Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Gestructureerd programmeren in C

GESPRG: inleiding

DE HAAGSE HOGESCHOOL

Werkvormen GESPRG

- GESPRG th1 + GESPRG pr1 = 3 ECTS = **84** SBU.
  - 21 lesuren theorie
    - 14 lesuren college
    - 7 lesuren werkcollege
  - 14 lesuren practicum.
  - 2 lesuren schriftelijke toets.
  - 49 uur zelfstudie = **7 uur/week zelfstudie + voorbereiden practicum!**
- Toetsing:
  - Schriftelijke toets **GESPRG th1** in week 8 en 10 van dit blok.
  - **GESPRG pr1** practicumopgaven worden afgetekend op het practicum. Alle opgaven moeten voldoende zijn.

Werkcolleges

- Verder/dieper in op de les
- Behandelen van vragen/opdrachten
- Behandelen van huiswerk
- Extra opdrachten
- Peer review
- Code conventions

Leermiddelen

- Boek: Jesse op den Brouw, *De programmeertaal C*.
  - Dit boek is te vinden via https://ds.opdenbrouw.nl/gesprg/book_c.pdf
  - Broncode is te vinden via https://github.com/jesseopdenbrouw/book_c

- Blackboard:
  - Practicumhandleiding
  - Studiewijzer
  - Etc.

- Gebruik van een ontwikkelomgeving
  - CodeBlocks 20.03 of Visual Studio 2019 Community Edition
  - Mac OS-X: Xcode

Wat weet je al?

- Welke programmeertalen ken jij al?
- Welke vormen van programmeren zijn er?
- Waarom met de programmeertaal C?
- Wat is gestructureerd programmeren?
- Waarom gestructureerd leren programmeren bij Elektrotechniek?

Wat is programmeren?

- Een programma vertelt een computer wat die moet doen.

- Welke basisbewerkingen zijn er?
  - Lezen en schrijven (invoer en uitvoer)
  - Onthouden (variabelen)
  - Rekenen
  - Herhalen
  - Beslissen

  - Delegeren (verdeel en heers ➔ functies)
  - Structureren (array en structure)

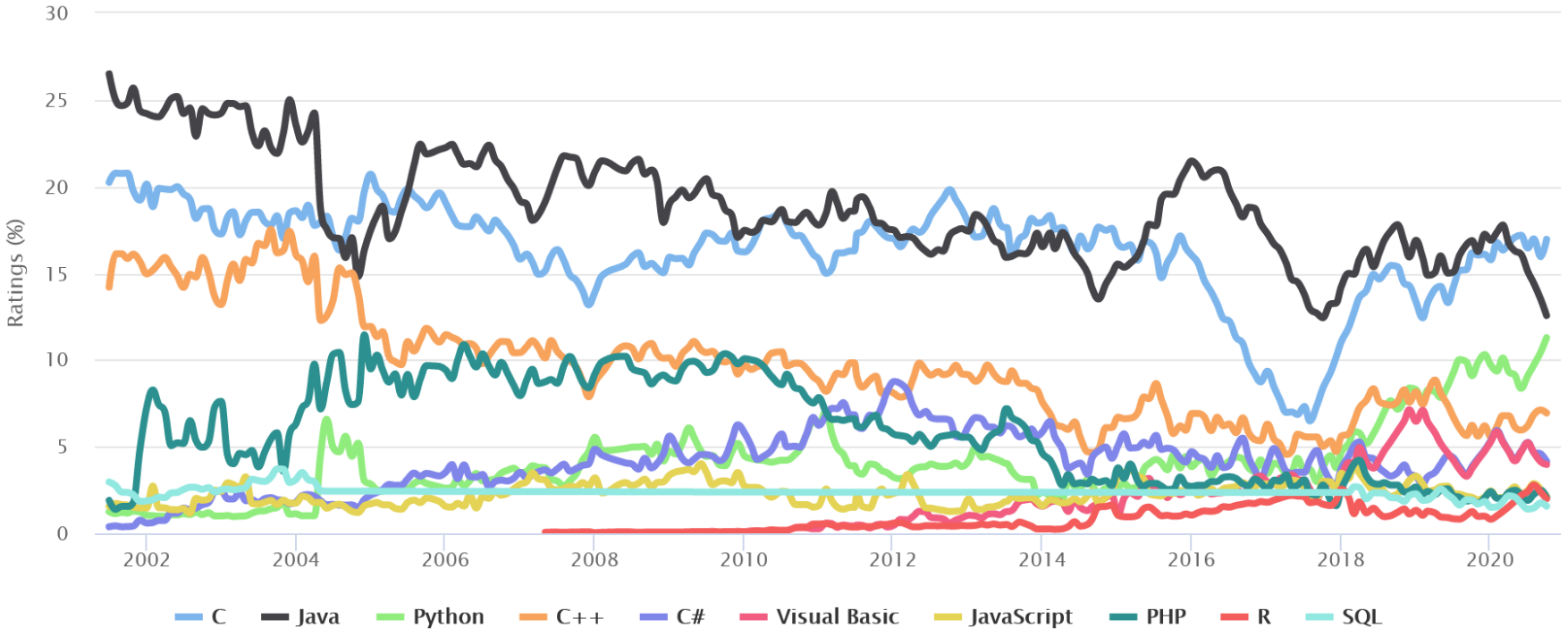
} **Vooral bij grotere programma's**

Gestructureerd programmeren

Wat is **gestructureerd** programmeren?

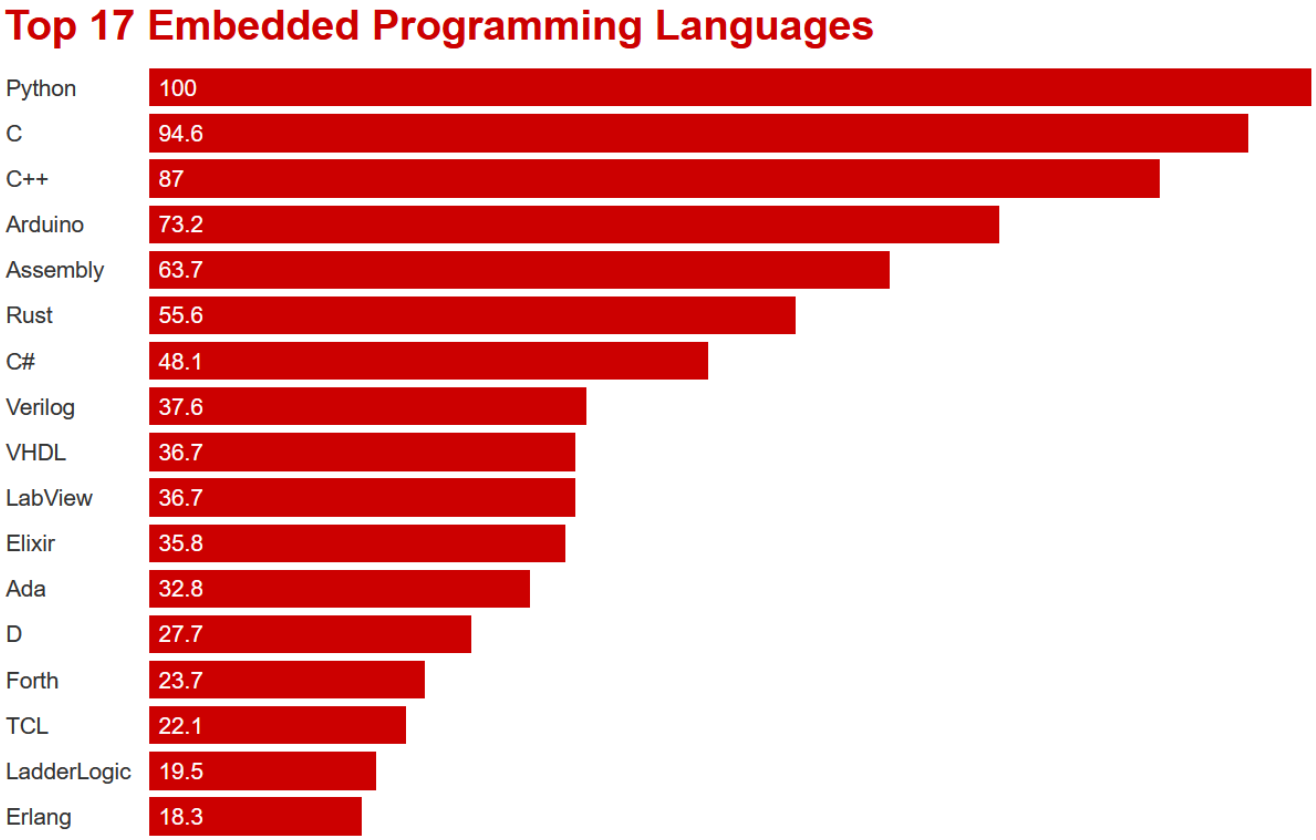
- Een (programmeer) paradigma
- Subdiscipline van procedureel programmeren (imperatief).
- Wat is procedureel programmeren?
  - Een programma wordt opgesteld in de vorm van opdrachten of statements die direct kunnen worden uitgevoerd en de state (toestand) van het programma aanpassen.

Waarom programmeren in C?



Bron: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Waarom programmeren in C?



Bron: https://insights.dice.com/2020/08/21/top-17-programming-languages-embedded-systems-work/

Plaats in het curriculum van GESPRG

- Voorbereiding voor:
  - Projecten in het vervolg van de opleiding
  - INLMIC en MICPRG (Microcontroller programmeren)
  - OGOPRG (Object georiënteerd programmeren in C++)
  - Vak in ECK (RTSYST = Real-time systemen)
  - Minor Embedded Systems

Inhoud GESPRG

- Gestructureerd Programmeren in C
  - Invoer en uitvoer (printf en scanf)
  - Rekenen met gehele (int) en floating point (double) getallen.
  - Herhalingsopdrachten (while, do while en for)
  - Keuzeopdrachten (if, if else en switch case)
  - Functies
  - Pointers
  - Arrays
  - Karakters en strings
  - Structs
  - Tekst files
  - C preprocessor
  - Pointers naar functies

Minimaal C-programma

- Een C-programma moet altijd de functie `main` bevatten.

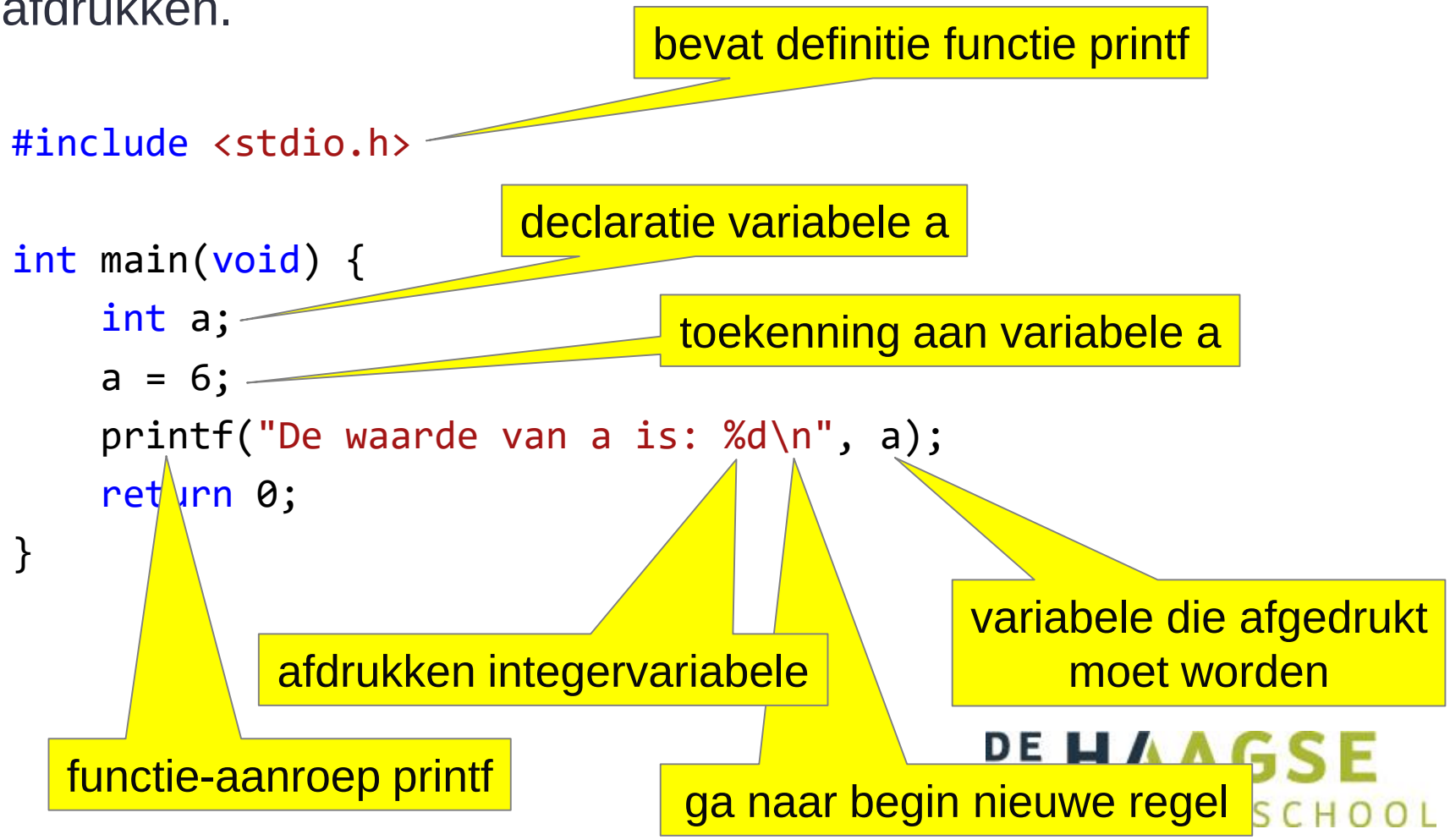
```
int main(void) {

   return 0;
}
```

- De functie `main` geeft een waarde terug aan het operating system (Windows, Mac OS-X, Linux) bij afsluiten.

DE HAAGSE HOGESCHOOL

Afdrukken op het scherm

- Met de functie `printf` kan je tekst en variabelen op het scherm afdrukken.



Eerste C programma

```c
#include <stdio.h>

int main(void) {
    int a, b, product;
    a = 6;
    b = 10;
    product = a * b;
    printf("Het product van %d en %d is: %d\n", a, b, product);
    printf("\nSluit dit venster door op een toets te drukken");
    getchar();
    return 0;
}
```

declaratie variabelen

toekenningen

functie-aanroep met argumenten

functie-aanroep zonder argumenten

Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Gestructureerd programmeren in C

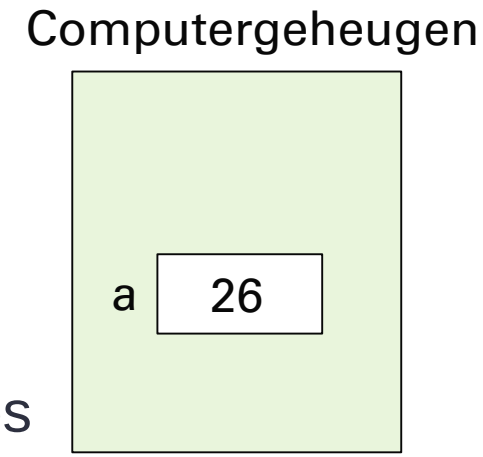
GESPRG: variabelen

DE HAAGSE HOGESCHOOL

Variabelen

- Een variabele is een klein stukje opslagruimte in het werkgeheugen van de computer. Voorbeeld:

```
int a;
```



- In het geheugen is er nu een stukje ruimte vrijgemaakt en deze heeft de naam *a*.

- Met behulp van een toekenning kun je waardes stoppen in deze variabele:

```
a = 26;
```

DE HAAGSE HOGESCHOOL

Basis variabelen

- Een variabele heeft een bepaald type. Dit vertelt hoeveel opslagruimte in het geheugen nodig is en hoe deze opslagruimte gebruikt wordt.
- Voorbeelden van type variabelen (gehele unsigned variabelen):

Variabelen	bits	Decimale waardes
unsigned char	8	$0 \ldots 2^8 - 1 \rightarrow 0 \ldots 255$
unsigned int	32	$0 \ldots 2^{32} - 1 \rightarrow 0 \ldots 4294967295$

DE HAAGSE HOGESCHOOL

Basis variabelen

- Gehele signed variabelen:

Variabelen	bits	Decimale waardes
signed char	8	$-2^7 \dots 2^7 - 1 \rightarrow -128 \dots 127$
signed int	32	$-2^{31} \dots 2^{31} - 1$ $\rightarrow -2147483648 \dots 2147483647$

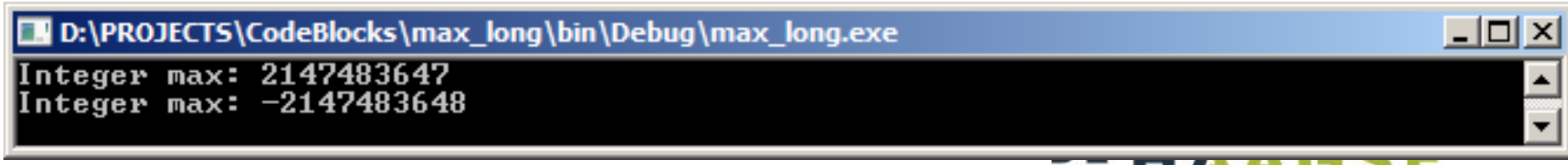
- Let op: een (unsigned/signed) int is bij de meeste grote processoren 32 bits maar bij kleinere (microcontrollers) vaak 16 bits (of zelfs 8 bits).

DE HAAGSE HOGESCHOOL

Basis variabelen

- Signed integers worden in two's complement opgeslagen.
- Er is geen test op overflow.

```
int main() {
    int smax = 2147483647;
    printf("Integer max: %d\n", smax);
    smax = smax + 1;
    printf("Integer max: %d\n", smax);
    return 0;
}
```



Basis variabelen

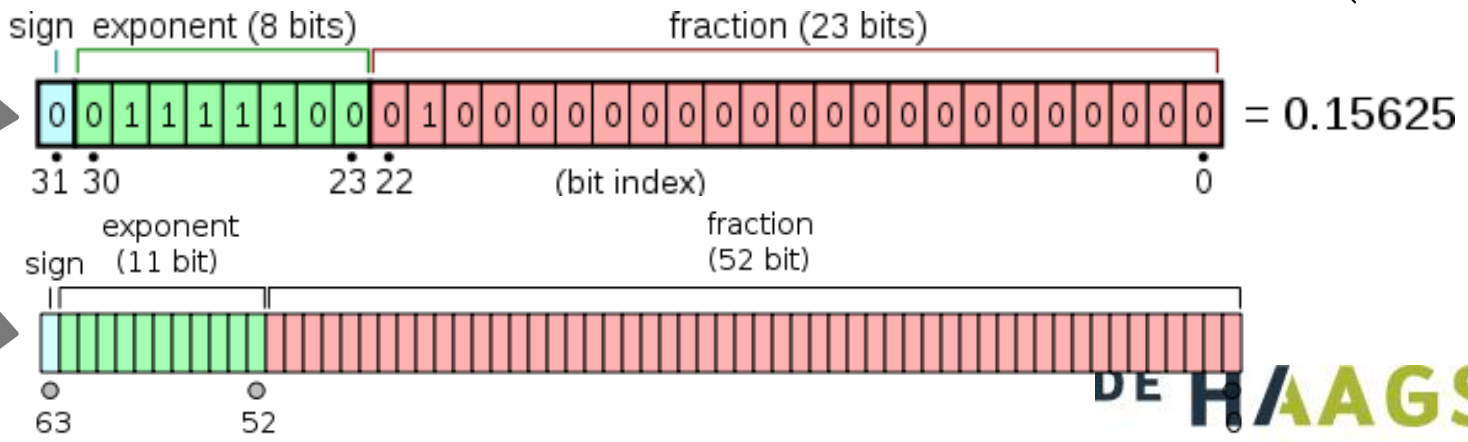
Geen leerstof

- Zwevende kommagetallen (floating point)

Variabelen	bits
float	32
double	64
long double	80/128

afhankelijk van compiler en processor

$$getal = (-1)^{sign} \times 2^{exponent-127} \times \left(1 + \frac{fraction}{2^{23}}\right)$$

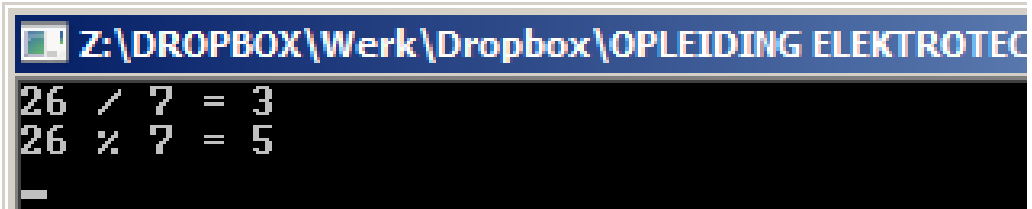


DE HAAGSE HOGESCHOOL

Rekenen met gehele getallen

actie	operator	opmerking
optellen	+	
aftrekken	-	
vermenigvuldigen	*	
delen	/	gehele deling zonder rest
modulo	%	de rest van de deling

```
int a = 26, b = 7;
printf("%d / %d = %d\n", a, b, a / b);
printf("%d %% %d = %d\n", a, b, a % b);
```



Rekenen met floating point

actie	operator
optellen	+
aftrekken	-
vermenigvuldigen	*
delen	/

```
float a = 25.4, b = 2.0;
printf("%f / %f = %f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25.400000 / 2.000000 = 12.700000
```

Uitvoer van variabelen

```
float a = 25.4;
int b = 2;
printf("%f / %d = %f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25.400000 / 2 = 12.700000
```

```
int a = 25.4;
float b = 2;
printf("%d / %f = %f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25 / 2.000000 = 12.500000
```

Uitvoer van variabelen

```
int a = 25.9;
float b = 2;
printf("%d / %f = %f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25 / 2.000000 = 12.500000
```

```
int a = 25.9;
int b = 2;
printf("%d / %d = %f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25 / 2 = 0.000000
```

Waarom nul?

main.c[8] warning: format '%f' expects argument of type 'double', but argument 4 has type 'int'

Uitvoer van variabelen

```
int a = 25.9;
int b = 2;
printf("%d / %d = %d\n", a, b, a / b);
```

Wat is de uitvoer?

```
25 / 2 = 12
```

```
int a = 26.9;
int b = 2.9;
printf("%d / %d = %d\n", a, b, a / b);
```

Wat is de uitvoer?

```
26 / 2 = 13
```

Afdrukken van floating point

```
float a = 25.4, b = 2.0;
printf("%.1f / %.1f = %.1f\n", a, b, a / b);
```

Wat is de uitvoer?

```
25.4 / 2.0 = 12.7
```

```
float a = 25.4, b = 2.0;
printf("%06.2f / %06.2f = %06.2f\n", a, b, a / b);
```

Wat is de uitvoer?

```
025.40 / 002.00 = 012.70
```

double versus float

- Een float wordt opgeslagen in 32 bits en is niet erg nauwkeurig. Minimaal 6 (max 9) significante cijfers.
- Een double wordt opgeslagen in 64 bits en is 2x zo nauwkeurig. Minimaal 15 (max 17) significante cijfers.

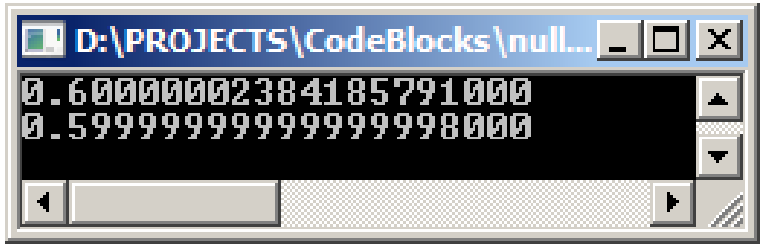
```
#include <stdio.h>

int main(void) {
    float f_derde = 1 / 3.0;
    double d_derde = 1 / 3.0;
    printf("%.20f\n", f_derde);
    printf("%.20f\n", d_derde);
    getchar();
    return 0;
}
```


```
0.33333334326744080000
0.33333333333331000
```

Niet alles kan…

- Niet alle “mooie” decimale breuken kunnen exact worden weergegeven in floating point getallen.

```
#include <stdio.h>

int main(void) {
    float f_nulkommazes = 0.6;
    double d_nulkommazes = 0.6;
    printf("%.20f\n", f_nulkommazes);
    printf("%.20f\n", d_nulkommazes);
    getchar();
    return 0;
}
```



Inlezen variabelen

- Met de functie `scanf` kan een waarde worden ingelezen en aan een variabele worden toegekend.

```
int a;
float b;
double c;

scanf("%d", &a);     /* lees integer van toetsenbord */
scanf("%f", &b);     /* lees float van het toetsenbord */
scanf("%lf", &c);    /* lees double van het toetsenbord */

/* lees integer, float en double van het toetsenbord */
scanf(“%d %f %lf", &a, &b, &c);
```

DE HAAGSE HOGESCHOOL

Voorbeeld: F → C

- Schrijf een programma waarmee een temperatuur in graden Fahrenheit omgerekend kan worden naar graden Celsius.
  - De temperatuur in graden Fahrenheit is een geheel getal.
  - De temperatuur in graden Celcius moet worden afgerond tot een geheel getal.

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

Voorbeelden:

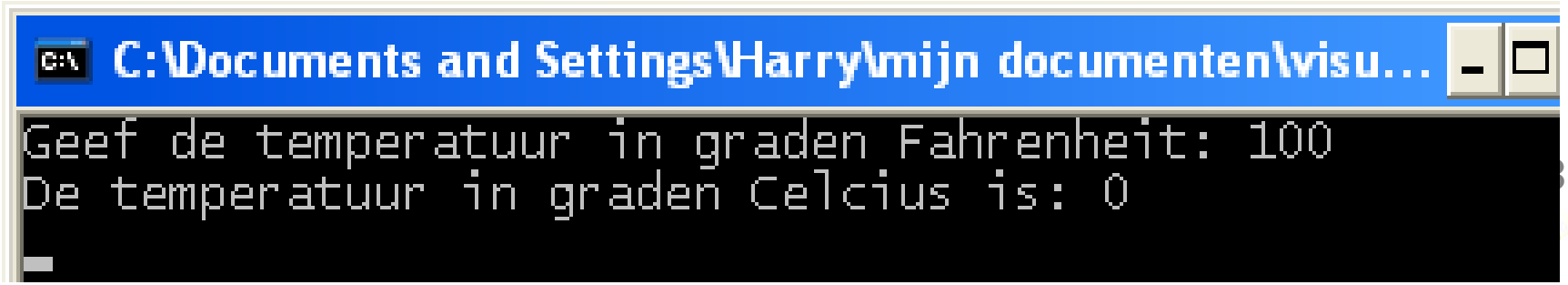
$T_F = 100$ ➔ $T_C = 37.7777\ldots$ ➔ $T_C = 38$

$T_F = 0$ ➔ $T_C = -17.7777\ldots$ ➔ $T_C = -18$

Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int C, F;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    C = (F - 32) * (5 / 9);
    printf("De temperatuur in graden Celcius is: %d\n", C);
    fflush(stdin);
    getchar();
    return 0;
}
```

Wat gaat er verkeerd? Oplossing?

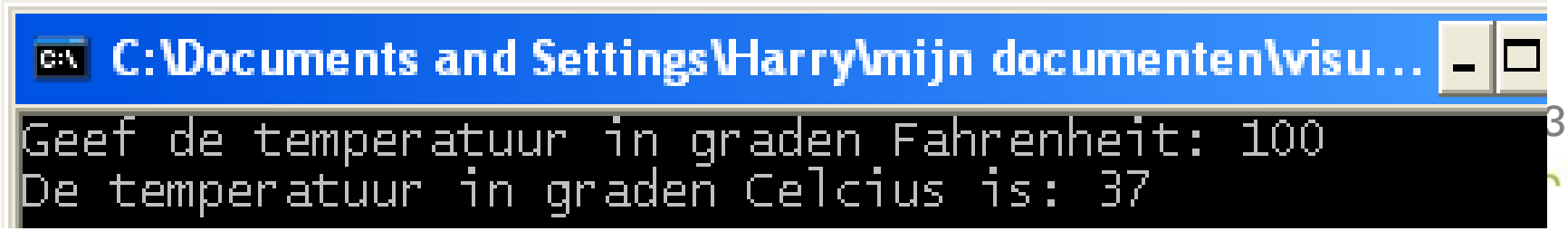


Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int C, F;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    C = (F - 32) * 5 / 9;
    printf("De temperatuur in graden Celcius is: %d\n", C);
    fflush(stdin);
    getchar();
    return 0;
}
```

Er wordt afgekapt in plaats van afgerond!

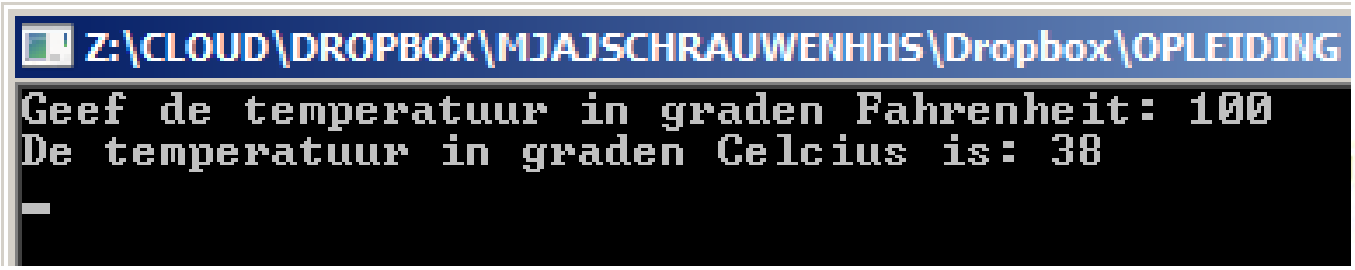
Oplossing?



Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int C, F;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    C = (F - 32) * 5.0 / 9 + 0.5;
    printf("De temperatuur in graden Celcius is: %d\n", C);
    fflush(stdin);
    getchar();
    return 0;
}
```

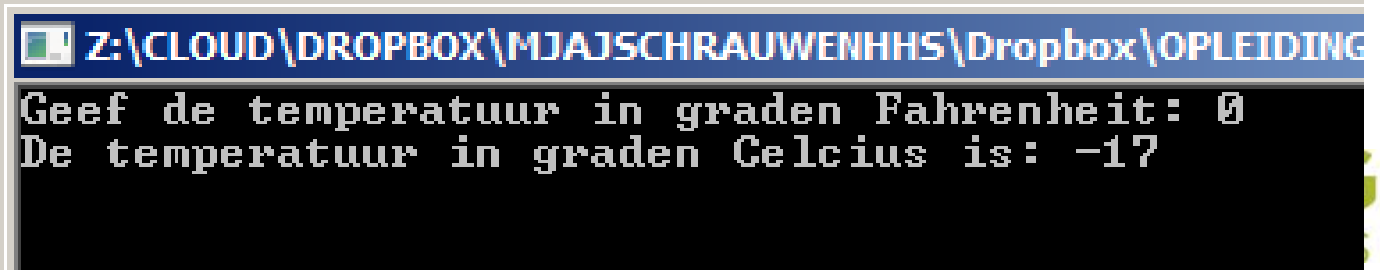
Is het nu goed?



Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int C, F;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    C = (F - 32) * 5.0 / 9 + 0.5;
    printf("De temperatuur in graden Celcius is: %d\n", C);
    fflush(stdin);
    getchar();
    return 0;
}
```

Rondt niet goed af als C negatief wordt.



Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int C, F;
    double Cdouble;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    Cdouble = (F - 32) * 5.0 / 9;
    if (Cdouble > 0) {
        C =  Cdouble + 0.5;
    }
    else {
        C = Cdouble - 0.5;
    }
    printf("De temperatuur in graden Celcius is: %d\n", C);
    fflush(stdin); getchar();
    return 0;
}
```

Kan dat niet eenvoudiger?

Voorbeeld: F → C

```c
#include <stdio.h>

int main(void) {
    int F;
    double C;
    printf("Geef de temperatuur in graden Fahrenheit: ");
    scanf("%d", &F);
    C = (F - 32) * 5.0 / 9;
    printf("De temperatuur in graden Celcius is: %.0f\n", C);
    fflush(stdin);
    getchar();
    return 0;
}
```

Werkt `C = (F - 32) * 5 / 9;` ook goed?

Huiswerk: F → C

- Schrijf een programma waarmee een temperatuur in graden Fahrenheit omgerekend kan worden naar graden Celsius.
  - De temperatuur in graden Fahrenheit is een floating point getal met 2 cijfers achter de decimale punt.
  - De temperatuur in graden Celcius moet worden afgerond op 2 cijfers achter de decimale punt.

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

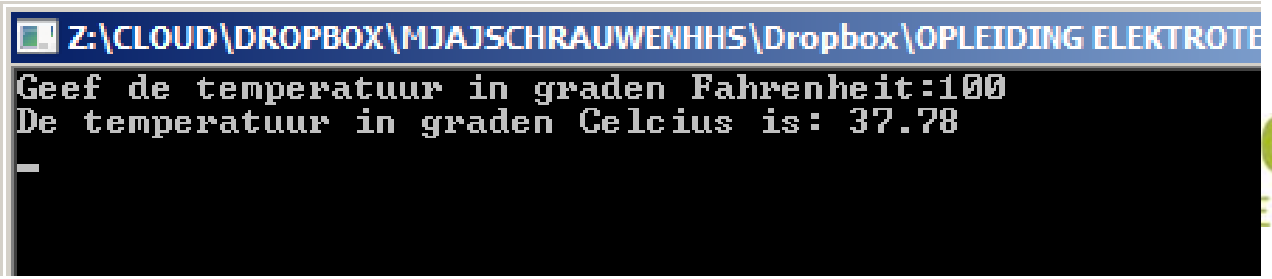
Uitwerking huiswerk: F → C

```
#include <stdio.h>

int main(void) {
 float C, F;
 printf("Geef de temperatuur in graden Fahrenheit:");
 scanf("%f", &F);
 C = (F - 32) * 5 / 9;
 printf("De temperatuur in graden Celcius is: %.2f\n", C);
 fflush(stdin);
 getchar();
 return 0;
}
```

Waarom werkt

`C = (F – 32) * 5 / 9` nu wel goed?

Werkt `C = 5 / 9 * (F – 32)` ook goed?



Huiswerk

- Bestudeer C boek :
  - Hoofdstuk 1 en 2

- Maak opdrachten:
  - 3, 5, 10, 12 van https://www.w3resource.com/c-programming-exercises/basic-declarations-and-expressions/index.php
  - Slide 37