

C

Een inleiding in de programmeertaal C

Jesse op den Brouw

Eerste druk

©2020 Jesse op den Brouw, Den Haag

Versie: 0.1

Datum: 31 maart 2020

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.



C van Jesse op den Brouw is in licentie gegeven volgens een [Creative Commons Naamsvermelding-NietCommercieel-GelijkDelen 3.0 Nederland-licentie](#).

Suggesties en/of opmerkingen over dit boek kunnen worden gestuurd naar:
J.E.J.opdenBrouw@hhs.nl.

Inhoudsopgave

1	Un tour de C	1
1.1	Ontwikkelssystemen	2
1.2	Eerste programma	3
2	Datatypes, variabelen en constanten	5
2.1	Datatypes	5
3	Flow control	7
4	Funcities	9
5	Pointers	11
5.1	Pointers naar enkelvoudige datatypes	12
5.2	De NULL-pointer	13
5.3	Pointer naar void	15
5.4	Afdrukken van pointers	15
5.5	Pointers naar array's	16
5.6	Strings	17
5.7	Rekenen met pointers	17
5.8	Pointers als functie-argumenten	19
5.9	Pointer als return-waarde	20
5.10	Complexe pointers: pointer naar pointer	21
5.11	Complexe pointers: pointer naar array van pointers	21
5.12	Argumenten meegeven aan een C-programma	21
5.13	Pointers naar funcities	21

1

Un tour de C

C is al een oude taal. De taal is rond 1970 ontworpen door Dennis Ritchie¹ en is dus al zo'n 50 jaar oud. Hij was bezig met het schrijven van een Operating System en zocht naar een mogelijkheid om dit te schrijven op een hoger niveau dat gebruikelijk voor die tijd. In die tijd werden operating systems geschreven in assembly, een taal die dicht 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-generatie-taal (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 operating system. Daarom is de taal geliefd bij programmeurs van kleine computersystemen waarop geen operating system 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 voor het schrijven van compacte programma's, dat te zien is in de vele, soms onduidelijke, taalconstructies. Het is eigenlijk niet geschikt voor beginnende programmeurs. Het is zeker mogelijk om programma's te schrijven die niet correct werken, maar met enige discipline kan ook de minder ervaren programmeur de taal prima gebruiken.

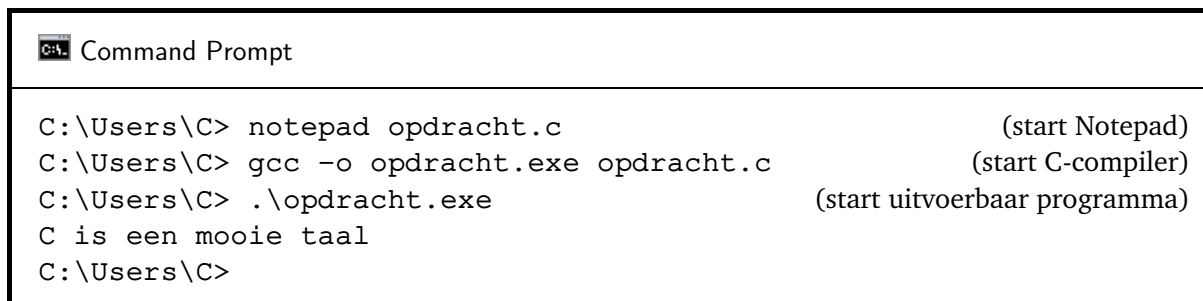
C is een algemeen bruikbare taal (*general purpose language*) en is dus niet voor een specifiek doeleind ontworpen (behalve voor het schrijven van operating systems). Zo kunnen met C wiskundige berekeningen worden uitgevoerd maar de programmeur moet zelf alles schrijven. Er zijn wel *functies* beschikbaar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

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.

C is een *imperatieve* programmeertaal, een van de bekende programmeerparadigma's² C moet concurreren met vele andere talen zoals C++, C#, Python en Java. C++ is ontwikkeld door Bjarne Stroustrup als “de betere C” en ondersteunt *objectgeoriënteerd* programmeren. C# is ontwikkeld door MicroSoft en is de *de facto* programmeertaal op Windows-systemen. Python is ontwikkeld door het Centrum voor Wiskunde en Informatica van de Universiteit van Amsterdam.

1.1 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. Deze werkwijze kan ook op Windows worden gebruikt. Een voorbeeld is te zien in onderstaande figuur.



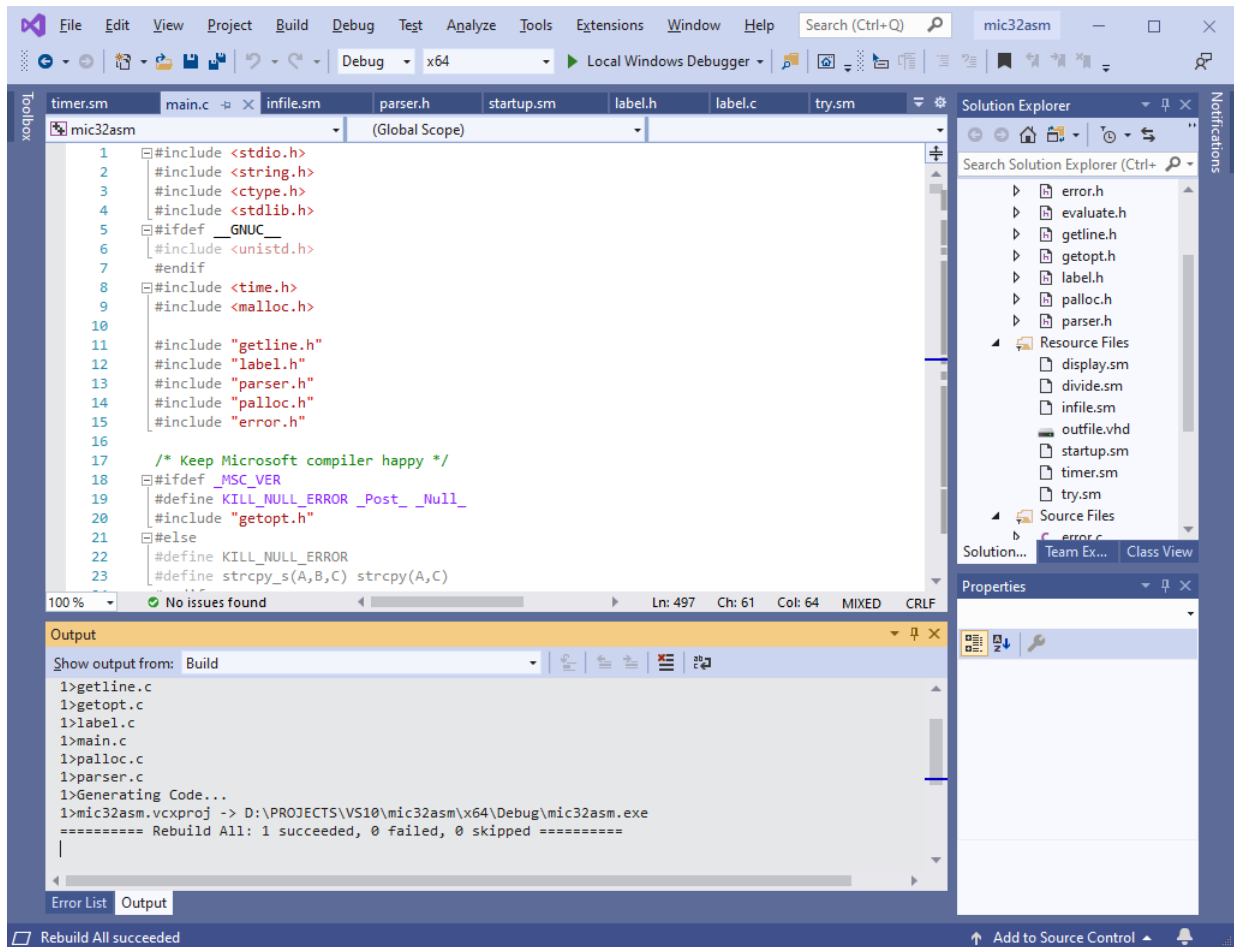
```
Command Prompt

C:\Users\C> notepad opdracht.c                (start Notepad)
C:\Users\C> gcc -o opdracht.exe opdracht.c      (start C-compiler)
C:\Users\C> .\opdracht.exe                     (start uitvoerbaar programma)
C is een mooie taal
C:\Users\C>
```

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 en Code::Blocks.

Een voorbeeld van Microsoft Visual Studio is te zien in figuur 1.1. 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.

² Een programmeerparadigma is een manier van programmeren en een wijze waarop een programma wordt vormgegeven.



Figuur 1.1: Voorbeeld van Microsoft Visual Stdio.

1.2 Eerste programma

Laten we beginnen met een eenvoudig voorbeeld. Het programma in listing 1.1 drukt de regel C is een mooie taal op het scherm af. We zullen het programma stap voor stap doorlopen.

```
#include <stdio.h>

int main(void) {

    printf("C is een mooie taal\n");
    return 0;
}
```

Listing 1.1: Eerste C-programma.

2

Datatypes, variabelen en constanten

2.1 Datatypes

C kent een aantal datatypes.

Tabel 2.1: *De datatypes die beschikbaar zijn in C.*

type	bits	bereik
char	8	−128 — +127
short int	16	−32768 — +32768
int	16	−32768 — +32768
	32	−2147483648 — +2147483647
long long int	64	−9223372036854775807 — +9223372036854775808

3

Flow control

4

Funcities

5

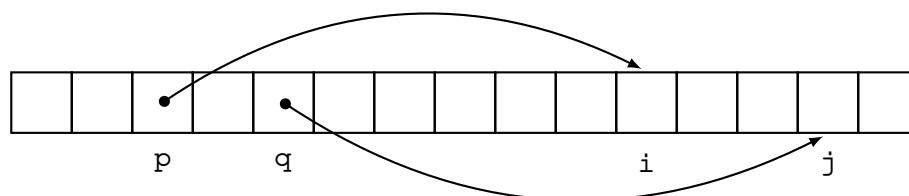
Pointers

Een pointer-variabele, of kortweg *pointer*, is een variabele waarvan de inhoud het adres is van een andere variabele. We zeggen dan ook wel dat de pointer *wijst* (Engels: “points to”) naar de andere variabele. Als een pointer naar een variabele wijst, is het mogelijk om via de pointer bij de variabele te komen.

Pointers zijn een krachtig middel om efficiënt gegevens te beheren en parameters over te dragen aan functies. Soms zijn pointers zelfs de enige manier voor het bewerken van data. Het is dan ook niet verwonderlijk dat in veel C-programma’s pointers gebruikt worden.

Pointers worden samen met het `goto`-statement in verband gebracht met het schrijven van ondoorzichtige programma’s. En zeker, onzorgvuldig gebruik van pointers komt de leesbaarheid en aantonen van correctheid van programma’s niet ten goede. Maar met discipline kunnen programma’s zeer efficiënt geschreven worden.

Een handige manier om pointers weer te geven, is door het geheugen van een computer voor te stellen als een rij vakjes. In figuur 5.1 is dat te zien. Elk vakje stelt een geheugenplaats voor¹. In de figuur zijn de variabelen *i* en *j* te zien. De twee pointers *p* en *q* wijzen respectievelijk naar variabele *i* en *j*. Dit is weergegeven met de twee pijlen. De inhoud van pointer *p* is dus het adres van variabele *i* en de inhoud van pointer *q* is het adres van variabele *j*.



Figuur 5.1: Uitbeelding van twee pointers naar variabelen in het geheugen wijzen.

¹ We gaan hier gemakshalve vanuit dat elke variabele precies één geheugenplaats in beslag neemt. In de praktijk bestaan variabelen en pointers meestal uit meer dan één geheugenplaats.

Een pointer kan wijzen naar een enkelvoudige variabele, een (element van een) array, een structure of (het begin van) een functie. Een pointer kan niet wijzen naar een constante, een uitdrukking en een register variabele.

Het is ook mogelijk om een pointer naar het datatype `void` te laten “wijzen”. Deze pointers worden generieke pointers genoemd. We zullen dit bespreken in paragraaf 5.3.

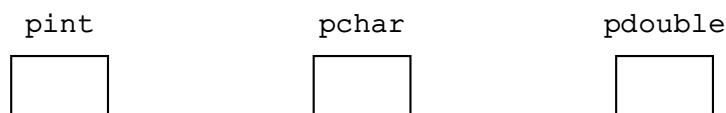
5.1 Pointers naar enkelvoudige datatypes

In listing 5.1 is de declaratie van enkele pointers van enkelvoudige datatypes te zien. Bij de declaratie moet het type variabele waarnaar de pointer wijst worden opgegeven. De asterisk (*) geeft aan dat het de declaratie van een pointer betreft.

```
int *pint;           /* pint is a pointer to an int */
char *pchar;         /* pchar is a pointer to a character */
double *pdouble;     /* pdouble is a pointer to a double */
```

Listing 5.1: Enkele declaraties van pointers.

We kunnen pointers uitbeelden door middel van vakjes, zoals te zien is in figuur 5.2. Elk vakje stelt een pointer voor. In beginsel hebben de pointers geen correcte inhoud. Er wordt dan wel gesproken dat de pointer nergens naar toe wijst, maar dat is feitelijk onjuist². Een pointer heeft altijd een adres als inhoud, maar het kan zijn dat de pointer niet naar een bekende variabele wijst.



Figuur 5.2: Voorstelling van drie pointers in het geheugen.

Aan een pointer is het adres van een variabele van hetzelfde type toe te kennen. Hiervoor gebruiken we de *adres-operator* & (ampersand). In listing 5.2 is een aantal toekenningen van adressen te zien.

```
int i;               /* the variables */
char c;
double d;

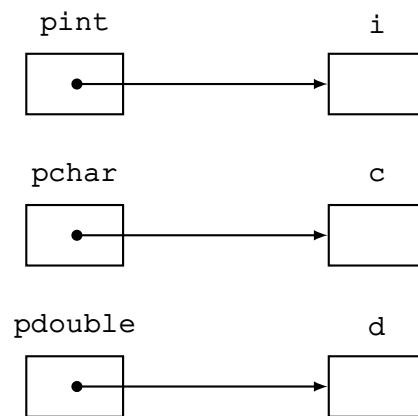
int *pint;           /* the pointers */
char *pchar;
double *pdouble;

pint = &i;           /* pint points to variable i */
pchar = &c;           /* pchar points to variable c */
pdouble = &d;         /* pdouble points to variable d */
```

Listing 5.2: Enkele toekenningen van adressen aan pointers.

² Er is één uitzondering. De NULL-pointer wordt algemeen aanvaard als een pointer die nergens naartoe wijst. Zie paragraaf 5.2.

Nu de pointers geïnitieerd zijn, kunnen we een voorstelling maken van de relatie tussen de pointers en de variabelen. Dit is te zien in figuur 5.3.



Figuur 5.3: Voorstelling van drie pointers die naar variabelen wijzen.

Via de pointers kunnen we de variabelen gebruiken. Stel dat we variabele `i` met één willen verhogen. Dat kunnen we doen door gebruik te maken van de *indirectie* of *dereference* operator `*`. Deze operator heeft voorrang op de optelling.

```
*pint = *pint + 1;    /* increment i with one */
```

Listing 5.3: Gebruik van een pointer bij een toekenning.

We mogen de dereference operator overal gebruiken waar een variabele gebruikt mag worden, bijvoorbeeld bij een optelling.

```
int i = 2, *pint;
pint = &i;
i = *pint + 1;    /* add 1 to variable i */
```

Listing 5.4: Gebruik van een pointer bij het afdrukken van een variabele.

Overigens kan tijdens declaratie ook gelijk de initialisatie van een pointer plaatsvinden. Deze declaratie kan verwarrend zijn. In onderstaande listing wordt de pointer `pint` gedeclareerd en geïnitieerd met het adres van variabele `i`. Het betreft hier dus *geen* dereference.

```
int i = 2;
int *pint = &i;    /* declare and initialize pint */
```

Listing 5.5: Declaratie en initialisatie van een pointer.

5.2 De NULL-pointer

In principe wijst een pointer naar een variabele (of beter: naar een geheugenadres). Om aan te geven dat een pointer niet naar een variabele wijst, kunnen we de *NULL-pointer* gebruiken.

Let erop dat de NULL-pointer niet hetzelfde is als een niet-geïnitieerde pointer. In C is een preprocessor-macro genaamd `NULL` te gebruiken om een pointer als NULL-pointer te initialiseren. De C-standaard schrijft voor dat `NULL` wordt gedefinieerd in `locale.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, `time.h`, en `wchar.h`. Slechts een van deze header-bestanden is noodzakelijk om `NULL` te definiëren.

```
#include <stdio.h>

int *p = NULL;    /* p is initialized as NULL-pointer */
```

Listing 5.6: Declaratie en initialisatie van een NULL-pointer.

NULL-pointers kunnen *niet* gebruikt worden bij dereferentie. Dat veroorzaakt over het algemeen dat de executie van een programma wordt afgebroken.

```
#include <stdio.h>

int *p = NULL;    /* p is initialized as NULL-pointer */

*p = *p + 1;      /* Oops, dereference of NULL-pointer! */
```

Listing 5.7: Dereferentie van een NULL-pointer.

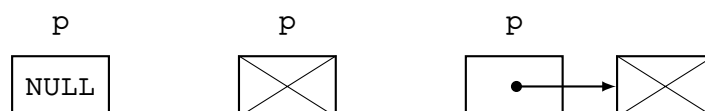
In een vergelijking zullen twee NULL-pointers altijd `true` opleveren.

```
int *p = NULL, *q = NULL;
...
if (p == q) {
    /* true */
}
```

Listing 5.8: Vergelijken van twee NULL-pointers.

De NULL-pointer wordt door diverse standaard functies gebruikt om aan te geven dat er een fout is geconstateerd. Zo geeft de functie `fopen` de waarde `NULL` terug als het niet gelukt is om een bestand te openen. De functie `malloc` geeft de waarde `NULL` terug als het niet gelukt is om een stuk geheugen te alloceren.

Bij het voorstellen van NULL-pointers zijn diverse mogelijkheden die gebruikt worden. Bij de linker voorstelling wordt het woord `NULL` in een vakje gezet, bij de middelste voorstelling wordt een kruis in het vakje gezet en bij de rechter voorstelling wordt een pijl getrokken naar een vakje met een kruis erin.



Figuur 5.4: Drie voorstellingen van NULL-pointers.

5.3 Pointer naar void

De void-pointer, ook wel *generieke pointer* genoemd, is een speciaal type pointer die naar elk type variabele kan wijzen. Een void-pointer wordt net als een gewone pointer gedeclareerd middels het keyword `void`. Toekenning aan een void-pointer gebeurt met de adres-operator `&`.

```
int i;
char c;
double d;

void *p; /* void-pointer */

p = &i; /* valid */
p = &c; /* valid */
p = &d; /* valid */
```

Listing 5.9: Declaratie en initialisatie van een void-pointer.

Omdat het type van een void-pointer niet bekend is, kan een void-pointer niet zonder meer in een deferentie gebruikt worden. De void-pointer moet expliciet gecast worden naar het correcte type.

```
int i = 2;

void *p = &i; /* void-pointer */

*(int *) p = *(int *) p + 1; /* explicit type cast */

printf("De waarde is %d\n", *(int *) p);
```

Listing 5.10: Declaratie en initialisatie van een void-pointer.

5.4 Afdrukken van pointers

Het afdrukken van de waarde van een pointer kan met de `printf`-functie en de format specifier `%p`. Merk op dat de pointer van het type `void` moet zijn, maar veel compilers accepteren pointers naar een datatype.

```
#include <stdio.h>

int main() {

    int i = 2, *p = &i;

    printf("Pointer: %p\n", (void *) p);

    return 0;
}
```

Listing 5.11: Afdrukken van een pointer.

Noot: Bij 32-bits compilers is de grootte van een pointer 32 bits (4 bytes). Zo'n pointer kan maximaal 4 GB adresseren. Bij 64-bits compilers is de grootte van een pointer 64 bits (8 bytes). Zo'n pointer kan maximaal 16 EB (exa-bytes) adresseren.

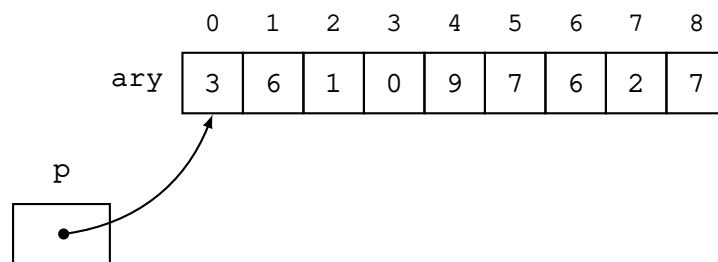
5.5 Pointers naar array's

We kunnen een pointer ook laten wijzen naar het eerste element van een array. Dit is te zien in listing 5.15. De array bestaat uit negen elementen van het type `int`. De pointer `p` laten we wijzen naar het eerste element van de array. We gebruiken hiervoor de adres-operator `&` en de index 0.

```
int ary[] = {3,6,1,0,9,7,6,2,7};  
  
int *p = &ary[0]; /* p points to first element for array */
```

Listing 5.12: Een pointer naar het eerste element van een array.

De uitbeelding hiervan is te zien in figuur 5.5.



Figuur 5.5: Uitbeelding van een pointer naar het eerste element van een array.

Omdat deze toekenning zeer vaak in een C-programma voorkomt, is er een verkorte notatie mogelijk. We kunnen in plaats van `&ary[0]` ook de naam van de array gebruiken: *de naam van een array een synoniem is voor een adres van het eerste element van de array*. Zie listing 5.13.

```
int ary[] = {3,6,1,0,9,7,6,2,7};  
  
int *p = ary; /* p points to first element of array */
```

Listing 5.13: Een pointer naar het eerste element van een array.

Omdat de naam van een array een synoniem is, mag het dus niet gebruikt worden aan de linkerkant van een toekenning.

```
int *p, ary[] = {3,6,1,0,9,7,6,2,7};  
  
p = &ary[0];      /* correct use of pointer */  
ary[2] = *p;      /* correct use of pointer and array */  
  
ary = p;          /* ERROR: name of array cannot be used */
```

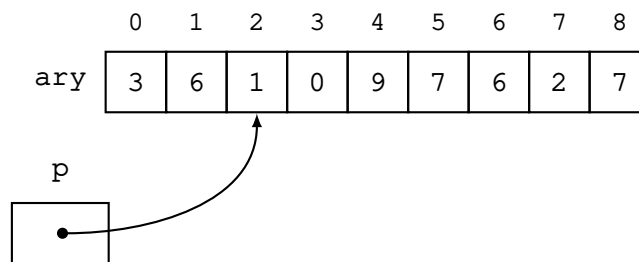
Listing 5.14: Een pointer naar het eerste element van een array.

Het is ook mogelijk om een pointer naar een ander element van een array te laten wijzen. De compiler test niet of de toekenning binnen de array-grenzen ligt.

```
int ary[] = {3, 6, 1, 0, 9, 7, 6, 2, 7};  
int *p = &ary[2]; /* p points to third element of array */
```

Listing 5.15: Een pointer naar het derde element van een array.

Een uitbeelding is te zien in figuur 5.6.



Figuur 5.6: Uitbeelding van een pointer naar het derde element van een array.

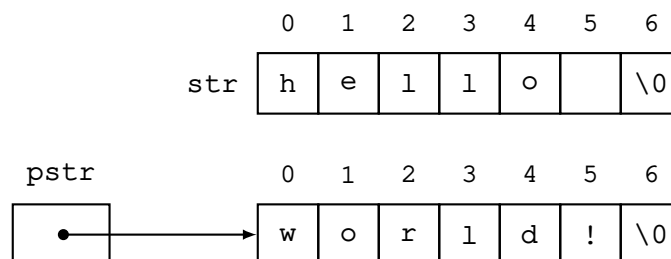
5.6 Strings

Een string in C is niets anders dan een array van karakters, afgesloten met een nul-karakter ('`\0`'). Er is dus altijd één geheugenplaats meer nodig dan het aantal karakters in de string. Een nul-karakter is niet hetzelfde als een NULL-pointer.

```
char str[] = "Hello ";  
char *pstr = "world!";
```

Listing 5.16: Declaratie en initialisatie van twee C-strings.

Merk op dat `str` niet aangepast mag worden, want dit is de naam van een array. Pointer `pstr` mag wel aangepast worden want `pstr` is een pointer naar het eerste element van de array. Een voorstelling van beide strings is te zien in figuur 5.7.



Figuur 5.7: Uitbeelding van twee C-strings.

5.7 Rekenen met pointers

Pointers kunnen rekenkundig worden aangepast dat vooral nuttig is bij het gebruik van array's. In het onderstaande programma wijst pointer `p` in eerste instantie naar het begin

van de array `ary` (dus `ary[0]`). Daarna wordt `p` twee maal met 1 verhoogd en daarna met 3 verhoogd. Bij rekenkundige operaties op pointers wordt rekening gehouden met de grootte van de datatypes. De grootte van een `int` is in de regel vier bytes. Door de pointer met 1 te verhogen wordt dus naar de volgende `int` gewezen.

```
int ary[] = {3,6,1,0,9,7,6,2,7};
int *p = ary; /* p pointe to ary[0] */

p = p + 1;    /* p points to ary[1] */
...
p = p + 1;    /* p points to ary[2] */
...
p = p + 3;    /* p points to ary[5] */
```

Listing 5.17: *Rekenen met pointers.*

Een mooi voorbeeld van het rekenen met pointers is het bepalen van de lengte van een C-string. In listing 5.18 wordt pointer `str` gedeclareerd en wijst naar het begin van de string. Pointer `begin` wijst ook naar het begin van de string. Daarna verhogen we pointer `str` totdat het einde van de string is bereikt. Daarna drukken we het verschil van de twee pointers af.

```
#include <stdio.h>

int main() {

    char *str = "Hallo wereld!";
    char *begin = str;

    while (*str != '\0') { /* while not end of string ... */
        str = str + 1;    /* point to the next character */
    }

    printf("Lengte is %d\n", str-begin);
}
```

Listing 5.18: *Berekenen van de lengte van een string met behulp van pointers.*

Let erop dat de twee pointers naar elementen in dezelfde array moeten wijzen (of één na het laatste element). Alleen dan levert de aftrekking `str-begin` een gedefinieerd resultaat. De aftrekking is van het type `ptrdiff_t` (dat meestal gelijk is aan een `int`) en levert het verschil in elementen. Het onderstaande programmafragment geeft als uitvoer de waarde 3.

```
int ary[] = {3,6,1,0,9,7,6,2,7};
int *p = &ary[2];
int *q = &ary[5];

printf("Verskil is %d\n", q-p);
```

Listing 5.19: *Het berekenen van het verschil van twee pointers.*

Vergelijken van twee pointers kan ook. Zo kunnen pointers op gelijkheid worden vergeleken, maar ongelijkheid kan ook. We zouden de `printf`-regel van listing 5.18 kunnen

vervangen door de onderstaande programmafragment. Uiteraard moeten de twee pointers naar hetzelfde datatype wijzen.

```
if (str>begin) {
    printf("Lengte is %d\n", str-begin);
} else {
    printf("De string is leeg\n");
}
```

Listing 5.20: *Vergelijken van twee pointers.*

5.8 Pointers als functie-argumenten

Net als “gewone” variabelen, kunnen ook pointers als argumenten bij het aanroepen van een functie gebruikt worden. Een typisch voorbeeld is een functie die een string kopieert naar een andere string. De functie krijgt twee pointers naar strings als argumenten mee. Bij het aanroepen van de functie worden de namen van de twee strings als argumenten meegegeven. De naam van een string is immers een pointer naar het eerste karakter van de string. Deze manier van argumentenoverdracht wordt *Call by Reference* genoemd.

```
void string_copy(char *to, char *from) {

    while (*from != '\0') {    /* while not end of string ... */
        *to = *from;          /* copy character */
        to = to + 1;          /* point to the next character */
        from = from + 1;
    }
    *to = '\0';               /* terminate string */
}

int main() {

    char stra[] = "Hello world!";
    char strb[100];

    string_copy(strb, stra); /* copy stra to strb */
}
```

Listing 5.21: *Functie voor het kopiëren van een string.*

Merk op dat de geheugenruimte voor de kopie groot genoeg moet zijn om de kopie op te slaan en dat de twee string elkaar in het geheugen niet mogen overlappen.

Noot: een array kan alleen maar via een pointer als argument aan een functie worden doorgegeven. Dit is veel efficiënter dan de hele array mee te geven. In het bovenstaande programma wordt dus *niet* de hele array meegegeven, maar alleen de pointers naar de eerste elementen. We mogen daarom de pointers *to* en *from* ook als namen van array's beschouwen. Zie onderstaande listing.

```
void string_copy(char to[], char from[]) {
```

```

int i=0;

while (from[i] != '\0') {    /* while not end of string ... */
    to[i] = from[i];        /* copy character */
    i = i + 1;              /* point to the next character */
}
to[i] = '\0';               /* terminate string */
}

```

Listing 5.22: Functie voor het kopiëren van een string.

5.9 Pointer als return-waarde

Een pointer kan ook als return-waarde dienen. In het onderstaande voorbeeld wordt in een string gezocht naar een bepaalde karakter in een string. Als het karakter gevonden is, wordt een pointer naar het karakter teruggegeven. Als het karakter niet wordt gevonden wordt NULL teruggegeven. Na het uitvoeren van de functie moet hier op getest worden. Tevens wordt in het begin getest of de pointer naar de string wel een geldige waarde heeft. Geldig wil zeggen dat de pointer niet NULL is. Het is *good practice* om altijd te testen of een pointer NULL is.

```

char *find_token(char *str, char ch) {

    if (str == NULL) {        /* sanity check */
        return NULL;
    }

    while (*str != '\0') {    /* while not end of string ... */
        if (*str == ch) {    /* if character found ... */
            return str;      /* return pointer */
        }
        str++;
    }
}

```

CALL BY REFERENCE...

In de programmeerwereld is het gebruikelijk om onderscheid te maken tussen twee manieren van argumentoverdracht: *Call by Value* en *Call by Reference*. Bij *Call by Value* wordt een kopie van de waarde van een variabele aan de functie meegegeven. Alleen de kopie kan veranderd worden door de functie. De variabele waarvan de kopie is gemaakt kan dus niet op deze manier veranderd worden. Bij *Call by Reference* wordt het adres van de variabele aan de functie meegegeven. Via dit adres is het dus wel mogelijk om de originele variabele te veranderen.

Sommige programmeurs beweren dat *Call by Reference* eigenlijk niet bestaat. En daar is wat voor te zeggen. Er wordt immers een waarde aan de functie meegegeven en dat is het adres van een variabele. Dit adres kan in de functie veranderd worden maar het originele adres waar de variabele in het geheugen staat wordt niet aangepast. Toch maken programmeurs onderscheid tussen deze twee manieren van argumentenoverdracht.


```

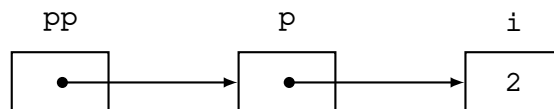
    }

    return NULL;          /* character not found */
}

```

Listing 5.23: Pointer als return-waarde.

5.10 Complexe pointers: pointer naar pointer



Figuur 5.8: Uitbeelding van een pointer naar een pointer naar een *int*.

```

int i = 2;          /* the integer */
int *p = &i;        /* p points to i */
int **pp = &p;      /* pp points to p */

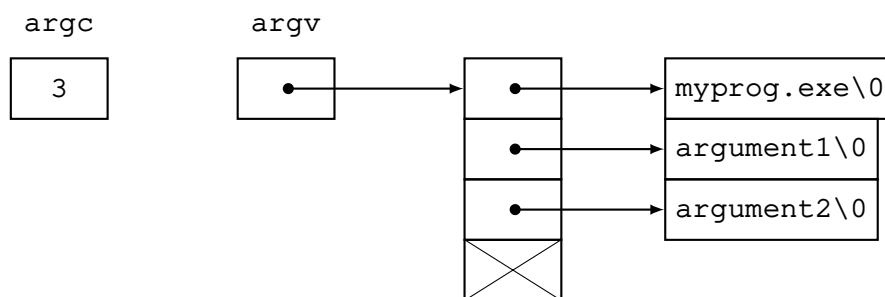
printf("De waarde is %d\n", **pp);

```

Listing 5.24: Voorbeeld van een pointer naar een pointer.

5.11 Complexe pointers: pointer naar array van pointers

5.12 Argumenten meegeven aan een C-programma



Figuur 5.9: Voorstelling van de variabelen *argc* en *argv*.

5.13 Pointers naar functies

Functies zijn stukken programma die ergens in het geheugen liggen opgeslagen. De naam van een functie is het adres van de eerste instructie van de functies. Het is dus mogelijk om de naam van een functie te gebruiken als een pointer.

