Hello and thank you for the opportunity.

If this were a real project, the first thing I would do after being assigned this coding task would be to ask/search for prior work. Most project work is comprised of feature adds and bug fixes, so generally it’s more refactoring than original work.

For the purposes of this assessment, I’m going to presume there is no prior work to leverage (thus no test suite either).

Making sure I understand as much as possible from my manager about what’s needs to be done and the timeframe and resources at my disposal, I’ll sit down and write out a flow of what needs to be done.

I created two test files. The first is from the assignment. The second is full of invalid address errors, bad ages. I also threw in a random address from Hong Kong, in English, then followed by Chinese lettering (via Google Translate).

For this assignment, I chose C# to code with (since I am most familiar with this language) and created a new console app in Visual Studio 2022. Here is my original flow:  
  
 // Output to the console, a brief introduction about this program

// Ask the user for the Address List Text File; Allow for the user to exit the program.

// Check if a valid txt file and path

// Note - For now presume that only a text file of comma separated strings is the only

// format for data input. Other forms are spreadsheets, XML, Database connection, or other custom formats.

// If no valid Address List Text File is found via the user input, ask again.

// Open the valid Address List Text File; display an error if for some reason the file cannot be opened and exit the program

// Create a <list> for invalid addresses

// Create a <list> for validated addresses

// While the Address List Text File is open, parse each line starting from the top. Each line parsed will be validated for correctness:

//

// 1. Each line in the text file is a 'complete address' that MUST contain 6 sting elements (these string elements can be of any language)

// 2. The last string element must be convertible to a numeric number >= to 0

// Note: Determine with business if there should be an upper age limit check

// Anything that resolves > 134 is almost certainly an error (The record holder is Jeanne Calment of France, almost age 123)

// 3. Add the validated parsed line to the <list> of validated addresses

// 4. Add the invalid parsed line to the <list> of invalid addresses

// Close the Address List Text File when EOF is reached.

// IF the <list> for invalid addresses is NOT empty; create a errors text file in the same path that Address List Text File resides

// Note - depending on the downstream consuming systems, this might need to be sent to a different connection as an text, xml,

// spreadsheet, db connection, etc. For now presume I will presume a text file at this location is acceptable.

// Process the <list> of validated addresses

// Rules:

// 1. Group by household (Use the Street Address + State. Ignore Case).

// 2. Get the Count of the total individuals grouped at this address (regardless of age).

// 3. In each Group, Sort by Last Name then first name

// 4. Display the list, except for individuals under 19:

// 5. Street Address and total occupants

// 6. Indent and display elements 2, 3, 6 that have the same addresses.

//

Now that I have my comments/outline, time to start coding. First, I will use the internet to see if there are snippets that I can leverage, especially for a data structure to put together that will be easy to work with.

I coded the console app to load a text file containing the mailing list (user choice) from any file share. On my local machine, I have a folder on my D drive containing test files.

If the file is locked or has other issues, or the user makes a typo, the user is notified that the string they entered is not valid and to try again.

The user always has the option to exit the app by entering ‘Q’. For a long file path name, one can use notepad to store the file path and copy/paste it into the console app prompt or simply drag and drop the file into the open console.

Now that I could reliably load a text file, I need to process this information. Looking at the list, I needed to parse the pseudo-csv text file so that I could manipulate the information. It was not straightforward:

1. The case was mixed, so I would need to account for that.
2. I noticed that an entry had a comma that was NOT a delimiter between the fields.
3. The age was a string, not a value. Also, the human lifespan is less than 135 years old (what if there was a typo?)
4. I worried about mistakes in a record of having too many (extraneous item such as a country) or missing a field.
5. I also needed to see if a foreign language would work (I chose Chinese... from the internet I found the most common name and a random Hong Kong address)

I made two list objects; one to hold ‘validated’ text and one to hold ‘invalid’ text.

I chose to use the *StreamReader* to read line by line from the text file. As each line was loaded, it was tested to meet the above criteria before added to the validated List. I also made sure that street, city, state was converted to the uppercase form (Note that the best practice per Microsoft for string usage in the .NET framework is to use uppercase. So that certain extended characters and foreign languages that has a single case, such as Chinese, will correctly compare). An Invalid text line was put into the invalid List rather than just ignored; this list could be reviewed by humans to correct and resubmit later.

I needed to go through the valid List and group by household (composition of street, city and state), sort on the last name in each household, and only display those household members over age 18.

Even with the preprocessing, I have a non-normalized dataset that I would have to make a complex comparison to group, sort and filter to obtain the desired output. What I really needed to make this part easier would be a SQL database with First Order normalized tables… Then it occurred to me, that I should be using a DataTable (and a string of any erroneous lines) instead of building a List objects.

I would keep the text file selector and preprocessing logic, but I would refactor my incomplete grouping/sorting/filtering (not in a working state) instead.

The steps I would now follow:

1. Create a DataTable object with the correct fields (First name, Last name, Street, City, State, Age)
2. Validate the text from the user supplied text file. Invalid lines were accumulated in a string, to display the invalidated lines of text with a brief comment of about the issue.
3. Sort the DataTable (by Last than First name) after validating the pseudo comma delimitative lines from the user specified text file. This was done to simplify the query.
4. Query the DataTable by group (Using a composite of Street, City, then State)
5. Display the output as stated in the homework assignment.

Step 1: Implementing this did not take long. I consider making the age a numeric but decided leaving it a string was acceptable.

Step 2: For the most part already completed before I decided to refactor. There were some tweaks as I formatted the output line. I also created a ‘bad’ text file to test the functionality.

I debated briefly whether the occupant’s age should be kept as a string or converted to a numeric. I decided to keep it as a string.

I also quickly discovered that the case of the addresses was mixed, and comparisons did not occur as expected. I made the street, city, and state upper case. I also made sure to trim out any excess whitespace and remove punctuation marks from these fields.

I spent time researching postal code standards since there could be use cases to test. Since the homework data were strictly from USA, I did not bother to look deeply into other country standards.

Step 3: I toyed with an idea of creating two data tables that had 1st normal form with each other (One for household addresses, the other for household names). How to ‘automate’ the split and creating the primary/foreign keys seemed more troublesome than it was worth. So, I just worked with a single table. For huge datasets to process, this might be worthwhile, especially if it were decided to make the output into a DB.

Step 4: This is where I got bogged down and spent too many hours. What was frustrating is that in a SQL DB, I could craft a query. However, C# was not so easy.

First, I had to figure out how to group by using a composite key. I found several examples online and was able to get this to work. But I still needed to list the occupants under each group. After several hours of researching and trying out various suggested methods, I got this to work.

Next, I needed to get all the occupants in a household and then sort them by last then first name. I just could not get the logic to both group AND sort on these fields as part (I could do one or the other). I came upon the solution of using a DataView to do the sorting of the DataTable first, then perform the Groupby query. That worked.

In order to get the occupant, count correct (all people, not just the adults), I dropped the Where clause of selecting > 18 years old. To obtain the occupant count, I used a counter in the Foreach loop that lists out the occupants. Using a count in the query just calculated out the number of households, not the number of occupants per household.

Step 5: Getting the output out correctly took a lot of tweaking. One criterion I did not meet was to list at the household level the occupant count. Because I got the count after I completed listing the occupants, I ended up putting this at the bottom of the group section.

In order to meet this, instead of listing out the occupants and household, I could just build a string instead, that way I would be able to list out the household and occupant level on the same line. Since time is short, I will leave this as a TBD.

I enjoyed working on this challenge. The solution and test files have been zipped together. Feel free to contact me if there are further questions relating to this exercise.  
  
Best regards,

Andrew Goldstein

(m) 425-829-3623

Addendum 10/17/2022:  
  
It bothered me that my code did not output the desired arrangement of having the household occupant count on the same header line. So, I spent a little time correcting this flaw and correcting a few misspellings I found.

Test Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Type | Input | Expected Results |
| Happy Path | End To End | Valid User Selected file with no issues, e.g., DITAddressList.txt | Process the text file without issues generating Tabulated results, no invalid data list |
| End To End User Gives path to non-text file | End To End | Any non-text file | User selected file rejected; asked again for valid file |
| Users Gives path to text file | End To End | TESTBadData.txt | Process the text file: Tabulated results for all valid lines, invalid lines shown to the user. |
| Process a text file that has whitespace between entries | Functional | TESTBadData.txt | Line marked as invalid |
| Process a line that has too many fields | Functional | TESTBadData.txt | Line marked as invalid; comment about invalid number of fields |
| Process a line that has too few fields | Functional | TESTBadData.txt | Line marked as invalid; comment about invalid number of fields |
| Only valid delimiters “,” separate fields | Functional | TESTBadData.txt | Line marked as invalid; comment about invalid number of fields |
| Delimiter has excessive white space; “ ,” or “, “ or “ , “ | Functional | TESTBadData.txt | Line considered Valid and is processed |
| Foreign Language (non-Latin character set) | Globalization | TESTBadData.txt (Chinese) | Line considered Valid and is processed |
| Any address field contains . or , | Functional | TESTBadData.txt | Punctuation stripped before processing and the line is considered valid |
| Mixed case Address fileds | Functional | DITAddressList.txt | All address fields are converted to UPPER case and the line is considered valid and grouped properly |
| Spelling Errors | Fit and Finish | Any text file | All console messages are of correct spelling (US standard English) |
| Formatting | Fit and Finish | Any text file | All console block messages do not extraneous excess spaces and line up.  Tabulated data is spaced so that reading is easy – Occupants listed are indented. |
| Program runs on supported windows OS (see https://learn.microsoft.com/en-us/lifecycle/products/windows-10-home-and-pro) | App compatibility | DITAddressList.txt | Program runs without issues |
| Test for memory leaks | Functional | TBD | No memory leaks when running |
| Injection | Security | TBD | Penetration attempts fail |