Template Week 4 – Software

Student number: 572121

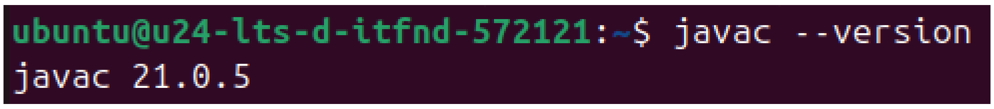
Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

Assignment 4.2: Programming languages

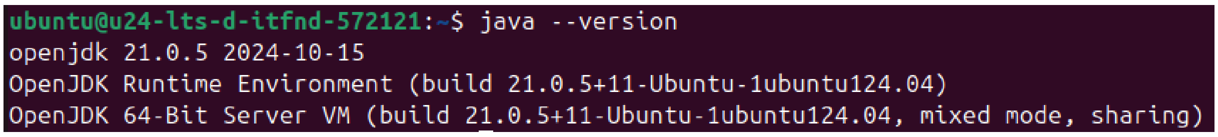
Take screenshots that the following commands work:

javac –version



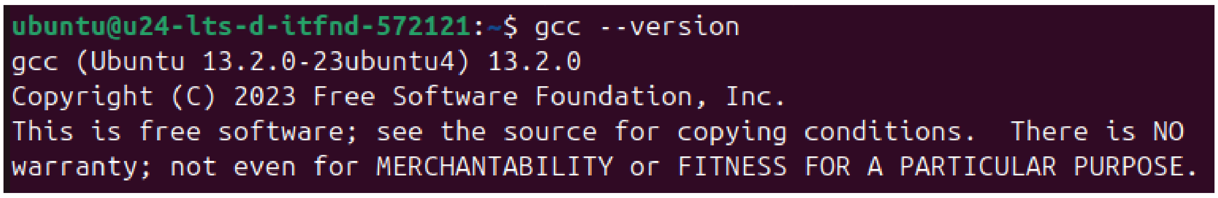
```
ubuntu@u24-lts-d-itfnd-572121:~$ javac --version
javac 21.0.5
```

java –version



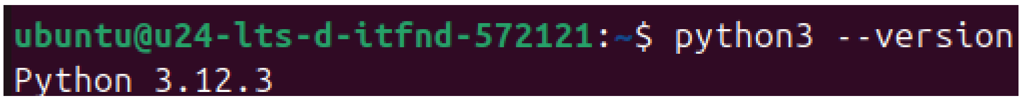
```
ubuntu@u24-lts-d-itfnd-572121:~$ java --version
openjdk 21.0.5 2024-10-15
OpenJDK Runtime Environment (build 21.0.5+11-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 21.0.5+11-Ubuntu-1ubuntu124.04, mixed mode, sharing)
```

gcc –version



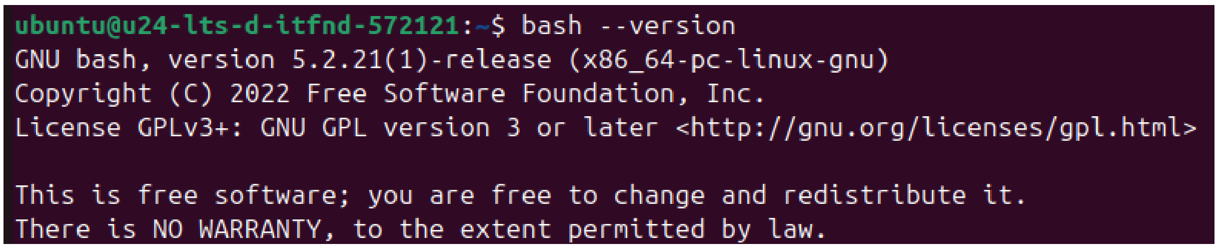
```
ubuntu@u24-lts-d-itfnd-572121:~$ gcc --version
gcc (Ubuntu 13.2.0-23ubuntu4) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

python3 –version



```
ubuntu@u24-lts-d-itfnd-572121:~$ python3 --version
Python 3.12.3
```

bash --version



```
ubuntu@u24-lts-d-itfnd-572121:~$ 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.
```

Assignment 4.3: Compile

Which of the above files need to be compiled before you can run them?
Fib.c
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
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?
javac Fibonacci.java && java Fibonacci

How do I run a Python program?
python3 fib.py

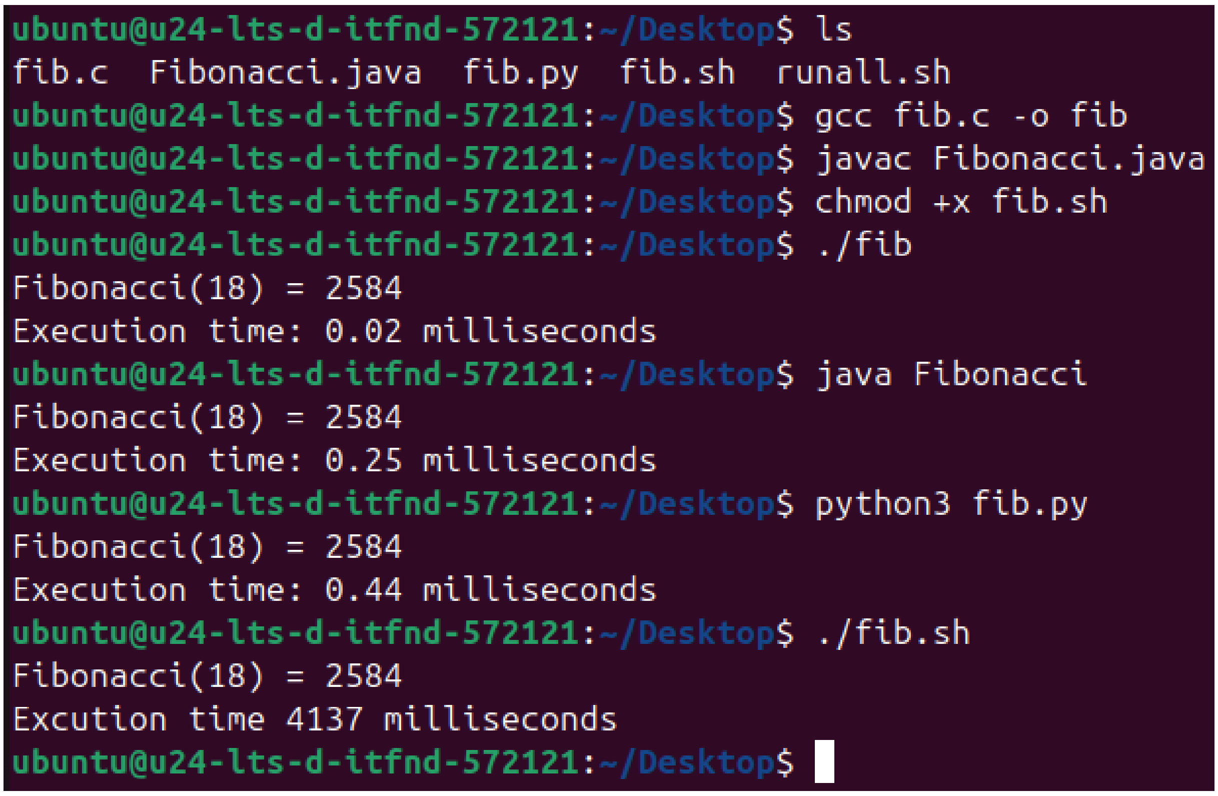
How do I run a C program?
gcc fib.c -o fib && ./fib

How do I run a Bash script?
chmod +x fib.sh (to give it the right permission)
bash fib.sh

If I compile the above source code, will a new file be created? If so, which file?
fib.c - fib
Fibonacci.java - Fibonacci.class
fib.py - No new file
fib.sh - No new file

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?



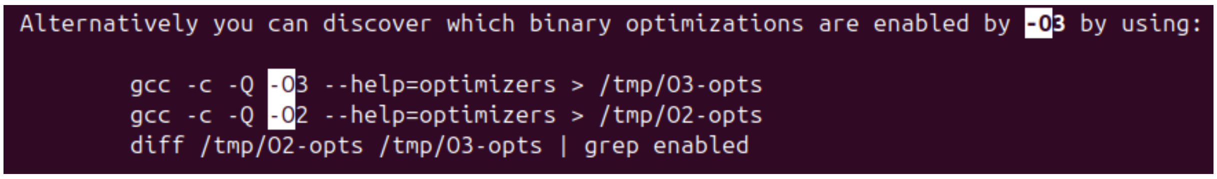
```
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ ls
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ gcc fib.c -o fib
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ javac Fibonacci.java
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ chmod +x fib.sh
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ ./fib
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.25 milliseconds
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.44 milliseconds
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ ./fib.sh
Fibonacci(18) = 2584
Excution time 4137 milliseconds
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ 
```

Fib.c was the fastest

Assignment 4.4: Optimize

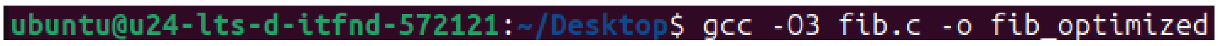
Take relevant screenshots of the following commands:

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.



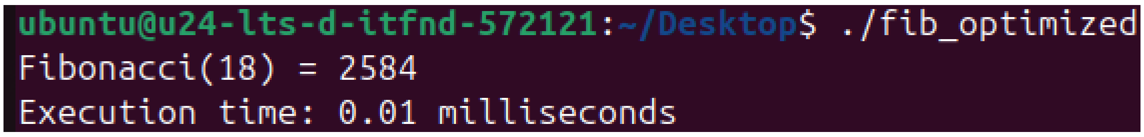
```
Alternatively you can discover which binary optimizations are enabled by -O3 by using:

        gcc -c -Q -O3 --help=optimizers > /tmp/O3-opts
        gcc -c -Q -O2 --help=optimizers > /tmp/O2-opts
        diff /tmp/O2-opts /tmp/O3-opts | grep enabled
```

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



```
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ gcc -O3 fib.c -o fib_optimized
```

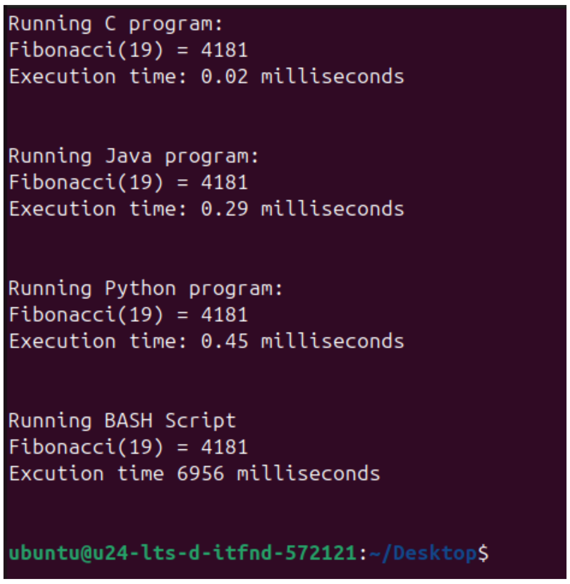
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



```
ubuntu@u24-lts-d-itfnd-572121:~/Desktop$ ./fib_optimized
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
```

Yes, it went down from 0.2 milliseconds to 0.1 milliseconds

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.



```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.02 milliseconds


Running Java program:
Fibonacci(19) = 4181
Execution time: 0.29 milliseconds


Running Python program:
Fibonacci(19) = 4181
Execution time: 0.45 milliseconds


Running BASH Script
Fibonacci(19) = 4181
Excution time 6956 milliseconds


ubuntu@u24-lts-d-itfnd-572121:~/Desktop$
```

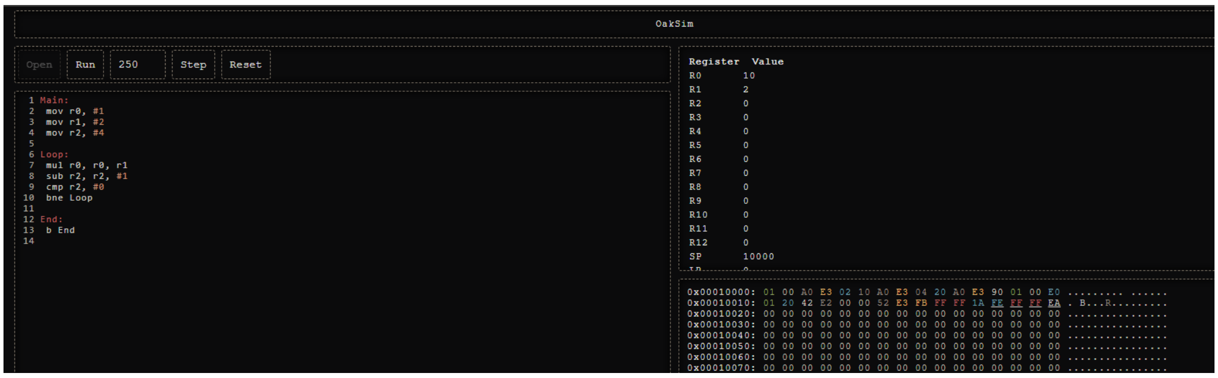
Bonus point assignment – week 4

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 r0, #1
 mov r1, #2
 mov r2, #4

Loop:
 mul r0, r0, r1
 sub r2, r2, #1
 cmp r2, #0
 bne Loop

End:
 b End
```

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

Screenshot of the completed code here.



```
Main:
 mov r0, #1
 mov r1, #2
 mov r2, #4

Loop:
 mul r0, r0, r1
 sub r2, r2, #1
 cmp r2, #0
 bne Loop

End:
 b End
```

```
Register  Value
R0        10
R1        2
R2        0
R3        0
R4        0
R5        0
R6        0
R7        0
R8        0
R9        0
R10       0
R11       0
R12       0
SP        10000
```

```
0x00010000: 01 00 A0 E3 02 10 A0 E3 04 20 A0 E3 90 01 00 E0 ......... ......
0x00010010: 01 20 42 E2 00 00 52 E3 FB FF FF 1A FE FF FF EA . B...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 ................
```

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