Een overzicht van C

Dit hoofdstuk biedt een overzicht van de programmeertaal C. Er worden enkele programma's gepresenteerd en de onderdelen van elk programma worden zorgvuldig toegelicht. Door het gehele boek heen wordt de nadruk gelegd op het experimenteren en het interactief werken. In dit hoofdstuk ligt de nadruk op het leren gebruiken van de belangrijkste invoer- en uitvoerfuncties van C (input/output).

Iedereen zou al het materiaal in dit hoofdstuk moeten doorlezen, met uitzondering van de paragrafen 1.8, 'Arrays, strings en pointers', en 1.9, 'Files'. Lezers die in een andere taal ervaring hebben opgedaan met arrays, pointers en files, kunnen het gehele hoofdstuk doorlezen om een volledig beeld van C te verkrijgen. De anderen kunnen er later desgewenst op terugkomen. Men leze dit hoofdstuk in het besef, dat technische details en verdere uitleg zullen volgen in de komende hoofdstukken.

1.1 Programmeren en voorbereiding

In de computer bevindt zich een collectie speciale programma's: het besturingssysteem. Tot de gangbare besturingssystemen behoren bijvoorbeeld UNIX, Linux, MAC OS X en MS Windows. Een besturingssysteem zorgt voor de organisatie van de processen in de computer, stelt software beschikbaar aan de gebruiker en treedt op als intermediair tussen de gebruiker en de hardware. Tot de vele software-pakketten die het systeem ter beschikking kan stellen, behoren de C-compiler en diverse tekstverwerkers. De voornaamste tekstverwerker op het UNIX systeem draagt de naam *vi*. In sommige systemen, zoals Turbo-C, zijn compiler en tekstverwerker tot een geheel samengevoegd. Wij gaan ervan uit dat de lezer gebruik kan maken van de een of andere tekstverwerker om bestanden te kunnen vervaardigen die C-code bevatten. Een dergelijk bestand draagt de naam bronbestand, en op de meeste UNIX-systemen geschiedt de compilatie met het commando cc, waarmee de C-compiler wordt opgeroepen. Ruw gezegd vertaalt een compiler de broncode naar objectcode, die uitvoerbaar is. In het UNIX-systeem wordt de gecompileerde code automatisch geplaatst in een bestand met de naam *a.out*. In een Windowssysteem krijgt het bestand met de gecompileerde code dezelfde naam als het *.c* bestand, met dien verstande dat de *.c* uitgang wordt vervangen door *.exe*. Aan het eind van dit hoofdstuk, in paragraaf 1.10, 'Het besturingssysteem', wordt uiteengezet welke stappen er nodig zijn om een programma te schrijven, te compileren en tot uitvoering te brengen.

1.2 Programma-uitvoer

Om nuttig te zijn moeten programma's kunnen communiceren. Ons eerste voorbeeld is een programma dat de frase 'hoezee voor helder C' op het scherm afdrukt. Het volledige programma is

```
#include      <stdio.h>

main()
{
       printf("hoezee voor helder C\n");
}
```

Met behulp van een tekstverwerker wordt dit ingetypt in een bestand waarvan de naam eindigt op *.c* . Men doet er goed aan de naam zinvol te kiezen en zo dat het menselijk geheugen er een aanknopingspunt in vindt. Laten wij aannemen dat het programma is ondergebracht in het bestand *zee.c*. Om het programma te compileren geven wij de opdracht

cc zee.c

Als er geen fouten zijn, zorgt dit commando voor het aanmaken van het uitvoerbare bestand *a.out*. Na het commando

```
a.out
```

wordt het programma uitgevoerd; op het scherm wordt afgedrukt:

```
hoezee voor helder C
```

Analyse van het programma *zee*

```
#include       <stdio.h>
```

De C-compiler bezit een preprocessor (een voorbereidingsprogramma). Als het commando tot compilatie is gegeven, wordt de code eerst met de preprocessor behandeld en daarna gecompileerd. Regels die met een # beginnen communiceren met de preprocessor. De regel `#include` zorgt ervoor dat er op deze plaats een kopie van het header-bestand *stdio.h* in de code wordt ingevoegd. Het header-bestand wordt door het C-systeem verstrekt. De spitse haakjes rondom *stdio.h* duiden aan dat het bestand op de gebruikelijke plaats kan worden gevonden (systeemafhankelijk). Het bestand werd ingevoegd omdat het informatie bevat over de functie `printf()`.

```
main()
```

Ieder programma heeft een functie met de naam main, waar de uitvoering begint. De haakjes achter main delen aan de compiler mee dat hier een functie staat.

```
{
```

De definitie-tekst van een functie wordt ingesloten tussen accolades. Accolades worden ook gebruikt om statements te groeperen.

```
printf()
```

Het C-systeem bevat een standaardbibliotheek met functies die gebruikt kunnen worden in programma's. `printf()` is een bibliotheekfunctie die informatie op het scherm afdrukt. Wij hebben het header-bestand *stdio.h* ingevoegd, dat aan de compiler zekere informatie verschaft omtrent de functie `printf()` (zie oefening 11).

```
"hoezee voor helder C\n"
```

In C wordt een string-constante gevormd door een reeks tekens tussen dubbele aanhalingstekens te plaatsen. Deze string is het argument voor de functie `printf()`. De twee tekens `\n` aan het einde van de string (lees: 'backslash n') staan voor een enkel teken, het *nieuwe-regel-teken (newline).* Dit wordt niet afgedrukt, maar zorgt ervoor dat de cursor aan het begin van de volgende regel komt te staan.

```
|       printf("hoezee voor helder C\n");
```

Dit is een statement. In C eindigt ieder statement met een puntkomma. De functie `printf()` wordt opgeroepen; deze bewerkstelligt dat de string-constante die als argument werd meegegeven, op het scherm verschijnt.

```
|       }
```

Deze rechter accolade is gepaard aan de hierboven staande linker accolade en brengt de functie main() tot een einde.

De functie `printf()` schrijft continu op het scherm. De cursor gaat naar de volgende regel als het newline-karakter is ingelezen. Het scherm is een tweedimensionaal medium waarop van links naar rechts en van boven naar onder wordt geschreven. Om goed leesbaar te zijn dient de output op correcte wijze gespatieerd op het scherm te verschijnen. Wij kunnen ons eerste programma als volgt herschrijven:

```
#include       <stdio.h>

main()
 {
        printf("hoezee ");
        printf("voor helder C");
        printf("\n");
 }
```

Dit programma verschilt van de eerste versie, maar het zal dezelfde output opleveren. Telkens als `printf()` wordt aangeroepen begint het schrijven (of afdrukken) op de positie waar `printf()` bij de vorige aanroep eindigde. Als we de frase op drie regels willen afdrukken, dan kunnen wij newline-karakters gebruiken.

```
#include       <stdio.h>

main()
 {
        printf("hoezee\n");
        printf("voor helder\nC\n");
 }
```

Met dit programma krijgen we als output te zien:

```
hoezee
voor helder
C
```

Wij schrijven nog een versie van dit programma; de frase zal nu in een door asterisken gevorm-

de rechthoek worden geplaatst. Het programma verduidelijkt de rol van elk karakter, inclusief het spatie-karakter en het newline-karakter.

```
#include       <stdio.h>

main()
 {
        printf("\n\n\n\n\n\n\n\n\n\n\n");
        printf("      ************************\n");
        printf("      *      hoezee          *\n");
        printf("      *      voor helder C   *\n");
        printf("      ************************\n");
        printf("\n\n\n\n\n\n\n\n\n\n\n");
 }
```

1.3 Variabelen, expressies en toewijzingen

Wij zullen een programma schrijven dat de afstand van een marathonloop omrekent van mijlen naar kilometers. Die afstand wordt in Engelse mijlen vastgesteld op 26 mijlen en 385 yards. Dit zijn gehele getallen — integers. Voor het omrekenen van mijlen naar kilometers moet worden vermenigvuldigd met de conversiefactor 1.609, een reëel getal. In het computer-geheugen worden reële getallen en gehele getallen (integers) op verschillende manieren gerepresenteerd. Om yards om te rekenen naar mijlen wordt gedeeld door 1760.0, en zoals zal blijken is het van wezenlijk belang dit getal te representeren als een reëel getal (een 'real'), en niet als integer.

Ons omrekeningsprogramma zal gebruik maken van variabelen die integer-waarden of reële waarden kunnen herbergen. In C moeten alle variabelen worden gedeclareerd, of benoemd, en wel aan het begin van het programma. De naam van een variabele, ook identifier genoemd, bestaat uit een reeks letters, cijfers en onderstrepingstekens, doch mag niet beginnen met een cijfer. Identifiers kiest men op een manier die hun betekenis in het programma weerspiegelt. Aldus kunnen zij ook dienen als documentatie en ze maken zo een programma beter leesbaar en begrijpelijk.

```
/* De afstand van een marathonloop in kilometers. */

#include       <stdio.h>

main()
{
   int    mijlen, yards;
   float  kilometers;

   mijlen = 26;
   yards  = 385;
   kilometers = 1.609 * (mijlen + yards / 1760.0);
   printf("\nEen marathonloop telt %f kilometers.\n\n", kilometers);
}
```

Het programma levert als output:

```
Een marathonloop telt 42.185970 kilometers.
```

Analyse van het programma *marathon*

```
/* De afstand van een marathonloop in kilometers. */
```

Alles wat voorkomt tussen de tekens `/*` en `*/` is commentaar en wordt door de compiler genegeerd. Alle programma's van dit boek die beginnen met commentaar zijn in de index te vinden.

```
int      mijlen, yards;
```

Dit is een declaratie. Declaraties en statements worden beëindigd met een puntkomma. int is een sleutelwoord en refereert aan een van de fundamentele typen van de taal. Aan de compiler wordt medegedeeld dat de navolgende variabelen van het type int zijn en integer waarden moeten aannemen. De variabelen `mijlen` en `yards` uit dit programma zijn dus van het type `int`.

```
float     kilometers;
```

Dit is een declaratie, float is een sleutelwoord en refereert aan een van de fundamentele typen van de taal. Aan de compiler wordt medegedeeld dat de navolgende variabelen van het type `float` zijn en reële waarden moeten aannemen. De variabele `kilometers` uit dit programma is dus van het type `float`.

```
mijlen = 26;
yards  = 385;
```

Dit zijn toewijzingsopdrachten. Het gelijk-teken (=) is de toewijzingsoperator. De twee getallen 26 en 385 zijn integer-constanten. Aan de variabele `mijlen` wordt de waarde `26` toegewezen, aan de variabele `yards` de waarde `385`.

```
kilometers = 1.609 * (mijlen + yards / 1760.0);
```

Dit is een toewijzingsopdracht. De waarde van de uitdrukking (of 'expressie') aan de rechterkant van het gelijkteken wordt toegewezen aan de variabele kilometers in het linkerlid. De operatoren *, + en / staan respectievelijk voor vermenigvuldigen, optellen en delen. Operaties binnen haakjes worden het eerst uitgevoerd. Omdat de deling een hogere prioriteit heeft dan de optelling (zie hoofdstuk 3), wordt eerst de waarde berekend van de volgende subexpressie.

```
yards / 1760.0
```

Die waarde wordt opgeteld bij de waarde van de variabele mijlen en daaruit ontstaat een waarde die wordt vermenigvuldigd met `1.609`. De zo verkregen uitkomst wordt toegewezen aan de variabele `kilometers`.

```
printf("\nEen marathonloop telt %f kilometers.\n\n", kilometers);
```

Dit is een statement waarin de functie printf() wordt aangeroepen. De functie `printf()` kan een variabel aantal argumenten hebben. Het eerste argument is altijd een string, de *formaat-string* genoemd. In het onderhavige voorbeeld is de formaat-string:

```
"\nEen marathonloop telt %f kilometers.\n\n"
```

Die vormt het eerste argument voor de functie `printf()`. Binnen in deze string bevindt zich de conversiespecificatie `%f`. De conversiespecificaties (ook wel *formaten* genoemd) in de formaat-string worden gekoppeld aan de overige argumenten in de functie `printf()`. In dit geval wordt `%f` gekoppeld aan het argument `kilometers`. Dat betekent dat de waarde van de variabele `kilometers` moet worden afgedrukt als een floating-point getal en in de afdrukstroom daar moet worden ingevoegd waar de specificatie `%f` voorkomt.

Bepaalde woorden, *keywords* genoemd, zijn gereserveerd en kunnen niet door de programmeur worden gebruikt als naam van een variabele. Zo zijn bijvoorbeeld `int`, `float` en `double` keywords. In hoofdstuk 2 wordt een tabel van de keywords opgevoerd. Er zijn andere namen die in het C-systeem bekend zijn en gewoonlijk niet door de programmeur zullen worden geherdefinieerd. De naam `printf` is zo'n voorbeeld. Omdat `printf` de naam is van een functie uit de standaardbibliotheek, wordt dit woord gewoonlijk niet gebruikt als naam voor een variabele.

De decimale punt in een getal duidt aan dat het gaat om een floating-constante en niet om een integer-constante. De getallen `37` en `37.0` zullen in een programma verschillend worden behandeld. Er zijn drie soorten floating-point typen, `float`, `double` en `long double`, en er kunnen voor elk van deze typen ook variabelen worden gedeclareerd; doch floating-constanten zijn automatisch van het type `double`.

Expressies treft men veelvuldig aan als argument voor functies en aan de rechterzijde van toewijzingen. De meest eenvoudige expressies zijn simpele constanten, zoals `385` en `1760.0`, die in het voorafgaande programma werden gebruikt. De naam van een variabele kan worden beschouwd als een expressie, en zinvolle combinaties van operatoren, variabelen en constanten leveren eveneens expressies op. Bij de evaluatie van expressies kunnen er conversieregels worden toegepast. Dit is een belangrijke kwestie. Bij deling van twee integers komt er een integer-waarde als uitkomst en een eventuele rest wordt weggelaten. Zo zal de expressie `7/2` het `int`-getal `3` als waarde opleveren. Doch in de expressie `7.0/2` wordt een `double` gedeeld door een `int`.
Tijdens de evaluatie van deze expressie wordt de waarde van de expressie `2` automatisch geconverteerd naar een `double`, ten gevolge waarvan de expressie `7.0/2` de waarde `3.5` heeft. Veronderstel eens dat in het voorafgaande programma het statement

```
kilometers = 1.609 * (mijlen + yards / 1760.0);
```

wordt gewijzigd tot

```
kilometers = 1.609 * (mijlen + yards / 1760);
```

Dat zal leiden tot een foutief programma. Omdat de variabele `yards` van het type `int` is, en de waarde `385` heeft, krijgt de expressie

```
yards / 1760
```

de `int`-waarde `0`. En dat is niet hetgeen gewenst wordt. Door de constante `1760.0` van het type `double` te gebruiken, wordt de fout hersteld.

1.4 Het gebruik van `#define` en `#include`

De C-compiler heeft een ingebouwde preprocessor. Regels die beginnen met het symbool # dragen de naam *directieven voor de preprocessor*. Als de regels

```
#define       LIMIET      100
#define       PI          3.14159
```

voorkomen in een bestand dat wordt gecompileerd, dan substitueert de preprocessor overal voor `LIMIET` de waarde `100` en voor `PI` de waarde `3.14159`, behalve dan in strings of in commentaar. De identifiers `LIMIET` en `PI` worden *symbolische constanten* genoemd. Een `#define`-regel kan overal in een programma staan. Hij heeft slechts invloed op die regels in het bestand, die er later op volgen.

Gewoonlijk worden alle `#define`-regels aan het begin van het bestand geplaatst. Het is gebruikelijk alle identifiers die door de preprocessor moeten worden gewijzigd, in hoofdletters te schrijven. De preprocessor verandert nimmer de inhoud van tussen aanhalingstekens geplaatste strings. Zo zal bijvoorbeeld in het statement

```
printf("PI = %f \n", PI);
```

alleen de tweede `PI` ten gevolge van de hierboven gegeven directieven aan de preprocessor worden gewijzigd. Het gebruik van symbolische constanten maakt een programma beter leesbaar. Belangrijker is echter, dat een constante die symbolisch gedefinieerd werd met behulp van `#define` en in het gehele programma werd gebruikt, later zonodig gemakkelijk kan worden gewijzigd. In de fysica wordt bijvoorbeeld de letter *c* gebruikt om de lichtsnelheid aan te geven. De waarde van *c* werd voorheen experimenteel bepaald, maar tegenwoordig wordt de waarde per definitie vastgesteld op `299792.458 km/sec`. Als wij schrijven

```
#define      c       299792.458    /* lichtsnelheid in km/sec */
```

en vervolgens de letter *c* in duizenden regels code gebruiken om symbolisch de constante `299792.458` te representeren, dan zal het gemakkelijk zijn op een later tijdstip de code te wijzigen, als bijvoorbeeld de fysici mochten besluiten de waarde te herdefiniëren. De totale code wordt eenvoudig aangepast door simpelweg de constante in de `#define`-regel te veranderen.
Een in een programma voorkomende regel zoals

```
#include    "my-file.h"
```

is ook een directief aan de preprocessor; het bewerkstelligt dat tijdens het compileren op die positie in het bronbestand een kopie van het bestand `my_file.h` wordt ingevoegd. Een `#include` mag overal in een bestand staan, ofschoon hij meestal aan het begin van het bestand wordt aangetroffen. De aanhalingstekens die de naam omsluiten zijn onontbeerlijk. Een include-bestand, ook wel genoemd 'header-bestand', kan `#define`-regels bevatten en ook andere `#include`-regels. Bij conventie eindigen de namen van header-bestanden op *.h*.

Het C-systeem voorziet in een aantal standaard header-bestanden. Als voorbeelden kunnen worden genoemd *stdio.h, string.h* en *math.h* .
Deze bestanden bevatten de declaraties van functies in de standaardbibliotheek, macro's, structuur-sjablonen en andere veel gebruikte elementen van het programmeren. Zoals wij reeds zagen veroorzaakt het preprocessor-directief

```
#include     <stdio.h>
```

dat er tijdens het compileren een kopie van het standaard header-bestand *stdio.h* in de code wordt tussengevoegd. In ANSI C moet telkens als de functies `printf()` of `scanf()` worden gebruikt, het standaard header- bestand *stdio.h* worden ingevoegd met behulp van `#include`. Dit bestand bevat de declaraties, preciezer gezegd de functie-prototypen, van deze functies. Zie paragraaf 1.7, 'Functies', voor een nadere toelichting.

De campus Santa Cruz van de Universiteit van California kijkt uit op de Monterey baai van de Stille Oceaan en op een deel van die oceaan ten noordwesten van die baai. Dit vanuit de campus zichtbare deel van de oceaan wordt door de campusbewoners 'Pacific Sea' genoemd. Ter illustratie van de `#include`-faciliteit zullen wij een programma schrijven dat in verscheidene maateenheden de grootte van het oppervlak van de Pacific Sea afdrukt. Wij vervaardigen eerst een header-bestand met de volgende regels

Bestand pacific_sea.h :

```
#include   <stdio.h>

#define OPPERVL                     2337
#define VIERK_MIJL_PER_VIERK_KM     0.3861021585424458
#define VIERK_VOET_PER_VIERK_MIJL   (5280 * 5280)
#define VIERK_INCH_PER_VIERK_VOET   144
#define ACRES_PER_VIERK_MIJL        640
```

Vervolgens schrijven wij de functie `main()` in een *.c* bestand.

Bestand pacific_sea.c :

```
/* Opmeten van de Pacific Sea */

#include    "pacific_sea.h"

main()
{
       const int pacific_sea = OPPERVL; /* in vierkante kilometers */
       double  acres, vierk_mijl, vierk_voet, vierk_inch;

       printf("\n De Pacific Sea heeft een oppervlakte van\n");
       printf("%d vierkante kilometer.\n", pacific_sea);
       vierk_mijl = VIERK_MIJL_PER_VIERK_KM * pacific_sea;
       vierk_voet = VIERK_VOET_PER_VIERK_MIJL * vierk_mijl;
       vierk_inch = VIERK_INCH_PER_VIERK_VOET * vierk_voet;
       acres = ACRES_PER_VIERK_MIJL * vierk_mijl;
       printf("ln andere eenheden wordt dit:\n\n");
       printf("%22.7e acres\n", acres);
       printf("%22.7e vierkante mijl\n", vierk_mijl);
       printf("%22.7e vierkante voet\n", vierk_voet);
       printf("%22.7e vierkante inch\n", vierk_inch);
}
```

Ons programma staat nu in twee bestanden, een *.h* bestand en een *.c* bestand. De output van dit programma is

```
De Pacific Sea heeft een oppervlakte van
2337 vierkante kilometer.
In andere eenheden wordt dit:

      5.7748528e+05 acres
      9.0232074e+02 vierkante mijl
      2.5155259e+10 vierkante voet
      3.6223572e+12 vierkante inch
```

De nieuwe aspecten van het programmeren worden nu besproken.

Analyse van het programma *pacific_sea*

```
#include    "pacific_sea.h"
```

Deze `#include`-regel is een preprocessor-directief. Hierdoor wordt tijdens het compileren een kopie van het header-bestand *pacific_sea.h* tussengevoegd. Omdat dit bestand de regel

```
#include      <stdio.h>
```

bevat, zal de preprocessor deze regel uitwerken en eveneens een kopie toevoegen van het standaard header-bestand *stdio.h.* Wij hebben *stdio.h* opgenomen, omdat gebruik wordt gemaakt van de functie `printf()`. In *pacific_sea.h* worden vijf symbolische constanten gedefinieerd.

```
#define     OPPERVL       2337
```

Deze `#define` regel is een preprocessor-directief. Hierdoor wordt overal in de rest van het bestand de identifier `OPPERVL` vervangen door `2337`. Volgens conventie worden hoofdletters gebruikt voor identifiers die door de preprocessor zullen worden gewijzigd. Als er ergens in de toekomst een nieuwe kaart wordt gemaakt en de oppervlakte van de Pacific Sea opnieuw wordt berekend, behoeft slechts deze regel veranderd te worden om het programma aan te passen.

```
#define     VIERK_MIJL_PER_VIERK_KM     0.3861021585424458
```

De floating-constante 0.3861021585424458 is een conversiefactor. Door een symbolische naam te gebruiken voor deze constante wordt het programma beter leesbaar.

```
#define     VIERK_VOET_PER_VIERK_MIJL  (5280 * 5280)
```

De preprocessor vervangt iedere instantie van de eerste reeks tekens door de tweede reeks tekens. Als een lezer van dit programma weet dat een mijl `5280` voet telt, dan zal hij snel inzien dat deze regel correct is. Wij hadden in plaats van `(5280 * 5280)` ook kunnen schrijven `27878400`. Merk op dat de haakjes, hoewel overbodig, geen kwaad doen. Om technische redenen is het dikwijls nodig symbolische