


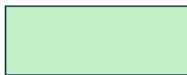

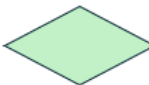

WAARDEVOLLE KENNIS VOOR TOETS

CONTENTS

FLOWCHARTS.....	2
Vormgeving van een flowchart:	2
Voorbeeld docent:.....	2
Input/Output.....	3
Variabelen en Expressies.....	4
Branching / Conditionals	5
Herhalingsstructuren	6
Codestyle	7
Methodes	8

FLOWCHARTS

VORMGEVING VAN EEN FLOWCHART:

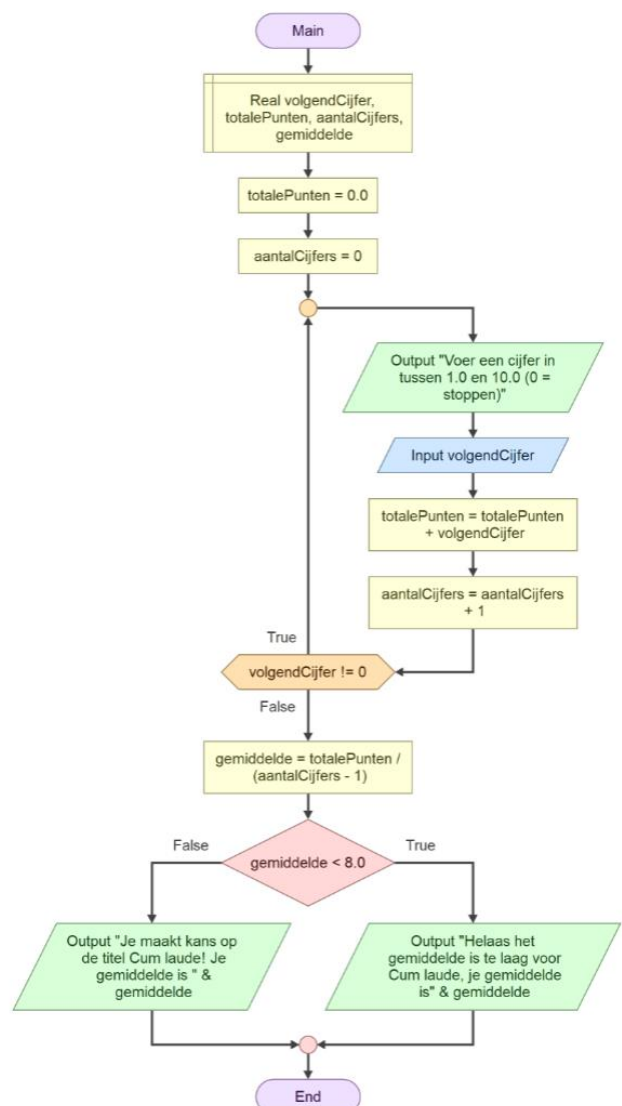
Symbol	Name	Function
	Oval	Represents the start or end of a process
	Rectangle	Denotes a process or operation step
	Arrow	Indicates the flow between steps
	Diamond	Signifies a point requiring a yes/no
	Parallelogram	Used for input or output operations

VOORBEELD DOCENT:

Maak een algoritme waarbij een student één of meerdere cijfers in kan voeren. De student kan dit invoeren van cijfers herhalen totdat het cijfer 0 ingevoerd wordt.

Daarna wordt het gemiddelde berekend.

Bepaal aan de hand van het gemiddelde of de student in aanmerking komt voor Cum laude (gemiddeld een 8.0 of hoger).



INPUT/OUTPUT

- Als een programma wacht op input van de gebruiker, wordt alle code daarna niet uitgevoerd totdat er een input is opgenomen.
- Een input wordt altijd gelezen als een string.
- Instellen van kleuren in console:
 - `Console.ForegroundColor = ConsoleColor.Red;`
 - `Console.BackgroundColor = Color.Blue;`
 - `Console.ResetColor();`

VARIABELEN EN EXPRESSIES

- **Variabele** = Een benoemde geheugenlocatie in een computerprogramma.
- Een variabele zonder initiële waarde zal zijn standaardwaarde gebruiken.
- Een boolean kan alleen “true” of “false” zijn. **Niet:** 1, 0, True, False.
 - !true en !false werken ook, dan worden ze dus omgedraaid.
- Komma getallen zoals 10.0, worden door c# altijd gezien als een double. Dit betekent dat **float x = 10.0;** niet gaat werken omdat een float geen double kan zijn. De correcte notatie wordt dan **float x = 10f;**
- Signedness zegt iets over het kunnen opslaan van negatieve getallen. Unsigned kan geen negatieve getallen bevatten.
 - Overflow van int gaat naar negatieve waarde
 - Overflow van uint gaat naar nul.

- Als integers door elkaar worden gedeeld, worden ze altijd omlaag afgerond.

int + int = ✓

int * int = ✓

int / int = ✓

int / double = ✓

double / int = ✓

double / double = ✓

- != is **geen** correcte syntax in c#, dan zal je maar > moeten gebruiken.
- 0.4 + 0.6 != 0.8, **omdat** de manier waarop doubles en floats worden opgeslagen niet geheel nauwkeurig is.

- Substring: Retourneert het deel van de string vanaf opgegeven index tot einde:
 - String tekst = “Hello, World!”; → tekst.Substring(7) → “World!”
 - String tekst = “Hello, World!”; → tekst.Substring(7, 4) → “Wor!”
- Alleen { } symbolen zijn genoeg om een scope te starten. Als iets hierbinnen wordt gedeclareerd, kan je het hier buiten niet meer oproepen.
- **Lazy evaluation:** Code op zo’n manier schrijven dat het is geoptimaliseerd. Bijvoorbeeld bij een if statement met &&, eerst checken voor iets met minder mogelijkheden.
- **Enum:** Een waarde type dat een set van benoemde constante waarde vertegenwoordigt.
 - Enum Seizoen {winter, lente, zomer, herfst} → winter = 0, lente = 1, enz. \
 - (Seizoen)1 → Lente.

BRANCHING / CONDITIONALS

- Unreachable cases zijn niet toegestaan:
 - `switch (x) {case >= 85, case >= 75 → 75 is niet reachable, dus foutmelding.`
- **Ternary operator:** `conditie ? expressie1 : expressie2`

HERHALINGSSTRUCTUREN

- Als een forloop wordt afgesloten: `for (int i =0; i < 3; i++); {...}`, dan gaat eerst `i` gewoon naar 3, en daarna volgt dan andere code.
- Als je de update-sectie bij een for loop vergeet (de laatste), dan kan de loop oneindig door blijven gaan...

CODESTYLE

- **camelCase:** eerste woord zonder hoofdletter, de rest wel. (camelCase, verbruikPerDag)
- **PascalCase:** Elk nieuw woord met een hoofdletter. (PascalCase, VerbruikPerDag)

- | Type | Regel | Voorbeelden |
|--------------------|--------------------|--|
| Variabelen | camelCase | <pre>string firstName;
DateTime startDate;</pre> |
| Klasse naam | PascalCase | <pre>public class Student
{
}

public class PhoneNumber
{
}</pre> |
| Constanten | PascalCase | <pre>public const double Pi = 3.14159;
public const int DaysInWeek = 7;
public const string WelcomeMessage = "Hello, World!";</pre> |
| Methoden | PascalCase | <pre>public override string ToString()
{
}

public PhoneNumber(string countryCode, string areaCode, string number)
{
}</pre> |
| Methode argumenten | camelCase | zie methode <code>PhoneNumber</code> hierboven |
| Interface | begint met een 'I' | <pre>public interface IShapeWithArea
{
}

public interface IShapeWithPerimeter
{
}

public class Rectangle : IShapeWithArea, IShapeWithPerimeter
{
}</pre> |

- Commentaar:
 - `/* */` gaat over meerdere regels
 - `//` gaat maar voor één regel

METHODES

- Standaard schrijfwijze van een methode:
 - **var** MethodeNaam (**var** parameter1, **var** parameter2, etc) { *code* + *return* }
- Een methode kan een andere methode returnen, of een datatype.

LIJSTEN

- Volgende acties kun je doen met een lijst:
 - **Initialisatie:** `List<datatype> = [data];`
 - **Toevoegen:** `list.Add(Toevoeging);`
 - **Remove:** `list.Remove(te verwijderen data);` OF `list.RemoveAt (te verwijderen index);`
 - **Wijzigen:** `list[index] = nieuwe data`
- Ook kun je filteren en sorteren:
 - **Sorteren:** `List.Sort();`
 - **Filteren:** `list.Find(All)(labda => sorteerwaarde("waarde"));`
 - **Bijv:** `List<string> selectie = namenLijst.FindAll(naam => naam.StartsWith("J"));`
 - Find = Eerste instantie gevonden
 - FindAll = Alle instanties

```
List<string> namenLijst = ["Marc", "Marcel", "William", "Robin", "Jan", "Johan", "Erik"];

string eersteGevonden = namenLijst.Find(naam => naam.StartsWith("J"));
Console.WriteLine(eersteGevonden);

List<string> selectie = namenLijst.FindAll(naam => naam.StartsWith("J"));
foreach(var naam in selectie)
{
    Console.WriteLine(naam);
}
```