Inleiding programmeren natuurkundigen

Ragnar Groot Koerkamp

11 juni 2015

Contents

1 Overzicht

In deze cursus leer je de basis van programmeren in `C++`. We beginnen we bij het opstarten van een project in een tekstverwerker. Daarna gaan we in op de structuur van een programma en wat de regels zijn bij het schrijven ervan. Vervolgens wordt aan de hand van veel voorbeelden en simpele opdrachten de basis van `C++`zelf uitgelegd. We gaan hierbij bijvoorbeeld in op variabelen, `if`-statements en `for`-loops.
Na afloop van deze cursus zou je van onderstaande code moeten kunnen achterhalen wat het doet en hoe het werkt:[1]

[1] of gewoon meteen hier uitleggen?

```
#include <iostream>
using namespace std;

int main(){

    int aantal;
    cout << "Voer het aantal personen in:" << endl;
    cin >> aantal;

    int som = 0;
    int kinderen = 0;

    for(int i=0; i<aantal; i++){
        int leeftijd;
        cout << "Wat is je leeftijd?" << endl;
        cin >> leeftijd;
        cout << "Dan ben je al meer dan ";
        cout << 356 * leeftijd;
        cout << " dagen oud!" << endl;

        som += leeftijd;
        if(leeftijd < 18)
            kinderen++;
    }

    cout << "De totale leeftijd is " << som << " jaar!" << endl;
    cout << "Er zijn " << kinderen << " kinderen"<<endl;

    return 0;
}
```

2 Visual studio/ of online?

Voor Manon Hier ook iets over compileren

3 Basis

We bekijken nu eerst hoe een programma globaal in elkaar zit. Elk programma heeft een `main` functie waar het altijd begint. Vandaag staat alle code in de main functie, dus tussen de accolades die aangeven welke code bij de `main` hoort. Bovenaan het programma zie je groene regels commentaar staan die beginnen met `//`. Deze regels worden altijd overgeslagen bij het uitvoeren. Het commentaar begint pas bij de `//`, dus ervoor kan nog wel code staan. Het is handig om commentaar bij de code te zetten om te verduidelijken wat de code doet. Dat is handig als je ergens later weer naar terug kijkt en dan niet opnieuw hoeft uit te zoeken wat de

code nou eigenlijk deed. Commentaar kan ook meer dan een regel omvatting. Dan gebruik je aan het begin `/*` en aan het einde `*/`.

```
// dit pakketje is nodig om straks
// naar de console te kunnen schrijven
#include <iostream>

// en deze regel maakt die functies alvast
// toegankelijk voor de rest van het programma
using namespace std;

/*
 * Het programma begint in de ‘main‘.
 * Dit is de functie die wordt aangeroepen
 * zodra je programmma begint.
 * De main moet altijd een geheeltallige waarde
 * retouneren, en meestal 0.
 * Elk getal ongelijk aan 0 betekent dat er een error is geweest.
 */
int main(){
    return 0; // of deze weglaten?????
}
```

Zoals je misschien al is opgevallen eindigt bijna elke regel van het voorbeeld met een puntkomma (`;`). Dit is nodig zodat het programma dat de code uitvoert weet wanneer een nieuwe regel met nieuwe code begint. In sommige gevallen zijn geen puntkomma's nodig, zoals voor of na een accolade.
In C++mag je overal zo veel witruimte (spaties, tabs, enters) neerzetten als je maar wilt. De enige eis is dat sommige dingen niet direct aan elkaar geschreven morgen worden, en er dus minstens een spatie tussen moet zitten.
Verder zie je ook nog `#include <iostream>` staan. `<iostream>` is een pakketje dat we later nodig zullen hebben om tekst te schrijven naar de console (dat zwarte schermpje, zie verderop) en om data in te lezen. Alle functies in dat pakketje behoren tot de C++-standaard-library[2]. Om ze toegankelijk te maken moeten we daarom ook `using namespace std;` bovenaan onze code zetten.
Het programma eindigt met `return 0;`[3]. Het getal dat achter `return` staat, staat voor een mogelijke error die het programma geeft. 0 betekent hier dat er geen errors zijn geweest.

[2] beter kan ik het niet maken. Suggesties?
[3] of dit gewoon weglaten?

3.1 Opdrachten

1. Type de code uit het voorbeeld over, maar laat het commentaar nog even weg. Start nu het programma.[4] Zorg ervoor dat dat zwarte schermpje niet meteen weer uit beeld verdwijnt, maar daar blijft staan en pas weg gaat als je op een toets drukt. Dat doe je met shift-`F5` in Visual Studio, of ... in IDE.

2. Voeg nu ook commentaar toe op de plekken waar dat in het voorbeeld ook is gedaan. Check dat het programma nog steeds werkt.

3. Probeer nu ook op binnen `main` je eigen commentaar toe te voegen van beide soorten.

4. Voeg nu een blok code (dus van de soort `/* */`) toe binnen een code regel, dus bijvoorbeeld tussen `int` en `main(){` of tussen `return` en `0;`. Omdat witruimte niet van belang is en het commentaar wordt weggegooid, zou je programma nog steeds moeten werken.

4 Console uitvoer

Wanneer je het programma start, zie je steeds een zwart scherm verschijnen. Dat heet de *console* of *terminal*. Daar kunnen we tekst neerzetten of juist van lezen. Het neerzetten van tekst kan met het commando `cout`, oftewel, Console-out. Je kan achter de `cout <<` bijvoorbeeld een string of een getal zetten. Ook is er het speciale commando `endl`, dat voor *endline* staat. Hiermee beëindig je dus een regel. Het is handig om te onthouden dat je de tekst 'in de console stopt' en de pijltjes dus richting `cout` staan. De algemene syntax is dus:
`cout <<` *string, cijfer, of nog iets anders* `;`
Het is soms handig om meer dan één ding tegelijk te kunnen schrijven. Dat kan met behulp van: `cout <<` *string, cijfer, of nog iets anders* `<<` *andere string, cijfer, etc* `<<` .. `;`
Als je, zoals in het voorbeeld, direct een `endline` gebruikt, heb je een speciale versie van de tweede, langere, variant gebruikt.

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     // Schrijf tekst naar de console
6     // en beeindig de regel
7     cout << "Mijn leeftijd is vandaag:" << endl;
8
9     // en schrijf een getal
```

[4] dat doe je door ... te doen, afhankelijk IDE.

```
10      cout << 22 << endl;
11  }
```

4.1 opdrachten

1. Schrijf nu alleen de tekst "How are you?" naar de console in plaats van "Mijn leeftijd is", en zorg dat het getal niet meer wordt geschreven door er commentaar van te maken.[5]

2. In plaats van twee dingen op één regel weg te schrijven, kunnen we ook twee regels gebruiken. Verander je code naar de twee regels `cout << "How are you ?";` en `cout << endl;`. Je ziet dat er nog steeds het zelfde gebeurt als je het programma uitvoert.

3. Zorg nu dat het programma twee keer "How are you?" zegt, op twee regels onder elkaar.

4. Kan je er ook voor zorgen dat het op één regel komt te staan?

5. Print nu de lijst getallen van 1 tot en met 5, waarbij elk getal op zijn eigen regel staat.

6. We kunnen die getallen ook op één regel zetten door steeds `endl` te vervangen door `" "`.

7. Zorg er nu voor dat je in je code maar één keer het woord `cout` gebruikt. Zoals je ziet kunnen we het uitvoeren van getallen en tekst door elkaar het gebruiken op dezelfde regel.

5 Variabelen

Er zijn meerdere soorten variabelen. De belangrijkste zijn getallen en strings. Een getal kan of geheel zijn, of reëel. In het eerste geval gebruiken we een `int`, wat staat voor *integer*. Als een getal niet altijd geheel is, hebben we een `double` nodig. Een double kan elke mogelijke waarde aannemen, dus bijvoorbeeld ook π.
Een `string` is een stuk tekst. In computertaal is dat gewoon een rij van tekens.

```
1  #include <iostream>
2  #include <string> // nodig voor strings
3  using namespace std;
4
5  int main(){
```

[5] splitsen naar 2 opdrachtjes?

```
6       /*
7        * Weinig inspiratie hier
8        * TODO:
9        * + - * /
10       * integer division
11       * (%)
12       */
13      int a = 10;
14      cout << "a = " << a << endl;
15      cout << "2*a = " << 2*a << endl;
16
17      int b = 3.5; // een int is altijd geheel
18      cout << "b = " << b << endl;
19
20      double c = 3.5; // een double kan elk getal zijn
21      cout << "c = " << c << endl;
22      cout << "c kwadraat = " << c*c << endl;
23
24      string s = "Hello world!";
25      cout << s << endl;
26
27      return 0; // no error
28  }
```

5.1 Opdrachten

1.

6 Console invoer

[6] Het is soms ook handig om juist invoer van de gebruiker te lezen. Daarvoor hebben we `cin`, console-in. Hieronder lezen we een getal in, wat we in de variabele `leeftijd` stoppen. De pijltjes staan hier naar rechts, omdat het getal uit de console komt, en in de variabele wordt gestopt. We kunnen ook naar een string lezen in plaats van naar een getal. De syntax is:
`cin >>` *variabele met het type dat je wilt lezen* `;`

```
1   #include <iostream>
2   using namespace std;
3
4   int main(){
5       cout << "Hoe oud ben je?" << endl;
6       int leeftijd;
7       cin >> leeftijd;
```

[6] Ik heb nu maar uitvoer-variabelen-invoer gedaan, omdat er op die manier geen vooruit-referenties nodig zijn.

```
8       cout << "Je bent " << leeftijd << " jaar oud" << endl;
9   }
```

6.1 Opdrachten

1. Print op een nieuwe regel ook nog de leeftijd die je volgend jaar bent, door `leeftijd + 1` te gebruiken in plaats van `leeftijd`.

2. In plaats van een `int`, kunnen we `leeftijd` ook een `double` maken, zodat hij niet geheel meer hoeft te zijn. Doe dat, en kijk wat er gebeurt als je bijvoorbeeld 1, 2.3, 1234.345 en -11234 invult. Vergelijk dit ook met het geval waarin het nog een integer was.

3. We kunnen er ook nog een string van maken. Als je de code van de eerste opdracht nog hebt, werkt die hier nog steeds. Wat doet de `+` blijkbaar met tekst als argumenten?

4. [7]

7 If

Tot nu toe lag het altijd precies vast welke code werd uitgevoerd en hoe vaak dat gebeurde. Het is ook mogelijk om iets alleen uit te voeren als aan een bepaalde voorwaarde is voldaan. Daarvoor gebruiken we het `if`-statement. De syntax is

```
1   if(conditie){
2       //code wanneer conditie waar (true) is;
3   } else {
4       //code wanneer conditie niet waar (false) is;
5   }
```

Hierin is `conditie` een expressie die tot een boolean evalueert, en dus waar of niet waar is. Voorbeelden zijn bijvoorbeeld `if(i == 1)` en `if(i < 10)`, waarbij i een integer is.
Soms hoeven we alleen maar iets te doen als de conditie waar is. In dat geval kunnen we ook den syntax

```
1   if(conditie){
2       // code wanneer de conditie waar is;
3   }
```

gebruiken.
In het volgende voorbeeld hangt de uitvoer van het programma af van de invoer van de gebruiker.

[7] Meer opdrachten nodig? Echt meer kan ik niet bedenken

```cpp
#include <iostream>
using namespace std;

int main(){
    cout << "voer je leeftijd in:" << endl;
    int leeftijd;
    cin >> leeftijd;
    if(leeftijd >= 18){
        cout << "Je bent volwassen!" << endl;
    } else {
        cout << "Je bent nog een kind!" << endl;
    }

    return 0;
}
```

8 Opdrachten

1. Kan je er voor zorgen dat de twee gevallen uit het voorbeeld verwisseld worden? Dus dat de *true* case nu `"Je bent nog een kind!"` print en de *false* case juist `"Je bent volwassen!"`.

2. Print nu ook nog de string `"Je mag nu autorijden!"` wanneer de leeftijd precies 18 jaar is. Hierbij heb je dus geen `else` nodig.

3. Bepaal nu of de leeftijd kleiner is dan 18, tussen de 18 en 65 is, of dat hij minstens 65 is. In het laatste geval moet je `"U bent gepensioneerd!"` schrijven naar de console.

4. Voor herhaalde `if`-statements is er de syntax

```cpp
if(conditie 1){
    // code wanneer conditie 1 waar is;
} else if(conditie 2){
    // code wanneer conditie 2 waar is;
} else {
    // code wanneer geen van beiden waar is.
}
```

   Gebruik dit nu voor de code van de vorige opdracht.

9 For

Als je een stukje code vaker dan één keer wilt uitvoeren, kan je hem natuurlijk een aantal keer onder elkaar kopiëren en plakken. Als het aantal keer dat je iets

wilt doen niet vast ligt, is dat niet meer handig[8]. Daarvoor is de `for`-loop. Deze herhaalt iets een bepaald aantal keer. Het kan ook zo zijn dat het aantal keer dat iets herhaalt wordt nog niet van tevoren vast ligt. De syntax is

```
for(initialisatie; conditie; verhoging){
    //code die herhaald wordt
}
```

In de initialisatie wordt de teller variabele meestal gedeclareerd en geïnitialiseerd[9]. Dat is dus bijvoorbeeld `int i = 0;`. De conditie lijkt op de conditie van het `if`-statement. Het is weer een boolean waarde die bepaald of we moeten doorgaan met het herhalen van de code of dat we al moeten stoppen. Vaak is dit iets als `i < 10`. De verhoging bepaald wat we na elke iteratie met de variabele doen. Dat is meestal iets als `i++`, wat equivalent is met `i+=1` en `i=i+1`. Als we dit combineren, zien we dat `i` begint op 0 en net zo lang 1 verhoogd wordt, tot dat hij niet meer kleiner is dan 10. Op dat moment zal `i` dus 10 zijn. In het algemeen is de code van een `for`-loop equivalent aan

```
initialisatie;
if(conditie){
    //code die herhaald wordt
    verhoging;
    if(conditie){
        //code die herhaald wordt
        verhoging;
        if(conditie){
            //code die herhaald wordt
            verhoging;
            if(conditie){
                // oneindig lang door
            }
        }
    }
}
```

Hier is een voorbeeld van het gebruik van `for`.

```
#include <iostream>
using namespace std;

int main(){
    cout << "drie hoeratjes:" << endl;
    for(int i = 0; i < 3; i++){
        cout << "hoera" << endl;
    }
```

[8] Merk op dat het nooit verstandig is om hier te kopiëren en plakken. Als je iets maar twee keer moet doen, kan dat prima met een for loop

[9] In `int i = 1;`, is `int i` de declaratie van `i`, en de `i = 1` is de initialisatie.

```
9
10      cout << "tellen tot 10:" << endl;
11      for(int getal = 0; getal <= 10; getal++){
12          cout << getal << endl;
13      }
14
15      return 0;
16  }
```

Als je dit intikt en uitvoert, zie je dat er eerst drie keer 'hoera' wordt geschreven, en daarna de getallen van 0 tot en met 10. De werking van een `for`-loop is als volgt:
In de `initialisatie` wordt meestal een variabele geïnitialiseerd, zoals hier `int i=0`. In de `conditie` kijken we steeds of we moeten doorgaan met de volgende iteratie, of dat we moeten stoppen met herhalen. Zolang de conditie waar is gaan we door, en als hij niet meer waar is stoppen we met herhalen. Hier gaan we dus alleen maar door wanneer `i` kleiner is dan 3. In de `verhoging` verhogen we de variabele waarover we lopen. Meestal verhogen we hem steeds met 1 tegelijk maar dat hoeft niet per se.

9.1 Opdrachten

1. Print nu 5 keer `"hoera"` op een nieuwe regel.
2. Schrijf nu alleen de even getallen tot en met 10 op.
3. TODO: MEER OPDRACHTEN

10 Arrays [optioneel]

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5
6      int kwadraten[10];
7
8      for(int i = 0; i < 10; i++){
9          kwadraten[i] = i*i;
10     }
11
12     int som = 0;
13     for(int i = 0; i < 10; i++){
14         som += kwadraten[i];
15     }
16
```

```
17      cout << "de som is " << som << endl;
18
19      return 0;
20  }
```

11 While [optioneel]

Tot slot hebben we nog de `while`-loop. Deze lijkt een beetje op de `for`-loop, maar is toch anders. Bij een `for`-loop weten we meestal van tevoren al hoe vaak we de code willen herhalen. Als we dat niet precies weten, is een `while`-loop handig. We herhalen de code nu net zo lang tot `conditie` onwaar wordt in

```
1  while(conditie){
2      code die herhaald wordt;
3  }
```

In de code hieronder zie je een voorbeeld. We beginnen hier met $n = 0$, en vervolgens schrijven we steeds n^2 op, net zo lang tot n^2 groter wordt dan 1000.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      cout << "alle kwadraten onder 1000" << endl;
6
7      int n = 0;
8      while(n * n <= 1000){
9          cout << (n * n) << endl;
10         n = n+1;
11     }
12
13     return 0;
14 }
```