C-programmeren

Boek : In zee met C, Leo van Moergestel, Academic Service,
ISBN 978 90 395 2479 4

Indeling van de cursus

- Per week 2 uur theorie en 2 uur practicum.
- Aanwezigheid bij de **theorie** is niet verplicht.
- **Aanwezigheid** bij het **practicum** is **verplicht** bij de default-optie; en er is een **inleververplichting per week**.
  Het practicum kent ook een zgn. individuele optie. **Iedereen komt automatisch terecht in de default-optie**. De individuele optie moet per e-mail worden aangevraagd bij de docent. Zie uitgedeelde stencil met alle eisen & details over de practicumregeling.
- Aan het eind van elk blok is een **theorie tentamen** over de behandelde stof. Dit theorie-cijfer wordt door Osiris uitsluitend & alleen toegekend (=in studie-punten uitbetaald) als er voor het practicum van dat blok een voldoende is toegekend.

Overzicht van de stof per week in Blok 1: (code : **V2CPRG1**)

weeknummer:	**boek:**	**onderwerpen:**
Week 1 : 3 sept	H 1, 3, 4, 5	inleiding, typen, operatoren, expressies, uitvoer, en het werken onder linux
Week 2 : 10 sept	H 2, 6, X	talstelsels, keuzen, inleiding pointers
Week 3 : 17 sept	H 7, 8, X	herhalingslussen, array's, invoer met scanf
Week 4 : 24 sept	H 9, 10	samengestelde datatypen, pointers
Week 5 : 1 okt	H 11, X	functies, mèèr van & met functies
Week 6 : 8 okt	H 12	bit-operaties
Week 7 : 15 okt	-	Uitloop, vragen en proeftentamen

X = extra stukje stof (resp over pointers, scanf en functies)

Opmerking: in V2CPRG2 zullen worden behandeld de hoofdstukken 13 t/m 19

Werken op de computer: (runnen van C-programma's)

Op school zullen we in het C-practicum werken onder Ubuntu Linux.
!! Bij het C-boek hoort een live CD. Lees eerst de waarschuwing op blz 220

Je kunt ook werken met DevC++
Opmerking: in dat geval moet je `system("pause");` opnemen in je code

Verschillen tussen C en Java:
In Java bestaat elk programma uit een aantal **klassen**.

In C bestaat elk programma uit een één ***main-source-file*** en combinaties van ***header-files*** en hun bijbehorende ***source-file***. Er is altijd één ***main-source-file***, namelijk de source-file met de functie `main()` erin. Alleen de sourcefile met de functie `main()` kan worden gerund (net als in Java). Die ***main-source-file*** kan worden opgeslagen onder de naam "main**.c**" maar je kunt ook een naam kiezen als "Opdracht1_1**.c**". De uitvoering van een C-programma begint bij de eerste instructie in die functie `main()` (en dat komt dus overeen met de gang van zaken in Java).

Verder is C een ***procedurele*** taal, d.w.z. dat C-programma's zijn opgebouwd uit een aantal ***functies***. I.p.v. functie zegt men ook wel ***procedure***. (in Java heten die dingen ***methoden***)

Een C-***header-file*** bestaat uit de ***functie-prototypes*** van een aantal ***functies*** (inclusief enige toeters & bellen). In de bijbehorende ***source-file*** staat van iedere functie de volledige ***functie-definitie*** Een ***header-file*** wordt opgeslagen als naamFile**.h** en de bijbehorende ***source-file*** als naamFile**.c**

In C zijn de bibliotheken georganiseerd in de vorm van zgn. ***header-files***. Dat geef je aan met

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

(vergelijkbaar met `import java.io.*;` voor de package `java.io` in Java).

Zeer eenvoudig C-programma:

```
#include <stdio.h>          // header-file standard.io voor printf()
//#include <stdlib.h>       // vanwege system("pause"); in DevC++

int main(void)
{
  printf("hello world");
  //system("PAUSE");        // nodig bij runnen in DevC++
  return 0;
}
```

Uitvoer:
hello world

Werkwijze:
Tik de programmatekst in, in de editor. En sla op, bijvoorbeeld onder de naam "Voorbeeld0.**c**".

Compileren
Compileren in C verloopt enigszins anders dan compileren in Java, omdat C beschikt over een ***preprocessor***. Deze preprocessor is een ***macro***processor. Vlak voor de compilatie wordt de brontekst van een C-programma door de ***preprocessor*** verwerkt. De preprocessor wordt ***bestuurd*** door commandoregels in de tekst (dat zijn regels tekst in de broncode die met **`#include`** beginnen)
De preprocessor verwijdert al deze instructies uit de bronfile en brengt aan de hand van die instructies de bijbehorende instructies aan in de tekst. De **resulterende** tekst wordt vervolgens door de compiler verwerkt.
De preprocessor verwerkt één regel tegelijk. De syntax van deze preprocessor-regels is onafhankelijk van de rest van de taal C. De preprocessor werkt ook onafhankelijk van de bereikregels van C. Preprocessor-regels zoals `#define` zijn van kracht tot het einde van de programma-eenheid. Doordat de preprocessor regelsgewijs werkt, zijn de grenzen tussen de regels voor de preprocessor van belang. Hiervoor worden zgn. ***white-space-characters*** (of *witruimtekarakters*) gebruikt. Zet dus niet per ongeluk een punt-komma aan het eind van een regel, die begint met `#include`. Verder wordt elk commentaar in het programma door de preprocessor vervangen door één spatie.

Linken:
Bij C bestaat een programma uit een aantal files: de ***header-files*** en de ***source-files***, waarbij doorgaans alle stukken apart worden gecompileerd. En ook wordt er gebruik gemaakt van kant-en-klare stukken programma. Vervolgens moeten al die stukken worden samengevoegd tot tot één ***executable***. Dat samenvoegen wordt gedaan door een een speciaal ***linking program*** of ***linker***. Deze linker produceert een ***executable file*** een (met extensie **.exe**). Bij een C-programma maak je daarom gebruik van de optie "**Build**" en niet zozeer van "Compile". En wanneer dat succesvol is verlopen, kies je vervolgens voor de optie "**Execute**"

Wanneer je werkt binnen een ontwikkelomgeving zoals bijvoorbeeld Anjuta of binnen DevC++ is er doorgaans een knop "Compile" om te compileren. Je kunt je programma ook rechtstreeks laten compileren. Ons programma "Voorbeeld0.**c**" wordt ***gecompileerd*** door:

cc –o Voorbeeld0 **Voorbeeld0.c**

en als het compileren sucesvol is verlopen kun je laten ***runnen*** door:

Voorbeeld0

Week 1 : Hoofdstukken 1, 3, 4 en 5

Opmerking:
In hoofdstuk 2 staat veel theorie, en ik wil zo snel mogelijk laten programmeren. Daarom doen we hfst 2 later. Omdat TI-studenten geen beginnende programmeurs zijn, zijn onze voorbeelden moeilijker dan die uit het boek.

Hoofdstuk 3: Heel kleine C-programma's

Voorbeeld 3_1
Dit is een heel eenvoudig programma, dat is bedoeld om te wennen aan de manier van programmeren in C. Er worden twee regels tekst op het beeldscherm afgedrukt (in een DOS-box)

```
#include <stdio.h>          // header-file standard.io voor printf()

int main(void)
{
  printf("Hiermee kun je tekst afdrukken\n");
  printf("En de overgang op een nieuwe regel gaat ook met
                                                  \"\n\"\n");
  return 0;
}
```

Uitvoer:
Hiermee kun je tekst afdrukken
En de overgang op een nieuwe regel gaat ook met "\n"

Opmerking:
Met character `'\n'` (type `char`) ga je over naar de volgende regel (net zoals dat in Java gebeurt).
Het character `'\"'` (type `char`) levert één dubel aanhalingsteken (ook dat is hetzelfde in Java). Die backslash heet ook hier: ***escape sequence***.
Het character `'\\'` (type `char`) levert één backslash. In bovenstaande `printf`-opdracht resulteert `\"\n\"` in het afdrukken van : "\n" (en daarvoor is dus **driemaal een escape-sequence** nodig)

Opdracht:
Schrijf een programma dat op het beeldscherm afdrukt:
Alle alligators hadden kiespijn
In die regel staan 4 a's

Voorbeeld 3_2
Nu moet met `printf()` een int-getal worden afgedrukt op het scherm.

```
#include <stdio.h>

int main(void)
{
  int som;
  som = 10 + 20;
  printf("De som van 10 en 20 is %d\n", som);
  return 0;
}
```

Uitvoer:
De som van 10 en 20 is 30

Opmerkingen:
Als een C-programma wordt uitgevoerd, worden door het besturings-systeem automatisch drie files geopend voor gebruik door dat programma. Die files zijn standaard-invoer, -uitvoer en –fout. Wanneer een programma gegevens moet afdrukken op het standaard output-medium (het beeldscherm), moet de regel

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

worden opgenomen in het programma. Daarmee is het mogelijk om de functie `printf()` te gebruiken. Deze functie heeft een aantal parameters, waarvan de eerste parameter (***formaat***) een string moet zijn. In het bovenstaande voorbeeld is die ***formaat***-string gelijk aan

```
"De som van 10 en 20 is %d\n"
```

Alle volgende parameters zijn ***expressies*** (*argumenten*), waarvan de resultaat-waarde moet worden afgedrukt. In het bovenstaande voorbeeld is die ***expressie*** **`som`**.
In de *formaat*-string is in dit vb ook de ***conversie-specificatie*** **`%d`** opgenomen. Voor elk gebruikt *argument* moet in de *formaat*-string een passende *conversie-specificatie* zijn opgenomen. Omdat variabele **`som`** van type int is, is de bijbehorende *conversie-specificatie* **`%d`** .

In het volgende voorbeeld staan meerdere *conversie-specificaties*

Voorbeeld 3_3 (voorbeeld met verschillende ***conversie-specificaties***)

```
/* Je moet commentaar op deze manier in je C-programma's opnemen.
** Dit is Voorbeeld 3_3 met variabelen van verschillende typen
*/
#include <stdio.h>

int main(void)
{
  int i = 1234;                          // 1234 is een int-constante
  printf("Dit is een int-waarde : i = %d\n", i);
  printf("En nu met een maat erbij : i = %6d\n", i);

  double d = 3.141592654;                // 3.141592654 is een double-const.
  printf("Dit is een double-waarde : d = %g\n", d);
  printf("En met een maat erbij : d = %8.2g\n", d);

  float y = 3.141592654f;                // 3.141592654f is een float-const
  printf("Dit is een float-waarde : y = %f\n", y);
  printf("En met een maat erbij : y = %8.4f\n", y);

  char ch = 'X';                         // 'X' is een char-constante (enkele quotes)
  printf("Dit is een char-waarde : ch = %c\n", ch);
  printf("En met een maat erbij : ch = %3c\n", ch);
  printf("In C is een char een int: int = %d\n", ch);
  printf("En ch als hexadecimaal : hexa = %x\n", ch);
  printf("En ch als octaal : oct = %o\n", ch);

  char tekst[ ] = "C is best wel lastig";
  // Alles met " en " heet string. Strings in C zijn knap lastig.
  printf("Een string-var. : tekst = %s\n", tekst);
  return 0;
}
```

Uitvoer:

```
Dit is een int-waarde : i = 1234
En nu met een maat erbij : i =     1234
Dit is een double-waarde : d = 3.14159
En met een maat erbij : d =       3.14
Dit is een float-waarde : y = 3.141593
En met een maat erbij : y =     3.1416
Dit is een char-waarde : ch = X
En met een maat erbij : ch =      X
In C is een char een int : int = 88
En ch als hexadecimaal : hexa = 58
En ch als octaal : oct = 130
Een string-var. : tekst = C is best wel lastig
```

Hoofdstuk 4: Variabelen en constanten.

Voorbeeld 4_1
Dit is een voorbeeld met constanten van type **int** en type **long**.

```
#include <stdio.h>
#include <limits.h>          // vanwege INT_MAX en LONG_MAX

int main(void)
{
  const int aantal = 1234;
  const int maximum = 2147483647;
  const int octaleConst = 0777;
  const int hexaConst = 0X1234AF;

  printf("Aantal bytes van een int is : %d\n", sizeof(int));
  printf("En van long : %d\n", sizeof(long));
  // Let op: sizeof is een operator, en heeft dus prioriteit.

  printf("aantal = %d\n", aantal);
  printf("maximum = %d\n", maximum);
  printf("octaleConst = %o (octaal)\n", octaleConst);
  printf("octaleConst = %d (dec.)\n", octaleConst);

  printf("hexaConst = %x (hexadec.)\n",  hexaConst);
  printf("hexaConst = %d (decimaal)\n",  hexaConst);

  printf("Opvragen van int-maximum : %d\n", INT_MAX);
  printf("Opvragen long-maximum : %ld\n", LONG_MAX); // NIET: L
  return 0;
}
```

Uitvoer:
Aantal bytes van een int is : 4
En van long : 4
aantal = 1234
maximum = 2147483647
octaleConst = 777 (octaal)
octaleConst = 511 (dec.)
hexaConst = 1234af (hexadec.)
hexaConst = 1193135 (decimaal)
Opvragen van int-maximum : 2147483647
Opvragen long-maximum : 2147483647

Opmerking: INT_MAX en LONG_MAX zijn implementatie-afhankelijk

Opmerking:
Voor constanten met type `long`, `float` of `double` staan hieronder enige voorbeelden:

```
const long getal = 2147483647L;          // of : l
const double pi = 3.141592654;
const float piFloat = 3.141592654f;      // of : F

printf("getal = %d\n", aantal);
printf("getal = %+12ld\n", aantal);
printf("pi = %e\n", pi);
printf("pi = %E\n", pi);
printf("pi = %+10.6E\n", pi);
printf("pi = %+10.6g\n", pi);
printf("piFloat = %+10.6f\n", piFloat);
```

Er wordt afgedrukt:

```
getal = 1234
getal =       +1234
pi = 3.141593e+000
pi = 3.141593E+000
pi = +3.141593E+000
pi =    +3.14159
piFloat =   +3.141593
```

`int`-**typen:**

`int`	32 bits = 4 bytes
`short`	16 bits = 2 bytes
`long`	32 bits = 4 bytes
`long long`	64 bits = 8 bytes
`char !!`	8 bits = 1 byte

`int`-**constanten:**

```
const int aantal = 1234;
const int minimum = -2147483647;
const int octaleConst = 0777;          // beginnen met een nul
const int hexaConst = 0X1234AF;        // beginnen met 0x of 0X
const long getal = 2147483647L;        // eindigen op L of : l
```

Voorbeeld 4_2

De characters (type `char`) en de karakter-strings (type `char[ ]`) zijn in C anders van aard dan in Java.

```
#include <stdio.h>

int main(void)
{
  char c = 'X';            // 'X' is een char-constante (enkele quotes)
  printf("De char-waarde : c = %c\n", c);
  printf("char is int : c (als int) = %d\n", c);
  printf("char als hexa : c (als hexa) = %x\n", c);
  printf("char als oct : c (als oct) = %o\n\n", c);

  char hex = '\x58';       // zie ASCII-set: naast letter X staat hexa 58
  if ( hex == c )
    printf("Dit gaat anders dan in Java.\n");

  char oct = '\130';       // en in ASCII-set: naast X staat oct-getal 130
  if ( oct == c )
    printf("En hij is ook gelijk aan een oct.\n");

  char tekst[ ] = "C is best wel lastig";  // met " " is een string
  printf("Een string-var. : tekst = %25s\n", tekst);
  printf("Met uitlijnen : tekst = %-25s\n", tekst);
  return 0;
}
```

Uitvoer:

De char-waarde : c = X
char is int : c (als int) = 88
char als hexa : c (als hexa) = 58
char als oct : c (als oct) = 130

Dit gaat anders dan in Java.
En hij is ook gelijk aan een oct.
Een string-var. : tekst = C is best wel lastig
Met uitlijnen : tekst = C is best wel lastig

Opmerking:
De `char`-constante `'\0'` heeft in C een speciale rol (`'\0'`= 0000 0000)

Voorbeeld 4_3

Dit is het programma van blz 26 uit het boek. Het programma demonstreert wat er gebeurt wanneer een waarde wordt toegekend, die buiten de range valt.

```
#include <stdio.h>

int main(void)
{
  int i;
  unsigned short counter;
  short temperatuur;

  i = 777;
  counter = 5;
  temperatuur = -5;
  printf("i = %d, counter = %d en temperatuur = %d\n",
                                    i, counter, temperatuur);
  i = 0777;
  counter = -5;
  temperatuur = 65000;
  printf("i = %d, counter = %d en temperatuur = %d\n",
                                    i, counter, temperatuur);
  i = 0x777;
  counter = 'a';
  temperatuur = 0xFFFF;
  printf("i = %d, counter = %d en temperatuur = %d\n",
                                    i, counter, temperatuur);
  return 0;
}
```

Uitvoer:

i = 777, counter = 5 en temperatuur = -5
i = 511, counter = 65531 en temperatuur = -536
i = 1911, counter = 97 en temperatuur = -1

Uitleg:
(I)
Variabele `i` heeft type `int` .
Dan is **`0777`** een ***octale*** waarde, nl. de waarde:

$7 * 8^2 + 7 * 8^1 + 7 * 1 = 7 * 64 + 7 * 8 + 7 = 511$

En **`0x777`** een ***hexadecimale*** waarde, nl. de waarde:

$7 * 16^2 + 7 * 16^1 + 7 * 1 = 7 * 256 + 7 * 16 + 7 = 1911$

(II)
Variabele `counter` heeft type **`unsigned`** `short` . Dat zijn de `int`-getallen van 0 tot en met 65535 (zie blz 25). Dit zijn 65536 getallen
Dan is **`-5`** een waarde buiten deze range. Je moet er dus 65536 bij optellen:

$-5 + 65536 = 65531$

En **`'a'`** is hetzelfde als de `int`-waarde 97 want de ASCII-waarde van letter a is 97

(III)
Variabele `temperatuur` heeft type `short` . Dat zijn de `int`-getallen van -32768 tot en met 32767 (zie blz 25). Dit zijn 65536 getallen
Dan is **`65000`** een waarde buiten deze range. Je moet er dus 65536 van aftrekken:

$65000 - 65536 = -536$

En **`0xFFFF`** is

$15 * 16^3 + 15 * 16^2 + 15 * 16^1 + 15 * 1 = 65535$

Dit getal ligt buiten de range. Met aftrek van 65536 ligt de waarde wel binnen de range:

$65535 - 65536 = -1$

Voorbeeld 4_4 !! Globale variabelen zijn verboden !!

Dit is een variant op het programma van blz 28 uit het boek. Het programma demonstreert de werking van ***globale*** en ***lokale*** variabelen. In onderstaand programma staat één ***globale variabele***. Functies komen in Hfst 11 aan de orde.

```
#include <stdio.h>

int n = 100;          // globale variabele n; met initialisatie.
                      // opmerking: keyword extern is verboden bij definitie

void f(int i, int j)
{
  n += i + j;
  printf("In f: n= %d i = %d j = %d\n", n, i, j);
  // hier eindigt de scope van variabelen i en j
}

void g(int k)
{
  n *= k;
  printf("In g: n= %d k = %d\n", n, k);
  // hier eindigt de scope van variabele k
}

int main(void)
{
  printf("In main: n= %d\n", n);
  n = 20;
  f(2, 3);
  printf("In main: n= %d\n", n);
  g(7);
  printf("In main: n= %d\n", n);
  return 0;
}
// hier eindigt de scope van variabele n
```

Uitvoer:

```
In main: n = 100
In f: n = 25 i = 2 j = 3
In main: n = 25
In g: n = 175 k = 7
In main: n = 175
```

Functies in C zijn extern. In C bestaan ook ***externe variabelen***. Een ***externe variabele*** of ***globale variabele*** is een variabele, die is gedeclareerd **buiten** een functie. De zogenaamde ***scope*** van een ***globale variabele*** beslaat dus een **file**.
Opmerking: variabele `n` is een ***globale variabele***

Een ***lokale variabele*** is een variabele die is gedeclareed **binnen** een functie (dat is de gebruikelijke gang van zaken). De ***scope*** van een ***lokale variabele*** beslaat dus een **blok**.

Externe variabelen (of ***globale variabelen***) worden tijdens het compileren geïnitialiseerd. Deze **initialisatie vindt één keer plaats**.
Wanneer er **geen** expliciete initialisatie plaats vindt, krijgen externe variabelen 0 als startwaarde (= initialisatie).
Opmerking: variabele `n` krijgt de waarde 100 mee als initialisatie.

Je kunt variabelen dus **indelen naar de plaats** waar de variabele wordt **gedeclareerd**: binnen een functie (= ***lokaal***) of buiten een functie (= ***globaal***).

Opmerking:
Declaratie van variabele `n`: `extern int n;`
Definitie van variabele `n`: **`n = 100;`**
Declaratie en ***initialisatie*** van `n`: `extern int` **`n = 100;`**

Opmerking:
Bij ***definitie*** van een ***globele variabele*** wordt de geheugenruimte gereserveerd; De geheugenruimte van een ***globale variabele*** is **permanent**.

Bij ***declaratie*** van een ***globale variabele*** wordt **geen** geheugenruimte gereserveerd. Deze zin krijgt pas betekenis op het moment dat we onze C-programma's gaan opdelen. Op de volgende bladzijde staat een opdeling van het programma van Voorbeeld 4_4

Dit is Voorbeeld 4_4, maar dan in twee stukken gesplitst
In onderstaande **twee files** staat (in totaal) één ***globale variabele***.

```
// file "hulp.c" ; kan apart worden gecompileerd  van "main.c".
// Dat heet separate compilatie
#include <stdio.h>

int n = 100;          // globale variabele n; met initialisatie.
                      // opmerking: keyword extern is verboden bij definitie

void f(int i, int j)
{
  n += i + j;
  printf("In f: n= %d i = %d j = %d\n", n, i, j);
}

void g(int k)
{
  n *= k;
  printf("In g: n= %d k = %d\n", n, k);
}
```

```
// file "main.c" ; kan apart worden gecompileerd  van "hulp.c".
#include <stdio.h>

extern int n;                 // declaratie van globale variabele n

void f(int, int);
void g(int);

int main(void)
{
  printf("In main: n= %d\n", n);
  n = 20;
  f(2, 3);
  printf("In main: n= %d\n", n);
  g(7);
  printf("In main: n= %d\n", n);
  return 0;
}
```

Uitvoer: hetzelfde als in Voorbeeld 4_4

Hoofdstuk 5: Bewerkingen.

Voorbeeld 5_1
Dit is een voorbeeld met ***type-conversie*** in C. In Java is ook sprake van type-conversie, maar in C speelt type-conversie een belangrijker rol. Hieronder staat een eenvoudig voorbeeld.

```
#include <stdio.h>

int main(void)
{
  char c = 'X';            // 'X' is een char-constante (enkele quotes)
  printf("Waarde van c – \'A\' is: %d\n", c – 'A' );

  int n = 10;
  int som = 56;
  double gem1 = som / n;
  printf("Het gemiddelde = %g; Gezakt!!\n", gem1);
  double gem2 = 1.0 * som / n;
  printf("Nu is dat zelfde gemiddelde : %g\n", gem2);
  return 0;
}
```

Uitvoer:
Waarde van c – 'A' is: 23
Het gemiddelde = 5; Gezakt!!
Nu is dat zelfde gemiddelde : 5.6

Opmerkingen:
- De deling `56 / 10` is een deling van twee int-getallen. Dan heeft het geen effect meer, als je de uitkomst ***converteert*** naar een double. Je moet dus voor de berekening plaats vindt al geconverteerd hebben.
- Je kunt type convesie (in dit vb naar een double) afdwingen met een **cast**, of door te vermenigvuldigen met **1.0** (voor een double) resp. **1.0f** (voor een float).
- Dit is een hele kleine demonstratie van ***type-conversie***.

Voorbeeld 5_2
Dit is een voorbeeld met de **komma-operator**. Ik ben zelf geen voorstander van de komma-operator, maar je moet wel weten hoe hij werkt.

```
#include <stdio.h>

int main(void)
{
  int s = 13, t = -7;
  s = (t = 2, t + 3 );                  // s = 5, t = 2
  printf("s = %d en t = %d\n", s, t);

  int k = 8, som = 45;
  s = 13;
  s = (som = 0, k = 1);                 // s = 1, som = 0, k = 1
  //s = som = 0, k = 1;                 // s = 0, som = 0, k = 1
  printf("s = %d, k = %d, som = %d\n", s, k, som);
  return 0 ;
}
```

Uitvoer:
s = 5 en t = 2
s = 1, k = 1, som = 0
// s = 0, k = 1, som = 0

Uitleg:
Met de opdracht:

```
  s = (t = 2, t + 3 );
```

wordt eerst aan variabele `t` de waarde 2 toegekend. Vervolgens wordt uitgevoerd:

```
        s = ( het resultaat is 2 en dit wordt weg gegooid , 5 );
```

Aan variabele `s` wordt dus de waarde 5 toegekend.

Voorbeeld 5_3
Dit is een compilatie van de voorbeelden uit het boek. Deze programma opdrachten demonstreren de prioriteitsregels van de operatoren.

Demo 1:

```
int a, b, c;
a = 24;              // a = 24
b = a / 6 * 4;       // b = 16
c = a % 3 * 4;       // c =  0

b = a = c;           // a = 0  en b = 0  en c = 0
```

Demo 2:

```
int i, j, k, m, n;
i = 20, j = 10;
(k = m = i += j += n = 3)++;
// n = 3  en j = 13  en i = 33  en m = 33  en k = 34
```

Prioriteitsregels van alle operatoren (van hoogste prio tot laagste prio):

```
()   []   ->   .
!    ~    ++   --   - (type-cast) *    &    sizeof   // allen unair
*    /    %
+    -
<<   >>
<    <=     >    >=
==   !=
&
^
|
&&
||
?:
=    +=     -=     *=     /=     %=   &=   |=     ^=     <<=    >>=
,                                  // de komma-operator; assoc. van L naar R
```

Niet alleen de prio-rangorde is van belang, maar ook de richting bij associatie
VB: de operatoren `*  /  %` associeren van links naar rechts (Demo 1)
VB: de operator `=` associeert van rechts naar links (Demo 1)
VB: de operatoren `=` en `+=` associeren van rechts naar links (Demo 2)

Practicum Week 1 (zowel Default Practicum als Individuele Practicum)

Opmerking 1: we kunnen nog niet inlezen van toetsenbord.
Opmerking 2: dit zijn opgaven om vertrouwd teraken methet eigen karakter van C. Ik test deze kennis niet op het tentamen. Ik test kunnen-programmeren in C.

Opdracht 1_1 (zie Vb 3_3 en Vb 4_1)

Kies:

```
char c = 'A';
```

Druk de waarde van c af als: character, decimale int, hexadecimale int en octale int

Tel nu bij `c` een int-constante op: bijv `c += 20;`
En druk weer 4 keer de waarde af van `c`

Opdracht 1_2 (zie Vb 5_1)

Shrijf een C-programma waarmee de tafel van 7 wordt afgedrukt op het beeldscherm **in nette kolommen** onder elkaar. Gebruik de for-lus, dwz

```
int i;
for ( i = startWaarde ; i < stopWaarde ; i++ ) {
  doeIets;
}
```

Schrijf code, die ook werkt voor alle andere tafels van 1 t/m 99

Opdracht 1_3 (zie Vb 3_2 , Vb 3_3, Vb 4 _3 en Vb 5_1)

Schrijf een programma om de effecten van unsigned versus signed te begrijpen.
Declareer met initialisatie:

Vraag eerst op de SHRT_MAX-waarde (headerfile `<limits.h>`)
(ik ga er van uit dat SHRT_MAX = 32767)

```
short sh = 32768;
signed short s_sh = 32768;
unsigned short u_sh = 32768;
// laat die 3 waardes afdrukken op beeldscherm, en verklaar het resultaat
int i, j, k;
i = s_sh * u_sh;
j = s_sh - 1;
k = u_sh * u_sh;
// laat die 3 waardes afdrukken op beeldscherm, en verklaar het resultaat
```

Oefenopdracht:

(kopieren naar C en runnen levert de antwoorden)

```
#include <stdio.h>

int main(void)
{
  printf("%d\n", - 123);
  printf("%g\n", .123);
  printf("%e\n", 10E-4);
  printf("%c\n", '4');
  printf("%s\n", "vier");
  printf("%c\n", '\z');    // niet OK
  printf("%d\n", 106);

  int i, j, m, n;
  float f, g;
  char c;
  i = j = 2;
  f = 1.2f;
  g = 3.4f;
  c = 'X';

  printf("%d\n", 12*m);
  printf("%d\n", m / j);
  printf("%d\n", n % j);
  printf("%d\n", m / j * j);
  printf("%f\n", (f + 10) * 20);
  printf("%d\n", (i++) * n);
  printf("i = %d\n", i);
  printf("%d\n", i++ * n);
  printf("i = %d\n", i);
  printf("%d\n", -12L *(g - f));
  printf("%d\n", 2 + c);
  printf("%d\n", (n++), (n++));
  printf("n = %d\n", n);
  printf("%d\n", m = n = --j);
  printf("m = %d\n", m);
  printf("n = %d\n", n);
  printf("%d\n", (int) f);
  printf("%d\n", (double) m);
  printf("%d\n", (double) m + 10);
  printf("%d\n", (double) (m + 10));
  return 0;
}
```