1 Objectives

1. Gain familiarity with c++ class inheritance
2. Begin investigating techniques using polymorphism

2 Tasks

Begin by implementing a class called Employee that satisfies the following prototype:

```cpp
class Employee {

protected:
    double salary;
    std::string name;

public:
    // non-default constructor
    Employee(std::string, double);

    // returns the amount the employee should be paid
    double pay();

    // prints to stdout the employee information
    void print() const;
};
```

1. Define a new class called Hourly that inherits from Employee, and has the following behavior:

   (a) It contains a variable for the number of hours worked, called hoursWorked.
   (b) It has a constructor which takes the same arguments as Employee, and initializes the hours worked to 0.
   (c) Has a method called addHours that increases the hours worked by a specified amount.
   (d) Overrides the pay method, returning the payRate times the hoursWorked, to get the total paycheck for that employee. Finally, once this gets called, the hours worked should be reset to 0, representing that the person has been paid.
   (e) Overrides the print method, calling the parent’s print method first. Then it should add a line to print the total hours worked so far.
2. Create another derived class from `Employee`, called `Executive`, with the following extra modifications:

   (a) It has a private variable to hold the executive’s bonus pay.
   (b) It has a method to set the bonus pay for the employee.
   (c) It has an analogous constructor which calls `Employee::Employee()`.
   (d) It overrides the `pay` method and:
      i. Computes the pay by using the parent `pay` method
      ii. Adds the executive bonus to that pay.
      iii. Resets the bonus to 0, since they have no been paid!
      iv. Returns the total pay.
   (e) It overrides the `print` method, but calls the parent `print` first, then adding a line for the bonus pay.

3. Test your code by creating some instances of each class, demonstrating that each method does what it is supposed to.

4. Test your code by creating several instances of each class, stored in an array of type `Employee*` (possible through Polymorphism!).

5. You may notice that there are some inconsistencies with how the `Employee` class is printed! Go back and fix this by using the `override` and `virtual` keywords where appropriate.

3 Submission

Upload your project files to the course canvas system in a single zipped folder: To zip on Linux:

1. To zip a single folder into a single archive called “myZip.zip”:
   ```
   zip -r myZip.zip folderName
   ```

2. To zip multiple files into a zip file called “myZip.zip”:
   ```
   zip myZip.zip file1.cpp file2.h file3 file4.cpp
   ```

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.