Inleiding programmeren in C++

FysiCie
i.s.m.
Ragnar Groot Koerkamp

18 juni 2015

Inhoudsopgave

1 Overzicht . . . 2
2 Compilers . . . 2
3 Basis . . . 3
4 Console uitvoer . . . 4
5 Variabelen . . . 6
5.1 Strings . . . 6
5.2 Declaratie en initializatie . . . 6
5.3 Rekenen met `int`s en `double`s . . . 7
5.4 Meer operatoren . . . 8
5.5 Booleans . . . 9
6 Console invoer . . . 10
7 If . . . 10
8 For . . . 11
9 Tips: Hoe nu verder? . . . 14

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;
}
```

Deze pdf en alle gebruikte code zijn beschikbaar op `http://github.com/RagnarGrootKoerkamp/cpp-tutorial`. We raden je echt aar om in het begin alle code over te typen uit de pdf, zodat je echt een gevoel krijgt voor waar je mee bezig bent.

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. Een computerprogramma 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 naar machinecode. Voor iedere programmeertaal bestaan er compilers die die specifieke 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 ¡iostream¿ 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`!

Opdrachten

1. Ga naar Cloud9 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*.)

2. Type nu de code uit het bovenstaande voorbeeld over in het bestand `hallo.cpp`, 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.

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

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

5. 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 Cloud9 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 Cloud9. 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 }
```

Opdrachten

1. Neem het bovenstaande voorbeeld over. Maak eerst van de regel waarop 22 wordt weggeschreven commentaar, en maak vervolgens van de twee regels voor 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. 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.

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

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

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

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

8. 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.

5.1 Strings

Strings zijn tekstvariabelen. Een tekstvariabele moet altijd tussen aanhalingstekens “ en ” staan. Bijvoorbeeld:

```
#include <iostream>

int main() {
 string naam = "Piet"; // Deze regel wordt in het volgende stukje
    uitgelegd!
 cout << "Mijn naam is " << naam << endl;
 cout << "Mijn naam is " << "Piet" << endl;
}
```

Opdrachten

- Print ‘Mijn naam is ... ’ en vul op de puntjes je eigen naam in.

5.2 Declaratie en initializatie

Voordat we aan een variabele een waarde kunnen toekennen, moeten we deze *declareren* (Engels: *declaration*). Hierbij moeten we specifiek aangeven van welk datatype (`int`, `double`, `bool`, `string`) een variabele is. Dit datatype staat daarna vast en kan niet meer veranderd worden. Het is verplicht om iedere variabele te declareren en dit declareren mag je maar één keer doen. De syntax van een declaratie is als volgt
`datatype naam;`
Bijvoorbeeld met `int a;` declareer je een variabele genaamd `a` van het datatype `int` (geheel getal). We kunnen met gebruik van de ‘=’-operator een waarde toekennen aan een variabele. Dit heet ook wel de *initialisatie* (Engels: *initialization*). Dat gaat als volgt: `a = 2;`. De variabele `a` heeft nu de waarde 2. Variabelen kunnen op twee manieren gedeclareerd worden. De variabele kan gedeclareerd worden voordat de variabele voor het eerst een waarde toegekend krijgt of de declaratie en het toekennen van de *eerste* waarde gebeuren tegelijk.

```
#include <iostream>
using namespace std;
int main(){
    // Eerst declareren, dan toekenning
    int a;
```

```
6       a = 2;
7       cout << a << endl;
8       // Declaratie en toekenning tegelijk
9       int b = 3;
10      cout << b << endl;
11  }
```

Hoewel een variabele maar één keer gedeclareerd kan worden en het datatype daarna vaststaat, kan wel de waarde van de variabele later in het programma worden aangepast. We kunnen de variabele `a` een nieuwe waarde toekennen op dezelfde manier als waarop hij zijn eerste waarde krijgt.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4       int a;
5       a = 2;
6       cout << "a = " << a << endl;
7       a = 3;
8       cout << "a = " << a << endl; // a heeft nu een nieuwe waarde,
            namelijk 3!
9  }
```

Voor `double`s werkt dit hetzelfde als voor `int`s behalve dat je bij de declaratie moet aangeven dat het een 'double' is in plaats van een 'int'.

Opdrachten

1. Declareer een `double` genaand `pi` en geef hem de waarde 3.1415. Print de waarde van c vervolgens naar de console.

2. Verander de waarde nu naar 3.14159265 en print nogmaals de waarde van `pi` naar de console.

5.3 Rekenen met `int`s en `double`s

We kunnen met `int`s en `double`s rekenen door het gebruik van de operatoren + - * /. De + - * operatoren werken precies zoals je zou verwachten. Onderstaand voorbeeld illustreert dit:

```
1  #include <iostream>
2  using namespace std;
3  int main () {
4       int a = 2;
5       int b = 3;
6       cout << "a + b = " << a + b << endl;
7       cout << "a - b = " << a - b << endl;
8       cout << "a * b = " << a * b << endl;
9  }
```

Bij deling werkt dit iets anders. Bij het delen van gehele getallen wordt het resultaat namelijk altijd naar beneden afgerond. Onderstaand voorbeeld illustreert dit:

```
1  #include <iostream>
2  using namespace std;
3  int main () {
4       int a = 3;
```

```
5       int b = 2;
6       cout << "a/b = " << a/b << endl;
7   }
```

Uit de berekening $\frac{3}{2}$ komt in dit geval 1, omdat 1.5 naar beneden wordt afgerond. Bij een berekening waarbij alleen `int`s worden gebruikt wordt het resultaat van de berekening ook een `int` en bij deze deling wordt 3/2 dus naar beneden afgerond naar de integer 1. Als je minstens één `double` gebruikt bij je berekening wordt het resultaat een `double` en voorkom je die afronding naar beneden (als je dat niet wilt tenminste).

```
1   #include <iostream>
2   using namespace std;
3   int main () {
4       double c = 2;
5       double d = 3;
6       cout << "c/d = " << c/d << endl;
7   }
```

Opdrachten

1. Een fietsen doet er 10 seconde over om een afstand van 75 meter af te leggen. Print zijn snelheid in meter per seconde naar de console, door deze met een deling uit te rekenen.

2. Wat is de snelheid in kilometer per uur? (Weer door het uit te programmeren!)

5.4 Meer operatoren

Het gebeurt vaak dat je een variabele wilt ophogen met een vaste waarde. Stel je wilt de variabele `a` uit het vorige voorbeeld met 2 ophogen, dan kun je dat als volgt doen: `a = a + 2`. Dit ziet er misschien raar uit, maar wat hier staat (in computertaal) is: "De nieuwe waarde van a *wordt* gelijk aan de oude waarde van a plus 2". Hierin zijn de `a` voor het gelijkheidsteken en de `a` na het gelijkheidsteken dus niet hetzelfde, hoewel het om dezelfde variabele gaat. De *waarde* van `a` voor het gelijkheidsteken betreft de nieuwe waarde van `a` en de `a` na het gelijkheidsteken betreft zijn oude waarde. Een verkorte notatie voor deze ophoging is `a += 2`. Deze verkorte notatie kan ook gebruikt worden voor -, * en /. Specifiek voor als je een variabele wilt ophogen met de waarde 1 bestaat er een nog kortere notatie: `a = a + 1` is hetzelfde als `a++`. Analoog hieraan verlaagt `a--` de waarde van `a` met 1.

```
1   #include <iostream>
2   using namespace std;
3   int main ()
4   {
5       int a = 2;
6       a = a + 3;
7       cout << a << endl; // a wordt met 3 opgehoogd
8       a += 3; // a wordt weer met 3 opgehoogd
9       cout << a << endl;
10  }
```

Bovenstaande geldt allemaal ook voor doubles. Wat nu als je een wortel of een sinus wilt gebruiken in je berekening? Daarvoor hebben we het pakketje `<cmath>` nodig dat veel handige functies bevat zoals `sqrt`, `sin`, `cos`, `exp`, `log` etc. Een voorbeeldje:

```
#include <iostream>
#include <cmath>
using namespace std;
int main () {
    double getal = 25;
    cout << sqrt(getal) << endl;
}
```

Opdrachten

1. Print de sinus van $\pi/2$ (zoek op 'cmath C++' om alle functies te vinden die in dit pakketje zitten, en kies de juiste!).

5.5 Booleans

Een boolean of `bool` is een variabele die slechts twee waarden kan aannemen, `true` of `false` (ook wel aangeduid met respectievelijk 1 en 0). Bij het testen of twee variabelen gelijk zijn aan elkaar is het resultaattype een boolean, ze zijn ofwel gelijk aan elkaar (true) of niet gelijk aan elkaar (false).

```
#include <iostream>
using namespace std;
int main() {
    bool t = true;
    bool f = false;
    // bools worden geprint als 1 of 0!
    cout << "t:\t" << t << endl; // \t zorgt voor een tab
    cout << "f:\t" << f << endl;
}
```

We kunnen testen of twee variabelen gelijk zijn aan elkaar met een dubbel gelijkheidsteken (`==`).

```
#include <iostream>
using namespace std;
int main () {
    cout << (2 == 2) << endl;
    double x = 3.2;
    double y = 4.5;
    cout << (x == y) << endl;
}
```

Naast de logische operator `==` voor het testen van gelijkheid bestaan ook de operator `!` die staat voor logische negatie. Dat betekent dat `!false` gelijk is aan `true`, en dat `!true` weer `false` wordt. Met `!=` kun je testen of twee dingen *niet* gelijk zijn aan elkaar.

```
#include <iostream>
using namespace std;
int main() {
    double t = true;
    cout << !t << endl; // dit geeft dus false (0)
    double x = 3.2;
    double y = 4.5;
    cout << (x != y) << endl; // dit gaf net false, nu geeft het true
        (1)!
}
```

Opdrachten

1. Test of de sinus van $\pi/2$ en de cosinus van $\pi/2$ gelijk zijn aan elkaar.
2. Test ook of de sinus van $\pi/4$ en de cosinus van $\pi/4$ gelijk zijn aan elkaar.

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 dat 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:

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

int main(){
    cout << "Hoe oud ben je?" << endl;
    int leeftijd;
    cin >> leeftijd;
    cout << "Je bent " << leeftijd << " jaar oud" << endl;
}
```

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 de 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;
    }
}
```

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[1]. 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 bepaalt 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 deze iteraties wordt de code tussen de accolades uitgevoerd, in dit geval gebeurt dat dus 10 keer.
Hieronder staat dezelfde 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
            }
        }
    }
}
```

[1]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

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.

Opdrachten

1. Print nu 5 keer `"hoera"` op een nieuwe regel.
2. Print nu alle getallen van 0 tot en met 10 *van groot naar klein*.
3. Doe nu hetzelfde, maar sla dan het getal 3 over. Dit kan je doen met een `if`-statement.
4. Schrijf nu alleen de even getallen tot en met 10 op.

Bonus Je kan een getal ook overslaan door `continue` te gebruiken. Probeer eens op internet op te zoeken hoe dit werkt, en pas het toe.

5. Bekijk nog eens de code uit de inleiding, en probeer te achterhalen wat het doet en hoe het werkt.

9 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. Bij een foutmelding staan altijd twee getallen vermeld: eerst het regelnummer en dan het kolumnummer waar de fout optreed, dit helpt je om de fout te zoeken. Als je niet weet wat een bepaalde foutmelding betekent kun je deze googlen.

- 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 `/* */`) om te testen of die regels de foutmelding veroorzaken.

- 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![2]

- 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[3]. Voor Linux computers is *g++* de meest gebruikte compiler. Meestal wordt deze standaard meegeïnstalleerd, en hoef je in de terminal alleen maar `g++ programma.cpp` in te tikken. De uitvoer heet dan `a.out`.

- 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.

[2] 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.

[3] Deze is verkrijgbaar via DreamSpark: ga naar `students.uu.nl/en/free-software` en klik “Microsoft Software”.

- 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 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. Kijk bijvoorbeeld eenss op: `http://www.worldbestlearningcenter.com/index_files/cpp-tutorial-variables_datatypes_exercises.htm`.