FC II

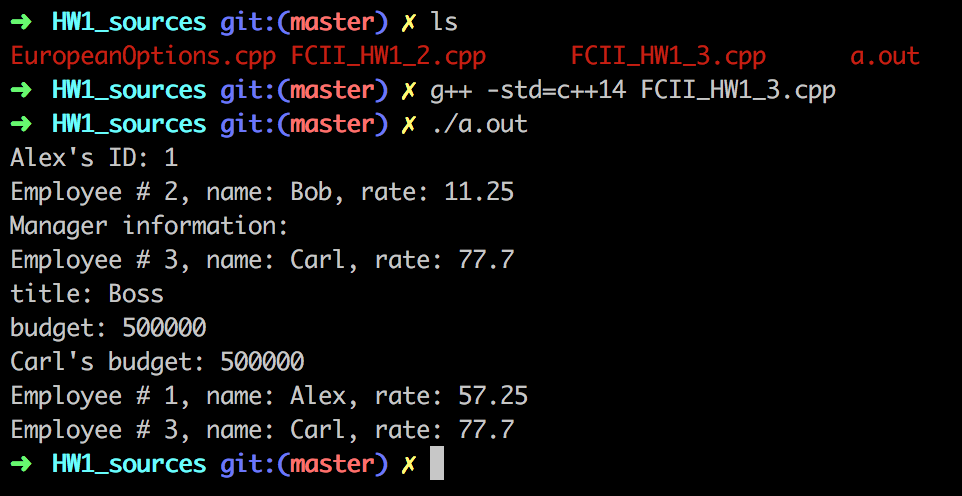
Group:

1. Harveen Oberoi
2. Daniel Rojas Coy
3. Lucas Duarte Bahia
4. Jordan Giebas

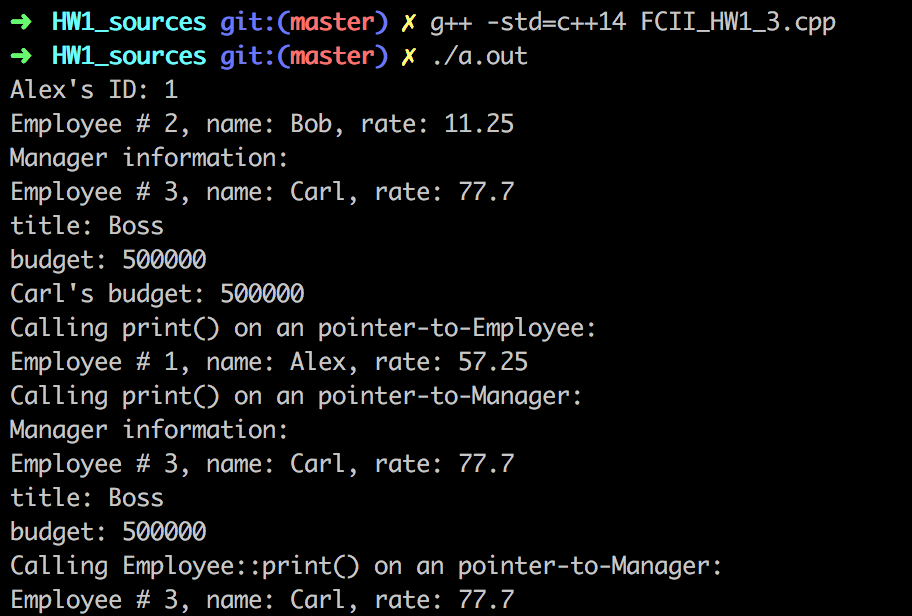
*Homework 1*

Problem 3

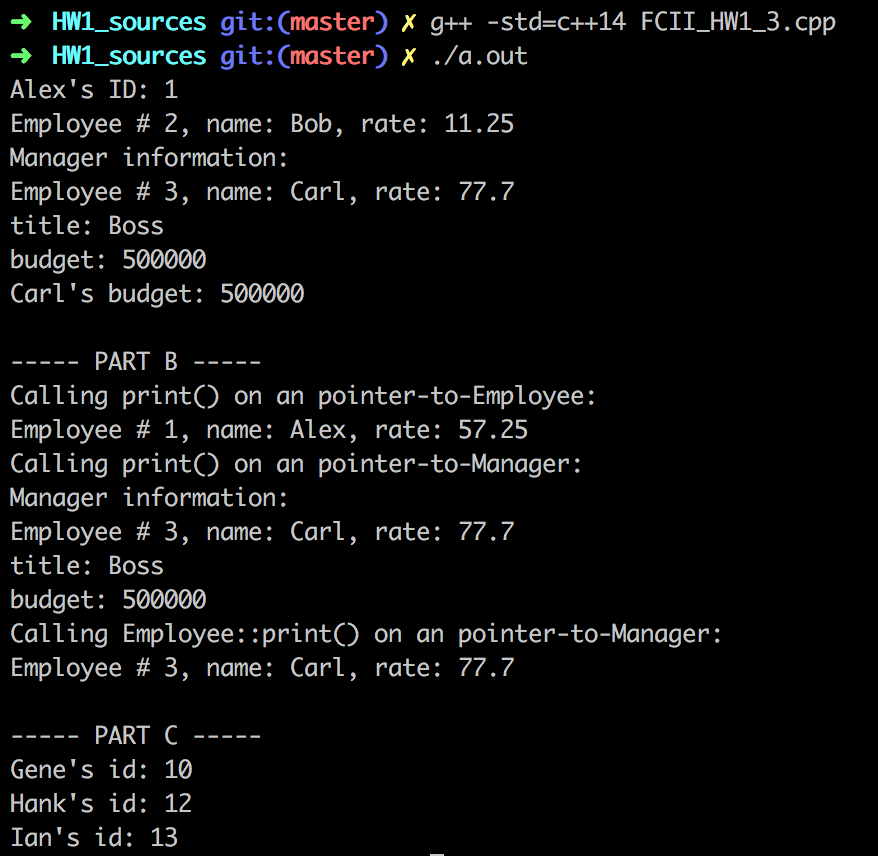
1. Below is a screenshot of my terminal that confirms that “**Manager** “is a kind of” **Employee**, and that it is okay to have a pointer-to-**Employee** point to a **Manager**. However, because **print()** in the base class is ***not*** declared **virtual**, the base class **print()** is called when a pointer-to-**Employee** contains the address of a **Manager** object.”



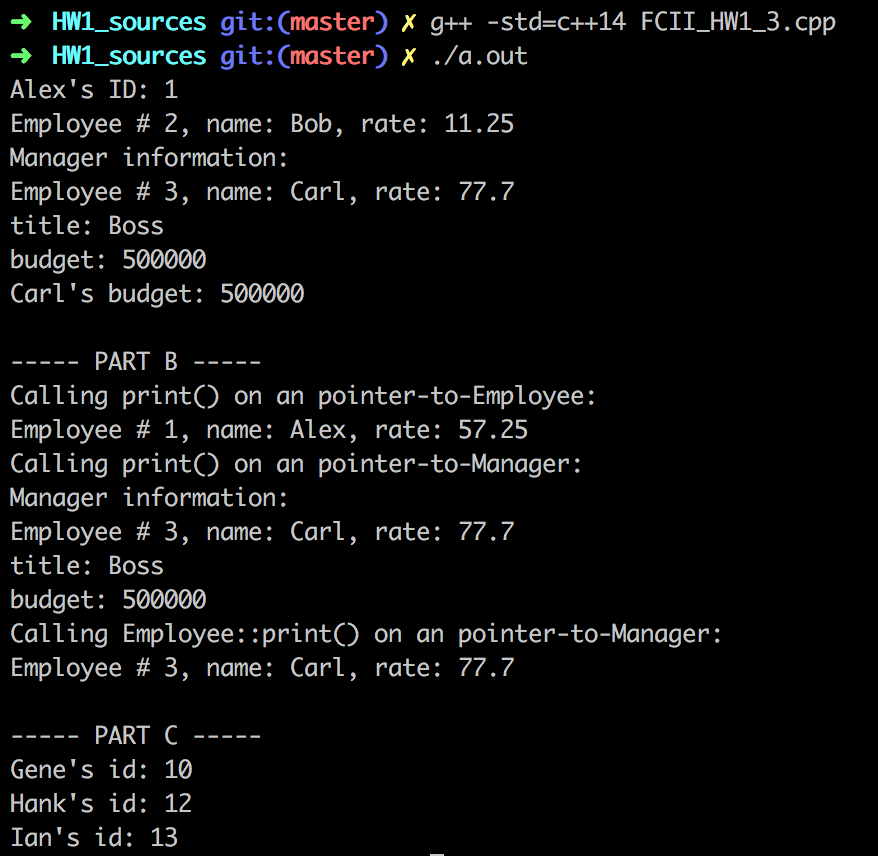
1. After setting the ‘print()’ function in the **Employee** class to virtual, we confirm what question 1(b) is asking. Also, we add a test case for calling **Employee**::print() from a pointer-to-**Manager** calling. The results are below.



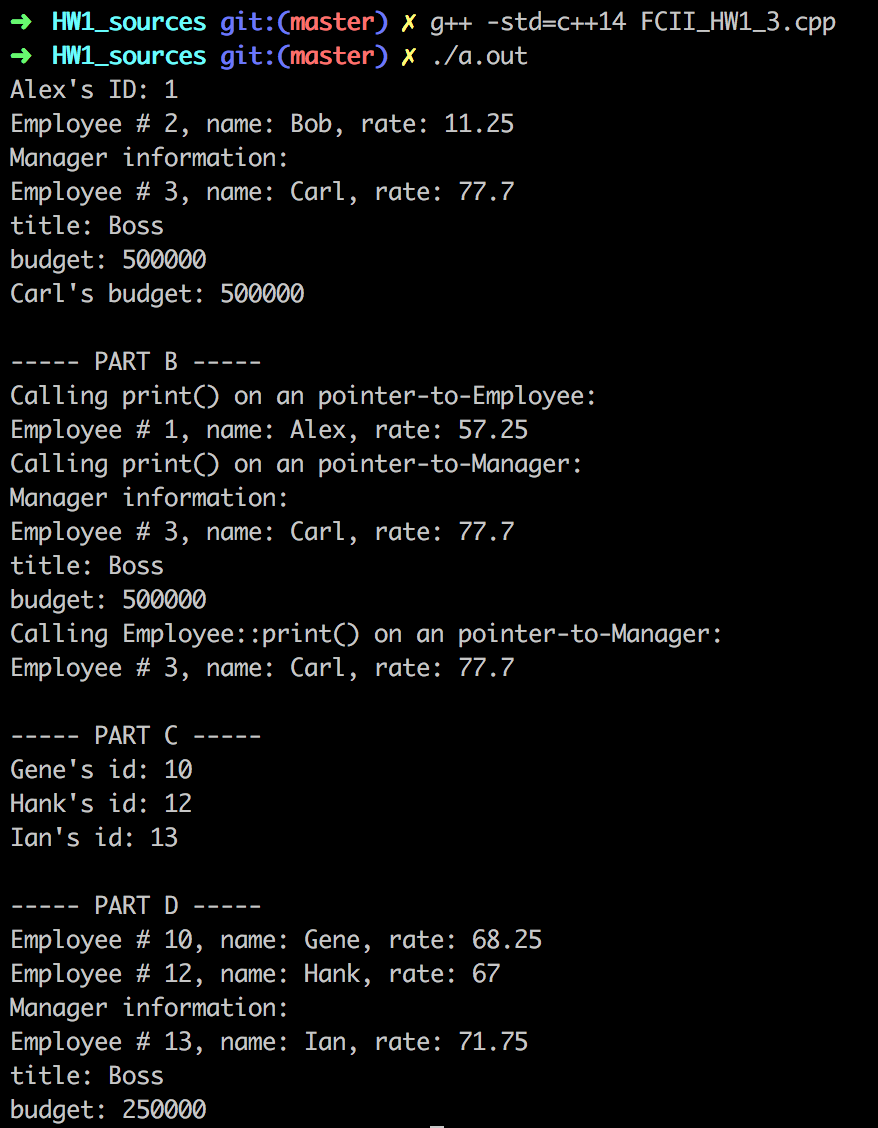
1. We changed the class access of the ‘const int id’ data member to private in the **Employee** class, and we implemented a ‘const int get\_id() const’ public function in the **Employee** as instructed. The following code snippet shows the result. (Note: Some other debugging / modifications had to be done than what was mentioned in the homework description file).



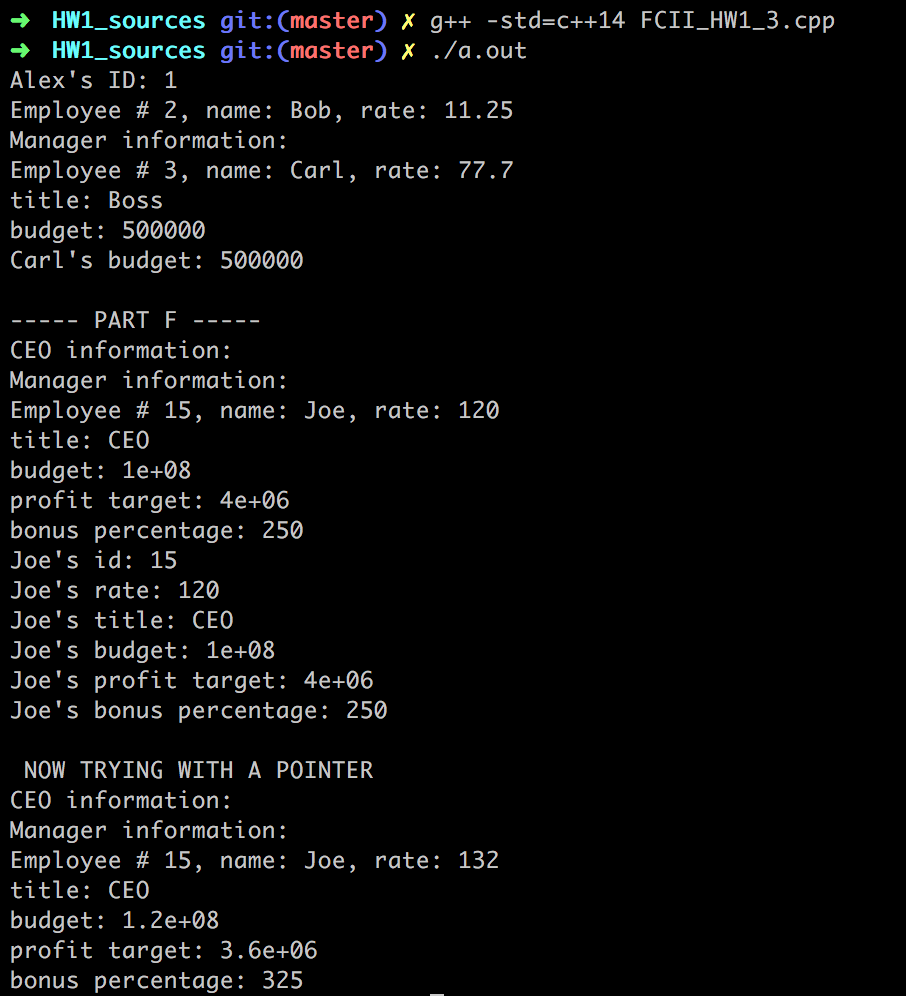
1. This certainly won’t compile. The reason is that ‘p’ is a pointer-to-**Employee**, and the object ‘hank’ is of type ‘const **Employee**’. Hence, we need to modify the pointer that was declared. When initialized to ‘&alex’, we changed it to ‘const Employee \*p(&alex);’ so that way it was able to work for ‘hank’. Below is the output of the code:



1. Changing the ‘print()’ function within the **Manager** class, as mentioned, will have no change since ‘virtual’ functions are percolated up to derived classes from the base class. Below is a code snippet confirming there is no change in behavior:

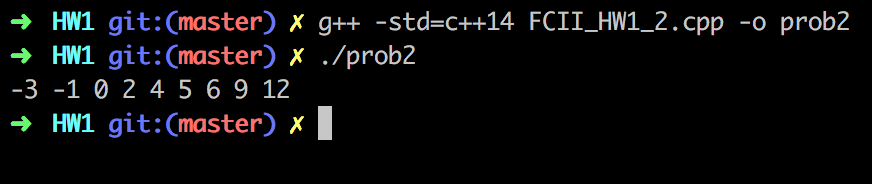


1. After implementing the **CEO** class and running the test code, here are the results:

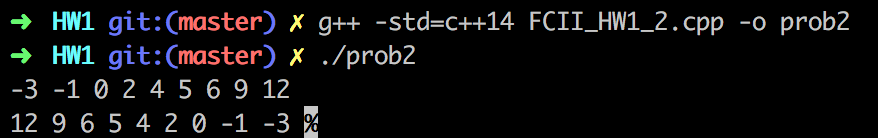


Problem 2

1. It is in ascending order:



1. This confirms it is in descending order:



1. Here’s using `typedef`. Honestly, I think `typedef` can be clear in some cases and confusing in others, so it depends. If our application were to constantly be using a pointer to some user defined class, typedef may be beneficial.



1. Even up odd down functionality:

