

数值分析及代码手册

Cpp etc

姜青娥

ImageNature

你不必知道所有.

©2023 姜青娥, ImageNature

Versie: ver 1

Datum: June 2, 2023

DE HAAGSE
HOGESCHOOL

De auteur kan niet aansprakelijk worden gesteld voor enige schade, in welke vorm dan ook, die voortvloeit uit informatie in dit boek. Evenmin kan de auteur aansprakelijk worden gesteld voor enige schade die voortvloeit uit het gebruik, het onvermogen tot gebruik of de resultaten van het gebruik van informatie in dit boek.

Dit boek mag overal voor gebruikt worden, commercieel en niet-commercieel, mits de wijzigingen worden gedeeld en naam van de originele auteur wordt gemeld. De broncode van dit boek is beschikbaar op https://github.com/jesseopdenbrouw/book_c.



数值分析及代码手册 Cpp etc van 姜青娥 is in licentie gegeven volgens een [Creative Commons Naamsvermelding-NietCommercieel-GelijkDelen 3.0 Nederland-licentie](#).

Suggesties en/of opmerkingen over dit boek kunnen worden gemaïld naar: imagenature@proton.me.

前言

Een van de beste boeken over C is *The C Programming Language* van Brian Kernighan en Dennis Ritchie [kernighan1988c].

The book is not an introductory programming manual; it assumes some familiarity with basic programming concepts like variables, assignment statements, loops, and functions. [...] C is not a big language, and it is not well served by a big book.

Dit boek is opgemaakt in L^AT_EX [latexwebsite] (L^AT_EX-engine = XeL^AT_EX 0.999995). L^AT_EX leent zich uitstekend voor het opmaken van lopende tekst, tabellen, figuren, programmacode, vergelijkingen en referenties. De gebruikte L^AT_EX-distributie is TexLive uit 2021 [texlivewebsite]. Als editor is TexStudio [texstudiowebsite] gebruikt. Tekst is gezet in Calibri [calibrifont], een van de standaard fonts op een bekend besturingssysteem. Code is opgemaakt in Consolas [consolasfont] met behulp van de *listings*-package [listingsctan]. Voor het tekenen van arrays, pointers en flowcharts is *TikZ/PGF* gebruikt [tikzctan]. Alle figuren zijn door de auteur zelf ontwikkeld, behalve de logo's van Creative Commons en De Haagse Hogeschool. Een aantal programmafragmenten is ontwikkeld door collega Harry Broeders, de overige fragmenten zijn door de auteur ontwikkeld.

van dit boek is beschikbaar op https://github.com/jesseopdenbrouw/book_c. We doen een klemmend beroep aan alle lezers om fouten en omissies te melden.

姜青娥, juni 2023.

Contents

前言	iii
1 Cpp 基础	1
1.1 De compiler	3
1.2 Ontwikkelssystemen	4
1.3 Een minimaal C-programma	5
1.4 Afdrukken op het scherm	7
1.5 Invoer van het toetsenbord	9
1.6 Keywords	11
1.7 Beslissingen	11
1.8 Herhalingen	12
1.9 Functies	13
1.10 Arrays	16
1.11 Strings	17
1.12 Programmeerstijlen	17
1.13 Commentaar	19
1.14 En verder...	20

1

C++ 基础

C is al een oude taal. De taal is rond 1970 ontworpen door Dennis Ritchie¹ en is dus al zo'n 53 jaar oud. Hij was bezig met het schrijven van een besturingssysteem (Engels: operating system)² en zocht naar een mogelijkheid om dit te schrijven op een hoger niveau dan gebruikelijk was voor die tijd. In die tijd werden besturingssystemen geschreven in assembly, een taal die dicht bij de hardware van de computer ligt. Programma's schrijven in assembly is tijdrovend en gevoelig voor fouten van de programmeur.

C is een zogenoemde derde-generatietaal (3GL) en zorgt ervoor dat programma's gestructureerd kunnen worden geschreven zonder dat de programmeur kennis hoeft te hebben van de hardware waarop het programma draait. Toch biedt de taal constructies om die hardware in te stellen, een voorwaarde voor het schrijven van een besturingssysteem. Daarom is de taal geliefd bij programmeurs van kleine computersystemen waarop geen besturingssysteem draait, de zogenoemde *bare metal systems*. Het is vaak ook de enige taal die gebruikt kan worden op dit soort systemen, naast assembly.

C is geschreven voor ervaren programmeurs die behoefte hebben aan het schrijven van compacte programma's. Dat is te zien aan de soms onduidelijke taalconstructies. C is daarom lastig voor beginnende programmeurs. Maar het is, zoals gezegd, vaak de enige taal die op een bepaald platform gebruikt kan worden en met enige discipline kan ook de minder ervaren programmeur de taal prima gebruiken.

¹ Dennis MacAlister Ritchie (1941 – 2011). Hij was de ontwerper van de programmeertaal C en was een van de ontwerpers van Unix. Bekende afgeleiden van Unix zijn Linux en FreeBSD.

² Een besturingssysteem is een programma dat draait op een computer en zorgt voor het beschikbaar stellen van *resources* aan de gebruiker (lees: programma's). Zo zorgt een besturingssysteem ervoor dat de bestanden op de harde schijf netjes worden opgeslagen en beschikbaar zijn. Daarnaast zorgt een besturingssysteem ervoor dat programma's netjes naast elkaar kunnen draaien. Drie bekende besturingssystemen zijn Windows, Linux en Apple's OS-X.

C is een algemeen bruikbare taal (*general purpose language*) en is dus niet voor een specifiek doeleind ontworpen (behalve voor het schrijven van besturingssystemen). Zo kunnen met C wiskundige berekeningen worden uitgevoerd maar de programmeur moet zelf alles schrijven. Er zijn wel *functies* beschikbaar voor bijvoorbeeld sinus, cosinus en tangens. Er zijn echter talen die dit veel beter ondersteunen. Ook het verwerken van bestanden en *strings* (een rij karakters) is mogelijk maar dat moet door de programmeur zelf worden uitgewerkt. Ook hier zijn diverse functies beschikbaar die het programmeerwerk enigszins verlichten.

C is een *imperatieve* programmeertaal, een van de bekende programmeerparadigma's³.

Een computer heeft naast de processor *geheugen* om de data en instructies op te slaan. Er zijn twee soorten geheugens: ROM (Read Only Memory) is geheugen waarvan de inhoud niet gewijzigd kan worden; RAM (Random Access Memory) is geheugen waarvan de inhoud wel gewijzigd kan worden. We zullen zo meteen zien waarvoor ROM en RAM worden gebruikt.

Naast ROM en RAM heeft de computer nog invoer- en uitvoermogelijkheden, anders is er geen communicatie met de buitenwereld mogelijk. De verzamelnaam is *I/O* dat "Input" en "Output" betekent. Het is best lastig om de invoer en uitvoer te realiseren. Daarom wordt bij de gangbare computers een stuk software geladen (of het is al aanwezig in de ROM) om dat voor de gebruiker (en programmeur) te vereenvoudigen. Die software wordt een *besturingssysteem* genoemd, maar gangbaar is om de Engelse naam *operating system* te gebruiken. Bekende besturingssystemen zijn Windows, Linux en Apple's macOS.

Niet alle gegevens zijn in het geheugen van de computer aanwezig. Een computer heeft (vaak) *secundair geheugen*. Voorbeelden van secundair geheugen zijn harddisk (harde schijf), USB-sticks en SD-cards. De informatie op deze geheugendragers blijft aanwezig ook al wordt de computer

BITS AND BYTES, SIZE DOES MATTER...

De meeste gangbare computersystemen slaan data op in *bytes*. Een byte is een eenheid van 8 *bits*. Het woord *bit* is een samentrekking van *binary digit*. Binary digit betekent *binair cijfer* en binair betekent *tweewaardig*. Dat betekent dat een bit twee verschillende waarden kan hebben. We noemen die waarden 0 of 1. Met behulp van bits en bytes kunnen we getallen representeren. Het kleinste getal in een byte is 00000000_2 waarbij het subscript 2 aangeeft dat het om een binair getal gaat, en het grootste getal is 11111111_2 . Het getal 00000000_2 komt overeen met het decimale getal 0 en 11111111_2 komt overeen met 255. Om grotere getallen te representeren, zijn meerdere bytes nodig. Moderne systemen kunnen *van nature* overweg met 32-bits eenheden, ze kunnen met 32 bits als één eenheid rekenen. Daarvoor zijn 4 bytes nodig. We spreken dan van een *32-bits systeem*. Daarnaast kunnen veel systemen ook overweg met 64-bits eenheden, dus zijn er 8 bytes nodig voor zo'n eenheid. We noemen dat een *64-bits systeem*.

³ Een programmeerparadigma is een manier van programmeren en een wijze waarop een programma wordt vormgegeven.

uitgezet en weer aangezet. Om de informatie beschikbaar te maken worden de gegevens in *bestanden* (Engels: files) gezet. De manier waarop dat wordt gerealiseerd wordt een *bestandssysteem* (Engels: file system) genoemd. Bekende bestandssystemen zijn NTFS op Windows en ext3 op Linux. Het besturingssysteem zorgt ervoor dat de bestanden op ordentelijke wijze kunnen worden benaderd.

In de praktijk komen we een drietal computersystemen tegen. Natuurlijk zijn er de algemeen bruikbare computers waarop een besturingssysteem draait (PC, laptops met Windows, Linux, Apple's macOS) en die een grote hoeveelheid aan verschillende programma's kan uitvoeren.

Daarnaast zijn er zogenoemde *embedded systems*. Een embedded system is meestal geschikt om één taak uit te voeren en is voorzien van een kleine processor met geheugen en I/O-faciliteiten. Al deze componenten zijn op één ic (Integrated Circuit of chip genoemd) geplaatst. We noemen zo'n processorsysteem een *microcontroller*. Voorbeelden van embedded systems zijn (de besturingen van) wasmachines, koelkasten, televisies, thermostaten, horloges en bloeddrukmeters. De markt voor embedded systems is trouwens vele malen groter dan de markt voor algemeen bruikbare computers. De microcontroller heeft meestal geen besturingssysteem. Dat noemen we *bare metal* systemen. Dat betekent dat de programmeur alles zelf moet ontwikkelen. Gelukkig leveren fabrikanten kant-en-klare ontwikkelsystemen om programma's voor microcontrollers te produceren. Op deze systemen is C (en C++) de enige taal die gebruikt kan worden. We gaan in dit boek hier verder niet op in.

We kunnen natuurlijk niet de *smartphones* vergeten. Zo'n beetje iedereen in Nederland heeft er een. We kunnen een smartphone vergelijken met een algemeen bruikbaar computersysteem, waarop de besturingssystemen Android of iOS draaien. Op deze systemen kunnen programma's worden geïnstalleerd, *apps* genaamd. Deze apps worden over het algemeen *niet* in C geprogrammeerd, en vallen verder buiten het bestek van dit boek.

Bij PC's en laptops is een klein programma in ROM aanwezig. Dit zorgt ervoor dat de PC kan opstarten. Het heeft tot doel om het besturingssysteem te laden (Windows, Linux). Het besturingssysteem en de programma's worden dus in RAM geladen, net als de data waarmee de programma's werken. Op smartphones is het besturingssysteem opgeslagen in ROM die herprogrammeerbaar is. Zo kunnen updates van het besturingssysteem geïnstalleerd worden. De apps en data worden geladen in de RAM. Bij microcontrollers is het complete programma geladen in ROM, die ook herprogrammeerbaar is. De data is geplaatst in RAM.

1.1 De compiler

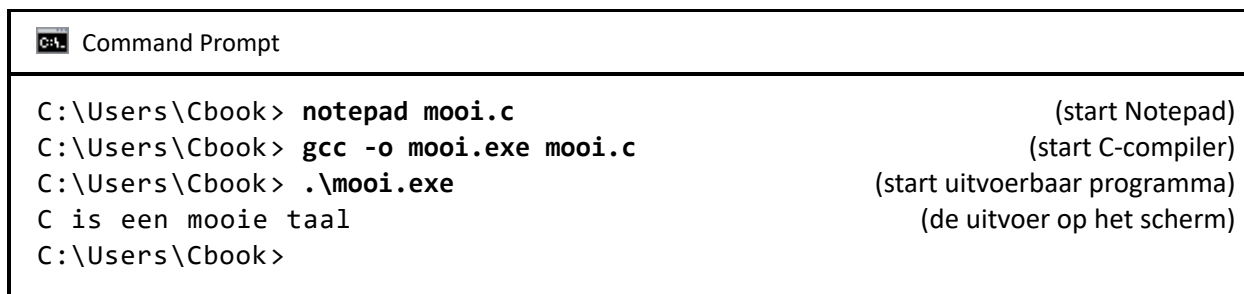
C is een zogenoemde derde-generatie-taal. De broncode van een C-programma is voor een mens gewoon te lezen. Deze broncode kan niet direct door de processor worden uitgevoerd. De *C-compiler* vertaalt de broncode naar instructies die de processor wel kan uitvoeren. Dat vertalen wordt *compileren* genoemd. De instructies worden in bitpatronen in het geheugen opgeslagen. De bitpatronen noemen we *machinecode*. Elk type processor heeft zijn eigen verzameling bitpatronen, dat wordt de *Instruction Set Architecture* genoemd. De ISA van een Intel-processor is dus anders dan die van een ATmega-processor, maar we kunnen wel zeggen dat elke processor

ongeveer dezelfde soort instructies bevat. De C-compiler moet weten voor welke processor een C-programma wordt vertaald.

Een C-programma kan niet worden uitgevoerd door de computer. We moeten eerst een programma compileren zodat instructies worden gegenereerd die de processor wel kan uitvoeren. De verzameling programma's om een C-programma te compileren wordt een *toolchain* genoemd. Het vertaalde programma wordt een *uitvoerbaar bestand* of *executable* genoemd. Een uitvoerbaar bestand wordt net als een C-programma opgeslagen op de *vaste schijf* (SSD, USB-stick, ...). Bij het ontwikkelen van programma's voor microcontrollers wordt gebruik gemaakt van PC of laptop met een besturingssysteem en een C-compiler specifiek voor de microcontroller. We noemen zo'n compiler een *cross compiler*. De uitvoerbare instructies worden via de PC in het geheugen van de microcontroller *geprogrammeerd*.

1.2 Ontwikkelssystemen

De meeste boeken gaan uit van een *command line interface* op een Unix-derivaat. Met behulp van een *editor* wordt een programma ingevoerd en wordt op de commandoregel de C-compiler gestart. Daarna wordt het programma (ook via de commandoregel) gestart. Natuurlijk is het ook mogelijk om alles met behulp van de command line te doen. Deze werkwijze wordt veel gebruikt op Linux-systemen maar kan ook op Windows worden gebruikt. Een voorbeeld is te zien in figuur 1.1.



```
C:\Users\Cbook> notepad mooi.c (start Notepad)
C:\Users\Cbook> gcc -o mooi.exe mooi.c (start C-compiler)
C:\Users\Cbook> .\mooi.exe (start uitvoerbaar programma)
C is een mooie taal (de uitvoer op het scherm)
C:\Users\Cbook>
```

Figuur 1.1: Een voorbeeld van een command line interface.

Dit is echter niet de werkwijze van veel programmeurs. Gelukkig zijn er goede ontwikkelssystemen (IDE: Integrated Development Environment) die het programmeerwerk verlichten. Bekende systemen zijn Microsoft Visual Studio [**vs2022**], Code::Blocks [**codeblocks2020**] en Apple's Xcode [**xcode2020**]. Zulke ontwikkelssystemen zorgen ervoor dat de programmeur gemakkelijk het programma kan invoeren, de compiler kan starten en het programma kan *debuggen*. Dat laatste is vaak nodig als blijkt dat de executie van een programma niet verloopt zoals de programmeur het voor ogen had. Met debuggen wordt het programma stap voor stap doorlopen en kan de programmeur (of is het debugger) de inhoud van *variabelen* bekijken. Ook kan de programmeur bepalen of de *statements* (opdrachten voor de computer) op de juiste volgorde worden uitgevoerd.

Een voorbeeld van Microsoft Visual Studio is te zien in figuur 1.2. Visual Studio ondersteunt het ontwikkelen van software voor Windows-computers met behulp van een groot aantal talen: C,

C++, C#, Python. Het is zelfs mogelijk om software te ontwikkelen voor Linux-systemen. Een korte introductie wordt gegeven in bijlage ??.

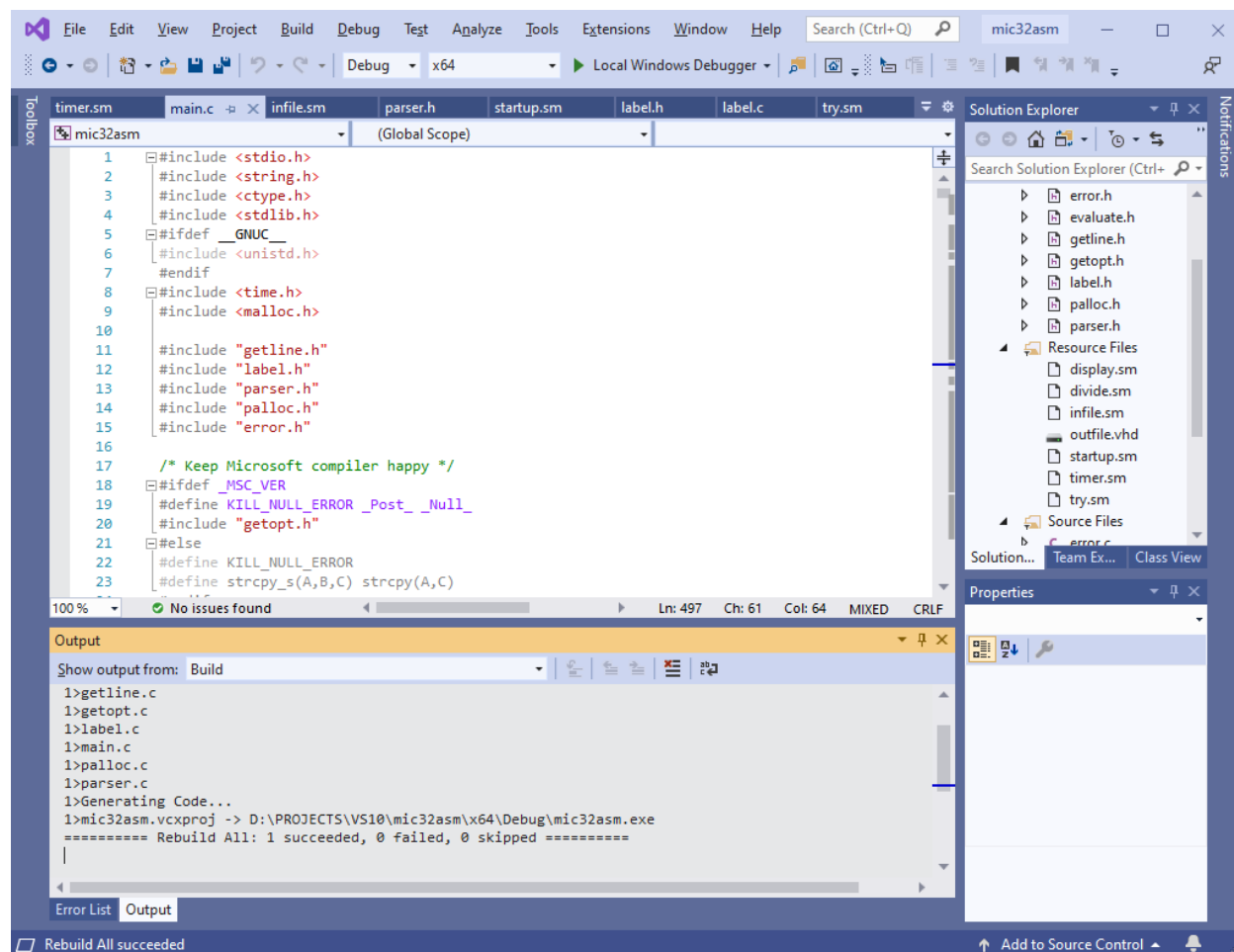


Figure 1.2: Voorbeeld van Microsoft Visual Studio.

We hebben nu een beeld van hoe op een computer een C-programma wordt vertaald naar instructies voor een computer. De computer kan het C-programma niet direct uitvoeren, het programma moet gecompileerd worden. Als we een wijziging in het C-programma willen doorvoeren, dan moeten we het C-programma uiteraard opnieuw compileren.

We zullen in de overige paragrafen in sneltreinvaart enkele concepten van C behandelen. In de volgende hoofdstukken wordt de taal verder uitgediept.

1.3 Een minimaal C-programma

We beginnen met het meest simpele C-programma dat mogelijk is. We willen hiermee uitleggen wat er gebeurt als dit programma gecompileerd en uitgevoerd wordt. Het programma is te zien in listing 1.1. Het programma doet in feite helemaal niets, althans niet dat de gebruiker van het programma kan waarnemen. Toch gebeuren er wel degelijk dingen 'onder de motorkap'.

```

1 int main(void)
2 {
3     return 0;
4 }

```

Listing 1.1: Een minimaal C-programma.

Dit programma kan niet direct door de computer worden uitgevoerd, het moet eerst worden gecompileerd. Na compilatie is een *uitvoerbaar bestand* beschikbaar dat wel door de computer kan worden uitgevoerd. De werking op een PC of laptop is als volgt. Als het uitvoerbare programma wordt gestart, dan zal het besturingssysteem het uitvoerbare programma in het geheugen van de computer laden. Dus ergens in het geheugen van de computer liggen de instructies die het programma vormen opgeslagen. Het besturingssysteem start het uitvoerbare programma door de processor naar de eerste instructie van het programma te leiden. Het programma (dat is het uitvoerbare programma) zal nu instructies uitvoeren. Na afloop van het programma wordt de besturing weer teruggegeven aan het besturingssysteem.

Het C-programma begint met de definitie van de *functie* *main*. Een functie is een aantal instructies samengepakt onder een gemeenschappelijke noemer. *Elk C-programma heeft een functie main*. Tussen de haken staat het keyword *void*, dat aangeeft dat het uitvoerbare programma geen gegevens meekrijgt van het besturingssysteem⁴. Vóór *main* staat het keyword *int* dat aangeeft dat *main* een geheel getal teruggeeft aan het besturingssysteem.

Statements die in het C-programma gebruikt worden, zijn afgebakend met *accolades*. Het Engelse woord hiervoor is *curly braces* of gewoon *braces*. De accolades geven het begin en einde aan van *blok*. Een blok begint met een accolade-openen ({) en eindigt met een accolade-sluiten (}). Over de plaats van de accolades zijn er diverse meningen. In listing 1.1 is de accolade-openen geplaatst onder de definitie van *main*, maar het is ook gebruikelijk om de accolade-openen te schrijven achter de definitie van *main*.

Binnen *main* zien we één *statement*. Een statement is een opdracht in de C-taal. Het statement wordt gevormd door het keyword *return* gevolgd door het getal 0 en een punt-komma. Bij uitvoering van dit statement wordt de waarde 0 teruggegeven aan het besturingssysteem. Dat behoeft enige uitleg. Een (gecompileerd) programma wordt gestart door het besturingssysteem. Aan het einde wordt het programma afgesloten. Het besturingssysteem “ruimt” het programma op en zorgt ervoor dat het gebruikte geheugen weer vrijgegeven wordt voor volgende programma’s. We kunnen aan het besturingssysteem een getal teruggeven, in dit geval 0. Het is aan het besturingssysteem om hier wat mee te doen. Gebruikelijk is om 0 terug te geven als alles goed verlopen is. Een ander getal dan 0 geeft over het algemeen aan dat er iets fout gegaan is. Vanaf C99 is het niet meer nodig om dit *return*-statement uit te voeren. Dan wordt automatisch het getal 0 teruggegeven.

⁴ Dat kan wel, zie hoofdstuk ??.

1.4 Afdrukken op het scherm

We kunnen het eerste programma interessanter maken door een regel tekst op het scherm af te drukken. Het programma in listing 1.2 drukt de regel *De som van 3 en 7 is 10* op het scherm af. We zullen het programma stap voor stap doorlopen.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 3;
6     int b = 7;
7
8     int som;
9
10    som = a + b;
11
12    printf("De som van %d en %d is %d\n", a, b, som);
13
14    return 0;
15 }
```

Listing 1.2: Afdrukken van de som van twee getallen.

In regel 1 wordt een zogenoemd *header-bestand* geladen, in dit geval het bestand `stdio.h`. We leggen zo meteen uit waarom dat nodig is.

In regel 3 wordt kenbaar gemaakt dat het programma de functie `main` heeft. Een C-programma heeft *altijd* de functie `main`. Het (gecompileerde) programma wordt hier gestart. Het programma wordt gestopt na het laatste statement in `main`. Dit is per definitie een `return`-statement.

In regel 5 en 6 worden twee *variabelen* gedefinieerd, de variabelen `a` en `b`. Technisch gezien is een variabele een plek in het geheugen van de computer. Een variabele kan door het programma gebruikt worden om gegevens te bewerken. Definitie wil zeggen dat een variabele kenbaar wordt gemaakt én dat er geheugenruimte wordt gereserveerd. Bij de definitie wordt opgegeven wat het type is van de variabele. Dit gebeurt middels het keyword `int`, dat betekent dat `a` en `b` alleen gehele getallen kunnen opslaan (Engels: *integer*). Aan de variabelen worden gelijk waarden toegekend. We noemen dit *initialisatie* van variabelen.

In regel 8 wordt de variabele `som` gedefinieerd zonder initialisatie. Dat betekent dat op dat moment de waarde (of inhoud) van de variabele *onbekend* is. Vervolgens wordt in regel 10 de som van `a` en `b` berekend middels de *optel-operator* `+` en wordt het resultaat *toegekend* aan variabele `som`. Vanaf regel 10 is de waarde van `som` dus 10. In regel 10 zien we zogenoemde *expressies*. Zo is `a + b` een expressie en is de toekenning `som = ...` ook een expressie. De term *expressie* komt veel voor in C.

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     double var1 = 3.2;
7     int var2 = 3;
8
9     for (int i = 0; i < var2; i++) {
10         cout << i << endl;
11     }
12
13     return 0;
14 }

```

Listing 1.3: *An very important code*

In regel 12 wordt de functie `printf` aangeroepen. Deze functie is al geschreven en zit in de standard library. De functie krijgt vier *argumenten* mee, gegevens die in de functie verwerkt worden. Het eerste argument is een *string*. Een string is een stukje tekst, maar we spreken ook wel van een rij karakters. Daarna volgen de (waarden) van de variabelen `a`, `b` en `som`.

Het eerste argument van `printf` wordt een *format string* genoemd. Deze format string bevat karakters, zogenoemde *format specifications* en een *escape sequence*. Een format specification begint met een procentteken gevolgd door een letter. De functie `printf` gebruikt deze format specifications om de gegevens af te drukken. De eerste `%d` zorgt ervoor dat variabele `a` wordt afgedrukt als een decimaal geheel getal. Op overeenkomstige wijze worden ook `b` en `som` afgedrukt. Aan het einde van de format string is een *escape sequence* te zien, in dit geval `\n`. Dit zorgt ervoor dat een volgende afdruk wordt begonnen aan het begin van de volgende regel (Engels: *newline*).

In regel 14 wordt met het keyword `return` aangegeven dat het programma wordt afgesloten. Dat de return-waarde een geheel getal moet zijn, kunnen we zien aan de definitie van de functie `main`. We zien in regel 3 dat `main` een geheel getal teruggeeft (keyword `int`) en geen argumenten meekrijgt (keyword `void`).

Hoe weet de C-compiler nu hoe de functie `printf` moet worden aangeroepen? Dat wordt geregeld met een *function prototype*. We hoeven dat zelf niet op te geven want dat is al gedaan en is te vinden in het header-bestand `stdio.h`. De eerste regel geeft dus aan dat dit bestand geladen moet worden. De tekens `<` en `>` geven aan dat gezocht moet worden op een bepaalde plek in het bestandssysteem. We hoeven dat verder niet te weten.

1.5 Invoer van het toetsenbord

We kunnen het vorige programma interessanter maken door aan de gebruiker te vragen om twee gehele getallen in te voeren. Naast het afdrukken van tekst met de functie `printf` maken we nu ook gebruik van de functie `scanf` om gehele getallen in te lezen. Het programma is te zien in listing 1.4.

```
1 #include <stdio.h>
2
3 /* Make Visual Studio happy */
4 #pragma warning(disable : 4996)
5
6 int main(void)
7 {
8     int a;
9     int b;
10
11     int som;
12
13     printf("Geef een getal: ");
14     scanf("%d", &a);
15
16     printf("Geef nog een getal: ");
17     scanf("%d", &b);
18
19     som = a + b;
20
21     printf("De som van %d en %d is %d\n", a, b, som);
22
23     return 0;
24 }
```

Listing 1.4: Programma om de som van twee getallen te bepalen.

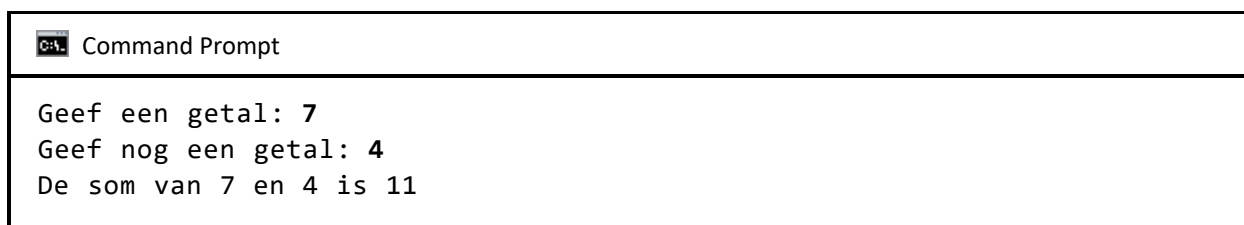
In regel 1 laden we weer het header-bestand `stdio.h`. Dit header-bestand is nodig om de functie-prototypes van `printf` en `scanf` te laden. In regel 4 maken we gebruik van een *pragma*. Dit is een aanwijzing voor de C-compiler om iets te doen of te laten. Waarom deze *pragma* nodig is, kunnen we lezen in het kader op pagina 10.

Het programma volgt verder de lijn van listing 1.2. In de functie `main` definiëren we drie variabelen. Daarna drukken we in regel 13 een stukje tekst af dat aangeeft wat de gebruiker moet doen. In regel 14 wordt de functie `scanf` aangeroepen die ervoor zorgt dat een geheel getal wordt ingelezen van het toetsenbord en in variabele `a` wordt gezet.

De format specification `%d` hadden we al eerder gezien, maar de constructie `&a` is nieuw. De ampersand (`&`) zorgt ervoor dat aan de functie `scanf` het *adres* van variabele `a` meegegeven wordt.

Het adres van de variabele is de plek waar de variabele in het geheugen ligt. Op deze manier kan `scanf` de informatie op je juiste plek zetten. In regel 17 wordt hetzelfde gedaan voor variabele `b`. We drukken voor het inlezen nog even netjes af hoe de gebruiker moet handelen. In regel 19 rekenen we de som uit van variabelen `a` en `b` en kennen die toe aan variabele `som`. In regel 21 drukken we de drie variabelen af. Het programma wordt afgesloten in regel 23.

Als we het programma starten dan moeten we twee gehele getallen invoeren. Een mogelijke uitvoer van het programma is te zien in figuur 1.3. De invoer van de gebruiker is vet afgedrukt. Overigens zullen ontwikkelsystemen zoals Visual Studio aan het einde van het programma wachten tot de gebruiker op een toets drukt, anders is door de snelheid van het uitvoeren van het programma niet te zien wat is afgedrukt.



```
C:\> Command Prompt

Geef een getal: 7
Geef nog een getal: 4
De som van 7 en 4 is 11
```

Figuur 1.3: Uitvoer van het programma in listing 1.4.

We hebben nu gezien hoe we invoer en uitvoer voor een programma kunnen gebruiken. In veel voorbeelden zullen we hiervan gebruik maken.

TO `scanf` OR NOT TO `scanf`...

De Microsoft C-compiler bestempelt `scanf` als “onveilig”. Een compilatie met `scanf` zal eindigen met een foutmelding. In plaats daarvan moet de functie `scanf_s` worden gebruikt. Helaas ondersteunen andere compilers deze functie niet. Dat zal resulteren in een programma dat niet door iedere compiler kan worden vertaald. Om het probleem te omzeilen hebben we gebruik gemaakt van een *pragma*:

```
#pragma warning(disable : 4996)
```

We geven aan dat de C-compiler fout 4996 moet negeren. Op andere compilers, bijvoorbeeld de GNU-C compiler, wordt deze regel overgeslagen (er volgt wel een waarschuwing en met behulp van *conditionele compilatie* kan deze *pragma* overgeslagen worden). Overigens wordt op vele fora gewaarschuwd voor de onveiligheid van `scanf` en worden alternatieven gegeven. Wij gebruiken `scanf` hier wel *for the sake of simplicity*. Het is beter om `scanf` te vermijden.

1.6 Keywords

In tabel 1.1 is een lijst te zien met gereserveerde woorden. Een aantal van deze woorden hebben we al gezien zoals `int`, `void` en `return`. Deze woorden worden *keywords* genoemd en vormen de vocabulaire van de taal C.

Table 1.1: Een lijst met keywords in de C-taal.

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>float</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>signed</code>	<code>void</code>
<code>default</code>	<code>goto</code>	<code>sizeof</code>	<code>volatile</code>
<code>do</code>	<code>if</code>	<code>static</code>	<code>while</code>

Een aantal van deze keywords dient als *qualifier* voor andere keywords. Zo kunnen we een variabele definiëren als

```
unsigned long int a;
```

Dit bepaalt de grootte (het aantal bits) van variabele `a`. Zie hoofdstuk ??.

1.7 Beslissingen

Met behulp van de keywords `if` en `else` kunnen we in het programma beslissingen nemen op basis van een *conditie*. Dit is te zien in listing 1.5. We definiëren twee variabelen `a` en `b` en kennen gelijk de waarden toe. In regel 8 is te zien dat getest wordt of `a` kleiner is dan `b`. We noemen het kleiner-dan-teken een *relationele operator*. Als inderdaad blijkt dat `a` kleiner is dan `b`, dat betekent dat de conditie `a < b` waar is, worden de statements tussen de accolades in de regels 9 en 11 uitgevoerd. Is de conditie niet waar, dan worden deze regels overgeslagen. Overigens mogen in dit geval de accolades weggelaten worden omdat maar één statement wordt uitgevoerd. Het is echter aan te bevelen om ze toch te gebruiken, omdat misschien later er nog statements toegevoegd worden. Een bekend voorbeeld van het niet-gebruiken van accolades is Apple's *goto-fail SSL security bug* [barr2014].

Een `if`-statement kan ook gevolgd worden door een `else`-keyword en een `else`-keyword kan ook weer gevolgd worden door een `if`-keyword. In het Engels wordt dit een *multiway branch* (to branch = vertakken) genoemd. Zie listing 1.6.

Technisch gezien hoort de `else` in regel 18 bij de `if` in regel 14. Let erop dat het vergelijken van `a` en `b` op gelijkheid in regel 12 een *dubbele is-gelijk-teken* (`==`) bevat.

1.8 Herhalingen

Stel dat we de kwadraten van 1 t/m 10 willen afdrukken. We kunnen ervoor kiezen om tien `print`-functies aan te roepen. Maar we merken direct op dat we in feite tien keer hetzelfde moeten doen. We kunnen het afdrukken van de tien kwadraten vormgeven met een *herhaling*. Een andere, veel gebruikte term is *lus*. Dit is te zien in listing 1.7.

We beginnen het programma met de definitie en initialisatie van een aantal variabelen. Hierin stellen we de ondergrens, bovengrens en stapgrootte in (regel 5). We willen beginnen bij de ondergrens, stoppen bij de bovengrens en bij elke herhaling nieuwe waarden afdrukken. Daarvoor gebruiken we de variabele `getal`. Zo'n variabele wordt een *lusvariabele* genoemd.

In regel 8 zetten we de lusvariabele op de ondergrens en gaan de lus uitvoeren. De lus wordt gekenmerkt door het keyword `while`. Achter het keyword `while` staat, tussen de haken, de conditie waarop de lus moet worden uitgevoerd. Zolang de conditie `getal <= bovengrens` waar is, worden de statements binnen de accolades uitgevoerd. We noemen dat een *iteratie*. Is de conditie niet waar dan wordt verder gegaan met het statement dat volgt op het `while`-statement.

Binnen de accolades van het `while`-statement zien we drie statements. In regel 11 wordt het kwadraat berekend van de (huidige) waarde van de lusvariabele. Daarna worden de lusvariabele en het kwadraat afgedrukt. In regel 13 wordt de lusvariabele aangepast naar de nieuwe (volgende) waarde door de stapgrootte erbij op te tellen⁵.

In geval van het kwadratenprogramma kunnen we ook een ander herhalingsstatement gebruiken: het `for`-statement. Het is een compacte schrijfwijze van het `while`-statement. We kunnen het kwadraat ook uitrekenen in de aanroep van de functie `printf`. Als argument van een functie mogen we namelijk een expressie gebruiken. Zo sparen we een variabele uit. Zie listing 1.8.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 7;
6     int b = 9;
7
8     if (a < b)
9     {
10         printf("a is kleiner dan b\n");
11     }
12
13     return 0;
14 }
```

Listing 1.5: Afdrukken van tekst op basis van een beslissing.

⁵ Het is tegenwoordig in het Nederlands gebruikelijk om het Engelse woord *updaten* te gebruiken.

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a, b;
6
7     printf("Geef twee getallen: ");
8     scanf("%d%d", &a, &b);
9
10    if (a < b)
11    {
12        printf("a is kleiner dan b\n");
13    }
14    else if (a == b)
15    {
16        printf("a is gelijk aan b\n");
17    }
18    else
19    {
20        printf("a is groter dan b\n");
21    }
22    return 0;
23 }

```

Listing 1.6: Afdrukken van tekst op basis van een beslissing.

1.9 Functies

C biedt een handige manier om een groep statement, bijvoorbeeld berekeningen, onder te brengen in een *functie*. Als de functie eenmaal geschreven is, hoeven we ons niet druk te maken over hoe de berekeningen worden gedaan, we moeten alleen maar weten hoe we de functie moeten gebruiken. We hebben al drie functies gezien: `printf`, `scanf` en `main`. De functies `printf` en `scanf` zijn al beschikbaar via de standard library. We hoeven niet te weten hoe de functies werken, alleen maar hoe ze moeten worden aangeroepen.

Laten we eens de wortels bepalen van de getallen 0 t/m 10. Daartoe schrijven we een functie `sqrt_babylonian` die de wortel van een getal bepaalt volgens de Babylonische methode. Dit is een iteratieve methode om de wortel van een getal steeds nauwkeuriger te benaderen. De manier waarop dat gebeurt is te zien in vergelijking (1.1).

$$\begin{array}{ll}
 x_0 = S & \text{beginsituatie} \\
 x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{S}{x_n} \right) & \text{nieuwe situatie} \\
 \sqrt{S} = \lim_{n \rightarrow \infty} x_n & \text{uiteindelijke resultaat}
 \end{array} \tag{1.1}$$

```

1 #include <stdio.h>
2
3 int main(void)
4 {
5     int ondergrens = 1, bovengrens = 10, stap = 1;
6     int getal, kwadraat;
7
8     getal = ondergrens;
9     while (getal <= bovengrens)
10    {
11        kwadraat = getal * getal;
12        printf("Het kwadraat van %3d is %3d\n", getal, kwadraat);
13        getal = getal + stap;
14    }
15
16    return 0;
17 }

```

Listing 1.7: Afdrukken van de kwadraten van 1 t/m 10..

```

1     for (getal = ondergrens; getal <= bovengrens;
2           getal = getal + stap)
3     {
4         printf("Het kwadraat van %3d is %3d\n",
5               getal, getal * getal);
6     }

```

Listing 1.8: Gebruik van een *for*-statement.

We lezen dit als volgt. We beginnen met het getal S waarvan we de wortel willen berekenen ($x_0 = S$). De tweede, iteratieve stap, is het berekenen van de volgende benadering. We berekenen dat met de middelste vergelijking uit (1.1). De laatste stap geeft aan dat, als we de wortel willen bepalen, de tweede stap oneindig vaak herhaald moet worden. Natuurlijk is dat niet realiseerbaar. We kunnen maar een eindig aantal stappen uitrekenen. Het blijkt dat voor de wortel slechts tien stappen nodig zijn om de wortel redelijk te benaderen.

De functie is te zien in listing 1.9. We maken hier gebruik van het datatype `double`, een floating point-datatype met een nauwkeurigheid van ongeveer 16 decimale cijfers. Zowel het argument als het returnwaarde is van dit datatype. We beginnen met de functiedefinitie

```
double square_root(double s)
```

Binnen de functie maken we gebruik van de twee variabelen `xiter` en `i`. `xiter` wordt gebruikt om de nieuwe waarde van de wortel te berekenen en `i` wordt gebruik om het aantal iteraties bij

te houden. We testen in regels 8 t/m 10 of het argument 0 is en geven dan direct de waarde 0 terug.

```
1 #include <stdio.h>
2
3 double square_root(double s)
4 {
5     double xiter = s;
6     int i;
7
8     if (s == 0.0)
9     {
10         return 0.0;
11     }
12
13     for (i = 0; i < 10; i = i + 1)
14     {
15         xiter = 0.5 * (xiter + s / xiter);
16     }
17
18     return xiter;
19 }
20
21 int main(void)
22 {
23     double i;
24
25     for (i = 0.0; i < 11.0; i = i + 1.0)
26     {
27         printf("De wortel van %6.3f is %6f\n", i, square_root(i));
28     }
29
30     return 0;
31 }
```

Listing 1.9: Programma om de wortels van 1 t/m 10 af te drukken.

In het for-statement worden tien iteraties van de middelste formule uit (1.1) uitgevoerd. Bij elke iteratie wordt de waarde van de wortel steeds beter benaderd. Als het for-statement klaar is, wordt deze waarde via return teruggegeven aan de aanroeper. In main worden met behulp van een for-statement de wortels van 0 t/m 10 berekend en afgedrukt.

Overigens bevat de *mathematical library* een implementatie van de wortel-functie. We hoeven dat niet zelf te schrijven. Functies worden behandeld in hoofdstuk ??.

1.10 Arrays

Een *array* is een lijst variabelen onder een gemeenschappelijke noemer. Zo definiëren we vijf floating point-getallen (getallen met een komma) met

```
double lijst[5];
```

en kunnen we gebruik maken van de variabelen

```
lijst[0] lijst[1] lijst[2] lijst[3] lijst[4]
```

Tussen de blokhaken `[]` staat het *elementnummer* en de variabelen worden de *elementen* van de array genoemd. In listing 1.10 is een programma te zien met een array. We hebben als getallen vijf wiskundige constanten genomen. De lezer wordt uitgedaagd uit te zoeken welke constanten dat zijn.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     double lijst[5] = {3.14159, 2.71828, 0.57721, 1.41421, 1.61803};
6     double som = 0.0;
7     int index;
8
9     for (index = 0; index < 5; index = index + 1)
10    {
11        som = som + lijst[index];
12    }
13
14    printf("Gemiddelde: %f\n", som / 5.0);
15
16    return 0;
17 }
```

Listing 1.10: *Gemiddelde van vijf getallen.*

De array mag bij definitie gelijk geïnitialiseerd worden zoals te zien is in regel 5. We berekenen van deze vijf elementen de gemiddelde waarde. We doen dat door eerst de som van de vijf element te berekenen en vervolgens te delen door 5. Met behulp van een for-statement wordt “langs de array gelopen”; bij elke iteratie selecteren we een element en tellen dat op bij de som van de tot dan toe gesommeerde elementen. De regel

```
som = som + lijst[index];
```

selecteert dus een element uit de array en telt dat op bij de som. In regel 13 drukken we het gemiddelde af door de som te delen door 5. We hoeven daarvoor niet een aparte variabele te definiëren, we kunnen als argument gewoon `som / 5.0` gebruiken.

1.11 Strings

Een veel gebruikte arraytype is de *string*. Dit is een array van karakters, afgesloten met een speciaal karakter dat het *null-karakter* of *null-byte* genoemd worden. C heeft geen ingebouwde taalconstructies die direct met strings werken, maar er zijn veel *functies* beschikbaar om strings te verwerken. Een voorbeeld van een string is:

```
1 char str[] = "Ik_ben_een_string";
```

Listing 1.11: Voorbeeld van een string.

We hoeven de lengte van de string in dit geval niet op te geven, dat wordt automatisch door de C-compiler uitgerekend. Dat C functies heeft voor het afhandelen van string is te zien in listing 1.12. In dit programma gebruiken we de functie `strlen` (string length) om de lengte van de string te bepalen. Wel moeten we het header-bestand `string.h` laden waarin de stringfuncties gedeclareerd zijn.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void)
5 {
6     char string[] = "Ik_ben_een_string";
7
8     int len = strlen(string);
9
10    printf("De_lengte_is_%d\n", len);
11
12    return 0;
13 }
```

Listing 1.12: Voorbeeld van een stringfunctie.

Strings worden behandeld in hoofdstuk ??.

1.12 Programmeerstijlen

C is erg coulant in het vormgeven van een programma. We kunnen regels overslaan (een lege regel) en we kunnen statements vooraf laten gaan door een willekeurig aantal spaties of we kunnen statements achter elkaar op een regel plaatsen. Ook het gebruik van variabelenamen en functienamen staat geheel los van de taal, behalve de functie `main`. We kunnen één letter gebruiken of een compleet woord.

Een mooi voorbeeld hoe we onze wortelfunctie onleesbaar kunnen maken, kunnen we zien in listing 1.13. De functie wordt door de C-compiler correct vertaald maar is voor een mens slecht leesbaar. De code in listing 1.9 is veel beter leesbaar.

```

1 double square_root(double s) {
2     double xiter = s; int i;
3     if (s==0.0) return 0.0;
4     for (i=0;i<10;i=i+1) xiter=0.5*(xiter+s/xiter); return xiter;}

```

Listing 1.13: Een functie om een wortel van een getal te bepalen.

Hoe een programma uiterlijk wordt vormgegeven noemen we een *programmeerstijl*. Er zijn behoorlijk wat stijlen. We geven hieronder de meest gebruikte aan.

Layout van het programma

In de K&R-stijl (Kernighan en Ritchie) wordt de accolade-openen *achter* een functie, beslissings- of herhalingsstatement geschreven. De statements hierbinnen worden *ingesprongen* (dat wordt in het Engels *indentation* genoemd). Bij gebruik van in elkaar verweven beslissings- of herhalingsstatement worden de statements verder of dieper ingesprongen. De reden van de K&R-stijl is dat ouderwetse editors per regel werkte (en niet full-screen zoals nu). Daardoor konden programmeurs zien of er een blok begon.

```

1 while (a < b) {
2     statements
3     if (a < 0) {
4         statements
5     }
6 }

```

Listing 1.14: K&R-stijl.

In de Allman-stijl wordt de accolade-openen *onder* een functie, beslissings- of herhalingsstatement geschreven. De statement hierbinnen worden *ingesprongen*. Bij gebruik van in elkaar verweven beslissings- of herhalingsstatement worden de statement verder of dieper ingesprongen.

```

1 if (a < b)
2 {
3     if (a < 0)
4     {
5         statements
6     }
7 }

```

Listing 1.15: Allman-stijl.

In de eerste hoofdstukken gebruiken we de Allman-stijl. In latere hoofdstukken gebruiken we de K&R-stijl.

Naamgeving van variabelen

Ook voor naamgeving van variabelen zijn diverse gebruiken in omloop. We zullen er drie behandelen.

In de K&R-stijl zijn er eigenlijk geen regels. Variabelennamen mogen uit één letter bestaan, of natuurlijk uit meerdere letters. Als namen syntactisch uit meerdere woorden bestaan, worden de woorden gescheiden door een *underscore*. Het is wel aan te raden dat namen zinvol zijn en dat ze de betekenis vertegenwoordigen. Overigens is het gebruik van eenletterige variabelennamen prima te verantwoorden, bijvoorbeeld binnen het gebruik van een kortdurende herhaling én als het programma niet al te groot is⁶.

In de camelCase-stijl worden variabelen weergegeven met kleine letters en hoofdletters geven scheidingen aan tussen woorden die syntactisch gescheiden zijn. Er worden geen underscores gebruikt. Deze stijl is afkomstig uit Java.

```
1  int dayOfTheWeek;  
2  int dayWithinMonth;  
3  int dayOfTheYear;
```

Listing 1.16: camelCase-stijl.

Bij de Hungarian Notation worden variabelennamen vooraf gegaan door een omschrijving van het datatype en het semantiek van de variabele. Zo kunnen we integers vooraf laten gaan door een *i*. Maar dat zien de meeste programmeurs ook wel. We kunnen van een integer ook het doel aangeven. Zo wordt bijvoorbeeld een variabele die dient als teller in een herhaling vooraf gegaan door *c* (van “count”):

```
1  int iDayOfTheWeek;  
2  int cLoops;
```

Listing 1.17: Voorbeeld Hungarian Notation.

Deze stijl wordt vooral gebruikt bij grote teams programmeurs die gezamenlijk aan één programma werken. Het nadeel van deze stijl is dat de typering niet gestandaardiseerd is. Overigens kan met ontwikkelomgevingen het type gevonden worden door op de variabele te klikken.

1.13 Commentaar

“To comment or not to comment — that’s the question.”

Commentaar kan op twee manieren worden toegevoegd. Ten eerste is er het *regelcommentaar* (vanaf C99). Zodra de compiler een dubbele *slash* `//` tegenkomt (maar *niet* in een string), wordt

⁶ In dit boek gebruiken we vooral de K&R-stijl. We gebruiken ook vaak eenletterige variabelen. Dat is te verklaren omdat veel voorbeelden kort zijn en we de lezer niet willen vermoeien met lange variabelennamen. Het gebruik van de accolade-openen achter een taalconstructie levert een besparing op van één programmaregel.

de rest van de regel als commentaar gezien en door de compiler overgeslagen. Zie listing 1.18. Regelcommentaar is vooral handig om even snel aan te geven wat de regel doet.

```
1 // Dit is een commentaarregel
2 int a = 2; // Vanaf nu is het een commentaarregel
```

Listing 1.18: Voorbeeld van commentaarregels.

Ten tweede is er het *blokcommentaar*. Het blokcommentaar begint met `/*` en eindigt met `*/`. Let erop dat het blokcommentaar niet *genest* mag zijn, dus een `/*` – `*/`-paar mag niet binnen een blokcommentaar voorkomen.

```
1 /* Dit is een blokcommentaar
2    en mag over
3    meerdere regels
4    gebruikt worden */
5 int a = 2;
```

Listing 1.19: Voorbeeld van commentaarregels.

Het is verstandig om elk bestand (in feite elk leesbaar bestand, dit is niet beperkt tot een C-programma) commentaar op te nemen over de auteur, de datum, de versie en het doel van het programma (of deelprogramma, zie hoofdstuk ??). Daarnaast is het verstandig om kort toe te lichten wat het programma doet.

```
1 /* File:    intro.c
2  * Author:   A.J. Kwak
3  * Date:     2021/08/16
4  * Version: 0.1
5  * Licence:  GPL v2
6  *
7  * This is just some comment to show how a useful
8  * comment can provide the reader some basic
9  * information */
```

Listing 1.20: Commentaar aan het begin van het bestand.

1.14 En verder...

We hebben nu enkele concepten van C uitgelegd. Maar de taal kan meer. Wat we niet behandeld hebben zijn *pointers* en *structures*, onderdelen van de taal. We hebben ook *bestandsverwerking* niet behandeld. Dit is technisch gezien geen onderdeel van de taal, maar wordt vaak gebruikt in programma's. Al deze concepten zullen in de volgende hoofdstukken worden uitgelegd.