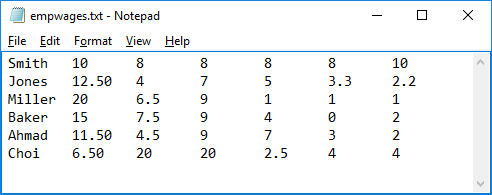
**CIS 221 PA5 - Ch6 & Ch13**

**Employee Payroll Summary**

For this assignment, you will write a program that reads employee work data from a text file, and then calculates payroll information based on this file. The text file contains one line of text for each employee. Each line consists of the following data (delimited by tabs):



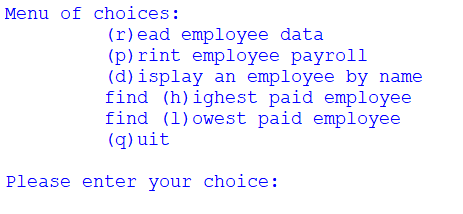
1. employee’s name
2. employee’s hourly wage rate
3. hours worked Monday
4. hours worked Tuesday
5. hours worked Wednesday
6. hours worked Thursday
7. hours worked Friday

The employee Smith makes $10 per hour, and worked 8 hours on Monday, 8 on Tuesday, 8 on Wednesday, 8 on Thursday, and 10 on Friday. You can see similar data for the other employees.

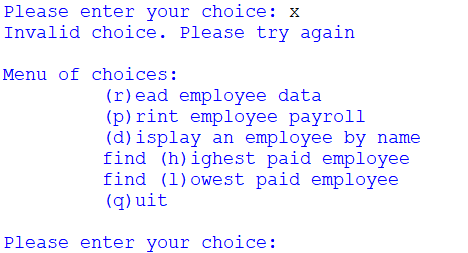
**Expected Behavior of your Program**

The following screen shots illustrate what your program should be able to do:

The program will repeatedly present a menu of options, as shown below.

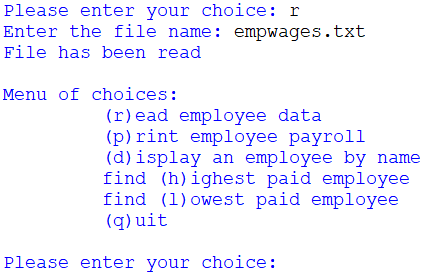


These are the only allowable options. If the user types anything other than r, p, d, h, l, or q, you should print an error message, and again present the menu, like this:

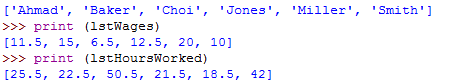


Assume the user selects to read a file. Your program will ask the user for the file name (it must be in the same folder as your .py file), attempt to read this file, and use it to populate three lists: a list containing each employee’s name, a list of each employee’s hourly wage, and a list containing each employee’s total hours worked for the week.

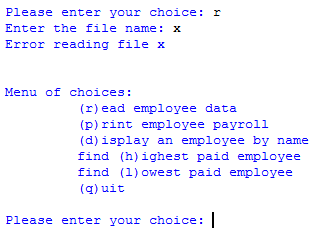
When this is completed, these lists should be sorted in order based on the employee names (more about this later). Assuming the file was successfully read, and data successfully input, you should present a confirmation message, like this, **followed again by the menu**.



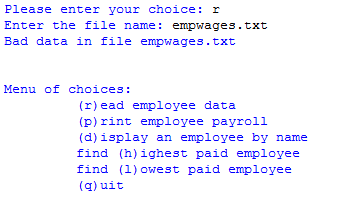
If the above file was read successfully into the three lists, they will look like this in memory:



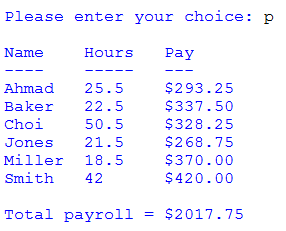
Note: it’s possible that the user could enter an invalid file name. In this case an error message should appear, like this:

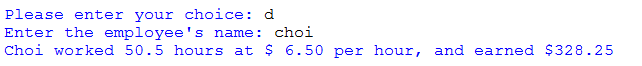


Even if the file does exist, you may still have problems processing the data. For example, if the hours worked or the hourly wages are missing or non-numeric, this is an error. In this case, you should also print out an error message, like in this display:



Once the file is successfully read and processed, the user can select the other menu options. The following screen shots show the payroll report, a display of a selected employee (based on name), the highest paid employee, and the lowest paid employee:

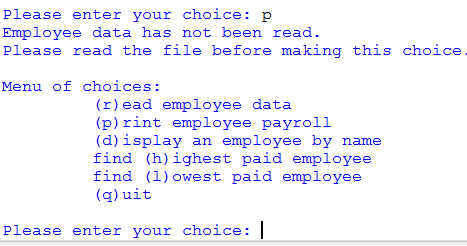








Note: none of these options should be allowed if no data has been loaded into the lists. If the lists are empty, and the user enters one of these options, you should display an error message and again present the menu, as shown below:

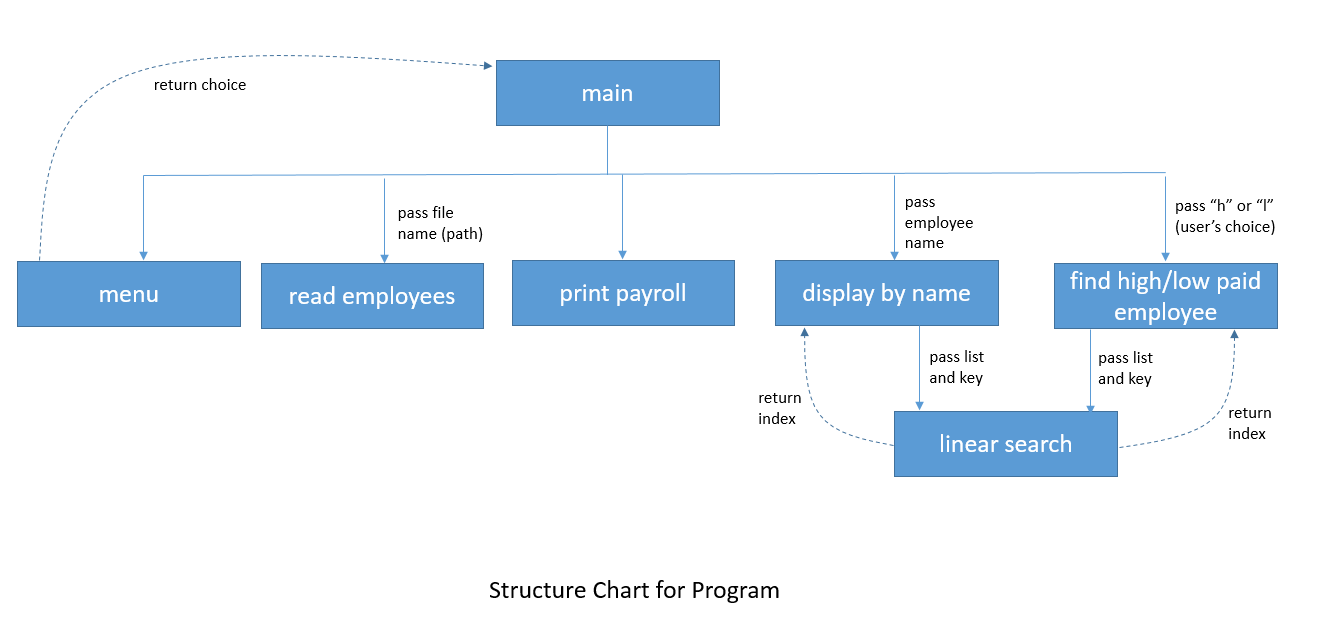


Finally, when the user wants to quit, print a goodbye message and terminate the program:



**Program Design Specifications**

The structure chart showing the top-down stepwise refinement for your program is shown below.



You will define each of the above functions. The main function will call each of the functions underneath it, and the display & find functions will in turn call linear search. The behaviors of each of these functions are described below:

1. **main** will loop continuously, calling the menu function and using the returned choice to determine which other function to call, until the user decides to quit.
2. **menu** will loop continuously, presenting the menu and inputting the user’s response, until the user enters a valid choice. Every time the user enters an incorrect choice, it will display an error message as shown above. Once the user enters a valid choice, the menu function will return that value.
3. **read employees** will take in the file name as an argument. It will open the desired file, and use exception handling to deal with non-existent file or file with bad data. It will read all lines into a list, and sort that list alphabetically. Then it will use this list to populate the employee names list, the hourly wage list, and the hours worked list. Note that the values in the hours worked list will be a sum of all the hours for the week (M, T, W, TH, F).
4. **print payroll** will display the payroll report as shown above. This includes a header, the list of each employee with hours worked for the week and total pay for the week.
5. **display by name** will receive an argument from main, use the linear search to find the index of that employee, and use this to display the name, hours worked, hourly wage, and total pay for the week, as shown above. The function should be flexible enough to recognize the employee name even if it is not capitalized. If the employee is not found, a message should be printed to that effect.
6. **Find high/low paid employee** will receive an argument from main (either “h” or “l”, i.e. the user’s choice). It will use this to find the maximum (or minimum) total pay for the week, and then do a linear search to find the employee with that total pay. Then it will display the employee’s name and earnings for that employee, as shown above. Note: we are looking for maximum or minimum total earnings, not max/min hourly wage.

Additional requirements:

1. The **menu** function can input data from the user (the user choice). This is the only function other than **main** that can input from user. All other functions must receive user inputs as arguments passed in. No function other than menu and main can call Python’s input() function.
2. The three lists (employee names, employee hourly wage rates, and employee hours worked) will all be global variables. All other variables in your program must be local variables of functions.
3. Your program must be fully documented with appropriate comments. There should be a comment for each function describing what it does. In addition, you should briefly describe different sections of code within each function.
4. Do not hard code the file name ***empwages.txt***; your program should work with any file that is properly formatted regardless of the number of employees. For example, if I name a file stuff.txt, place it in the same folder as your program and it is correctly formatted, then your program should open that file successfully.

**Hint**: tackle these functions one at a time. Make sure menu works before trying read. Make sure read works before trying to print payroll, etc.

**Deliverables:**

* You will turn in your .py file and your planning doc (pseudocode or flowchart). Zip these files into a .zip file and *upload that single .zip file to Canvas*.
* You **don’t** have to upload the empwages.txt file.