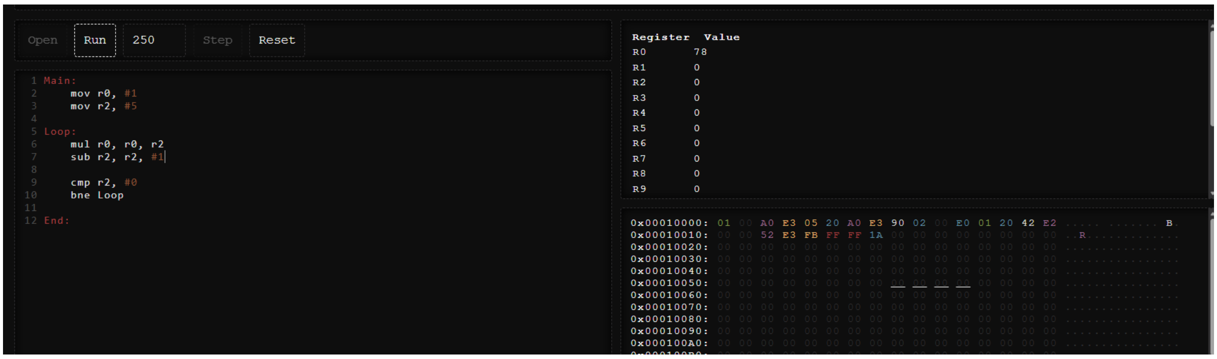
# Template Week 4 – Software

Student number: 579053 (Andy Melkonian)

### Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:



```
Main:
    mov r0, #1
    mov r2, #5

Loop:
    mul r0, r0, r2
    sub r2, r2, #1

    cmp r2, #0
    bne Loop

End:
```

```
Register  Value
R0        78
R1        0
R2        0
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
```

```
0x00010000: 01 00 A0 E3 05 20 A0 E3 90 02 00 E0 01 20 42 E2 ..............B.
0x00010010: 00 00 52 E3 FB FF FF 1A 00 00 00 00 00 00 00 00 ..R.............
0x00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x00010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

78 is hexadecimaal maar staat gelijk aan 120 want 8 + 16 * 7 = 120

### Assignment 4.2: Programming languages

Take screenshots that the following commands work:

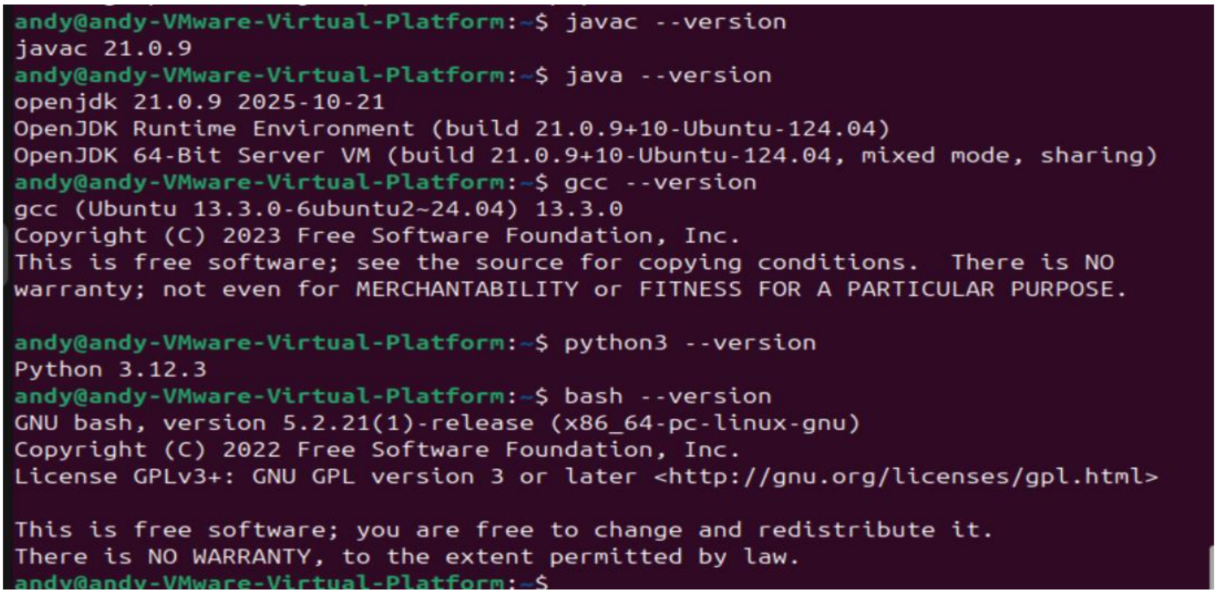
javac --version

java --version

gcc --version

python3 --version

bash --version



```
andy@andy-VMware-Virtual-Platform:~$ javac --version
javac 21.0.9
andy@andy-VMware-Virtual-Platform:~$ java --version
openjdk 21.0.9 2025-10-21
OpenJDK Runtime Environment (build 21.0.9+10-Ubuntu-124.04)
OpenJDK 64-Bit Server VM (build 21.0.9+10-Ubuntu-124.04, mixed mode, sharing)
andy@andy-VMware-Virtual-Platform:~$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

andy@andy-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
andy@andy-VMware-Virtual-Platform:~$ bash --version
GNU bash, version 5.2.21(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
andy@andy-VMware-Virtual-Platform:~$
```

**Assignment 4.3: Compile**

Which of the above files need to be compiled before you can run them?

**Fib.c en Fibonacci.java**

Which source code files are compiled into machine code and then directly executable by a processor?

**fib.c**

Which source code files are compiled to byte code?

**Fibonacci.java**

Which source code files are interpreted by an interpreter?

**Fib.py en fib.sh**

These source code files will perform the same calculation after compilation/interpretation. Which one is expected to do the calculation the fastest?

**Fib.c**

How do I run a Java program?

**Compile and execute, javac en java**

How do I run a Python program?

**Python3 filename.py**

How do I run a C program?

Compileren: gcc fib.c -o fib

Uitvoeren: ./fib

How do I run a Bash script?

**Bash filename.sh**

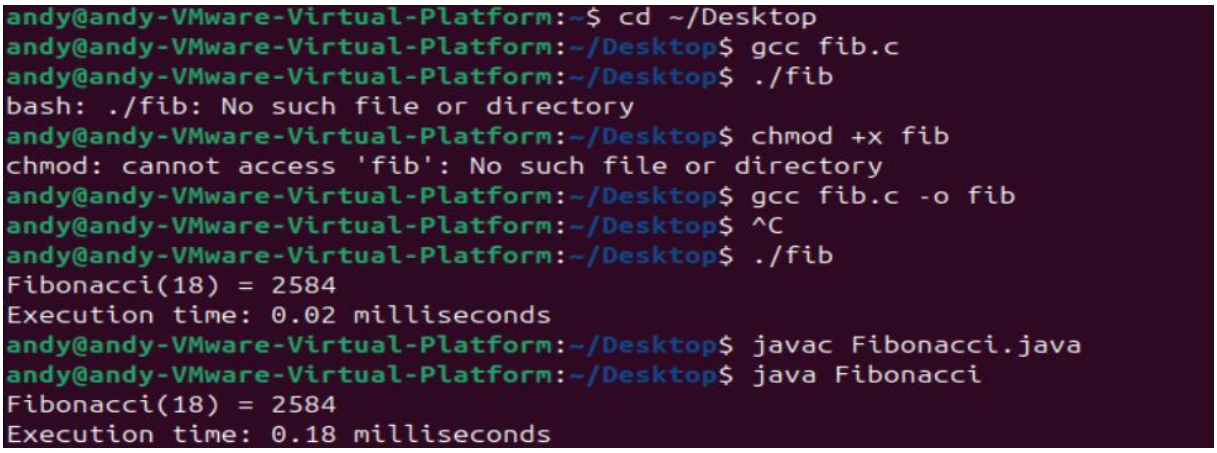
If I compile the above source code, will a new file be created? If so, which file?

**Bij C (fib.c):** Ja a.out

**Bij Java (Fibonacci.java):** Ja Fibonacci.class.

Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?



```
andy@andy-VMware-Virtual-Platform:~$ cd ~/Desktop
andy@andy-VMware-Virtual-Platform:~/Desktop$ gcc fib.c
andy@andy-VMware-Virtual-Platform:~/Desktop$ ./fib
bash: ./fib: No such file or directory
andy@andy-VMware-Virtual-Platform:~/Desktop$ chmod +x fib
chmod: cannot access 'fib': No such file or directory
andy@andy-VMware-Virtual-Platform:~/Desktop$ gcc fib.c -o fib
andy@andy-VMware-Virtual-Platform:~/Desktop$ ^C
andy@andy-VMware-Virtual-Platform:~/Desktop$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
andy@andy-VMware-Virtual-Platform:~/Desktop$ javac Fibonacci.java
andy@andy-VMware-Virtual-Platform:~/Desktop$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.18 milliseconds
```

Fib.c (dus in C) is het snelst want dat word direct omgezet naar machine code

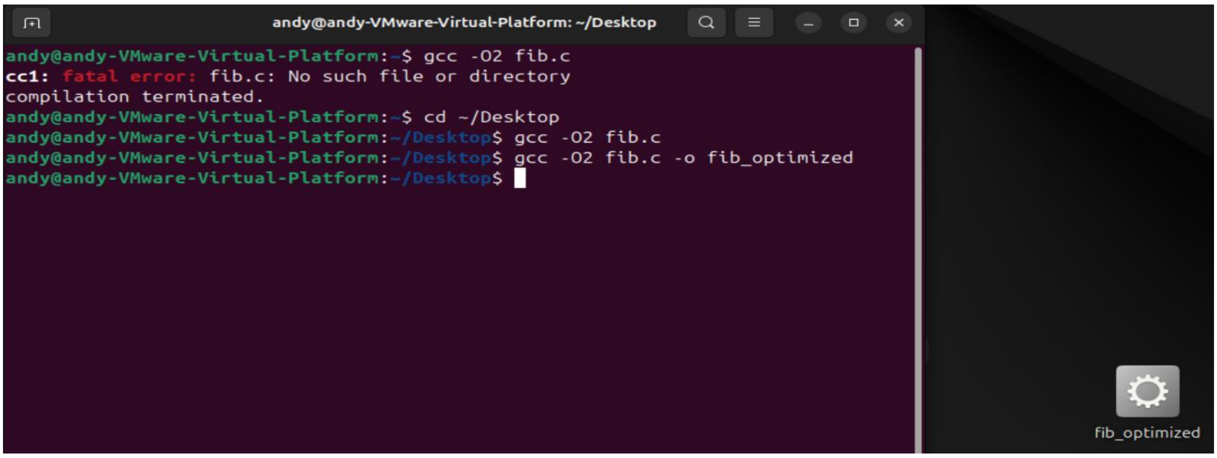
**Assignment 4.4: Optimize**

Take relevant screenshots of the following commands:

a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book, but find a better optimization in the man pages. Please note that Linux is case sensitive.

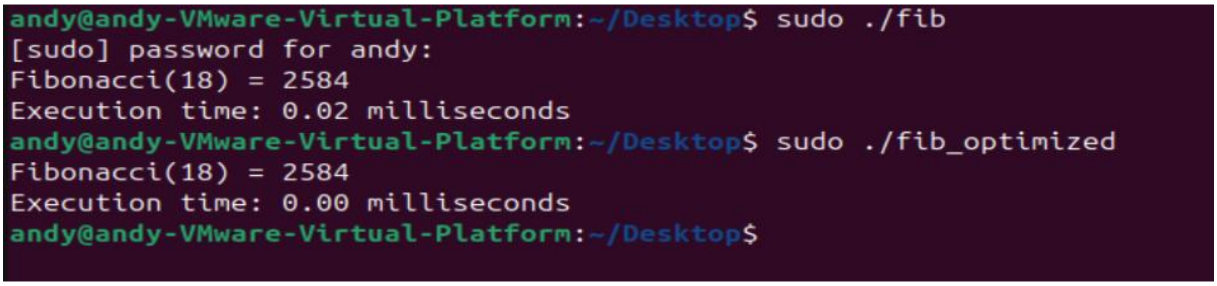
   Volgens het boek is dat -O, de standaard met de beste optimalizatie is -O2

b) Compile **fib.c** again with the optimization parameters



```
andy@andy-VMware-Virtual-Platform:~$ gcc -O2 fib.c
cc1: fatal error: fib.c: No such file or directory
compilation terminated.
andy@andy-VMware-Virtual-Platform:~$ cd ~/Desktop
andy@andy-VMware-Virtual-Platform:~/Desktop$ gcc -O2 fib.c
andy@andy-VMware-Virtual-Platform:~/Desktop$ gcc -O2 fib.c -o fib_optimized
andy@andy-VMware-Virtual-Platform:~/Desktop$ 
```

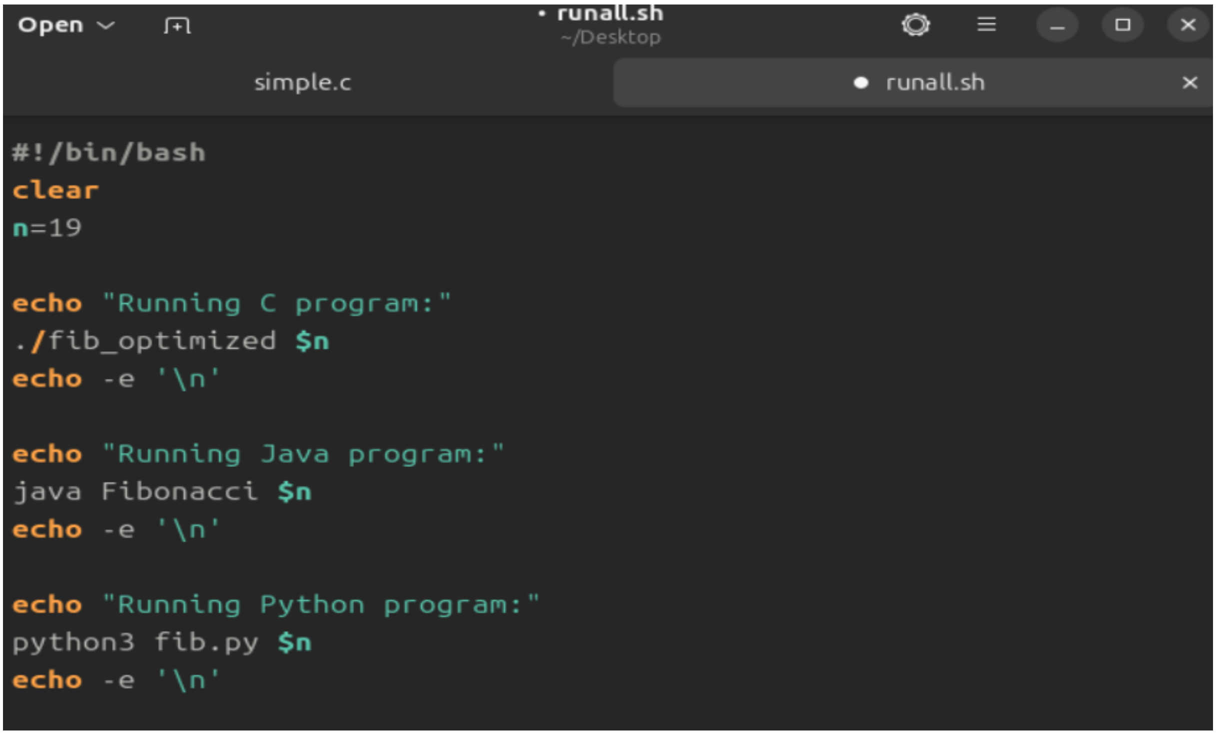
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
andy@andy-VMware-Virtual-Platform:~/Desktop$ sudo ./fib
[sudo] password for andy:
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
andy@andy-VMware-Virtual-Platform:~/Desktop$ sudo ./fib_optimized
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds
andy@andy-VMware-Virtual-Platform:~/Desktop$
```

   Ja het is dus sneller

d) Edit the file **runall.sh**, so you can perform all four calculations in a row using this Bash script. So the (compiled/interpreted) C, Java, Python and Bash versions of Fibonacci one after the other.

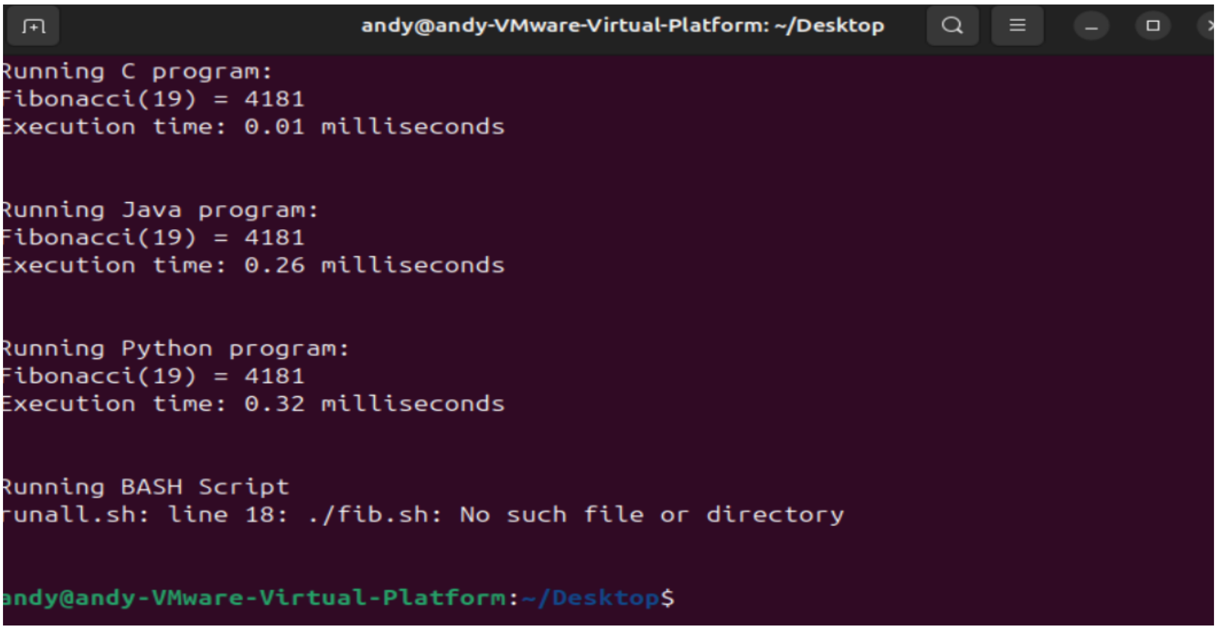


```
#!/bin/bash
clear
n=19

echo "Running C program:"
./fib_optimized $n
echo -e '\n'

echo "Running Java program:"
java Fibonacci $n
echo -e '\n'

echo "Running Python program:"
python3 fib.py $n
echo -e '\n'
```



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.26 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.32 milliseconds


Running BASH Script
runall.sh: line 18: ./fib.sh: No such file or directory


andy@andy-VMware-Virtual-Platform:~/Desktop$
```

**Assignment 4.5: More ARM Assembly**

Like the factorial example, you can also implement the calculation of a power of 2 in assembly. For example you want to calculate $2^4 = 16$. Use iteration to calculate the result. Store the result in r0.

```
Main:
    mov r1, #2
    mov r2, #4
    mov r0, #2
Loop:
    cmp r2, #1
    beq End
    mul r0, r0, r1
    sub r2, r2, #1
    b Loop
End:
```

Complete the code. See the PowerPoint slides of week 4.

Screenshot of the completed code here.

Ready? Save this file and export it as a pdf file with the name: **week4.pdf**