Inleiding programmeren natuurkundigen

Fysicie
i.s.m.
Ragnar Groot Koerkamp

11 juni 2015

Inhoudsopgave

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:

```
#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 Compilers

Een computer heeft heel veel rekenkracht en hier kunnen zeker natuurkundigen nuttig gebruik van maken. Met behulp van een computer kunnen we bijvoorbeeld numerieke oplossingen vinden van vergelijkingen die geen analytische oplossing hebben. Maar om de computer iets voor ons te laten uitrekenen, zullen we hem moeten vertellen wat hij precies moet doen. We moeten dus tegen de computer kunnen praten, en dit praten gebeurt in een programmeertaal. Er zijn verschillende programmeertalen zoals C, C++ , Java en Python, en een programma schrijven we in zo'n programmeertaal. De computer kan echter deze programmeertalen nog niet direct begrijpen, daarvoor is nog een extra tussenstap nodig. De broncode, geschreven door ons in een programmeertaal, moet worden vertaald naar machinecode. Deze machinecode bevat de opdrachten die de processor moet uitvoeren. Een compiler vertaalt de programmeertaal voor ons naar die machinecode. Voor

iedere programmeertaal bestaan er compilers die die programmeertaal naar machinecode vertalen.
Er zijn verschillende C++ compilers. Veel compilers moet je installeren op je computer. Dat gaan we vandaag niet doen, in plaats daarvan gaan we gebruik maken van cloud9, dat is een compiler in een cloud. Deze kun je vinden op http://c9.io. Verdere uitleg over hoe je precies gebruik maakt van deze compiler vind je bij de eerste opgave.

3 Basis

We bekijken nu eerst hoe een programma globaal in elkaar zit. Elk programma heeft een main functie. Bij het uitvoeren van het programma wordt als eerste alles wat in de main functie staat uitgevoerd. Dit gebeurt regel voor regel, van boven naar beneden. 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 één regel omvatten. Dan gebruik je aan het begin /* en aan het einde */. Hieronder staat een voorbeeld van een basis voor een programma.

```
1  // dit pakketje is nodig om straks
2  // naar de console te kunnen schrijven
3  #include <iostream>
4
5  // en deze regel maakt die functies alvast
6  // toegankelijk voor de rest van het programma
7  using namespace std;
8
9  /*
10  * Het programma begint in de ‘main‘.
11  * Dit is de functie die wordt aangeroepen
12  * zodra je programmma begint.
13
14 (De sterretjes aan het begin van de regel
15 zijn hier niet per se nodig)
16 */
17
18 int main(){
19     // de code komt hier
20 }
```

In C++ moeten de meeste regels met een puntkomma (;) eindigen. Dat is nodig zodat de compiler weet wanneer een nieuwe regel met nieuwe code begint. Alleen in sommige gevallen zijn geen puntkomma's nodig, zoals direct na een accolade of achter een #include.
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 en om data in te lezen. (De console is een scherm waar je output naar toe kunt schrijven en waar je input uit terug kunt vragen. Bij cloud9 is dat het schermpje dat onder in je beeldscherm staat, zoals je

straks vanzelf zult zien.) Alle functies in dat pakketje behoren tot de C++ -standaard-library. Om ze makkelijk te gebruiken moeten we `using namespace std;` bovenaan onze code zetten.
Let op het gebruik van hoofdletters en kleine letters in de code. C++ is hoofdlettergevoelig, dus `Int` is niet hezelfde als `int`!

3.1 Opdrachten

1. Type de code uit het voorbeeld over, maar laat het commentaar nog even weg. Om dit te kunnen doen in cloud9 volg je de volgende stappen:

   Ga naar **Cloud 9** op `http://c9.io/`. Maak een account aan. Als je ingelogd bent, klik je op 'create workspace' en vervolgens weer op 'create workspace'. Kies een leuke naam voor je workspace. Selecteer C/C++ , de rest kun je laten staan zoals het nu staat. Klik op create en wacht even. (Het duurt even voordat de workspace is gemaakt, ook al spring je direct weer terug naar het dashboard.) Klik links onder 'my projects' op de workspace die je net hebt gecreëerd. Wacht even tot er 'start editing' staat en klik hierop. Ga vervolgens naar 'File' en klik op 'New File'. Sla op met CTRL + S en verander de naam van de file naar `hallo.cpp`. De naam van je programma mag altijd alleen letters en underscores bevatten. De `.cpp` zorgt ervoor dat de compiler weet dat het een C++ programma is. (De p's staan voor *plus*.)

   Type nu de code uit het voorbeeld over in dit bestand, maar laat het commentaar nog even weg. Start het programma door op 'Run' bovenaan in de werkbalk te klikken. Als alles goed gaat zie je nu onderin het beeld een blauw schermpje met de tekst 'Running /home/ubuntu/workspace/hallo.cpp'. Omdat het programma nog niets doet, staat hier nog geen uitvoer onder.

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 **om** binnen `main` je eigen commentaar toe te voegen van beide soorten, dus met `//` en `/* */`.

4. Voeg nu nogmaals commentaar toe in de vorm van een blok (dus van de soort `/* */`) toe, maar doe dit nu ergens midden in een code regel, dus bijvoorbeeld tussen `int` en `main()`. Omdat witruimte niet van belang is en het commentaar wordt weggegooid, zou je programma nog steeds moeten werken.

4 Console uitvoer

Bij Cloud 9 staat onderaan je beeldscherm een breed, rechthoekig blauw schermpje. Dat heet de *console* of *terminal*. Dit kan ook een zwart popup-venster zijn of een linux-terminal, afhankelijk van de programmeer omgeving. Vandaag houden we het echter bij Cloud 9. In de console kunnen we tekst wegschrijven of juist lezen. Het neerzetten van tekst kan met het commando `cout`, oftewel, Console-out. Je kan achter de `cout <<` bijvoorbeeld een tekst 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 syntax van een programmeertaal omvat de spellingsregels/taalregels van de programmeertaal. De syntax beschrijft dus hoe je een bepaalde opdracht die je aan de computer

wilt geven op moet schrijven. De algemene syntax van cout is:

`cout <<` *string, cijfer, of nog iets anders* `;`

Hierbij staat de tekst 'string, cijfer, of nog iets anders' schuingedrukt, omdat je deze niet letterlijk zo in je programma mag zetten. Hier moet je bij het schrijven van je programma nog een specifieke invulling aan geven. 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* `<< .. ;`

```
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
10     cout << 22 << endl;
11
12     // of meerdere dingen tegelijk
13     cout << "Het is vandaag:" << endl;
14     cout << 18 << " juni " << 2015 << endl; // let op de spaties!
15
16 }
```

4.1 opdrachten

1. Maak eerst van de regel waarop 22 wordt weggeschreven commentaar, en maak vervolgens van het wegschrijven van de datum blok-commentaar.

2. Schrijf nu alleen de tekst "How are you?"naar de console in plaats van "Mijn leeftijd is".

3. **Verplichte opdracht:** Verander alle dubbele pijltjes << nu naar een enkel pijltje < en voer het programma uit. Je zult zien dat er geen "How are youïn de console komt te staan. In plaats daarvan krijgen we een foutmelding te zien. Aangezien je nog wel veel vaker een fout zult maken en foutmeldingen tegen zult komen, is het handig om deze te leren lezen. Op de tweede regel van de console staat nu:
   `/home/ubuntu/workspace/hallo.cpp: In function 'int main()':`
   Hier staat dat er in de main functie van het programma hallo.cpp een fout is opgetreden. Aangezien ons programma alleen uit een main functie bestaat komen we hier nog niet erg verder mee. Als je naar de volgende regel kijkt staat er waarschijnlijk iets als:
   `/home/ubuntu/workspace/hallo.cc:7:29: error: invalid operands of types 'bool' and '<unresolved overloaded function type>' to binary 'operator<'`
   `cout < "How are you?"< endl;`
   Hierbij duiden de getallen 7 en 29 na hallo.cpp respectievelijk het regelnummer en kolomnummer aan waarop de fout is opgetreden. Let op: dit is het regelnummer

waarop de compiler merkt dat er een fout is gemaakt. Dit is niet altijd het regelnummer waar de fout staat, maar meestal wel. Op de rest van deze regel staat een uitleg van wat de fout precies betekent. In dit geval hebben we de `operator<` gebruikt die voor booleans (bool) bedoelt is op dingen die geen booleans zijn. (Een boolean is een variabele die waar of niet waar is.) Als je niet weet wat een bepaalde foutmelding betekent loont het ook altijd om deze even te googlen, maar nu hoeft dat nog niet.

4. 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.

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

6. Kan je er ook voor zorgen dat er op één regel twee keer "How are you"komt te staan.

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

8. We kunnen die getallen ook op één regel zetten door steeds `endl` te vervangen door `" "`. Probeer dit eens uit.

9. Zorg er nu voor dat je in je code maar één keer het woord `cout` gebruikt. Zoals je ziet kunnen we in het programma er op één regel zowel voor zorgen dat er getallen als tekst naar de console worden geschreven.

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 1.2 of $\sqrt{2}$.
Een `string` is een stuk tekst. In computertaal is dat gewoon een rij van tekens. Verder hebben we nog `bool`s. Dat zijn waarden die waar (`true`) of onwaar (`false`) zijn.
Zorg dat je alle operaties in de code hieronder snapt, en controleer of de uitvoer klopt met wat je verwacht.[1] Let er op dat we nu ook de pakketjes `<string>` en `<cmath>` nodig hebben voor de strings en voor het worteltrekken en machtsverheffen.

```
1  #include <iostream>
2  #include <string> // nodig voor strings
3  #include <cmath> // voor de sqrt
4  using namespace std;
5
6  int main(){
7      // ints
8      int a = 2;
9      int b = 3;
10
11     cout << "a:\t" << a << endl;
12     cout << "b:\t" << b << endl;
```

[1] Deze code is ook online beschikbaar op `http://github.com/ragnargrootkoerkamp/cpp-tutorial` in `code/vars.cpp`

```
13
14      cout << "a+b:\t" << a+b << endl;
15      cout << "a-b:\t" << a-b << endl;
16      cout << "a*b:\t" << a*b << endl;
17
18      // en integer division, met afronding naar beneden
19      cout << "a/b:\t" << a/b << endl;
20
21      // verhoog a met 1
22      a = a + 1;
23      cout << "a:\t" << a << endl;
24
25      // en nog een keer
26      a += 1;
27      cout << "a:\t" << a << endl;
28
29      // en nog een keer
30      a++;
31      cout << "a:\t" << a << endl;
32
33
34      // of verlaag b met 2
35      b = b - 2;
36      cout << "b:\t" << b << endl;
37
38      // en nog een keer
39      b -= 2;
40      cout << "b:\t" << b << endl;
41
42      // Dit werkt ook voor +-*/
43
44
45      // doubles
46      double c = 2.5;
47      double d = 4.0;
48
49      cout << "c:\t" << c << endl;
50      cout << "d:\t" << d << endl;
51
52      cout << "c+d:\t" << c+d << endl;
53      cout << "c-d:\t" << c-d << endl;
54      cout << "c*d:\t" << c*d << endl;
55      cout << "c/d:\t" << c/d << endl;
56
57      cout << "sqrt(d):\t" << sqrt(d) << endl;
58      cout << "c^d:\t" << pow(c,d) << endl;
59
60      // strings
61      string s = "dit is string s";
62      string r = "dit is string r";
63
64      cout << "s:\t" << s << endl;
65      cout << "r:\t" << r << endl;
66
67      cout << "s+r:\t" << s+r << endl;
68
69      // ook hier kunnen we s+=r gebruiken
70      s += r;
```

```
71      cout << "s:\t" << s << endl;
72
73
74      // booleans
75      bool t = true;
76      bool f = false;
77
78      // bool worden geprint als 1 of 0!
79      cout << "t:\t" << t << endl;
80      cout << "f:\t" << f << endl;
81
82      cout << "niet t:\t" << !t << endl;
83      cout << "niet f:\t" << !f << endl;
84
85      cout << "t en f:\t" << (t&&f) << endl;
86      cout << "t of f:\t" << (t||f) << endl;
87
88      // en hier kunnen t &&= f en t ||= f;
89
90      // vergelijkingen
91      cout << "2 = 2:\t" << (2 == 2) << endl;
92      cout << "1 = 2:\t" << (1 == 2) << endl;
93      cout << "1 != 2:\t" << (1 != 2) << endl;
94
95  }
```

6 Console invoer

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* `;`

Bijvoorbeeld:

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      cout << "Hoe oud ben je?" << endl;
6      int leeftijd;
7      cin >> leeftijd;
8      cout << "Je bent " << leeftijd << " jaar oud" << endl;
9  }
```

6.1 Opdrachten

1. Print op een nieuwe regel de leeftijd die je over een jaar hebt, door `leeftijd + 1` te gebruiken in plaats van `leeftijd`. Test het programma door het uit te voeren en in de console je leeftijd in te voeren en op enter te drukken.

2. Als je nu voor je leeftijd een getal invult dat niet geheel is, zul je zien dat deze naar beneden wordt afgerond. Dat komt doordat we de leeftijd opslaan als `int`, en een integer is altijd geheel. 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 nu bijvoorbeeld 1, 2.3, 1234.345 en -1234 invult in de console.

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

```
if(conditie){
    //code wanneer conditie waar (true) is;
} else {
    //code wanneer conditie niet waar (false) is;
}
```

Hierin is `conditie` een expressie die tot een boolean evalueert, en dus waar of onwaar 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

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

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

```
#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;
    }
}
```

7.1 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 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

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

   Gebruik dit nu voor de code van de vorige opdracht.

8 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[2]. Daarvoor is de `for`-loop. Deze herhaalt iets een bepaald aantal keer. Het kan ook zo zijn dat het aantal keer dat iets herhaald wordt nog niet van tevoren vast ligt. De syntax is

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

De `for`-loop heeft een teller-variabele die voor iedere iteratie (herhaling) van de loop wordt opgehoogd. We zullen stap voor stap bekijken hoe een `for`-loop werkt:

1. De initialisatie wordt uitgevoerd. Hierbij krijgt de teller zijn beginwaarde, bijvoorbeeld `int i = 0;`. De initialisatie wordt alleen voor de allereerste iteratie uitgevoerd.

2. Er wordt gecontroleerd of de conditie *true* is (meestal hangt de conditie van de teller af). Dit werkt op dezelfde manier als de conditie bij het `if`-statement, dus de conditie is een boolean waarde die bepaald hoe lang de `for`-loop doorgaat. De conditie is vaak iets als `i < 10`. Als de conditie *true* is, gaat het programma verder met de stappen hieronder. Als de conditie *false* is wordt de for-loop afgebroken.

3. De code die tussen de accolades staat wordt uitgevoerd. Dit kan van alles zijn.

4. De teller wordt opgehoogd. Vaak wordt de teller met één opgehoogd, door middel van `i++`, wat equivalent is met `i+=1` en `i=i+1`. Je kunt de teller bijvoorbeeld ook met 2 verlagen, wat je kunt doen met `i-=2` of `i=i-2`.

5. Stap 2 t/m 4 worden herhaald net zo lang totdat bij een bepaalde iteratie de conditie bij 2 niet meer *true* is. De for-loop is dan ten einde.

Als je de kleine voorbeelden die hierboven stonden combineert dan zie je dat: `i` begint op 0 en wordt net zo lang met 1 verhoogd totdat hij niet langer kleiner is dan 10. Voor al

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

deze iteraties wordt de code tussen de accolades uitgevoerd, in dit geval gebeurt dat dus 10 keer.
Hieronder staat de zelfde uitleg nog een keer, maar dan in code in plaats van in woorden. 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;
    }

    cout << "tellen tot 10:" << endl;
    for(int getal = 0; getal <= 10; getal++){
        cout << getal << endl;
    }
}
```

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.

8.1 Opdrachten

1. Print nu 5 keer `"hoera"` op een nieuwe regel.

2. Schrijf nu alleen de even getallen tot en met 10 op.

9 Arrays [optioneel]

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

int main(){

    int kwadraten[10];

    for(int i = 0; i < 10; i++){
        kwadraten[i] = i*i;
    }

    int som = 0;
    for(int i = 0; i < 10; i++){
        som += kwadraten[i];
    }

    cout << "de som is " << som << endl;

    return 0;
}
```

10 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

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

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.

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

int main(){
    cout << "alle kwadraten onder 1000" << endl;

    int n = 0;
    while(n * n <= 1000){
        cout << (n * n) << endl;
        n = n+1;
    }
}
```

11 Tips: Hoe nu verder?

In twee uur kunnen we je helaas niet alles leren wat er te leren valt over `C++` , dus hier komen een aantal tips die je kunt gebruiken als je zelfstandig verder wilt gaan met leren programmeren in `C++` .

- Raak niet in paniek als je foutmeldingen krijgt, ook niet als het er heel veel zijn. Het is heel normaal om fouten te maken (bijvoorbeeld typefouten) en dit is hoe de compiler je waarschuwt voor deze fouten Kijk in welk regelnummer de fout zich heeft voorgedaan (zie opdracht) en google de specifieke foutmelding als je niet weet wat deze betekent. Soms kan één kleine typefout al een hele reeks foutmeldingen genereren, maar laat je daardoor niet afschrikken!

- Soms kan het gebeuren dat je een foutmelding toch niet kan vinden, ondanks het regelnummer dat staat aangegeven bij de foutmelding. Het kan handig zijn om dan specifieke delen van je programma waarvan je vermoedt dat ze de fout veroorzaken even uit te zetten met behulp van commentaar (dus `//` of `/* */`). Als het programma het dan wel doet, dan weet je dat de fout zich bevindt in de regels die nu in het commentaar staan!

- Er is ontzettend veel te vinden over `C++` (en ook andere programmeertalen) op internet. Eigenlijk is er maar één goede tactiek als je nog meer wilt leren: google je suf![3]

- Handige links zijn onder andere:

  1. `http://stackoverflow.com`, een erg groot forum waar op al je vragen een antwoord te vinden is. Meestal zijn dit de nuttige resultaten die je via google vindt.
  2. `http://cppreference.com`, een online `C++` documentatie. Dit is vooral handig als je wilt weten wat een functie, waarvan je de naam al weet, precies doet.
  3. `http://cplusplus.com`, een andere documentatie, probeer ze allebei, en kijk welke je fijner vindt.

- Als je een compiler wilt installeren op een Windows computer dan zijn handige alternatieven: Code::Blocks en Visual Studio Express 2013 for Windows Desktop. Er zijn van Visual Studio ook gratis versies voor studenten.

- Als je twijfelt of je berekeningen goed gaan kan het handig zijn om als tussenstappen de waarde van een variabele te printen met `cout`. Er is ook nog `cerr`, wat speciaal gemaakt is voor errors, maar verder op precies dezelfde manier gebruikt wordt.

- Als het je leuk lijkt om een programmeervak te gaan volgen, kun je eens naar deze vakken gaan kijken: bij natuurkunde 'Numerieke Methoden' (C en Mathematica), bij wiskunde 'Programmeren in de wiskunde' (Python) en bij informatica 'Imperatief programmeren' (C#, eerstejaarsvak, dus goed te doen voor natuurkundigen ook!). Geen van deze vakken wordt in `C++` gegeven, maar ze worden wel in soortgelijke

[3] Ook ik (Ragnar) google tijdens het programmeren ruim 10 keer per uur naar kleine dingen. Als je iets even niet meer precies weet: Meteen opzoeken. Op die manier moet je dingen soms meerdere keren opzoeken, maar uiteindelijk onthoud je ze vanzelf.

talen gegeven. Als je eenmaal in een taal kunt programmeren is het heel makkelijk om over te stappen naar een andere taal, dus ook deze vakken kunnen je uiteindelijk zeker helpen om beter te worden in `C++` .

- We hebben express te veel opdrachten gemaakt voor vandaag, dus niet getreurd, je kunt thuis gewoon verder gaan met oefenen! Daarnaast zijn er op het internet ook heel veel beginnersopdrachten in `C++` te vinden, zoals op: `http://www.worldbestlearningcenter.com/index_files/cpp-tutorial-variables_datatypes_exercises.htm`.