Week 3 : Hoofdstukken 7 en 8; eXtra stof: invoer met scanf()

e**X**tra aanvulling op het boek: ***inlezen met scanf()*** in C

Voorbeeld Extra_4

In dit voorbeeld lezen we in van toetsenbord met `scanf()`. Let op de correcte aanroep van scanf() : `scanf("%d", &n);`

```
#include <stdio.h>

int main(void)
{
  int n = 2;
  printf("Tik nu de waarde voor n in : ");
  scanf("%d", &n);
  printf("n = %d\n\n", n);

  int dag, maand, jaar;
  printf("Tik in dag-maand-jaar als dd/mm/jjjj : ");
  scanf("%d/%d/%d", &dag, &maand, &jaar);
  printf("dag = %d maand = %d jaar = %d\n", dag, maand, jaar);

  printf("Tik nu in dg-mnd-jr als dd-mm-jjjj : ");
  scanf("%2d-%2d-%4d", &dag, &maand, &jaar);
  printf("dag = %d maand = %d jaar = %d\n", dag, maand, jaar);

  char ch1, ch2;
  printf("Tik nu dg-mnd-jr als dd-mm/jjjj : ");
  scanf ("%d%c%d%c%d", &dag, &ch1, &maand, &ch2, &jaar);
  printf("dag = %d ch1 = %c maand = %d ch2 = %c jaar = %d\n",
                              dag, ch1, maand, ch2, jaar);
  return 0;
}
```

Uitvoer:

```
Tik nu de waarde voor n in : 34
n = 34

Tik in dag-maand-jaar als dd/mm/jjjj : 31/8/2005
dag = 31 maand = 8 jaar = 2005
Tik nu in dg-mnd-jr als dd-mm-jjjj : 31-8-2005
dag = 31 maand = 8 jaar = 2005
Tik nu dg-mnd-jr als dd-mm/jjjj : 31-8/2005
dag = 31 ch1 = - maand = 8 ch2 = / jaar = 2005    // opm: 31$8@2005 ook OK
```

Opmerking
De functie-aanroep van `scanf()` is (evenals `printf()`) van de vorm:

```
scanf(formaat, argument1, argument2, . . .);
```

Elk argument `argument-i` is bestemd voor het inlezen van één variabele. Die ingelezen waarde moet door de functie `scanf()` worden ingelezen, en vervolgens uit `scanf()` worden **uitgevoerd** als zgn. **uitvoer**-parameter. ***Uitvoer-parameters*** heten ook wel ***var-parameters*** (die ***var*** komt van ***variabele***). In feite wordt er helemaal niet een waarde uit de functie uitgevoerd. De functie-aanroep

```
scanf("%d", &n);
```
(of ook: `scanf("%i", &n);`)

leidt tot de volgende aktie:
- **n** is een variabele van type **signed** `int` van de functie `main()`
- aan functie `scanf()` wordt mee gegeven het ***adres*** van variabele **n** in het interne geheugen (laten we aannemmen dat dit geheugen-adres bijv. 5678 is). Binnen de functie `scanf()` wordt de waarde van de variabele vervolgens ingelezen in de ***geheugenplaats***, die behoort bij ***adres*** 5678.
- Na afloop van de aanroep van `scanf()` kunnen we in functie `main()` beschikken over de waarde van variabele `n` .
- We kunnen ook altijd het geheugen-adres van variabele **n** opvragen. Dit doe je met de opdracht:

```
printf("Adres van n is : %X", &n);
```

In Java zijn **alle** parameters **altijd invoer**-parameters. Dat mechanisme heet ook wel ***Call-by-value***. De bijbehorende parameters heten ***value-parameters*** of ***waarde-parameters***.

In C bestaan naast de ***value-parameters*** dus ook de ***var-parameters***. In C moet de programmeur altijd een **keuze** maken. Via de ***return-waarde*** kun je **één waarde uitvoeren** uit de functie. Door gebruik te maken van ***uitvoer-parameters*** heb je in C de mogelijkheid om **meer dan één waarde uit te voeren** uit een functie. Dat biedt veel meer mogelijkheden bij het programmeren.

Functies worden behandeld in Hoofdstuk 11 (in Week 5). Geheugen-***adressen*** en ***pointers*** in Hoofdstuk 10 (Week 4).

Met de functie `scanf()` worden getallen ingelezen van toetsenbord. De functie geeft een `int`-getal als return-waarde terug.

- Het resultaat, dat wordt afgeleverd, is het **aantal** geslaagde toekenningen. Je kunt deze return-waarde laten afdrukken op beeldscherm:

```
int n = 2;
printf("Tik nu de waarde voor n in : ");
printf("scanf() = %d\n", scanf("%d", &n));

int dag, maand, jaar;
printf("Tik in dag-maand-jaar als dd/mm/jjjj : ");
printf("scanf() = %d\n",
               scanf("%d/%d/%d", &dag, &maand, &jaar));
char ch1, ch2;
printf("Tik nu dg-mnd-jr als dd-mm/jjjj : ");
printf("scanf() = %d\n",
  scanf ("%d%c%d%c%d", &dag, &ch1, &maand, &ch2, &jaar));
```

  Bij runnen wordt afgedrukt op beeldscherm:

```
Tik nu de waarde voor n in : 56 <ENTER>
scanf() = 1
Tik in dag-maand-jaar als dd/mm/jjjj : 11/09/2001 <ENTER>
scanf() = 3
Tik nu dg-mnd-jr in als dd-mm/jjjj : 11/09/2001 <ENTER>
scanf() = 5
```

- Als het succesvol gelezen aantal getallen kleiner is dan het aantal opgegeven argumenten, is het lezen voortijdig afgebroken. Omdat het einde van de file is bereikt, of omdat er een fout is opgetreden.
- Als het einde van de file wordt bereikt voordat er toekenningen zijn uitgevoerd, levert de functie als resultaat `EOF` af.

Met de functie `scanf()` kunnen ook characters en strings worden ingelezen van toetsenbord. Bij het inlezen van getallen worden zgn. ***white space characters*** (***witruimtekarakters***) voorafgaande aan het eerste getal over geslagen. De ***white space characters*** tussen de getallen in fungeren als scheiding van de invoergetallen en worden over geslagen. ***White space characters*** zijn de spatie, TAB en ENTER (in C-notatie: `' '` en `'\t'` en `'\n'`)

Bij invoer met conversiespecificatie `%c` wordt ieder karakter gelezen, dus ook white space characters.

Bij invoer van ***strings***, met conversiespecificatie `%s`, worden voorafgaande white space characters over geslagen; en vervolgens wordt gelezen tot aan het eerste white space character. Wanneer je een hele zin wilt inlezen is `scanf()` dus niet geschikt, omdat per keer één woord wordt gelezen. Inlezen van een zin gaat wel met `gets()` en (beter) met `fgets()`

Voor type `char` zijn `printf()` en `scanf()` nogal omslachtig. In dat geval kun je beter werken met `getchar()` en `putchar()`.

In plaats van de code:

```
char ch;
scanf("%c", &ch);
printf("%c", ch);
```

krijg je dan:

```
char ch;
ch = getchar();
putchar(ch);
```

Voorbeeld Extra_5

Dit is een voorbeeld met `scanf()` voor de typen `float` en `double`.

```
#include <stdio.h>

int main(void)
{
  float x;
  double y;
  printf("Invoer: "); scanf("%f", &x);
  printf("Uitvoer: "); printf("x = %f\n", x);

  printf("Invoer: "); scanf("%lf", &y);
  printf("Uitvoer: "); printf("y = %f\n", y);
  printf("Uitvoer: "); printf("y = %E\n", y);

  printf("Invoer: "); scanf("%f", &x);
  printf("Uitvoer: "); printf("x = %f\n", x);

  printf("Invoer: "); scanf("%3f%lf", &x, &y);
  printf("Uitvoer: "); printf("x = %f en y = %f\n", x, y);

  printf("Invoer: "); scanf("%le", &y);
  printf("Uitvoer: "); printf("y = %f\n", y);

  printf("Invoer: "); scanf("%2f%lg", &x, &y);
  printf("Uitvoer: "); printf("x = %f en y = %f\n", x, y);

  return 0;
}
```

Uitvoer:

Invoer: ΔΔ12.34
Uitvoer: x = 12.340000
Invoer: 1.23456789
Uitvoer: y = 1.234568
Uitvoer: y = 1.234568E+000
Invoer: - 1.2508E+3
Uitvoer: x = - 1250.800049
Invoer: - 6.21E+1
Uitvoer: x = - 6.000000, y = 210.000000
Invoer: 1.
Uitvoer: y = 1.000000
Invoer: 1.79
Uitvoer: x = 1.000000, y = 79.000000

De ***conversies*** `e`, `f` en `g` verwachten als ***argument*** een ***pointer*** naar een `float`. Er vindt conversie plaats naar een `signed` decimaal floating-point getal met toekenning aan een variabele van type `float`, `double` of `long`.

Met een ***lengtespecificatie*** `l` ervoor wordt een ***pointer*** naar een `double` verwacht (en met ***lengtespecificatie*** `L`: een `long double`).

Met de ***modifier*** * lees je wel, maar je onderdrukt de toekenning.
Het teken Δ staat voor een **spatie** (een echte spatie is slecht zichtbaar

Bij `printf()` hebben de conversieoperaties e en E tot gevolg dat in de uitvoer een e resp. E komt te staan (E+3 betekent 10^3).
Je kunt de ***veldbreedte*** opgeven; en bij gebroken getallen de ***precisie*** (dwz het aantal decimalen). Met `printf("%8.2f", y);` wordt getal `y` afgedrukt met 2 ***decimalen***, op in totaal 8 ***posities*** (dus inclusief decimale punt, eventueel teken, en de 2 decimalen)

Hoofdstuk 7: Herhaalstructuren

Eerst komt een voorbeeld met een ***for-statement***; vervolgens één met een ***while-statement***; en tenslotte één met een ***do . . . while-statement***

Voorbeeld 7_1 (dit is Voorbeeld 6_6 uit Les 2)
Dit programma drukt een gelijkbenige driehoek van sterretjes af. De eerste regel bestaat uit 1 sterretje, de tweede uit 3 sterretjes, gecentreerd onder de eerste. Etc. Er zijn 5 regels in totaal.

```
#include <stdio.h>
#define AANTAL  5          // als je 6 regels wilt maken: verander 5 in 6

int main(void)
{
  int regel, spaties, ster;
  for (regel = 1 ; regel <= AANTAL ; regel++)
  {
    for (spaties = 1 ; spaties <= AANTAL - regel ; spaties++)
    {
      putchar(' ');
    }

    for (ster = 1 ; ster <= 2*regel - 1 ; ster++)
    {
      putchar('*');
    }
    putchar('\n');
  }

  printf("Dit zijn %d regels met * \' s\n", AANTAL);
  return 0;
}
```

Uitvoer:

```
    *
   ***
  *****
 *******
*********
```

Dit zijn 5 regels met * ' s

Opmerking

Het for-statement heeft de vorm:

```
for (initialisatie ; conditie ; ophoging/verlaging)
  statement
```

Dit is equivalent met:

```
initialisatie;            // 1
while ( conditie )        // 2a, b, c, . . .
{
  statement;              // 3a, b, c, . . .
  ophoging/verlaging;     // 4a, b, c, . . Is de allerlaatste aktie !
}
```

1. De ***initialisatie*** wordt precies één keer uitgevoerd, nl helemaal aan het begin. Daarna nooit meer.
2. Vervolgens wordt de ***conditie*** getest. Heeft de conditie de waarde false (in C dus: 0), dan wordt de hele lus overgeslagen. Heeft de coditie de waarde true (in C dus elke int-waarde != 0), dan wordt
3. het ***statement-blok*** uitgevoerd. Na dit statement-blok wordt de
4. ***lopende variabele*** opgehoogd/verlaagd.

Daarna wordt de ***conditie*** getest. Is die false, dan stopt de lus; is die true dan wordt opnieuw het ***statement-blok*** uitgevoerd; en de ***lopende variabele*** opgehoogd, verlaagd. Etc.

(2) Voorbeeld met start-conditie = false:

```
for (int i = 0 ; i < 0 ; i++)
  printf("test\n");
```

Hier gebeurt niets, omdat de conditie waarde `(0 < 0)= false` heeft.

(1) `int i = 0;` is een ***statement***. En `i = 0` is een ***expressie***
Ik noem het: ***initialisatie***

(3) Dat ***statement*** is vaak een heel ***blok*** `{dit soort accoladen, met opdrachten ertussen}`
Een ***statement*** kan ook bestaan uit één enkele opdracht.

(4) Variabele `i` heet de ***lopende variabele***. Dat mag zijn: alles wat `++` heeft (en dan bestaat dus eveneens de `--`). Dus types als `int, char, float,` ***pointertypes***, . . .

Voorbeeld waarin de lopende variabele een `char` is:

```
char ch;
for ( ch = 'a' ; ch <= 'z' ; ch++ )
  putchar(ch);
putchar('\n');
```

Uitvoer:
abcdefghijklmnopqrstuvwxyz

Opmerking 1 bij (4):
Bovenstaande code is correct. Er wordt altijd één statement uitgevoerd in nummertje (3) van blz 8. Je hoeft van `putchar(ch);` dus geen compound statement te maken, maar het mag wel:

```
char ch;
for ( ch = 'a' ; ch <= 'z' ; ch++ )
{
  putchar(ch);
}
putchar('\n');
```

Opmerking 2 bij (4):
Onderstaande code is wel correct, maar het programma doet niet wat de programmeur zou verwachten. Dat is de schuld van de punt-komma `;`

```
char ch;
for ( ch = 'a' ; ch <= 'z' ; ch++ ) ;
  putchar(ch);
putchar('\n');
```

Er wordt (in werkelijkheid) uitgevoerd:

```
char ch;
for ( ch = 'a' ; ch <= 'z' ; ch++ )
{   }
putchar(ch);              // na de 'z' komt de '{' in de ASCII-tabel
putchar('\n');
```

Voorbeeld 7_2 (zie ook 7.2 in het boek)

Het is in C gebruikelijk om tegelijkertijd in te lezen, en – per direct – te testen op de ***afsluiter***. Dit kan met `getchar()` ; en ook met `scanf()`

Het boek kiest voor ***afsluiter*** `EOF` bij lezen van karakters. Ik kies voor een karakter omdat dat implementatie-onafhankelijk is.

```
#include <stdio.h>

int main(void)
{
  char ch;
  printf("Tik maar wat in; sluit invoer af met een punt:\n");
  while ( (ch = getchar()) != `.' )
    putchar(ch);
  printf("\nEINDE van getchar()\n\n");

  float getal, totaal = 0.0f;
  printf("Tik aantal floats in; sluit af met getal < 0:\n");
  while ( scanf("%f", &getal) && getal > 0.0 )
    totaal += getal;
  printf("%3.2f\nEINDE van scanf()\n", totaal);
  return 0;
}
```

Uitvoer 1:

Tik maar wat in; sluit invoer af met een punt:
De invoer is gebufferd. Dat komt later wel.<ENTER>
De invoer is gebufferd (opm: dus niet de punt . en niet de tweede zin)
EINDE van getchar()

Tik aantal floats in; sluit af met getal < 0:
0.00 (opm: de rotzooi in de buffer ruineert het verdere programma)
EINDE van scanf()

Uitvoer 2:

Tik maar wat in; sluit invoer af met een punt:
De invoer is gebufferd.<ENTER>
De invoer is gebufferd
EINDE van getchar()

Tik aantal floats in; sluit af met getal < 0:
1.23 4.65 -9.0 <ENTER>
5.88
EINDE van scanf()

Opmerking
De constructies

```
while ( scanf("%f", &getal) && getal > 0.0 ){    }
```

resp

```
while ( (ch = getchar()) != '.' ){   }
```

zijn typisch C : zowel `scanf()` als `getchar()` leveren een return-waarde af.

scanf()
met ***boolean conditie***

```
while ( scanf( "%f", &getal) && getal > 0.0) {    }
```

wordt uitgevoerd als (de > heeft hogere proriteit dan && Dus gaat goed)

```
while ( scanf( "%f", &getal) && (getal > 0.0)) {    }
```

Het return-type van `scanf()` is een `int`. Het resultaat dat wordt afgeleverd is het **aantal** geslaagde toekenningen . Als dit aantal kleiner is dan het aantal opgegeven argumenten, is het lezen voortijdig afgebroken, omdat het einde van de file is bereikt, of omdat er een fout is opgetreden.
Als het einde van de file wordt bereikt voordat er toekenningen zijn uitgevoerd, levert de functie als resultaat `EOF` af.

getchar()
met ***boolean conditie***

```
while ( ch = getchar() != '.' ){   }
```

ontstaat een typische C-fout. De operator `=` heeft lagere prioriteit dan de operator `!=` Dan wordt deze code dus uitgevoerd als:

```
while ( ch = (getchar() != '.') ){   }
```

Er wordt eerst een karakter ingelezen met `getchar();` vervolgens wordt de return-waarde van `getchar()` vergeleken met `'.'` (is vrijwel altijd ongelijk, dus true, dus `int` 1) En teslotte wordt dit `int`-resultaat toegekend aan `ch` (opm: ASCII-teken met `int`-waarde 1 is niet-afdrukbaar)

Voorbeeld 7_3
De ***do . . . while*** heeft als meest kenmerkende eigenschap dat eerst de statement(s) worden uitgevoerd, en dat pas daarna wordt getest op de conditie. (de ***while*** test eerst, en voert daarna pas – afhankelijk van de uitkomst – de statement(s) uit).
In onderstaand voorbeeld moet eerst worden ingelezen hoeveel getallen verwerkt moeten worden. Wanneer dat aantal getallen niet correct kan worden aangeleverd, heeft de rest van het programma geen zin. Daarom is het inlezen van dit aantal getallen in een lus geplaatst. In dit geval een ***do. . .while***, omdat minimaal één keer de waarde voor `aantal` moet worden ingelezen.

```
#include <stdio.h>

int main(void)
{
  int teller, aantal;
  float getal, som = 0.0f;
  do
  {
    printf("Hoeveel getallen moeten verwerkt? ");

  } while ( scanf("%d", &aantal) && aantal <=0 );
  // nu is ( aantal > 0 ) == true

  teller = aantal;
  printf("Tik nu %d float-getallen in: ", teller);
  while ( teller -- )
  {
    scanf("%f", &getal);
    som += getal;
  }
  printf("Het gemiddelde is : %4.2f\n", som/aantal);
  return 0;
}
```

Uitvoer:
Hoeveel getallen moeten verwerkt? -9
Hoeveel getallen moeten verwerkt? 0
Hoeveel getallen moeten verwerkt? 6
Tik nu 6 float-getallen in: 4 5 6 7 8 9 10 <ENTER>
Het gemiddelde is : 6.50

Opmerking: er zijn 7 getallen ingetikt; in de buffer staat nog 10 <ENTER>

Voorbeeld 7_4 (break en continue)

Dit programma leest een aantal `float`-getallen van toetsenbord. Elk getal stelt een **lengte** voor. Het programma berekent **het gemiddelde** van al deze lengten. Voor elke lengte geldt de **eis** : 0.0 <= lengte <= 250.0 Ingevoerde lengtes, die niet aan deze eis voldoen, moeten worden genegeerd (=continue). Er worden `MAX` getallen ingelezen, tenzij er iets fout loopt bij het inlezen. In dat geval moet worden gestopt met inlezen (= spring uit de for-lus met een break). In het voorbeeld is voor MAX de waarde 12 gekozen. Er had ook 12000 kunnen staan.

```
#include <stdio.h>
#define MAX 12

int main(void)
{
  int aantal = 0;
  float lengte, som = 0.0f;

  for (   ; aantal < MAX ;   )
  {
    printf("Tik in lengte nr %d : \n", 1+aantal);

    if ( !scanf("%f", &lengte) )
      break;

    if ( lengte <= 0.0 || lengte >= 250.0 )
      continue;

    som += lengte;
    aantal++;
  }

  if ( aantal != 0 ) {
    printf("Gemiddelde = %4.2f\n", som/aantal);
  }
  return 0;
}
```

Uitvoer:

```
Tik in lengte nr 1 : -6
Tik in lengte nr 1 : 3
Tik in lengte nr 2 : 6
Tik in lengte nr 3 : 8
Tik in lengte nr 4 : g
Gemiddelde = 5.67
```

Voorbeeld 7_5 A
C staat toe om open plekken te hebben in de for-lus:

```
int i;
for ( i = 0 ;     ; i++ )
{
  // voer hier van alles uit
  if (gewensteEindsituatie)
    break;
}
```

Hier staat **geen conditie** binnen de for-lus. Er wordt uit de lus gesprongen met een `break`. Dat mag van C, maar niet van de juf ("De break is voor slechte programmeurs."). Dit fragment is heel eenvoudig te herschrijven tot:

```
int i;
for ( i = 0 ;  !gewensteEindsituatie ; i++ )
{
  // voer hier van alles uit
  // en als die gewensteEindsituatie true geworden is, stopt de lus, omdat in dat
  // geval geldt dat !gewensteEindsituatie = 0
}
```

In sommige gevallen **moet** je een break gebruiken, maar dat komt heel weinig voor.

Voorbeeld 7_5 B
Onderstaande code wordt toegestaan door de compiler, maar niet door de juf.

```
#include <stdio.h>

int main(void)
{
  int i;
  for (i = 19 ; i > 0 ; i = i - 3)    // dit mag best
  {
    i -= 1;                           // gruwelijk slechte code
    printf("%d ", i);
  }
  putchar('\n');
  return 0;
}
```

Uitvoer:
18 14 10 6 2

De for-lus is speciaal bedoeld voor lussen waarvan het aantal slagen van te voren reeds bekend is. Dat is hier niet het geval. Het was gebruikelijker geweest om hier te kiezen voor een ***while-statement***. Maar het kan best wel met een ***for-statement***

Voorbeeld 7_5 C
Dit programma leest een aantal karakters van toetsenbord, en "echo" t ze naar het beeldscherm. De punt is afsluiter.

```
#include <stdio.h>
#define AFSLUITER    '.'

int main(void)
{
  char c;
  for (  ; (c = getchar()) != AFSLUITER ;  ) {
    putchar(c);
  }
  putchar('\n');
  return 0;
}
```

Uitvoer:
C lijkt op Java<ENTER>
C lijkt op Java
Denk om de punt.<ENTER>
Denk om de punt

Voorbeeld 7_6

Geneste lussen in C gedragen zich hetzelfde als geneste lussen in Java. Onderstaand programma heeft geneste lussen. Er wordt een plaatje op het beeldscherm getekend.

```
#include <stdio.h>

int main(void)
{
  int i, j;
  for ( i = 0 ; i < 10 ; i ++)
  {
    for (j = 10 –i ; j >= 0 ; j--)
    {
      if (j/2 == i)
        printf(“$”);
      else
        printf(“*”);
    }
    printf(“\n”);
  }
  printf(“\n”);
  return 0;
}
```

Uitvoer:

```
*********$$
******$$**
***$$****
$$******
*******
******
*****
****
***
**
```

Voorbeeld 7_7
De invoer van onderstaand programma bestaat uit een (onbekend) aantal positieve float-getallen. De invoer wordt **afgesloten door een negatief getal**. Er moet op het beeldscherm worden afgedrukt: het **kleinste** ingevoerde getal, het **een-na-kleinste**, en het **grootste** ingevoerde getal

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

int main(void)
{
  float getal;
  float min = FLT_MAX, ena = FLT_MAX, max = 0.0;
  int teller = 0;
  printf("Tik een aantal float getallen in;
                       sluit af met een negatief getal\n");
  while ( scanf("%f", &getal) && getal > 0.0 )
  {
    teller++;
    if ( getal > max ) max = getal;     // beslist geen else doen

    if ( getal < min )                  // eenvoudige if
    {
      ena = min;
      min = getal;
    }
    else          // nu is getal >= min ; het geval getal == min hoeft niet
    {
      if ( getal > min && getal < ena )
        ena = getal;
    }
  }
  printf("Er zijn %d getallen ingevoerd\n", teller);
  printf("min = %2.2f ena = %2.2f max = %2.2f", min,ena,max);
}
```

Uitvoer:
Tik een aantal float getallen in; sluit af met een negatief getal
6 4 5 -9<ENTER>
Er zijn 3 getallen ingevoerd
min = 4.00 ena = 5.00 max = 6.00

Hoofdstuk 8: Arrays en herhalingen.

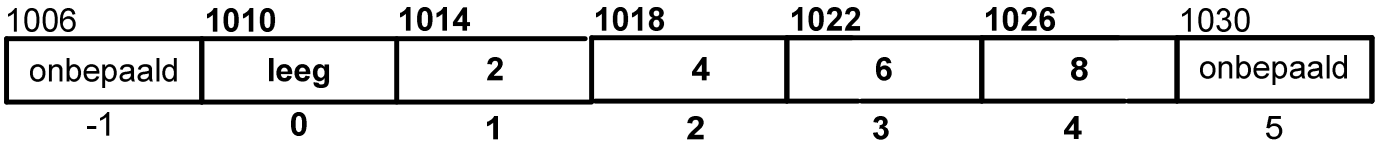
Voorbeeld 8_1
In dit voorbeeld wordt een C-array gebruikt. Pas op met het forse verschil tussen array's in C en array's in Java.

```
#include <stdio.h>

int main(void)
{
  int reeks[5];
  int i;
  for (i = 1 ; i < 5 ; i++) {
    reeks[i] = 2*i;
  }

  printf("waarde van reeks[0] is %d\n", reeks[0]);
  printf("waarde van reeks[1] is %d\n", reeks[1]);
  printf("waarde van reeks[4] is %d\n", reeks[4]);
  printf("waarde van reeks[5] is %d\n", reeks[5]);
  printf("waarde van reeks[-1] is %d\n", reeks[-1]);
  return 0;
}
```

Uitvoer:
waarde van reeks[0] is 2009118740 ➔
waarde van reeks[1] is 2
waarde van reeks[4] is 8
waarde van reeks[5] is 51575844 ➔
waarde van reeks[-1] is 5 ➔



array reeks begint op adres 1010
Er zijn 5 geheugenplaatsen gereserveerd,
namelijk voor index 0, 1, 2, 3 en 4

Een array in C IS **geen** pointer; maar hij GEDRAAGT zich **wel** als een pointer.

Voorbeeld 8_2
Dit Voorbeeld 8_1, maar dan in een fraaiere stijl geprogrammeerd.

```
#include <stdio.h>
#define AR_SIZE   5

int main(void)
{
  int reeks[AR_SIZE];
  int i;
  for (i = 0 ; i < AR_SIZE ; i++) {
    reeks[i] = 2*i;
  }

  for (i = 0 ; i < AR_SIZE ; i++) {
    printf("reeks[%d] = %d\n", i, reeks[i]);
  }
  printf("waarde van reeks[5] is %d\n", reeks[5]);
  printf("waarde van reeks[-1] is %d\n", reeks[-1]);
  return 0;
}
```

Uitvoer:
reeks[0] = 0
reeks[1] = 2
reeks[2] = 4
reeks[3] = 6
reeks[4] = 8
waarde van reeks[5] is 51575844 ➔
waarde van reeks[-1] is 5 ➔

Opmerking:
Globale variabelen zijn – bijna altijd – verboden. De enkele uitzondering die er is, is dan meestal een array.

Voorbeeld 8_3
Er bestaan in C ook ***strings*** (C-strings verschillen werelden van Java-strings). Het karakteristieke kenmerk van iedere C-string is het afsluitende ***nulkarakter*** `'\0'` = 0000 0000
Dit ***nulkarakter*** geeft het einde van de string aan. Zonder afsluitend nulkarakter eindigt een string dus niet. De compiler "kent" het begin-adres van de string (in dit vb is dat het adres van `zin[0]`).

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

int main(void)
{
  char zin[100];            // nu is het nog geen string
  char c;
  int teller = 0;
  while ( (c = getchar()) != '\n' && teller < 99)
  {
    zin[teller] = c;
    teller++;
  }
  zin[teller] = '\0';       // nu is het wel een string
  printf("%s\n", zin);
  printf("%c\n", zin[0]);
  printf("%d\n", &zin[0]);
  return 0;
}
```

Uitvoer:
De blaadjes gaan vallen <ENTER>
De blaadjes gaan vallen
D
2293504 ➔

Opmerking:
Bij de opdracht

```
  printf("%s\n", zin);
```

wordt begonnen op adres 2293504 en de inhoud van die geheugenplaats wordt afgedrukt. En vervolgens wordt iedere keer 1 byte bij het adres opgeteld (want type `char`) en de inhoud afgedrukt, net zo lang **tot het nulkarakter** wordt aangetroffen. Wordt dat nulkarakter niet gevonden, dan heb je een probleem.

Let op de verschillen!!:
Er bestaat een essentieel verschil tussen een array en een ***karakterstring***.

- `int` **`ari1`**`[] = { 13, -8, 7, 199, 24 };`
  `ari1` heeft type `int[5]`.
  `ari1``[0] = 13`
  `ari1``[4] = 24`
  `ari1``[5]` bestaat niet, maar de compiler klaagt niet.

- `int` **`ari2`**`[100];`
  En vervolgens worden een aantal getallen ingelezen in de array. De array zal niet noodzakelijkerwijs altijd helemaal gevuld zijn. Bij array's van getallen moet je daarom altijd **met een tellertje bijhouden** tot hoever de array is gevuld.

- `char` **`arc`**`[] = { 'H', 'a', 'l', 'l', 'o' };`
  `arc` is een array van karakters, maar **`arc`** is **geen karakterstring!!** Dit soort character-array wordt weinig gebruikt. Je moet ook in dit geval met een tellertje bijhouden tot hoever de array is gevuld. Wanneer je één extra item toevoegt namelijk het zogenaamde ***nulkarakter*** (`'\0'` : 0000 0000) krijg je de beschikking over veel meer mogelijkheden. Zie d)
  `arc` heeft type **`char[5]`**

- `char` **`ars`**`[] = { 'H', 'a', 'l', 'l', 'o', '\0'};`
  Nu is **`ars`** een ***karakterstring***, omdat het einde van de array wordt aangegeven d.m.v. het ***nulkarakter***. En **`ars`** heeft type **`char[6].`**
  Je kunt nu gebruikmaken van de vele faciliteiten van de standaard-bibliotheek `string.h` Het einde van een string wordt aangegeven door een nulkarakter. Daarbij is het irrelevant of er nog waarden in de array staan opgeslagen achter dit nulkarakter.

Karakterstrings: declaratie en initialisatie

```
char ars1[] = { 'H', 'a', 'l', 'l', 'o', '\0'};
char ars2[6] = { 'H', 'a', 'l', 'l', 'o', '\0'};
char ars3[10] = { 'H', 'a', 'l', 'l', 'o', '\0'};
char ars4[] = "Hallo";
// verboden is om daarna te doen : ars4 = "Goodbye";
char ars5[10] = "Hallo";
char *s = "Hallo";
s = "Goodbye";
```

Toeters en Bellen:
`*"ABC"` is toegestaan : de string wordt opgevat als **het adres** van het eerste karakter van de string; en van dat adres wordt **de inhoud** (vanwege dat *) bedoeld. Dat is dus de letter `'A'`
`*("ABC" + 2)` is dan dus de letter `'C'` en
`*("ABC" + 3)` is dan dus het nulkarakter `'\0'`

`"ABC"[0]` is gewoon de eerste letter, dus letter `'A'`

`char *p; p = "ABC";`	dit is toegestaan
`char woord[] = "OK";`	dit ook toegestaan
`char woord[3]; woord = "OK";`	dit is niet toegestaan, omdat een array geen ***l-value*** is.

```
char *p;
p = "ABC";
```
`p` heeft lengte 3
`p[0]` is de inhoud van de eerste geheugenplaats: dat is de letter `'A'`
`p++;` nu is `p[0]` de inhoud van de eerste geheugenplaats waar `p` naar wijst: de letter `'B'` En de lengte van `p` is nu gelijk aan 2

`char *t = "oliebol";`	de lengte van `t` is 7
`t[4] = '\0';`	lengte is nu 4; waarde is `"olie"`
	// Opmerking: DevC++ slaat hiervan op tilt

Practicum Week 3 (Default Practicum)

Opdracht 3_1 (zie Vb 7_3, Vb 7_4 en Vb 7_7)
je mag geen array gebruiken

Invoer:
Via het toetsenbord wordt een (onbekend) aantal positieve gehele getallen ingelezen. De invoerrij wordt **afgesloten** door het intikken van een letter. Wanneer de invoerrij een negatief getal bevat, moet dat worden **genegeerd**. En het programma moet gewoon verder gaan met het volgende getal uit de invoerrij.
Uitvoer:
Het grootste ingelezen getal wordt getoond op het beeldscherm, en er wordt ook getoond hoe vaak dit grootste getal in de invoerrij voor kwam.

Voorbeeld: Invoer : 23 4 5 7 17 23 -9 23 5 K <ENTER>
Uitvoer : Grootste getal: 23; komt 3 keer voor

Opdracht 3_2 (zie Vb 8_1 en Vb 8_3)
Schrijf een C-programma waarin een ***char-array*** wordt gevuld met de 26 letters van het alfabet. Druk vervolgens de inhoud van de array af; eerst van voren naar achteren; en dan van achter naar voren.
Maak nu een ***string*** waarin alle 26 letters van het alfabet staan. Druk ook nu alle letters af, eveneens de beide kanten op.

Opdracht 3_3 (zie Vb 8_3)
Schrijf een programma, waarin twee ***strings*** moeten worden ingelezen van toetsenbord. Laat het programma de **eerstse letter** zoeken waarbij deze strings verschillen, en druk de waarde van de array-index af.

Voorbeeld: de invoer is "In zee met C" en "In zee leven kwallen".
Dan is de uitvoer: de eerste letter, waarin de invoerzinnen verschillen, is lettertje 'm' in de eerste zin en lettertje 'l' in de tweede zin met index = 7.