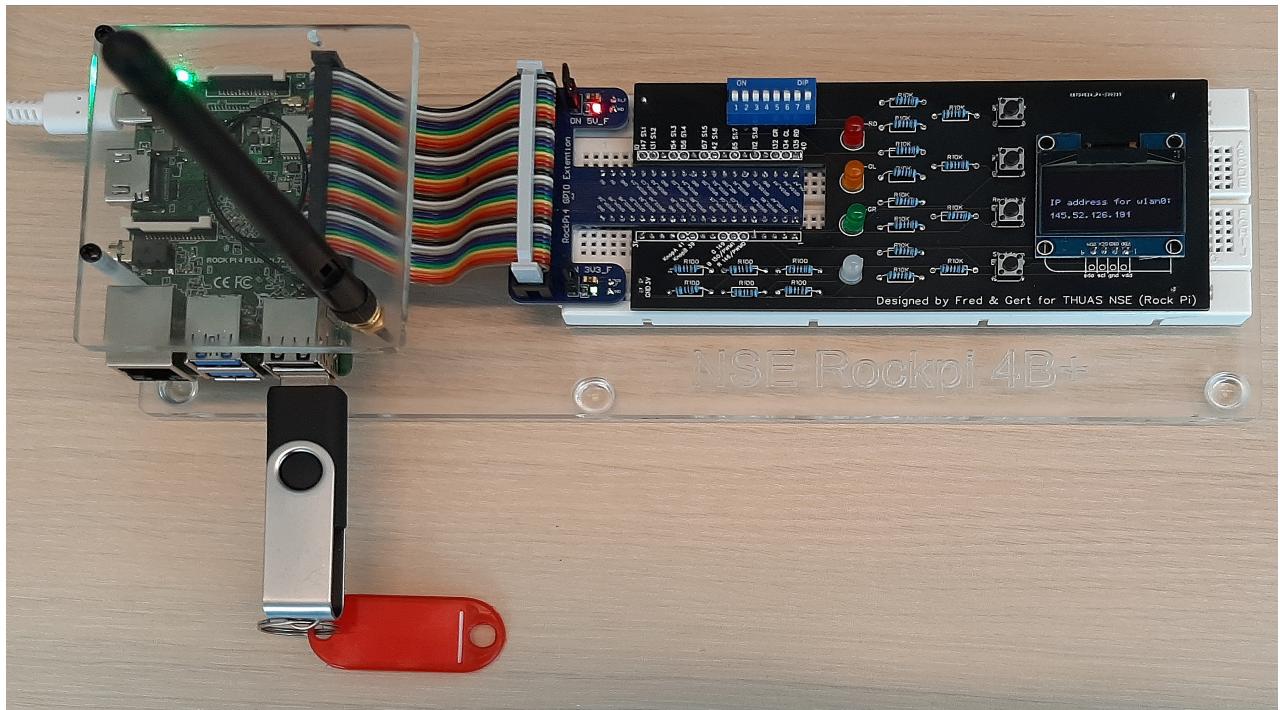


Object Georiënteerd Programmeren

(Introductie)

(Practicumhandleiding inclusief de practicumopstelling voor de RockPi)

Versie 1



Inhoudsopgave

1 Weektabel	4
2 Inleiding	5
2.1 Voorbereiding - Software installeren	5
2.2 Eerste gebruik van de RockPi	6
2.3 Inloggen met VNC	7
2.4 USB Stick voorbereiden	8
3 Klassen en objecten in C++	10
3.1 Werken met de RockPi.	10
3.1.1 Verbinding opzetten met de RockPi	10
3.1.2 Het aansturen van een LED.	12
3.2 Het werken vanuit Visual Studio Code	14
3.3 Het werken met de DDD visuele debugger.	18
4 Afgeleide klassen en objecten in C++	22
4.1 De Klasse SingleLed	24
4.2 De Klasse RGBLed	26
5 Compositie en aggregatie.	30
5.1 De klasse Tijdsduur.	31
5.2 De klasse Logled	33
6 Het gebruik van Exceptions bij meerdere objecten.	37
6.1 Exceptions	37
6.2 Toepassen van 1 op 0..5 multipliciteit.	38
7 Introductie in unit testing	41
7.1 Deel 1, introductie	41
7.2 Deel2, een eigen test	44
Appendices	45
A Installeren van de practicum omgeving	47
A.1 De VNC viewer	47
A.2 Het gebruik van 'Visual Studio Code'	47
B Tips&Tricks&Troubleshooting voor Linux omgeving	54
B.1 Troubleshooting	54
B.2 Tips	54
B.3 Linux commando's (zie ook tips in B.2)	55

B.4 Uitleg GPIO poorten in Linux	56
--	----

1 Weektabel

In tabel 1.1 wordt de planning van het practicum weergegeven.

Hierbij is rekening gehouden met eventuele lesuitval in verband met feestdagen.

Zorg dat je op schema loopt. Je mag altijd vooruit werken!

lesweek	Opdracht	Practicum dictaat	Herkansing opdracht
10	Introductie in Visual Studio Code en de klasse Led.	Hoofdstuk 3 tot 3.3	Week 12
11	LED objecten, zichtbaar in de DDD debugger	Hoofdstuk 3.3	Week 13
12&13	Afgeleide klassen en objecten	Hoofdstuk 4	Week 14
14	Compositie en aggregatie	Hoofdstuk 5	Week 15
15&16	Exceptions	Hoofdstuk 6	Week 17
17	Unit testing	Hoofdstuk 7	Week 18
18	Uitloop		

Tabel 1.1: Planning van het practicum.

2 Inleiding

Voor het practicum OOPR1 zijn (ongeveer) 25 RockPi's beschikbaar. Dit zijn Raspberry Pi achtige ‘Single Board Computers’ die met een Linux variant werken, in dit geval Debian 10. De RockPi's zijn aangeschaft omdat Raspberry Pi's niet leverbaar (of te duur) waren. Tijdens het practicum gebruik je een RockPi van school. Je kunt deze alleen op school gebruiken, je mag hem niet meenemen naar huis. Dat is dan ook het belangrijkste nadeel.

Het gebruik van de RockPi heeft een aantal voordelen:

- Je hoeft zelf niets aan te schaffen
- Je werkt onder gecontroleerde omstandigheden: de verstrekte RockPi bevat alles wat nodig is voor het practicum. Dat betekent dat jij- of de docent niet eerst moet puzzelen om de omgeving aan de praat te krijgen.

Bij het practicum heb je *je eigen* USB stick nodig om je bestanden op te zetten (*geen gedeelde!*). Een USB stick van 1GB is goed genoeg (128+ MB). Groter mag, maar is zinloos.

- Je werkt op je eigen **USB stick** en niet op het bestandssysteem van de RockPi.
- Na afloop van het practicum lever je de RockPi in zonder daar bestanden op achter te laten.
- Ga er vanuit dat de RockPi's na een practicum gewist worden!

Dit is géén security practicum, de RockPi's zijn slecht beveiligd.

Ga dus niet klooien en klieren op de RockPi van iemand anders.

Als je overlast veroorzaakt zal de docent je uit het practicum verwijderen!!

2.1 Voorbereiding - Software installeren

Zorg dat je vóór het practicum de benodigde software geïnstalleerd hebt. Zie Bijlage A voor software installatie.

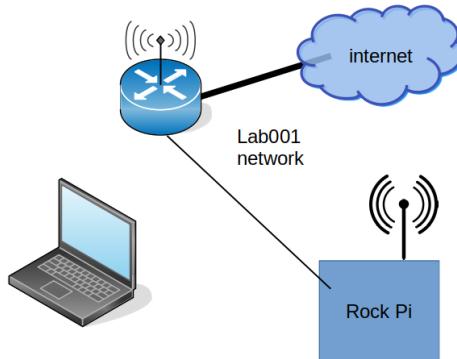
2.2 Eerste gebruik van de RockPi

We gaan er van uit dat je de RockPi in lokaal D2.001 of D2.003 van HHS Delft gebruikt.

De RockPi is al ingesteld voor Wi-Fi netwerk in D2.001: **Lab001**.

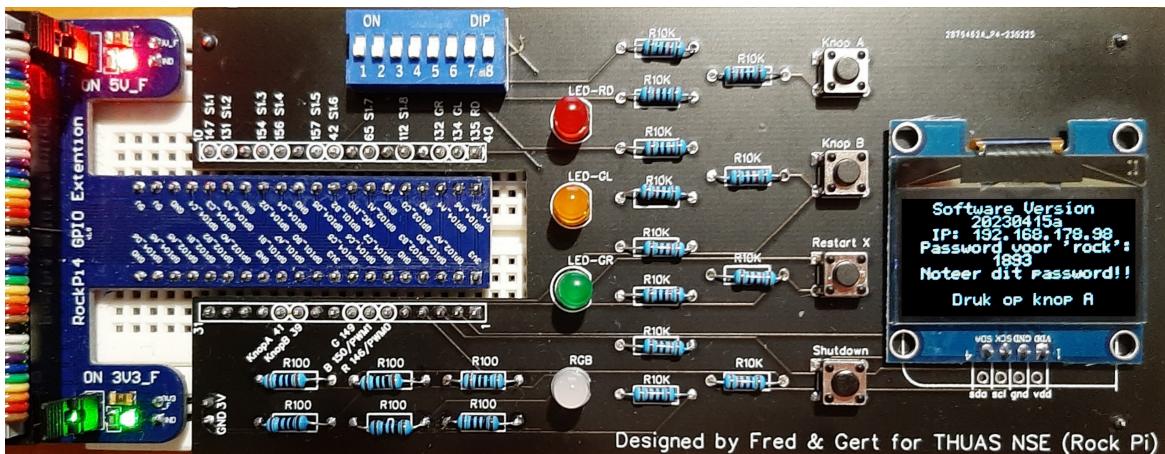
Log ook met je laptop in op dit netwerk. Het password is **Lab001WiFi**.

In Figuur 2.1 wordt weergegeven hoe de RockPi op het lab netwerk is aangesloten.



Figuur 2.1: De RockPi in het lab netwerk

Sluit de USB-C voedingskabel aan zodat de RockPi gaat opstarten (op de RockPi gaan een groene led branden). Verder hoef je nog niets aan te sluiten!



Figuur 2.2: Het uitbreidingsbord van de RockPi

Controleer de zelftest: Om zeker te weten dat het bovenstaande bordje OK is gaat de RockPi alle leds even aan en uitzetten.

Als je de RockPi in het lab (D2.001) aanzet, verschijnt op het oled display een IP adres van het Lab001 Wi-Fi netwerk (zie Figuur 2.2). Nu kun je via **SSH** en/of **VNC** een verbinding maken met dit IP adres en inloggen op de RockPi.

De username is **rock**, het password voor deze sessie wordt weergegeven op het display, zie Figuur 2.2 (elke keer als de RockPi opnieuw opstart krijg je een nieuw password). Mocht er een sudo password gevraagd worden, dan is dit hetzelfde password.

Overigens kun je ook een ethernet kabel aansluiten. Zorg wel dat je PC op hetzelfde netwerk is aangesloten. Als VNC het niet wil doen, maar SSH wel, dan zitten je PC en

de RockPi mogelijk op verschillende Wi-Fi access-points. Als VNC niet werkt, maar je RockPi en laptop zitten wel op hetzelfde netwerk, dan kun je het knopje 'Restart_X' indrukken, dit herstart de grafische schil op de RockPi.

Zodra je bent ingelogd met VNC, kun je je USB stick aansluiten en configureren. Het maakt niet uit welke USB poort je gebruikt.

Als je klaar bent met je practicum, moet je de RockPi netjes afsluiten. Door op het knopje 'Shutdown' te drukken, wordt het operating system netjes afgesloten zodat het bestandssysteem niet corrupt raakt (wat kan gebeuren als je de RockPi gewoon uitzet).

'Knop_A' kun je gebruiken om tijdens het opstarten voor volledig GPIO gebruik te kiezen; druk de knop in voordat je de RockPi aanzet, en houd hem ingedrukt tot de LED zelftest klaar is. Dit is zo gedaan omdat wisselen tussen PWM gebruik (voor de rode en blauwe leds van de driekleuren led) en GPIO (alleen aan en uit, geen 'analoge' waarden) niet lukt zonder opnieuw op te starten.

Aan 'Knop_A', 'Knop_B' en de DIP switches (blauwe blokje) zijn verder geen functies toegewezen. Daar kun je zelf code voor schrijven.

2.3 Inloggen met VNC

Controleer even of het werkt. Je gaat dit later pas gebruiken.

Voer het IP adres zoals getoond op het oled display in op [VNC](#) (Figuur 2.2).

Log in op de RockPi en klik op het 'Terminal' icoon (rood omcirkeld in Figuur 2.3):



Figuur 2.3: Terminal icoon

Nu verschijnt het Terminal venster. Van hieruit kun je allerlei commando's invoeren (en tegelijk wordt o.a. schermresolutie ingesteld en wordt de screensaver uitgezet):

```
Terminal - rock@rockpi-4b: ~
File Edit View Terminal Tabs Help
Screen 0: minimum 320 x 200, current 1680 x 1050, maximum 8192 x 8192
HDMI-1 disconnected primary (normal left inverted right x axis y axis)
1680x1050 60.00 60.00
Pas de schermresolutie in vnc aan met (bijvoorbeeld) xrandr --fb 1920x1080
Om dit automatisch/permanent te doen, wijzig dit via nano ~/.bashrc
Werk uitsluitend in de /home/rock/Documents map.
Sla nergens anders bestanden op!
Deze Documents map staat op een USB stick met EXT4 bestandssysteem.
Zie de practicumhandleiding hoe je de USB stick voorbereidt.
Have fun... Gert
rock@rockpi-4b:~$
```

Figuur 2.4: Het Terminal scherm

2.4 USB Stick voorbereiden

LET OP!!

- Met onderstaande procedure verwijder je alle bestanden van je USB stick!
- Je hebt genoeg aan een [kleine USB stick](#) (ergens tussen 128MB en 4 GB).
- Je kunt hierna alléén met Linux op je USB stick kijken, niet meer met Windows!

Formatteer nu je USB stick met EXT4 filesystem. Dit is nodig omdat Linux d.m.v. attributen in het bestandssysteem aangeeft of een bestand uitvoerbaar ('executable') is. Dat is op een Windows compatible USB stick (met FAT of NTFS bestandsysteem) niet mogelijk. Kijk of je USB stick gezien wordt. Geef het commando **lsblk** en kijk of daar een device bij zit met sda in de naam (zie hieronder in de listing van lsblk). In dit geval is deze er. De partitie waar we gebruik van maken is /dev/sda1.

```
rock@rockpi-4b:~$ lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0    1   1.9G  0 disk 
└-sda1     8:1    1   1.9G  0 part /media/rock/9C02-C2F1
mmcblk1    179:0   0 115.2G  0 disk 
├-mmcblk1p1 179:1   0   3.9M  0 part 
├-mmcblk1p2 179:2   0     4M  0 part 
├-mmcblk1p3 179:3   0     4M  0 part 
├-mmcblk1p4 179:4   0   512M  0 part 
└-mmcblk1p5 179:5   0   7.8G  0 part /
mmcblk1boot0 179:32  0     4M  1 disk 
mmcblk1boot1 179:64  0     4M  1 disk 
mmcblk1rpmb 179:96  0     4M  0 disk 
}
```

Controle:

Als je het commando **mount** geeft, dan verwacht je in de uitvoer het device (de disk) /dev/sda1 te zien:

```
/dev/sda1 on /media/rock/9C02-C2F1 type vfat (rw,nosuid,nodev,relatime,
    uid=1000,gid=1000,fmask=0022,dmask=0022,codepage=936,iocharset=utf8,
    shortname=mixed,showexec,utf8,flush,errors=remount-ro,uhelper=udisks2
)
```

Hierboven zie je dat disk ge-'mount' is. Om te formatteren moet je eerst 'umount' doen:

umount /dev/sda1

Nu USB disk formatteren met ext4 filesystem (het *sudo* password is hetzelfde als het password van user *rock*; het password van het display):

sudo mkfs -t ext4 /dev/sda

Vervolgens disklabel 'Documents' instellen:

sudo e2label /dev/sda Documents

Trek nu de USB stick uit de RockPi en steek hem terug zodat deze ge-'mount' wordt onder de naam 'Documents'.

Als laatste gebruiker *rock* eigenaar maken van de USB stick:

sudo chown rock /home/rock/Documents

Hieronder zie je de output van bovenstaande commando's:

```
umount: /dev/sda#: No such file or directory
rock@rockpi-4b:~\$ sudo mkfs -t ext4 /dev/sda
mke2fs 1.44.5 (15-Dec-2018)
/dev/sda contains a vfat file system
Proceed anyway? (y,N) y
Creating filesystem with 491520 4k blocks and 123120 inodes
Filesystem UUID: 1b2275ec-6dbd-4f3b-8d41-feaf4da0c7b7
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

Als het formatteren en labelen klaar is, kun je testen of de USB stick correct gezien wordt:

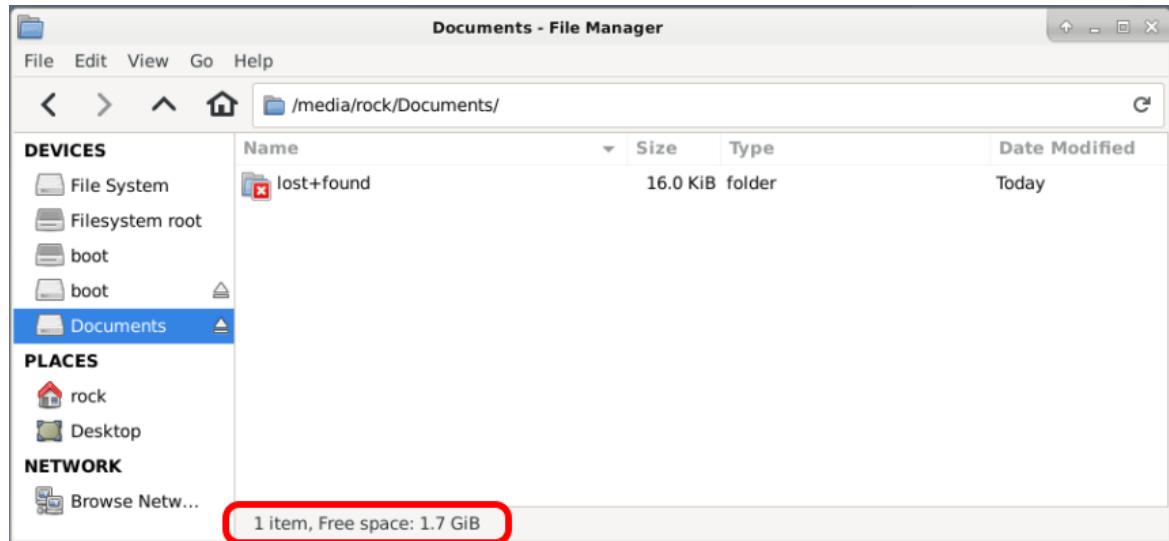
Verwijder de USB stick, wacht even en steek hem weer terug in een USB poort.
Klik in VNC op het 'File Manager' icoon (rood omcirkeld in Figuur 2.5):



Figuur 2.5: Terminal icoon

Ga in de 'File Manager' naar het 'Documents' device.

Als dit werkt, dan zie je onderin de statusbalk van de 'File Manager' hoeveel vrije ruimte je USB stick heeft (rood omcirkeld in Figuur 2.5)



Figuur 2.6: File Manager

Met de 'File Manager' kun je in het bestandssysteem van Linux kijken.

Bij het practicum begin je altijd in de home directory van gebruiker **rock** (/home/rock).

Je werkt *uitsluitend* in de map Documents (/home/rock/Documents).

Als je meer wilt weten over het Linux bestandssysteem, [klik dan hier](#).

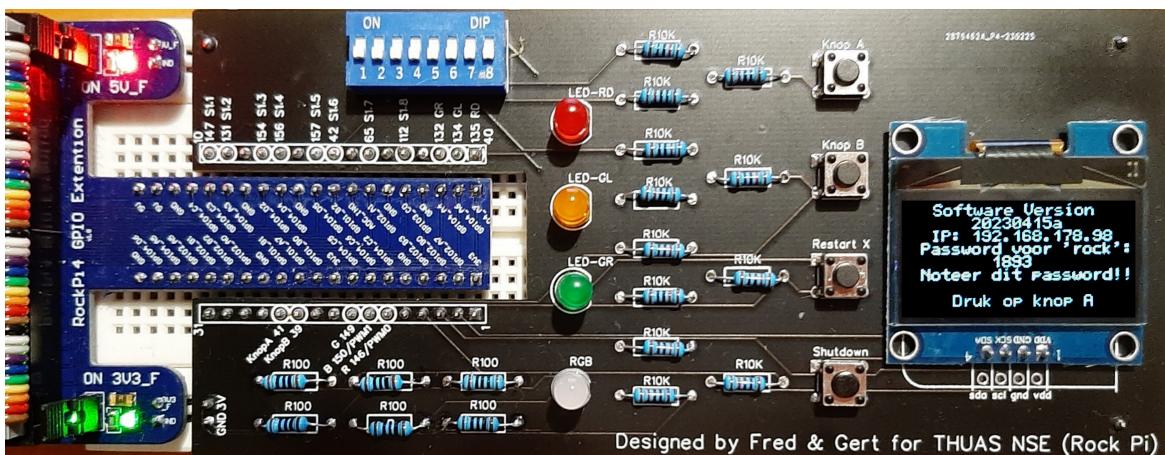
3 Klassen en objecten in C++

Als eerste wordt er verbinding gemaakt vanaf je laptop met de RockPi, waarna een LEDje aan- en uitgezet wordt. Vervolgens wordt door middel van een programma een aantal LEDs aangestuurd.

3.1 Werken met de RockPi.

3.1.1 Verbinding opzetten met de RockPi

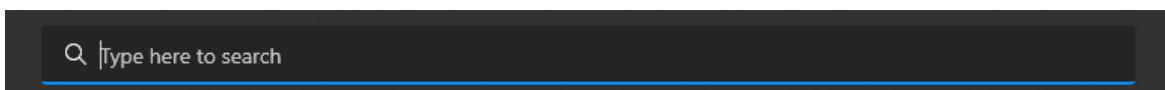
In eerste instantie wordt er verbinding gemaakt met de RockPi via het *SSH* commando. Het IP adres van de RockPi is af te lezen via het oled display zoals in Figuur 3.1 wordt



Figuur 3.1: De RockPi met het IP adres.

weergegeven. In het voorbeeld hierna wordt het IP adres *192.168.178.89* gebruikt.

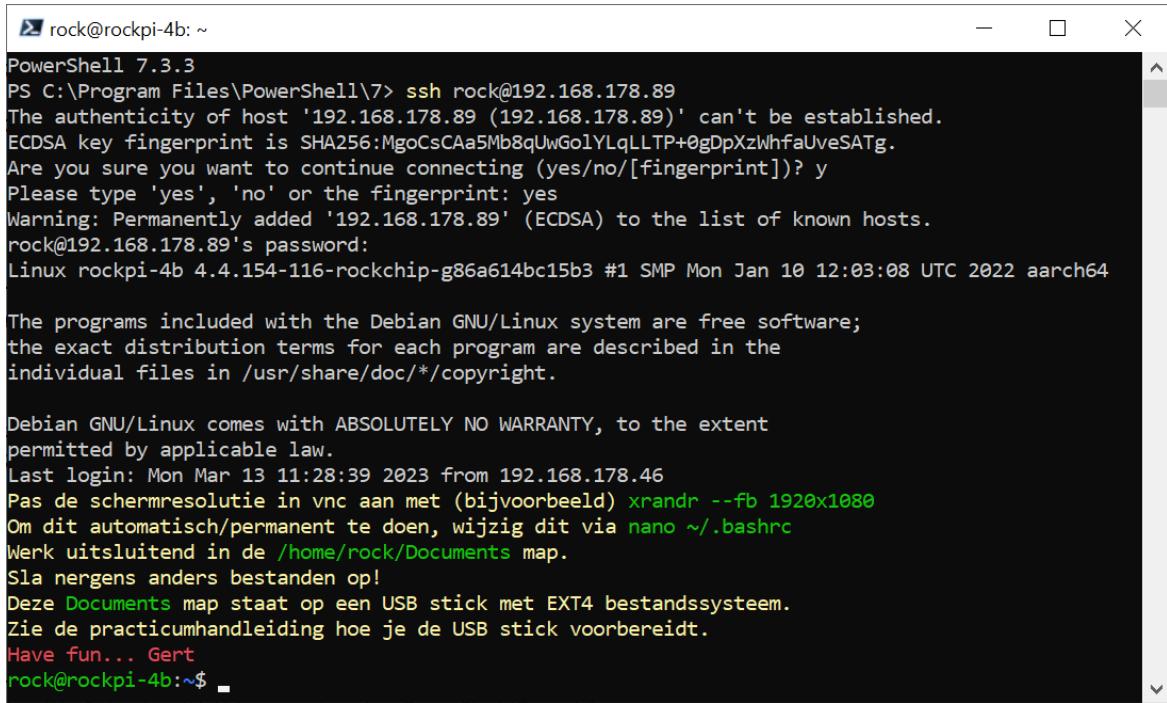
Open in Windows een terminal (b.v. een PowerShell, cmd prompt of een extern programma zoals b.v. [KiTTY](#)). We gebruiken voor nu PowerShell: Ga naar Windows Search: druk Windows+S (de knop met het Windows vlaggetje + de 'S'), zie Figuur 3.2 en tik vervolgens in: *powershell*. In de PowerShell kan het *SSH* commando met



Figuur 3.2: Zoekscherf in Windows

de username en het IP adres gegeven worden, zoals in Figuur 3.3 wordt weergegeven. Bij de eerste keer zal de melding komen of de host key moet worden opgeslagen, kies *yes*. Vervolgens zal om om het password gevraagd worden. Het eenmalige password voor deze sessie wordt weergegeven op het display (zie Figuur 3.1).

Wanneer het inloggen gelukt is, verschijnt een verhaaltje dat eindigt met de prompt van de RockPi, zoals in Figuur 3.3 te zien is. Vanaf nu werk je in een Linux ter-



```
rock@rockpi-4b: ~
PowerShell 7.3.3
PS C:\Program Files\PowerShell\7> ssh rock@192.168.178.89
The authenticity of host '192.168.178.89 (192.168.178.89)' can't be established.
ECDSA key fingerprint is SHA256:MgoCsCaa5Mb8qUwGolYLqLLTP+0gDpXzWhfaUveSATg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.178.89' (ECDSA) to the list of known hosts.
rock@192.168.178.89's password:
Linux rockpi-4b 4.4.154-116-rockchip-g86a614bc15b3 #1 SMP Mon Jan 10 12:03:08 UTC 2022 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

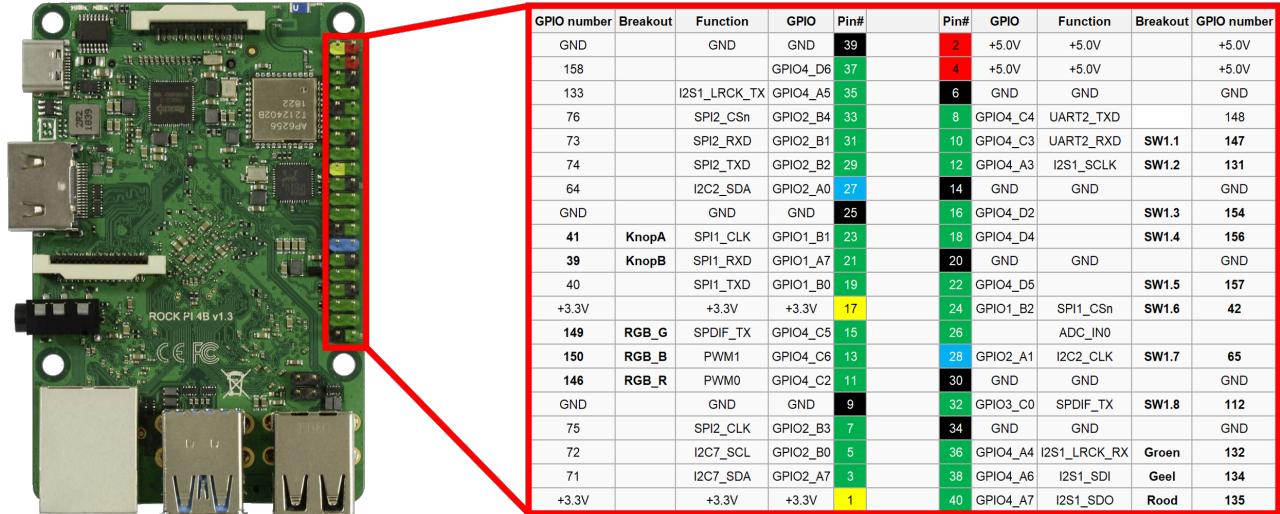
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar 13 11:28:39 2023 from 192.168.178.46
Pas de schermresolutie in vnc aan met (bijvoorbeeld) xrandr --fb 1920x1080
Om dit automatisch/permanent te doen, wijzig dit via nano ~/.bashrc
Werk uitsluitend in de /home/rock/Documents map.
Sla nergens anders bestanden op!
Deze Documents map staat op een USB stick met EXT4 bestandssysteem.
Zie de practicumhandleiding hoe je de USB stick voorbereidt.
Have fun... Gert
rock@rockpi-4b:~$ -
```

Figuur 3.3: Ingelogd in de RockPi

minal omgeving. Een listing van een directory kan opgevraagd worden met het `ls` commando (list files and directories), het veranderen van een directory kan gedaan worden met het `cd` commando (change directory) en het maken van een directory met het `mkdir` commando (make directory). Verdere Linux commando's en tips zijn te vinden in [de Bijlage](#).

3.1.2 Het aansturen van een LED.

Wanneer contact is gemaakt met de RockPi, zoals beschreven in hoofdstuk 3.1.1, kunnen de LEDs aangestuurd worden. De LED's zitten aangesloten op de GPIO ('General Purpose Input Output') connector van de RockPI, deze is te zien in Figuur 3.4. De buitenste kolom geeft het GPIO nummer aan waarop deze geprogrammeerd



Figuur 3.4: GPIO connector van de RockPI

kan worden. Zo zit de rode LED aangesloten op GPIO nummer 135. De **dikgedrukte** pinnen worden gebruikt met de printplaat. In de kolom **breakout** zie je wat op die pinnen zit.

Opdracht 1a, het eerste programmaatje op de RockPi

1. Plaats de [USB stick](#) in de RockPi.
2. Ga naar de directory van je USB stick
 - *cd Documents* (het hele pad is */media/rock/Documents*)
3. Maak een directory *ledje* aan en ga daar na toe.
 - *mkdir ledje*
 - *cd ledje*
4. Download files *gpiofuncties.h*, *gpiofuncties.cpp*, *test.cpp* van Github. Het main programma (*test.cpp*) wordt in Listing 3.1 weergegeven
 - *git clone https://github.com/JohnVi-hhs/oop.git*
 - *cd oop*
 - *ls -l*
 - *nano test.cpp* Druk Ctrl+X om nano weer te sluiten.

```

rock@rockpi-4b:~/Documents$ cd Documents
rock@rockpi-4b:~/Documents$ mkdir ledje
rock@rockpi-4b:~/Documents$ cd ledje
rock@rockpi-4b:~/Documents/ledje$ git clone https://github.com/JohnVi-hhs/oop.git
Cloning into 'oop'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 57 (delta 18), reused 29 (delta 6), pack-reused 0
Unpacking objects: 100% (57/57), done.
rock@rockpi-4b:~/Documents/ledje$ cd oop
rock@rockpi-4b:~/Documents/ledje/oop$ ls -l
total 16
-rw-r--r-- 1 rock rock 258 Mar 16 21:25 README.md
-rw-r--r-- 1 rock rock 1119 Mar 16 21:25 gpiofuncties.cpp
-rw-r--r-- 1 rock rock 91 Mar 16 21:25 gpiofuncties.h
-rw-r--r-- 1 rock rock 530 Mar 16 21:25 test.cpp
rock@rockpi-4b:~/Documents/ledje/oop$ nano test.cpp

```

Figuur 3.5: Schermuitvoer na bovenstaande opdrachten

```

#include <unistd.h>
#include <iostream>
#include "gpiofuncties.h"

using namespace std;
#define RODELED 135

int main() {

    cout<<"Hi_NSE"<<endl;
    int b=setPinOpOutput(RODELED); //return waarde of het gelukt is.
    if(b == 0) { //if(!b) mag ook.
        cout<<"Fout je bedankt"<<endl;
        return 0;
    }
    cout<<"b=_ "<<b<<endl; //return waarde of het gelukt is.
    b=setPinWaarde(RODELED,1); //Zet de rode LED aan.
    usleep(1000000);
    b=setPinWaarde(RODELED,0); //Zet de rode LED uit.
    cout<<"einde"<<endl;
}

```

Listing 3.1: Zet LED aan en uit

5. Compileer en run het testprogramma.

- Compileren van het programma.

*g++ -g3 *.cpp -o tst*
 optie *-g3* heeft te maken met debug mogelijkheden.
 Optie *-o* geeft een naam aan de output file (*tst*)

- Voer het zojuist gecompileerde programma *tst* uit.
./tst

Resultaat:

De rode LED gaat 1 seconde aan.

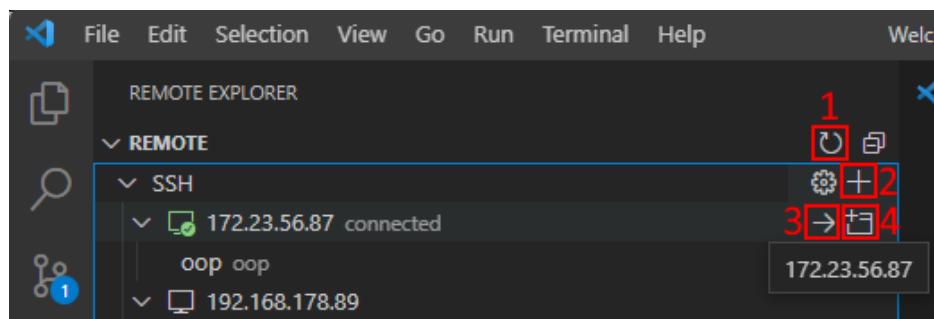
6. Pas het programma zodanig aan, zodat eerst de groene LED aangaat, daarna de gele LED en vervolgens de rode LED.

In de linux terminal, kunnen verschillende editors gebruikt worden, waarvan [nano](#) één van de meest gebruikte is, een ander beroemde/beruchte editor is [vim](#). Als je nano gebruikt: druk Ctrl+O om op te slaan en Ctrl+X om af te sluiten. Bij vim, druk/tik `:help`

3.2 Het werken vanuit Visual Studio Code

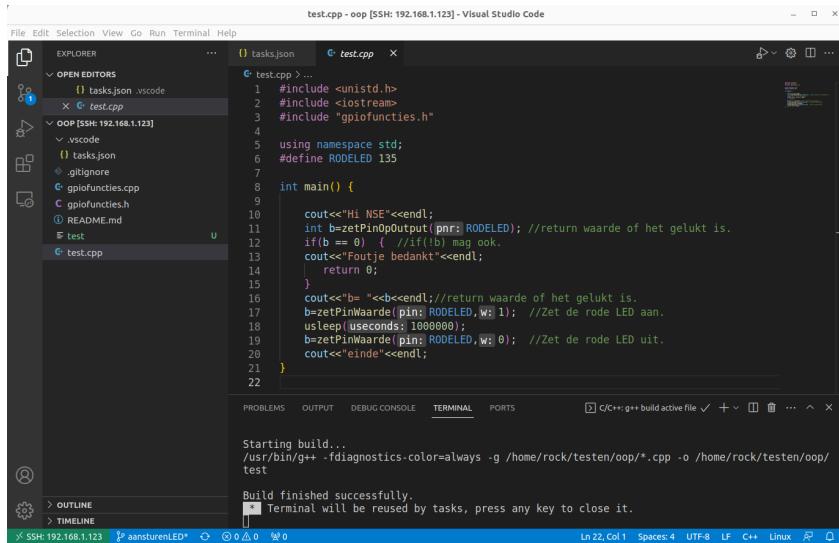
Hierbij wordt vanuit 'Visual Studio Code' (VSC) gewerkt, en wordt de klasse Led geïmplementeerd. Er wordt vanuit gegaan dat VSC geïnstalleerd is zoals in appendix [A.2](#) beschreven staat.

1. Start VSC op en maak verbinding met de RockPi.
 - Klik op remote Explorer . De IP nummers van de remote system(en) die al eerder gebruikt zijn worden zichtbaar, zoals te zien is in Figuur [3.6](#). Als dit niet zo is, klik dan op [1] om de lijst te verversen, als je een nieuw systeem wilt toevoegen, klik [2].



Figuur 3.6: Remote Explorer van VSC

- Maak nu verbinding met de RockPi door te klikken op [3] of [4] ([3] = in hetzelfde venster), [4] = nieuw venster).
2. Nadat een verbinding gemaakt is met de RockPi, open een remote folder door op te klikken (of druk **Ctrl+K Ctrl+O**) en ga naar de directory van de vorige opdracht: `/home/rock/Documents/ledje/oop/`
Aan de linkerkant worden de files zichtbaar en onderaan een statusbalk met informatie. Dit is te zien in Figuur [3.7](#)



Figuur 3.7: VSCode in de gewenste directory

Klik op select folder en selecteer de folder waarin de files staan.

- Selecteer in het 'Explorer' paneel de file *test.cpp* en compileer de files¹ (Terminal → Run Build Task... of **Ctrl+Shift+B**). (*Als je rechtsonder in het venster een melding krijgt over CMake, klik deze dan weg, we gebruiken geen CMake.*)
- Klik links van regelnummer 10, er verschijnt een rood bolletje, dit is een breakpoint.
- Start het Debuggen: druk **F5** of selecteer bij het pijltje rechtsboven de Debug optie en klik op de pijl (of Run → Start Debugging), De debugger wordt gestart en een scherm als in Figuur 3.8 verschijnt. (*Als je een foutmelding krijgt: zorg dat je scherm eruit ziet als in Figuur 3.7 en probeer opnieuw Debug te starten.*)

Met de debugknoppen rechtsboven kan nu stap voor stap het programma doorlopen worden (naar uitleg, druk **Alt+←** om hier terug te komen).

Doorloop het programma stap voor stap met 'Step Over' (**F10**), zodat de LED ook daadwerkelijk bij een stap aan- en uitgaat.

3. Bij deze opdracht wordt de eerste klasse gemaakt.

- Maak een nieuwe terminal aan (Terminal → New Terminal of **Ctrl+Shift+`**) en maak een nieuwe directory aan:

```
mkdir ~/Documents/opdrLedH gevuld door cd ~/Documents/opdrLedH
```
- clone de volgende code:

```
git clone --branch opdrLedH https://github.com/JohnVi-hhs/oop.git
```
- Sluit in VSC de huidige folder (**Ctrl+K F**) en open de folder

¹Compileren lukt niet: Controleer of de C++ extension () is geïnstalleerd en is geactiveerd.

```

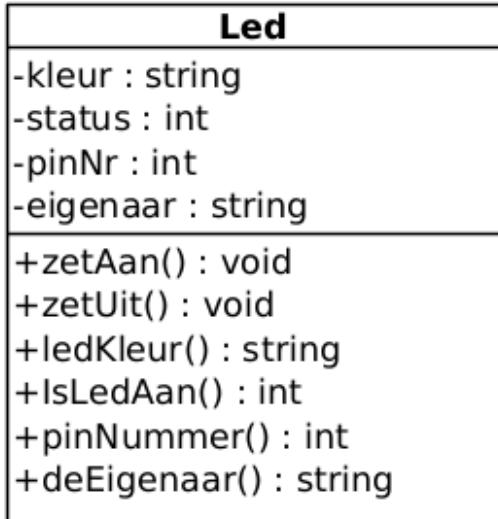
File Edit Selection View Go Run Terminal Help
No Con Search Replace ...
tasks.json test.cpp M ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
SSH: 192.168.1.123 aansturenLED* 0 0 0 0 0 0
Ln 10, Col 1 Spaces: 4 UTF-8 LF C++ Linux

```

Figuur 3.8: Start van de debug sessie.

/home/rock/Documents/opdrLedH/oop/

- De UML notatie van de klasse **Led** wordt weergegeven in Figuur 3.9 en de headerfile in listing 3.2



Figuur 3.9: UML diagram van de klasse Led

```

class Led
{
public:
    Led(int);
    Led(int, string);
    Led(int, string, string);
    ~Led();
    void zetAan();
    void zetUit();
    string ledKleur() const;
    int isLedAan() const;
    int pinNummer() const;
    string deEigenaar() const;

private:
    string kleur;
    int pinNr;
    int status;
    string eigenaar;
};

```

Listing 3.2: LED declaratie file(.h)

Opdracht: Implementeer de Led.cpp zodat het hoofdprogramma van listing 3.3 zonder errors en warning gecompileerd en uitgevoerd kan worden.

```
#include <unistd.h>
#include <iostream>
#include <string>
#include "Led.h"

using namespace std;
#define RODELED 135
#define GROENELED 132
#define GELELED 134

int main() {

    cout<<"Hi_NSE"<<endl;
    Led rood(RODELED, "Rood", "Pietje_Puk");
    Led geel(GELELED, "Geel");
    Led groen(GROENELED);

    groen.zetAan();
    usleep(1000000);
    groen.zetUit();
    geel.zetAan();
    usleep(1000000);
    geel.zetUit();
    rood.zetAan();
    usleep(1000000);
    rood.zetUit();

    cout<<"einde"<<endl;
}
```

Listing 3.3: Hoofdprogramma om de LED uit te testen

Opdracht: We gaan hierbij stap voor stap het programma doorlopen (debuggen), waarbij de inhoud van de objecten van de klasse Led wordt getoond.

- a We laten je dit nu eerst in VSC zien, in het volgende hoofdstuk doen we hetzelfde met de DDD debugger.
 - i Zet een breakpoint op regel 13, de regel met *Led rood (RODELED, "Rood", "Pietje Puk");* Als het goed is staat nu vóór de 13 een rood bolletje.
 - ii Zorg dat in het 'Explorer' paneel *test.cpp* is geselecteerd.
Start nu het debuggen (druk **F5**). Het programma wordt uitgevoerd tot het breakpoint. Er verschijnt een pijl over het breakpoint.
 - iii In het linker paneel bovenaan zie je 'VARIABLES' met daaronder de locale variabelen. Klap 'rood' en 'geel' open, je kijkt nu naar gedeclareerde- maar nog niet-geïnitialiseerde objecten.
 - iv Klik/druk op 'Step Over' (**F10**). Regel 13 wordt uitgevoerd.
Beantwoord: Wat zie je bij 'rood' veranderen?
De pijl komt voor de regel *Led geel(GELELED, "Geel");* te staan.

- v Klik/druk op 'Step Into' (**F11**), met dit commando wordt naar de constructor van de klasse Led gestapt.
- vi Nu zie je links bij 'VARIABLES' andere variabelen; *pin* en *kleur*.
Beantwoord: Waar komt de waarde van deze variabelen vandaan?
- vii Klik 'this' open en druk 'Step Over' (**F10**).
(*TIP*: Zorg dat er een commando in de constructor staat.)
Beantwoord: Wat zie je in/onder 'this' veranderen? Waardoor komt dat?
- viii Doorloop het programma verder stap voor stap; experimenteer met 'Step Over' (**F10**), 'Step Into' (**F11**) en 'Step Out' (**Shift+F11**)

3.3 Het werken met de DDD visuele debugger.

Bij dit onderdeel van de opdracht wordt gewerkt met een grafische debugger. Hiermee wordt een grafische weergave gedaan wat een object is en wat een object van een afgeleide klasse inhoudt (dit laatste komt in opgave 2 aan de orde). Verder worden de associaties tussen objecten duidelijk weergegeven (dit wordt in week 4 en 5 gedaan).

De grafische debugger die gebruikt wordt is de DDD debugger (Data Display Debugger)

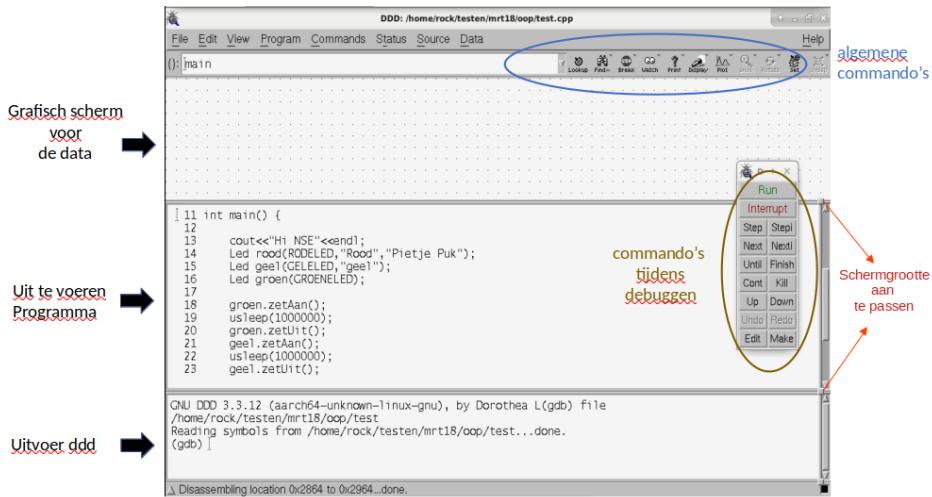
Een paar handige links hierbij zijn:

- DDD manual in [html](#) en [pdf](#)
- [GNU DDD project](#)
- [taufanlubis.wordpress.com](#)
- [Swarthmore](#)
- [linuxfocus](#)

We gaan nu met de DDD debugger werken. Om met DDD te kunnen werken hebben we een grafische omgeving nodig. Een handige methode is om de VNC viewer van de host te gebruiken. Deze viewer toont de grafische uitvoer van de RockPi. Open in de VNC een terminal, ga naar de directory die in de vorige opdracht gemaakt is (waar de files test.cpp, Led.cpp en Led.h *test* staan) en start de DDD debugger op: *ddd test*, vervolgens wordt het scherm, zoals weergegeven in Figuur 3.10, getoond.

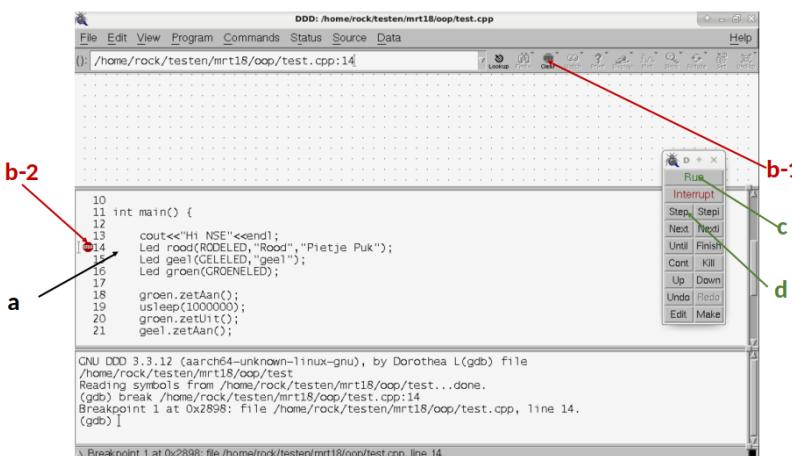
Opdracht: We gaan hierbij stap voor stap het programma doorlopen, waarbij de inhoud van de objecten van de klasse Led wordt getoond.

- a Als eerste een korte kennismaking met de DDD debugger.
 - i Zet de cursor op de regel *Led rood (RODELED, "Rood", "Pietje Puk");* (regelnummer 13)



Figuur 3.10: het DDD opstartscherf.

- ii Klik op *Break* (bij ‘algemene commando’s‘, zie Figuur 3.10). Voor het begin van de regel verschijnt nu een breakpoint (een rood STOP-bordje).



Figuur 3.11: het zetten van een breakpoint.

- iii Klik op *Run* (bij ‘commando’s tijdens debuggen‘), het programma wordt uitgevoerd tot het breakpoint.
Er verschijnt een groene pijl voor het breakpoint.
- iv Klik op *Next*, de regel wordt uitgevoerd (dit doet hetzelfde als de ‘step over’ bij VSC).
De groene pijl komt voor de regel *Led geel(GELELED, "Geel")*; te staan.
- v Klik op *Step*, met dit commando wordt in de constructor van de klasse *Led* gestapt (dit doet hetzelfde als de ‘step into’ bij VSC).
Doorloop het programma verder stap voor stap.

- b Het zichtbaar maken van de inhoud van de objecten.
- i Start de DDD debugger op (*ddd test*):
 - ii Zet een breakpoint op de regel *groen.zetAan()*;
 - iii Klik met de linkermuisknop op variabele *rood*. Klik of 1) op ‘Display’ in balk b-1 van Figuur 3.11, of 2) Klik met de rechtermuisknop (*hou ingedrukt!*) op *rood* en vervolgens op Display.
Het object *rood* van de Klasse Led wordt nu weergegeven.
 - iv Doe hetzelfde met de variabele *geel* en *groen*.
 - v Als het goed is ziet de debugger er ongeveer uit zoals Figuur 3.12, alleen met je **eigen naam**.

*Als je de grootte van de panelen in het DDD venster wilt aanpassen, zie de pijlen ‘Schermgrootte aan te passen’ in Figuur 3.10.
Klik op zo’n klein blokje (muis ingedrukt houden) en versleep deze.*
 - vi Ga met het Step commando de methode *zetAan* van *groen* in.
 - vii Ga met het Step commando de methode *zetAan* van *geel* in. Je ziet dat de methode hetzelfde is, alleen de attributen hebben een andere waarde.
- c Maak een screenshot die lijkt op Figuur 3.12 alleen met je **eigen naam** in plaats van ”Pietje Puk” en upload deze op Brightspace.
- d Plaats de screenshot in je portfolio onder hoofdstuk programmeren \Rightarrow subhoofdstuk 3.3. Laat de opdracht aftekenen met onder andere een zichtbaar screenshot.

DDD: /home/rock/testen/mrt18/oop/test.cpp

File Edit View Program Commands Status Source Data Help

(0: gee1)

1: rood
kleur = "Rood"
pinNr = 135
status = 0
eigenaar = "Pietje Puk"

2: gee1
kleur = "geel"
pinNr = 134
status = 0
eigenaar = "Anoniem"

3: groen
kleur = "Kleurloos"
pinNr = 132
status = 0
eigenaar = "Anoniem"

Run
Interrupt
Step StepI
Next NextI
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make

```

6 using namespace std;
7 #define RODELED 135
8 #define GROENELED 132
9 #define CELELED 134
10
11 int main() {
12
13     cout<<"Hi NSE"<<endl;
14     Led rood(RODELED, "Rood", "Pietje Puk");
15     Led gee1(GELELED, "Gee1");
16     Led groen(GROENELED);
17
18     groen.zetAan();
19     usleep(1000000);
20     groen.zetUit();
21     gee1.zetAan();
22     usleep(1000000);

```

Hi NSE

Breakpoint 1, main () at /home/rock/testen/mrt18/oop/test.cpp:18
(gdb) graph display rood
(gdb) graph display gee1
(gdb) graph display groen
(gdb) [

Display 2: gee1 (enabled, scope main, address 0x7fffffff028)

Figuur 3.12: Weergave van drie objecten van de klasse Led.

4 Afgeleide klassen en objecten in C++

Bij deze opdracht kijken we hoe bij OO componenten Overerving en Polymorfisme kunnen testen met een debugger.

Deze opdracht bestaat uit de deelopdrachten A en B. Er moeten 6 screenshots gemaakt worden die op Brightspace moeten worden geüpload. Alle deelopdrachten moet je laten aftekenen door de docent.

Er zijn diverse soorten LED's, zoals in Figuur 4.1 te zien zijn. Deze zijn:

- SingleLed: deze LED's hebben 1 kleur, twee pootjes en worden aan 1 poort op de RockPi aangesloten, zoals de rode, oranje en groene LED. Een voorbeeld wordt weergegeven in Figuur 4.1a
- DualLed: deze LEDs hebben 2 kleuren; rood en groen, drie pootjes en worden aan 2 poorten (1 per kleur) aan de RockPi verbonden. Een voorbeeld wordt weergegeven in Figuur 4.1b
- RGB Led: deze LED's hebben de kleuren rood, groen en blauw in 1 behuizing en hebben vier pootjes, waarvan 3 worden aangesloten op de poorten (1 per kleur) van de RockPi. De LED met de witte kleur op het practicum bord is een RGB LED. Een voorbeeld van een RGB LED wordt weergegeven in Figuur 4.1c



a een single LED



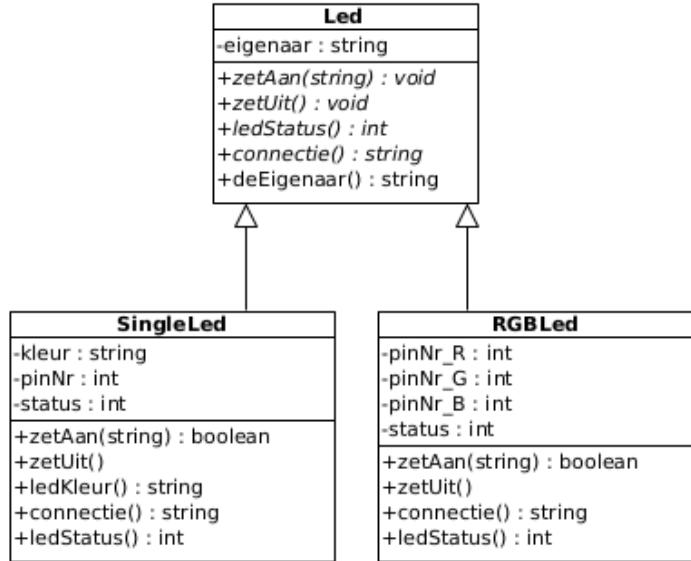
b een dual LED



c een RGB LED

Figuur 4.1: Verschillende type LEDs

De analist die de eisen voor de controller-software voor de LED controllers opstelt, heeft bedacht dat het in de toekomst mogelijk moet kunnen zijn om nieuwe LED types toe te voegen, bijvoorbeeld 3 kleuren LEDS. De controller software moet dus zo veel mogelijk onafhankelijk van het concrete LED type gemaakt worden. In Figuur 3.9 wordt de UML weergave getoond van zowel de SingleLed als van de RGBLed



Figuur 4.2: De afgeleide klassen *singleLed* en *RGBLed*.

De werking is als volgt.

- Een LED wordt aangezet door de methode bool `zetAan(string k)`; waarbij de parameter de kleur is die aangezet moet worden.
 - Wordt bij een groene LED "groen" meegegeven, wordt de LED aangezet en true geretourneerd.
 - Wordt bij een groene LED "rood" meegegeven, wordt de LED niet aangezet en wordt false geretourneerd.
- Een LED wordt uitgezet door de methode `void zetUit()`; Dit houdt in dat bij een RGBLed alle kleuren uitgezet worden.
- De methode `string connectie()`; geeft het gpioNummer van het aangesloten platform mee terug. In het geval van de RGBLed wordt een string mee teruggegeven met alle drie de gpioNummers gescheiden door een spatie.
- Doordat de status van LED's verschillend zijn (een singleLed kan alleen aan en uit terwijl bij de RGBLed kleur 1, kleur 2, kleur 3 of een combinatie van kleuren kan aan- en uitgaan), heeft elke afgeleide LED een eigen status.
- Omdat bij een RGBLed al bekend is wat de kleuren zijn (Rood, Groen en Blauw), hoeft de kleur van de LED niet opgevraagd te worden. In tegenstelling tot een singleLed die maar één kleur heeft, dit kunnen overigens verschillende kleuren zijn.

4.1 De Klasse SingleLed

Zoals in Figuur 4.2 wordt aangegeven is de singleLed een speciale vorm van Led. Dit is ook terug te zien in de code, zoals Listing 4.2 laat zien.

```
class Led
{
    public:
        /*
            Implementeer hier
            de constructor(s)
            en de methoden
        */

    private:
        string eigenaar;
};
```

Listing 4.1: LED declaratie file(.h)

```
class SingleLed : public Led
{
    public:
        /*
            Implementeer hier
            de constructor(s)
            en de methoden
        */

    private:
        string kleur;
        int pinNr;
        int status;
};
```

Listing 4.2: SingleLed declaratie file(.h)

Opdracht

- (a) Clone de code van git:

```
git clone --branch opdracht31 https://github.com/JohnVi-hhs/oop.git
```

- (b) Implementeer de klasse Led en SingleLed (zowel de .h als de .cpp file) en run de main code van Listing 4.3.

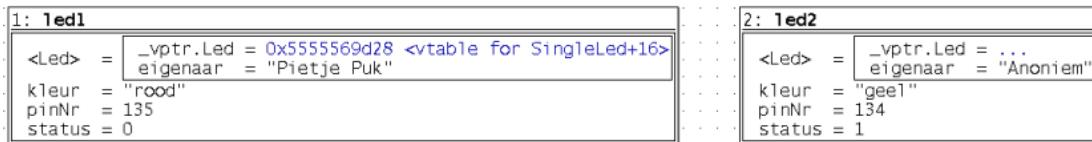
```
1 #define GROENELED 132
2 #define GELELED 134
3 void LedInfo(Led& l) {
4     cout<<"De_eigenaar_is:"<<l.deEigenaar()<<endl;
5     cout<<"De_Led_is_aangesloten_op_pinnen"<<l.connectie()<<endl;
6     cout<<"De_status_van_de_Led_is:"<<l.ledStatus()<<endl;
7 }
8
9 int main() {
10     SingleLed led1(RODELED, "rood", "Pietje_Puk");
11     SingleLed led2(GELELED, "geel");
12
13     Led &lrl1(led1);
14     lrl1.zetAan("rood");
15     usleep(1000000);
16     led2.zetAan("geel");
17     usleep(1000000);
18     led1.zetUit();
19     usleep(1000000);
20
21     LedInfo(led1);
22     LedInfo(led2);
23     LedInfo(lrl1);
24
25     led2.zetUit();
26     return 0;
27 }
```

Listing 4.3: main functie om de klasse SingleLed te testen.

De uitkomst van het programma is als volgt:

```
De eigenaar is:Pietje Puk
De Led is aangesloten op pinnen135
De status van de Led is:0
De eigenaar is:Anoniem
De Led is aangesloten op pinnen134
De status van de Led is:1
De eigenaar is:Pietje Puk
De Led is aangesloten op pinnen135
De status van de Led is:0
```

- (c)
- Start de VNC viewer op met hierin de DDD debugger.
 - Plaats een breakpoint op regel 21 van Listing 4.3 (`LedInfo(led1)`).
(Met **Alt + N** zet je de weergave van regelnummers aan.)
 - Run het programma, de debugger zal stoppen op regel 21.
 - Toon de objecten led1 en led2, zoals getoond in Figuur 4.3. De tekst achter



Figuur 4.3: De inhoud van objecten led1 en led2 .

`_vptr.led` kan verborgen worden door deze te selecteren en in de pull-down menu *hide All* te kiezen.

Verder is duidelijk te zien dat de base-klasse **Led**, met het attribuut `eigenaar`, een onderdeel is van de afgeleide klasse **SingleLed**.

- (d) Maak een screenshot van beide objecten in de DDD debugger, uiteraard met je **eigen** naam en plaatst deze met de **code** in je portfolio onder het hoofdstuk programmeren \Rightarrow subhoofdstuk 4.1.
Laat de opdracht aftekenen met de screenshots zichtbaar.

4.2 De Klasse RGBLed

Zoals te zien in het klassendiagram van Figuur 4.2 is de RGBLed ook een Led. Doordat bij een RGB LED de kleuren al bekend zijn, heeft deze klasse geen attribuut kleur. De klasse RGBLed heeft verder:

- Drie aansluitingen op de RockPi, namelijk voor elke kleur 1. Deze aansluitingen zijn:
 - 146 rood
 - 149 groen
 - 150 blauw
- De methode connectie geeft de 3 aansluitnummers als een string mee terug, waarbij de spatie een onderscheid maakt tussen de pin-nummers.
- De status geeft aan welke kleur aan is. Dit wordt gedaan door elke kleur een bit toe te wijzen. De toewijzing is als volgt:
 - bit 0 rood
 - bit 1 groen
 - bit 2 blauw

Opdracht

- (a) Implementeer de klasse RGBLed (zowel de .h als de .cpp file) en run de main code van Listing 4.4.

```
1 void LedInfo(Led& l) {
2     cout<<"De_eigenaar_is:_"<<l.deEigenaar()<<endl;
3     cout<<"De_Led_is_aangesloten_op_pinnen:_"<<l.connectie()<<endl;
4     cout<<"De_status_van_de_Led_is:_"<<l.ledStatus()<<endl;
5 }
6
7 int main() {
8     cout<<"Hi_NSE,_welkom_bij_opdracht_2"<<endl;
9
10    RGBLed kleuren(RGB_R,RGB_G,RGB_B, "Pietje_Puk");
11    kleuren.zetAan("rood");
12    usleep(1000000);
13    kleuren.zetUit();
14    kleuren.zetAan("groen");
15    usleep(1000000);
16    kleuren.zetUit();
17    kleuren.zetAan("blauw");
18    usleep(1000000);
19    kleuren.zetAan("groen");
20    usleep(1000000);
21    kleuren.zetUit();
22    kleuren.zetAan("rood");
23    kleuren.zetAan("groen");
24    usleep(1000000);
25    kleuren.zetUit();
26    kleuren.zetAan("rood");
27    kleuren.zetAan("blauw");
28    usleep(1000000);
29    kleuren.zetAan("groen");
30    usleep(1000000);
31    LedInfo(kleuren);
32    kleuren.zetUit();
33
34    cout<<"einde"<<endl;
35 }
```

Listing 4.4: main functie om de klasse SingleLed te testen.

De uitkomst van het programma wordt weergegeven op de volgende bladzijde.

```

Hi NSE, welkom bij opdracht 2
De eigenaar is: Pietje Puk
De Led is aangesloten op pinnen: 146 149 146
De status van de Led is: 7
einde

```

Plaats de code van de RGBLed (zowel de .h als de .cpp) file in je portfolio onder het hoofdstuk programmeren \Rightarrow subhoofdstuk 4.2.

- (b) In deze opdracht worden zowel de SingleLed als de RGBLed gebruikt. Verder worden objecten van de klasse **SingleLed** en **RGBLed** in de DDD debugger getoond. Compileer de code van Listing 4.5 en run deze.

```

1 void LedInfo(Led& l) {
2     cout<<"De_eigenaar_is:_"<<l.deEigenaar()<<endl;
3     cout<<"De_Led_is_aangesloten_op_pinnen:_"<<l.connectie()<<endl;
4     cout<<"De_status_van_de_Led_is:_"<<l.ledStatus()<<endl;
5 }
6
7 int main() {
8
9     cout<<"Hi_NSE,_welkom_bij_opdracht_2"<<endl;
10
11    RGBLed kleuren(RGB_R,RGB_G,RGB_B,"Pietje_Puk");
12    SingleLed led1(RODELED,"rood","Pietje");
13    Led& refL1(led1);
14
15    kleuren.zetAan("rood");
16    usleep(1000000);
17    led1.zetAan("rood");
18    usleep(1000000);
19    refL1.zetUit();
20
21    LedInfo(kleuren);
22    LedInfo(refL1);
23
24    cout<<"einde"<<endl;
25 }

```

Listing 4.5: main functie om de klasse SingleLed te testen.

De uitvoer ziet er als volgt uit:

```

Hi NSE, welkom bij opdracht 2
Informatie over de RGBLed:
De eigenaar is: Pietje Puk
De Led is aangesloten op pinnen: 146 149 146
De status van de Led is: 1
Informatie over de SingleLed:
De eigenaar is: Pietje
De Led is aangesloten op pinnen: 135
De status van de Led is: 0
einde

```

- (c) Start de VNC viewer, open een terminal, ga naar de directory waar de code van Listing 4.5 staat en run de DDD debugger. (ddd ./test)
- Zet een breakpoint op regel 21 van Listing 4.5, de regel met `LedInfo(kleuren);`
 - Run het programma, de debugger stopt bij het statement `LedInfo(kleuren);`
 - Display de objecten `kleuren` en `led1`. Het resultaat zal er ongeveer uitzien als Figuur 4.4, uiteraard alleen met je eigen naam. Hierbij is

1: kleuren		2: led1	
<Led>	= _vptr.Led = ...	<Led>	= _vptr.Led = ...
eigenaar	= "Pietje Puk"	eigenaar	= "Pietje"
pinNr_R	= 146	kleur	= "rood"
pinNr_G	= 149	pinNr	= 135
pinNr_B	= 150	status	= 0
status	= 1		

Figuur 4.4: De inhoud van de objecten `kleuren` en `led1`.

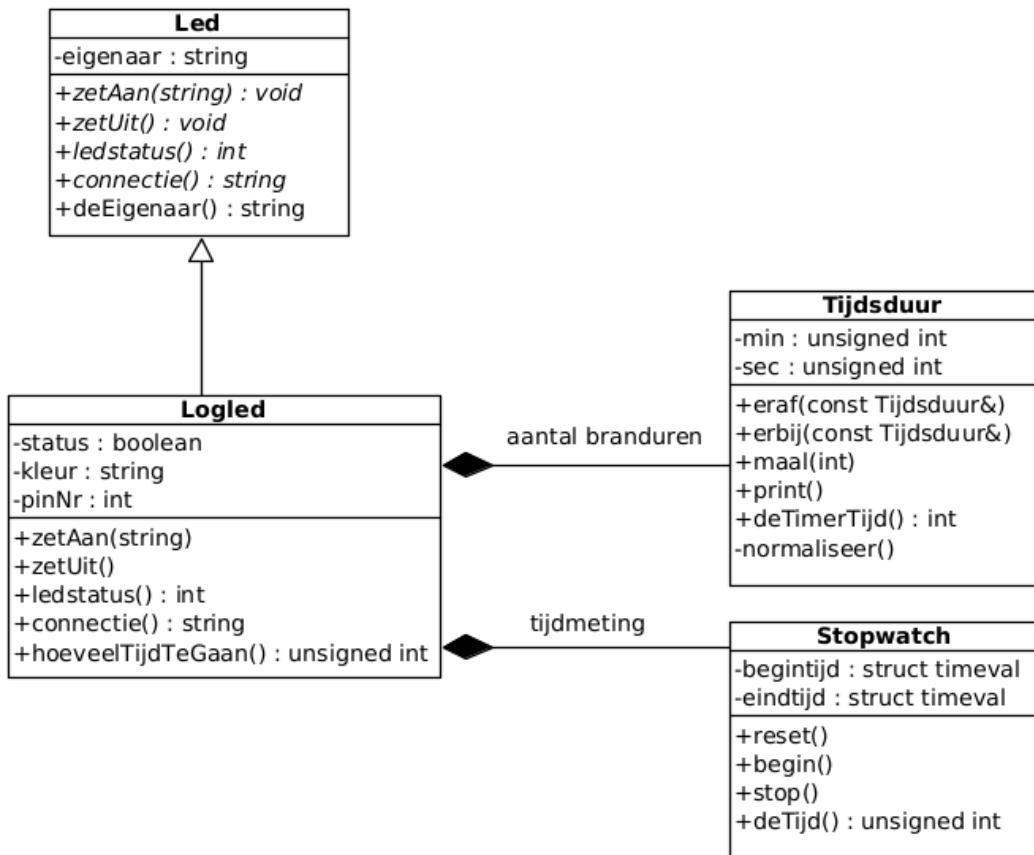
duidelijk te zien dat de `Led` een onderdeel is van de afgeleide klasse en dat beide afgeleide klassen van `Led`, ieder z'n eigen attributen heeft.

- Maak een screenshot van beide objecten en zet dit in je portfolio onder de code van `RGBLed`.
Upload je portfolio op Brightspace bij week 3.
Laat de opdracht aftekenen met de screenshots zichtbaar.

5 Compositie en aggregatie.

Bij deze opdracht worden aggregatie en compositie in C++ geïmplementeerd. Met behulp van de DDD debugger worden objecten en hun onderlinge verbindingen zichtbaar gemaakt. Hiervan zullen verschillende screenshots gemaakt moeten worden die opgenomen moeten worden in je portfolio. De portfolio moet aan het einde van de week geüpload worden op Brightspace. Tijdens het aftekenen moeten de screenshots zichtbaar zijn. Op de screenshots zal je **eigen naam** en **studienummer** moeten staan.

De opdracht bestaat uit een speciaal type LED, de Logled. De Logled heeft als extra een Stopwatch en een Tijdsduur. De klasse wordt weergegeven in Figuur 5.1. De



Figuur 5.1: De logled met timer en stopwatch.

Logled is speciaal ontwikkeld om ervoor te zorgen dat een maximale levensduur voor de LED kan worden ingesteld. Daarna moet een nieuwe Logled gekocht worden.

- Met de tijdsduur wordt bijgehouden hoeveel minuten de Logled nog aan mag.
- Met de stopwatch wordt elke keer gecheckt hoe lang de Logled voor dat moment aan is.

- De Led uit de vorige opgave kan hergebruikt worden. De stopwatch krijg je cadeau en is te downloaden van Github.
- De klasse Logled en Tijdsduur moet je zelf maken.

5.1 De klasse Tijdsduur.

We willen een ADT (Abstract Data Type), oftewel een 'zelfgedefinieerd datatype' maken waarin een tijdsduur in minuten en seconden kan worden opgeslagen. De totale tijd in seconden kan ook worden opgevraagd. We noemen dit zelf gedefinieerde datatype

Tijdsduur	
-min : unsigned int	
-sec : unsigned int	
+eraf(const Tijdsduur&)	
+erbij(const Tijdsduur&)	
+maal(int)	
+print()	
+deTimerTijd() : int	
-normaliseer()	

Figuur 5.2: weergave van de klasse *Tijdsduur*.

Tijdsduur. Figuur 5.2 geeft de klasse weer van Tijdsduur. Het ADT Tijdsduur kan in C++ als volgt gedeclareerd worden in de header file (tijdsduur.h):

```
#ifndef TIJDSUUUR_H
#define TIJDSUUUR_H

// De declaratie van de ADT Tijdsduur:
class Tijdsduur {
public:
//...
void eraf(Tijdsduur t);
//...

private:
int sec;
//...

};

#endif // TIJDSUUUR_H
```

Listing 5.1: de headerfile van de klasse *Tijdsduur*

De implementatie (`tijdsduur.cpp`) ziet er tot nu toe als volgt uit:

```
#include <iostream>
#include "tijdsduur.h"
#include <iomanip>
using namespace std;

// De definities van de memberfunctie van de ADT Tijdsduur, oftewel:
// de implementatie van de ADT Tijdsduur:
void Tijdsduur::eraf(Tijdsduur t) {
    sec-=t.sec;
    //...
}
```

Listing 5.2: de implementatiefie van de klasse *Tijdsduur*

Het hoofdprogramma (`testTijdsduur.cpp`) ziet er als volgt uit:

```
#include <iostream> // nodig voor cout (schrijven naar scherm)
#include <iomanip> // nodig voor setw (veldbreedte definieren )
#include "tijdsduur.h"
using namespace std;

int main() {
    Tijdsduur t1(3,50); // t1 is 3 minuten en 50 seconden
    cout<<"t1="; t1.print(); cout<<endl;
    const Tijdsduur kw(15); // kw is 15 seconden
    cout<<"kw="; kw.print(); cout<<endl;
    t1.erbij(kw); // Tel kw bij t1 op
    cout<<"t1="; t1.print(); cout<<endl;
    Tijdsduur t2(t1); // t2 is een kopie van t1
    t2.eraf(kw); // Trek kw van t2 af
    cout<<"t2="; t2.print(); cout<<endl;
    t2.maal(7); // Vermenigvuldig t2 met 7
    cout<<"t2="; t2.print(); cout<<endl;
    Tijdsduur t3(3,-122); // t3 is 3 minuten minus 122 seconden
    cout<<"t3="; t3.print(); cout<<endl;
    t3.eraf(t2);
    cout<<"t3="; t3.print(); cout<<endl;
    Tijdsduur t4(3,122); // t4 is 3 minuten plus 122 seconden
    cout<<"t4="; t4.print(); cout<<endl;
    cout<<"het totaal aantal seconde van t4=" <<t4.deTimerTijd()<<endl
    ;
    return 0;
}
```

Listing 5.3: de implementatiefie van het hoofdprogramma

De uitvoer moet dan zijn:

```
t1= 3 minuten en 50 seconden
kw = 15 seconden
t1 = 4 minuten en 5 seconden
t2 = 3 minuten en 50 seconden
t2 = 26 minuten en 50 seconden
t3 = 58 seconden
t3 = 0 seconden
t4 = 5 minuten en 2 seconden
```

Opdracht

- a De code van de implementatie van de klasse tijdsduur, zoals hierboven vermeld is, is verre van compleet. Download opg13.zip van Brightspace of clone deze:
`git clone --branch logled https://github.com/JohnVi-hhs/oop.git`

Vul de declaratie en de implementatie van de ADT genaamd Tijdsduur verder in. Zorg ervoor dat het hoofdprogramma (`testTijdsduur.cpp`) zonder warnings te compileren is en de gewenste uitvoer produceert. Zoek (indien nodig) inspiratie bij de in de les behandelde **class** Breuk. Tip: Zorg ervoor dat de opgeslagen seconden altijd ≥ 0 en < 60 zijn.

- b Voer zelf nog een aantal testen met Tijdsduur uit, bijvoorbeeld:

- Het testen op een negatieve tijd (een negatieve tijd bestaat niet).
- Het gebruik van de methode `int deTimerTijd()`.

- c Laat de opdracht aftekenen.

5.2 De klasse Logled

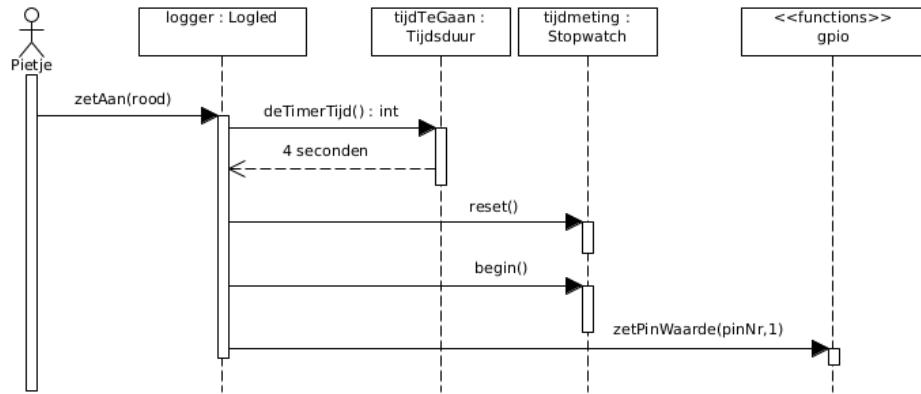
Zoals uit Figuur 5.1 blijkt heeft de klasse **Logled** twee compositie klassen, namelijk de klassen **Tijdsduur** en **Stopwatch**. Hierdoor kan het maximum aan minuten en seconden dat een Logled in totaal aan kan gaan, worden opgegeven. Stel een Logled wordt gecreëerd met de volgende regel:

```
Logled logger(135, "rood", "Pietje_Puk", 0, 4);
```

Dit houdt in dat een Logled wordt aangemaakt met als naam `logger`. De led wordt aangesloten op pin 135 en heeft de kleur rood. De eigenaar is "Pietje Puk". De maximale tijd dat een led in totaal aan kan gaan is 0 minuten en 4 seconden. De werking van de methoden `zetAan` en `zetUit` van Logled zijn als volgt:

- De methode `void zetAan("groen");`
 1. Er wordt gecontroleerd of de meegegeven kleur overeenkomt met de kleur van de led.
 - Nee: return.
 2. De tijd dat de led nog aan mag, wordt opgehaald.
 3. Opgehaalde tijd > 0 ?
 - (a) Reset de stopwatch.
 - (b) Begin met tijdmeting.
 - (c) Zet de led aan.

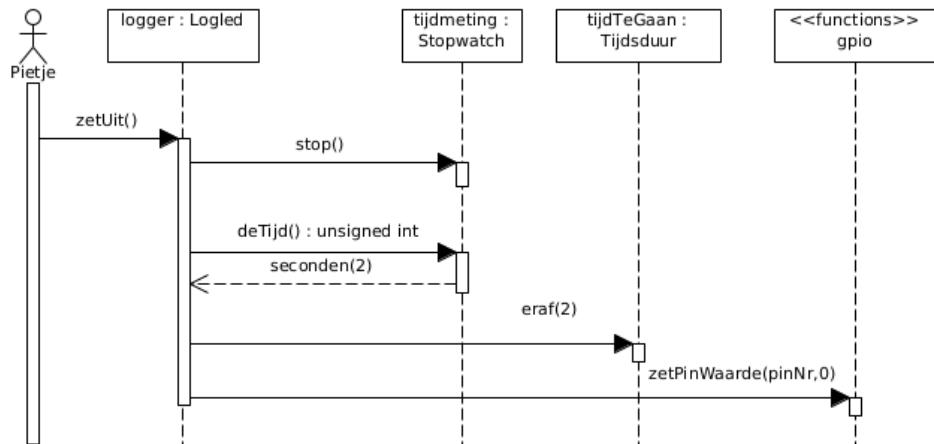
In het sequentiediagram van Figuur 5.3 wordt een scenario weergegeven wanneer de Logled aangezet wordt en deze nog 4 seconden te gaan heeft.



Figuur 5.3: De werking van de methode `zetAan`.

- De methode `void zetUit()` doet het volgende:
 - Stop de stopwatch.
 - Haal de stopwachttijd op.
 - Breng deze tijd in mindering bij tijdTeGaan.
 - Zet de led uit.

In het sequentiediagram van Figuur 5.4 wordt een scenario weergegeven wanneer de Logled wordt uitgezet nadat deze 2 seconden heeft aangestaan.



Figuur 5.4: De werking van de methode `zetUit`.

Opdracht

- a Implementeer de klasse Logled, zodat de code van listing 5.4 zonder compiler errors en warnings werkt.

```
1 #include <iostream> // nodig voor cout (schrijven naar scherm)
2 #include <unistd.h>
3 #include "Logled.h"
4
5 using namespace std;
6
7 #define RODELEDPIN 135
8 #define GROENELEDPIN 132
9
10#define TWEE_SEC 2000000
11#define DRIE_SEC 3000000
12
13 int main() {
14     //Een object van de klasse Logled met een maximale 'aan' tijd van
15     //2 seconde
16     Logled logger(RODELEDPIN, "rood", "Pietje_Puk", 0, 2);
17     logger.zetAan("rood");
18     usleep(DRIE_SEC); //wacht 3 seconden
19     logger.zetUit();
20     logger.zetAan("rood"); //led gaat niet meer aan.
21     return 0;
22 }
```

Listing 5.4: Een test van de klasse LogLed.

- b Pas de main() aan zodat deze eruit ziet als listing 5.5

```
1 int main() {
2
3     Logled logger(RODELEDPIN, "rood", "Pietje_Puk", 0, 4);
4     Logled sportled(GROENELEDPIN, "groen", "Bb", 0, 2);
5
6     logger.zetAan("rood");
7     sportled.zetAan("groen");
8     usleep(DRIE_SEC); //wacht 3 seconden
9     logger.zetUit();
10    sportled.zetUit();
11    usleep(TWEE_SEC); //wacht 2 seconden
12    logger.zetAan("rood");
13    sportled.zetAan("groen");
14    usleep(TWEE_SEC); //wacht 2 seconden
15    logger.zetUit();
16    sportled.zetUit();
17
18    return 0;
19 }
```

Listing 5.5: Twee objecten van de klasse LogLed.

Compileer het programma en laat het runnen. Plaats de code (.h en .cpp) file van de klasse **Logled** in je portfolio.

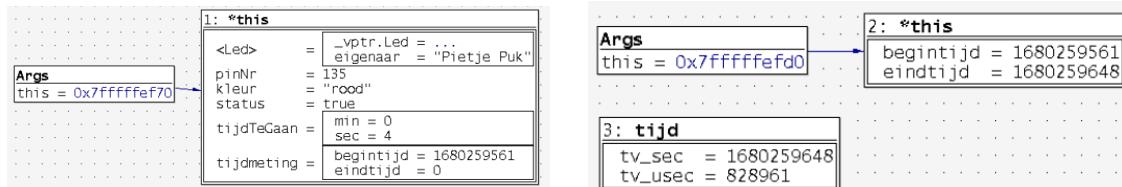
- c Open de VNC viewer, run de DDD debugger (ddd ./testlogled), zet een breakpoint bij het statement usleep (TWEEL_SEC) ; (regel 11 van listing 5.5) en



Figuur 5.5: Twee objecten van de klasse Logled

display beide objecten (logger en sportled) van de klasse **Logled**. Als het goed is krijg je iets te zien zoals Figuur 5.5, uiteraard met je **eigen** naam. Maak van de afbeelding een screenshot en plaats deze in je portfolio.

- d Voer het programma nog een keer uit en maak gedurende de uitvoer twee **screenshots** die lijken op Figuur 5.6. Indien Args niet zichtbaar is, kan deze zichtbaar gemaakt worden via Data → Display Arguments (ALT + U). Led op: De begintijd van 5.6a en 5.6b zijn hetzelfde, de stopwatch is immers een compositie van Logled. Bij Figuur 5.6b zie je de waarde van de attributen van het object van de



- a Logled met composities Tijdsduur en Stopwatch. b Van het object stopwatch, de attributen en lokale variabele tijd.

Figuur 5.6: Inhoud van het object Logled bij het aanroepen van de zetUit() en vervolgens stap voor stap naar methode stop() van stopwatch.

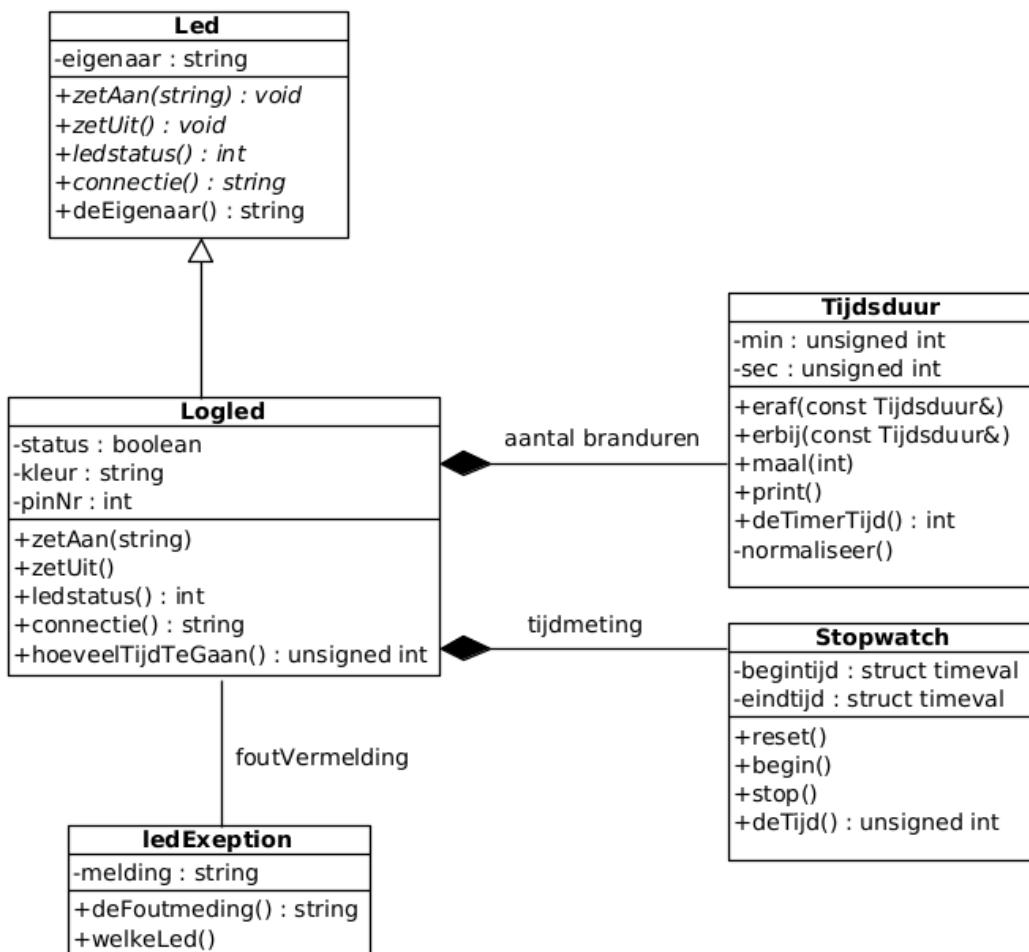
klasse *Stopwatch* waarbij **tijd** een lokale variabele is binnen de methoden *void begin()* en *void stop()* van de klasse *Stopwatch*.

6 Het gebruik van Exceptions bij meerdere objecten.

Bij deze opgave worden voor het eerst Exceptions toegepast. Deze worden vervolgens gebruikt in een multipliciteit situatie, bijvoorbeeld: 1 device heeft 0..5 leds.

6.1 Exceptions

Bij het aanzetten van de Logled (de methode void zetAan(string)) kunnen meerdere soorten fouten optreden. Fouten zoals niet aangesloten, geen tijd meer, verkeerde kleur. Daarom is besloten om exceptions te gaan toepassen. Dit wordt in Figuur 6.1 aangegeven. Hierbij wordt behalve de melding ook een verwijzing meege-



Figuur 6.1: De Logled die met exceptions kan gooien.

geven naar de led waarbij de fout optreedt.

Opdracht

1. Implementeer de klasse **LedException** (zowel de.h als de .cpp file).
2. Pas de klasse Logled aan, zodat de exceptions gegenereerd kunnen worden.
3. Wanneer de main van listing 6.1 laat runnen, wordt het volgende uitgeprint:
Verkeerde kleur. De ledkleur is:groen Aangesloten op pin:132

```
1 int main()
2 {
3
4
5     Logled logger(RODELEDPIN, "rood", "Pietje_Puk", 0, 2);
6     Logled sportled(GROENELEDPIN, "groen", "Bb", 0, 4);
7
8     try
9     {
10         logger.zetAan("rood");
11         sportled.zetAan("rood");
12     }
13     catch (LedException &le)
14     {
15         cout << le.defoutmelding() << "Aangesloten op pin:" << le.
16             welkeLed()->connectie() << endl;
17     }
18     usleep(DRIE_SEC);
19     logger.zetUit();
20     sportled.zetUit();
21
22     return 0;
}
```

Listing 6.1: Twee objecten van de klasse *LogLed*.

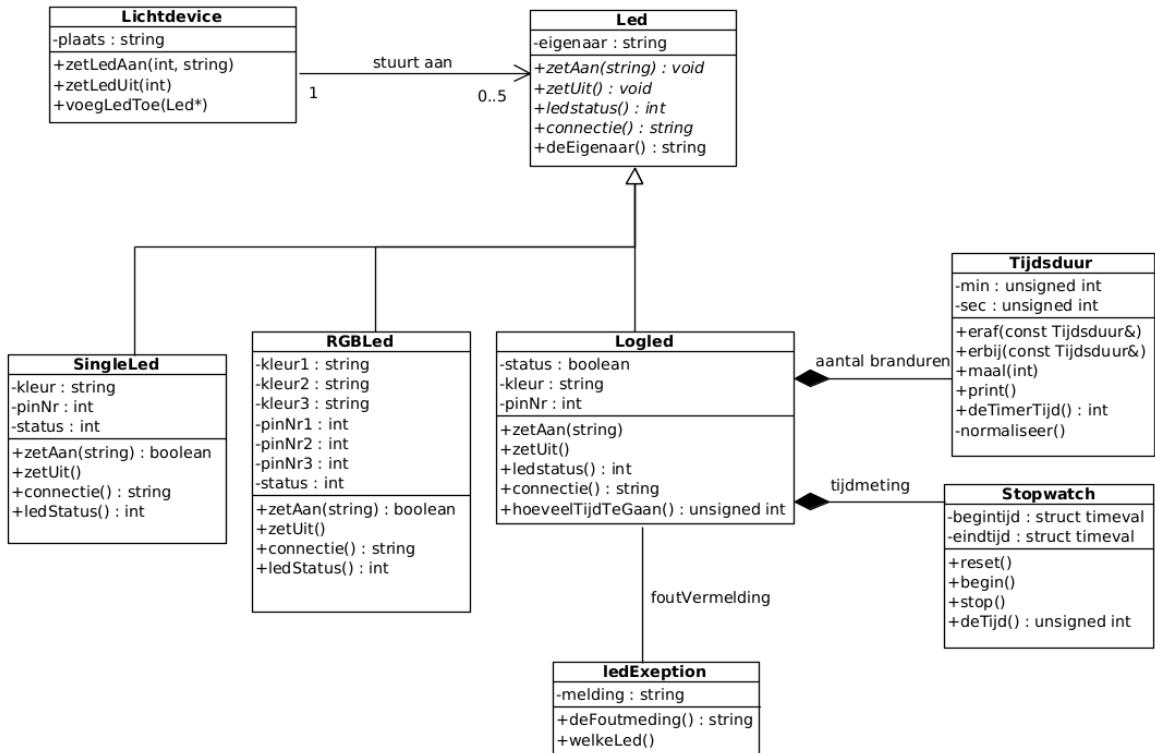
Laat het programma van Listing 6.1 runnen, met hetzelfde resultaat.

6.2 Toepassen van 1 op 0..5 multipliciteit.

Het komt vaak voor dat het ene device verschillende diverse type leds kan bevatten. Zo kan de klasse DeviceLed van Figuur 6.2 tot een maximum van 5 LEDs bevatten, die ieder afzonderlijk aangestuurd kan worden.

Opdracht

1. Maak de klasse LedDevice. Houd hier wel rekening mee dat sommige leds met exceptions kunnen gooien. Indien dit zo is, stuur de ontvangen exception door.



Figuur 6.2: Een device LED controleert meerdere leds.

2. Run het programma van listing 6.2

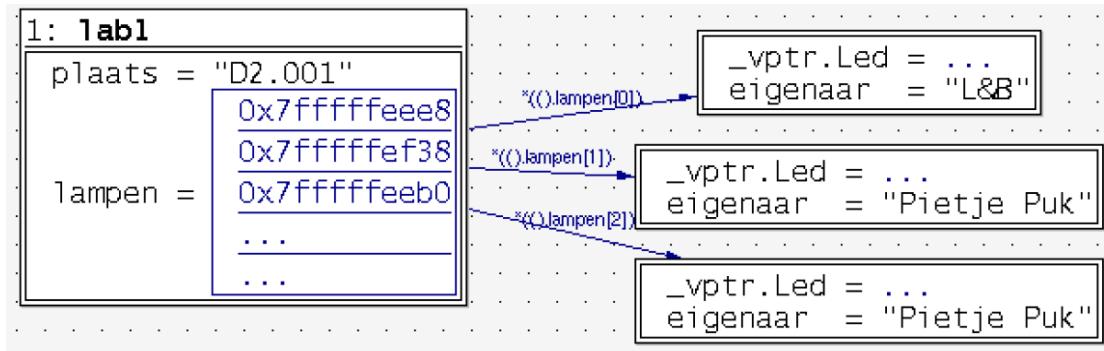
```

1 #include <iostream> // nodig voor cout (schrijven naar scherm)
2
3 int main() {
4     Logled logger(RODELEDPIN, "rood", "Pietje_Puk", 0, 4);
5     // Logled sportled(GROENELEDPIN, "groen", "Bb", 0, 2);
6     SingleLed s11(GROENELEDPIN, "groen", "L&B");
7     RGBLed k1(RGB_R, RGB_G, RGB_B, "Pietje_Puk");
8     Lichtdevice lab1("D2.001");
9     lab1.voegLedToe(&s11);
10    lab1.voegLedToe(&logger);
11    lab1.voegLedToe(&k1);
12
13    lab1.zetLedAan(0, "groen");
14    usleep(TWEE_SEC);
15    lab1.zetLedAan(2, "blauw");
16    usleep(TWEE_SEC);
17    lab1.zetLedAan(1, "rood");
18    usleep(TWEE_SEC);
19    for(int i=0; i<3; ++i)
20        lab1.zetLedUit(i);
21 }
```

Listing 6.2: Twee objecten van de klasse Logled.

3. Pas listing 6.2 aan, zodat een exception ontvangen kan worden en run deze.
4. Start de VNC viewer op en run de DDD debugger en zet een breakpoint .
5. Zet een breakpoint bij het statement: `lab1.zetLedAan(1, "rood");` en display de

Lichtdevice (lab1). Als het goed is krijg je een afbeelding dat lijkt op Figuur 6.3. Hierin is duidelijk te zien dat de verwijzingen(pointers) naar de base klasse Led



Figuur 6.3: Object **Lichtdevice** met 3 led objecten.

zijn.

6. Print uit hoeveel tijd de logled (logger) nog heeft aan het einde van het programma.
7. Plaats het volgende in je portfolio:
 - De code van de klasse **Lichtdevice** (zowel de .h als de .cpp file).
 - De code van de `int main();` functie waarbij de exceptions worden afangen.
 - Je eigen afbeelding die lijkt op Figuur 6.3.
 - De code om de restant tijd van het object logger uit te printen.

7 Introductie in unit testing

Deze opdracht bestaat uit twee onderdelen die beide afgetekend dienen te worden. Het eerste onderdeel is een introductie in een unit test. Het tweede onderdeel is een het toepassen van een unit test op een klasse uit de vorige opgave.

7.1 Deel 1, introductie

Unit testen zijn testen die gedaan worden op code niveau en in eerste instantie door de programmeur zelf. Er zijn diverse frameworks in omloop om de unit testen uit te kunnen uitvoeren. Een relatief veel gebruikt framework voor de C++ omgeving is het [google framework](#). Bij deze opgave wordt gebruik gemaakt van het [CppUTest](#) framework. Dit is een relatief eenvoudig framework en kan geïnstalleerd worden via het commando:

```
sudo apt-get install cpputest.
```

Een voorbeeld van het CppUTest framework wordt gegeven in listing 7.1 Hierbij worden twee testen uitgevoerd die allebei falen.

```
1 #include <CppUTest/CommandLineTestRunner.h>
2 #include <CppUTest/TestHarness.h>
3
4 /* The definition of a TEST_GROUP, the name is sample */
5 TEST_GROUP(FirstTestGroup)
6 {};
7
8 /* The definition of a belonging to the TEST_GROUP TEST, the name is
   ret_int_success */
9 TEST(FirstTestGroup, FirstTest)
10 {
11     FAIL("Fail_me!");
12 }
13
14 /* The definition of a belonging to the TEST_GROUP TEST, the name is
   ret_int_failed */
15 TEST(FirstTestGroup, SecondTest)
16 {
17     STRCMP_EQUAL("hello", "world");
18 }
19
20 int main(int argc, char *argv[])
21 {
22     CommandLineTestRunner::RunAllTests(argc, argv);
23     return 0;
24 }
```

Listing 7.1: Een eenvoudige unittest.

Download het *test1.c* van git:

```
git clone -- branch unittest https://github.com/JohnVi-hhs/oop.git  
compileer het programma g++ test1.c -lCppUTest -o test en run het programma ./test
```

Het resultaat van test1 is dat de beide testen *TEST(FirstTestGroup, FirstTest)* en *TEST(FirstTestGroup, SecondTest)* falen.

Indien we naar de code kijken, zien we dat deze uit één testgroep en twee losse testen bestaan. De rede dat beide testen falen heeft te maken met de keywords FAIL en STRCMP. De test aan de hand van het keyword FAIL, faalt altijd. De test aan de hand van het keyword STRCMP vergelijkt 2 strings, aangezien de string "hello" niet gelijk is aan "world" faalt ook deze test. Andere keywords (ook wel assertions genoemd) zijn terug te vinden in de [manual](#).

Opdracht

- (A) Zorg ervoor dat één van beide testen **niet** meer faalt, bedenk hierna zelf een 3e test, maak hierbij gebruik van een testkeyword (assertion) die nog niet in dit voorbeeld gebruikt wordt.
- (B) In de TEST_GROUP kunnen ook nog de functies void setup(); en void teardown(); gedefinieerd worden. Dit is te zien in listing 7.2.

```
1 #include <iostream>  
2 using namespace std;  
3  
4 TEST_GROUP(FirstTestGroup)  
5 {  
6     void setup()  
7     {  
8         cout<<"voor_de_test"<<endl;  
9     }  
10    void teardown()  
11    {  
12        cout<<"Na_de_test"<<endl;  
13    }  
14}
```

Listing 7.2: De TEST_GROUP met de setup en teardown functies.

De functie setup wordt aangeroepen voordat een test wordt aangeroepen, de functie teardown wordt aangeroepen nadat een test is uitgevoerd.

Voeg de functies setup en teardown toe aan de test en voer de test opnieuw uit.

- (C) De testomgeving zoals hierboven beschreven is, is niet echt spannend. Het zou beter zijn wanneer een eigen klasse getest kan worden. In listing 7.3 en 7.4 wordt de klasse FrontBoek weergegeven.

```

class FrontBoek {
public:
    FrontBoek(string, string);
    virtual ~FrontBoek();
    string naamSchrijver() const;
    string naamTitel() const;
    void verhoogSchrijver();
private:
    string schrijver;
    string titel;
};

```

Listing 7.3: *FrontBoek declaratie file(.h)*

```

FrontBoek::FrontBoek(string n, string t):
    schrijver(n), titel(t) {
}
FrontBoek::~FrontBoek() {
}
string FrontBoek::naamSchrijver() const {
    return schrijver;
}
string FrontBoek::naamTitel() const {
    return titel;
}
void FrontBoek::verhoogSchrijver() {
    schrijver += 1;
}

```

Listing 7.4: *FrontBoek implementatie file(.cpp)*

Het is een eenvoudige klasse met de naam van de schrijver en de titel als private variabele. Verder heeft iemand nog een functie *void verhoogSchrijver()*; verzonnen, waarmee een indicatie verkregen kan worden hoeveel personen aan het boek hebben meegewerkten. Indien de klasse met het framework getest gaat worden, zou deze eruit kunnen zien zoals in listing 7.5 wordt weergegeven.

```

1 #include <CppUTest/CommandLineTestRunner.h>
2 #include <CppUTest/TestHarness.h>
3 #include <iostream>
4 using namespace std;
5
6 #include "FrontBoek.h"
7
8 TEST_GROUP(boekengroep)
9 {
10     void setup() {
11         cout << "De_boekentest" << endl;
12     }
13     void teardown() {
14         cout << "Einde_boekentest" << endl;
15     }
16 };
17 TEST(boekengroep, front)
18 {
19     FrontBoek fb1("L&B", "lab");
20     STRCMP_EQUAL("L&B", fb1.naamSchrijver().c_str());
21 }
22
23 int main(int argc, char *argv[])
24 {
25     CommandLineTestRunner::RunAllTests(argc, argv);
26     return 0;
27 }

```

Listing 7.5: *De Frontboek test.*

Compileer Listing 7.5 (test3.c). Houd er bij het compileren rekening mee dat de klasse FrontBoek ook gecompileerd wordt. Dit kan gedaan worden door de file FrontBoek.cpp mee te nemen in de compileer regel: g++ FrontBoek.cpp test3.c -lCppUTest -o test3 en run het programma ./test3

- (D) Maak een eigen test waarmee de functie verhoogschrijver van de klasse FrontBoek getest wordt en voer de test uit.

7.2 Deel2, een eigen test

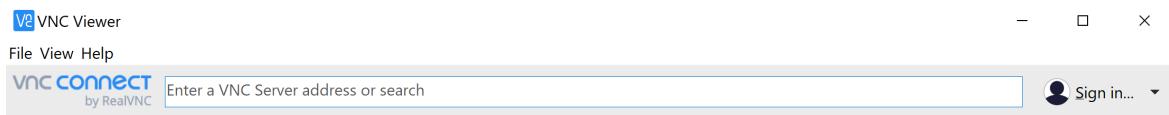
- (A) Maak eerst een unit test voor de klasse RGBLed en vervolgens voor de klasse Logled. Bedenk dat er kritisch getest moet worden.
- (B) Laat het resultaat aftekenen en plaats de volledig test in je portfolio.

Appendices

A Installeren van de practicum omgeving

A.1 De VNC viewer

Installeer [VNC Viewer](#) op je laptop. Op de RockPi is VNC Server al geïnstalleerd. Start op je laptop de VNC viewer op. Vul in het scherm *VNC CONNECT*, zie Figuur A.1, het IP adres van de RockPi in.

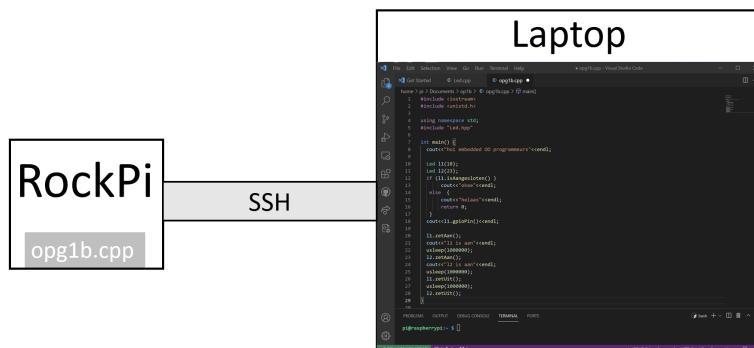


Figuur A.1: VNC verbinding met de RockPi maken.

Als het goed is zie je nu de grafische omgeving van de RockPi. Als dit niet werkt, druk dan eens op de '**Restart_X**' knop op het uitbreidingsbordje. Dit herstart de X-Windows grafische schil op de RockPi. Probeer daarna opnieuw te verbinden.

A.2 Het gebruik van 'Visual Studio Code'

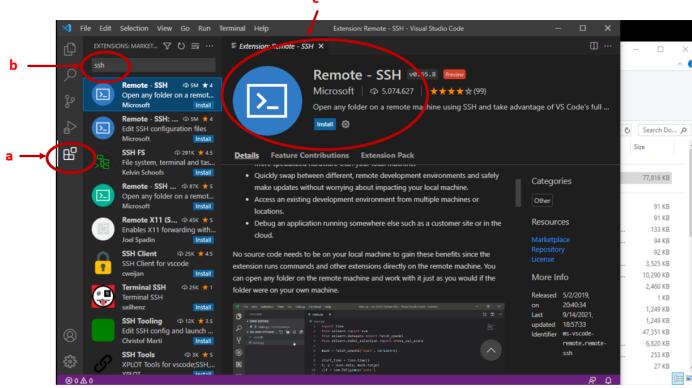
[Visual Studio Code](#) is een cross platform editor uit de Microsoft omgeving. Je kan vanuit je laptop/desktop omgeving direct files editeren op een embedded platform zoals een RockPi. Dit principe is te zien in Figuur A.2. Op de laptop draait 'Visual Studio



Figuur A.2: 'Visual Studio Code' en de RockPi.

Code' (VSC) en de source file is de file `opg1.cpp` op de RockPi. Het compileren kan via VSC, maar kan ook via de terminal in VSC of via een externe terminal b.v. Windows PowerShell, KiTTY / PuTTY of een andere.

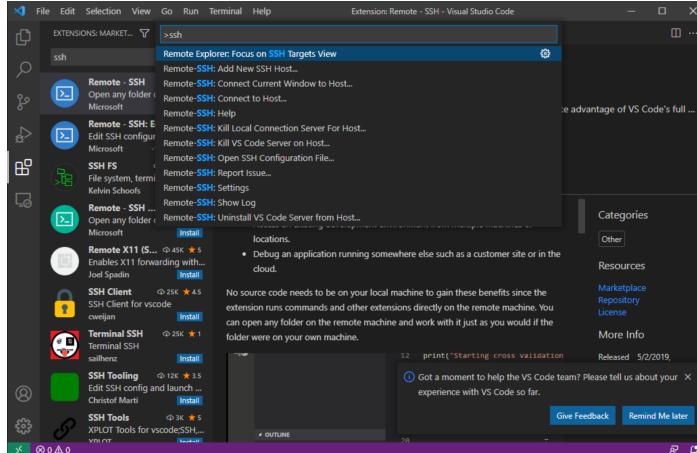
1. Download en installeer VCS [Visual studio code](#)
2. Bij Visual Studio Code kunnen meerdere extensions geïnstalleerd worden. Het installeren van de *remote SSH* extension kan als volgt gedaan worden:



Figuur A.3: Installeren van Remote SSH in VCS.

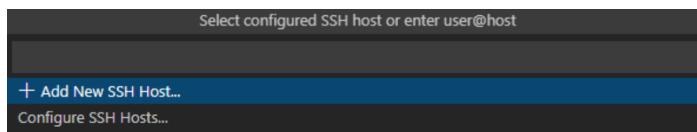
- (a) Klik op Extension of CTRL + Shift + X
 - (b) Zoek naar SSH
 - (c) klik op install
3. Verbinding maken met de RockPi.

- (a) Ga naar het 'Command Palette' (CTRL+Shift+P). Voer het commando **Remote-SSH:Connect to Host...**



Figuur A.4: SSH verbinding naar de Host.

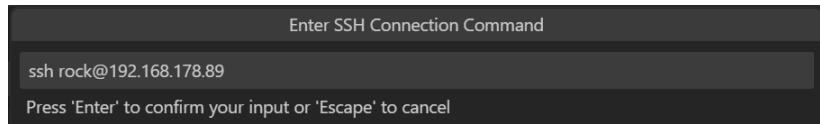
Zoals te zien is in Figuur A.4. VSC komt nu met het scherm zoals te zien is in Figuur A.5.



Figuur A.5: toevoegen van een host.

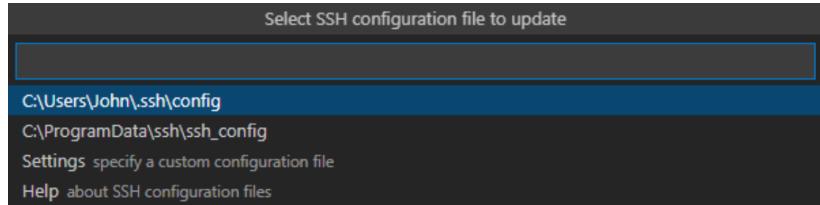
- (b) klik op **+ Add New SSH Host...**. VSC vraagt vervolgens om een SSH command: type in: ssh rock@ip.nr. van de host zoals te zien is in Figuur A.6.

Uiteraard wel met je eigen IP nummer. VSC wil de gegevens opslaan en



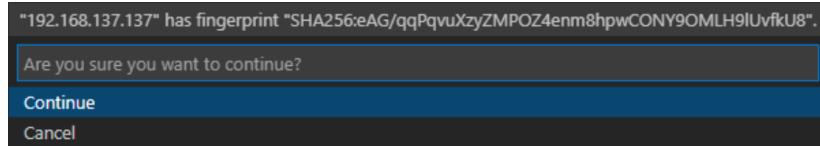
Figuur A.6: connectie maken met de host.

vraagt vervolgens waar deze opgeslagen moet worden, zoals te zien is in Figuur A.7.



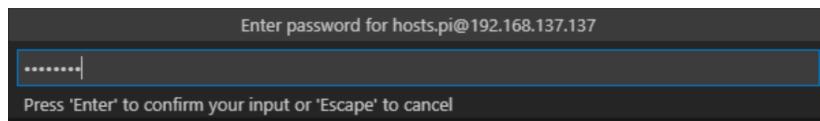
Figuur A.7: waar de gegevens moeten worden opgeslagen.

VSC slaat de gegevens op en geeft een bericht (rechtsonder) of de config file geopend moet worden of dat een connectie gemaakt moet worden. Maak een connectie, VSC komt vervolgens met de vraag of je zeker weet dat je doorgaat, zoals Figuur A.8 laat zien.



Figuur A.8: waar de gegevens moeten worden opgeslagen.

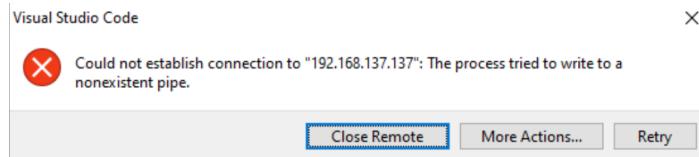
- (c) Klik op *Continue*. VSC vraagt om een password zoals Figuur A.9 laat zien. Het password is: *rock*



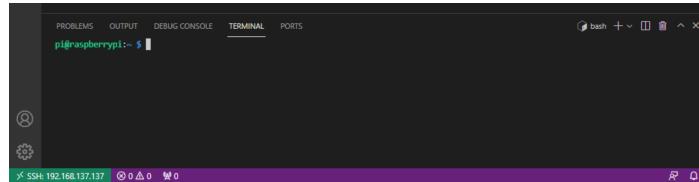
Figuur A.9: invullen van een password.

Het kan zijn dat een foutmelding gegeven wordt zoals b.v. in Figuur A.10 laat zien. Raak niet in paniek en probeer desgewenst weer opnieuw.

- (d) Maak een nieuwe terminal aan (Terminal → New Terminal). Als het goed is verschijnt de terminal in het onderste deel van VSC met een SSH verbinding naar de RockPi, zoals Figuur A.11 laat zien.
- Met het *ls* commando vraag je de listing op van de huidige directory.



Figuur A.10: een foutmelding.



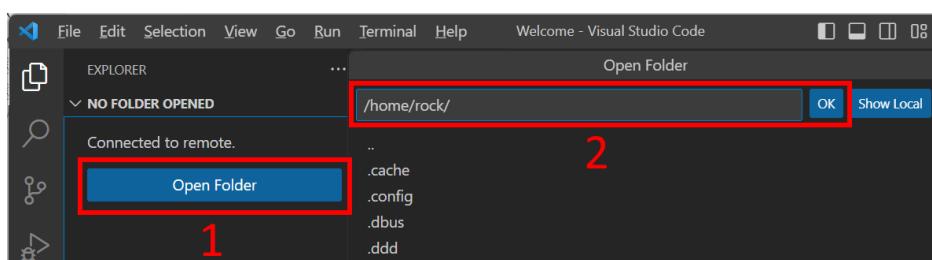
Figuur A.11: Een geopende terminal in VSC.

- ii. Met het *pwd* commando wordt het pad van de huidige directory zichtbaar.
pwd <enter> → /home/rock.
- iii. Met het *cd* commando wordt naar een directory gegaan
(LET OP: Ik ga er van uit dat je een **USB stick** gebruikt!):
cd Documents <enter> → rock@rockpi-4b:~/Documents \$
- iv. Met het *mkdir* commando wordt een directory aangemaakt.
Maak in de directory *Documents* een directory *intro* aan.

Ga naar de directory *intro* (*cd intro*) en vraagt het path op.
pwd → /home/rock/Documents/intro

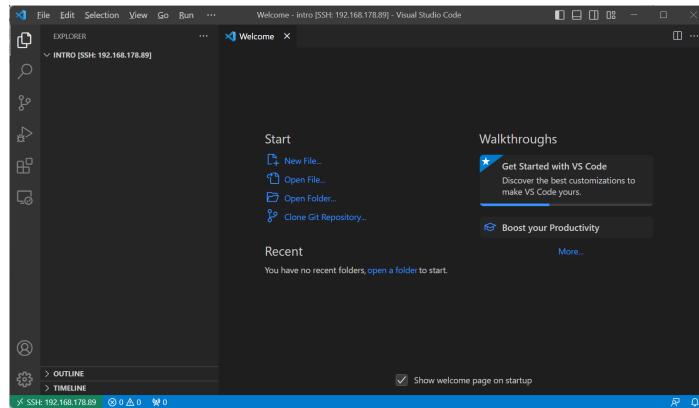
4. Het eerste programma op de RockPi.

- (a) VSC werkt voornamelijk met directory's (folders). Het openen van een folder in VSC op de RockPi kan door te klikken op Explorer of Ctrl+Shift+E. VSC opent een scherm, zoiets als [1] in Figuur A.12.



Figuur A.12: De directory die geopend moet worden.

Voer bij [2] de directory */home/rock/Documents/intro* in en klik op 'OK'. VSC kan vragen of je de auteur vertrouwt. Klik op 'Yes'. In het *EXPLORER* veld verschijnt vervolgens de nu nog lege werkdirctory, zoals in Figuur A.13 te zien is.



Figuur A.13: In het EXPLORER veld is de werkdirectory te zien.

- (b) Het programma van listing A.1 kan: gecreëerd worden door een nieuwe file *hoi.cpp* aan te maken [i], gedownload worden van Brightspace [ii] of gecloned worden van Github [iii].
 - i. Klik op New File en geef de filenaam *hoi.cpp*. Type/kopieer de tekst van Listing A.1 in de file *hoi.cpp*

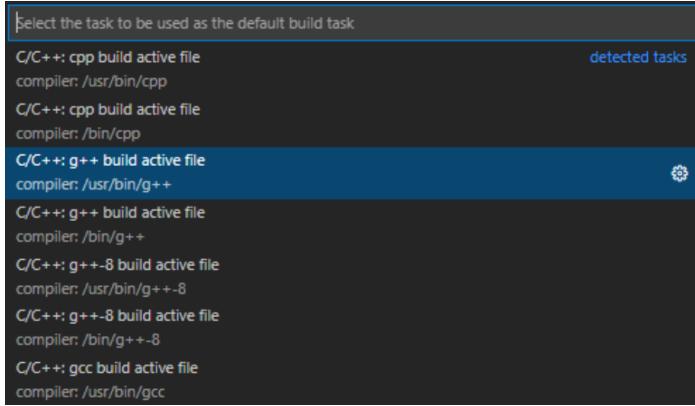
```
#include <stdio.h>
int main() {
    printf("Hoi_programmeurs_van_de_wereld\n");
    return 0;
}
```

Listing A.1: het meest gebruikte voorbeeld programma.

 - ii. Download de file *hoi.cpp* van Brightspace en plaats deze in de directory. Het laatste kan zelfs gedaan worden door middel van slepen van de file.
 - iii. Via git: *git clone - -branch intro https://github.com/Johnny63Vi/oopr1.git*¹
Het nadeel van deze methode is dat een directory oopr1 wordt aangemaakt waarin de file komt te staan.
 - (c) Open de bestaande terminal of open een nieuwe terminal *Ctrl+Shift+T*. VSC opent een terminal in de werkdirectory. compileer de file *hoi.cpp* met de g++ compiler (*g++ hoi.cpp*) en run het gecompileerde programma *./a.out*.
Uiteraard kan ook gekozen worden voor een externe terminal zoals, *Windows PowerShell*, *PuTTY*, etc.
Delete a.out (*rm a.out*).
5. Compileren via VSC.
Je kan het programma ook via VSC laten compileren.
- (a) Daarvoor moet de C++ omgeving in Visual Studio Code worden geïnstalleerd (is al gedaan bij de RockPi):

¹bij uitvoering geen spatie tussen - -

- i. Klik op Extension  of Ctrl+Shift+X.
 - ii. Type in C++ en installeer de C++ , het Extension pack en C/C++ Themes.
- (b) Activeer het tabblad *hoi.cpp*.
- (c) Klik op *Terminal* → *Configure Default Build Task...*. VSC komt met een scherm zoals Figuur A.14. Kies voor C/C++: g++ build active



Figuur A.14: Het kiezen van de compiler.

file. VSC maakt nu een task.json file aan waarin de gegevens betreffende compiler worden opgeslagen.

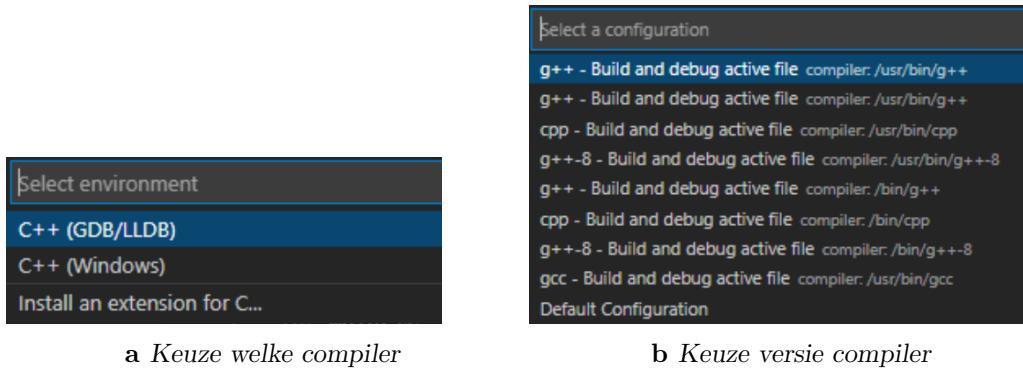
- (d) Activeer opnieuw *hoi.cpp* tab. Ga naar Terminal → Run Build Task... of *Ctrl+Shift+B*. VSC zal nu *hoi.cpp* compileren.
- (e) Het gecompileerde programma kan nu gerund worden in b.v. een terminal. Open een terminal of maak een nieuwe terminal aan *Ctrl+Shift+`* en run *hoi* (*./hoi*).

6. Run/Debug via VSC.

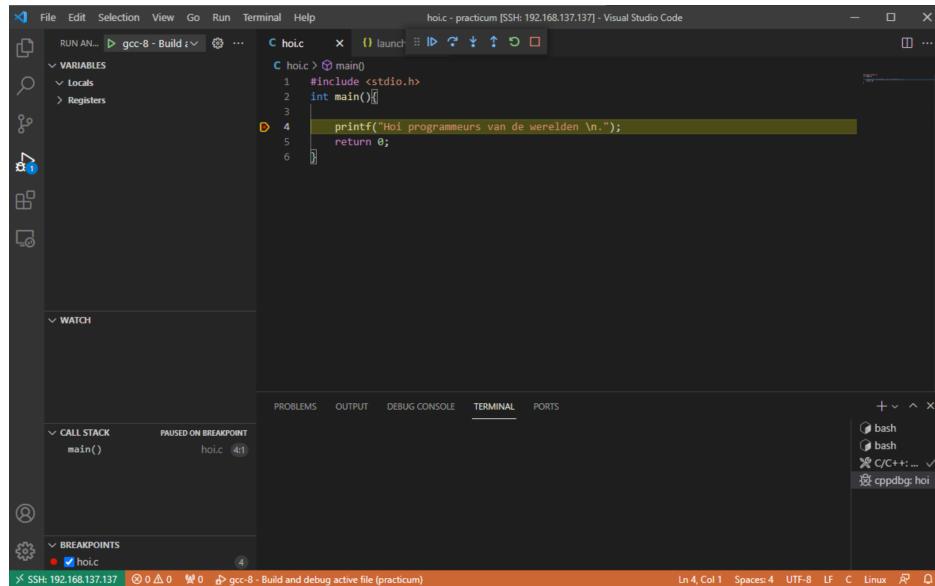
Je kan een programma ook direct via VSC laten runnen en Debuggen.

- (a) Activeer het tabblad *hoi.cpp*.
- (b) Klik op *Run and Debug (Ctrl+Shift+D)*  en vervolgens op **Run and Debug**. Afhankelijk van instellingen kan VSC kiezen voor Figuur A.15a. Kies hierbij C++ (GDB/LLDB), VSC laat hierna Figuur A.15b zien. Kies voor: *g++ - Build and Debug active file compiler /usr/bin/g++*. VSC maakt o.a. *launch.json* file aan. Hierin worden diverse gegevens met betrekking tot de compiler/debugger in opgeslagen. Het zou ook kunnen dat VSC de DEBUG CONSOLE openet.
- (c) Klik op tabblad *hoi.cpp* en plaats vervolgens een breakpoint voor regel 4 (linker muisklik voor regel 4) (*printf*), een rode stip verschijnt voor de 4.
- (d) Start de debugger:Run → Start Debugging of F5 of klik op . Het programma wordt op de RockPi uitgevoerd en stopt op regel 4, zoals te zien is

in Figuur A.16.



Figuur A.15: Keuze compiler te gebruiken door VSC



Figuur A.16: Debuggen met VSC.

Met behulp van de debugknoppen kan stap voor stap door het programma gelopen worden. Hieronder staan de belangrijkste:

	'Continue (F5)'	Vervolg het programma zonder onderbrekingen.
	'Step Over (F10)'	Stap over de volgende regel heen (ook als het een functieaanroep is).
	'Step into (F11)'	Voer de regel uit. Als het een functieaanroep is: ga de functie in. <i>Als het een systeemfunctie is, dan wordt het vaak onbegrijpelijk!</i>

B Tips&Tricks&Troubleshooting voor Linux omgeving

B.1 Troubleshooting

Geen beeld in VNC.

Je hebt VNC opgestart, maar na een tijdje heb je geen beeld meer. Waarschijnlijk komt dat door de screensaver. VNC reageert niet op een ‘virtuele’ toetsenbord of muis. Druk dan het knopje ‘**restart_x**’ in log opnieuw in op VNC. *De screensaver wordt uitgeschakeld als je na het inloggen in VNC een terminal venster opent.* Daarmee worden direct een paar commando’s uitgevoerd waarmee de screensaver wordt uitgezet. **DDD ‘doet raar’.**

De instellingen van de DDD debugger staan in de folder .ddd die staat in de home directory (*/home/rock*). Soms is het eenvoudiger om deze directory te deleten dan de instellingen aan te passen (tips: `ls -al ~` en `rm -rf ~/ddd`).

B.2 Tips

Je kunt de tips en trucs toepassen bij de commando’s in B.3

- Linux is in tegenstelling tot Windows ‘case-sensitive’. Het maakt dus verschil of je hoofdletters of kleine letters gebruikt. Voorbeeld: Als je map Documents heet, dan werkt `cd Documents` wel en `cd documents` niet.
- Tip bij het invoeren van map- en bestandsnamen: tik de eerste 3 letters in en druk de Tab toets in, Linux vult dan automatisch de rest van de naam in. Voordeel: je hoeft minder te tikken en het is gelijk een foutcontrole.
- Op en neer pijltjestoetsen in de terminal: door vorige commando’s heen gaan.
- Nog een verschil met Windows; Windows werkt met bestandsextensies: *.exe is een uitvoerbaar (executable) bestand. In Linux doet de extensie er niet toe, maar zet je met `chmod +x` het ‘executable’ bitje aan, een attribuut op het bestand.
- Hulp vragen bij commando’s: `<commando> --help`. Voorbeeld: `ls --help` Uitgebreidere informatie: `man <commando>`. Voorbeeld: `man ls` Voor de bedieningstoetsen van man druk ‘h’.
- **Uitleg en gebruik van het sudo commando** (‘**S**uper **U**ser **D**o’); admin rechten.
- Zoeken in het bestandssysteem: `sudo find / -name <bestandsnaam>` zoekt naar `<bestandsnaam>` vanaf de root ‘/’ in het gehele bestandssysteem.
- **Uitleg van- en navigeren door het bestandssysteem.**

- Wil je meer uitleg, zoek dan naar 'Linux Command Cheatsheet'.
Ik vind deze uitleg wel prettig.
- Om bestanden vanaf je PC naar en van de RockPi te kopiëren kun je WinSCP gebruiken. Gebruik als protocol SCP, dit gebruikt SSH. Maak een nieuwe site aan, vul IP adres, username en password in. Kies Opslaan (met wachtwoord) en vul bij 'map' in /home/rock. Het gebruik wijst zichzelf.

B.3 Linux commando's (zie ook tips in B.2)

Commando	Uitleg
<code>ls -l</code>	Lange directory listing - Opvragen van bestanden in een directory
<code>cd</code>	change directory; <code>cd ..</code> betekent 1 niveau omhoog naar de root '/'
<code>cp</code>	copy
<code>rm</code>	remove - Verwijderen van bestanden en mappen.
<code>mv</code>	move - Verplaatsen van bestanden en mappen.
<code>cat</code>	bestand bekijken
<code>pwd</code>	In welke directory ben ik nu? ('present working directory')
<code>mkdir</code>	make directory
<code>rmdir</code>	remove (lege) directory
<code>history</code>	Laat lijst zien van laatst uitgevoerde commando's
<code>sudo</code>	Zet dit vóór je commando om het met admin rechten uit te voeren. <i>Alle onderstaande commando's hebben sudo nodig!</i>
<code>chmod</code>	(lees/schrijfrechten wijzigen, +x maakt bestand executable.
<code>chown</code>	eigenaar van bestand / map wijzigen
<code>journalctl</code>	journalctl -n 30 laat de laatste 30 regels linux systemlog zien.
<code>reboot</code>	opnieuw opstarten

B.4 Uitleg GPIO poorten in Linux

In Linux wordt alles gerepresenteerd als een bestand. Of het nu een volledige disk of een device is, je kunt het als een bestand openen en lezen en schrijven. Dit geldt ook voor GPIO pinnen ('General Purpose Input/Output' pinnen). Deze pinnen gebruik je op dezelfde manier als de pinnen van een microcontroller (denk aan een Arduino Uno, of een Microbit). Je kunt pinnen als uitgang zetten en hoog of laag maken, of als ingang instellen en uitlezen of een pin hoog of laag is (en nog meer, bijvoorbeeld pull-up en pull-down en reageren op events, maar dat gaan we niet behandelen).

Omdat pinnen gerepresenteerd worden door bestanden, geldt ook dat er rechten op gezet worden in het bestandssysteem. Meestal kan een standaard gebruiker (zoal 'rock') er niet bij. Alleen 'root' kan er standaard bij. Voor het practicum hebben we de rechten zo ingesteld dat gebruiker 'rock' standaard gebruik kan maken van de op de breakout board beschikbare pinnen.

Voorbeeld van de handelingen om een pin in te stellen en uit te lezen of aan te sturen, het hekje '#' aan het begin van de regel geeft aan dat het commentaar (uitleg) is:

```
# GPIO pin 135 (de rode led) aanzetten:  
echo 135 > /sys/class/gpio/export  
# GPIO pin 135 (de rode led) als uitgang instellen:  
echo out > /sys/class/gpio/gpio135/direction  
# GPIO pin 135 (de rode led) hoog maken:  
echo 1 > /sys/class/gpio/gpio135/value  
# 1 seconde wachten  
sleep 1  
# GPIO pin 135 (de rode led) laag maken:  
echo 0 > /sys/class/gpio/gpio135/value
```

Je kunt ook de hele bestandstructuur van de pin bekijken:

```
rock@rockpi-4b:~$ ls -l /sys/class/gpio/gpio135/  
total 0  
-rwSrwsr-- 1 root gpio 4096 Mar 12 18:49 active_low  
lrxwxrwxrwx 1 root gpio 0 Mar 12 18:49 device -> ../../../../../pinctrl  
-rwSrwsr-- 1 root gpio 4096 Mar 12 18:49 direction  
-rwSrwsr-- 1 root gpio 4096 Mar 12 18:49 edge  
drwsrwsr-x 2 root gpio 0 Mar 12 18:49 power  
lrxwxrwxrwx 1 root gpio 0 Mar 12 18:49 subsystem -> ../../../../../../  
    class/gpio  
-rwSrwsr-- 1 root gpio 4096 Mar 12 18:49 uevent  
-rwSrwsr-- 1 root gpio 4096 Mar 13 10:55 value
```

En een bepaalde status uitlezen (in dit geval of de pin als ingang of uitgang is ingesteld):

```
rock@rockpi-4b:~$ cat /sys/class/gpio/gpio135/direction  
out  
rock@rockpi-4b:~$
```

Tot zover de uitleg. Ik laat GPIO pin als ingang en gebruik met PWM voor nu achterwege. Als je daar meer van wilt weten, kijk dan even naar het zelftest scriptje: sudo cat /usr/local/bin/pwmtest.sh