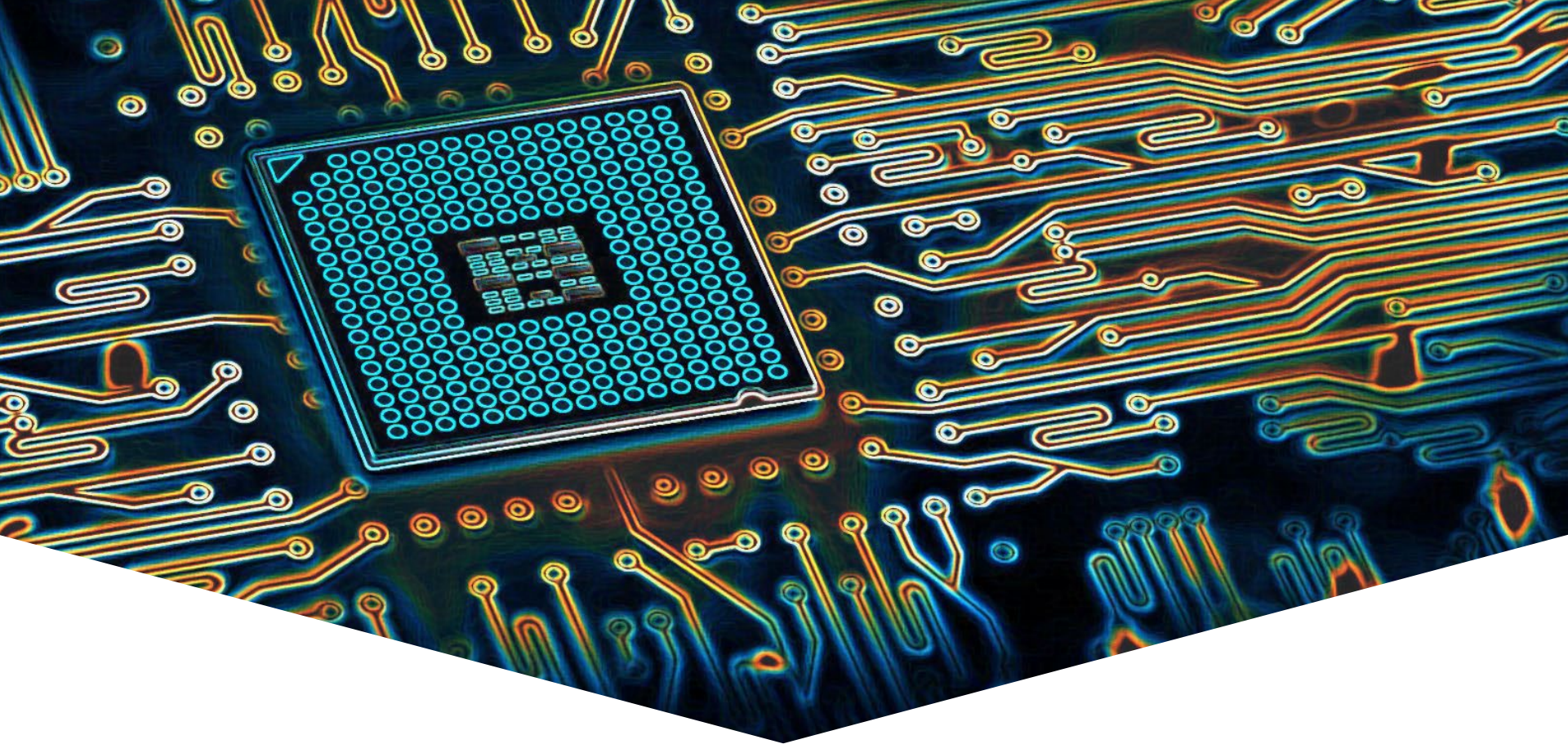


# EMS30 Introductie

# Embedded Systems: Jouw toekomst?



## Vacature HBO Elektrotechniek (Embedded Systems):

- Bare-metal C/C++ programmeren (en documenteren) op embedded systemen;
- Communicatieprotocollen opstellen en vastleggen;
- Vertalen van (bestaande) schema's naar een pcb-ontwerp;
- Ervaring met hardware (op componentniveau) gekoppeld aan embedded systemen;
- Kunnen werken met multimeter en oscilloscoop;


Bron: https://toptech.nl/vacatures/Software/Embedded-Software-Engineer/2854

# Embedded Systems: Jouw toekomst?



## Vacature HBO Elektrotechniek (Embedded Systems):

- Je hebt een bachelor- of mastergraad in een technische richting, zoals bijv. Technische Informatica, Elektrotechniek of Embedded Systems.
- Je heb gedegen kennis van softwareontwikkeling in C/C++ en Python, die verder gaat dan de code. Je kunt de architectuur van software ontwerpen en bent bedreven in CI/automatisch testen.
- Ervaring met hardware design en draadloze communicatietechnieken is een groot pluspunt.


Bron: https://careers.abelsensors.com/medior-senior-embedded-systems-engineer/nl

# Embedded Systems: Jouw toekomst?



## Vacature HBO Elektrotechniek (Embedded Systems):

- Een afgeronde technische opleiding zoals (Technische) Informatica, Computer Science, Elektrotechniek, Mechatronica of Embedded Systems op hbo- of wo niveau;
- Ervaring met de programmeertalen C en C++;
- Ervaring met embedded systemen;
- Kennis van (embedded) Linux.


Bron: https://www.alten.nl/career-nl/139-embedded-software-engineer/

# Embedded Systems: Jouw toekomst?



## Vacature HBO Elektrotechniek (Embedded Systems):

- experience in embedded C/C++ development and Linux;
- Good understanding of object-oriented design and design patterns;
- Understanding of data communication protocols like Modbus, BACnet, DALI, and/or EnOcean is a nice to have;
- Experience with Yocto, GitHub, Robot Framework and GoogleTest is a nice to have;

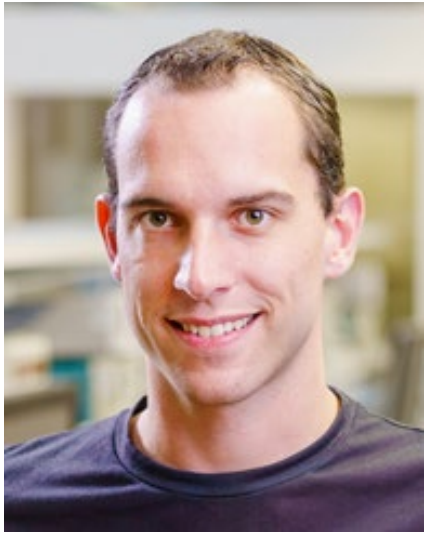

Bron: https://careers.priva.com/o/embedded-engineer

# Docenten EMS30



Ron Verhagen
VeRon@hr.nl

Daniël Versluis
VersD@hr.nl



Roy Bakker
BaRoy@hr.nl

Harry Broeders
BroJZ@hr.nl

# Leerdoelen EMS30



## Leerdoelen EMS30.

Je leert in week **1 t/m 4**:

- hoe je een betere **C**-programmeur wordt.

Je leert in week **4 t/m 8**:

- de basisbeginselen van objectgeoriënteerd programmeren in **C++**;
- de basisbeginselen van objectgeoriënteerd ontwerpen met **UML**.

# Leerdoelen EMS30 week 1 t/m 4



| # | Niveau | Weging | De student is in staat om . . . |
|---|---|---|---|
| 1 | C | 5 % | . . . versiebeheer toe te passen (GIT) zodanig dat de student in groepsverband software revisies kan beheren. |
| 2 | C | 25 % | . . . een dynamische datastructuur te implementeren in C met behulp van pointers en de functies `malloc` en `free`. |
| 3 | C | 20 % | . . . de kwaliteit van code te verbeteren door het toepassen van unit testen, code coverage en coding standards. |

# Toetsing en studielast EMS30



| Toets | Leerdoelen | Weging | Deadline |
|---|---|---|---|
| Opdracht 1 | 1, 2 en 3 | 50 % | Lesweek 4, zondag 23.59 uur |
| Opdracht 2 | 4, 5 en 6 | 50 % | Lesweek T4, zondag 23.59 uur |

Je werkt in **tweetallen** samen m.b.v. git (*je krijgt van ons een repository*) en levert je werk ook in op dat repository.

| Werkvorm | Omschrijving | Studielast |
|---|---|---|
| Theorielessen | Theorie volgen. Week 5 t/m 8 | 8 klokuren |
| Practicum | Begin maken met de labopdrachten. | 36 klokuren |
| Zelfstudie | Bestuderen van het studiemateriaal. Uitwerken van alle labopdrachten. | 124 klokuren |

EMS30 levert je **6** studiepunten op dat betekent dat je 6/15 = 2/5 = **2 dagen / week** aan EMS30 dient te besteden.

# Planning week 1 t/m 4



| Week | Werkvorm | Beschrijving |
|---|---|---|
| 4.1 | Les | 1: Modules in C, 2: Dynamisch geheugenallocatie |
| 4.2 | Les | 1: Git, 2: Unittesten |
| 4.3 | Les | 1: Memory tests en test coverage, 2: Coding standards en static code analysis |
| 4.4 | Les<br>**Opdracht 1** | 1+2: Werken aan opdracht 1<br>Deadline zondag 23.59 uur |

# Studiemateriaal vind je op de wiki:



HR_ELEKTRO / Embedded / EMS30

Wiki

Create page

Clone wiki

**EMS30 / Home**

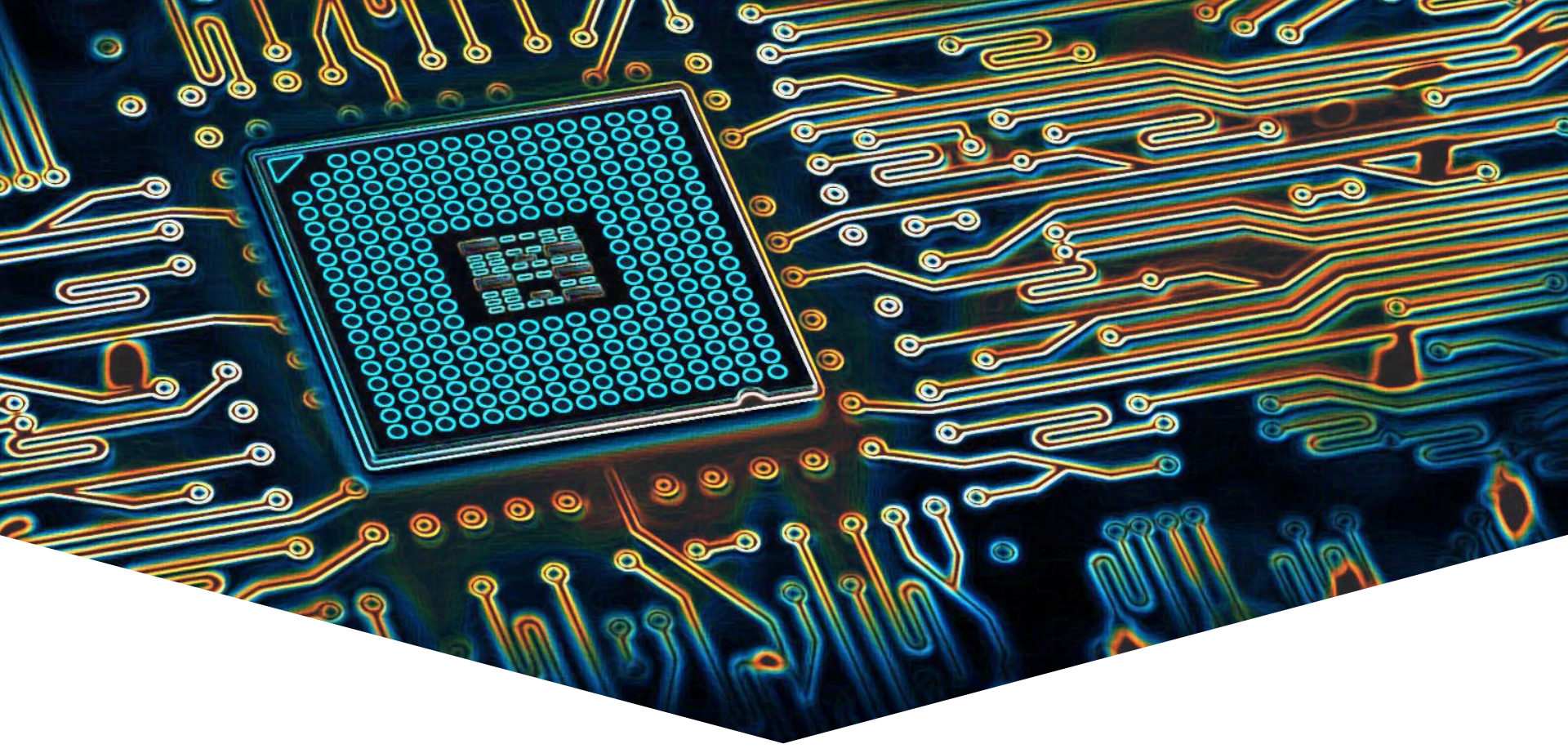
View | History | Edit

## EMS30 - Embedded Systems 3

Dit repository is bedoeld voor studenten en docenten van de opleiding Elektrotechniek van de Hogeschool Rotterdam en wordt gebruikt het studiemateriaal voor de cursus "EMS30 - Embedded Systems 3" te verspreiden.

**Let op! Deze wiki is nog niet volledig voor studiejaar 2022-2023.**

De informatie in dit repository is zoals alle mensenwerk niet foutloos, verbeteringen en suggesties zijn altijd welkom! Maak als je ons feedback wilt geven een issue aan.



# EMS30 Week 1 Les 1: Modules in C

# Modules in C



**Leerdoelen week 1 les 1.** Je leert hoe je:

- de interface van een module in een `.h` bestand kunt declareren;
- de implementatie van een module in een `.c` bestand kunt definiëren;
- deze module kunt gebruiken en testen op een pc;
- deze module kunt gebruiken en testen op een CC3220S Launchpad.

# Voorbeeld van een module in C



Als voorbeeld bekijken we een module `breuk`.

- Waarom zou je breuken willen gebruiken i.p.v. `double`'s?

# Module breuk

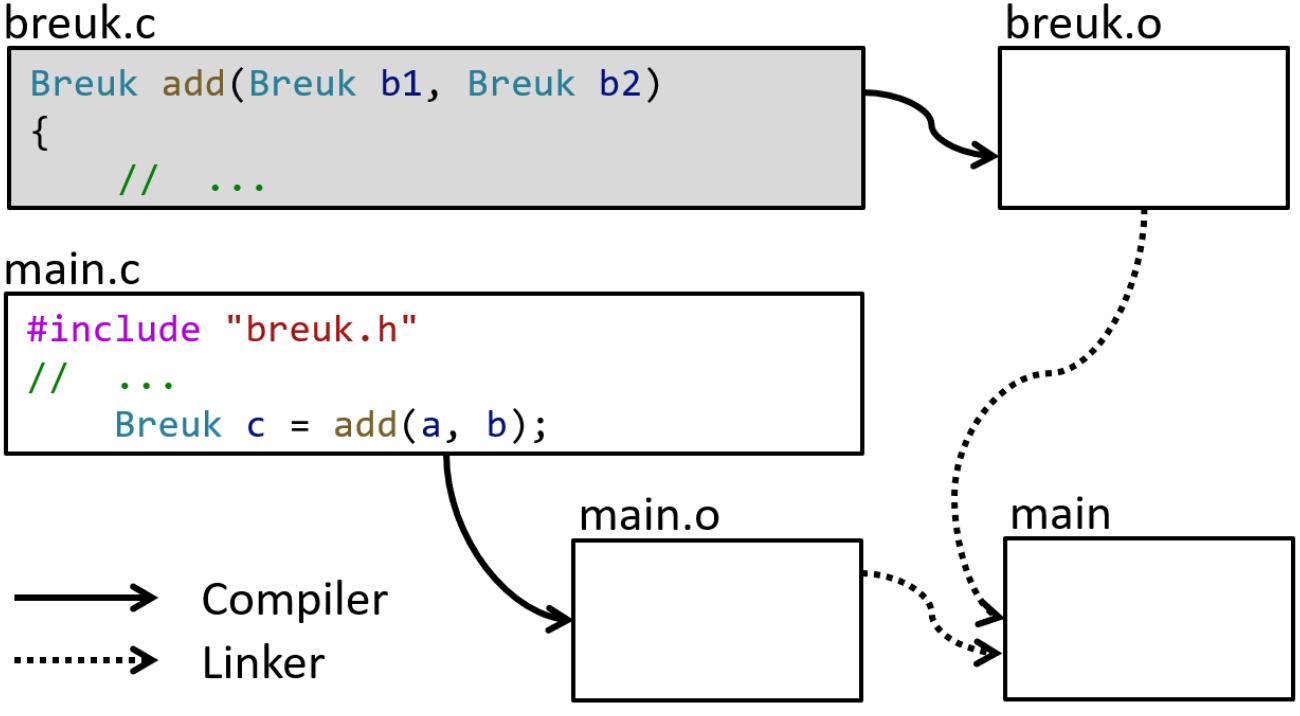


breuk.h

```
typedef struct {/* ... */} Breuk;
extern Breuk add(Breuk b1, Breuk b2);
extern Breuk mul(Breuk b1, Breuk b2);
```

breuk.c

```
Breuk add(Breuk b1, Breuk b2)
{
    //  ...
```



# Module breuk



We splitsen de module in een `.h` en een `.c` bestand:

- `.h` bestand bevat de definitie van alle types (`Breuk`) en de declaratie van alle functies (prototypes) (`add` en `mul`).
- `.c` bestand bevat de implementatie van alle functies.
- Het `.h` bestand is nodig om code die de module gebruikt te compileren, het `.h` bestand moet ge-include worden.
- Het `.c` bestand (of een gecompileerde versie daarvan) is nodig om code die de module gebruikt te linken (tot een executable), het gecompileerde `.c` bestand moet meegelinkt worden.

# breuk.h



```
#ifndef _HR_BroJZ_Breuk_
#define _HR_BroJZ_Breuk_

typedef struct
{
    int teller;
    int noemer;
} Breuk;

extern Breuk add(Breuk b1, Breuk b2);
extern Breuk mul(Breuk b1, Breuk b2);

#endif
```

Include guard voorkomt problemen bij meerdere keren includen in dezelfde `.c` file

# breuk.c



```c
#include <assert.h>
#include "breuk.h“

static Breuk normaliseer(Breuk b)
{
    assert(b.noemer != 0);
    // ...
    return b;
}

Breuk add(Breuk b1, Breuk b2)
{
    Breuk som;
    som.teller = b1.teller * b2.noemer + b1.noemer * b2.teller;
    som.noemer = b1.noemer * b2.noemer;
    return normaliseer(som);
}
// ...
```

`static` functie is alleen in deze `.c` file zichtbaar (is dus verborgen voor gebruikers van de module Breuk)

# main.c



```c
#include <stdio.h>

#include "breuk.h"

int main(void)
{
    Breuk a = {-2, 4}, b = {6, -8};

    Breuk c = add(a, b);
    printf("c = %d/%d\n", c.teller, c.noemer);

    Breuk d = mul(a, b);
    printf("d = %d/%d\n", d.teller, d.noemer);

    return 0;
}
```

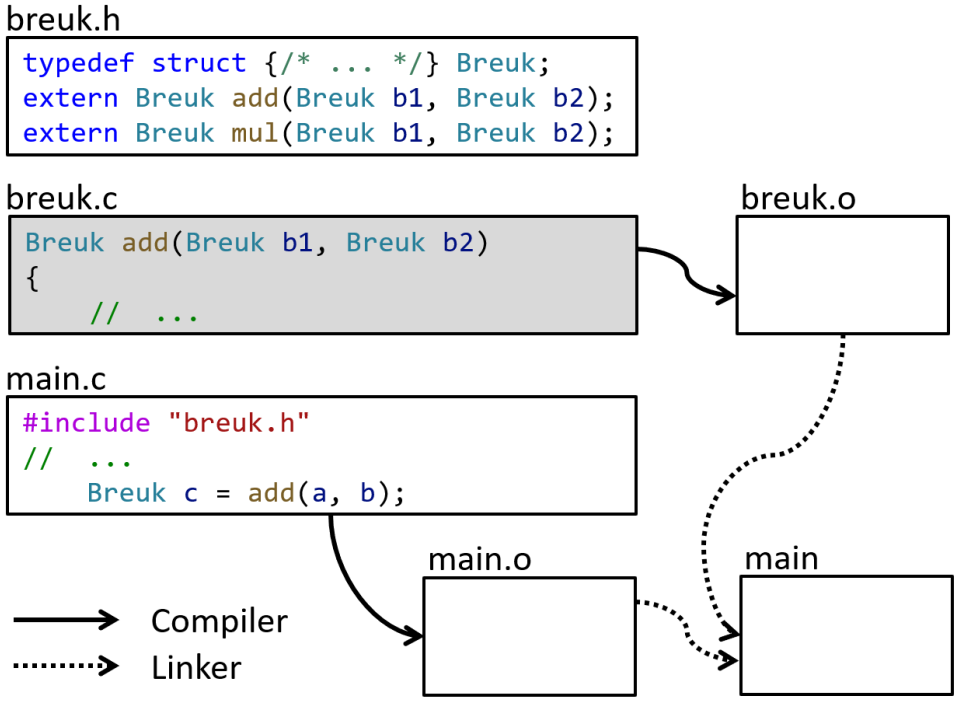
Output:

```
c = -5/4
d = 3/8
```

# Compileren en linken



```
$ gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c breuk.c
$ gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c main.c
$ gcc breuk.o main.o -o main
$ ./main
c = -5/4
d = 3/8
```



# Automatiseren build proces: make



## makefile:

IDE's (CCS) gebruiken make onder de motorkap

target

dependencies

```
main : breuk.o main.o
	breuk.o main.o -o main

breuk.o : breuk.c breuk.h
    gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c breuk.c

main.o : main.c breuk.h
    gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c main.c
```

tab

command

Veel meer mogelijkheden, zoek zelf maar uit:
http://www.gnu.org/software/make/manual/make.html

# Automatiseren build proces: make



```
$ make
gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c breuk.c
gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c main.c
gcc breuk.o main.o -o main
$ touch main.c
$ make
gcc -std=c18 -Wall -Wextra -pedantic-errors -g3 -O0 -c main.c
gcc breuk.o main.o -o main
```



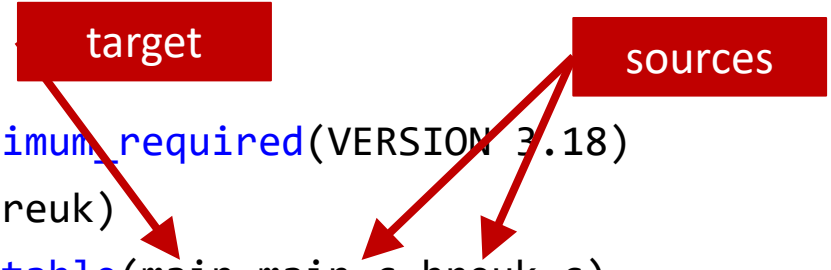
Zie: https://bitbucket.org/HR_ELEKTRO/ems30/raw/master/Programmas/breuk.zip

# Automatiseren build proces: CMake



Cmake genereerd makefile en bepaald dependencies

CMakeLists.txt:




```
cmake_minimum_required(VERSION 3.18)
project(breuk)
add_executable(main main.c breuk.c)
target_compile_options(main PRIVATE -std=c18 -Wall -Wextra -Wpedantic -g3 -O0)
```

Veel meer mogelijkheden, zoek zelf maar uit: https://cmake.org/

# Automatiseren build proces: CMake



```
$ mkdir build
$ cd build
$ cmake ..
-- The C compiler identification is GNU 12.2.1
-- The CXX compiler identification is GNU 12.2.1
-- …
-- Build files have been written to: /home/ems/breuk/build
$ make
[ 33%] Building C object CMakeFiles/main.dir/main.c.o
[ 66%] Building C object CMakeFiles/main.dir/breuk.c.o
[100%] Linking C executable main
[100%] Built target main
```



Zie: https://bitbucket.org/HR_ELEKTRO/ems30/raw/master/Programmas/breuk.zip

# CMake in Visual Studio Code

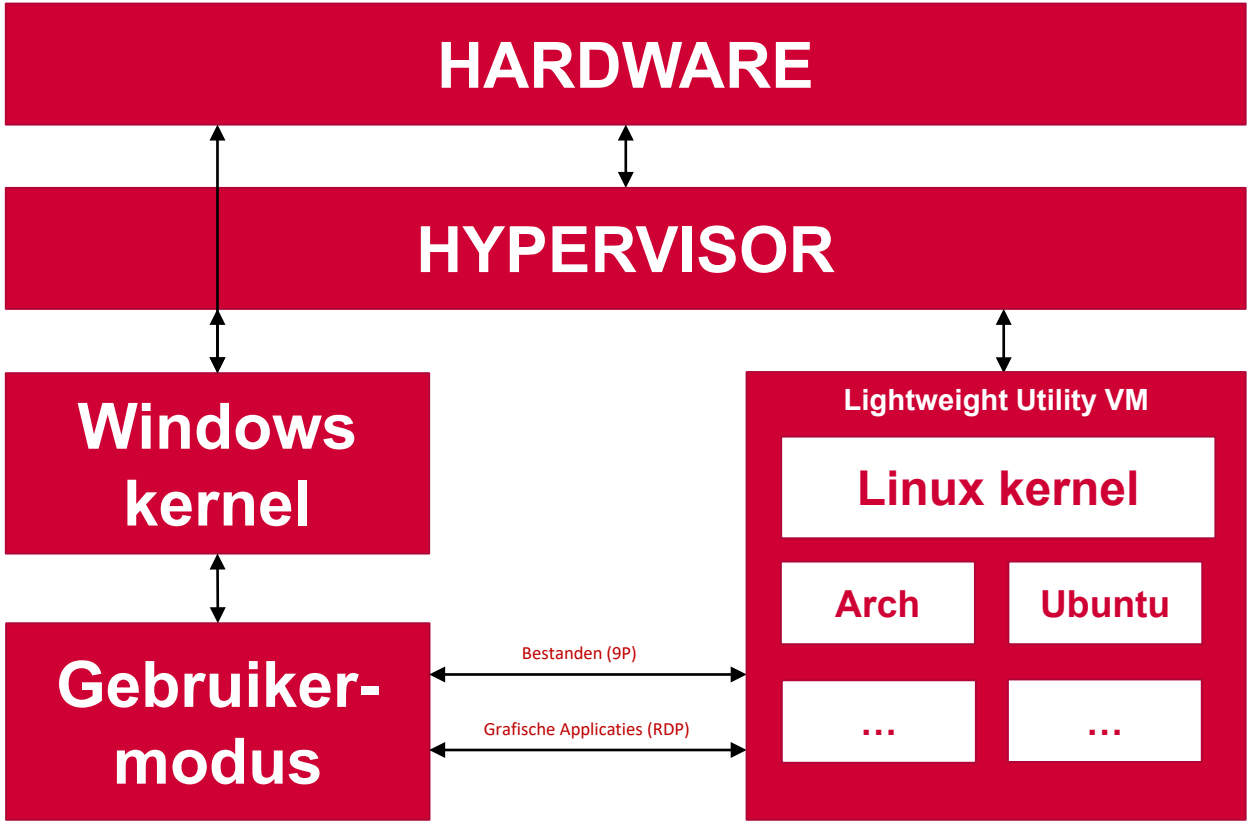





Zie: https://github.com/microsoft/vscode-cmake-tools/tree/main/docs

# WSL 2




HARDWARE
HYPERVISOR
Windows kernel
Gebruiker-modus
Bestanden (9P)
Grafische Applicaties (RDP)
Lightweight Utility VM
Linux kernel
Arch
Ubuntu
...
...

# Volgende les…



Dynamisch geheugenallocatie

# Aan de slag!



Aan de slag met Opdrachten_Week_1_Les_1.pdf