In this project we created a contractor class with 3 member variables, a subcontractor child class that inherits from the contractor parent class and adds 2 additional variables. Then, in the main program we used a user-terminated while loop to collect these 5 items and add them to a subcontractor object which then printed the computed value for pay based on hourly wage times total hours since start date. When the user entered a night shift into the program it would then add a 3% increase to their total pay. The user is then asked to end the program or run it again and collect a new set of data points.

We began the project by laying the class foundations in contractor.cs and subcontractor.cs. There was some time spent frustrated by the next step of adding get and set methods for each variable; however, after surmounting this, it was relatively simple to create a main program.cs file that called the subcontractor class to hold the user inputs. The last step was to put this inside of a while loop that allows the user to enter the information for an indefinite number of subcontractor objects and indicate when they're finished. At each stumbling block we relied most heavily on the Microsoft c# documentation, which is very thorough and user-friendly.

With more time, I would replace the current object creation with a list of objects and a counter, so that each object can be stored continuously and new objects are appended to that list. However, within this timeframe I didn't figure out a viable way to create the list at the top of the main program and then append new objects to it. In an even more advanced vision of this program, it might be optimal to find a way to use a database or some other external resource to let us capture the user's name input for the name of the object itself. This was one dead-end I explored that ended up failing because C#'s just-in-time compiling doesn't allow accessing the inputs to be reinserted into the running program.

Additionally there is room for improvement on the user interface front. With more time we could include better input validation. The program will break if the user enters a string instead of a number for the wage the program will break and there is no way of handling negative inputs. There are also potential issues with using DateTime.now instead of an end date. If the user is even a day late the employee would be paid the wrong amount. Another thing I would have liked to add would be a method for calculating overtime pay in the case of a contractor who's worked more than 40 hours.