PIC 10B Section 1 - Homework # 4 (due Friday, April 26, by 6 pm)

You should upload each .cpp and .h file separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.
Be sure you upload files with the precise name requested of you and that it matches the names you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. Also be sure your code compiles and runs on Visual Studio 2017.

SIMPLE COMPANY

In this homework, you'll get to work with polymorphism and apply it to modelling employees in a company. In the end, you will submit 6 files:
Employee.h (defining an **Employee** class and declaring its constructors/member functions and a utility function);
HourlyWorker.h (declaring an **HourlyWorker** class and declaring its constructors/member functions);
SalariedWorker.h (declaring a **SalariedWorker** class and declaring its constructors/member functions);
Company.h (declaring a **Company** class and declaring its constructors/member functions);
Company.cpp (giving definitions of the previously declared functions, etc.) and
main.cpp with a simple main routine.

The idea is that a **company** has **employee**s who may be **paid hourly** or by a **salary**. The program is very simple: until a user chooses to quit, they can:

- **Create** a new employee by selecting "C";
- **Display** the company directory by selecting "D";
- **Set** hours worked by hourly employees by selecting "S";
- **Print** payroll information by selecting "P".

Information on each employee is stored in a text file that can be read from and written to in the case of updates.

In **Creating** an employee, a user is asked if they wish to:

- Create a **Salaried** employee by selecting "S" and then giving the name, email address, and salary; or
- Create an **Hourly** employee by selecting "H" and then giving the name, email address, and hourly rate.

In **Displaying**, all employees of the company are printed to the console with their name, employee id, and email address.

In **Setting** hours worked, for each hourly worker in the company, the user is prompted for how many hours they worked.

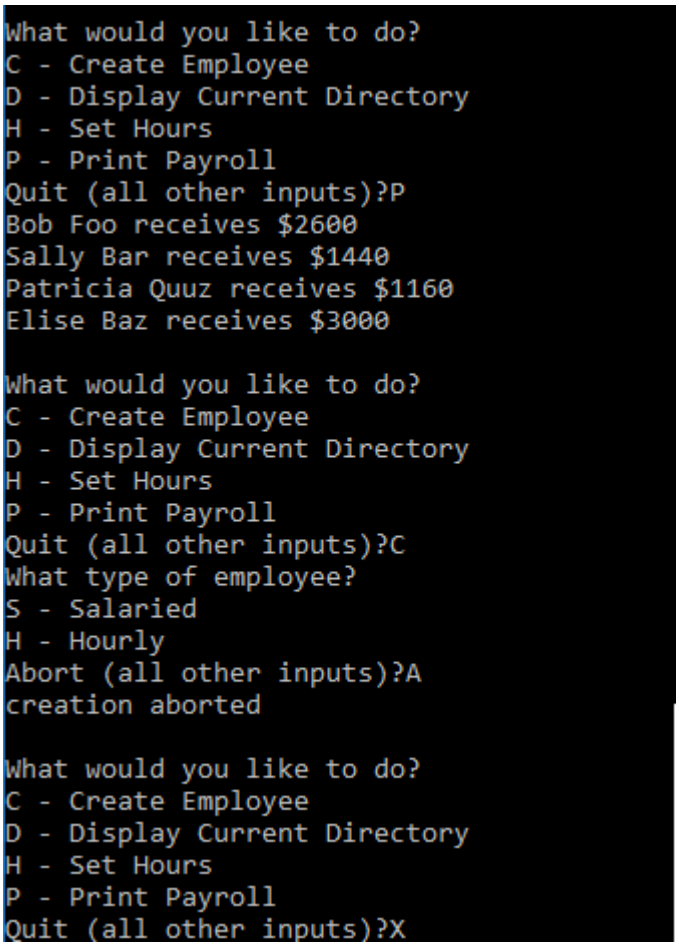
In **Printing payroll** information, each employee is printed to the console with how much money should be on their pay cheque.

The program below illustrates how the program runs with more details to follow. **Given the included files "employee0.txt" and "employee1.txt"** the proper behaviour is illustrated in the screen shots below. The files **"employee2.txt" and "employee3.txt" are generated** through the running of the program below.

```
What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?D
There are 2 employees:
Bob Foo 0       bob@foo.com
Sally Bar       1       sally@bar.com

What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?P
Bob Foo receives $2600
Sally Bar receives $0
```

```
What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?C
What type of employee?
S - Salaried
H - Hourly
Abort (all other inputs)?H
Enter name: Patricia Quuz
Enter email: patricia@quuz.com
Enter hourly rate: $29

What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?C
What type of employee?
S - Salaried
H - Hourly
Abort (all other inputs)?S
Enter name: Elise Baz
Enter email: elise@baz.com
Enter salary: $3000
```

```
What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?D
There are 4 employees:
Bob Foo 0       bob@foo.com
Sally Bar       1       sally@bar.com
Patricia Quuz   2       patricia@quuz.com
Elise Baz       3       elise@baz.com

What would you like to do?
C - Create Employee
D - Display Current Directory
H - Set Hours
P - Print Payroll
Quit (all other inputs)?H
How many hours did Sally Bar work? 36
How many hours did Patricia Quuz work? 40
```



The file "employee1.txt" *will be modified through the running of the program.*

There are some **absolute requirements listed below** but outside of these requirements, you have freedom over other helper/member functions. To adhere to best coding practices, you will need to add at least a few more functions and carefully consider your use of virtual functions, protected functions, etc.

The requirements are imposed because the purpose of this assignment is to learn about inheritance, virtual functions, casting, static members, etc., so there are very precise requirements given.

Requirements:

The **Employee** class should:

- store a member variable **name** for their name;
- store a member variable **email** for their email address;
- store a member variable **id** for their id (an integer);
- store a static integer member variable **next_id** that stores the value of the next **id** to be created*
- have a **constructor** that accepts a name and email address, and creates an **Employee** from that;
- have a **constructor** that accepts an input file stream and reads a name, email, and id from the stream#;
- have a **print** function that prints directory information of the form:
  [NAME] [TAB] [ID] [TAB] [EMAIL]
  to the console;
- have a **write_data** function (that could be overridden) that writes the name, email, and id of the **Employee** to a file, each entry separated by a tab; and
- have pure virtual **print_pay** and **save** functions that mandate functionality for printing payment information and being able to save employee data.

*: id's are created started from the id 0, going up by 1 every time. Every employee has their own id but the pattern is predictable.

#: the constructor accepting an input file stream will be reading from a file that already stores an id in it, whereas the other constructor is creating a brand new employee so the id must be generated.

File naming: all files storing employee data are named in the precise format **employee[ID].txt**, for example "employee3.txt", "employee8.txt", etc.

The **HourlyWorker** class should inherit from **Employee** and:

- have a member variable **hours** to track the integer number of hours worked;
- have a member variable **rate** to track the hourly pay rate;
- have a **constructor** accepting a name, email, and hourly rate, setting the hours worked to 0 initially;
- have a **constructor** accepting an input file stream to read in the name, email, id, hours worked, and hourly rate;
- have a **set_hours** function that sets the hours worked;
- implement the **print_pay** function to print:
  [NAME] receives $ [AMOUNT]
  to the console; and
- implement the **save** function that saves the employee information to their file.

The **SalariedWorker** class should inherit from **Employee** and:

- have a member variable **salary** for their salary;
- have a **constructor** accepting a name, email, and salary;
- have a **constructor** accepting an input file stream to read in the name, email, id, and salary;
- implement the **print_pay** function to print:
  [NAME] receives $ [AMOUNT]
  to the console; and
- implement the **save** function that saves the employee information to their file.

Format of files: the formats of the files are illustrated below. Use
salaried [*TAB*] [*NAME*] [*TAB*] [*EMAIL*] [*TAB*] [*ID*] [*TAB*] [*SALARY*]
for a **salaried worker** and
hourly [*TAB*] [*NAME*] [*TAB*] [*EMAIL*] [*TAB*] [*ID*] [*TAB*] [*HOURS*] [*TAB*] [*RATE*]
for an **hourly worker**.
Note how the first word is either "salaried" or "hourly": this is necessary so that the **Company** class described below can determine whether an employee is salaried or paid hourly.

The **Company** class should:

- store a member **employees** of type **std::vector<std:shared_ptr<Employee>>** and *store no other member variables*;
- have a **default constructor** that reads through all the files and creates **Employee**s of suitable type for each file, storing them as pointers;
- have a **print_directory** function that displays how many employees are in the company and then runs through the vector and calls **print** on all entries;
- have a **do_hours** function that runs through the vector and **through using a dynamic_pointer_cast**, for the **HourlyWorkers**, prompts the user
  How many hours did [NAME] work?
  and following the user input, it should call **set_hours** on the employee to set their hours and
  call **save** on the employee to save the changes;
- have a **print_payroll** function that runs through the vector and calls the **print_pay** function on all elements;
- have a **create_salaried** function that prompts a user for a name, email address, and salary for a **SalariedWorker** and instantiates that class, saves it, and adds it to **employees**;
- have a **create_hourly** function that prompts a user for a name, email address, and hourly pay rate for an **HourlyWorker** and instantiates that class, saves it, and adds it to **employees**;
- have a **create_employee** function that asks a user what type of employee they want to create with "S" for salaried, "H" for hourly worker; and other commands aborting the creation;

You will write two free functions, too:

- **find_next_id** that serves as a helper function for setting the static member **Employee::next_id**: it should read through all files "employee0.txt", "employee1.txt", etc., until it fails to open a file at which point the integer index it failed at should be returned and be the next id;
- **run_events** that creates a **company** object and repeatedly prompts a user if they would like to create/display/set/print as described previously, terminating when they do not enter a "C", "D", "H", or "P".

Hint: the **std::getline** function is overloaded and can be given three arguments (not just 2). When called with
std::getline(in, str, c);
the stream **in** will read into the string **str** until it encounters the char **c**, then remove that char. The normal **std::getline(in, str)** effectively has **c** set to **'\n'**.

Some initial guidance to get you on your way...

1. Write the **Employee** class first and don't worry about the static member variable.
2. To start, do not include any pure virtual functions in **Employee** and make sure you can construct it in the two prescribed manners (if you are ignoring the static member variable then just generate a **std::rand()** id for now).
3. It might help to make all variables/functions public initially and only later make things more object-oriented and encapsulated.
4. After **Employee** is written, write the derived classes.
5. Now start to think about the static member variable and the **find_next_id**.
6. Test each function and make sure they all work before proceeding to **Company**.