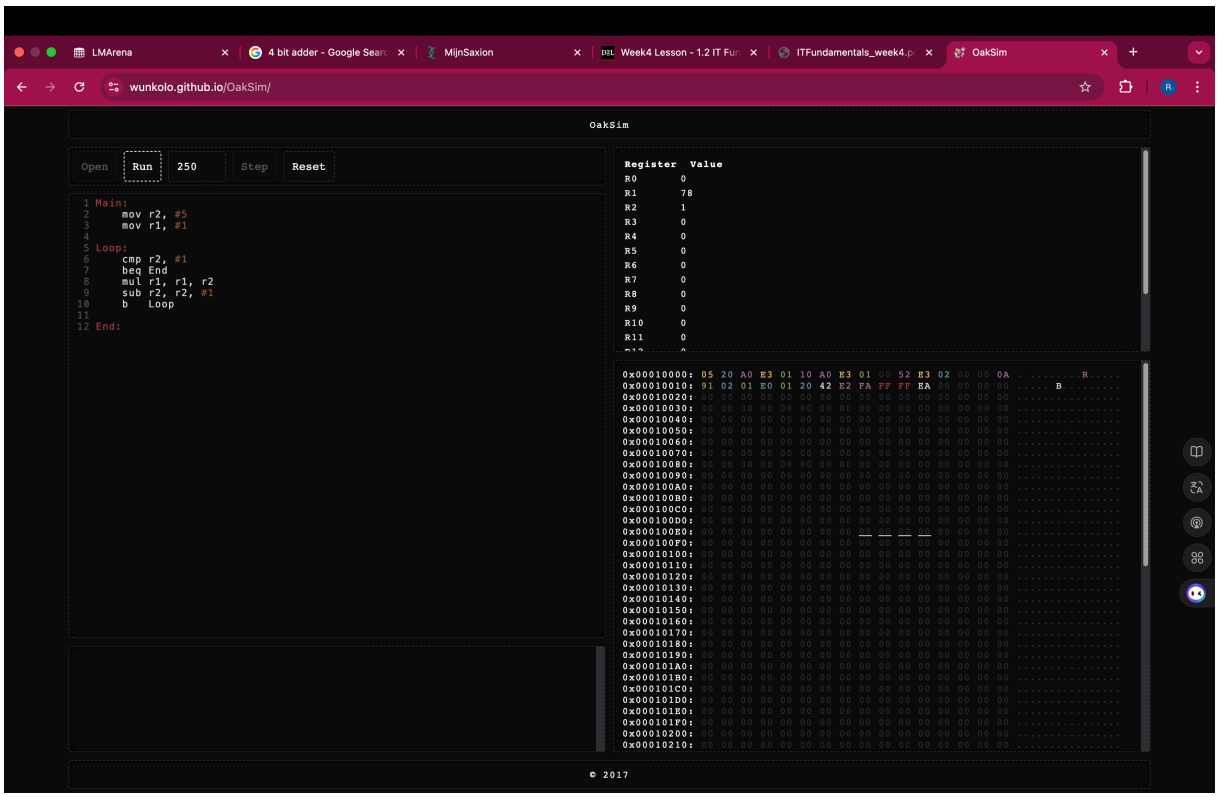
Week 4 – Software

Student number: 585732

Assignment 4.1: ARM assembly

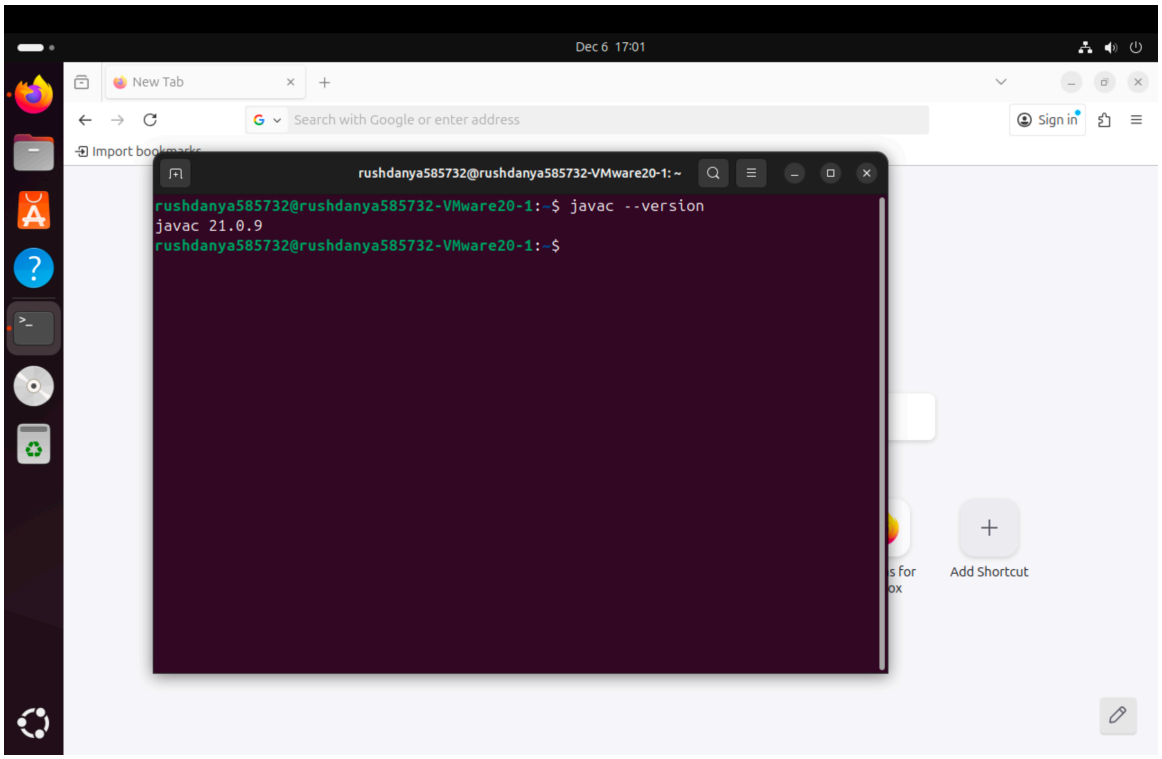
Screenshot of working assembly code of factorial calculation:



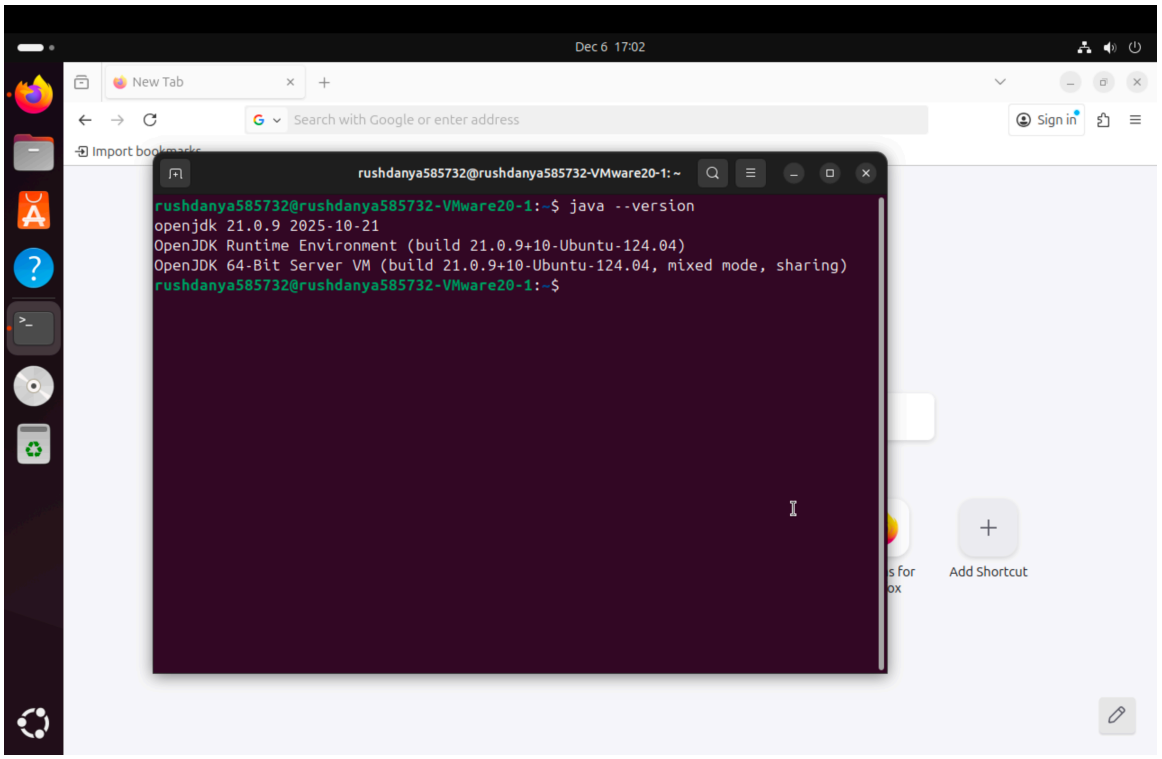
Assignment 4.2: Programming languages

Take screenshots that the following commands work:

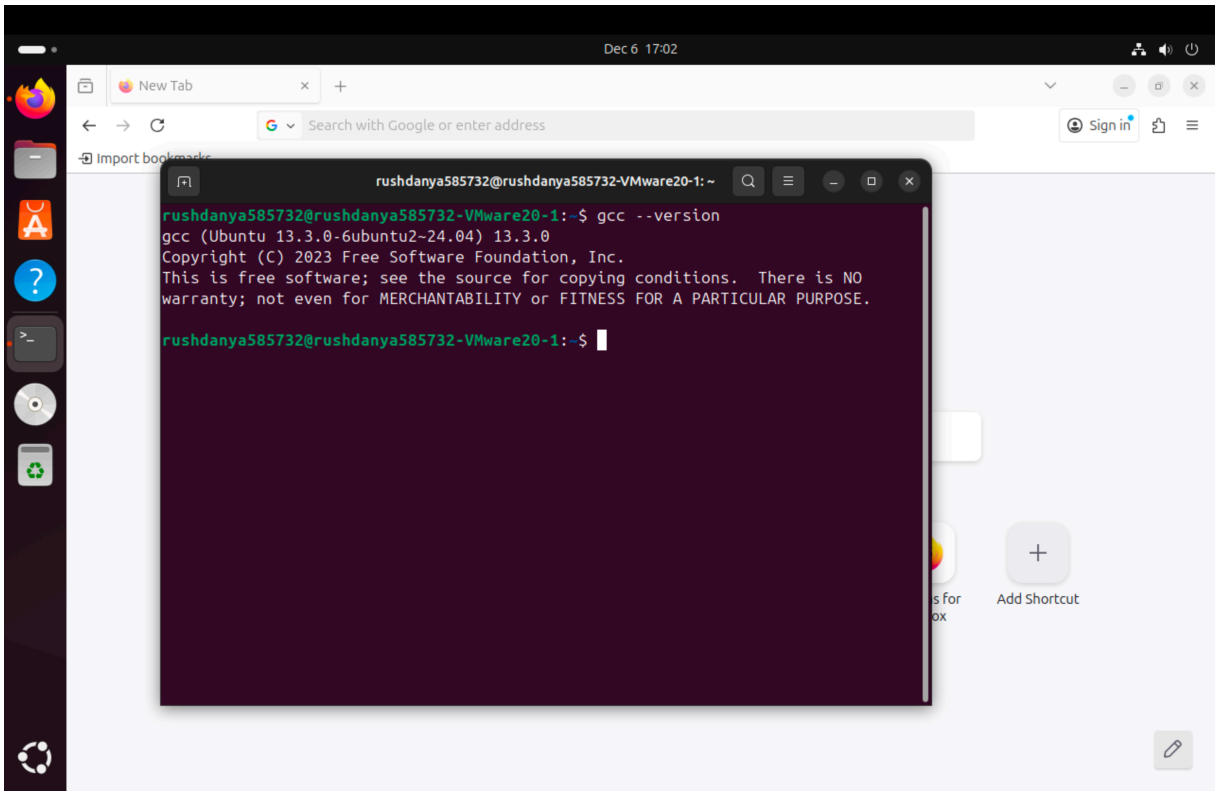
javac –version



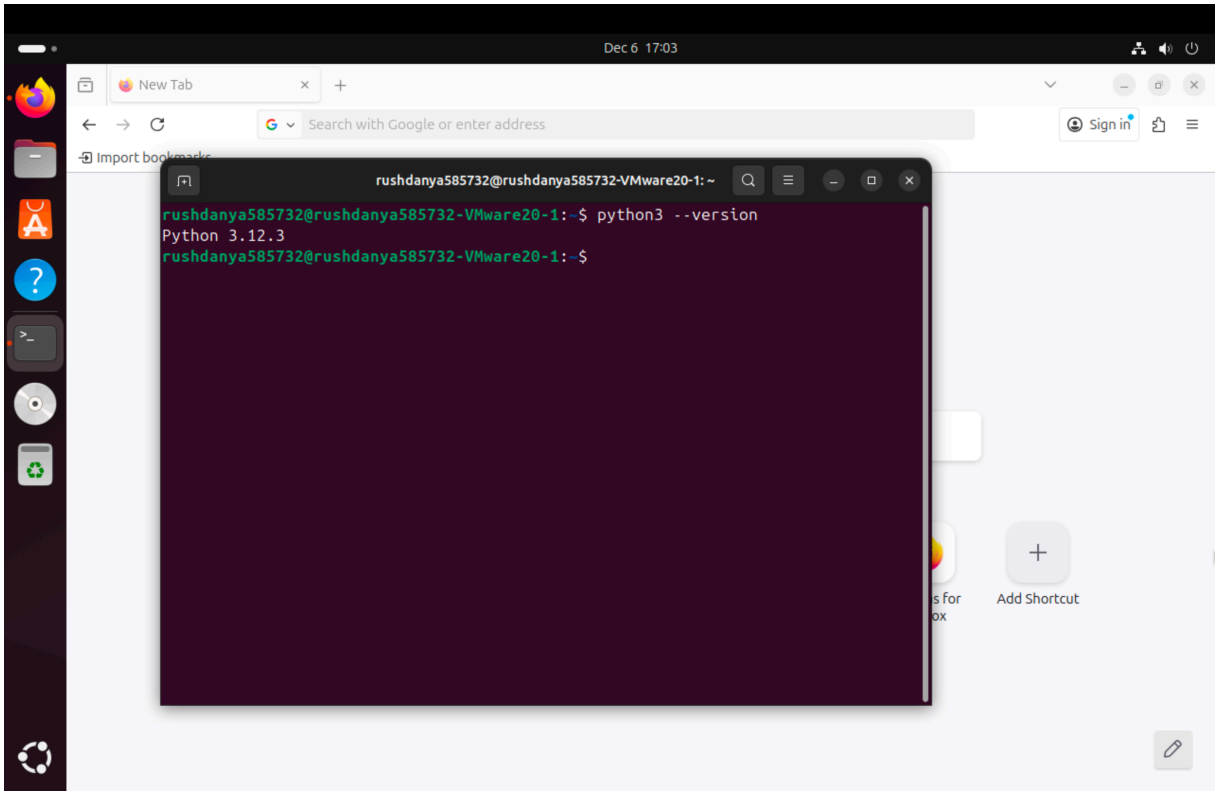
java –version



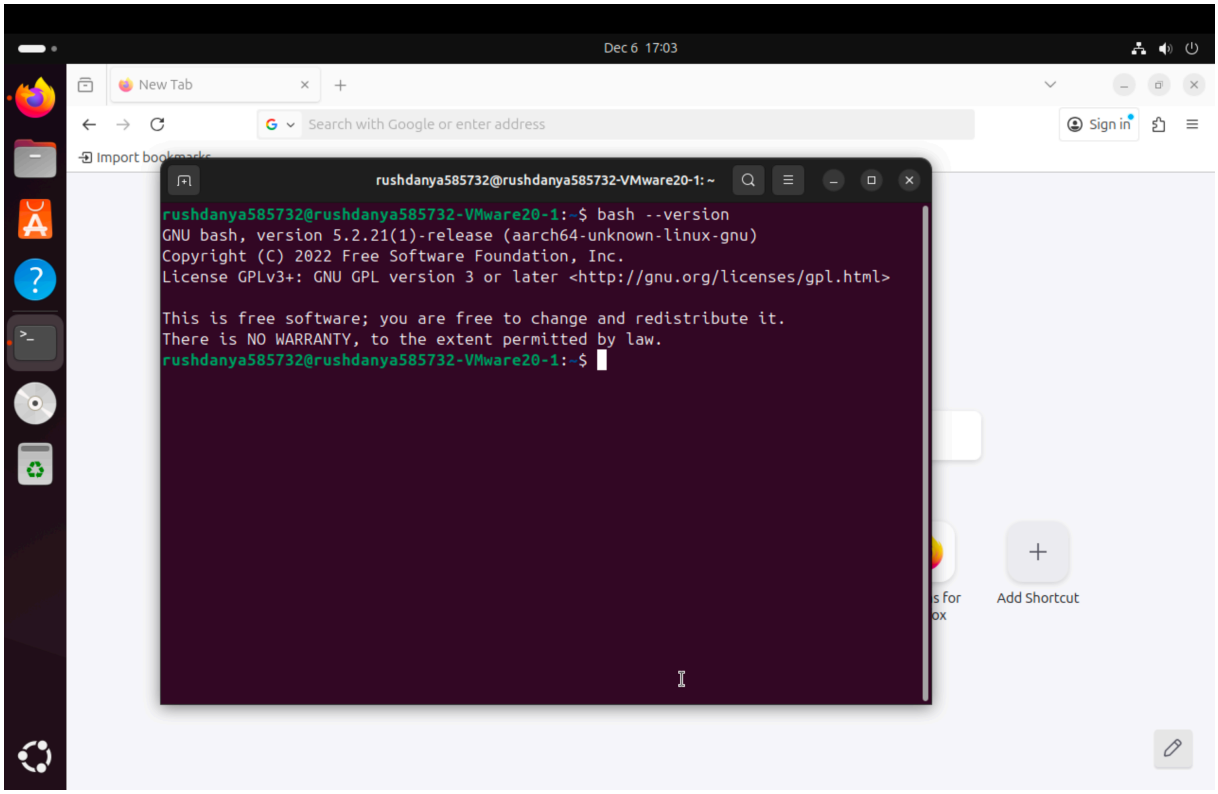
gcc --version



python3 –version



bash --version



Assignment 4.3: Compile

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

= Fibonacci.java and fib.c

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

= The C program (fib.c, after complilation)

Which source code files are compiled to byte code?

= The Java program (Fibonacci.java into Fibonacci.class)

Which source code files are interpreted by an interpreter?

= The Python program(fib.py) and the Bash script (fib.sh)

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

= The compiled C program.

How do I run a Java program?

= 1. Compile: javac Fibonacci.java

2. Run: java Fibonacci

How do I run a Python program?

= python3 fib.py

How do I run a C program?

= Compile: gcc fib.c -o fib_c

How do I run a Bash script?

= 1. Make executable: chmod a+x fib.sh

2. Run: ./fib.sh (or bash fib.sh)

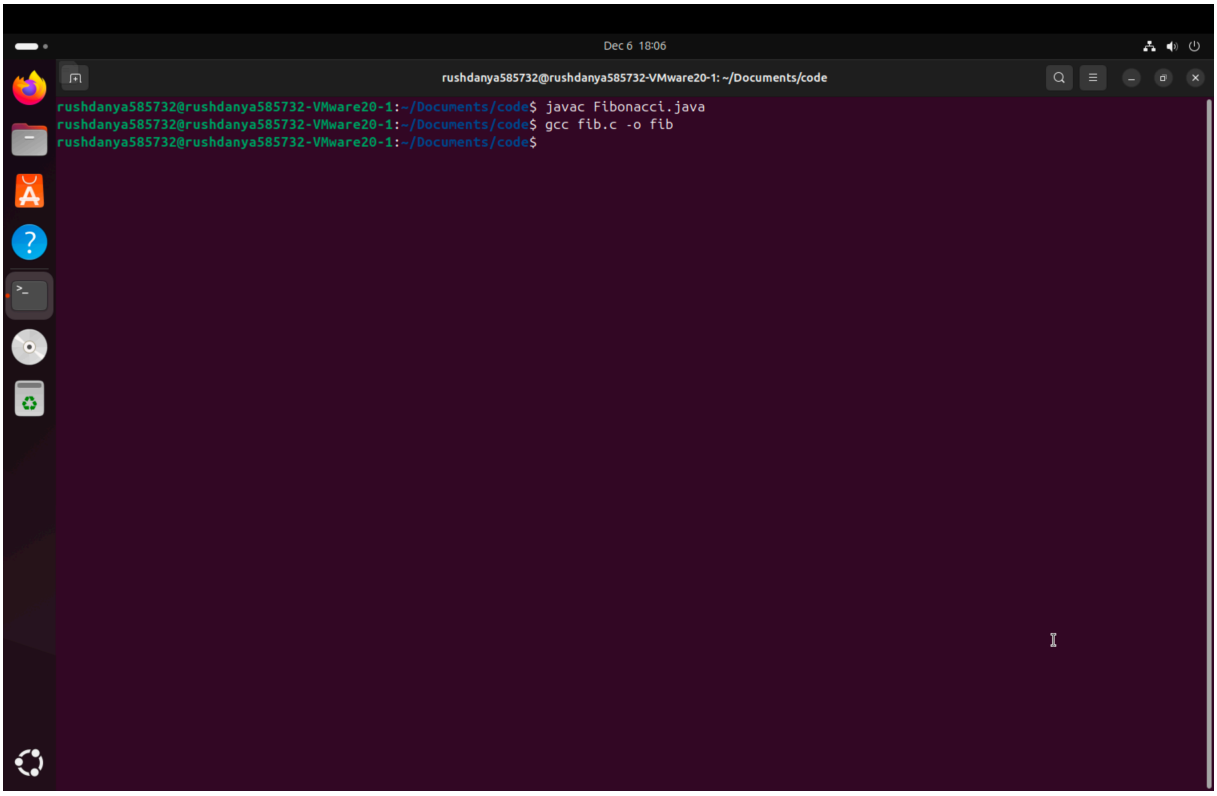
If I compile the above source code, will a new file be created? If so, which file?

= Compiling Fibonacci.java creates Fibonacci.class

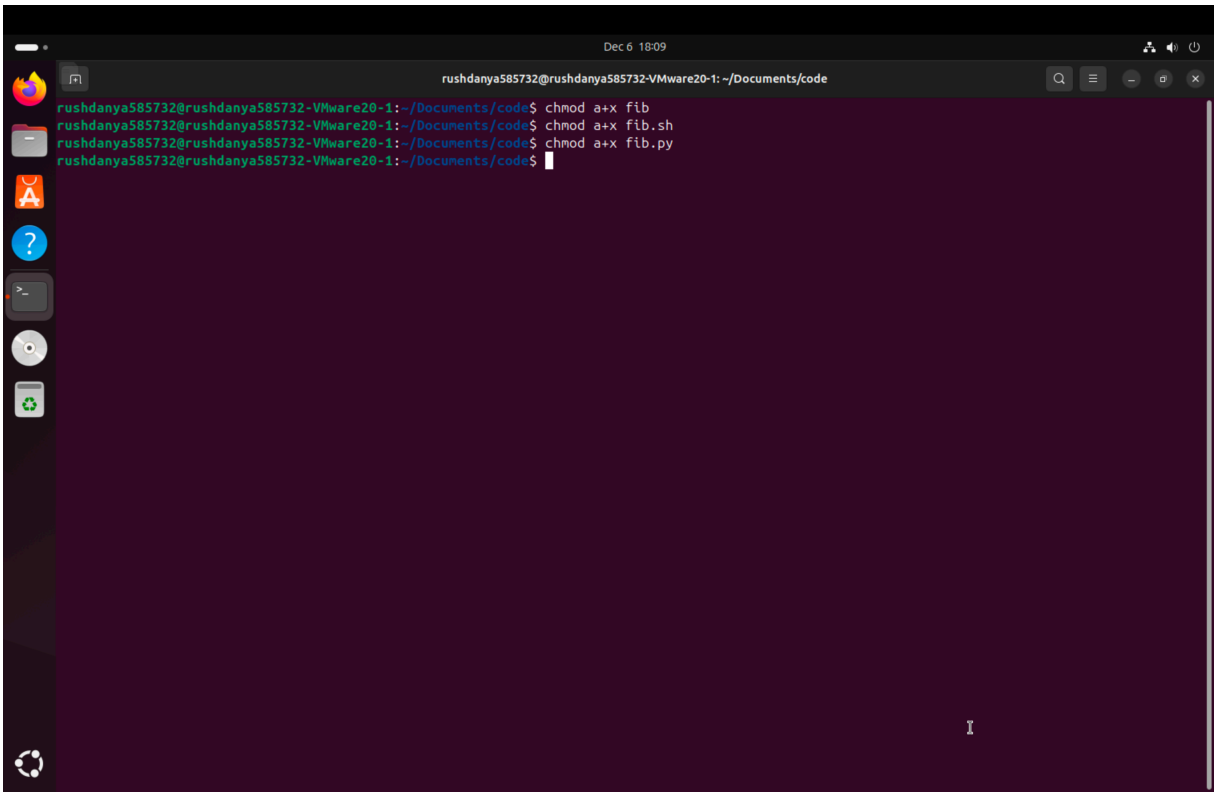
Compiling fib.c (example: gcc fib.c -o fib_c) creates the executable fib_c .

Take relevant screenshots of the following commands:

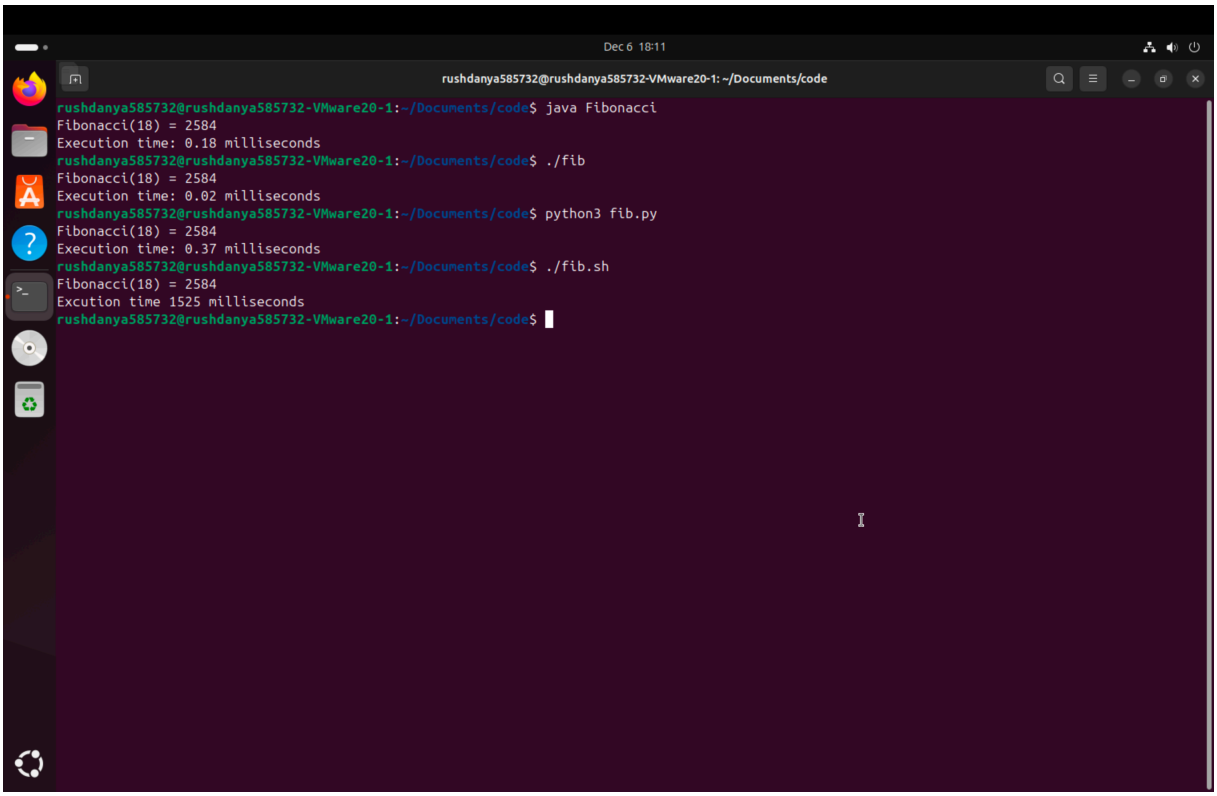
- Compile the source files where necessary



- Make them executable



- Run them



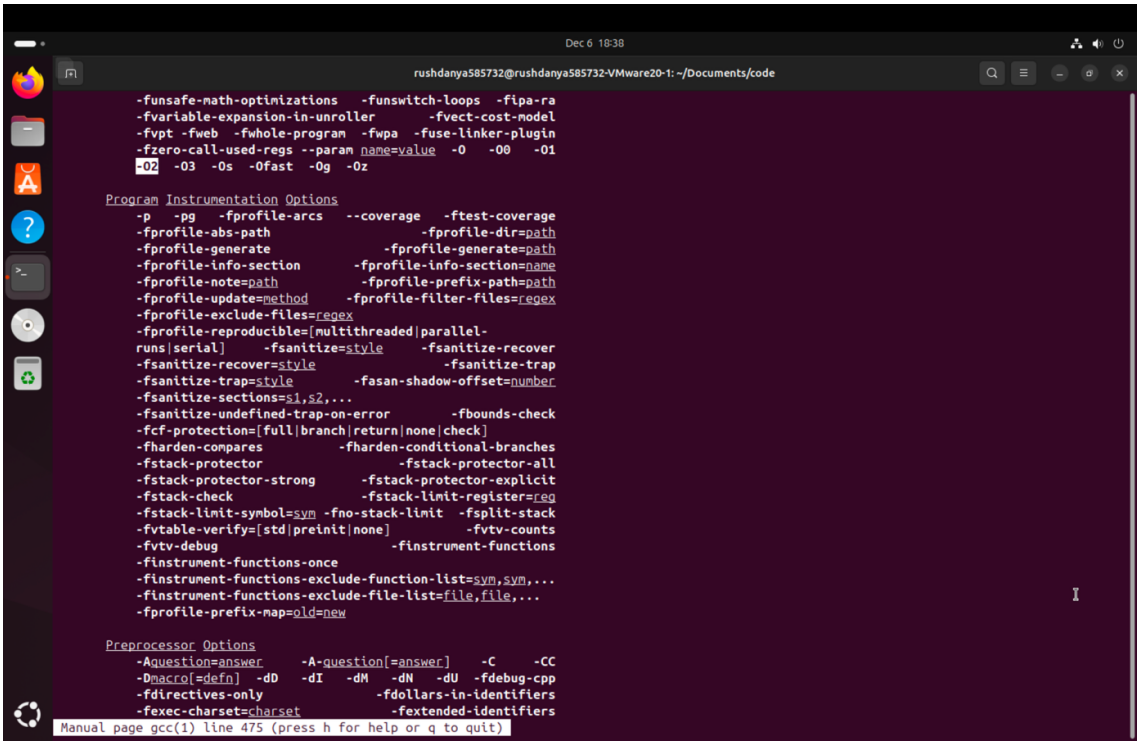
- Which (compiled) source code file performs the calculation the fastest?

  After compiling and running all the programs, we can see that the C version turned out to be the fastest. In my test, it completed the Fibonacci calculation in about 0.02 milliseconds, while the Java version took about 0.18 milliseconds. So the compiled souce code file that performs the calculation the fastest is fib.c.

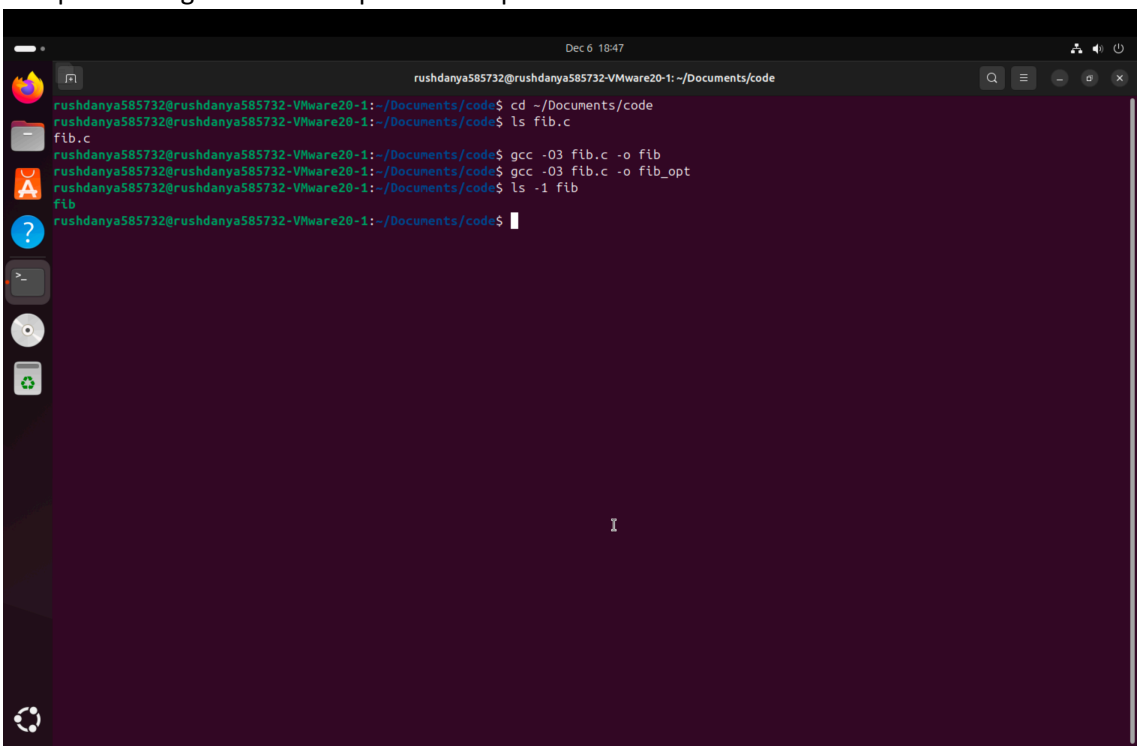
Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

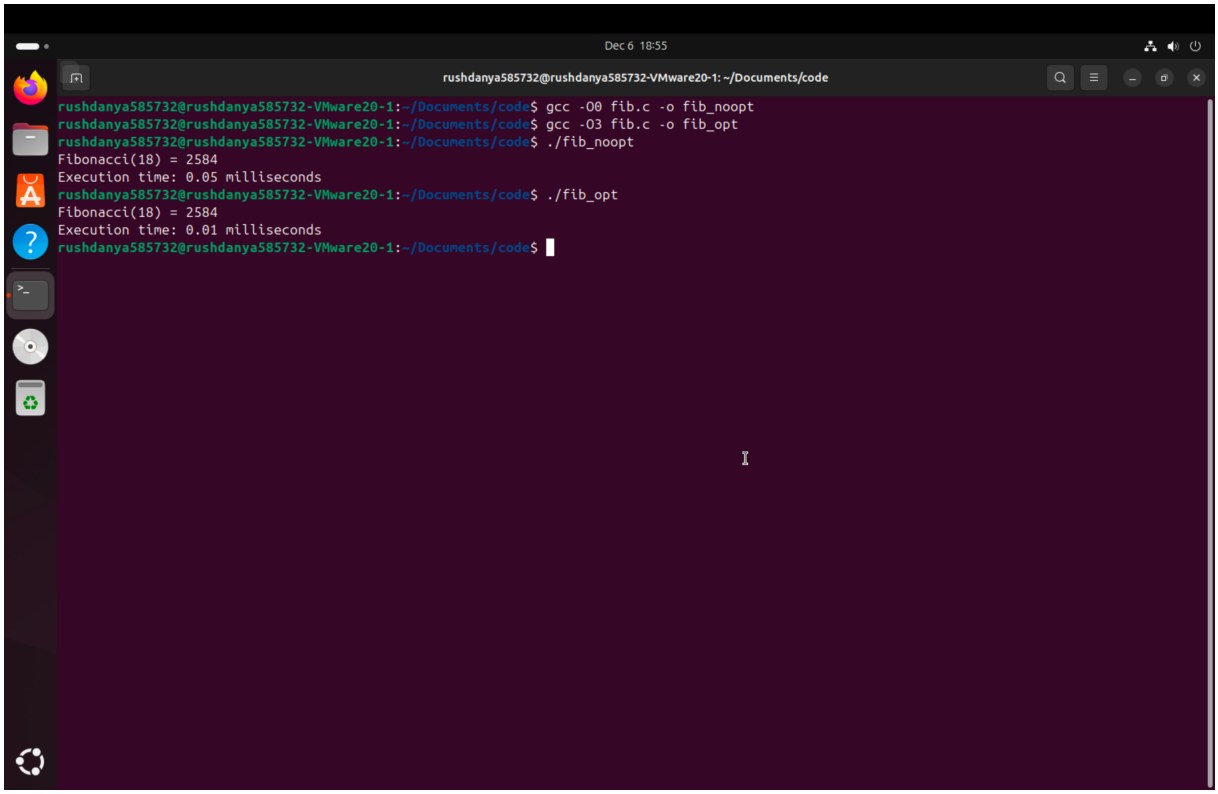
a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book, but find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



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

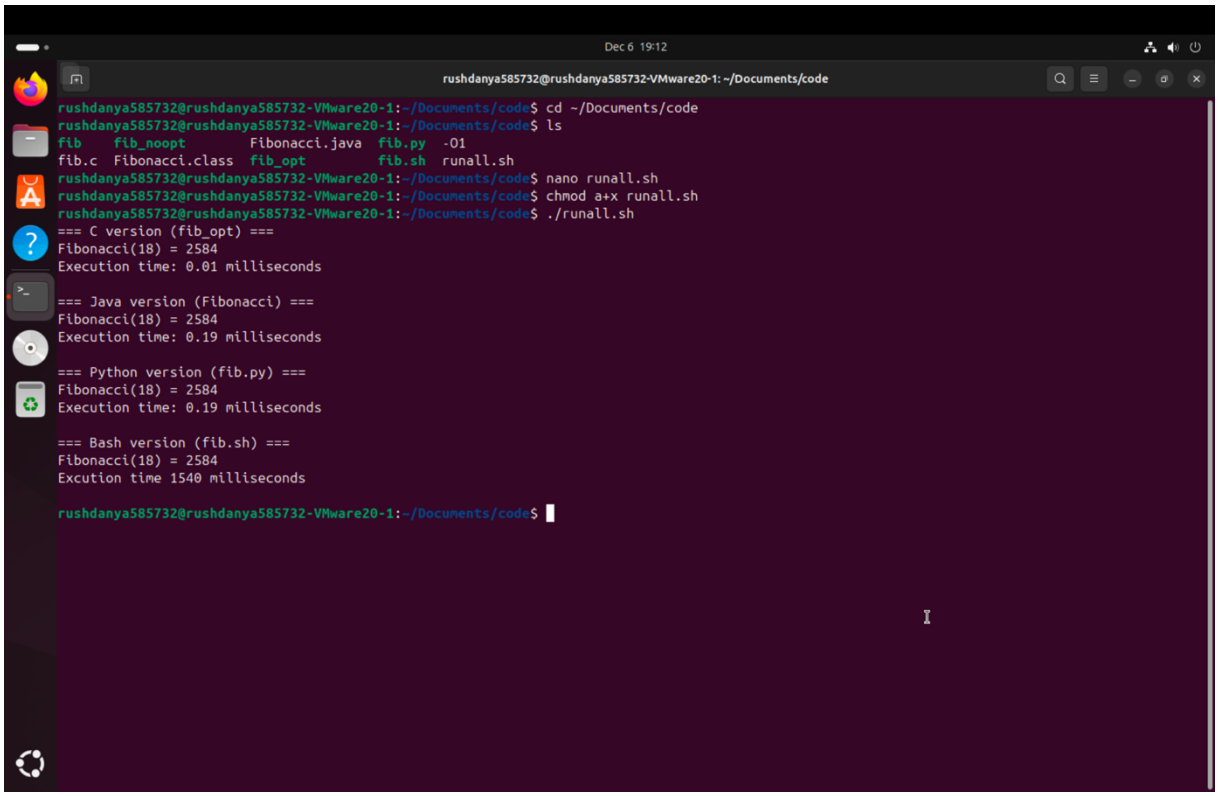


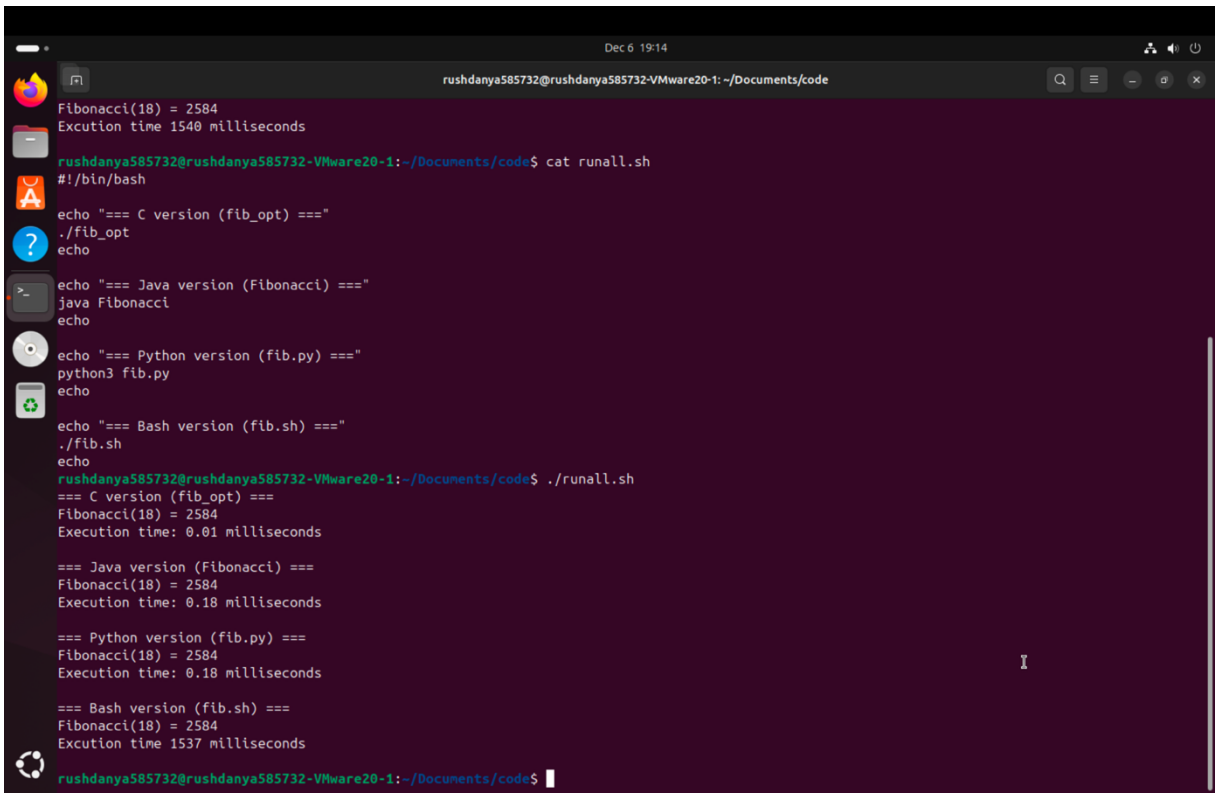
c) Run the newly compiled program. Is it true that it now performs the calculation faster?



Yes, it performs the calculation faster after compiling with optimisations. In my run, the unoptimised version (fib_noopt) took about 0.05 ms, while the optimised version (fib_opt, compiled with -O3) took about 0.01 ms for the same input (Fibonacci(18) = 2584). This shows the optimised compiled program runs faster.

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.





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, #1

Loop:
    mul r0, r0, r1
    subs r2, r2, #1
    bne Loop
End:
```

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

Screenshot of the completed code here.