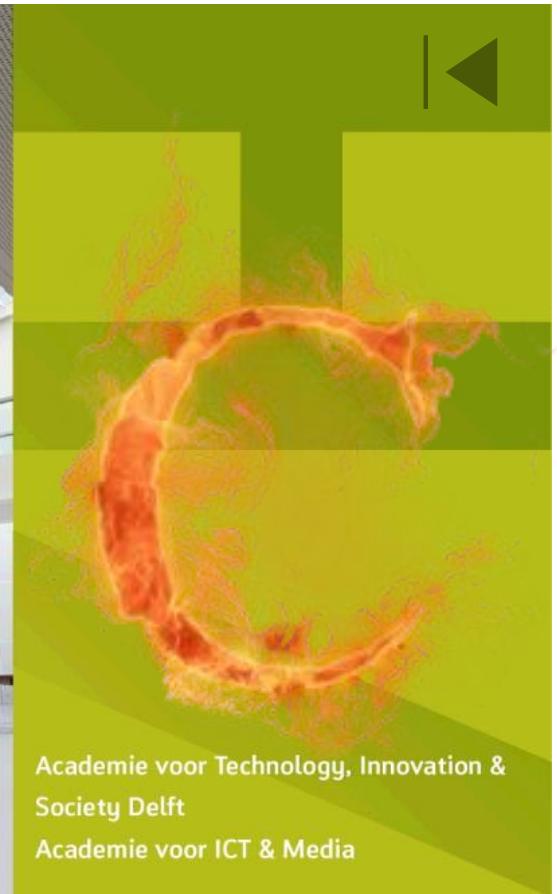


1



Gestructureerd programmeren in C

Introductie

Bijzondere dank aan...

- Harry broeders
 - Heeft dit vak opgezet

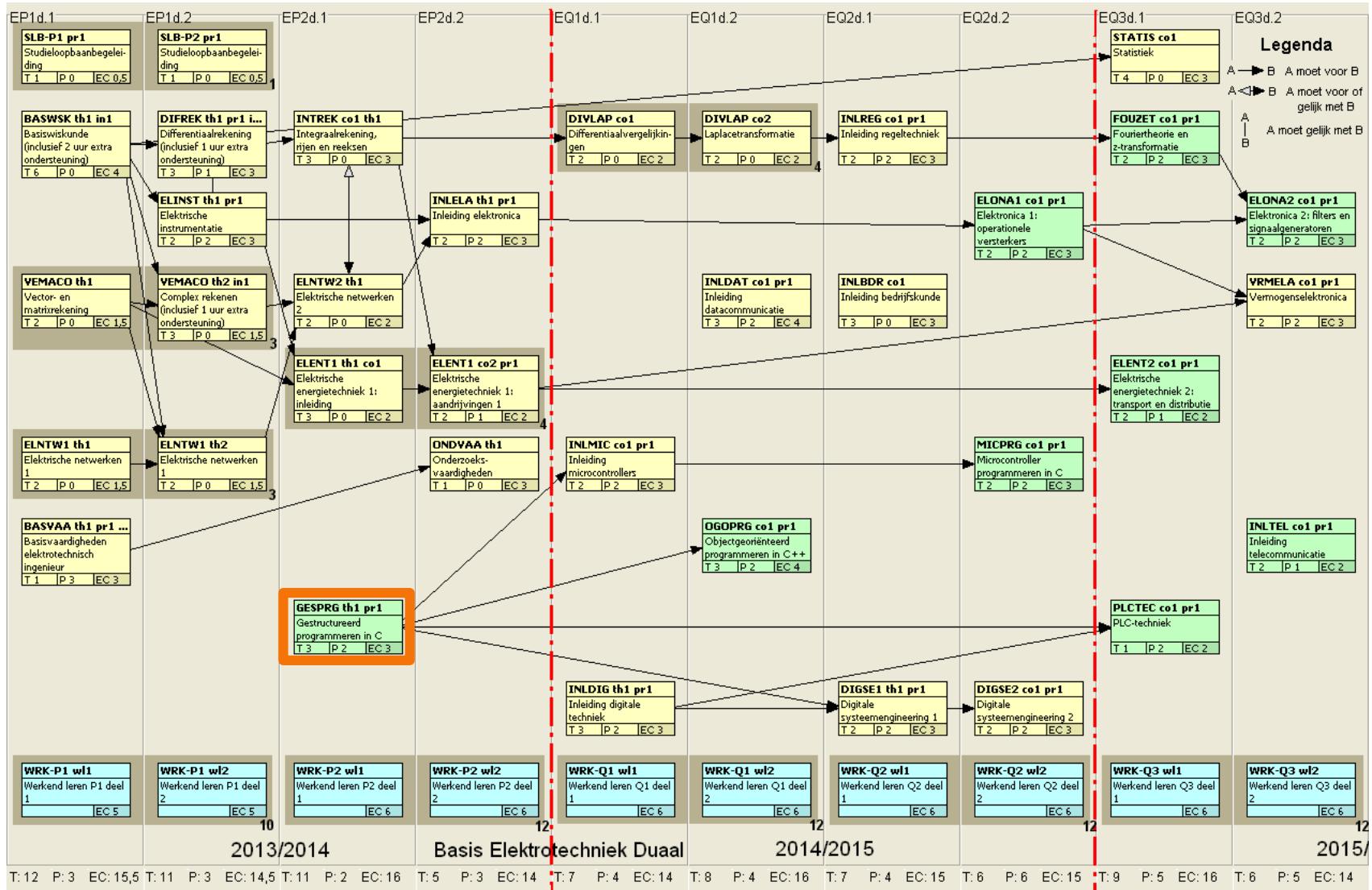


Introductie



- Theorie:
 - Mark Schrauwen
 - Werkzaam bij Bewegingstechnologie (4 dagen)
 - Werkzaam bij Elektrotechniek (1 dag)
 - mjschrau@hhs.nl (zet in het onderwerp: **GESPRG: ...**)
 - 0704458279
- Practicum:
 - Mark Schrauwen
 - Ben Kuiper

Samenhang en belang

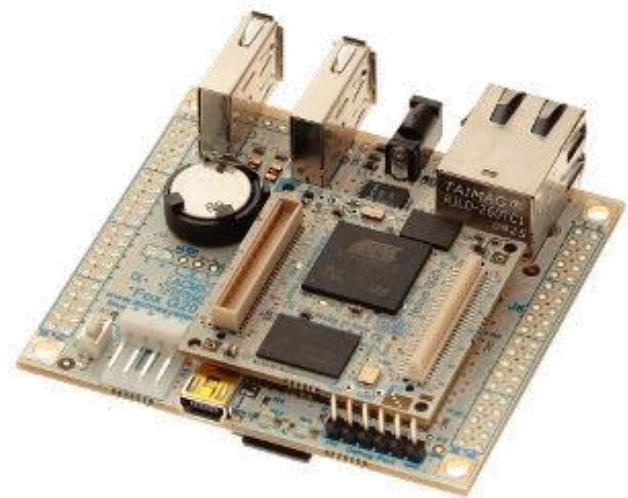


Plaats in het curriculum



- Voorbereiding voor:
 - PRO-P2 (Eind Project P)
 - INLMIC en MICPRG (Microcontroller programmeren)
 - OGOPRG (Object georiënteerd programmeren in C++)
 - Vak in ECV (RTSYST = Real-time systemen)
 - Keuzevak in EVMIN (ALGODS = Algoritmen en datastructuren)
 - Minor in ECN (Embedded Systems)

**PROEPP → FOX Board G20
Linux bordje
te programmeren in C**





Werkvormen GESPRG

- GESPRG th1 + GESPRG pr1 = **112 SBU**.
 - 21 lesuren theorie
 - 14 lesuren college
 - 7 lesuren werkcollege
 - 14 lesuren practicum.
 - 2 lesuren uitloop practicum (**lesweek 10 op afspraak**)
 - 2 lesuren schriftelijke toets.
 - 75 uur zelfstudie = **8 uur/week zelfstudie + voorbereiden practicum!**
- Toetsing:
 - Schriftelijke toets **GESPRG th1** in lesweek 8 en 10 van dit blok.
 - **GESPRG pr1** practicumopgaven worden afgetekend op het practicum. **Alle opgaven moeten voldoende zijn.**



Werkcolleges

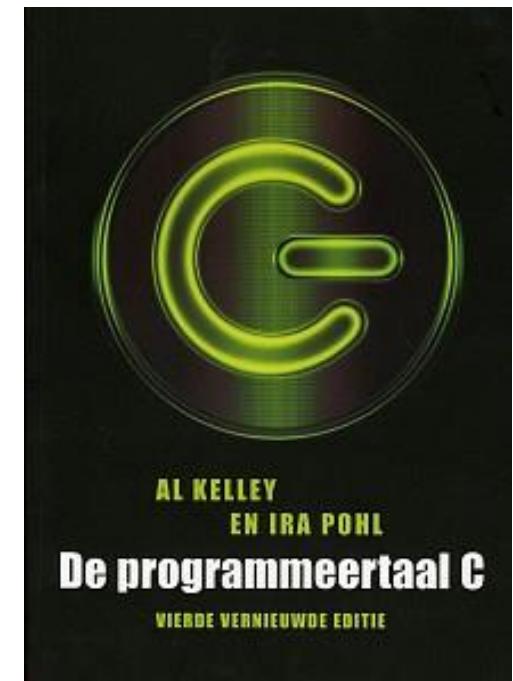
- Verder/dieper in op de les
- Behandelen van vragen/opdrachten
- Behandelen van huiswerk
- Extra opdrachten
- Peer review



Leermiddelen



- Boek: **De programmeertaal C**,
4e vernieuwde editie,
Al Kelley en Ira Pohl,
ISBN 9789043016698.
- [Microsoft visual studio 2013 express](#)



Tentamen



- Leerstof:
 - Opgegeven huiswerk
 - Deze presentatie
- Vaardigheden opgedaan in het practicum zijn nodig voor het tentamen!
- Het tentamen wordt altijd onderschat!
- Oefententamens staan op Blackboard
- TIPS

Gestructureerd Programmeren in C (GESPRG).

© Harry Broeders. Laatste wijziging: 10 september 2014

Deze pagina is bestemd voor studenten van de Haagse Hogeschool - Academie voor Technology, Innovation & Society Delft.

Inleiding.

De practicumopdrachten van dit vak kun je op BB vinden onder documents. Deze website bevat vooral veel informatie over de theorie.

The screenshot shows a Blackboard course page for 'PRACTICA'. The left sidebar contains a navigation menu with sections like '1415 GESPRG: Gestructureerd programmeren in C 2014/2015', 'Leerstof', 'Berichten', 'Docenten', 'COURSE MANAGEMENT', 'Control Panel', 'My Content', 'Course Tools', 'Evaluation', 'Grade Centre', 'Users and Groups', 'Customisation', 'Packages and Utilities', and 'Help'. The main content area has a green header 'PRACTICA'. Below it, there are four items: 'Practica opdrachten' (with a document icon), 'Inleiding in het gebruik van Visual C++ Express Edition 2013' (with a document icon), 'Practicumkaart' (with a document icon), and 'Bestanden nodig voor practicumopdracht 6' (with a folder icon). Each item has a brief description and a link to more information.

- Fout op pagina 30 voorbeeld arrays:

- De onderste for-lus heeft geen compound-statement (accolades). Als je dit programma uitvoert in MVS2012 worden de ingevoerde waarden niet geprint. Wel wordt een array-waarde geprint die buiten het array valt. De juiste versie van het voorbeeld staat hier [BoekVerbeterdeCodePagina30.c](#).

COLLEGES:

- Pas op: De onderstaande presentaties bevatten uitwerkingen van het huiswerk. **Probeer eerst zelf je huiswerk te maken!**



Inhoud cursus

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

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 struct)





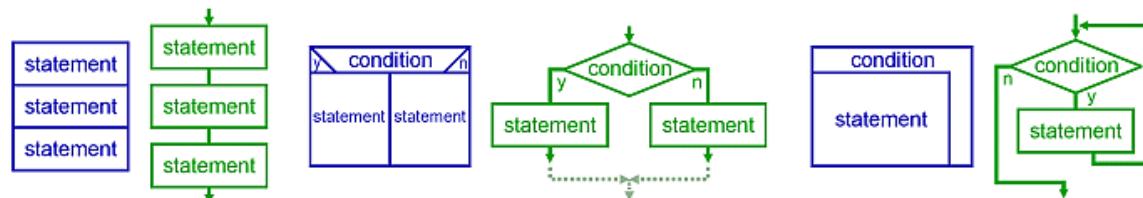
Wat weet je al?

- Welke programmeertalen ken jij al?
- Waarom met de programmeertaal C?
- Wat is gestructureerd programmeren?
- Waarom gestructureerd leren programmeren bij E?

Gestructureerd programmeren



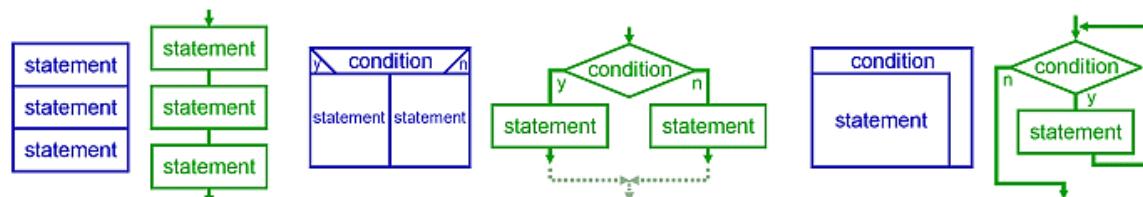
- Wat is **gestructureerd** programmeren?
 - Een (programmeer) paradigma
 - Subdiscipline van procedureel programmeren
- Waarom kiezen we voor gestructureerd programmeren?
 - Bewezen effectief (tot zekere hoogte)
 - Veel gebruikt (aansluiting werkveld)



Gestructureerd programmeren

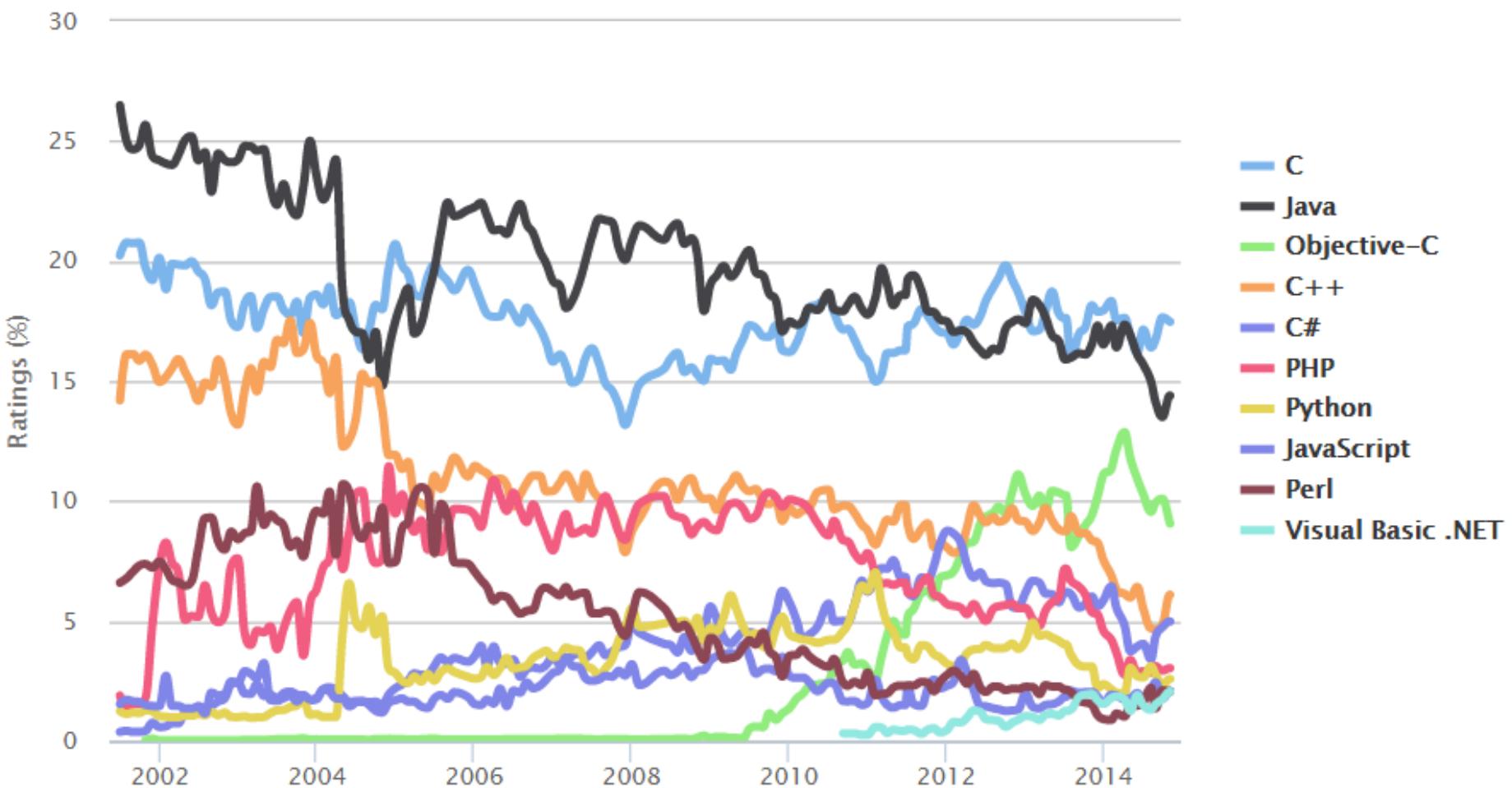


- Belangrijkste verschillen tussen gestructureerd en procedureel:
 - Gebruik van subroutines (in C functies genoemd)
 - Gebruik van ‘block structures’ groepering van code
 - Maar de verschillen zijn vaag...



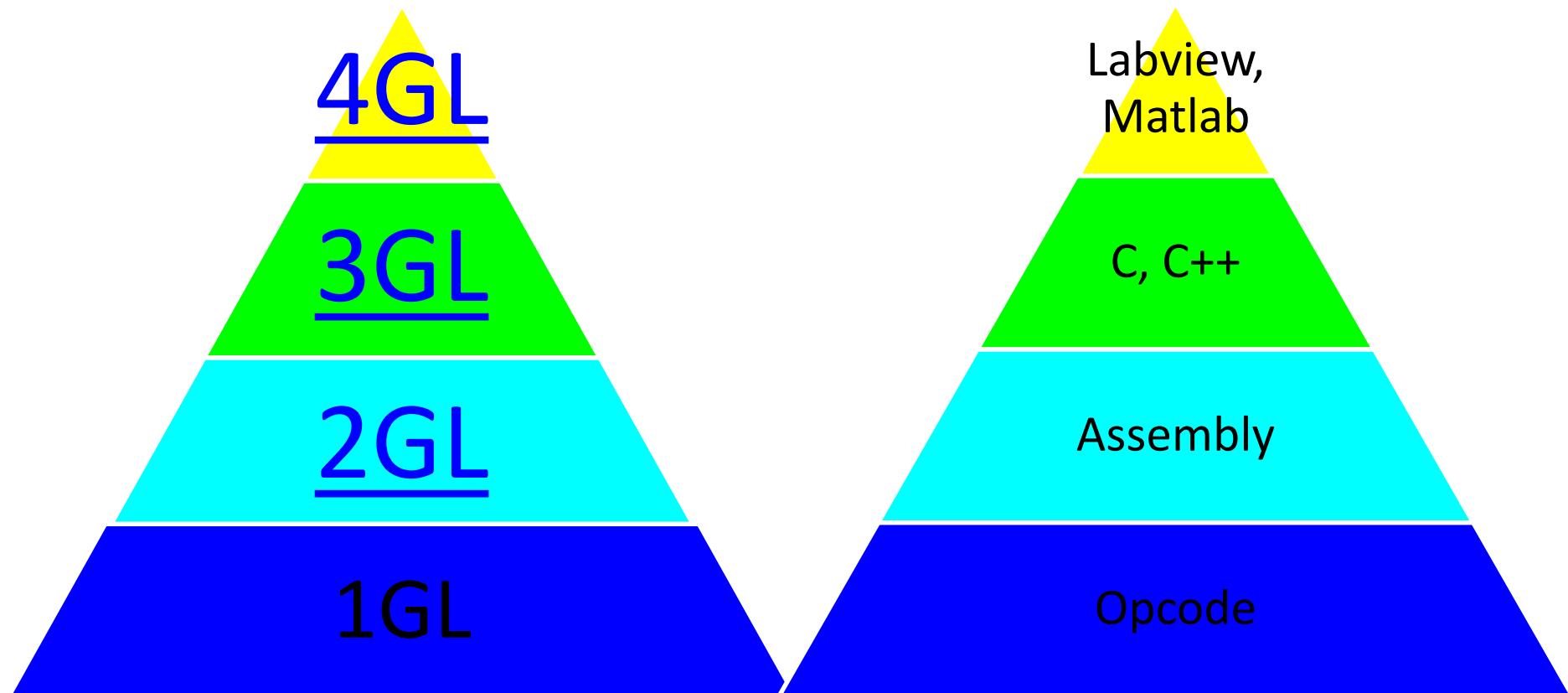


Waarom programmeren in C?

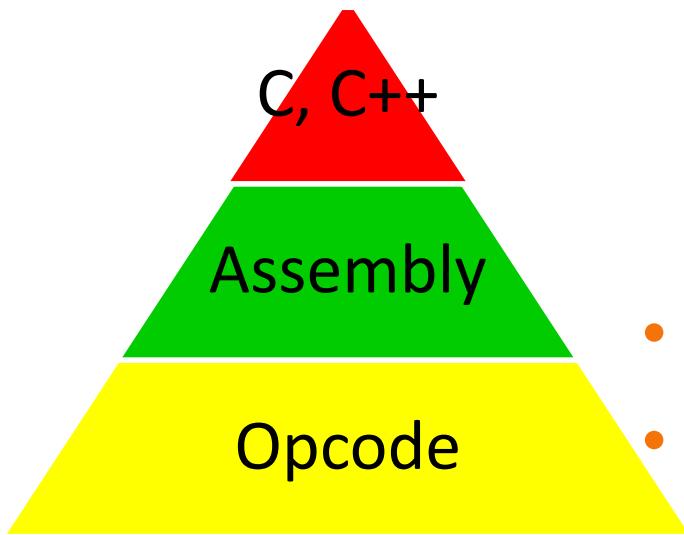


Bron: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Hiërarchie programmeertalen



Waarom programmeren in C?



- Op hardware niveau worden processor instructies weergegeven met *opcodes*:
 - A1 FE 00 A7 56
- Nadelen van opcodes?
- Assembly bestaat uit mnemonische opcodes. Assembly is ‘leesbaarder’





Eerste C programma





Eerste C programma

```
#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;
}
```



```
Z:\DROPBOX\Werk\Dropbox\OPLEIDING ELEKTROTECHNIEK\GSPRG\MS CODE\LES1\Les1-1\Debug\Les...
Het product van 6 en 10 is: 60
Sluit dit venster door op een toets te drukken
```

Huiswerk



- Bestudeer C boek:
 - hoofdstuk 1 tot paragraaf 1.4.
- Maak opdrachten:
 - 1a, 1b en 4 van paragraaf 1.12.



2



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

Gestructureerd programmeren in C

Variabelen



Basis variabelen

- Gehele unsigned variabelen

Variabelen	bits	Decimale waardes
unsigned char	8	$0 \dots 2^8 - 1 \rightarrow 0 \dots 255$
unsigned int	16	$0 \dots 2^{16} - 1 \rightarrow 0 \dots 65535$
unsigned long	32	$0 \dots 2^{32} - 1 \rightarrow 0 \dots 4294967296$

Mede afhankelijk van
versie OS (32/64) bit



Basis variabelen

- Gehele signed variabelen

Variabelen	Bits	Decimale waardes
Signed char	8	$-2^7 \dots 2^7 - 1 \rightarrow -128 \dots 127$
Signed int	16	$-2^{15} \dots 2^{15} - 1 \rightarrow -32768 \dots 32767$
Signed long	32	$-2^{31} \dots 2^{31} - 1$ $\rightarrow -2147483648 \dots 2147483647$

Basis variabelen



- Gehele variabelen

Variabelen	bits	Decimale waardes
char	8	?
int	16	?
long	32	?

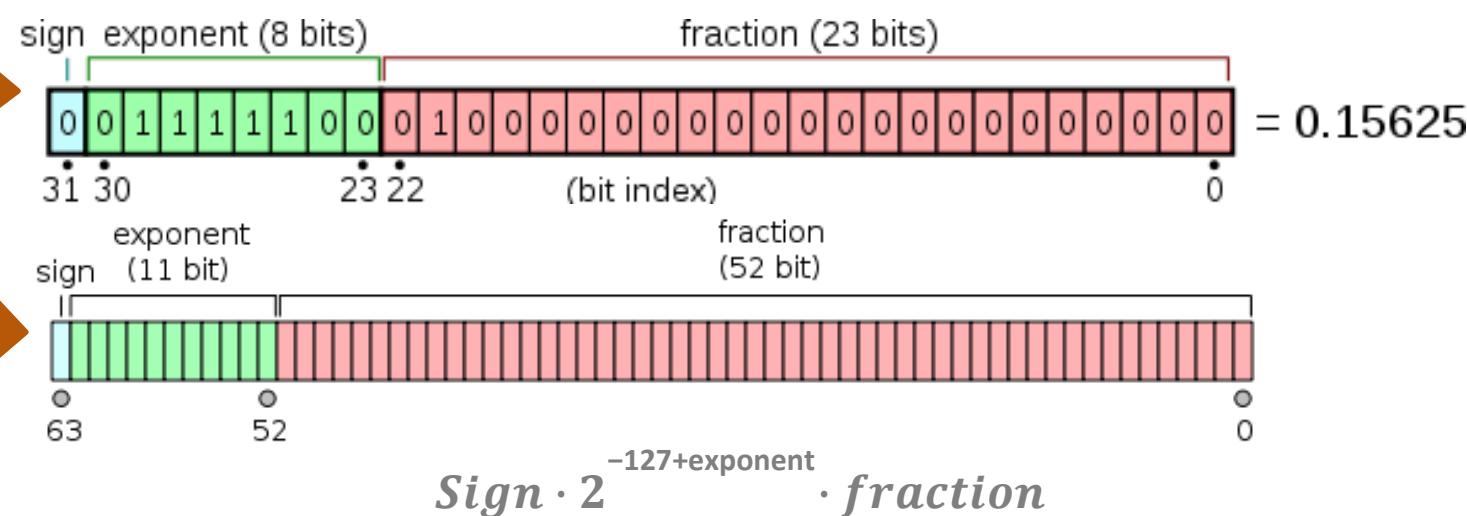


Basis variabelen

- Zwevendekommagetallen (floating point)

Variabelen	bits
float	32
double	64
Long double	128

Geen leerstof





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);
```

```
Z:\DROPBOX\Werk\Dropbox\OPLEIDING ELEKTROTEC  
26 / 7 = 3  
26 % 7 = 5
```



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?



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
0.333333333333331000
```



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 correct worden afgerond tot **een geheel getal**.

Voorbeelden:

$$T_F = 100 \Rightarrow T_C = \frac{5}{9} (T_F - 32) \Rightarrow T_C = \frac{5}{9} (100 - 32) \Rightarrow T_C = \frac{5}{9} \cdot 68 \Rightarrow T_C = 38$$

$$T_F = 0 \Rightarrow T_C = \frac{5}{9} (T_F - 32) \Rightarrow T_C = \frac{5}{9} (0 - 32) \Rightarrow T_C = \frac{5}{9} \cdot (-32) \Rightarrow T_C = -17.7777\ldots \Rightarrow T_C \approx -18$$



Voorbeeld: F → 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?

The screenshot shows a Windows command prompt window. The title bar reads "C:\Documents and Settings\Harry\mijn documenten\visu...". The window contains the following text:
Geef de temperatuur in graden Fahrenheit: 100
De temperatuur in graden Celcius is: 0



Voorbeeld: F → 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?

The screenshot shows a Windows command prompt window. The title bar reads "C:\Documents and Settings\Harry\mijn documenten\visu...". The command "Geef de temperatuur in graden Fahrenheit: 100" is entered, followed by the output "De temperatuur in graden Celcius is: 37".



Voorbeeld: F → 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?

```
Z:\CLOUD\DROPBOX\MJAJSCHRAUWENHHS\Dropbox\OPLEIDING
Geef de temperatuur in graden Fahrenheit: 100
De temperatuur in graden Celcius is: 38
```



Voorbeeld: F → 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.

```
Z:\CLOUD\DROPBOX\MJAJSCHRAUWENHHS\Dropbox\OPLEIDING
Geef de temperatuur in graden Fahrenheit: 0
De temperatuur in graden Celcius is: -17
```

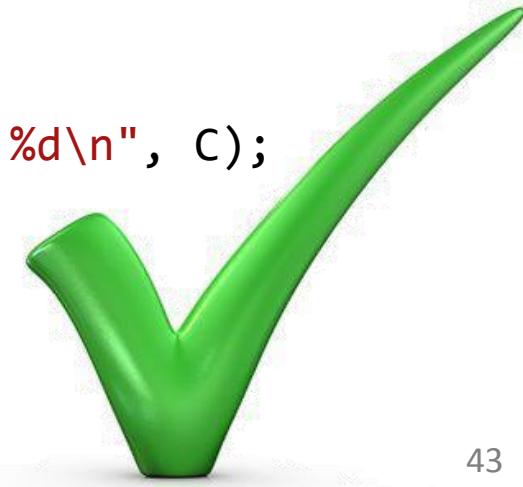
Voorbeeld: F → C



```
#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

```
#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**.
 - De temperatuur in graden Celcius is een **floating point getal en** 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?

```
Z:\CLOUD\DROPBOX\MJAJSCHRAUWENHHS\Dropbox\OPLEIDING ELEKTROTE  
Geef de temperatuur in graden Fahrenheit:100  
De temperatuur in graden Celcius is: 37.78
```

Huiswerk



- Bestudeer C boek:
 - paragrafen 2.3 t/m 2.6.
 - paragrafen 2.8 t/m 2.9.
 - paragraaf 3.1.
 - paragraaf 3.4.
 - paragraaf 3.6.

- Maak opdrachten:
 - 5, 7 en 9 van paragraaf 2.15.
 - 1 van paragraaf 3.13.



3



Gestructureerd programmeren in C

Vergelijken & herhalen

Programmeren



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

Deze les

Subcategorie:

- Vergelijken





Vergelijken

- Relationale operatoren:

Relationale operator	Betekenis	Teken in de wiskunde
>	Groter dan	>
<	Kleiner dan	<
>=	Groter of gelijk aan	\geq
<=	Kleiner of gelijk aan	\leq
==	Is gelijk aan	=
!=	Ongelijk aan	\neq

Resultaat is een int (ONWAAR → 0, WAAR → 1)

Let op verschil in C tussen = en ==

Vergelijken met relationele operatoren



- OPDRACHT: pak pen en papier en schrijf voor elke onderstaande `printf()`, waarin een relationele vergelijking staat, wat de uitvoer moet zijn.

Code:

```
#include <stdio.h>

int main(void) {
    char a = 'a';
    char b = 'b';

    printf("VERGELIJKEN MAAR! \n");
    printf("5 > 3    -> %d \n", 5 > 3);
    printf("3 > 5    -> %d \n", 3 > 5);

    printf("5 >= 5   -> %d \n", 5 >= 5);
    printf("5 >= 3   -> %d \n", 5 >= 3 );
    printf("3 >= 5   -> %d \n", 3 >= 5);

    printf("5 == 5   -> %d \n", 5 == 5);
    printf("5 == 3   -> %d \n", 5 == 3);
    printf("a == a   -> %d \n", a == a);
    printf("b == a   -> %d \n", b == a);

    printf("5 != 5   -> %d \n", 5 != 5);
    printf("5 != 3   -> %d \n", 5 != 3);
    printf("a != a   -> %d \n", a != a);
    printf("b != a   -> %d \n", b != a);

    fflush(stdin); getchar(); return 0;
}
```

Uitvoer:

```
5 > 3    -> 1
3 > 5    -> 0
5 >= 5   -> 1
5 >= 3   -> 1
3 >= 5   -> 0
5 == 5   -> 1
5 == 3   -> 0
```

```
a == a   -> 1
b == a   -> 0
5 != 5   -> 0
5 != 3   -> 1
a != a   -> 0
b != a   -> 1
```



Herhalen

- Er zijn in C **3** herhalingsopdrachten
 - **for**
 - **do while**
 - **while**

In het boek wordt niet uitgelegd wanneer je welke herhalingsopdracht moet gebruiken!

for



- Gebruik een **for** als het **aantal** herhalingen bij het programmeren “bekend” is.

```
#include <stdio.h>
int main(void) {
    int i;
    for (i = 1; i != 10; i = i + 1) {
        printf("hallo %d\n", i);
    }
    getchar();
    return 0;
}
```

initialisatie

Zolang de voorwaarde WAAR is

doe telkens aan einde



Uitvoer van een for-lus

Code:

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 1; i != 10; i = i + 1) {
        printf("hallo %d\n", i);
    }
    getchar();
    return 0;
}
```

Uitvoer:

hallo 1	hallo 6
hallo 2	hallo 7
hallo 3	hallo 8
hallo 4	hallo 9
hallo 5	



Uitvoer van een for-lus

Code: #include <stdio.h>

```
int main(void) {           Gaat het nu nog goed?  
    int i;  
    for (i = 1; i != 10; i = i + 2) {  
        printf("hallo %d\n", i);  
    }  
    getchar();  
    return 0;  
}
```

Uitvoer:

hallo 1	hallo 11
hallo 3	hallo 13
hallo 5	hallo 15
hallo 7	hallo 17
hallo 9	...



Alternatieve voorwaarde

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 1; i < 10; i = i + 2) {
        printf("hallo %d\n", i);
    }
    getchar();
    return 0;
}
```

Is dit beter ?

{ bla; bla; bla; }



- Compound statement
 - Body van een statement
 - Als een compound statement uit slechts 1 statement bestaat dan wordt je niet verplicht accolade punctuatoren te gebruiken. Maar je kunt dan alleen dat ene statement gebruiken.

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 1; i < 10; i = i + 1)
        printf("hallo %d\n", i);
    getchar();
    return 0;
}
```

Is dit aan te raden?



Inspringen

- Maak je programma **leesbaar** door netjes in te springen.

```
#include <stdio.h>
int main(void)
{
    int i;
    for (i = 1; i < 10; i = i + 1)
    {
        printf("hallo %d\n", i);
    }
    getchar();
    return 0;
}
```



Inspiringen

- Maak je programma **leesbaar** door netjes in te springen.

```
#include <stdio.h>

int main(void)
{
    int i;
    for (i = 1; i < 10; i = i + 1)
    {
        printf("hallo %d\n", i);
    }
    getchar();
    return 0;
}
```

Er zijn verschillende veel
gebruikte manieren. Kies zelf,
maar blijf wel consequent!



Voorbeeld: $1+2+3+\dots+100 = ?$

```
#include <stdio.h>

int main(void) {
    int i, som = 0;
    for ( ??? ) {
        som = som + i;
    }
    printf("som = %d\n", som);
    getchar();
    return 0;
}
```

Kan dit slimmer?

http://nl.wikipedia.org/wiki/Somformule_van_Gauss





do while

- Gebruik een **do while** als het **aantal** herhalingen bij het programmeren “onbekend” is en ≥ 1 .

```
#include <stdio.h>

int main(void) {
    int getal;
    do {
        printf("Geef een positief getal: ");
        scanf("%d", &getal);
    } while (getal <= 0);
    printf("Het ingevoerde getal = %d\n", getal);
    fflush(stdin);
    getchar();
    return 0;
}
```

Wat is de vertaling van het Engelse woord: while?



Voorbeeld: do-while

Code:

```
#include <stdio.h>

int main(void) {
    int getal;
    do {
        printf("Geef een positief getal: ");
        scanf("%d", &getal);
    } while (getal <= 0);
    printf("Het ingevoerde getal = %d\n", getal);
    fflush(stdin);
    getchar();
    return 0;
}
```

Uitvoer:

```
Geef een positief getal: -100
Geef een positief getal: 0
Geef een positief getal: 0
Geef een positief getal: -1
Geef een positief getal: 1
Het ingevoerde getal = 1
```



while

- Gebruik een **while** als het **aantal** herhalingen bij het programmeren “onbekend” is en ≥ 0 is.

```
#include <stdio.h>
int main(void) {
    int getal;
    printf("Geef een positief getal: ");
    scanf("%d", &getal);
    while (getal <= 0) {
        printf("Nee dombo! Geef een positief getal: ");
        scanf("%d", &getal);
    }
    printf("Het ingevoerde getal = %d\n", getal);
    fflush(stdin);
    getchar();
    return 0;
}
```

zolang...



Voorbeeld: while

Code:

```
#include <stdio.h>
int main(void) {
    int getal;
    printf("Geef een positief getal: ");
    scanf("%d", &getal);
    while (getal <= 0) {
        printf("Nee dombo! Geef een positief getal: ");
        scanf("%d", &getal);
    }
    printf("Het ingevoerde getal = %d\n", getal);
    fflush(stdin);
    getchar();
    return 0;
}
```

Uitvoer:

```
Geef een positief getal: -1
Nee dombo! Geef een positief getal: 0
Nee dombo! Geef een positief getal: 1
Het ingevoerde getal = 1
```

Wat heeft je voorkeur in dit voorbeeld? Een do-while of een while lus?



Hoelang is een...

- Hoelang wordt de onderstaande code uitgevoerd?

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

```
int main(void) {
    while(1) {
        printf("Hallo");
    }
}
```

- Hoe ziet de for-lus versie van deze code eruit?



Veel gemaakte fout

```
int main(void) {  
    /* Doe vanalles */  
    if(d=0){  
        /* Doe vanalles */  
    }  
    else{  
        /* Doe vanalles */  
    }  
    return 0;  
}
```

Wat is het verschil
tussen = en ==?

Wat is hier fout?



Vraag

```
for(expressie1; expressie2; expressie3)  
    statements  
}
```

Zijn deze herhalingslussen
in elkaar om te zetten?

<i>expressie1</i>	<i>expressie1</i>
<i>while</i> (<i>expressie2</i>) {	do
<i>statements</i>	{
<i>expressie3</i>	<i>statements</i>
}	<i>expressie3</i>
	}
	<i>while</i> (<i>expressie2</i>)



Vraag

```
for(expressie1; expressie2; expressie3)  
    statements  
}
```

Wanneer wordt elke
herhalingslus herhaald?

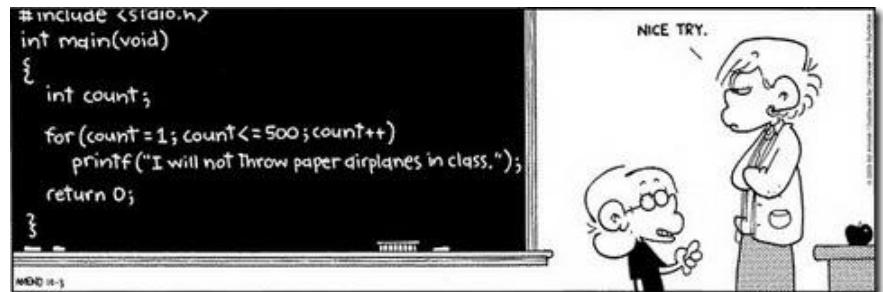
```
expressie1                                expressie1  
while(expressie2){                      do  
    statements                            {  
    expressie3                            statements  
}  
                                         expressie3  
                                         } while(expressie2)
```

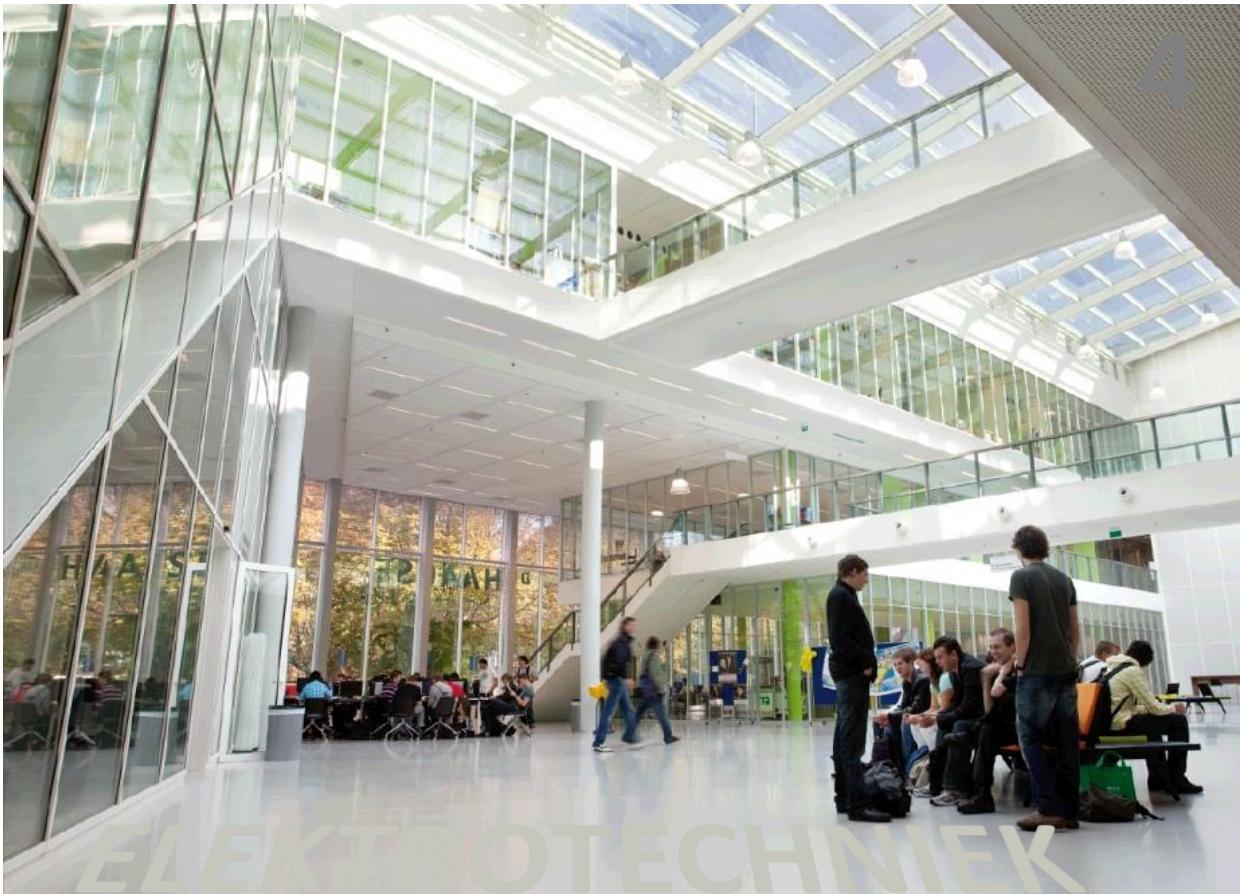
Als expressie 1/2/3 ONWAAR/WAAR is

Huiswerk

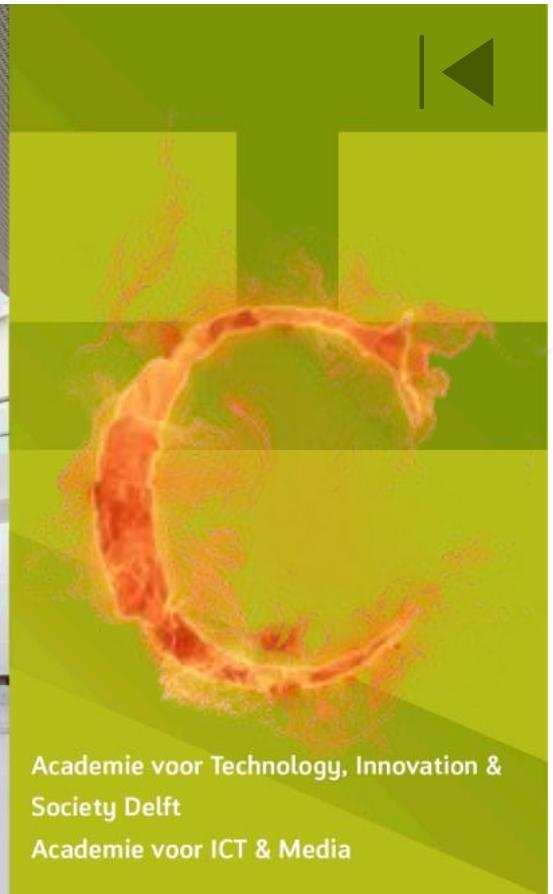


- Schrijf een programma dat de tafels van 1 t/m 5 netjes naast elkaar afstuurt.
- Bestudeer C boek:
 - paragraaf 1.6.
 - paragrafen 4.1 t/m 4.3.
 - paragraaf 4.5
 - paragrafen 4.8 t/m 4.9
 - paragrafen 4.12
- Maak opdrachten:
 - 7 en 10 van paragraaf 1.12.





4



Gestructureerd programmeren in C

Beslissingen

Programmeren



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

Deze les

Subcategorie:

- Vergelijken



Beslissen



- Er zijn in C 3 beslisopdrachten:
 - `if`
 - `if else`
 - `Switch`
- Maar eerst...





Booleaanse operatoren

- And **&&** (binaire operatie)
- Or **||** (binaire operatie)
- Not **!** (unaire operatie)

```
do {  
    printf("Geef je cijfer: ");  
    scanf("%d", &cijfer);  
} while (cijfer < 0 || cijfer > 10);
```

Veel gemaakt fout:

$!(0 \leq cijfer \leq 10)$

Conversie van WAAR en ONWAAR



- Een expressie wordt in C als dat nodig is impliciet (automatisch) omgezet naar WAAR of ONWAAR.
 - Een expressie met de waarde 0 wordt ONWAAR.
 - Een expressie met een waarde ongelijk aan 0 wordt WAAR.

Dus:

```
if (i) {  
    printf("Hallo");  
}
```

Is hetzelfde als:

```
if (i != 0) {  
    printf("Hallo");  
}
```

Het is beter om expliciet te zeggen wat je bedoeld.

if



- Als de expressie WAAR is, wordt *statement1* uitgevoerd.

```
if (expressie) {  
    statement1;  
}
```

if



- Lees 2 gehele getallen in en druk de grootste af

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

```
int main(void) {
    int max, getal;
    printf("Geef een getal: ");
    scanf("%d", &max);
    printf("Geef nog een getal: ");
    scanf("%d", &getal);
    if (getal > max) {
        max = getal;
    }
    printf("Het maximum is: %d\n", max);
    fflush(stdin);
    getchar();
    return 0;
}
```



if else



- Als de expressie WAAR is, wordt *statement1* uitgevoerd.
- Als de expressie ONWAAR is, wordt *statement2* uitgevoerd.

```
if (expressie) {  
    statement1;  
}  
else {  
    statement2;  
}
```

if else



- Lees 2 gehele getallen in en druk de grootste af

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

```
int main(void) {
    int a, b, max;
    printf("Geef een getal: ");
    scanf("%d", &a);
    printf("Geef nog een getal: ");
    scanf("%d", &b);
    if (a > b) {
        max = a;
    }
    else {
        max = b;
    }
    printf("Het maximum is: %d\n", max);
    fflush(stdin); getchar(); return 0;
}
```

Welke van deze twee programma's vind jij **beter**?



Bij welke if hoort else ?



```
if (a > 3)
    if (a < 2)
        printf("a\n");
else
    printf("b\n");
```

```
if (a > 3)
    if (a < 2)
        printf("a\n");
else
    printf("b\n");
```

Geven beide programmadelen dezelfde uitvoer als a = 1?

Welke uitvoer?

Bij welke if hoort else ?



```
if (a > 3) {  
    if (a < 2) {  
        printf("a\n");  
    }  
}  
else {  
    printf("b\n");  
}
```

```
if (a > 3) {  
    if (a < 2) {  
        printf("a\n");  
    }  
    else {  
        printf("b\n");  
    }  
}
```

Geven beide programmadelen dezelfde uitvoer als a = 1?

Welke uitvoer?

switch



- Zet Nederlands toetscijfer om naar Amerikaans resultaat. Ga uit van:

Nederlands	Amerikaans
8, 9 of 10	A
7	B
6	C
5	D
0, 1, 2, 3 of 4	F

Switch



```
switch (cijfer) {  
    case 10:  
    case 9:  
    case 8:  
        letter = 'A'; break;  
    case 7:  
        letter = 'B'; break;  
    case 6:  
        letter = 'C'; break;  
    case 5:  
        letter = 'D'; break;  
    default:  
        letter = 'F'; break;  
}
```



if else



```
if (cijfer == 10 || cijfer == 9 || cijfer == 8) {  
    letter = 'A';  
}  
else if (cijfer == 7) {  
    letter = 'B';  
}  
else if (cijfer == 6) {  
    letter = 'C';  
}  
else if (cijfer == 5) {  
    letter = 'D';  
}  
else {  
    letter = 'F';  
}
```

Is deze else nodig?



if



```
if (cijfer == 10 || cijfer == 9 || cijfer == 8) {  
    letter = 'A';  
}  
if (cijfer == 7) {  
    letter = 'B';  
}  
if (cijfer == 6) {  
    letter = 'C';  
}  
if (cijfer == 5) {  
    letter = 'D';  
}  
if (cijfer < 5 || cijfer > 10) {  
    letter = 'F';  
}
```

Wat is het verschil met de vorige sheet?

Waar wordt hier geen rekening mee gehouden?

Wat is het verschil in gedrag?



Short-circuit evaluation



- Bij het uitvoeren van de booleanse operatoren `||` en `&&` wordt gestopt zodra de uitkomst bekend is.
- Als a deelbaar is door b dan ...

```
if (a % b == 0) ...
```

Gaat fout als `b == 0`

```
if (b != 0 && a % b == 0) ...
```

Gaat goed als `b == 0`

Dankzij short-circuit evaluation





Oefening operatoren

```
char a = 'a', b = 'b', g = 4;
```

```
printf("BOOLEAANSE EN RELATIONELE OPERATOREN, VERGELIJKEN MAAR! \n");
```

```
printf("5>3 && a!=b -> %d \n", 5>3 && a!=b);
```

```
printf("5>3 && a==b -> %d \n", 5>3 && a==b);
```

```
printf("!(5>3 && a!=b) -> %d \n", !(5>3 && a!=b));
```

```
printf("5<3 && a==b -> %d \n", 5<3 && a==b);
```

```
printf("5>3 || a!=b -> %d \n", 5>3 || a!=b);
```

```
printf("5>3 || a==b -> %d \n", 5>3 || a==b);
```

```
printf("!(5>3 || a==b) -> %d \n", !(5>3 || a==b));
```

```
printf("!(5<3) || a==b -> %d \n", !(5<3) || a==b);
```

```
printf("!(!(5<3) || a==b) -> %d \n", !(!(5<3) || a==b));
```

```
printf("!((5<3 || a==b) && 1) -> %d \n", !((5<3 || a==b) && 1));
```

```
printf("!(!(5<g) || a==b) -> %d \n", !(!(5<g) || a==b));
```

Pak pen en papier en schrijf de uitvoer van dit programma op



Vergelijken met operatoren

BOOLEAANSE EN RELATIONELE OPERATOREN, VERGELIJKEN MAAR!

Uitvoer: 5>3 && a!=b -> 1

5>3 && a==b -> 0

!(5>3 && a!=b) -> 0

5<3 && a==b -> 0

5>3 || a!=b -> 1

5>3 || a==b -> 1

!(5>3 || a==b) -> 0

!(5<3) || a==b -> 1

!(!!(5<3) || a==b) -> 0

!((5<3 || a==b) && 1) -> 1

(!((5<3) || a==b)) -> 1

Programmeren == Moeilijk ?



- Schrijf een programma dat...
 - Hoe bedenk je een programma?
- Stap voor stap...
 - Stapgewijze verfijning
 - http://bd.eduweb.hhs.nl/gesprg/tafels_stap_voor_stap.htm

Vind je de practica
moeilijk let dan
vooral goed op!





Tafels stap voor stap

- Schrijf een programma dat een geheel getal $0 < n < 7$ inleest en vervolgens de tafels van 1 t/m n naast elkaar afdrukt.



Stap 0: Bezint eer gij begint



- Snap je de opdracht?
- Bedenk mogelijke testgevallen.
 - Bij de invoer 4 moet het programma de volgende uitvoer produceren:

$1 \times 1 =$	1	$1 \times 2 =$	2	$1 \times 3 =$	3	$1 \times 4 =$	4
$2 \times 1 =$	2	$2 \times 2 =$	4	$2 \times 3 =$	6	$2 \times 4 =$	8
$3 \times 1 =$	3	$3 \times 2 =$	6	$3 \times 3 =$	9	$3 \times 4 =$	12
$4 \times 1 =$	4	$4 \times 2 =$	8	$4 \times 3 =$	12	$4 \times 4 =$	16
$5 \times 1 =$	5	$5 \times 2 =$	10	$5 \times 3 =$	15	$5 \times 4 =$	20
$6 \times 1 =$	6	$6 \times 2 =$	12	$6 \times 3 =$	18	$6 \times 4 =$	24
$7 \times 1 =$	7	$7 \times 2 =$	14	$7 \times 3 =$	21	$7 \times 4 =$	28
$8 \times 1 =$	8	$8 \times 2 =$	16	$8 \times 3 =$	24	$8 \times 4 =$	32
$9 \times 1 =$	9	$9 \times 2 =$	18	$9 \times 3 =$	27	$9 \times 4 =$	36
$10 \times 1 =$	10	$10 \times 2 =$	20	$10 \times 3 =$	30	$10 \times 4 =$	40

Stap 1: Alle begin is ~~moeilijk~~ makkelijk.



```
#include <stdio.h>

/* © 2013 Naam programmeur */

int main(void) {
    /* Hier komt de code */

    fflush(stdin);
    getchar();
    return 0;
}
```



Stap 2: Invoer.

```
#include <stdio.h>

/* © 2013 Naam programmeur */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void) {
    int n;

    printf("Geef de waarde van n (1..6): ");
    scanf("%d", &n);
    printf("Test n = %d", n);

    fflush(stdin);
    getchar();
    return 0;
}
```



Stap 3: Controle op invoer.

```
#include <stdio.h>

int main(void) {
    int n;

    do {
        printf("Geef de waarde van n (1..6): ");
        scanf("%d", &n);
    } while(n < 1 || n > 6);

    printf("Test n = %d", n);

    fflush(stdin);
    getchar();
    return 0;
}
```



Stap 4. Eerste regel van de tafels afdrukken.

```
#include <stdio.h>

int main(void) {
    int n, tafel;

    do {
        printf("Geef de waarde van n (1..6): ");
        scanf("%d", &n);
    } while(n < 1 || n > 6);

    for (tafel = 1; tafel < n + 1; tafel = tafel + 1) {
        printf(" 1 x %d = %2d ", tafel, 1 * tafel);
    }
    printf("\n");

    fflush(stdin);
    getchar(); return 0;
}
```



Stap 5: Alle regels afdrukken

```
#include <stdio.h>

int main(void) {
    int n, tafel, regel;

// ...

for (regel = 1; regel < 11; regel = regel + 1) {
    for (tafel = 1; tafel < n + 1; tafel = tafel + 1) {
        printf("%2d x %d = %2d ", regel, tafel, regel*tafel);
    }
    printf("\n");
}
// ...
```



Stap 6: Een laatste verbetering

```
#include <stdio.h>

int main(void) {
    int factor, tafel, n;

    do {
        printf("Geef de waarde van n (1..6): ");
        fflush(stdin);
    } while (scanf("%d", &n) != 1 || n < 1 || n > 6);

// ...
```



Veel gemaakte fout

Wat is hier fout?

```
do {  
    if (0 == 0){  
        printf("Geef je cijfer: ");  
        scanf("%d", &cijfer);  
    }  
} while (0 <= cijfer <= 10);
```

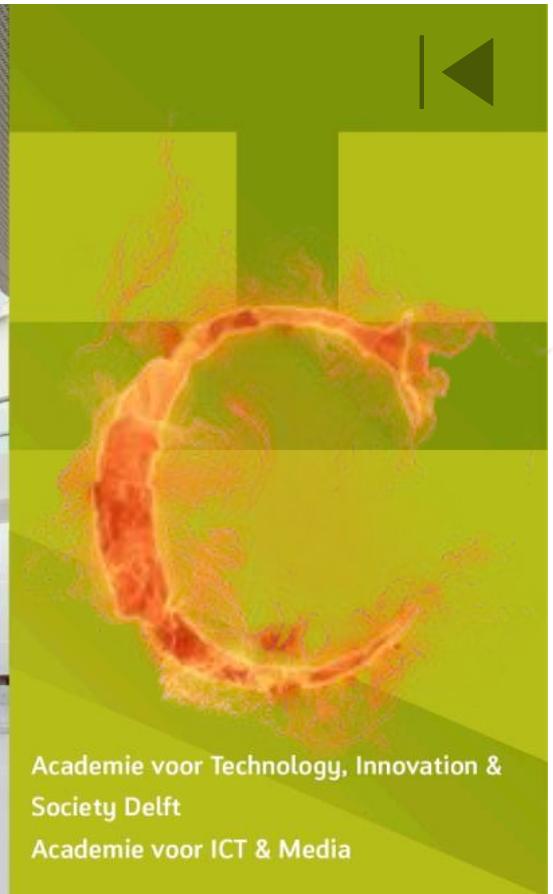
Dit is FOUT!

Wat moet het dan zijn?

Huiswerk



- Bestudeer C boek:
 - paragraaf 4.4.
 - paragraaf 4.7.
 - paragraaf 4.16.
- Maak opdrachten:
 - 9 en 33 van paragraaf 4.19.



Gestructureerd programmeren in C

Verdeel en heers



Vraag vorige stof

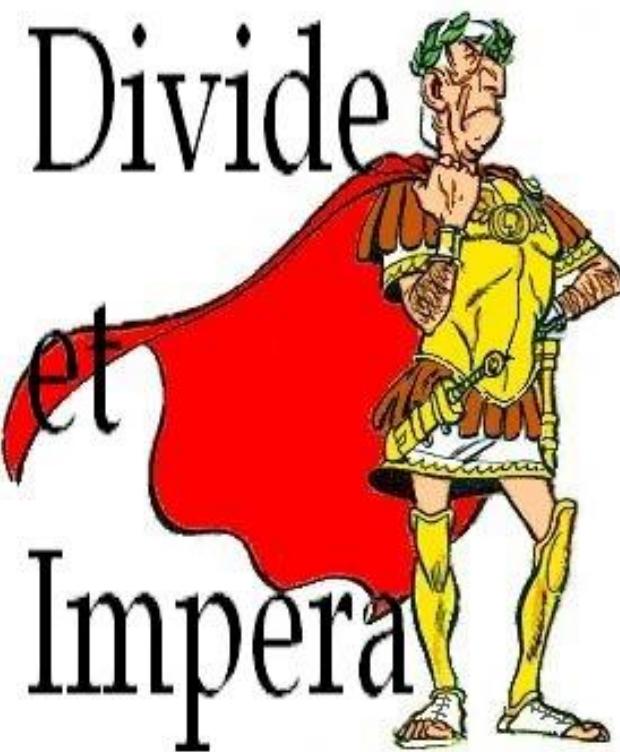
- Hoe kun je met de modulo (%) testen of een getal *even* of *oneven* is?
 - `(evengetal % 2) == 0` //even getal
 - `(evengetal % 2) != 0` //oneven getal

Snap je waarom?



Verdeel en heers

Divide
et
Impera



Problemen bij grotere prog's



- Programma wordt heel lang
 - programma **opsplitsen** in delen maakt het duidelijker.
- Sommige stukken code komen meerdere keren in het programma voor (b.v. inlezen van een positief getal + controle).
 - **slecht** voor de **onderhoudbaarheid**.
- Deel van een programma is niet eenvoudig te gebruiken in een ander programma.
 - **slecht** voor de **herbruikbaarheid**.

Functionele decompositie



- Programma opdelen in blokken met code genaamd **functies**.
- Deze functies kunnen meerdere keren worden aangeroepen.
- Goed voor **aanpasbaarheid, onderhoudbaarheid en herbruikbaarheid**.



Voorbeeld

```
int main(void) {  
    ...  
    printf("\n");  
    printf("\n");  
    printf("\n"); } Sla 3 regels over  
    ...  
    printf("\n");  
    printf("\n"); } Sla 3 regels over  
    printf("\n");  
    ...  
    return 0;  
}
```

Schrijf een functie om 3 regels
over te slaan

Opbouw van een functie



return-type functie_naam(parameter-lijst)

```
{  
    Declaraties;  
    Statements;  
    return waarde;  
}
```

opbouw

```
int naamVanFunctie(int parameter){  
    int onzin = 0;  
    onzin = onzin*parameter;  
    return onzin;  
}
```

voorbeeld



Voorbeeld

```
void sla3RegelsOver(void) {  
    printf("\n");  
    printf("\n");  
    printf("\n");  
}  
  
int main(void) {  
    ...  
    sla3RegelsOver(); } Functie aanroep  
    ...  
    sla3RegelsOver(); } Functie aanroep  
    ...  
    return 0;  
}
```

}

Functie **definitie**

void betekent leeg.

Deze functie geeft niets terug en heeft geen parameters (zie verderop)



Voorbeeld

```
int main(void) {  
    void sla3RegelsOver(void); } Functie declaratie  
... (functie prototype)  
...  
    sla3RegelsOver(); } Functie aanroep  
...  
    sla3RegelsOver(); } Functie aanroep  
...  
    return 0;  
}  
  
void sla3RegelsOver(void) {  
    printf("\n");  
    printf("\n");  
    printf("\n"); } Functie definitie  
}
```



Voorbeeld

```
int main(void) {  
    ...  
    printf("\n");  
    printf("\n");  
    printf("\n"); } Sla 3 regels over  
    ...  
    printf("\n");  
    printf("\n");  
    printf("\n");  
    printf("\n"); } Sla 4 regels over  
    ...  
    return 0;  
}
```

Schrijf een functie om een
aantal regels over te slaan

Opbouw van een functie



return-type functie_naam(parameter-lijst)

```
{  
    Declaraties;  
    Statements;  
    return waarde;  
}
```

opbouw

```
int naamVanFunctie(int parameter){  
    int onzin = 0;  
    onzin = onzin*parameter;  
    return onzin;  
}
```

voorbeeld

Wat moet worden
aangepast om een **x**
aantal regels over te
slaan?

Voorbeeld



```
void slaRegelsOver(int aantal) {  
    int teller; ——————→ lokale variabele  
    for (teller = 0; teller < aantal; teller = teller + 1) {  
        printf("\n");  
    }  
}  
  
int main(void) {  
...  
    slaRegelsOver(3);  
...  
    slaRegelsOver(4);  
...  
    return 0;  
}
```

Bij aanroep van de functie wordt de waarde van het argument gekopieerd naar de parameter (*call by value*)

argument
argument

Een parameter is een lokale variabele (die bij de functieaanroep wordt geïnitialiseerd).



Voorbeeld

```
int aantal;
```

globale variabele

```
void slaRegelsOver(void) {  
    int teller;  
    for (teller = 0; teller < aantal; teller = teller + 1) {  
        printf("\n");  
    }  
}  
  
int main(void) {  
    ...  
    aantal = 3; slaRegelsOver();  
    ...  
    aantal = 4; slaRegelsOver();  
    ...  
}
```

Waarom is het verkeerd om globale variabelen te gebruiken?

Functies: voorbeeld



```
#include <stdio.h>

char printAsciiSymbool(int asciiWaarde);

Code: int main(void) {
    int i = 0;
    for (i = 25; i < 101; i=i+1)
    {
        printAsciiSymbool(i);
    }

    fflush(stdin);
    getchar();
    return 0;
}

char printAsciiSymbool(int asciiWaarde){
    printf("%d = '%c', ",asciiWaarde,asciiWaarde);
}
```

Wat is overbodig aan deze code?

Functies: voorbeeld uitvoer



Uitvoer:

```
25 = '↓', 26 = '→', 27 = '←', 28 = '↖',  
29 = '↖', 30 = '↑', 31 = '↙', 32 = '↗',  
33 = '!', 34 = '"', 35 = '#', 36 = '$',  
37 = '%', 38 = '&', 39 = '·', 40 = '(',  
41 = ')', 42 = '*', 43 = '+', 44 = ',',  
45 = '-', 46 = '.', 47 = '/', 48 = '0',  
49 = '1', 50 = '2', 51 = '3', 52 = '4',  
53 = '5', 54 = '6', 55 = '7', 56 = '8',  
57 = '9', 58 = '⋮', 59 = '⋮', 60 = '<',  
61 = '=', 62 = '>', 63 = '?', 64 = '@',  
65 = 'A', 66 = 'B', 67 = 'C', 68 = 'D',  
69 = 'E', 70 = 'F', 71 = 'G', 72 = 'H',  
73 = 'I', 74 = 'J', 75 = 'K', 76 = 'L',  
77 = 'M', 78 = 'N', 79 = 'O', 80 = 'P',  
81 = 'Q', 82 = 'R', 83 = 'S', 84 = 'T',  
85 = 'U', 86 = 'V', 87 = 'W', 88 = 'X',  
89 = 'Y', 90 = 'Z', 91 = '[' , 92 = '\'',  
93 = ']', 94 = '^', 95 = '_', 96 = '.',  
97 = 'a', 98 = 'b', 99 = 'c', 100 = 'd'
```



Voorbeeld

- Lees een geheel getal en controleer op een minimale en maximale waarde.

```
int leesGetal(int min, int max) {  
    int getal;  
    do {  
        printf("Geef een getal [%d..%d]: ", min, max);  
        scanf("%d", &getal);  
    }  
    while (getal < min || getal > max);  
    return getal;  
}  
  
int main(void) {  
    int toetscijfer = leesGetal(1, 10);  
    ...
```

Na **aloop** van de functie wordt de **waarde** van het return statement gekopieerd naar de functieaanroep (return by value)



Veel gemaakte fout

```
int bepaalMaximumEnMinimum(int x1, int x2, int x3) {  
    /* Doe vanalles */  
    return max, min;  
}
```

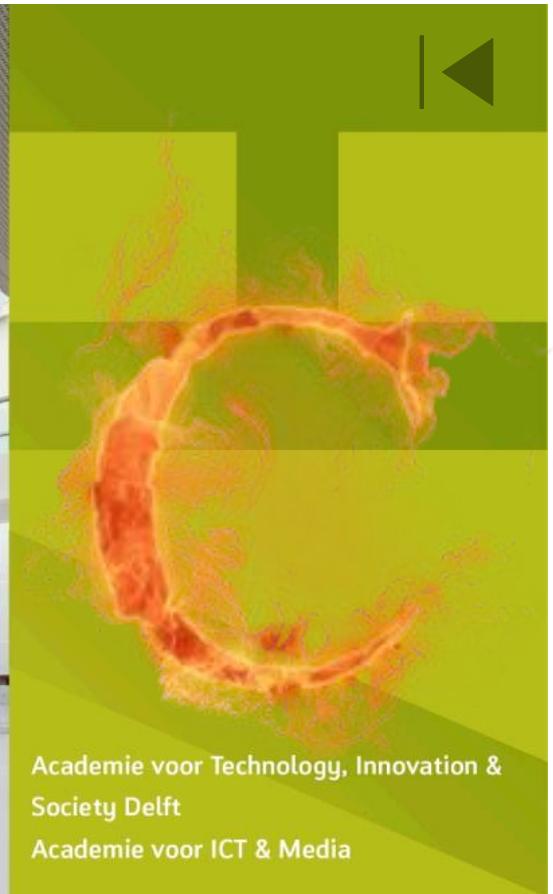
Je kunt maar één
variabele
teruggeven

Later leer je hoe je met dit probleem
kunt omgaan. (pointers)

Huiswerk



- Bestudeer C boek:
 - paragraaf 1.7.
 - paragrafen 5.1 t/m 5.5.
- Maak opdrachten:
 - 1, 2 en 3 van paragraaf 5.17.



Gestructureerd programmeren in C

Pointers

Verwisselen

Eerste poging



- Schrijf een **functie** waarmee twee **int** variabelen verwisseld kunnen worden.

```
void wissel(int a, int b) {  
    int hulpje = a;  
    a = b;  
    b = hulpje;  
}
```

```
int main(void) {  
    int x = 7, y = 8;  
    printf("x = %d en y = %d\n", x, y);  
    wissel(x, y);  
    printf("x = %d en y = %d\n", x, y);  
    getchar(); return 0;  
}
```

Niet goed!

```
x = 7 en y = 8  
x = 7 en y = 8
```

Bij aanroep van de functie wordt de waarde van het argument gekopieerd naar de parameter (*call by value*)

Geheugen



- Het geheugen van een computer bestaat uit **bytes** (**grootte**).
- Een variabele is opgeslagen in het geheugen vanaf een bepaald **adres** (een nummer).
- Het **type** van de variabele bepaalt het aantal bytes wat nodig is om de waarde van de variabele op te slaan.
- Ken je nog andere **kenmerken** van een variabele?
 - **naam** en **inhoud**.





Adres opvragen

- Je kunt het **adres** van een variabele opvragen met behulp van de unaire operator &.

```
int getal;  
scanf("%d", &getal);
```

Adres van getal, zodat scanf de ingelezen waarde op dat adres kan opslaan.

- Je kunt het aantal bytes dat een variabele in beslag neemt (**grootte**), opvragen met de unaire operator **sizeof**.



Adres opvragen

```
#include <stdio.h>

int global = 4;

int main(void) {
    double local = 7.2;

    printf("global staat op adres %p\n", &global);
    printf("global is %d bytes groot\n", sizeof global);
    printf("local staat op adres %p\n", &local);
    printf("local is %d bytes groot\n", sizeof local);

    getchar();
    return 0;
}
```

Gebruik conversie karakter
%p voor het printen van
een pointer- adres

Adres wordt in
hexadecimale talstelsel
geprint



```
Z:\CLOUD\Dropbox\OPLEIDING ELEKT...
global staat op adres 00E18000
global is 4 bytes groot
local staat op adres 002DFC74
local is 8 bytes groot
```

Dit adres is telkens
anders, weet je
waarom?

Pointers



- C kent naast de “gewone” variabelen ook pointer variabelen
- Een **pointer** wijst naar een *andere variabele*
- Een **pointer** is een variabele met een **type** (**int**, **float**, etc.)
- Een **pointer** heeft als **inhoud** het **adres** van een *andere variabele*

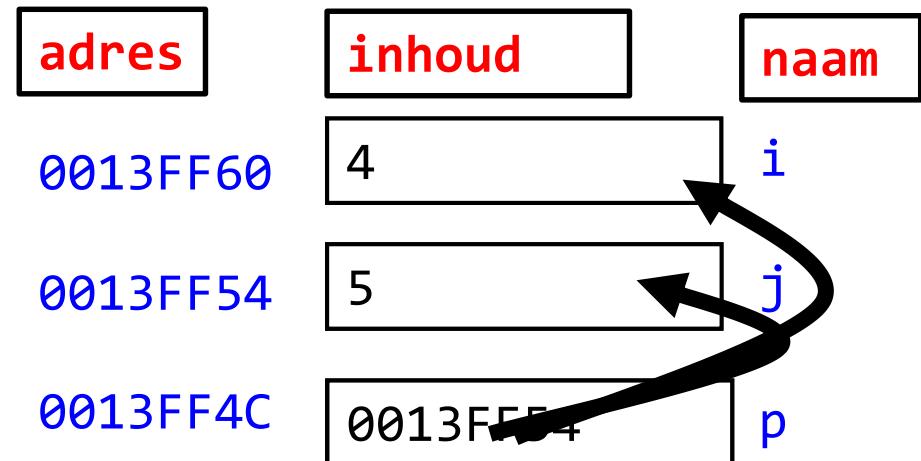


Pointers



- Je kunt het **adres** van een variabele **opslaan** in een **pointer** variabele.

```
int i = 3, j = 17;  
int *p;  
p = &i;  
*p = *p + 1;  
p = &j;  
*p = i + 1;  
p = p + 1;
```



*p → waar p naar wijst

p is een int pointer die wordt geïnitialiseerd

met het adres

Wat betekent dit ?

P naar volgende variabele laten wijzen (arrays)



Let op!

- De volgende code kan verwarringend zijn:

```
int *p = &i;
```

- Wordt hier het **adres** van i toegekend aan de **inhoud** van de variabele waar de pointer p naar wijst?
 - Nee
- Deze code bestaat uit twee delen:

```
int *p; //declaratie  
p = &i; //initialisatie
```

Herhaling: adres opvragen



- Je kunt het **adres** van een variabele opvragen met behulp van de unaire operator **&**.
- De unaire operator ***** wordt de **dereferentie** of **indirectie** operator genoemd.

Enig idee waarom?

```
int *p = &i;
```

- Je kunt het **aantal bytes** dat een variabele in beslag neemt, opvragen met de unaire operator **sizeof**.



Oefening met pointers

- Doe deze oefening niet op een laptop/pc maar **met pen en papier!**

Code:

```
int i = 30, j = 33;  
1. int *p2 = &j;  
2. int *p = &i;  
3. *p = *p + 13;  
4. p = &j;  
5. *p = i + 1;  
6. *p = 15;  
7. p = p2;  
8. *p = i;  
9. *p = 2*(*p2);
```

adres i = 0025F984	
adres j = 0025F978	
1. p2 = 0025F978	
2. p = 0025F984	
3. *p = 43	
4. p = 0025F978	
5. *p = 44	
6. *p = 15	
7. p = 0025F978	
8. *p = 43	
9. *p = 86	



Verwisselen

Tweede poging



- Schrijf een **functie** waarmee twee **int** variabelen verwisseld kunnen worden.

```
void wissel(int *p, int *q) {  
    int hulpje = *p;  
    *p = *q;  
    *q = hulpje;  
}
```

```
int main(void) {  
    int x = 7, y = 8;  
    printf("x = %d en y = %d\n", x, y);  
    wissel(&x, &y); call by reference  
    printf("x = %d en y = %d\n", x, y);  
    getchar(); return 0;  
}
```

Wel goed!

Kan het nog op een
andere manier?





Samenvatting pointers

- Een pointer is een variabele
- Een pointer heeft een **type** (`int`, `float`, etc.)
- Een pointer heeft een **inhoud** namelijk het **adres** van een andere variabele
- Een pointer “**wijst**” naar een variabele
- De **inhoud** van de variabele waar de pointer *naar wijst*, kan worden benaderd met de **dereferentie operator** *

dereferentie operator *

Huiswerk



- Bestudeer de werking van de functie `wissel`. Teken de variabelen `x`, `y`, `p`, `q` en hulpje en bepaal hun inhoud gedurende de uitvoering van het programma.
- Bestudeer C boek:
 - paragrafen 6.2 en 6.3.

Oude tentamen opgave



- Geef de uitvoer van het programma

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

```
void ikgalekker(int *a, int b, int *c, int d){  
    int temp = *c;  
    *c = *a;  
    b = temp;  
    d = b;  
}
```

```
void main(void){  
    int w = 5, x = 1, y = 2, z = 3;
```

```
    z = w%y;  
    w += z;  
  
    ikgalekker(&x, y, &z, w);
```

```
    printf("%d %d %d %d", w, x, y, z);  
    getchar();
```

```
}
```

Zelfstudie!

Maak de opgave op
papier (net als op het
tentamen)

Antwoorden...



Oude tentamen opgave



- Geef de uitvoer van het programma

```
#include <stdio.h>

void gajelekker(int *a, int b, int *c){
    int yuup = *a;
    *c = b;
    *a = *a * *c;
}

void main(void){
    int w = 7, x = 5, y = 3, z = 4;

    z -= x % y;
    w += z;

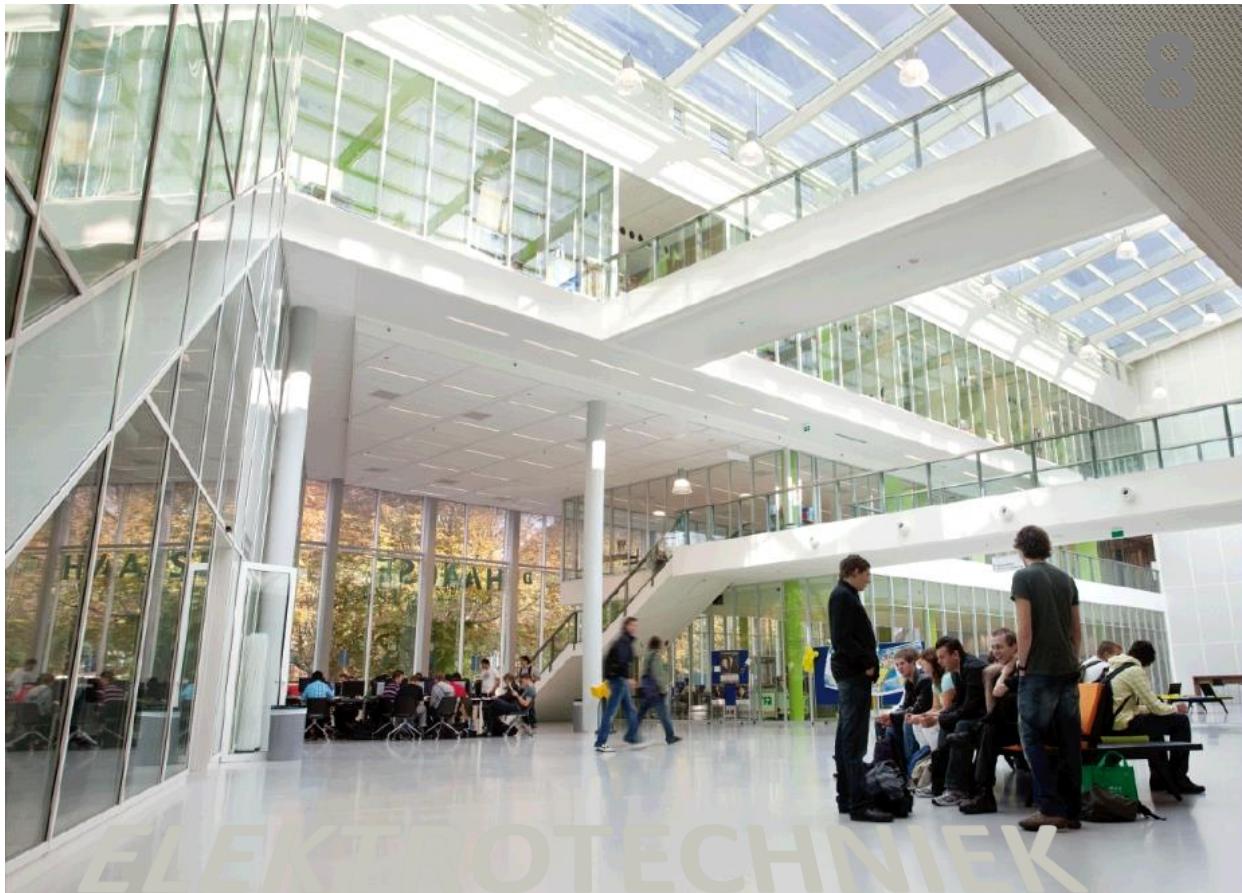
    gajelekker(&x, y, &x);

    printf("%d %d %d %d", z, x, y, w);
    getchar();
}
```

Zelfstudie!

Maak de opgave op papier (net als op het tentamen)





Gestructureerd programmeren in C

Arrays



Array

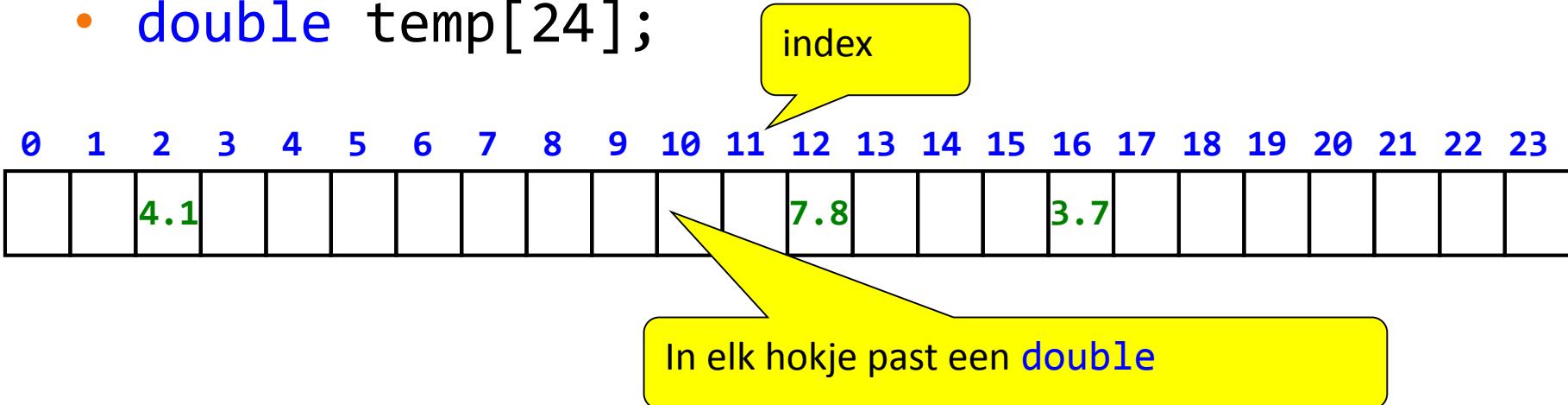
- Meerdere variabelen van **hetzelfde type** kun je samennemen in één **array** variabele.
- Stel in een embedded systeem wordt ieder uur de temperatuur gemeten en opgeslagen.
 - Aparte variabele voor elk uur:
`double temp0, temp1, temp2, temp3, temp4, temp5,
temp6, temp7, temp8, temp9, temp10, temp11, temp12,
temp13, temp14, temp15, temp16, temp17, temp18,
temp19, temp20, temp21, temp22, temp23;`
 - Array variabele:
`double temp[24];`

Voordeel?



Array

- `double temp[24];`

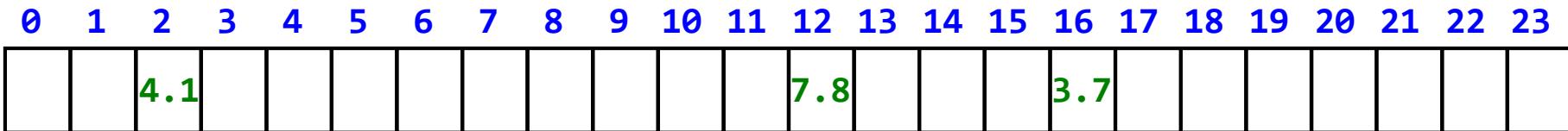


- Index begint bij `0`.
- Element kan geselecteerd worden met index operator `[]`
 - `temp[2] = 4.1;`
 - `temp[16] = 3.7;`
 - `temp[12] = temp[2] + temp[16];`

Random access



- Elk array element kan **even snel** bereikt worden.



- Executietijd van `temp[i]` is **onafhankelijk** van de waarde van `i`.
- Hoe kan dat?
 - adres van element `i` \leftrightarrow beginadres van de array + `i * aantal bytes per hokje`
 - `&temp[i]` \leftrightarrow `&temp[0] + i * sizeof temp[0]`



Voorbeeld array

```
#include <stdio.h>

int main(void) {
    double temp[24], totaal;
    int i;
    for (i = 0; i < 24; i = i + 1) {
        printf("Geef temperatuur om %d uur: ", i);
        scanf("%lf", &temp[i]);
    }
    totaal = 0.0;
    for (i = 0; i < 24; i = i + 1) {
        totaal = totaal + temp[i];
    }
    printf("Gem. temp. is %f graden.", totaal / 24);
    fflush(stdin);
    getchar();
    return 0;
}
```





Initialiseren

```
int i = 8;
```

```
int rij[] = { 23, 4, 2, 5 };
```

Aantal elementen hoeft niet opgegeven te worden (wordt in dit geval 4).

```
int rij[10] = { 23, 4, 2, 5 };
```

Aantal elementen mag wel opgegeven te worden
Elementen 4 t/m 9 worden in dit geval met 0 geïnitialiseerd.

```
int rij[10] = { 0 };
```

Hoe wordt de array gevuld?

```
int rij[] = { 0 };
```

En nu?



Initialiseren: a closer look



```
int rij[] = { 23, 4, 2, 5 };
```

Dit is een variabele

Dan is dit een ... ?

We kennen een toe aan een ... ?

```
char tekens[] = { 23, 4, 2, 5 };
```

```
char tekens[] = "23,4,2,5";
```

Wat is het verschil?

Rij met getallen

Wat is dit dan?

Één string (tekst)

Strings worden later behandeld

Rij met symbolen



Array en functies

- Array als **returnwaarde**. Niet mogelijk in C!
- Array als **parameter**. Altijd **call by reference**.
 - Beginadres van de array wordt doorgegeven.

Waarom call by reference?



Array functie parameter

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

Beginadres van de array wordt doorgegeven!

```
void leesin(double a[], int n) {  
    int i;  
    for (i = 0; i < n; i = i + 1) {  
        printf("Geef element %d: ", i);  
        scanf("%lf", &a[i]);  
    }
```

Aantal elementen moet als aparte parameter doorgegeven worden!

```
int main(void) {  
    double d[5];  
    leesin(d, 5);  
}
```

...



Array functie parameter

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

```
void leesin(double a[], int n) {  
    int i;  
    for (i = 0; i < n; i = i + 1) {  
        printf("Geef element %d: ", i);  
        scanf("%lf", &a[i]);  
    }
```

```
}
```

```
int main(void) {  
    double d[5];  
    leesin(&d[0], 5);  
}
```

```
...
```

Alternatieve manier om argument door te geven



Array functie parameter

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

Alternatieve manier om
parameter te declareren

```
void leesin(double *a, int n) {  
    int i;  
    for (i = 0; i < n; i = i + 1) {  
        printf("Geef element %d: ", i);  
        scanf("%lf", &a[i]);
```

Hier gebruik je de
parameter als een array

```
}
```

```
}
```

```
int main(void) {  
    double d[5];  
    leesin(&d[0], 5);
```

```
...
```

Alternatieve manier om
argument door te geven



Array functie parameter

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

Alternatieve manier om
parameter te declareren

```
void leesin(double *a, int n) {  
    int i;  
    for (i = 0; i < n; i = i + 1) {  
        printf("Geef element %d: ", i);  
        scanf("%lf", &a[i]);
```

```
}
```

```
int main(void) {  
    double d[5];  
    leesin(d, 5);
```

```
...
```

Array functie parameter



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

```
void leesin(double *a, int n) {  
    int i;  
    for (i = 0; i < n; i = i + 1)  
        printf("Geef element %d: ", i);  
        scanf("%lf", a + i);  
}
```

a + i

Hoe weet het programma waar het
in het geheugen moet zijn?

```
int main(void) {  
    double d[5];  
    leesin(d, 5);  
}
```

Alternatieve manier om adres van
element te specificeren

&a[i] \leftrightarrow a + i
a[i] \leftrightarrow *(a + i)

...



Array functie parameter

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

```
void leesin(double a[], int n) {
    int i;
    for (i = 0; i < n; i = i + 1) {
        printf("Geef element %d: ", i);
        scanf("%lf", &a[i]);
    }
}

int main(void) {
    double d[5];
    leesin(d, sizeof d / sizeof d[0]);
}
```

Alternatieve manier om aantal elementen door te geven

Voordeel?

...

Idee! WERKT NIET



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

```
void leesin(double a[]) {  
    int i;  
    for (i = 0; i < sizeof a / sizeof a[0]; i = i + 1) {  
        printf("Geef element %d: ", i);  
        scanf("%lf", &a[i]);  
    }  
}  
  
int main(void) {  
    double d[5];  
    leesin(d);  
    ...
```

Snap je waarom?

Er wordt niets ingelezen!

`sizeof a` = size of adres!





Array functie parameter

...

```
double gem(double a[], int n) {  
    int i;  
    double totaal = 0.0;  
    for (i = 0; i < n; i = i + 1) {  
        totaal = totaal + a[i];  
    }  
    if (n > 0) return totaal / n;  
    else return 0;  
}
```

Wat is onverstandig aan deze code (de syntax)?

```
int main(void) {  
    double d[5];  
    size_t s = sizeof d / sizeof d[0];  
    leesin(d, s);  
    printf("Het gemiddelde is %f", gem(d, s));  
}
```

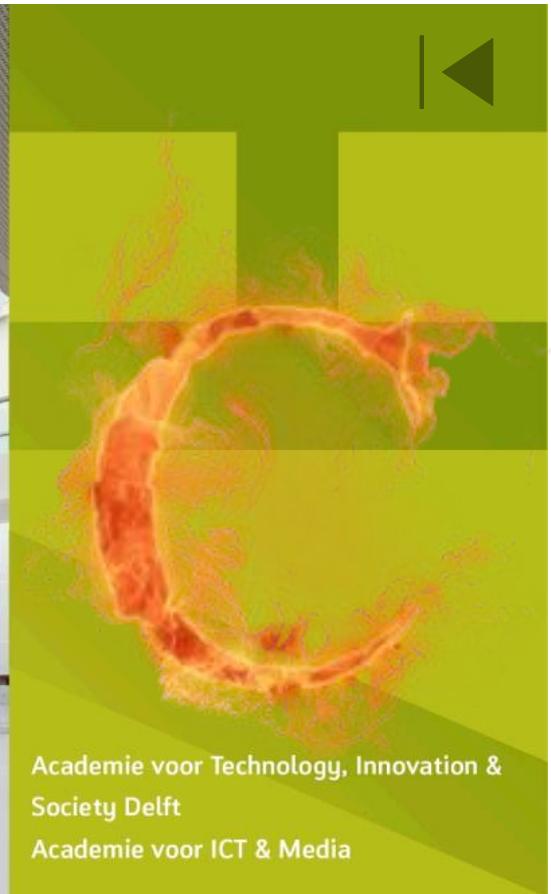
sizeof operator geeft getal van **type size_t**

Huiswerk



- Schrijf een functie met de naam `reverse` waarmee de inhoud van een rij gehele getallen omgedraaid kan worden. Als de rij de getallen 0, 1, 2 en 3 bevat dan moet de rij na afloop van de functie de getallen 3, 2, 1 en 0 bevatten.
- Bestudeer C boek:
 - paragraaf 3.7.
 - paragrafen 6.1, 6.4 en 6.6.
- Maak opdrachten:
 - 8 van paragraaf 6.18.





Gestructureerd programmeren in C

Tekst en bestanden



Karakters

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

Bibliotheek met karakter functies
<http://en.wikipedia.org/wiki/Ctype.h>

```
int main(void) {
    char c; —————— Karakter variabele
    printf("Geef een karakter: ");
    scanf("%c", &c); —————— Inlezen karakter variabele
    if (isalpha(c)) {
        printf("Dat is een letter.\n");
        if (islower(c)) {
            printf("De bijbehorende hoofdletter is: %c\n",
                   toupper(c));
        }
    }
    else {
        printf("Dat is geen letter.\n");
    }
    fflush(stdin); getchar(); return 0;
}
```

Geef een karakter: h
Dat is een letter.
De bijbehorende hoofdletter is: H

Printen karakter variabele
toupper(c));



C strings

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

Bibliotheek met string functies
<http://en.wikipedia.org/wiki/String.h>

```
int main(void) {
    char s[] = "Hallo";
    char *p;
    printf("%s\n", s);
    printf("sizeof(s) = %d\n", sizeof(s));
    printf("strlen(s) = %d\n", strlen(s));
    p = strchr(s, 'a');
    if (p != NULL) {
        *p = 'e';
    }
    printf("%s\n", s);
    fflush(stdin);
    getchar();
    return 0;
}
```

s is een string variabele

Afdrukken string variabele

string variabele wordt afgesloten met het karakter '\0'

Handig als je wilt testen of je bij het einde van een string bent.

```
Haloo
sizeof(s) = 6
strlen(s) = 5
Hello
```



Tekstfiles

- Een tekstfile bevat in **ASCII** gecodeerde data (**char's**).
- Tekstfiles kunnen **bewerkt** worden (b.v. met Notepad.) 
- Tekstfiles kunnen ook met een **C programma aangemaakt, beschreven en uitgelezen** worden.

Tekstfile maken in C



```
#include <stdio.h>
int main(void) {
    FILE * outfile;
    outfile = fopen("output.txt", "w");
    if (outfile == NULL) {
        printf("File output.txt kan niet aangemaakt worden.\n");
    }
    else {
        int i;
        for (i = 0; i < 10; i = i + 1) {
            fprintf(outfile, "Het kwadraat van %d is %d.\n", i, i * i);
        }
        fclose(outfile);
        printf("File output.txt is aangemaakt.\n");
    }
    getchar();
    return 0;
}
```

Een FILE* verwijst naar een bestand (file).

fopen opent een bestand.

"w" (write) opent /maakt het bestand voor (over)schrijven.

fopen geeft NULL terug als openen niet gelukt is.

fprintf schrijft in een file.

fclose sluit een FILE* .

Andere opties

The screenshot shows a Windows Notepad window with the title 'output.txt - Kladblok'. The window contains the following text:

```
Het kwadraat van 0 is 0.
Het kwadraat van 1 is 1.
Het kwadraat van 2 is 4.
Het kwadraat van 3 is 9.
Het kwadraat van 4 is 16.
Het kwadraat van 5 is 25.
Het kwadraat van 6 is 36.
Het kwadraat van 7 is 49.
Het kwadraat van 8 is 64.
Het kwadraat van 9 is 81.
```



Tekstfile lezen in C



```
#include <stdio.h>

int main(void) {
    char naam[80];
    int punten;
    FILE* infile;
    infile = fopen("stand.txt", "r");
    if (infile == NULL) {
        printf("File stand.txt kan niet gelezen worden.\n");
    } else {
        fscanf(infile, "%79s%d", naam, &punten);
        while (fscanf(infile, "%79s%d", naam, &punten) == 2) {
            printf("Speler %s heeft %d punten.\n", naam, punten);
        }
        fclose(infile);
    }
    getchar();
    return 0;
}
```

"r" opent de file voor lezen (read).

fscanf leest uit een file.

fscanf geeft aantal correct ingelezen variabelen terug.

%79s leest maximaal 79 karakters in.

Waarom geven we hier geen adres op?





Huiswerk char en char[]

- Bestudeer C boek:
 - paragraaf 3.3.
 - paragraaf 8.6.
 - paragrafen 6.9 en 6.10.
- Maak opdracht:
 - 14 van paragraaf 6.18.
- Bekijk:
 - <http://en.wikipedia.org/wiki/Ctype.h>
 - <http://en.wikipedia.org/wiki/String.h>



6



Gestructureerd programmeren in C

Scope, lifetime, recursiviteit



Scope versus Lifetime

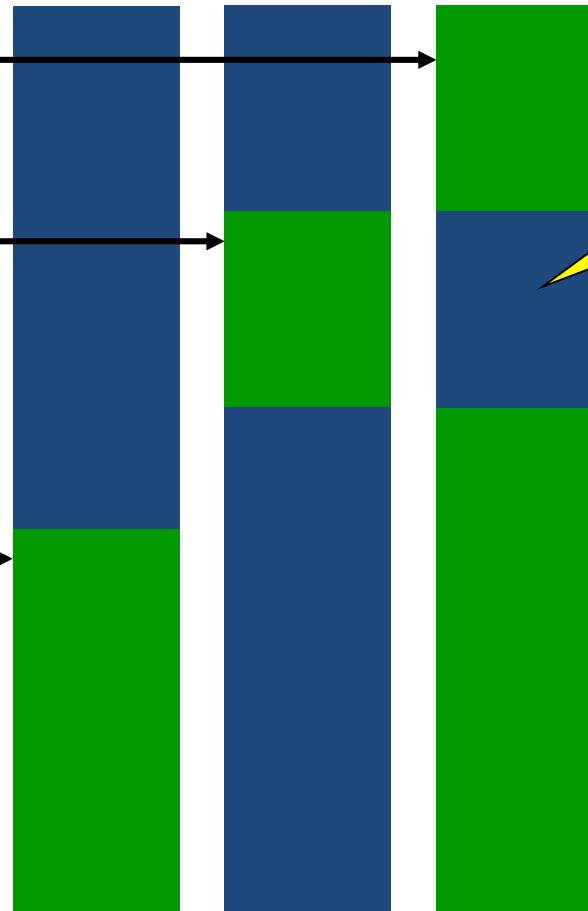
- Elke variabele heeft een:
 - **Scope** (bereik) → waar kun je de variabele gebruiken (tijdens het **coderen** van het programma)
 - **Lifetime** (levensduur) → wanneer bestaat de variabele (tijdens het **uitvoeren** van het programma)

Variabele	Scope	Lifetime
Globaal	Hele programma (kan verborgen zijn)	Bestaat alleen als het programma wordt uitgevoerd
Lokaal	Functie	Bestaat alleen als de functie wordt uitgevoerd



Scope

```
int a; _____  
  
void func(void) {  
    int a, b; _____  
    ...  
}  
  
int main(void) {  
    int b; _____  
    ...  
    func();  
    ...  
    return 0;  
}
```



In scope

Out of scope

Lifetime



```
int a;                                Reserveer geheugenruimte voor globale a  
  
void func(void) {  
    int a, b;                            Reserveer geheugenruimte  
                                         voor lokale a en b in func  
}  
...  
                                         Geef geheugenruimte vrij  
                                         van lokale a en b in func  
  
int main(void) {  
    int b;                                Reserveer geheugenruimte  
                                         voor lokale b in main  
    ...  
    func();  
    ...  
    return 0;                            Geef geheugenruimte  
                                         vrij van lokale b in main  
}  
                                         Geef geheugenruimte vrij voor globale a
```

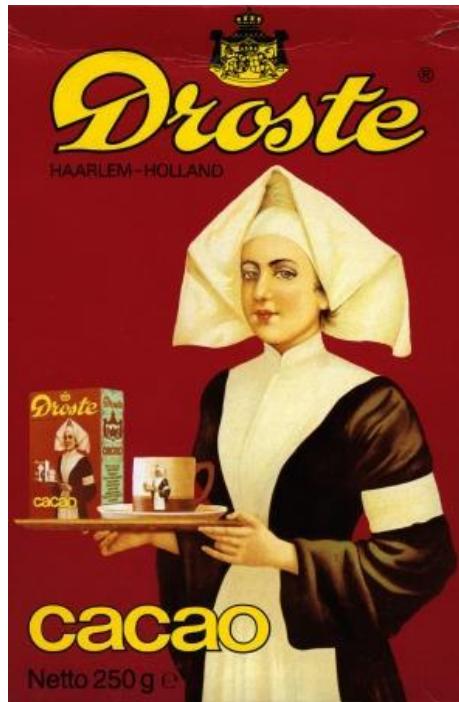
Recursieve functies



- Een recursieve functie roept zichzelf aan
 - In de functie definitie staat een **functie-aanroep** naar zichzelf.



Inception



Recurсive functies



- Een recursieve functie roept zichzelf aan
 - In de functie definitie staat een **functie-aanroep** naar zichzelf.
- Heeft **altijd** een stopconditie
- Roept zichzelf een **x aantal** keren aan.
- In een recursieve functie zitten (bijna) nooit, **herhalingslussen** (**while**, **do-while**, **for**).

Waarom?

Waarom is dat
(bijna) nooit
nodig?



Faculteit



- Bereken $n! = 1 * 2 * 3 * \dots * n$

```
int faculteit(int n) {  
    int i, res = 1;  
    for (i = 1; i <= n; i = i + 1) {  
        res = res * i;  
    }  
    return res;  
}
```

Iteratieve implementatie

$$n! = \begin{cases} 1 \text{ als } n = 0 \\ n * (n - 1)! \text{ als } n > 0 \end{cases}$$

Recursieve implementatie

```
int faculteit(int n) {  
    if (n > 0)  
        return n * faculteit(n - 1);  
    return 1;  
}
```

Welke is beter ?

Recursieve aanroep (voorbeeld)



```
int j = 24;
```

```
    ↓   ↑  
return 24;  
    ↓   ↑  
    ↓   ↑  
return 6;
```

```
    ↓   ↑  
    ↓   ↑
```

```
return 2;
```

```
    ↓   ↑
```

```
return 1;
```

```
    ↓   ↑
```

```
return 1;
```

```
int faculteit(int n) {  
    if (n > 0)  
        return n * faculteit(n - 1);  
    return 1;  
}
```

n boven k



- Bereken n boven k

Wat is beter ?

$$- \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Iteratieve implementatie

```
int n_boven_k(int n, int k) {
    return faculteit(n) / (faculteit(k) * faculteit(n - k));
}
```

- n boven k

$$- \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Recursieve implementatie

```
int n_boven_k(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    return n_boven_k(n - 1, k - 1) + n_boven_k(n - 1, k);
}
```

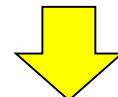


Recursieve aanroep (voorbeeld)

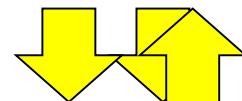


```
int n_boven_k(int n, int k) {  
    if (k == 0 || k == n) return 1;  
    return n_boven_k(n - 1, k - 1) + n_boven_k(n - 1, k);  
}
```

```
int j = n_boven_k(4, 2);
```



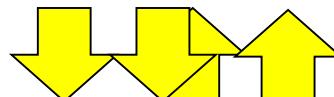
```
return 3 + n_boven_k(3, 2);
```



```
return 1 + n_boven_k(2, 1); n_boven_k(2, 1);
```



```
return 1 + return 1 + 1; 1); n_boven_k(1, 1);
```



```
ret return 1;
```

Domein



- Faculteit
 - Domein = [0..12]
- **n** boven k
 - Domein versie met faculteit = [0..12]
 - Domein recursieve versie = [0..33]

In dit geval is de recursieve versie dus “beter”.



Huiswerk



- Schrijf een **recursieve** functie `int fib(int n)` die het n^{de} getal uit de fibonacci rij berekent. Zie:
http://en.wikipedia.org/wiki/Fibonacci_number
Paragraaf 4.13 geeft een iteratieve oplossing.
- Bestudeer C boek:
 - paragrafen 5.6, 5.7 en 5.15.





Gestructureerd programmeren in C

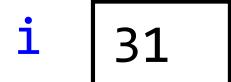
2 dimensionale array

2 dimensionale array

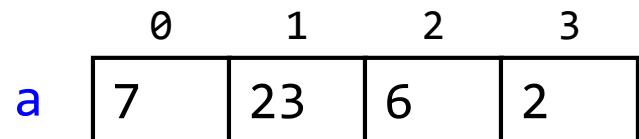


- Array van array's

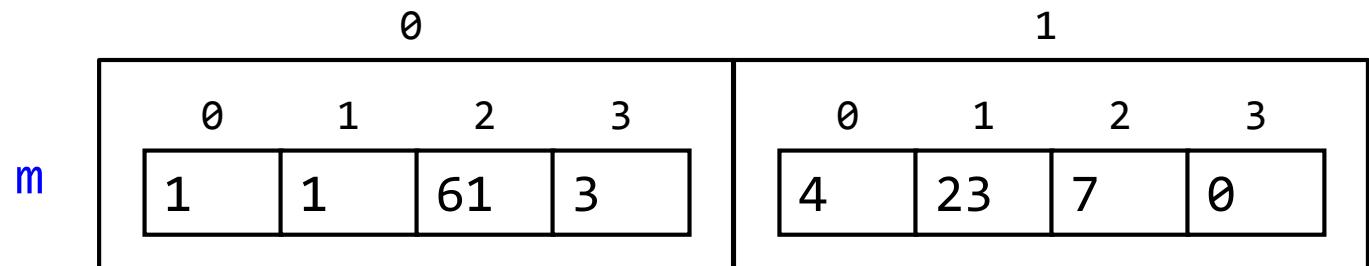
```
int i = 31;
```



```
int a[4] = {7, 23, 6, 2};
```



```
int m[2][4] = { {1, 1, 61, 3}, {4, 23, 7, 0} };
```





2 dimensionale array

- Array van array's

```
int m[2][4] = { {1, 1, 61, 3}, {4, 23, 7, 0} };
```

	0		1	
m	0	1	2	3
	1	1	61	3
	18	23	7	0

m	0	1	2	3
0	1	1	61	3
1	18	23	7	0

$m[1][0] = 18;$

Row-major order

Sudoku



8	6			2				
			7			5	9	
			6		8			
4								
	5	3				7		
2				6				
	7	5	9					

```
int puzzel[9][9] = {  
    {8 ,6 ,0 ,0 ,2 ,0 ,0 ,0 ,0 },  
    {0 ,0 ,0 ,7 ,0 ,0 ,0 ,5 ,9},  
    {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0},  
    {0 ,0 ,0 ,0 ,6 ,0 ,8 ,0 ,0},  
    {0 ,4 ,0 ,0 ,0 ,0 ,0 ,0 ,0},  
    {0 ,0 ,5 ,3 ,0 ,0 ,0 ,0 ,7},  
    {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0},  
    {0 ,2 ,0 ,0 ,0 ,0 ,6 ,0 ,0},  
    {0 ,0 ,7 ,5 ,0 ,9 ,0 ,0 ,0} };
```

Huiswerk



- Schrijf een functie `areRowsValid` die controleert of in een matrix van 9 bij 9 niet meerdere malen hetzelfde cijfer op dezelfde **regel** voorkomt. Een lege plaats wordt aangegeven met de waarde 0.
 - De functie geeft **1** terug als er niet meerdere malen hetzelfde cijfer op dezelfde rij voorkomt.
 - De functie geeft **0** terug als er meerdere malen hetzelfde cijfer op dezelfde rij voorkomt.
- Bestudeer C boek:
 - paragraaf 6.11.

**Nodig voor het
practicum**

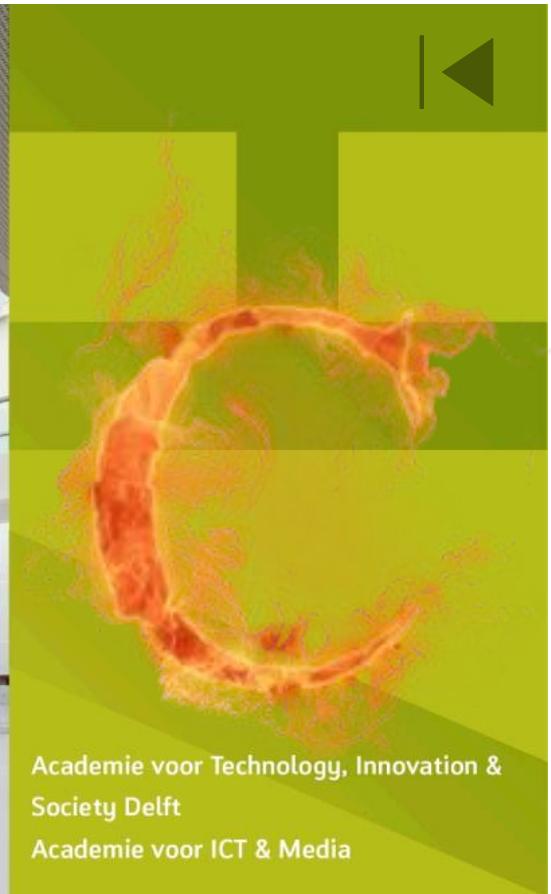


Huiswerk tekstfiles



- Schrijf een functie `readPuzzle` die een Sudoku puzzel kan inlezen uit een bestand. Een lege plaats wordt aangegeven met de waarde 0.
- Schrijf een functie `savePuzzle` die een Sudoku puzzel kan wegschrijven naar een bestand.
- Bestudeer C boek:
 - paragrafen 11.3 en 11.4.

**Nodig voor
het
practicum**



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

Gestructureerd programmeren in C

Sudoku

Sudoku oplossing huiswerk:



```
int areRowsValid(int m[][][9]) {  
    int r;  
    for (r = 0; r < 9; r = r + 1) {  
        int c, checklist[9] = {0};  
        for (c = 0; c < 9; c = c + 1) {  
            int digit = m[r][c];  
            if (digit != 0) {  
                if (checklist[digit - 1] == 0) {  
                    checklist[digit - 1] = 1;  
                }  
                else {  
                    return 0;  
                }  
            }  
        }  
    }  
    return 1;  
}
```

Waarom is het **noodzakelijk** om alle dimensies behalve de eerste op te geven?

Deze variabelen zijn **lokaal** gedeclareerd. Zie je waarom?

Sudoku



```
int isValid(int m[][][9]) {  
    return areRowsValid(m) && areColumnsValid(m) &&  
           areBlocksValid(m);  
}
```

De functies `areColumnsValid` en `areBlocksValid` gaan we bij het practicum maken.



Where are the X's and O's
supposed to go in this sudoku?

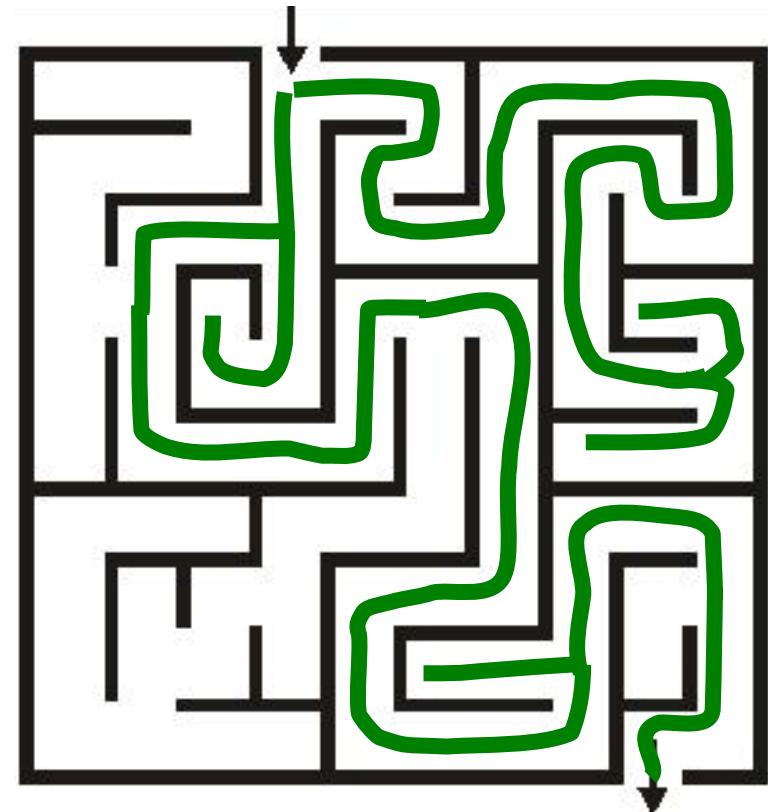


Sudoku solve

- Algoritme: Backtracking

- Voorbeeld: Doolhof

- Kies bij elke splitsing de weg die het meest aan je linkse hand ligt (die je nog niet hebt onderzocht)
- **Keer terug** als je vastloopt naar de laatste splitsing.
- [Klik voor een Sudoku variant hier.](#)



Sudoku backtracking (recursief)



- Zoek het eerste lege vakje. Vul een **1** in en als de puzzel valid is los de puzzel dan op (recursie).
 - Valid en oplossen gelukt → return gelukt.
 - Invalid of oplossen niet gelukt → Vul een **2** in als de puzzel nog valid is los de puzzel dan op (recursie).
 - Valid en oplossen gelukt → return gelukt.
 - Invalid of oplossen niet gelukt → Vul een **3** in en als de puzzel nog valid is los de puzzel dan op (recursie)
 - ...

Als bij invullen van **9** de puzzel invalid is of als het oplossen (recursief) niet is gelukt → return niet gelukt.



Sudoku solve

```
int solve(int m[][]){  
    int r, c, digit;  
    for (r = 0; r < 9; r = r + 1) {  
        for (c = 0; c < 9; c = c + 1) {  
            if (m[r][c] == 0) {  
                for (digit = 1; digit <= 9; digit = digit + 1) {  
                    m[r][c] = digit;  
                    if (isValid(m) && solve(m) == 1) {  
                        return 1;  
                    }  
                    m[r][c] = 0;  
                }  
                return 0;  
            }  
        }  
    }  
    return 1;  
}
```

Probleem?

Sudoku probleem



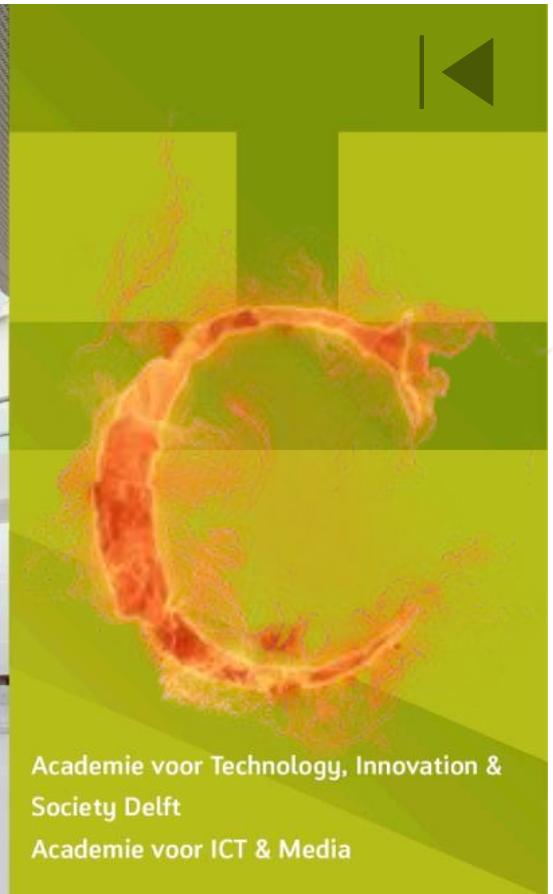
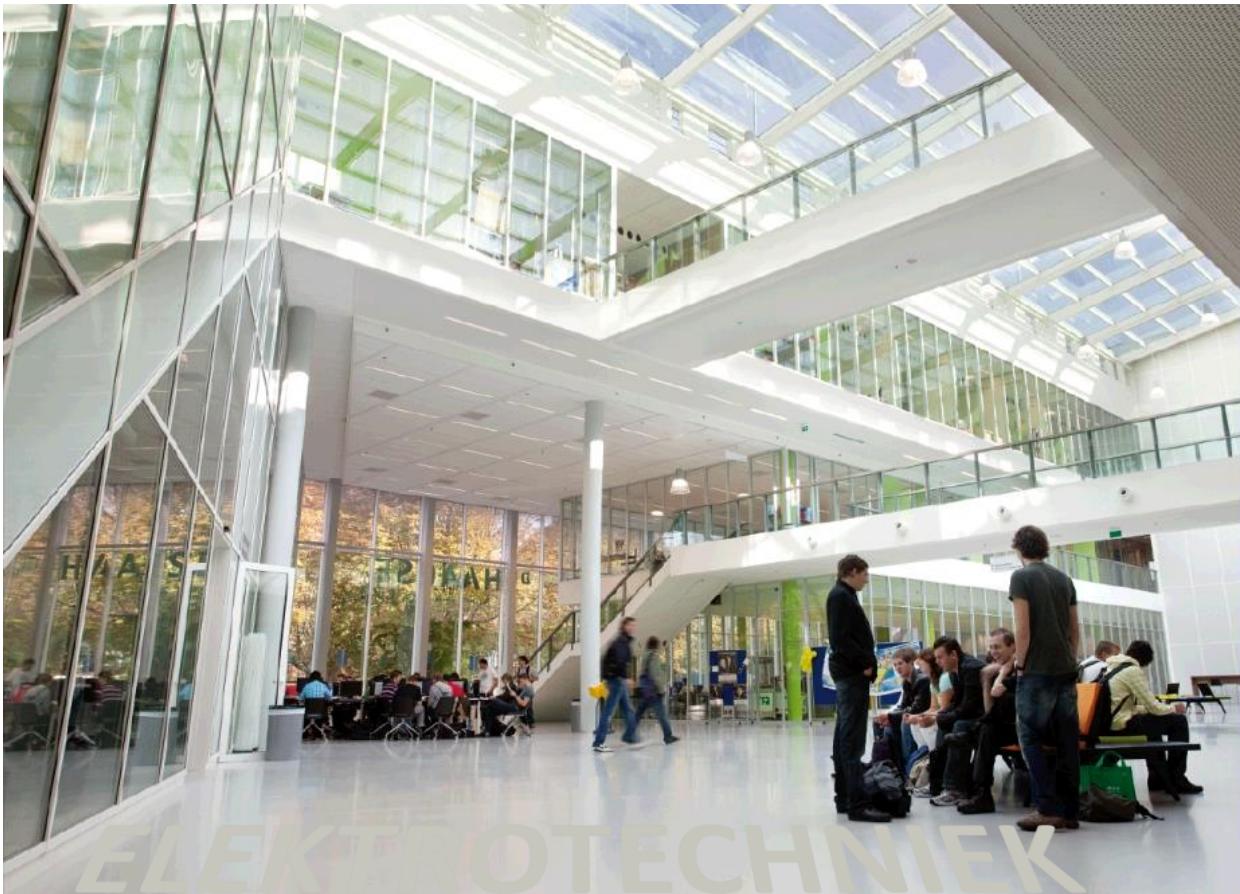
- De gepresenteerde functie kan **elke** “valid” Sudoku oplossen (zelfs een lege).
- Een Sudoku mag maar **één** oplossing hebben.
 - Hoe kun je dat controleren?

Zie laatste practicumopdracht.

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0



1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 1 4 3 6 5 8 9 7
3 6 5 8 9 7 2 1 4
8 9 7 2 1 4 3 6 5
5 3 1 6 4 2 9 7 8
6 4 2 9 7 8 5 3 1
9 7 8 5 3 1 6 4 2



Gestructureerd programmeren in C

Verkorte notatie



C verkorte notatie

- Alle **rekenoperatoren** kunnen **gecombineerd** worden met **operator=** als resultaat in de linker operand moet worden opgeslagen.

a = a + b; → **a += b;**

x = x * y; → **x *= y;**

Deze assignment operatoren worden van rechts naar links geëvalueerd. Zie prioriteitentabel.

```
int a = 6, b = 7, c = 8;  
a += b += c;  
printf("a=%d b=%d c=%d\n", a, b, c);  
(a += b) += c;  
printf("a=%d b=%d c=%d\n", a, b, c);
```

C prioriteitentabel (hoog naar laag)



Operator	Description	Associativity
()	Parentheses (function call) (see Note 1)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right

Snap je wat de
'prioriteit' van een
operator betekent?



Operator	Description	Associativity
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= =	Bitwise exclusive/inclusive OR assignment	
<=>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right



C verkorte notatie

- Increment en decrement operatoren.

$a = a + 1;$ → **$++a;$** of **$a++;$**

$b = b - 1;$ → **$--b;$** of **$b--;$**

Prefix operator

Postfix operator

- Als een increment of decrement operator in een expressie wordt gecombineerd met meerdere operatoren dan wordt de prefix **vooraf** en de postfix **achteraf** uitgevoerd.



Postfix & Prefix

- Prefix wordt vooraf uitgevoerd en een postfix achteraf. Maar vooraf/achteraan aan wat?

Code:

```
#include <stdio.h>
int a = 6, b, c;

void tmp(){
    b = a++;
    c = ++a;
    printf("a=%d b=%d c=%d\n", a, b, c);
    getchar();
}

void main() {
    tmp();
    tmp();
    getchar();
}
```

Uitvoer:

a=8 b=6 c=8

a=10 b=8 c=10



C verkorte notatie

- Conditionele operator **? :**

```
int abs(int i) {      →      int abs(int i) {  
    if (i >= 0) {  
        return i;  
    }  
    else {  
        return -i;  
    }  
}
```

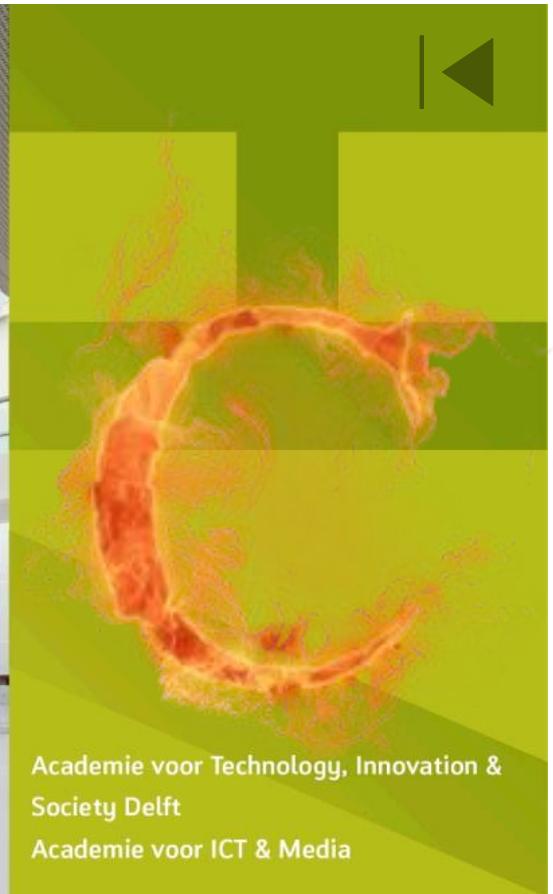
```
                    }  
        return i >= 0 ? i : -i;
```

Huiswerk



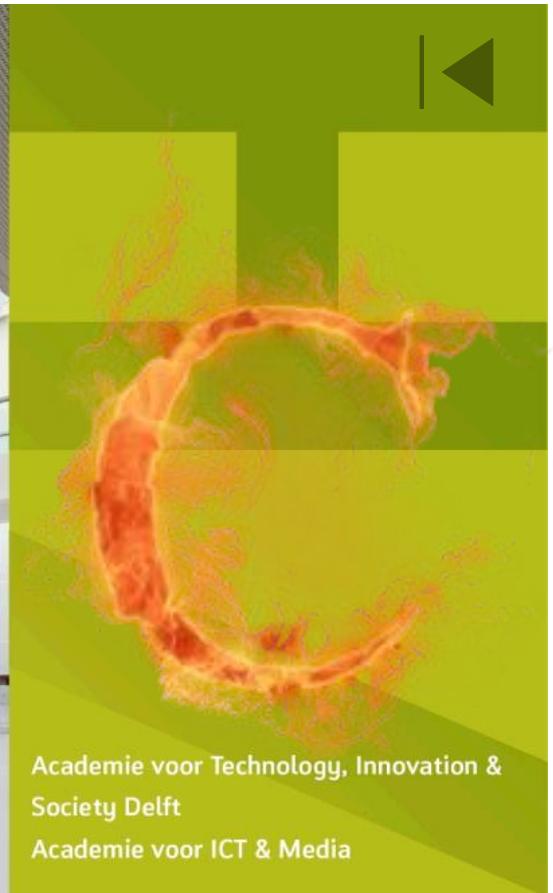
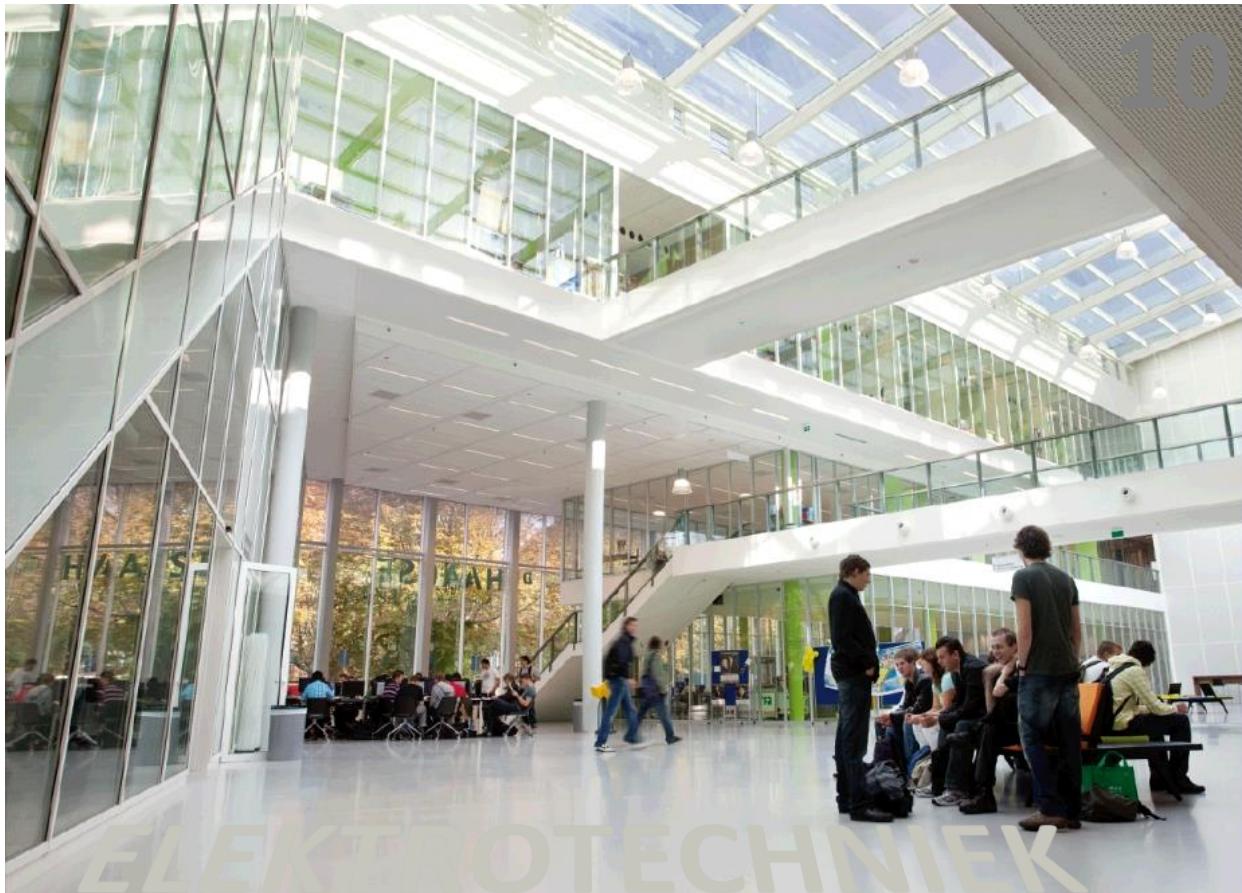
- Bestudeer C boek:
 - paragrafen 2.10 t/m 2.12.
 - paragraaf 4.17.
- Schrijf een zo **kort** mogelijke implementatie van de onderstaande functie:

```
void reverse(int a[], int n) {
    int first = 0, last = n - 1;
    while (first < last) {
        wissel(&a[first], &a[last]);
        first = first + 1;
        last = last - 1;
    }
}
```



Gestructureerd programmeren in C

Vanaf dit punt volgen geavanceerdere onderwerpen



Gestructureerd programmeren in C

Structs

Een nieuwe variabele: struct



- Variabele met een ‘vaste’ statische **struct**urering
- Een **struct** bestaat bevat één of (meestal) meerdere variabelen (members)
- De members kunnen van verschillende typen zijn
- De opbouw van een struct:

```
struct naam {  
    members  
};  
  
struct meting{  
    int uur, min, sec;  
    double temp;  
};
```

Member



- De struct variabele bevat andere variabelen die verschillend in **type** kunnen zijn.
- Hoe selecteren we een member van een struct?
 - Met een **selectie operator** en de **membername**



De selectie operator .

- In C zijn er twee selectie operatoren:
 - We gebruiken voor structs vaak de ‘**direct member selector**’.
 - Deze heeft de vorm:
 - **Naam_struct.Identifier**
 - Er is nog een andere operator die gebruikt wordt i.c.m. structs en pointers
 - De ‘**indirect member selector**’ operator **->**
 - Deze heeft de vorm:
 - **Naam_pointer->Identifier**

struct in C



- `struct {
 int uur, min, sec;
 double temp;
} meting;`

	uur	min	sec	temp
meting	14			13.7

- `meting.temp = 13.7;`
- `meting.uur = 14;`

struct in C



- Hoe gebruik je een struct?

Code:

```
int main(void) {  
  
    //definieren struct  
    struct student {  
        int nummer;  
        char naam[20];  
        float cijfer;  
    };  
  
    //declareren struct  
    struct student s1, s2;  
    //werken met struct  
    s1.nummer = 13;  
}
```



struct in C

- Hoe gebruik je een struct?
 - Alternatieve versie

Code:

```
int main(void) {  
  
    //definieren en declareren structs  
    struct student {  
        int nummer;  
        char naam[20];  
        float cijfer;  
    } s1, s2;  
  
    //werken met struct  
    s1.nummer = 13;  
}
```

[Kijk voor meer info op BB!](#)

struct in een Array



Code:

```
int main(void) {  
  
    //definieren en declareren structs  
    struct student {  
        int nummer;  
        char naam[20];  
        float cijfer;  
    } s[100];  
  
    //werken met struct  
    s[0].nummer = 13;  
}
```

Voorbeeld

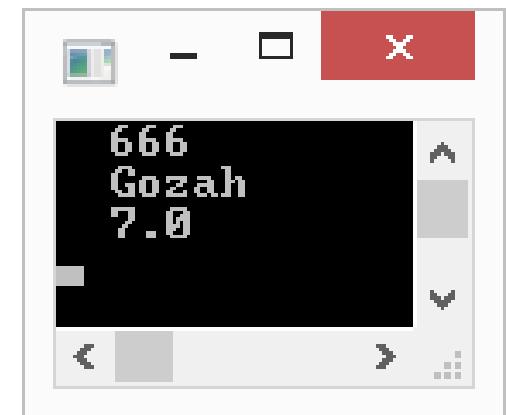


```
#include <stdio.h>

struct Student {
    int nummer;
    char naam[20];
    float cijfer;
} s1 = { 666, "Gozah", 6.0 };

void printStudent() {
    printf("%5d\n  %-20s \n %4.1f\n", s1.nummer, s1.naam, s1.cijfer);
}

void geefCijfer(struct Student *ps, float c) {
    if (c > ps->cijfer) {
        (*ps).cijfer = c;
    }
}
```





Opmerking

- De struct wordt bijna altijd gebruikt in combinatie met een **typedef**
- Dat maakt het gebruik van de struct veel gemakkelijker

[Kijk voor meer info op BB!](#)

Huiswerk



- Bestudeer
 - <http://bd.eduweb.hhs.nl/micprg/structs.htm>
- Bestudeer C boek:
 - paragraaf 9.2 t\m 9.6



Tentamenvraag

- In het tekstbestand 'namentoonladders.txt' staat het volgende:

Maak de opgave op papier (net als op het tentamen)

Blues
Major
Dorian
Phrygian
Lydian
Dominant
Natural

Zelfstudie!

- Ontwerp een struct array (definitie) zodat voor de opgegeven toonladders zowel de naam van de toonladder wordt opgeslagen als een index nummer. Dus de blues-toonladder heeft index nummer 0 en de dorian-toonladder heeft index nummer 2



Gestructureerd programmeren in C

Typedef



typedef

- Met een **typedef** kunnen we een synoniem van een variabele maken.
- De gebruiker kan zijn eigen ‘datatype’ creëren.
- De file pointer (**FILE***) is een datatype die is aangemaakt met een **typedef** en een **struct**.



typedef

Code:

```
#include <stdio.h>

int main(void) {
    typedef int stinksok;

    stinksok sok1 = 2, sok2 = 3;
    printf("sok1 = %d\nsok2 = %d",sok1,sok2);

    getchar();
    return 0;
}
```

Uitvoer:

```
sok1 = 2
sok2 = 3
```



typedef

- De **definitie** van een variabele kan worden gescheiden van de **declaratie**:

```
#include <stdio.h>

int main(void) {
    typedef int stinksok; //definitie

    stinksok sok1 = 2, sok2 = 3; //declaratie
    printf("sok1 = %d\nsok2 = %d",sok1,sok2);

    getchar();
    return 0;
}
```



Voorbeeld

```
#include <stdio.h>

int main(void){
    typedef int Vector[10];
    typedef Vector Matrix[10];

    Vector vector = { 0 };
    vector[1] = 2;

    Matrix matrix = { { 0 }, { 0 } };
    matrix[0][0] = 1;
    matrix[0][1] = 2;

    return 0;
}
```



typedef

- Een typedef wordt vaak gebruikt in combinatie met een struct:

Code:

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

int main(void) {
    typedef struct Deelnemer{
        char naam[80];
        int punten;
    } student;
    student s1 = {"Dennis",7};
    printf("aantal punten = %d",s1.punten);
    getchar();
    return 0;
}
```

Uitvoer:

```
aantal punten = 7
```



Statische datastructuren

- **struct** kan array(s) bevatten.
 - ```
typedef struct {
 char naam[80];
 int punten;
} Deelnemer ;
```
- Array kan uit **structs** bestaan.
  - ```
Deelnemer deelnemers[100];
```



Voorbeeld

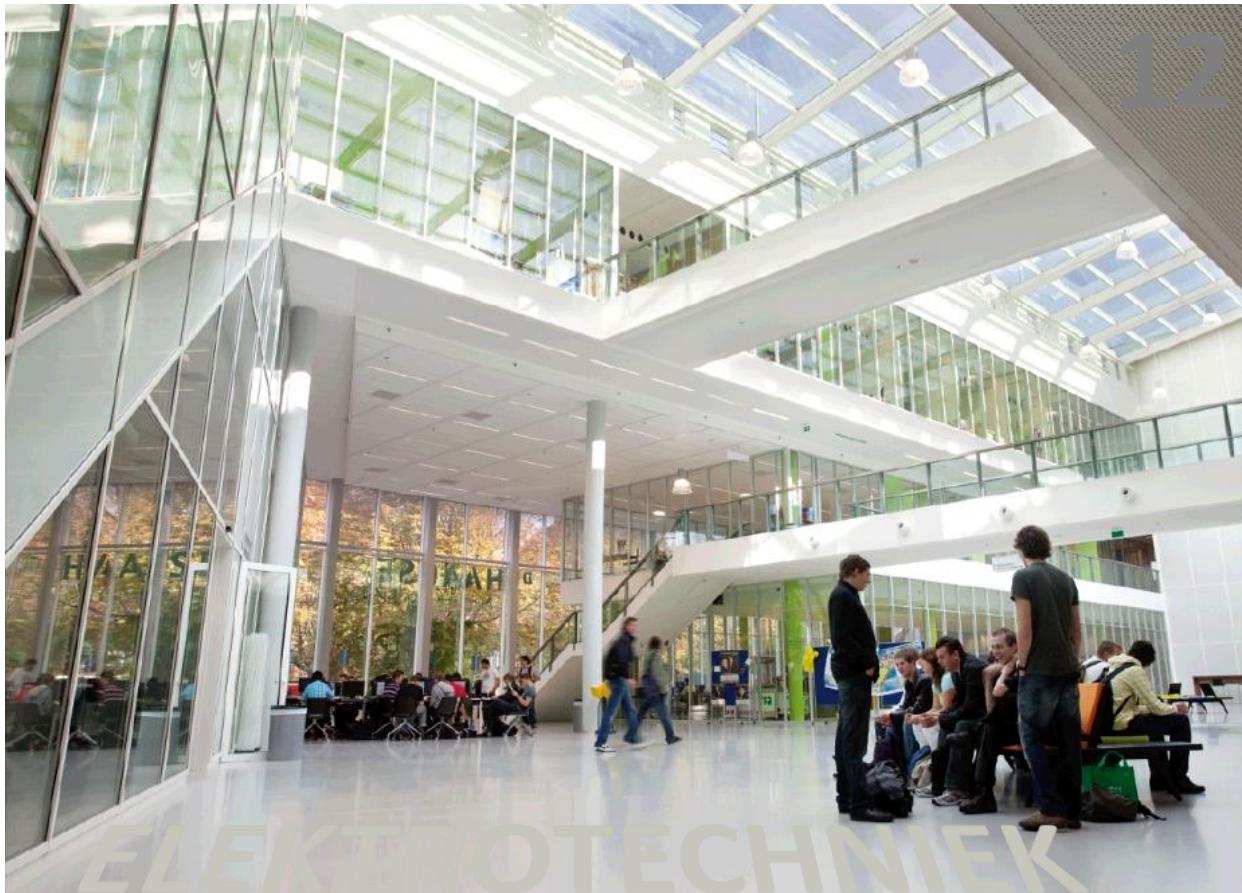
```
#include <stdio.h>
#include <string.h>

typedef struct {
    char * naam;
    int punten;
} Deelnemer;

printStruct(Deelnemer d){
    printf("Naam: %s\n", d.naam);
    printf("Punten: %d\n\n", d.punten);
}

int main(void) {
    Deelnemer deelnemers[100];
    deelnemers[0].naam = "lalaaLekkerlang";
    deelnemers[0].punten = 2;
    printStruct(deelnemers[0]);
    getchar();
}
```

Waarom geeft dit
geen foutmelding?



Gestructureerd programmeren in C

Macro's en modules

Huiswerk Uitwerking



```
void reverse(int a[], int n) {  
    int first = 0, last = n - 1;  
    while (first < last) {  
        wissel(&a[first], &a[last]);  
        first = first + 1;  
        last = last - 1;  
    }  
}
```

Zelfstudie!

```
void reverse(int a[], int n) {  
    int first = 0, last = n - 1;  
    while (first < last) {  
        wissel(&a[first++], &a[last--]);  
    }  
}
```

Kan het nog korter?



Huiswerk Uitwerking



```
void reverse(int a[], int n) {  
    int first = 0, last = n - 1;  
    while (first < last) {  
        wissel(&a[first++], &a[last--]);  
    }  
}
```

Zelfstudie!

```
void reverse(int a[], int n) {  
    int first = 0, last = n;  
    while (first < --last) {  
        wissel(&a[first++], &a[last]);  
    }  
}
```

Kan het nog korter?

Huiswerk Uitwerking



```
void reverse(int a[], int n) {  
    int first = 0, last = n;  
    while (first < --last) {  
        wissel(&a[first++], &a[last]);  
    }  
}
```

Zelfstudie!

```
void reverse(int *a, int n) {  
    int *last = a + n;  
    while (a < --last) {  
        wissel(a++, last);  
    }  
}
```

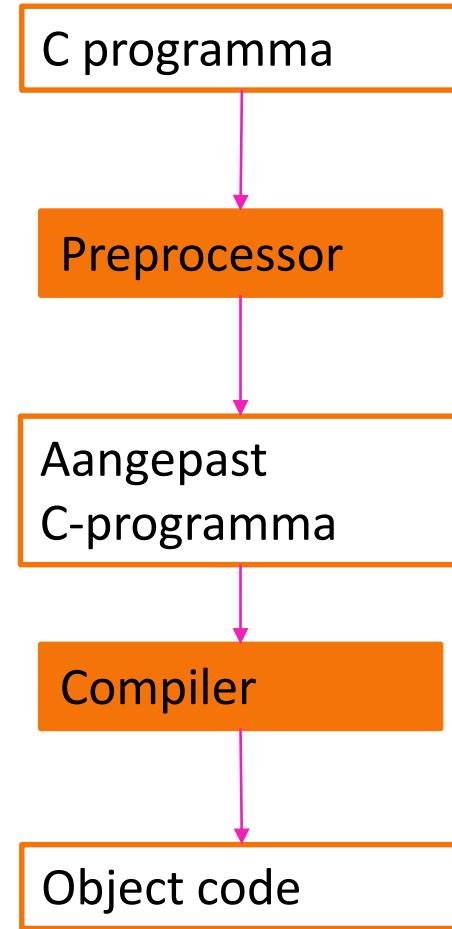
Kan het nog korter?



C preprocessor

- Wordt uitgevoerd voordat de “echte” compiler wordt gestart.
 - `#include` invoegen van andere bestanden
 - `#define` definiëren van `macro's`.
 - `#if` conditionele compilatie
- Directieven voor de preprocessor
- Syntaxis is onafhankelijk van de C syntaxis
 - Onder syntax (syntaxis) worden de ‘taalregels’ van de programmeertaal verstaan.
 - In het Engels: syntax

C preprocessor





#include

- Include file bevat prototypes (code zit in library die meegelinkt wordt).

```
/*oops include vergeten!*/
int main(void) {
    printf("%.15lf", sin(1,2));
    getchar();
    return 0;
}
```

Compiler:

```
Warning: 'printf' undefined
Warning: 'sin' undefined
Warning: 'getchar' undefined
```

Uitvoer:

```
0.000000000000000
```



#include

- Include file bevat **declaraties** (definities zitten in library die meegelinkt wordt).

```
#include <stdio.h>
#include <math.h>
int main(void) {
    printf("%.15lf", sin(1,2));
    getchar();
    return 0;
}
```

Compiler:

Error: 'sin' : too many arguments for call



#include

- Include file bevat **declaraties** (definities zitten in library die meegelinkt wordt).

```
#include <stdio.h>
#include <math.h>
int main(void) {
    printf("%.15lf", sin(1.2));
    getchar();
    return 0;
}
```

Uitvoer:

```
0.932039085967226
```



Zelfgemaakte include files

- `#include <stdio.h>`
Zoek in de standaard include directories.
- `#include "homemade.h"`
Zoek in de directory waar de .c file staat en daarna in standaard include directories.
- `#include "h:/mijn_includes/homemade.h"`
Zoek in het directory h:/mijn_includes/.
- Een *zelfgemaakte* header file i.c.m. een *bijbehorende* source file wordt vaak een *module* genoemd.
- Tijdens het compileren wordt de header file code en de source file.

Voorbeeld van een gekoppelde header en een gekoppelde bronbestand



Code:

```
#include <stdio.h>           /*homemade-1.h*/
#include "homemade-1.h"        void functieInheader(int getal);

int main(void) {               /*homemade-1.c*/
    functieInheader(3);       void functieInheader(int getal){
    getchar();                printf("\nIk sta in homemade-1.h\n");
    return 0;                  printf("en print: %d\n", getal);
}
}
```

Let op! Deze code staat
in **main.c**

Let op! Deze code staat
in **homemade-1.c**

Uitvoer:

```
Ik sta in homemade-1.h
en print: 3
```

Voorbeeld zelfgemaakte include



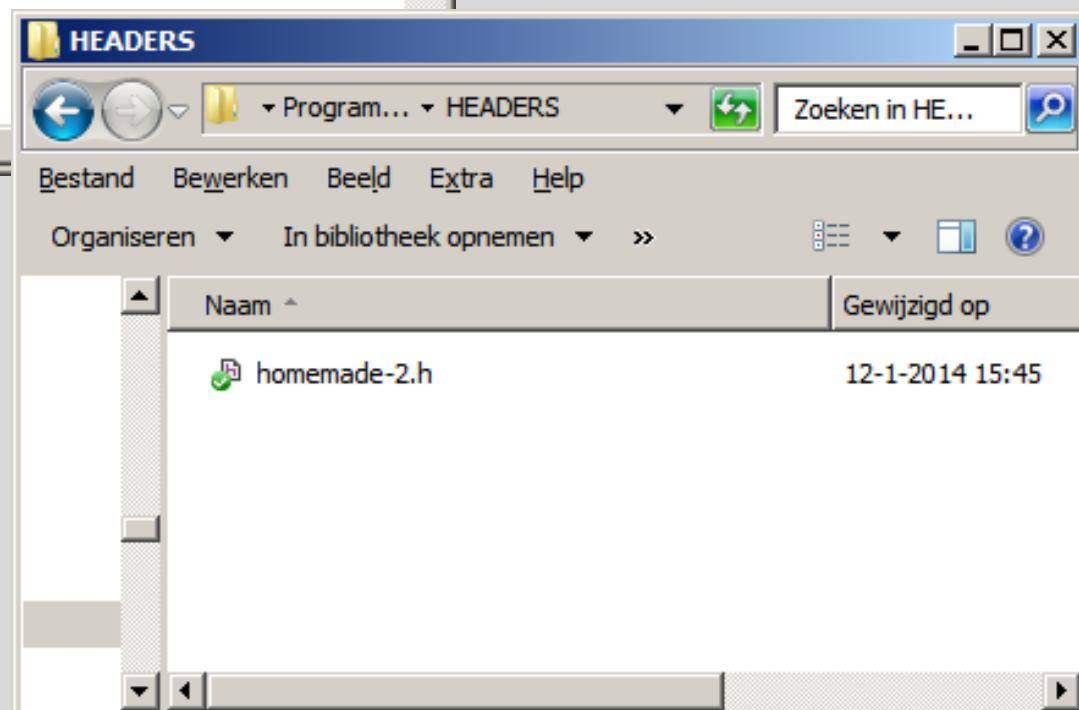
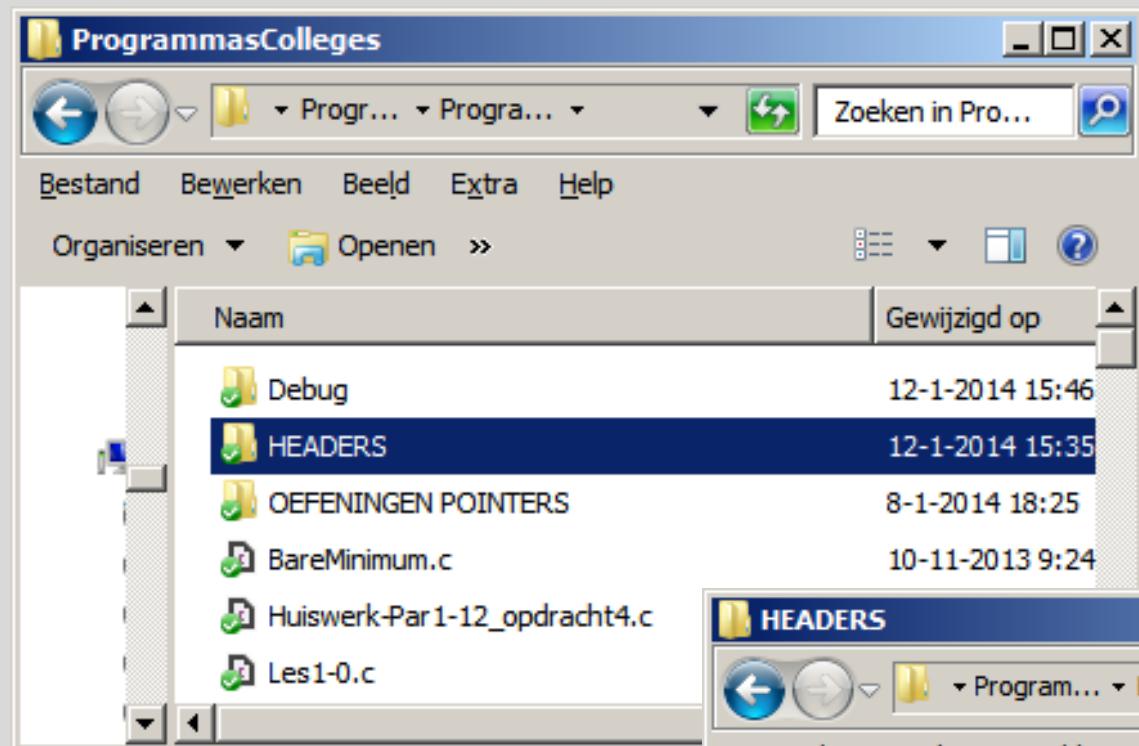
- Wat als een header bestand in een andere map staat?
 - Gebruik een relatief pad

Code:

```
#include <stdio.h>
//homemade-2.h bevat dezelfde code als homemade-1.h
#include "..\ProgrammasColleges\HEADERS\homemade-2.h"
//dit mag ook: #include "HEADERS\homemade-2.h"
// '..\' geeft aan dat je één relatieve directory boven het uit
// te voeren .c bestand kijkt.
int main(void) {
    functieInheader(3);
    getchar();
    return 0;
}
```

Uitvoer:

```
Ik sta in homemade-2.h
en print: 3
```





Macro's (zonder argumenten)

- Zoek en vervang (voor het compileren).

```
#include <stdio.h>
#define AANTAL 10
```

```
int main(void) {
    int kwadraten[AANTAL];
    int i;
    for (i = 0; i < AANTAL; i = i + 1) {
        kwadraten[i] = i * i;
    }
```

Voordeel?

Macro's (met argumenten)



- Zoek en vervang (voor het compileren) met parameter(s).



```
#include <stdio.h>
#define KWAD(x) x * x
```

```
int main(void) {
    printf("%d\n", KWAD(8));
    printf("%d\n", KWAD(4 + 4));
    getchar();
    return 0;
}
```

Uitvoer:

```
64
24
```

Wat is er mis?

Macro's (met argumenten)



- Zoek en vervang (voor het compileren) met parameter(s).

```
#include <stdio.h>
#define KWAD(x) ((x) * (x))
```

```
int main(void) {
    int i = 7;
    printf("%d\n", KWAD(8));
    printf("%d\n", KWAD(4 + 4));
    printf("%d\n", KWAD(++i));
    getchar();
    return 0;
}
```

Uitvoer:

```
64
64
81
```

Wat is er mis?



No Macro's (met argumenten)



- Gebruik functie's

```
#include <stdio.h>

int kwad(int x) {
    return x * x;
}

int main(void) {
    int i = 7;
    printf("%d\n", kwad(8));
    printf("%d\n", kwad(4 + 4));
    printf("%d\n", kwad(++i));
    getchar();
    return 0;
}
```

Uitvoer:

```
64
64
64
```



Conditionele compilatie

- `#if`
- `#ifdef`
- `#ifndef`
- `#elif`
- `#else`

- `#endif`



Conditionele compilatie

Code:

```
#include <stdio.h>

#define DEBUG 1

#if DEBUG
    #include "homemade-debug-1.h"
#else
    #include "homemade-1.h"
#endif

int main(void) {
    functieInheader(3);
    getchar();
    return 0;
}
```

Uitvoer:

```
Ik sta in homemade-debug-1.h
en print: 3
```



Conditionele compilatie

Code:

```
#include <stdio.h>

#define DEBUG 0

#if DEBUG
    #include "homemade-debug-1.h"
#else
    #include "homemade-1.h"
#endif

int main(void) {
    functieInheader(3);
    getchar();
    return 0;
}
```

Uitvoer:

```
Ik sta in homemade-1.h
en print: 3
```



Conditionele compilatie

Code:

```
#include <stdio.h>

#define CHECKENV 1

#if CHECKENV == 1
    #error "Je zit in de verkeerde omgeving"
#else
    #include "homemade-1.h"
#endif

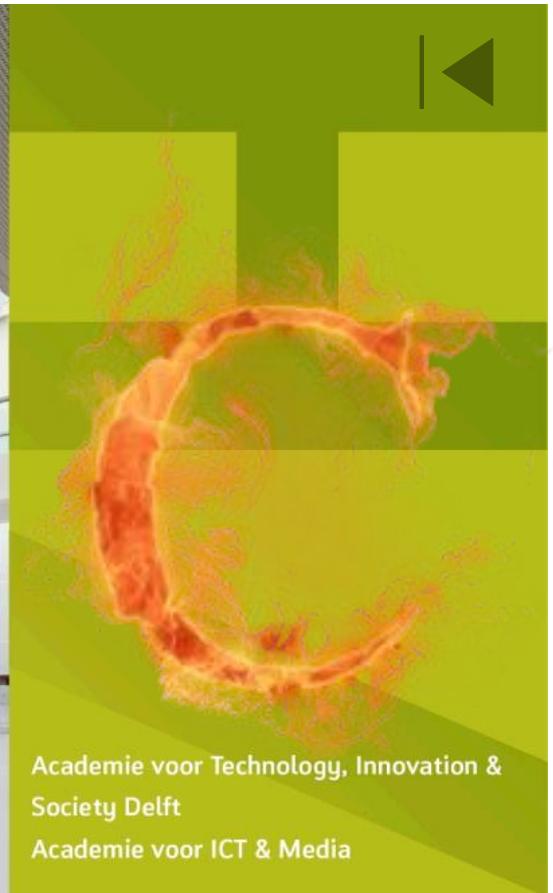
int main(void) {
    functieInheader(3);
    getchar();
    return 0;
}
```

Uitvoer: Error 1 error C1189: #error : "Je zit in de verkeerde omgeving"

Huiswerk



- Bestudeer C boek:
 - paragrafen 8.1 t/m 8.3.
 - paragraaf 8.7.



Gestructureerd programmeren in C

Details



Details!

- The devil is in the details.





Type Specifiers (C89)

- short int `sizeof(short) ≤ sizeof(int)` meestal 2
- long int `sizeof(long) ≥ sizeof(int)` meestal 4
- signed int / signed char two's complement values
- unsigned int / unsigned char only values ≥ 0

Verwarrend!



(Extra) Type Specifiers (C99)

- `stdint.h`
 - `int8_t` en `uint8_t`
 - `int16_t` en `uint16_t`
 - `int32_t` en `uint32_t`
 - `int64_t` en `uint64_t`
- `stdbool.h`
 - `bool` en de constanten: `false` en `true`
- `complex.h`
 - `complex` en diverse functies



Block Scope (C89)

- Variabelen mogen **alleen** aan het begin van een compound statement gedefinieerd worden, dus na **{**.
- De **scope** (zichtbaarheid) is tot einde van het betreffende compound statement behalve als naam verborgen is (door variabele met dezelfde naam).
- De lifetime (levensduur) tot einde van compound statement **}** uitgevoerd is.

```
int a[] = {1, 2, 3, 4, 5} , som = 0;  
int i = 0;  
for (i = 0; i < sizeof a / sizeof a[0]; i++) {  
    som += a[i];  
}
```



Block Scope (C99)

- Variabelen mogen **overal** in een compound statement gedefinieerd worden.
- De **scope** (zichtbaarheid) is tot einde van het betreffende compound statement behalve als naam verborgen is (door variabele met dezelfde naam).
- De lifetime (levensduur) tot einde van compound statement } uitgevoerd is.

```
int a[] = {1, 2, 3, 4, 5} , som = 0;
for (int i = 0; i < sizeof a / sizeof a[0]; i++) {
    som += a[i];
}
```



break

- **break**
 - Verlaten **switch**
 - Verlaten **for**, **while** of **do ... while**
 - **Niet** voor de **if!**

Komt de duidelijkheid meestal niet ten goede!

continue



- **continue**
 - Ga meteen naar de test van een **for**, **while** of **do ... while**
 - herhalingsopdrachten

Komt de duidelijkheid meestal niet ten goede!

```
int a[] = {1, 2, 3, 4, 5} , som = 0;
for (int i = 0; i < sizeof a / sizeof a[0]; i++) {
    continue;
    som += a[i];
}
```

Wat is de waarde
van som als de for-
lus klaar is?

Niet behandeld in deze lessen



- Static
- Extern
- Union
- Volatile
- Const
- Enum
- Register
- Default
- Etc.
- Assert()

Waar leer je dit?

- Pointers naar functies
- PSD's
- Structurele peer review
- Dynamische geheugen aanmaken/verwijderen
- Good practices



Toekomst

- PROEPP (blok 4):
 - gebruik **Linux** bordje, programmeren in C
- MICPRG (tweede jaar):
 - **shift** operators
 - **bitwise** operators
 - hexadecimale constanten
- OGOPRG (tweede jaar):
 - object georiënteerd programmeren in **C++**
 - modeleren met **UML**



Verre toekomst



- ECV (derde jaar)
 - RTSYST (verplicht)
 - Real-Time software en Real-Time Operating System
 - ALGODS (keuze)
 - Algoritmen en datastructuren



Huiswerk



- Bestudeer C boek:
 - paragraaf 4.7 (laatste deel op pagina 126).
 - paragraaf 4.15.
- Maak opdracht:
 - 9 van paragraaf 4.19.
- Bekijk (eventueel):
 - http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdint.h
 - http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdbool.h
 - http://en.wikibooks.org/wiki/C_Programming/C_Reference/complex.h



Tentamen tips

- Zorg dat je **tenminste** deze onderwerpen goed kent:
 - functies
 - operatoren
 - arrays & pointers
 - structs
 - file in- en output
 - Strings

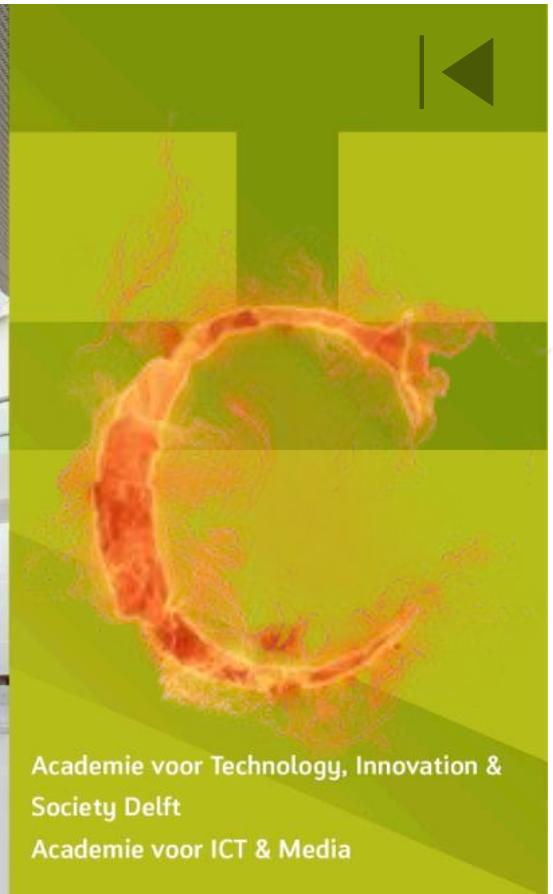
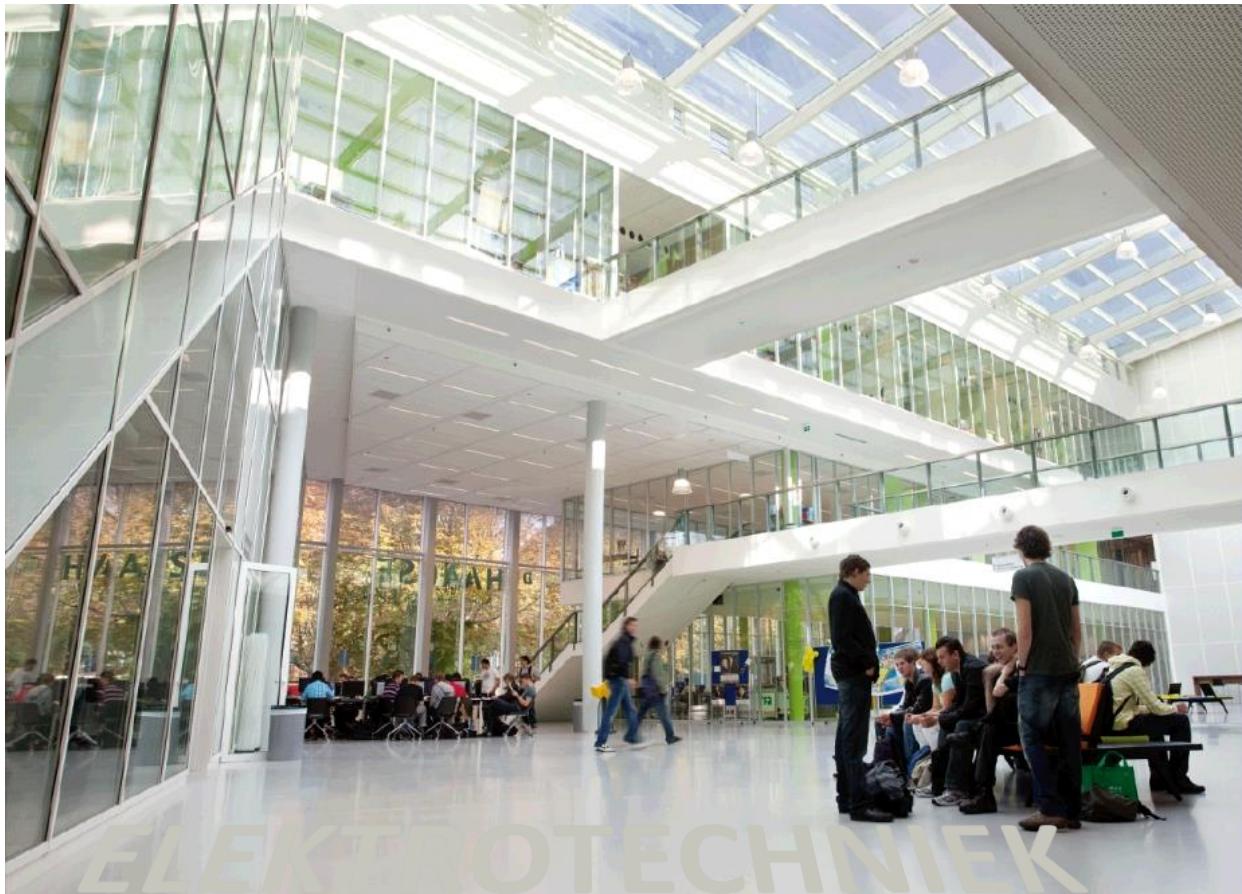


Tentamen tips

- Lees de vraag goed!
- Probeer altijd iets in te vullen!
- Heb je alles al gelezen en gezien wat op BB staat?
 - Oefententamens
 - Tips



**Succes met het
(voorbereiden
van het)
tentamen!**



Gestructureerd programmeren in C

Advanced: tekst en bestanden

Inkorten van een string



- Het inkorten van een string m.b.v. een pointer?

Code:

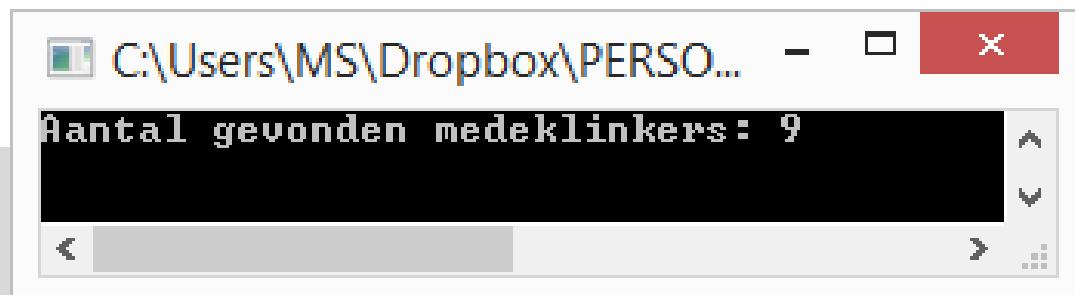
```
#include <stdio.h>
int main(void) {
    char * str = "Hottentottententoonstelling";
    char medeKlinkers[] = "AEIOUaeiou";
    int medeKlinkerTeller = 0;

    while (*str != '\0'){
        if (strchr(medeKlinkers, *str)){
            medeKlinkerTeller = medeKlinkerTeller +1;
        }
        str++;
    }
    printf("Aantal gevonden medeklinkers: %d", medeKlinkerTeller);
    getchar();
    return 0;
}
```

String met willekeurige tekst.

Wat doet deze test?

Wat gebeurt er hier?



String array en pointers



- Hoe gebruik je een string array m.b.v. pointers?

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

void printTekst(char tekst[][10], int aantal){
    int i = 0;
    for (int i = 0; i < aantal; i++)
    {
        printf("Tekst %d is: %s\n", i, tekst[i][]);
    }
}

void main(void){
    char tekst[][10] = { "ROOD", "BLAUW", "GROEN" };
    printTekst(tekst, 3);
    getchar();
}
```

Waarom werkt dit niet?

String array en pointers



- Hoe gebruik je een string array m.b.v. pointers?

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

void printTekst(char tekst[][10], int aantal){
    int i = 0;
    for (int i = 0; i < aantal; i++)
    {
        printf("Tekst %d is: %s\n", i, *(tekst + i));
    }
}

void main(void){
    char tekst[][10] = { "ROOD", "BLAUW", "GROEN" };
    printTekst(tekst, 3);
    getchar();
}
```

Alternatieve manier om een array element te specificeren:
 $a[i] \leftrightarrow *(a + i)$

Wat is een string ook alweer?

Werkt dit?

String array en pointers



- Wat wordt er nu afgedrukt?

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

void printTekst(char tekst[][10], int aantal){
    int i = 0;
    for (int i = 0; i < aantal; i++)
    {
        printf("Tekst %d is: %c\n", i, *(*(tekst + i)));
    }
}

void main(void){
    char tekst[][10] = { "ROOD", "BLAUW", "GROEN" };
    printTekst(tekst, 3);
    getchar();
}
```

String array en pointers



- Wat wordt er nu afgedrukt?

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

void printTekst(char tekst[][10], int aantal){
    int i = 0;
    for (int i = 0; i < aantal; i++)
    {
        printf("Tekst %d is: %c\n", i, *(*(tekst + i)+1));
    }
}

void main(void){
    char tekst[][10] = { "ROOD", "BLAUW", "GROEN" };
    printTekst(tekst, 3);
    getchar();
}
```

String array en pointers



- Hoe gebruik je een string array m.b.v. pointers?

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

void printTekst(char tekst[][10], int aantal){
    int i = 0;
    for (int i = 0; i < aantal; i++)
    {
        printf("Tekst %d is: %s\n", i, tekst[i]);
    }
}

void main(void){
    char tekst[][10] = { "ROOD", "BLAUW", "GROEN" };
    printTekst(tekst, 3);
    getchar();
}
```

Werkt dit?

Ja

stdin



- De onderstaande regel hebben we vaker gezien:
 - `fflush(stdin)`
- Waar gebruikte we deze regel voor?
- Wat is `stdin`?
 - `FILE* stdin`
 - Standaard FILE *
 - Gebruikt voor invoer (input stream)

Intermezzo: FILE*



- Wat is type van “FILE* outfile”?
- FILE *
- Q: Hoe verhoudt dit type zich met andere datatypes?
- A: Dat hoef je niet te weten
 - Het is een data-abstractie
 - Als programmeur moeten we met abstracties kunnen omgaan
 - Het enige datatype om bestanden te manipuleren

Code:

```
struct _iobuf {
    char *_ptr;
    int   _cnt;
    char *_base;
    int   _flag;
    int   _file;
    int   _charbuf;
    int   _bufsiz;
    char *_tmpfname;
};

typedef struct _iobuf FILE;
```

C reference Card (ANSI)



Input/Output <stdio.h>

Standard I/O

standard input stream
standard output stream
standard error stream
end of file (type is int)
get a character
print a character
print formatted data
print to string s
read formatted data
read from string s
print string s

stdin
stdout
stderr
EOF
getchar()
putchar(*chr*)
printf("format", *arg*1,...)
sprintf(s,"format", *arg*1,...)
scanf("format", &*name*1,...)
sscanf(s,"format", &*name*1,...)
puts(s)

File I/O

declare file pointer
pointer to named file
modes: r (read), w (write), a (append), b (binary)
get a character
write a character
write to file
read from file
read and store n elts to *ptr
write n elts from *ptr to file
close file
non-zero if error
non-zero if already reached EOF
read line to string s (< max chars)
write string s

FILE *fp;
fopen("name", "mode")
getc(fp)
putc(*chr*, fp)
fprintf(fp, "format", *arg*1,...)
fscanf(fp, "format", *arg*1,...)
fread(*ptr, eltsize, n, fp)
fwrite(*ptr, eltsize, n, fp)
fclose(fp)
ferror(fp)
feof(fp)
fgets(s, max, fp)
fputs(s, fp)

De volledige reference card zit standaard bij ieder tentamen

stdout



- Wat doet deze code?

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 0; i < 10; i = i + 1) {
        fprintf(stdout, "Kwadraten: %d.\n", i * i);
    }

    getchar();
    return 0;
}
```

stdout / stdin



- Wat doet deze code?

```
#include <stdio.h>

int main(void) {
    fprintf(stdout, "Geef een karakter op: ");
    char karakter = getc(stdin);
    fprintf(stdout, "\nIngevoerd karakter %c \n", karakter);

    getchar();  getchar();
    return 0;
}
```



Gestructureerd programmeren in C

Pointers naar functies

Pointers naar functies



- In C kun je een pointer naar een functie definiëren.
 - De waarde van de pointer is het beginadres (van de code) van de functie.

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

```
int kwadraat(int c) {  
    return c * c;  
}
```

```
int dubbel(int c) {  
    return c + c;  
}
```

pnf is een **pointer naar een functie** met een int als parameter en een int returnwaarde

```
int main(void) {  
    int a = 7, b;  
    int (*pnf)(int);  
    pnf = &dubbel;  
    b = (*pnf)(a);
```

pnf wijst naar de functie dubbel (pnf wordt gelijk aan het adres van de functie dubbel)

Waarom haakjes?

Pointers naar functies



- Verkorte schrijfwijze.
 - Naam van een functie \leftrightarrow beginadres (van de code) van de functie.

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

```
int kwadraat(int c) {  
    return c * c;  
}
```

```
int dubbel(int c) {  
    return c + c;  
}
```

```
int main(void) {  
    int a = 7, b;  
    int (*pnf)(int);  
    pnf = dubbel;  
    b = pnf(a);
```

Pointers naar functies



- Wat is het nut?
 - Functie als parameter.

```
#include <stdio.h>
/* ... */
void printTabel(int (*p)(int), int van, int tot, int stap) {
    int x;
    for (x = van; x < tot; x += stap) {
        printf("%10d %10d\n", x, (*p)(x));
    }
}
int main(void) {
    printf("De kwadraten van 1 t/m 10\n");
    printTabel(&kwadraat, 1, 11, 1);
    printf("De dubbelen van de drievouden van 0 t/m 30\n");
    printTabel(&dubbel, 0, 31, 3);
}
```

Uitvoer



De kwadraten van 1 t/m 10

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

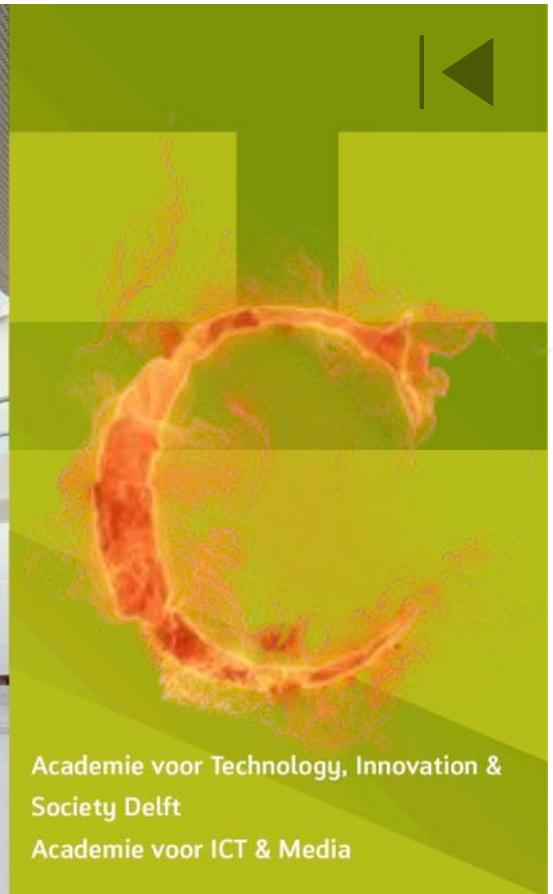
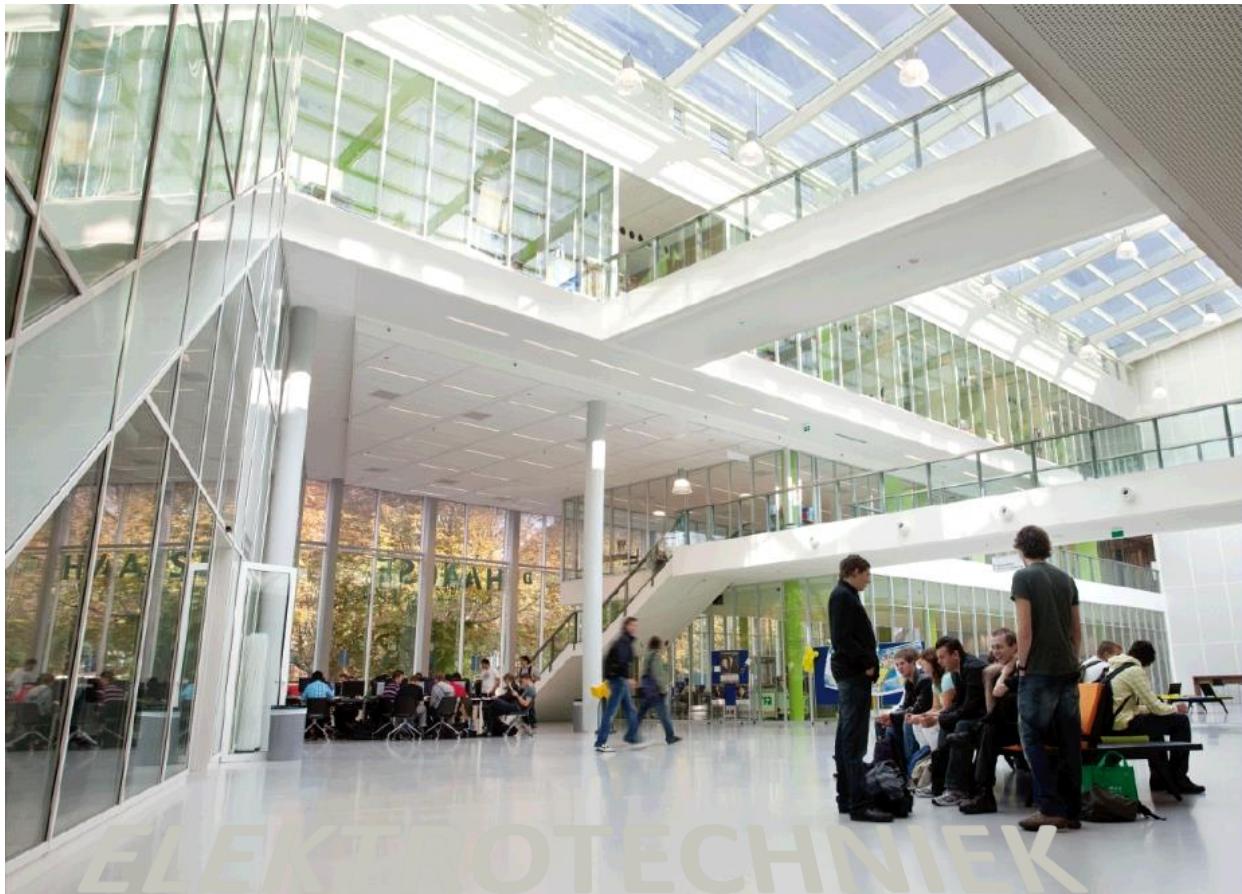
De dubbelen van de drievouden van 0 t/m 30

0	0
3	6
6	12
9	18
12	24
15	30
18	36
21	42
24	48
27	54
30	60

Huiswerk



- Bestudeer C boek:
 - paragraaf 6.15.



Gestructureerd programmeren in C

Foutcontrole

Domein (van een functie)



- Faculteit (met int)
 - Domein = [0..12]
- n boven k (met int's)
 - Domein versie met faculteit = [0..12]
 - Domein recursieve versie = [0..33]

Wat was dat ook
alweer?



Ligt een argument in het domein?



- Aanroeper (caller)
 - De **aanroeper** moet voor aanroep van **faculteit(i)** controleren of **i** in het domein van de functie valt.
 - Wat moet de functie doen als **i** niet in het domein valt?

Niets! Misschien wel tijdens ontwikkelfase...
Gebruik assert (zie volgende sheet).

- Aangeroepene (callee)
 - De **functie** moet na aanroep van **faculteit(i)** controleren of **i** in het domein van de functie valt.
 - Wat moet de functie doen als **i** niet in het domein valt?

Foutmelding geven? Waar (denk aan embedded systeem)?
Foutcode teruggeven? Caller moet checken!

assert



```
#include <cassert.h>

int faculteit(int n) {
    int i, res = 1;
    assert(n >= 0 && n <= 12);
    for (i = 1; i <= n; i = i + 1) {
        res = res * i;
    }
    return res;
}
```

Ik beweer: $0 \leq n \leq 12$

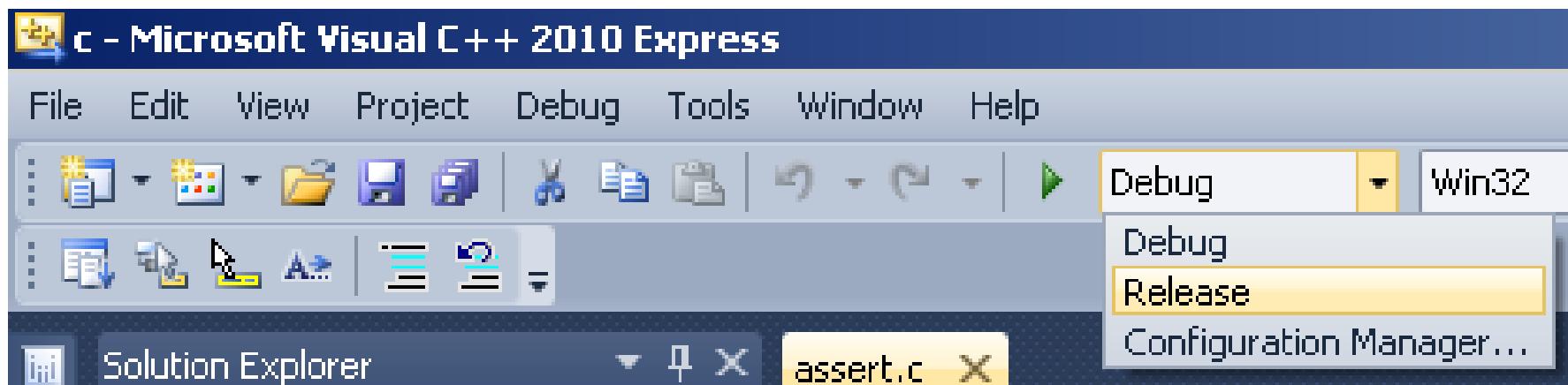


MAKE THIS PLEDGE:



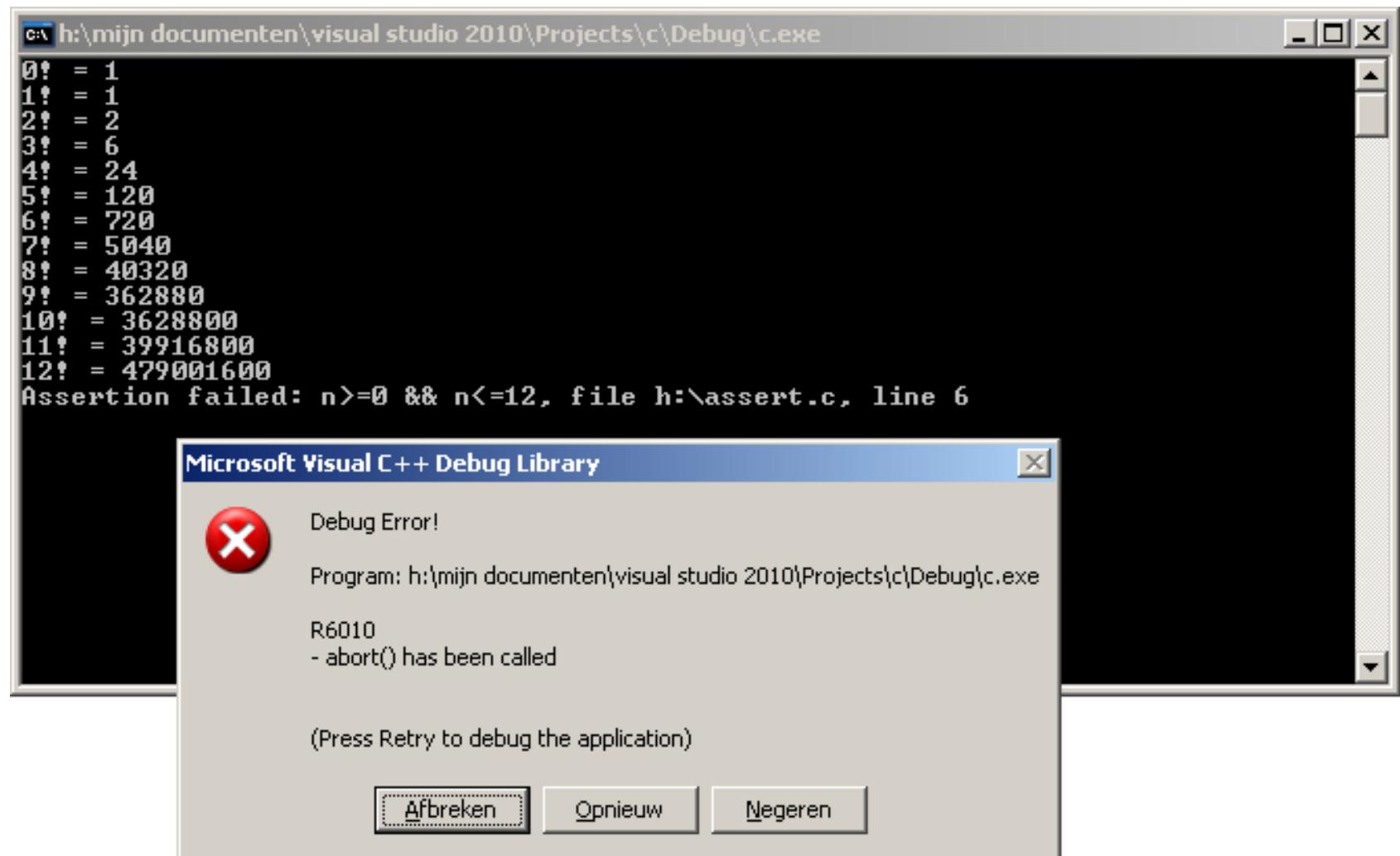
assert

- Compile in **Debug** mode: Functie geeft foutmelding en stopt programma als argument van assert 0 is.
- Compile in **Release** mode: Functie doet niets.





assert in Debug mode





assert in Release mode

```
c:\ h:\mijn documenten\visual studio 2010\Projects\c\Release\c.exe
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
-
```



Foutcode

```
int faculteit(int n) {
    int i, res = 1;
    if (n >= 0 && n <= 12) {
        for (i = 1; i <= n; i = i + 1) {
            res = res * i;
        }
    }
    else {
        res = -1;
    }
    return res;
}
```



Foutcode controleren

```
int main(void) {
    int n, fac;
    for (n=0; n<=13; n=n+1) {
        fac = faculteit(n);
        if (fac != -1) {
            printf("%d! = %d\n", n, fac);
        }
        else {
            printf("%d! = te groot\n", n);
        }
    }
    getchar();
    return 0;
}
```



Uitvoer

```
h:\mijn documenten\visual studio 2010\Projects\c\Debug\c.exe
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = te groot
-
```



Huiswerk (deel 2)

- Als het goed is, heb je na les 6 een **recursieve** functie `int fib(int n)` geschreven die het n^{de} getal uit de fibonacci rij berekent.
- Wat is het **domein** van deze functie?
- Pas de functie aan zodat gecontroleerd wordt of n in het domein ligt.
 - Als de **caller** verantwoordelijk is voor de controle
 - Als de **callee** verantwoordelijk is voor de controle
- Bestudeer C boek:
 - paragraaf 8.10.



Uitwerking Reverse

```
void wissel(int *p, int *q) {
    int hulpje = *p;
    *p = *q;
    *q = hulpje;
}

void reverse(int a[], int n) {
    int first = 0, last = n - 1;
    while (first < last) {
        wissel(&a[first], &a[last]);
        first = first + 1;
        last = last - 1;
    }
}
```

