Assignment 3
Deadline: 11:59PM on Apr 5

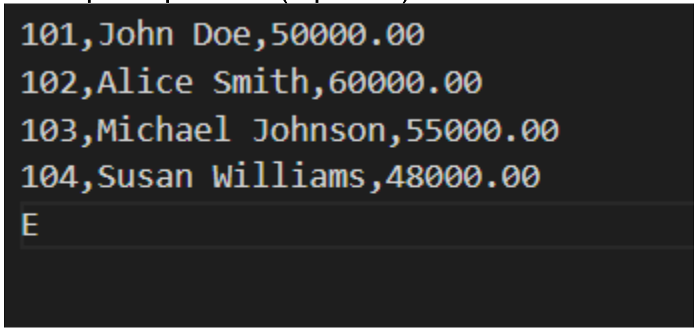
Requirements

You are tasked to implement a program in C to sort employee records alphabetically by name. Here each employee record consists:

- Employee ID (integer)
- Employee name (string), i.e,FirstName, LastName
- Employee salary (float)

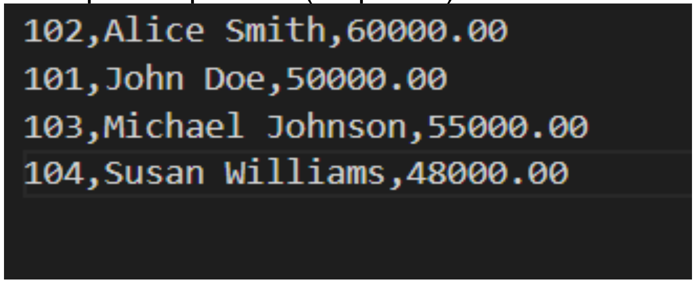
The Employee ID is a unique number for each employee. Your task is to create a program that accepts input from a text file. The input file should contain employee records in the following format:
Employee ID, Employee Name, Employee Salary (one record per line)
Each record field should be separated by commas.
The input file should end with an 'E' designator to mark the end of the file.
Your task is to output the sorted employee records to a text file.
The input and output file names are specified through command line arguments. Your code should read command line arguments for input and output file names e.g
./a.out input.txt output.txt

Example Input File (input.txt):



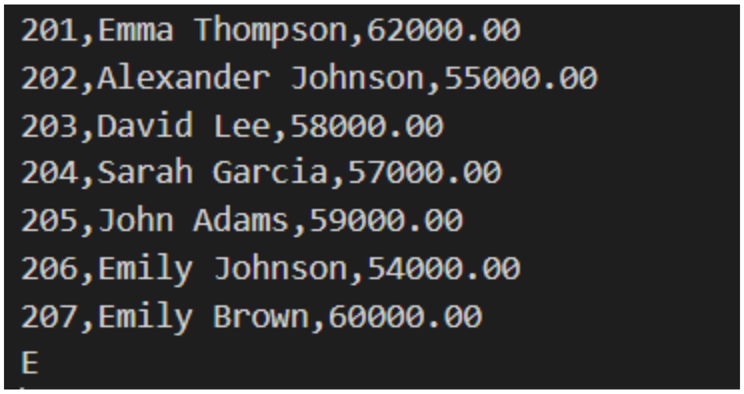
```
101,John Doe,50000.00
102,Alice Smith,60000.00
103,Michael Johnson,55000.00
104,Susan Williams,48000.00
E
```

Example Output File (output.txt):



```
102,Alice Smith,60000.00
101,John Doe,50000.00
103,Michael Johnson,55000.00
104,Susan Williams,48000.00
```

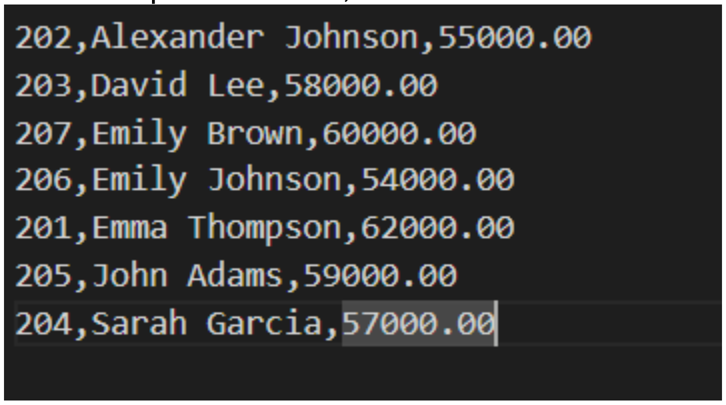
Here is another example.
input2.txt



```
201,Emma Thompson,62000.00
202,Alexander Johnson,55000.00
203,David Lee,58000.00
204,Sarah Garcia,57000.00
205,John Adams,59000.00
206,Emily Johnson,54000.00
207,Emily Brown,60000.00
E
```

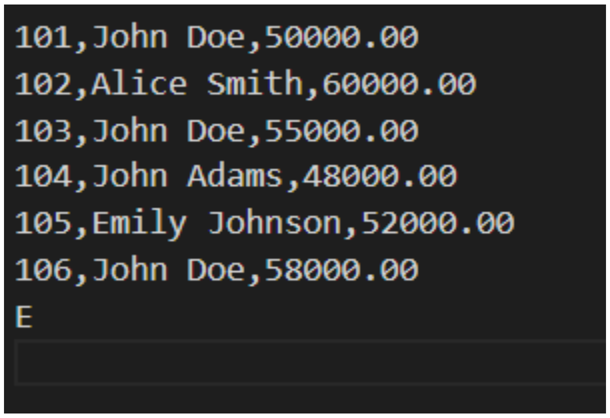
In the given example, the sorting algorithm must handle cases where multiple employees share the same first name, such as "Emily". To ensure correct sorting, the algorithm considers both the first name and, if necessary, the starting characters of the last name. This means that if two employees have the same first name, their records are ordered based on their last names. In the case of Emily Johnson and Emily Brown, their records are sorted alphabetically by their last names after being sorted by their common first name "Emily". Here “Brown” and “Johnson” are the last names so alphabetically ordering them “B” comes before “J” so “Emily Brown” is placed first. This approach ensures that the resulting output file maintains alphabetical order based on both first and last names, accurately reflecting the original data while resolving any potential conflicts arising from employees sharing the same first name.

So the output should be,



```
202,Alexander Johnson,55000.00
203,David Lee,58000.00
207,Emily Brown,60000.00
206,Emily Johnson,54000.00
201,Emma Thompson,62000.00
205,John Adams,59000.00
204,Sarah Garcia,57000.00
```

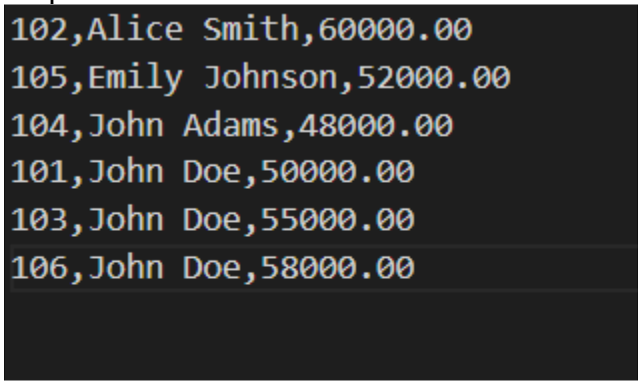
Here is another example,
input3.txt



```
101,John Doe,50000.00
102,Alice Smith,60000.00
103,John Doe,55000.00
104,John Adams,48000.00
105,Emily Johnson,52000.00
106,John Doe,58000.00
E
```

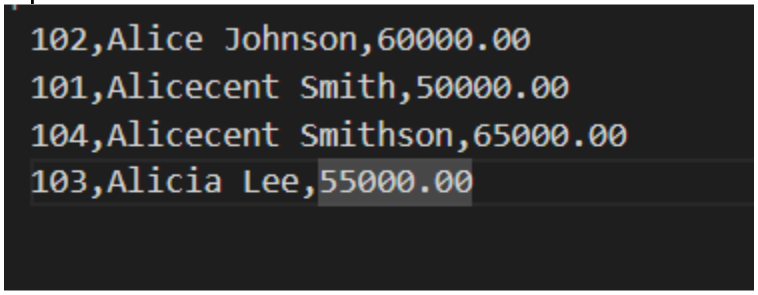
In the given example, the sorting algorithm must handle cases where multiple employees share the same name "John Doe". To sort these records, the algorithm first arranges them alphabetically by the employee's full name. Within this alphabetical sorting, individuals with the same name are further sorted by their employee ID in ascending order. This ensures that even if employees share the same name, their records are ordered uniquely based on their ID. The resulting output file maintains the alphabetical order by name while also ensuring that individuals with the same name are sorted by their ID, providing a clear and organized representation of the employee records.

here is the output ,
output3.txt



```
102,Alice Smith,60000.00
105,Emily Johnson,52000.00
104,John Adams,48000.00
101,John Doe,50000.00
103,John Doe,55000.00
106,John Doe,58000.00
```

Here is a last example,
input4.txt

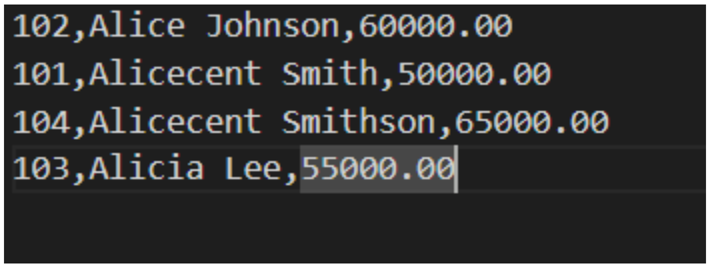


```
102,Alice Johnson,60000.00
101,Alicecent Smith,50000.00
104,Alicecent Smithson,65000.00
103,Alicia Lee,55000.00
```

In the provided example, when comparing "Alicecent" and "Alice," the sorting algorithm examines the entire names character by character. Since "Alice" is a substring of "Alicecent" and "Alicecent" is longer, the shorter name "Alice" is considered to come before "Alicecent" in the sorted order. This comparison ensures that even when two names share a common substring, the shorter name is prioritized, maintaining the desired alphabetical order based on the full names of the employees.

In case of “Alicecent Smith” and “Alicent Smithson” since they share same first name and the last name in one is a substring of the other we should keep the short one first when comparing "Smith" and "Smithson," .Since "Smith" is a substring of "Smithson," and you should prioritizes shorter names when they share a common prefix, "Smith" is considered to come before "Smithson" in the sorted order. This comparison ensures that even when two names share a common substring, the shorter name is prioritized, maintaining the desired alphabetical order based on the full names of the employees.

Here is the output,
output4.txt



```
102,Alice Johnson,60000.00
101,Alicecent Smith,50000.00
104,Alicecent Smithson,65000.00
103,Alicia Lee,55000.00
```

Handling Errors

When working on this, it's crucial to handle these errors:

- Commas between Fields: Ensure that there are commas separating each field in the input file. Check for missing commas, especially between the employee ID and name, and between the name and salary.

- Employee Name Format: Confirm that each employee's name is in the format "firstname lastname". If one name is present, both the first name and last name should be included.

- Valid Salary Values: Check for valid float values for employee salaries. Ensure that salaries are not negative.

- Unique Employee IDs: Verify that each employee ID is unique. Look out for duplicate IDs, as each employee should have a distinct identifier.

- Valid Employee IDs: Ensure that employee IDs are positive integers. Negative or non-numeric IDs are invalid and should be corrected.

- End Designator: Check for the presence of the 'E' designator at the end of the input file to mark its termination.

- Empty Records: Watch out for empty records or lines in the input file. These should be identified and handled appropriately to prevent errors during processing.

How to Compile and Run

- The Makefile for the assignment is provided.
- The Makefile is supposed to work with assignment3.c, input.txt, output.txt and ref.txt files so, make sure to save your files accordingly.
- Run the following command in vs code Terminal.
  ```
  make
  ```
  It should compile the code without any errors.
  ```
  make convert_input
  ```
  It should convert the input.txt file to unix encoding.
  ```
  make run
  ```
  It should run the compiled code.
- Run the following command to delete the out file.
  ```
  make clean
  ```
  It should delete the specified file.
  ```
  make convert_output
  ```
  It should convert the output.txt file to unix encoding.
- Run the following command to test your output with provided relevant reference output.
  ```
  make check
  ```
- You are not supposed to make any changes in the Makefile.
- Make sure to install dos2unix utility, if not already installed using the following command:
  ```
  sudo apt-get install dos2unix
  ```
  For Mac
  ```
  brew install dos2unix
  ```

Restrictions

- No new restrictions added in this lab
- In any corner cases, write “Error” to the output file and exit gracefully

Grading

Any grading failure due to not following instructions will result in 0. (1 point) All files are submitted correctly using the instructions below.

- (4 point) Generate a correct solution to the problem(s) in this lab. Three test inputs will be used.
- (4 point) Handle corner cases gracefully.

Submission Files

- You must submit only one file named: **assignment3.c**
- Submit it to learning hub before the deadline