weerstand? 30  
Wat is de stroomsterkte? 5  
De spanning is 150,00.
```

```
Wat wil je berekenen:spanning, weerstand of stroomsterkte?  
weerstand  
Wat is de spanning? 220  
Wat is de stroomsterkte? 10  
De weerstand is 22,00.
```

```
Wat wil je berekenen:spanning, weerstand of stroomsterkte?  
stroomsterkte  
Wat is de weerstand? 20  
Wat is de spanning? 30  
De stroomsterkte is 1,50.
```

Keuzemenu's maken

Leerdoelen:

- Conditionele functionaliteit

Functionele analyse:

We willen dat de gebruiker een menu ter beschikking heeft om eerst te kiezen uit welk hoofdstuk er een oefening moet gekozen worden en vervolgens welke oefening er moet uitgevoerd worden.

Technische analyse

Maak in je klasse Program een hoofdmenu. Deze methode laat de gebruiker kiezen uit één van de hoofdstukken. Na de keuze wordt het scherm leeggemaakt en dan wordt de methode `Keuzemenu` van de juiste klasse opgeroepen.

Elke klasse die je tot hiertoe in dit project gemaakt hebt, voorzie je dus van een methode `Keuzemenu`, zodat de gebruiker kan kiezen uit de oefeningen (=methodes) binnen het gekozen hoofdstuk.

Voorbeeldinteractie

Welke oefening kies je?

- 1 - Hoofdstuk 1 - Werken met Visual Studio Code
- 2 - Hoofdstuk 2 - Variabelen en datatypes
- 3 - Hoofdstuk 3 - Strings en hun methoden
- 4 - Hoofdstuk 4 - Beslissingen

5

Ongeldige keuze

Welke oefening kies je?

- 1 - Hoofdstuk 1 - Werken met Visual Studio Code
- 2 - Hoofdstuk 2 - Variabelen en datatypes
- 3 - Hoofdstuk 3 - Strings en hun methoden
- 4 - Hoofdstuk 4 - Beslissingen

1