Genome 540 discussion

January 7th, 2025
Joe Min

Genome Sciences
UNIVERSITY OF WASHINGTON

Agenda

Discussion structure

Homework advice

Choosing a language

Managing programming environments

Discussion structure

Discussion structure

Technical topic of the day (30m)

- Ideas relevant to homework
- Interesting or thought provoking issues

Office hours (20m)

- Fully optional
- Homework/general questions

Homework advice

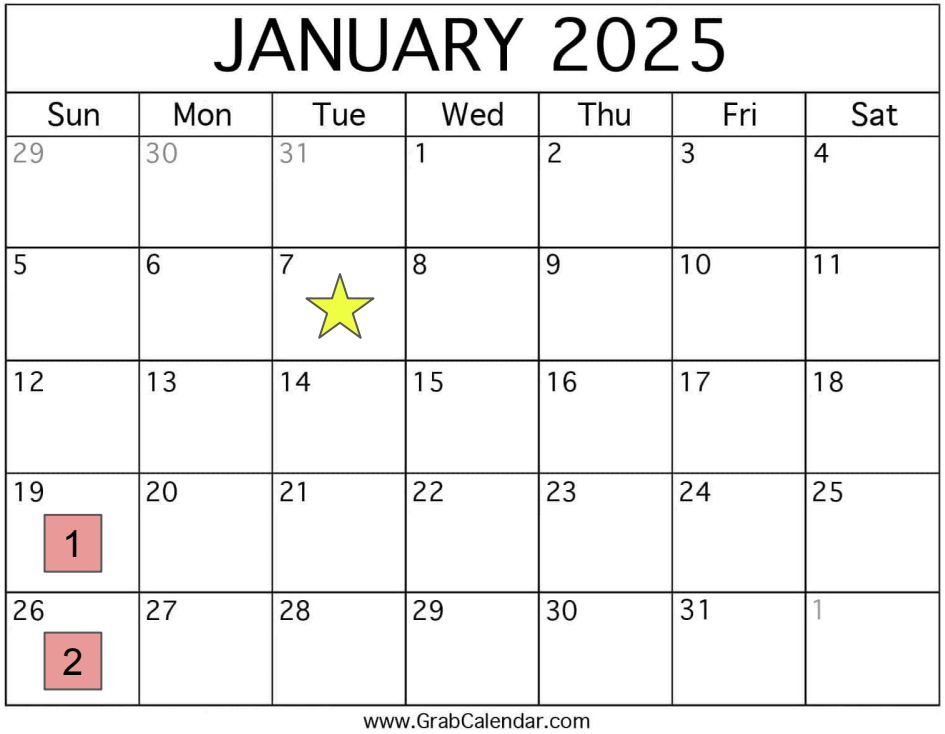
Start early!

Start early

- Especially the first assignment

Submit early

- You will receive feedback within 1 day and can resubmit



Using A.I.

Do use it as a tool

- Translating python to C++
- Learning syntax for a new language
- Debugging specific problems
  - Use it like a quicker version of stack overflow

Don’t ask it to do your assignment

- You won’t learn anything if it does all the thinking
- If it’s wrong, debugging might be harder than doing the assignment

Write readable code

Use intuitive variable/function names

x = 0 vs. number_of_friends = 0

Comments

- As headers describing functional chunks

  # this calculates the number of friends in your life from your phone's data

- To describe complex lines of code

  # this is just a verbose way to say zero

  num_my_friends = (a^-exp(24*b))/5 - (a^-exp(24*b))/5

How to approach assignments

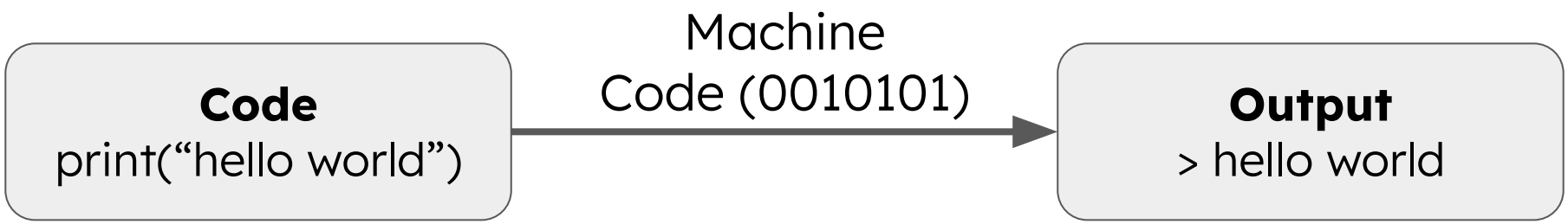
1. Understand the algorithm
2. Outline your code
    a. Start to think of your code in the abstract first and write a skeleton
3. Fill it in
4. Evaluate if things are working with small tests
    a. We will provide some test inputs; try to think of additional edge cases
5. Compare your results on the test data against the provided test results

Choosing a language

Which language should I use?

- You are free to choose
- Most people use C++, C, or Python
- Why one over the other?

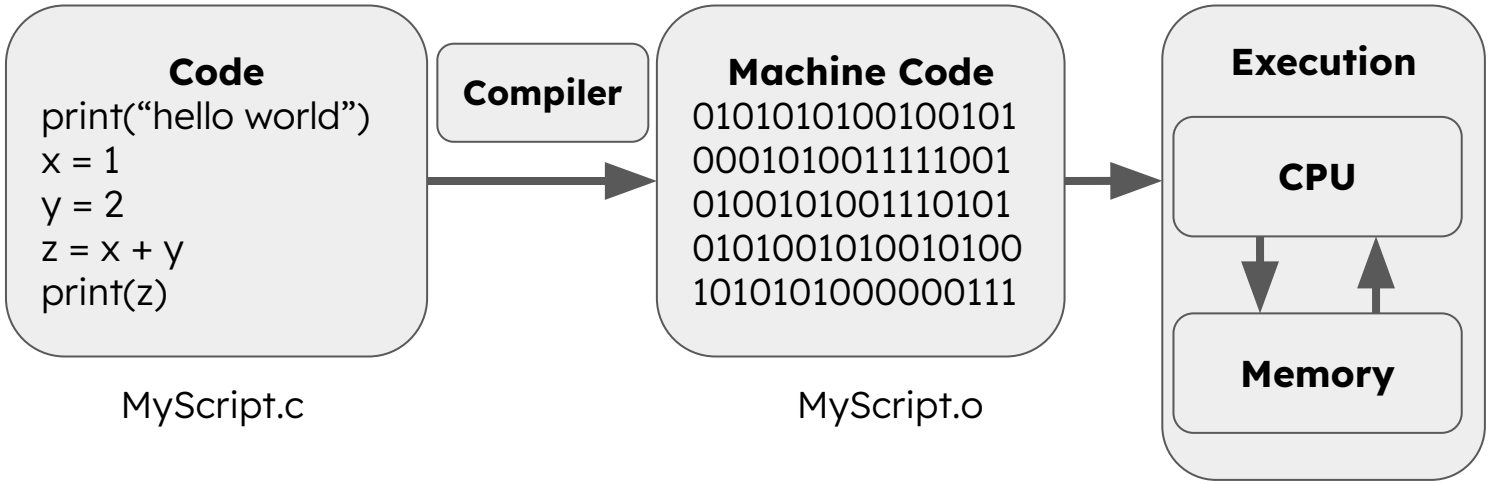
Compiled vs. Interpreted Languages



https://www.youtube.com/watch?v=_C5AHaS1mOA

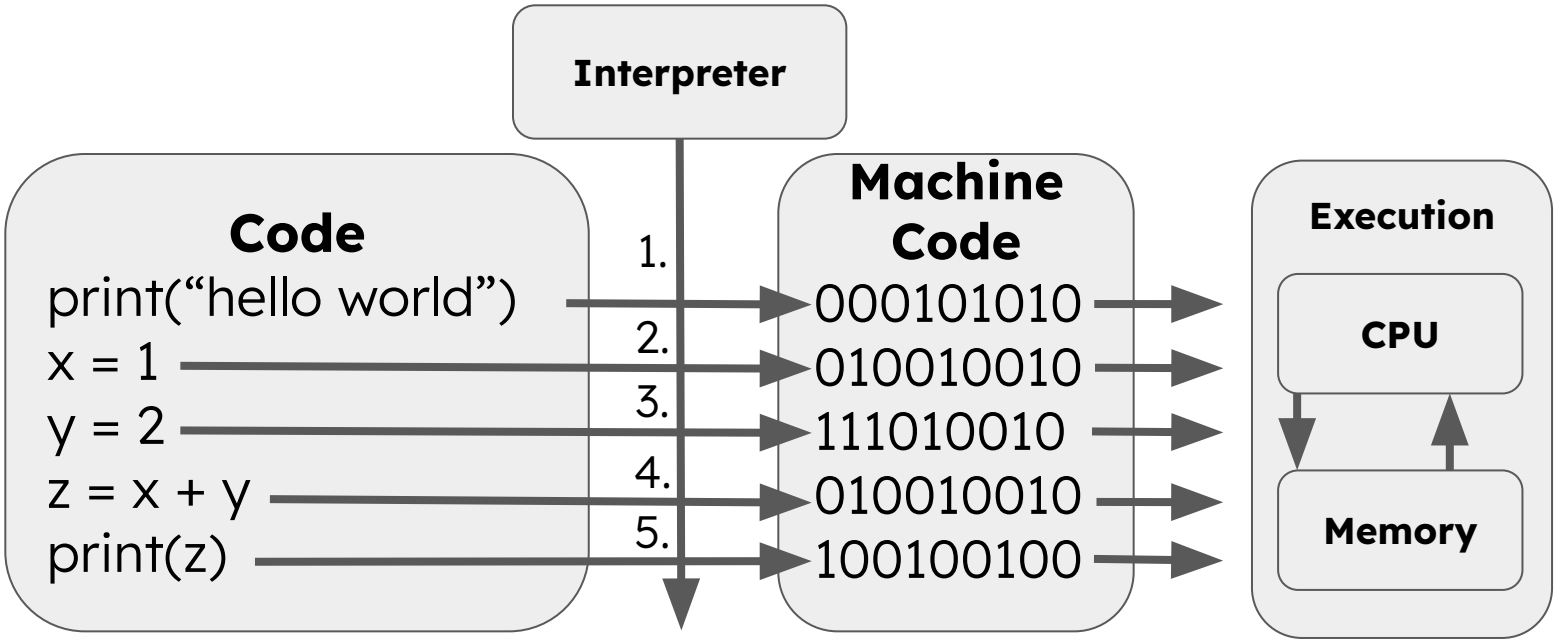
*** The following explanations are gross oversimplifications

Compiled vs. Interpreted Languages



- Compiler translates code into machine code
- Machine code can be run over and over (assuming correct OS/architecture)

Compiled vs. **Interpreted** Languages



- Program executed line by line at runtime
- Need an interpreter to run program

Static vs. Dynamic, Strong vs. Weak

Python is a dynamic strongly typed language

- Don’t need to declare type: x = 5

```
x = 5
y = 3.14

# Strong typing: This will raise a TypeError
sum_result = x + y  # TypeError: unsupported operand type(s) for +: 'int' and 'float'
```

C++ is a static weakly typed language

- Need to declare type: int x = 5;

```
int x = 5;
double y = 3.14;

// Weak typing: The compiler allows implicit conversion
double sum = x + y;  // 'x' is implicitly converted to 'double' before addition
```

Which language should I use?

Your choice

- C++ will give the biggest improvement on the 1st assignment
- C++ will be ~10x faster in pretty much all cases
- Python will work for all assignments but you have to know how to get around limitations of the language
- Python will be easier to learn/write/debug

Managing programming environments

Environments for new languages and workflows

Phenomenon: different programs have different and often conflicting dependencies (language/library/OS)

Problem: computers generally only like to have one version of each thing otherwise things get messy

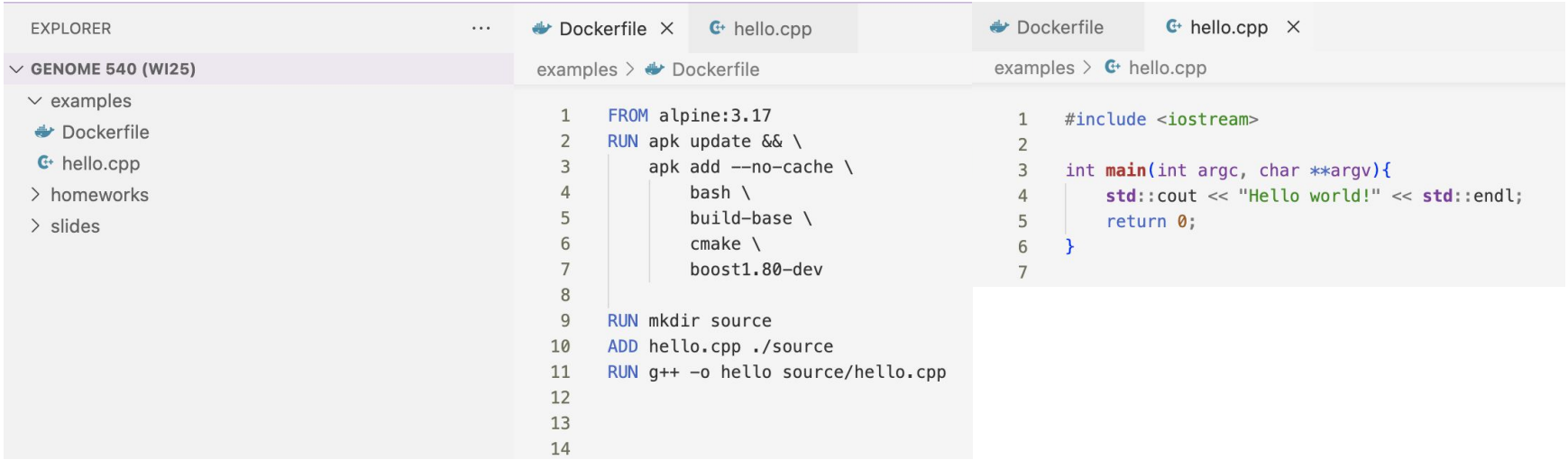
Existing solutions: virtual machines; virtual environments; containers

Docker (or Apptainer) containers

Containers are isolated environments that have virtual operating systems (OS)

- Can install everything from scratch and ensure it’s the same every time
- Can distribute these environments for others to use (no more “well it works on my computer!”)
- Can easily be scaled up for variable use (e.g., just spin up another container to handle more web traffic)

Docker/Apptainer containers



Dockerfiles shows exactly how the environment is set up, including files you may want inside your running container

Images vs. containers

Images

- Defined in Dockerfiles
- The “permanent” version (i.e., the “DNA”)

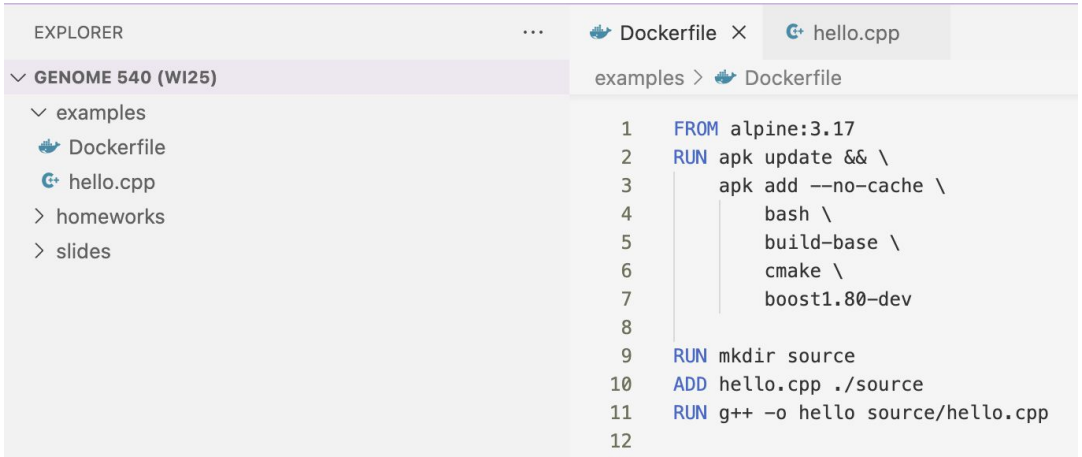
Containers

- Essentially just functional copies of images (i.e., the “RNA”)
- Designed to be ephemeral and easily replaced

Building images

docker build -t hello_image /examples/

- -t “tags” your image with a name so it’s easily referenced
- /examples/ is the directory containing the Dockerfile and hello.cpp



Running containers

Dockerfile | hello.cpp

examples > Dockerfile

```
1   FROM alpine:3.17
2   RUN apk update && \
3       apk add --no-cache \
4           bash \
5           build-base \
6           cmake \
7           boost1.80-dev
8
9   RUN mkdir source
10  ADD hello.cpp ./source
11  RUN g++ -o hello source/hello.cpp
12
```

Dockerfile | hello.cpp

examples > hello.cpp

```
1   #include <iostream>
2
3   int main(int argc, char **argv){
4       std::cout << "Hello world!" << std::endl;
5       return 0;
6   }
7
```

```
josephmin ~ 🍉 docker run -it hello /bin/bash
1df8a82c556e:/# ls
bin     hello   media   proc    sbin    sys     var
dev     home    mnt     root    source  tmp
etc     lib     opt     run     srv     usr
1df8a82c556e:/# ./hello
Hello world!
```

source directory is still there from building the image

As is the executable we made on line 11

The program can run from inside the container!

Can get complicated, but if done right, can be a lot easier than managing multiple environments

Potential discussion topics

What do you want to learn more about?

- Scalable bioinformatics pipelines (Snakemake)
- General programming tips
- Specific languages: Python, C++, Unix tools
- Dynamic programming
- Machine learning
- Version control/Github

Next time

Getting started in C++

- Pointers and why they’re important

Getting around limitations in Python

- Simulating pointers
- Overriding classes