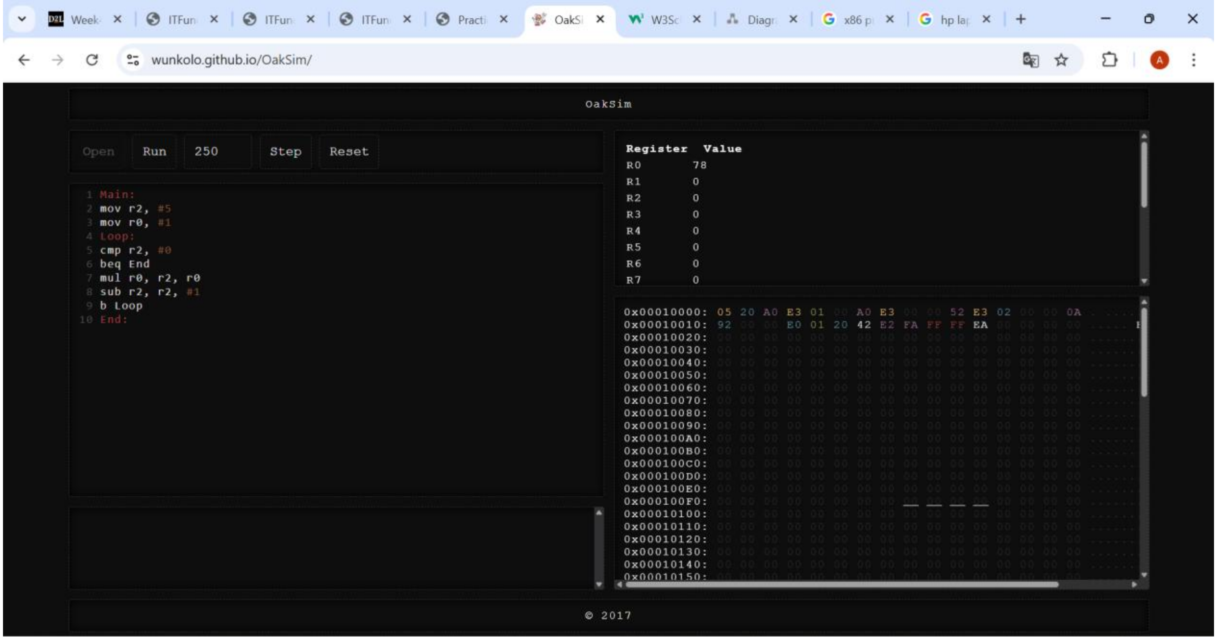
Template Week 4 – Software

Student number: 591905

Assignment 4.1: ARM assembly

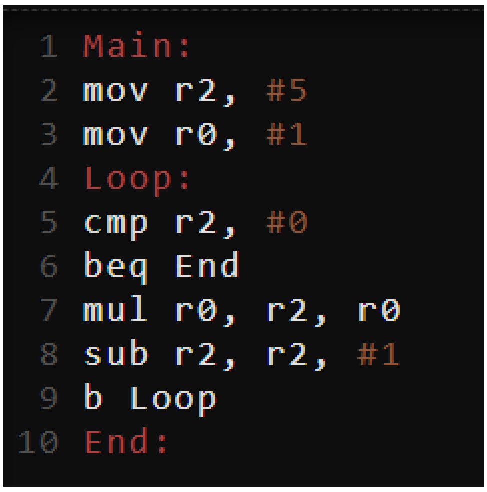
Screenshot of working assembly code of factorial calculation:



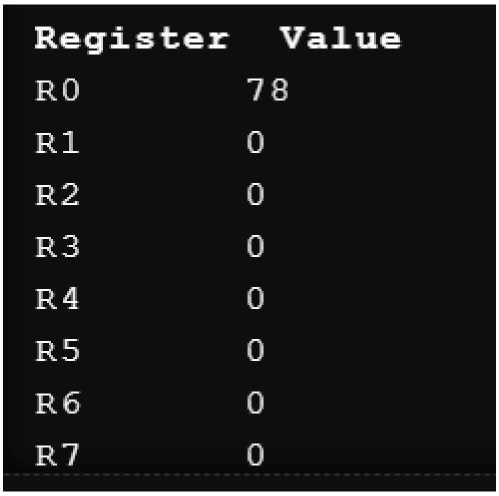
```
Main:
mov r2, #5
mov r0, #1
Loop:
cmp r2, #0
beq End
mul r0, r2, r0
sub r2, r2, #1
b Loop
End:
```

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

```
0x00010000: 05 20 A0 E3 01 00 A0 E3 00 00 52 E3 02 00 00 0A
0x00010010: 92 00 00 E0 01 20 42 E2 FA FF FF EA 00 00 00 00
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
0x000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00010150:
```



```
Main:
mov r2, #5
mov r0, #1
Loop:
cmp r2, #0
beq End
mul r0, r2, r0
sub r2, r2, #1
b Loop
End:
```

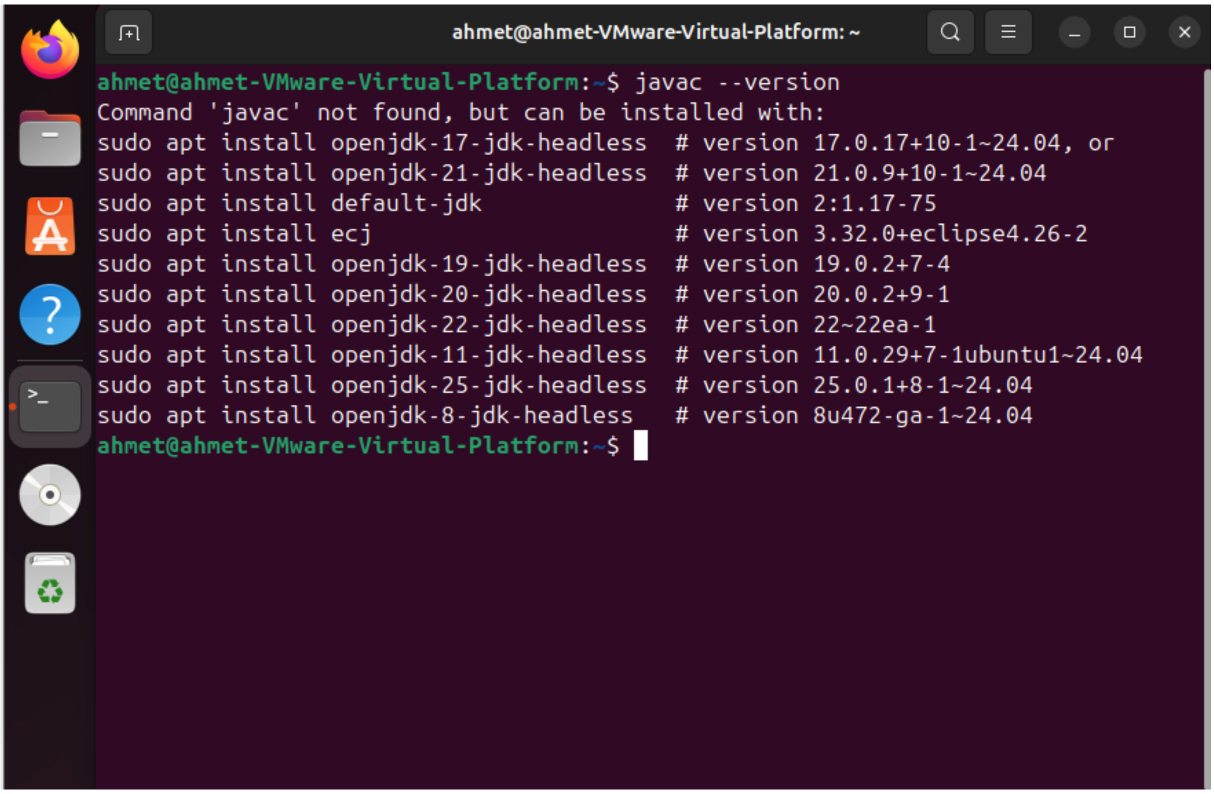


```
Register  Value
R0         78
R1         0
R2         0
R3         0
R4         0
R5         0
R6         0
R7         0
```

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

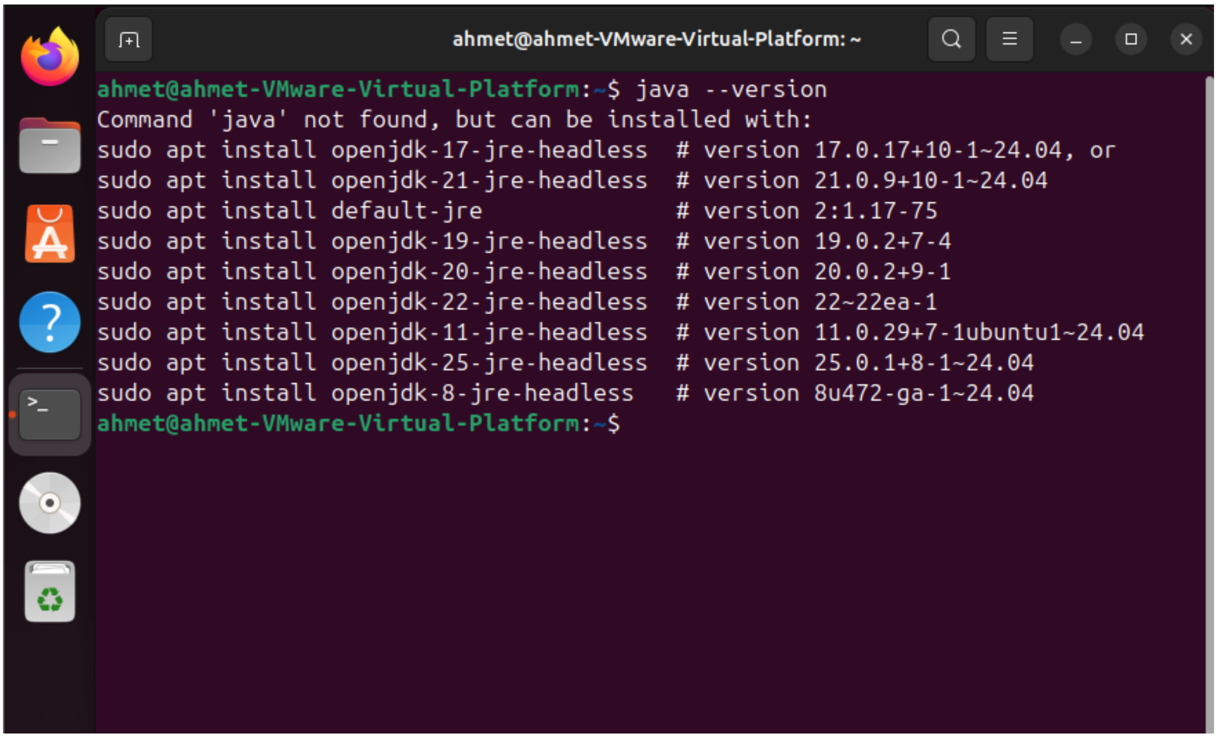
javac –version



```
ahmet@ahmet-VMware-Virtual-Platform:~$ javac --version
Command 'javac' not found, but can be installed with:
sudo apt install openjdk-17-jdk-headless  # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jdk-headless  # version 21.0.9+10-1~24.04
sudo apt install default-jdk              # version 2:1.17-75
sudo apt install ecj                      # version 3.32.0+eclipse4.26-2
sudo apt install openjdk-19-jdk-headless  # version 19.0.2+7-4
sudo apt install openjdk-20-jdk-headless  # version 20.0.2+9-1
sudo apt install openjdk-22-jdk-headless  # version 22~22ea-1
sudo apt install openjdk-11-jdk-headless  # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jdk-headless  # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jdk-headless   # version 8u472-ga-1~24.04
ahmet@ahmet-VMware-Virtual-Platform:~$ 
```

geinstalleerd

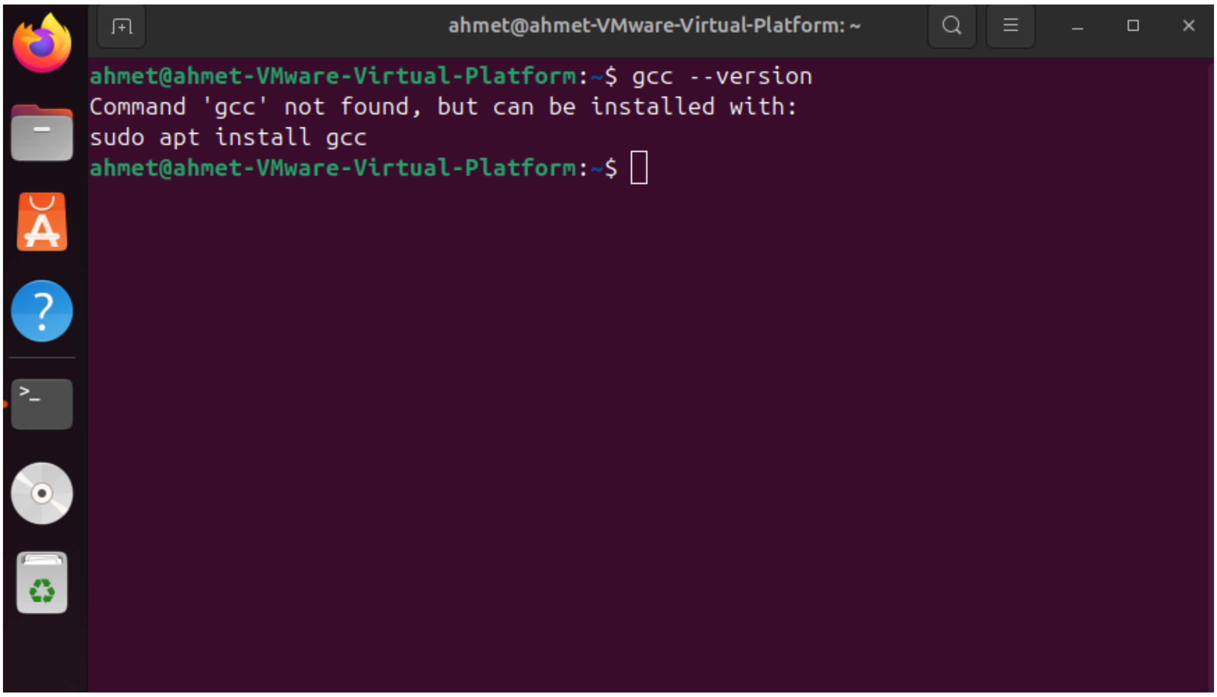
java –version



```
ahmet@ahmet-VMware-Virtual-Platform:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless  # version 17.0.17+10-1~24.04, or
sudo apt install openjdk-21-jre-headless  # version 21.0.9+10-1~24.04
sudo apt install default-jre              # version 2:1.17-75
sudo apt install openjdk-19-jre-headless  # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless  # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless  # version 22~22ea-1
sudo apt install openjdk-11-jre-headless  # version 11.0.29+7-1ubuntu1~24.04
sudo apt install openjdk-25-jre-headless  # version 25.0.1+8-1~24.04
sudo apt install openjdk-8-jre-headless   # version 8u472-ga-1~24.04
ahmet@ahmet-VMware-Virtual-Platform:~$
```

geinstalleerd

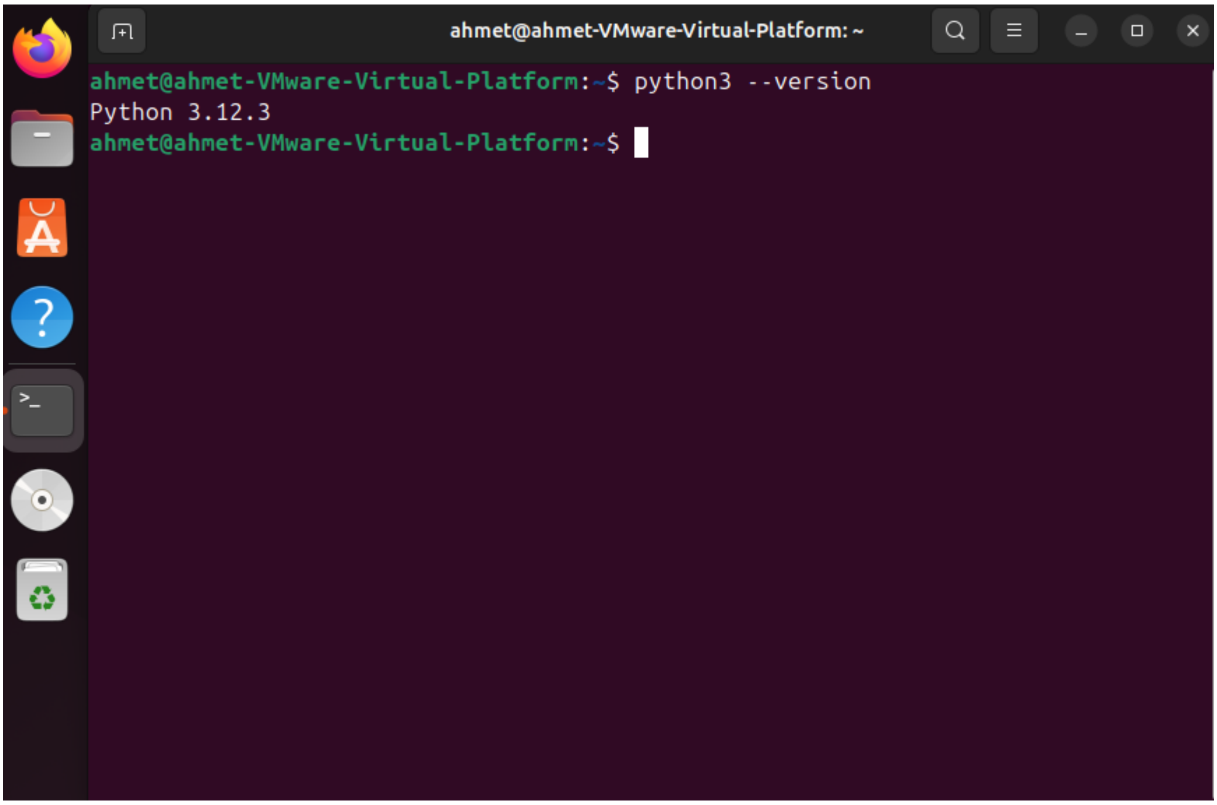
gcc –version



```
ahmet@ahmet-VMware-Virtual-Platform:~$ gcc --version
Command 'gcc' not found, but can be installed with:
sudo apt install gcc
ahmet@ahmet-VMware-Virtual-Platform:~$ 
```

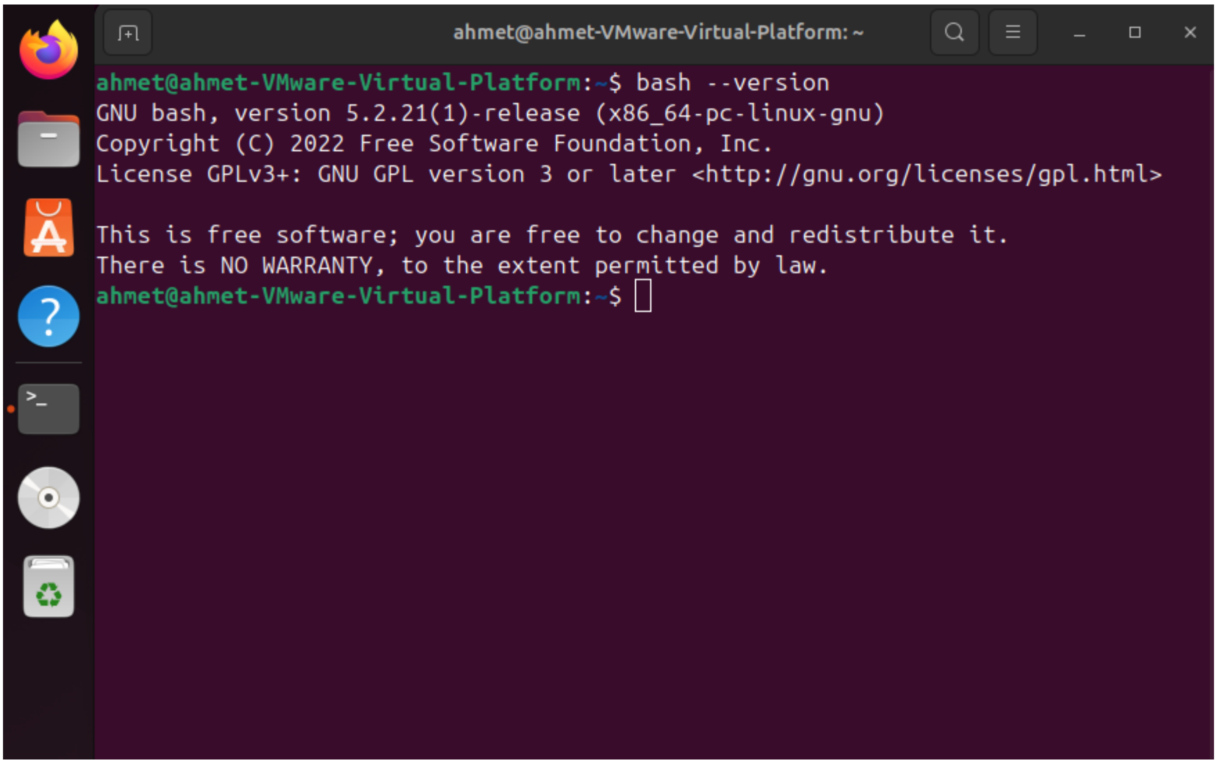
Geinstalleerd

python3 –version



```
ahmet@ahmet-VMware-Virtual-Platform:~$ python3 --version
Python 3.12.3
ahmet@ahmet-VMware-Virtual-Platform:~$ 
```

bash –version



```
ahmet@ahmet-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.
ahmet@ahmet-VMware-Virtual-Platform:~$ 
```

Assignment 4.3: Compile

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

fib.c (C), het is een compiled source code

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

fib.c (C)

Which source code files are compiled to byte code?

Fibonacci.java (Java). Die wordt Bytecode voor de JVM.

Which source code files are interpreted by an interpreter?

Fib.py (Python) en fib.sh (Bash), het zijn interpreted source codes.

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

fib.c (C). Dit draait direct op de processor (machine code) zonder tussenlaag (intermediate code) wat zorgt voor meer snelheid.

How do I run a Java program?

Compileren: javac Fibonacci.java, en daarna uitvoeren: java Fibonacci

How do I run a Python program?

Met python3 fib.py

How do I run a C program?

Compileren: gcc fib.c, en daarna uitvoeren: ./a.out

How do I Run a Bash Script?

Met bash fib.sh

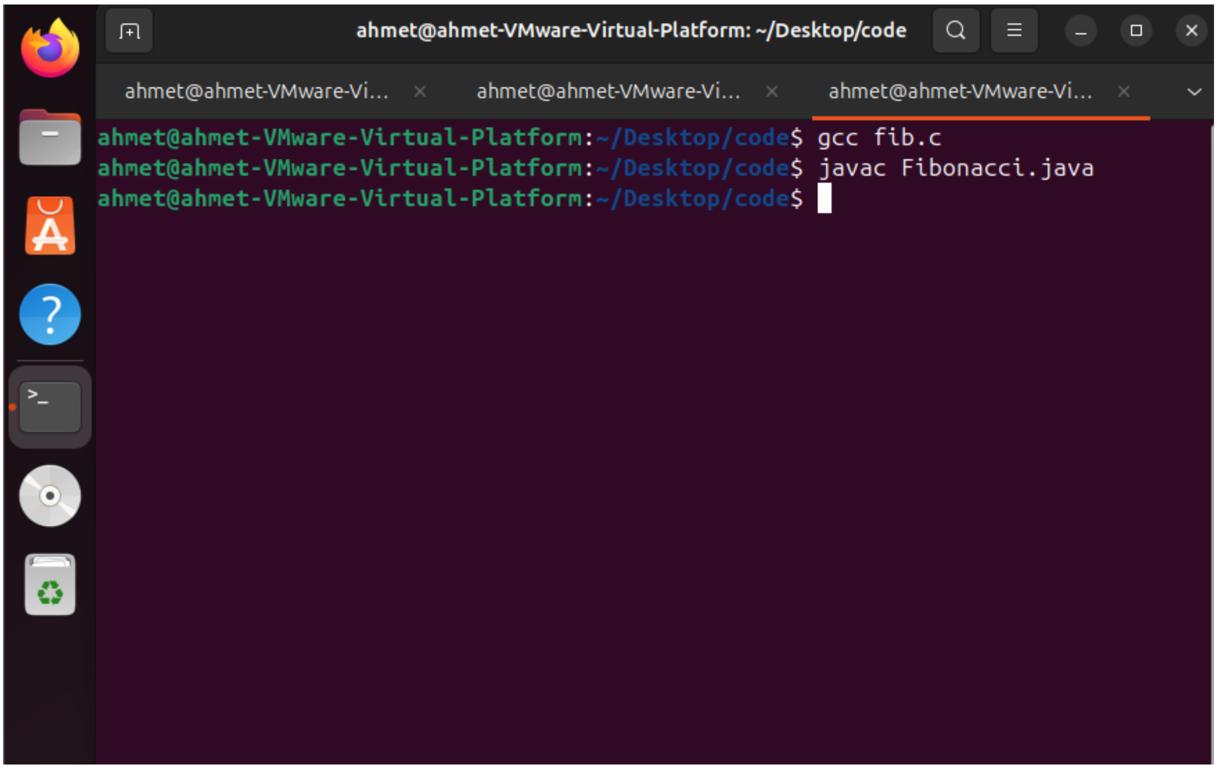
If I compile the above source code, will a new file be created? If so, which file?

Ja, bij de compileertalen. Bij C (fib.c) ontstaat er een uitvoerbaar bestand (a.out) die in machinecode is omgezet. Bij Java (Fibonacci.java) ontstaat er een bytecode bestand (Fibonacci.class) die door de JVM zal worden uitgevoerd.

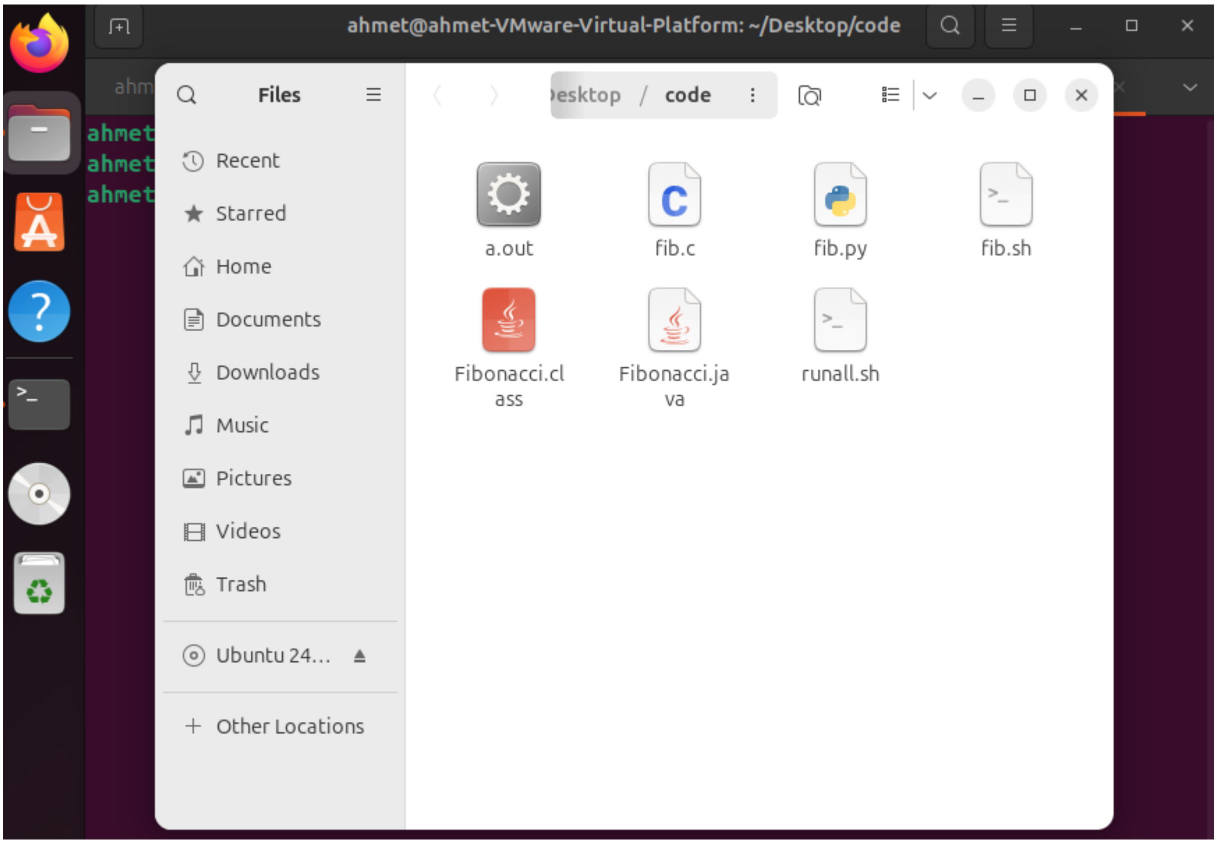
(Bij Python en Bash worden geen nieuwe bestanden aangemaakt om het te kunnen draaien omdat ze interpreted zijn).

Take relevant screenshots of the following commands:

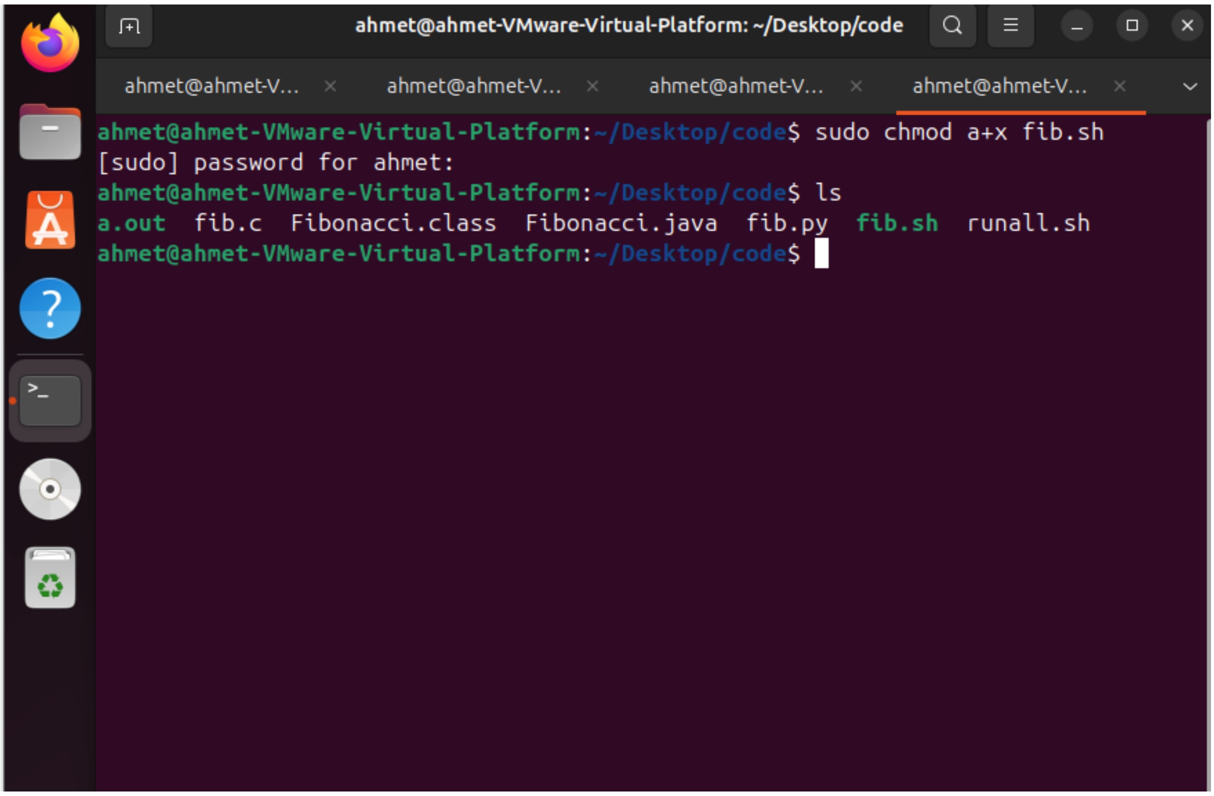
- Compile the source files where necessary



```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ gcc fib.c
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ javac Fibonacci.java
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```

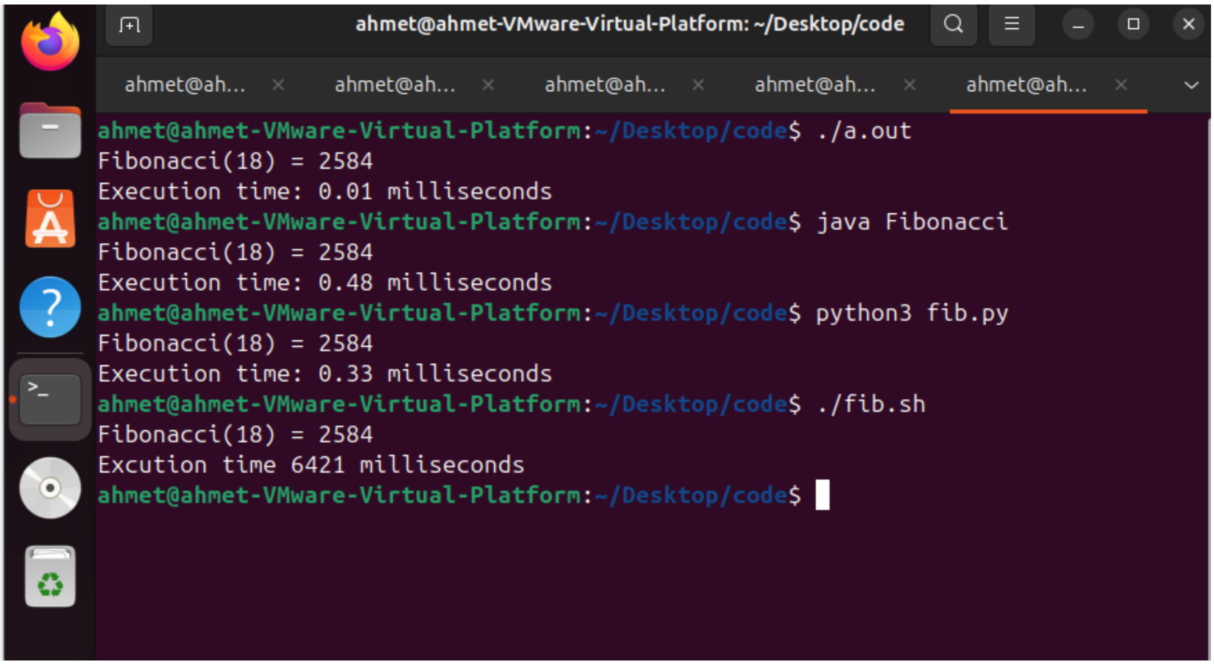


- Make them executable



```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ sudo chmod a+x fib.sh
[sudo] password for ahmet:
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ ls
a.out  fib.c  Fibonacci.class  Fibonacci.java  fib.py  fib.sh  runall.sh
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```

- Run them



```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ ./a.out
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.48 milliseconds
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.33 milliseconds
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 6421 milliseconds
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```

- Which (compiled) source code file performs the calculation the fastest?

Het C-programma (fib.c) (a.out). C wordt direct vertaald naar machinecode (de taal van de processor). Java heeft een virtuele machine nodig en Python/Bash zijn interpreters die regel voor regel lezen, wat trager is.

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.

In het boek staat er iets over -O (hoofdletter O).

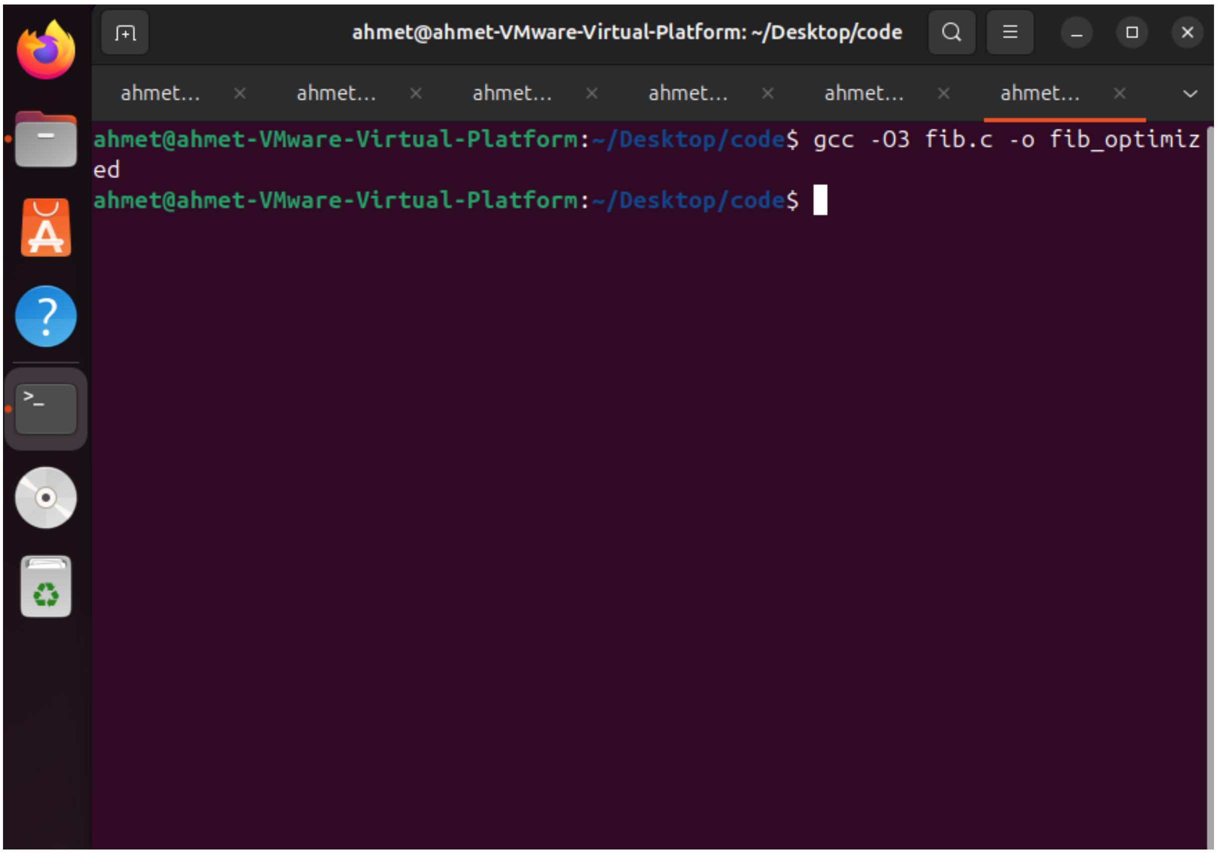
-O0: Geen optimalisatie (standaard).

-O1: Beetje optimalisatie.

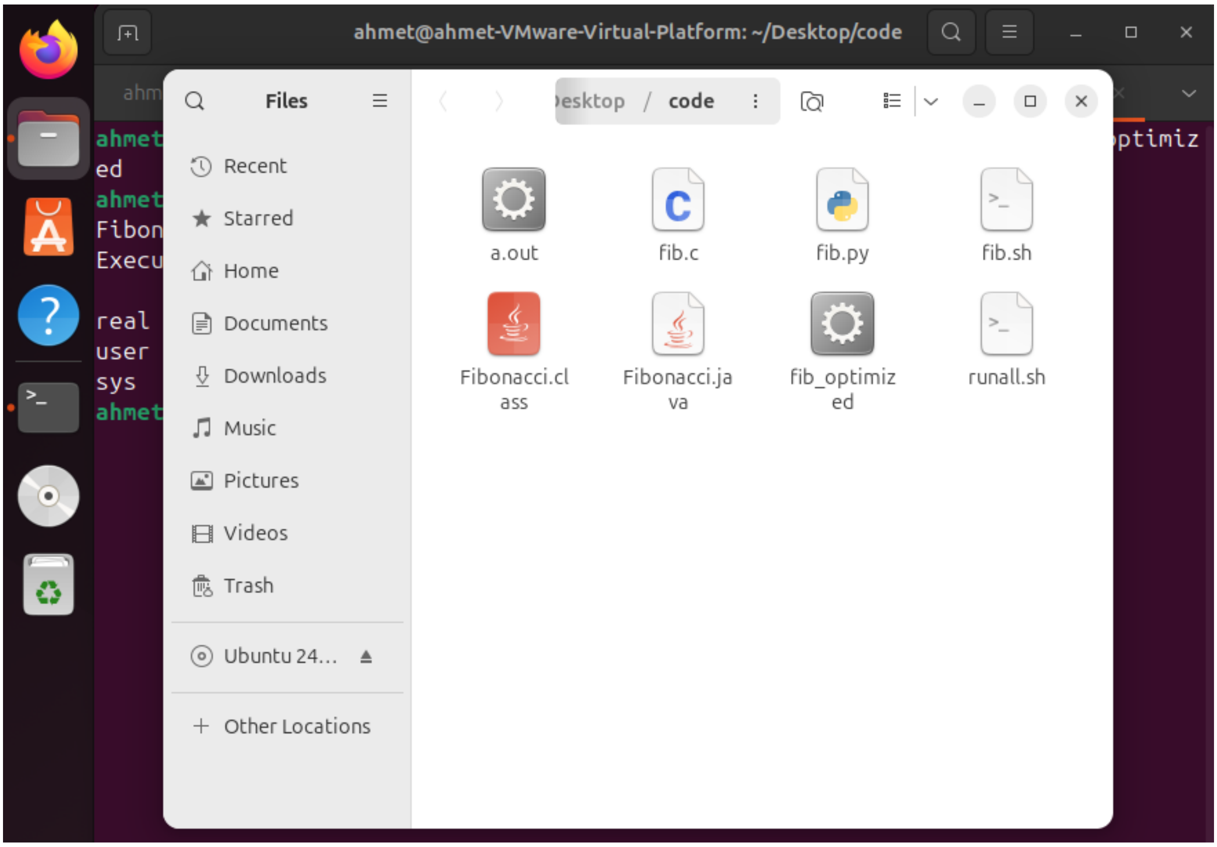
-O2: Veel optimalisatie (standaard voor programma's die je uitbrengt).

-O3: Maximale snelheid optimalisatie (agressief).

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



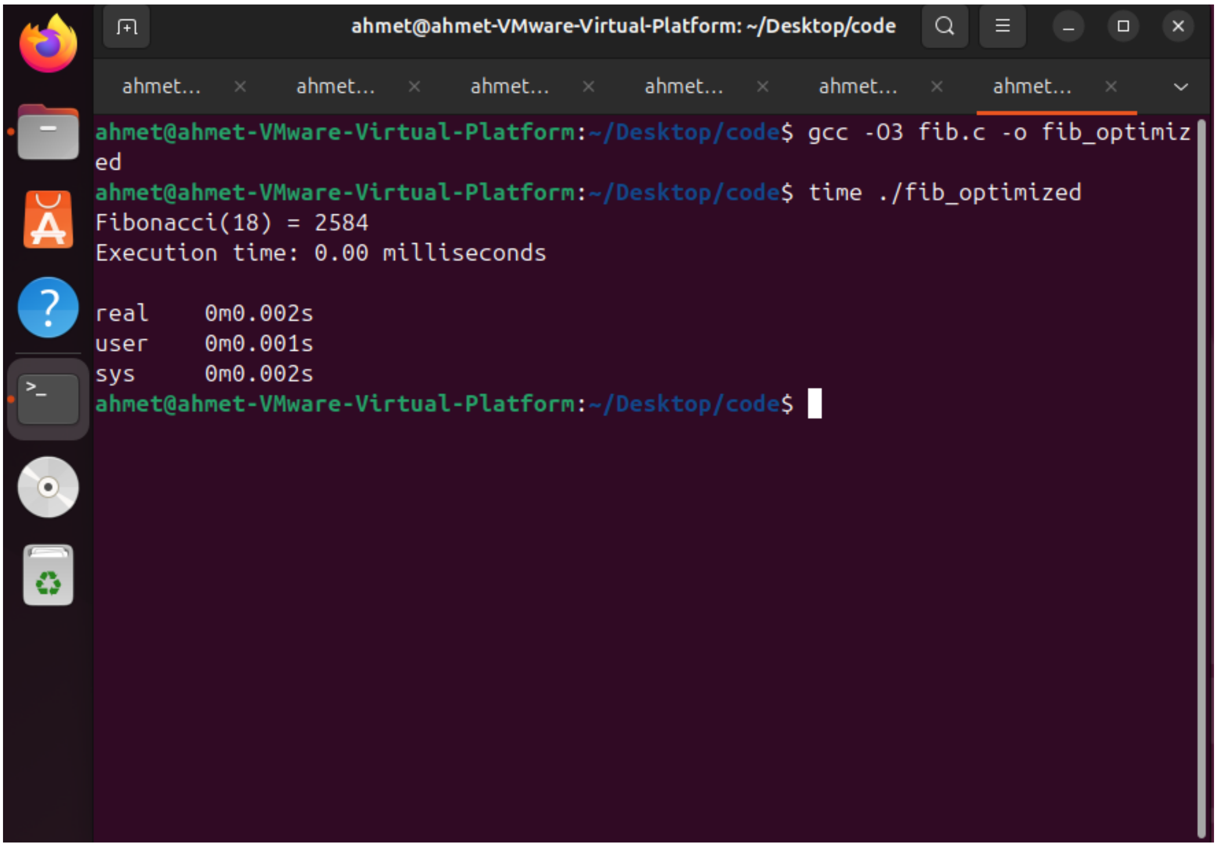
```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 fib.c -o fib_optimiz
ed
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```



```
ahmet
ed
ahmet
Fibon
Execu

real
user
sys
ahmet
```

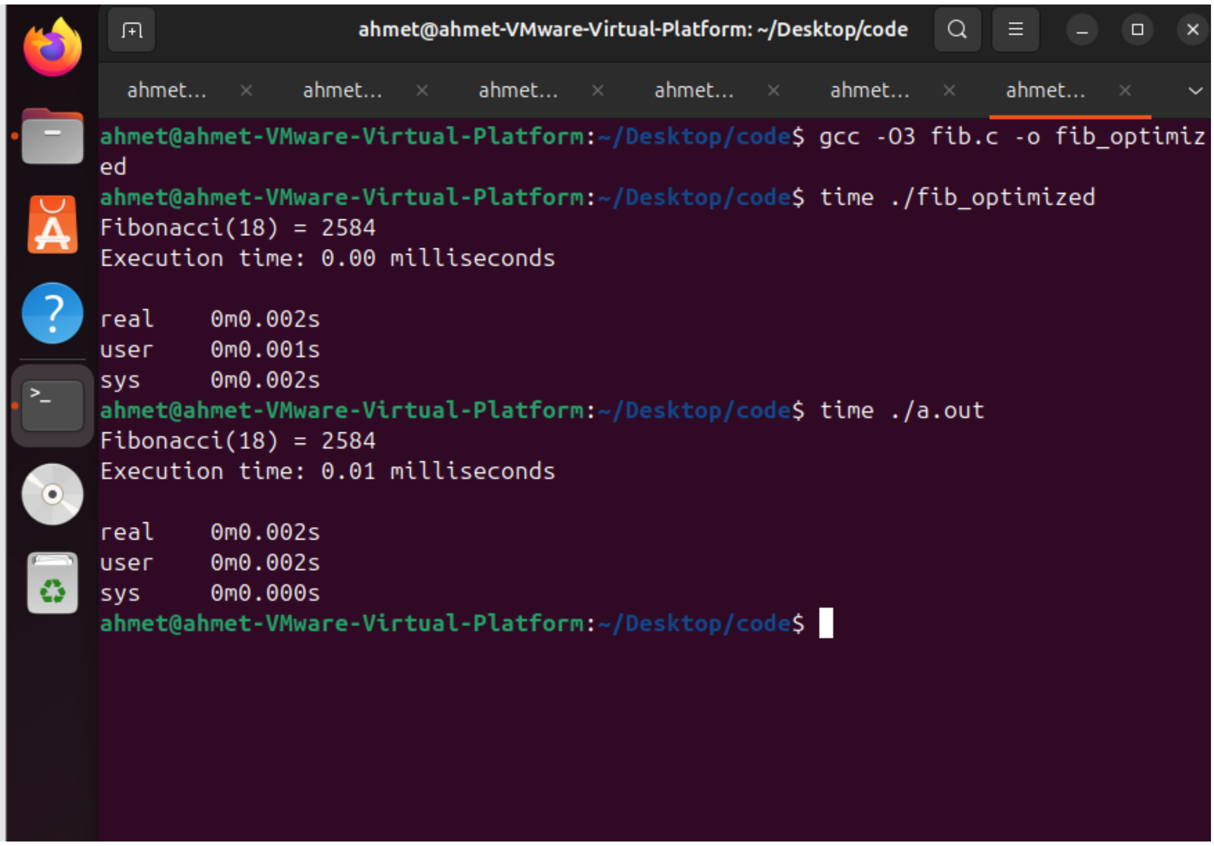
```
ptimiz
```

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



ahmet@ahmet-VMware-Virtual-Platform: ~/Desktop/code
ahmet… ahmet… ahmet… ahmet… ahmet… ahmet…

```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 fib.c -o fib_optimiz
ed
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ time ./fib_optimized
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds

real	0m0.002s
user	0m0.001s
sys	0m0.002s
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```



ahmet@ahmet-VMware-Virtual-Platform: ~/Desktop/code
ahmet… ahmet… ahmet… ahmet… ahmet… ahmet…

```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ gcc -O3 fib.c -o fib_optimiz
ed
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ time ./fib_optimized
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds

real	0m0.002s
user	0m0.001s
sys	0m0.002s
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ time ./a.out
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real	0m0.002s
user	0m0.002s
sys	0m0.000s
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ 
```

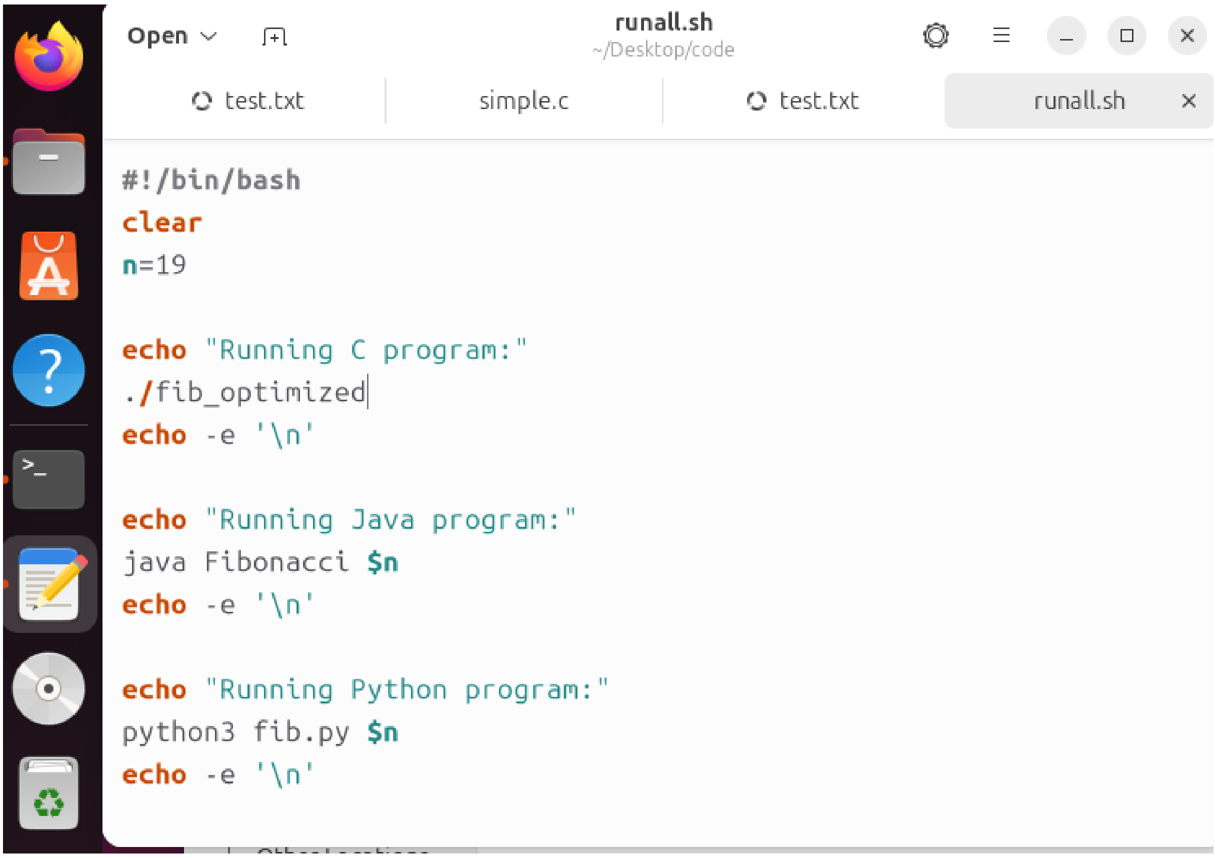
De geoptimaliseerde C-versie is sneller. Dus dat klopt

C (geoptimaliseerd): 0.00 milliseconden

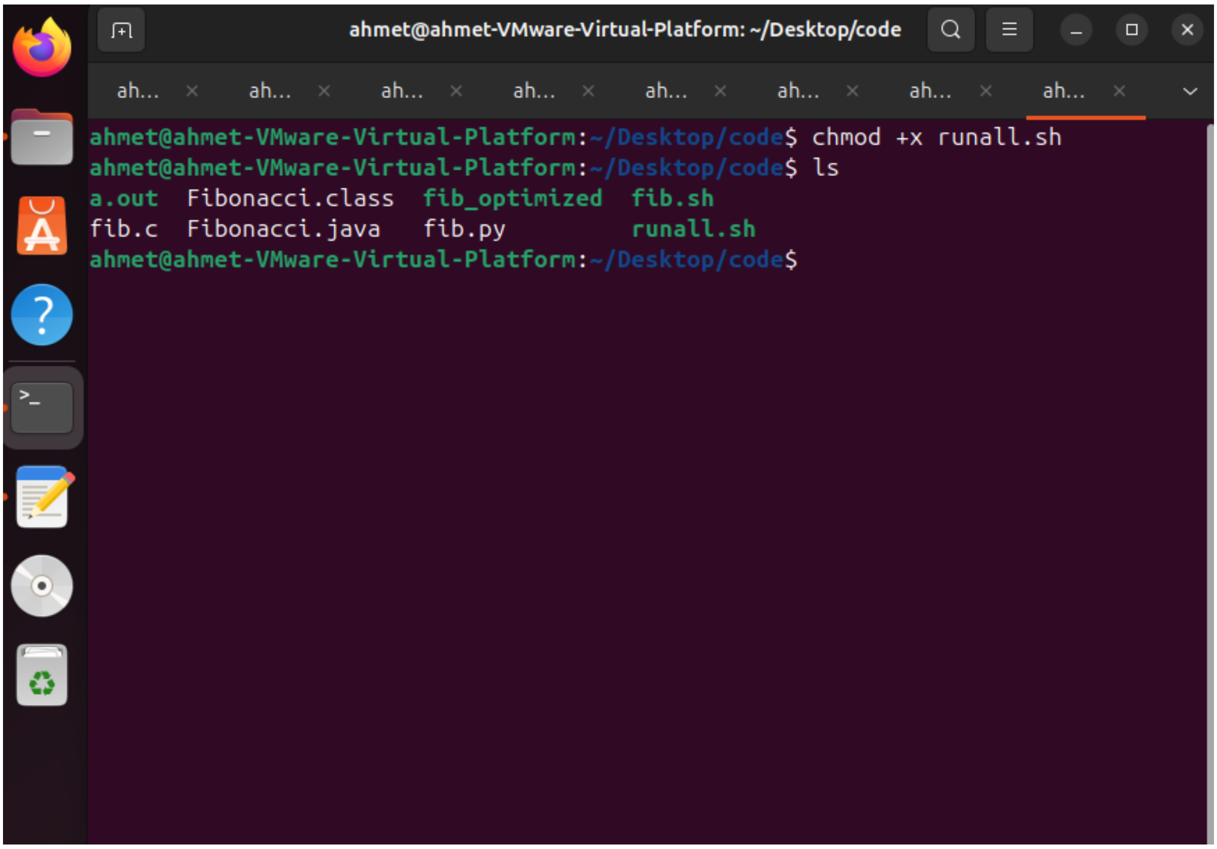
C (ongeoptimaliseerd): 0.01 milliseconden

De compiler-optimalisatie (-O3) heeft de uitvoeringstijd dus verkort.

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.

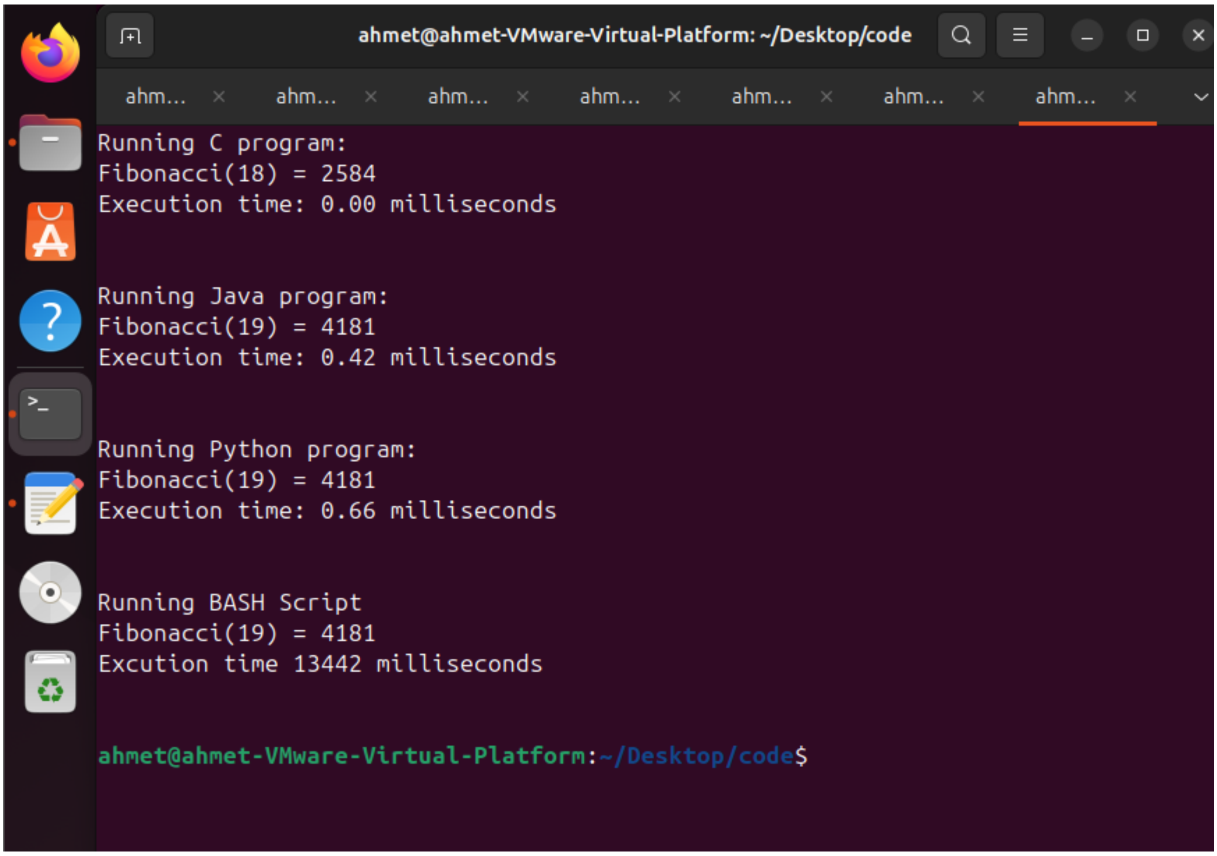


```
#!/bin/bash
clear
n=19

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

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

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



ahmet@ahmet-VMware-Virtual-Platform: ~/Desktop/code
ah... ah... ah... ah... ah... ah... ah... ah...

```
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ chmod +x runall.sh
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$ ls
a.out   Fibonacci.class  fib_optimized  fib.sh
fib.c   Fibonacci.java   fib.py         runall.sh
ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$
```



ahmet@ahmet-VMware-Virtual-Platform: ~/Desktop/code
ahm... ahm... ahm... ahm... ahm... ahm... ahm...

```
Running C program:
Fibonacci(18) = 2584
Execution time: 0.00 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.42 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.66 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 13442 milliseconds


ahmet@ahmet-VMware-Virtual-Platform:~/Desktop/code$
```

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**