C++ Introductie

Het doel van ons profielwerkstuk is om een programma te schrijven dat eilanden genereert. Dit doen we (grotendeels) in de taal C++. Om de code te lezen is het dus handig om wat basiskennis van C++ te hebben en die zullen we in dit hoofdstuk proberen te geven.

De versie van C++ die we gebruiken is C++11. C++11 is een versie van C++ die in 2011 is goedgekeurd door de ISO (de Internationale Organisatie voor Standaardisatie). We hebben deze versie gekozen omdat deze oud genoeg is om goed ondersteund te worden, maar ook nieuw genoeg om vele goede functies te hebben. De belangrijkste functie die we nodig hadden uit C++11 was een goede pseudo-willekeurigheid generator, over deze generator vertellen we verderop meer.

Objectgeoriënteerd programmeren

Om C++ code te kunnen lezen moet je eerst meer weten over het objectgeoriënteerd programmeren, vaak afgekort als OOP van het Engelse Object-Oriented Programming. C++ is namelijk oorspronkelijk ontworpen om aan de programmeertaal C de mogelijkheid voor objectgeoriënteerd programmeren toe te voegen.

De belangrijkste twee concepten in objectgeoriënteerd programmeren zijn klassen en objecten. Een klasse dient als een blauwdruk voor objecten. In een klasse twee soorten informatie over de objecten die eruit kunnen ontstaan worden gedefinieerd:

- Welke *attributen* het object heeft waarmee de toestand en de eigenschappen van het object worden opgeslagen.
- Welke *methoden* het object heeft. Deze methoden zijn functies die de attributen van het object kunnen aanpassen of inlezen en daarmee dingen kunnen doen, zoals een van de attributen afdrukken op het scherm.

Voorbeeld van klassen en objecten

We zullen de concepten van klassen en objecten proberen te verduidelijken met hulp van een klein voorbeeld. We hebben een klasse `Persoon` die de attributen `naam` en `leeftijd` definieert en de methode `zegHalloTegen` dat een argument aanneemt, namelijk de `aangesprokene` waartegen het object hallo moet zeggen. Deze methode zou dan zo kunnen zijn gemaakt dat hij afdrukt: “Hallo `aangesprokene`, ik ben `naam` en ik ben `leeftijd` jaar oud”.

Vervolgens zou je met deze klasse een object `jan` kunnen initialiseren met de `naam` “Jan” en de `leeftijd` 17. Als je tegen dit object dan `zegHalloTegen` “Bas” zou aanroepen, zou het object het volgende afdrukken: “Hallo Bas, ik ben Jan en ik ben 17 jaar oud”.

Objectgeoriënteerd programmeren in C++

In C++ worden de definities van klassen vaak opgesplitst in een header bestand dat aangeeft welke attributen en methoden de klasse heeft en een apart bestand dat de implementaties van de methoden aangeeft. Ook heeft elke klasse in C++ minimaal één constructor, een speciale functie die een object

kan initialiseren vanuit de klasse. De constructor zou in onze voorbeeld met een klasse voor personen bijvoorbeeld twee argumenten kunnen aannemen, de `naam` en de `leeftijd`.

Het header bestand voor de `Persoon` klasse zou bijvoorbeeld een bestand genaamd “persoon.h” kunnen zijn, omdat het gebruikelijk is om header bestanden de extensie “.h” te geven. De inhoud hiervan zou er als volgt uit kunnen zien:

```
#include <string>

class Persoon {
public:
    Persoon(std::string naam, int leeftijd);
    void zegHalloTegen(std::string aangesprokene);

private:
    std::string naam;
    int leeftijd;
};
```

Voorbeeld 1: persoon.h

In dit bestand wordt dus vastgesteld welke attributen en methodes de klasse `Persoon` bevat. Om te begrijpen hoe dit wordt genoteerd zullen we regel voor regel de code doorlopen.

```
#include <string>
```

Regels die beginnen met een hekje ('#') zijn instructies voor de compiler. De compiler is een programma die de C++ code die we schrijven omzet naar een programma dat kan worden uitgevoerd. In dit geval verteld `#include <string>` aan de compiler dat we de module `string` nodig hebben. Deze gebruiken we verderop namelijk voor het type `std::string`.

```
class Persoon {
```

Dit geeft aan dat we een klasse gaan definiëren met de naam persoon, de definitie van de klasse is alles binnen de accolades ('{' en '}').

```
public:
```

Dit geeft aan dat de komende regels allemaal over attributen en methodes van de klasse gaan die publiek zijn. Publiek geeft aan dat een methode of attribuut ook door gebruikers van de objecten die vanuit de klasse worden gemaakt kan worden gebruikt. De private attributen en methoden kunnen alleen door methoden van het object zelf worden gebruikt.

```
    Persoon(std::string naam, int leeftijd);
```

Hier wordt de constructor gedefinieerd, waarmee dus een object van deze klasse kan worden geïnitialiseerd. Je kan zien dat dit de constructor is doordat de naam precies hetzelfde is als de naam van de gehele klasse, namelijk `Persoon`. Na `Persoon` staan de twee argumenten die je aan de constructor moet meegeven tussen haakje, gescheiden door een komma.

Het eerste is `std::string naam` dit geeft een argument aan van het type `std::string`. Dit

is het type waarmee stukjes tekst worden aangegeven in C++, die vaak gewoon kortaf strings worden genoemd. `std::` geeft aan dat het om het type `string` gaat dat in de standaard is gedefinieerd (std is kort voor standard), het is vooral belangrijk om te onthouden dat `std::string` een stukje tekst aangeeft. Achter het type staat de naam van het argument, wat in principe niet belangrijk is voor de klasse, maar wel handig voor de mensen die het lezen om te zien wat het argument aangeeft.

Het tweede is `int leeftijd`, een argument van het type int. Een int is een integer, dus een geheel getal. Achter `int` staat wederom de naam van het argument, die ook hier niet belangrijk is, maar wel aangeeft waar het argument voor zal worden gebruikt. Na de argumenten komt de punt-komma om aan te geven dat dit het einde van de definitie van de constructor is.

```
    void zegHalloTegen(std::string aangesprokene);
```

Hier wordt de eerste (en meteen ook de laatste) echte methode gedefinieerd, namelijk de methode genaamd `zegHalloTegen`. Voor de naam van de methode staat `void` dit betekent dat zegHalloTegen niks terug geeft (void betekent leegte in het Engels). Dit heet het *return type*, dat dus aangeeft wat de methode "*return*t". Dit *return type* is nodig omdat sommige methoden iets terug moeten geven, zodat een andere methode dat kan gebruiken.

Tussen haakjes staan achter de naam de argumenten die deze methode mee moet krijgen gegeven, dit is er in dit geval maar één namelijk `std::string aangesprokene`. Dit is dus de naam van degene die door het object moet worden begroet. Deze naam is ook weer van het type `std::string` en geeft dus een stukje tekst aan. Na dit argument volgt de punt-komma om aan te geven dat de definitie van deze methode hier eindigt.

```
private:
```

Dit geeft aan dat de volgende definities private attributen en methoden aangeven.

```
    std::string naam;
```

Dit is het eerste attribuut van de klasse, genaamd `naam`. Dit attribuut is van het type `std::string`, dus weer een stukje tekst. Bij een attribuut is de naam die eraan wordt gegeven in het header bestand wel belangrijk in tegenstelling tot de namen van argumenten van methodes. De naam van een attribuut wordt namelijk gebruikt om dat attribuut te kunnen veranderen of bekijken in de code.

```
    int leeftijd;
```

Dit is het tweede attribuut van de klasse, met de naam `leeftijd`. Dit attribuut is van het type `int`, dus het gaat om een getal.

```
};
```

Een sluitende accolade gevolgd door een punt-komma geeft het eind van de definitie van attributen en methoden van de klasse aan.

Nu gaan we kijken naar het bestand dat de methoden en de constructor ook echt definieert. Dit bestand zou bijvoorbeeld “persoon.cpp” kunnen heten. De extensie “.cpp” (“.cpp” staat voor C Plus Plus) is gebruikelijk voor de echte code bestanden waarin de methodes van de klassen worden uitgewerkt.

```
#include <iostream>
#include "persoon.h"

Persoon::Persoon(std::string naam, int leeftijd)
{
    this->naam = naam;
    this->leeftijd = leeftijd;
}

void Persoon::zegHalloTegen(std::string aangesprokene)
{
    std::cout << "Hallo " << aangesprokene
              << ", ik ben " << this->naam
              << " en ik ben " << this-> leeftijd
              << " jaar oud" << std::endl;
}
```

Voorbeeld 2: persoon.cpp

Laten we weer stukje voor stukje door de code heen lopen:

```
#include <iostream>
```

Dit vertelt de compiler dat we `iostream` nodig hebben, hierin zijn de functies gedefinieerd die ons helpen om dingen af te drukken op het scherm.

```
#include "persoon.h"
```

Dit vertelt de compiler dat we de definities van welke methoden en attributen de klasse `Persoon` heeft nodig hebben. Die hebben we namelijk opgeslagen in “persoon.h”. We gebruiken hier aanhalingstekens in plaats de hoekige haakjes ('<' en '>'), omdat dit een bestand is dat we zelf hebben gedefinieerd en niet iets wat al door iemand anders is gedefinieerd.

```
Persoon::Persoon(std::string naam, int leeftijd)
```

Deze regel geeft aan dat we nu de code voor de constructor van `Persoon` gaan definiëren. De eerste keer dat er “Persoon” staat verwijst dat naar de klasse waar dit bij hoort, daarna komt de naam van hetgene wat we nu van die klasse gaan beschrijven. Dat is dus hier de constructor, die dezelfde naam heeft als de klasse zelf. Na de naam komen tussen haakjes weer de argumenten. Je kan zien dat de twee types weer hetzelfde zijn, met daarachter de namen van de argumenten. De namen van de argumenten maken deze keer wel uit, omdat de namen zijn waarmee deze argumenten kunnen worden gebruikt in de code van deze methode.

```
{
```

Dit geeft aan dat vanaf deze openings-accolade tot de bijhorende sluitings-accolade de code van

deze methode is.

```
    this->naam = naam;
```

Dit is een *statement*, een stukje code dat wordt uitgevoerd zodra deze methode wordt gebruikt. In het midden staat de =-operator, die wordt gebruikt om de waarde van de rechter variabele in de linker te stoppen. In dit geval wordt de waarde van `naam`, die we in deze constructor als argument mee hebben gekregen in `this->naam` gezet. De "`this`" hierin staat voor dit object, dus het object dat we met deze constructor gaan bouwen. Het "`->naam`" staat voor het attribuut `naam` hiervan. Dus deze regel kopieert de waarde van het argument `naam` wat deze constructor heeft megekregen in het attribuut `naam` van het object wat we met deze constructor maken.

```
    this->leeftijd = leeftijd;
```

Hier gebeurt hetzelfde als in de vorige regel, maar dan wordt de waarde van het argument `leeftijd` in het attribuut `leeftijd` gezet.

```
}
```

Dit geeft aan dat hiermee de code die bij de constructor hoort afgelopen is.

```
void Persoon::zegHalloTegen(std::string aangesprokene)
```