Shilp Shah (RUID: 170009205)

Mathew Varghese (RUID 170009651)

Assignment 0 Documentation

Approach

- We thought about the design of the project first before starting the program. We were deciding between 3 different cases. Case 1 was using a linked list and doing merge sort on the linked list to sort the CSV file. Case 2 was figuring out a way to find the size of the file and making an array of appropriate size and doing merge sort on that array. Case 3, was making a linked list and putting each node into an array then doing merge sort. There was a difficult aspect to each case. We decided to try to implement case 3 because we felt it was hard to do merge sort on a linked list. However, we had a lot of trouble making an array of nodes we ended up implementing case 1, and doing merge sort on a linked list.

Starting Out

- The first step of starting the project was thoroughly reading the instructions and the required parts of the program. It was difficult to understand exactly how the project was going to be tested as well. The first thing we worked on was determining the command prompt inputs to properly open the CSV file. We hard coded the parameters based on the test command given the assignment description. The command was:

```
cat inputFile.csv | ./simpleCSVsorter.c -c columnName > outputFile.csv
```

- Our program reads from the stdin and take the parameter at argument 2, which is the column name. We used the getline( ) function to get the lines from stdin and stored it in a buffer.

String Tokenizer

- At first, we used the predefined strtok( ) method from the library but we realized that it would not work with quotations marks. So we made our own tokenizer method to handle quotation marks and new line characters. It works similar to the strtok( ) method where we replace the delimiter with the null terminator character and return the string read. We made a function that takes in a the whole row line and gets the proper data under the column entered by the user.
- getField( ) method: This method used the tokenizer function to traverse the row and get the appropriate data under the specified column.

Linked List

- At this point we were able to get the full row data, and the data under a specified column. By putting this in a loop were able to do it with each row. So the next step was to create a linked list with nodes. We designed the struct node to have a field for the column data, the full row data that is unchanged and a next pointer. We designed the node to have a 2 separate fields for integers and chars. If the column name data type was an integer, then we would set the intValue of the node to the data, and similarly, if the column name data type was a char, we would set the charValue of the node to the data. This way, were are

able to store both data types and sort on the appropriate one. Next, we had a while loop to loop through each row and get the data then make a node and link it to the rest of the list.

Array of Linked List

- Our initial approach was to traverse the linked list and make an array of pointers that point to each node. This way, we can do merge sort on the array instead of a linked list. We ran into some problems here because handling all the node pointers got confusing. Below, there is some code we used to create the array but ultimately we deleted the code and ended up doing merge sort on a linked list.

```
node *(*putIntoArray(node * head, int totalNodes))
{
  node *nodeArray[totalNodes];
  node *(*p)[] = &nodeArray;

  node * current = head;
  int j = 0;

  //Traverse linked list and point array index to nodes
  while(j < totalNodes && current != NULL)
  {
    (*p)[j] = current;

    current = current->next;
    j++;
  }
  return p;
}
```

The code above properly creates an array of pointers from the linked list. The issue we ran into, was creating a mergesort function to take in an array of pointers and properly sorting on that. We scratched this code and starting working on a mergesort function for linked lists.

Mergesort on Linked List

The mergesort for a linked list takes in a head point and returns the head pointer of the sorted list. It works similar to an array mergesort, where it breaks down the linked list and reassigns pointers based on if the node value is less than or greater than another node value. We had to do multiple mergesort function for chars and integers because the comparators for the datatypes are different. We tried just having 1 generic mergesort function but the way our node structure was set up, it was difficult to determine which datatype to sort on. So, we call the appropriate mergesort from the main( ) function to sort on integers or chars. When doing mergesort on a linked list, we move one pointer by two and another by just one. Therefore, when the first pointer gets to the end of the linked list, the other one will be halfway through the linked list. This yields the place where to split the list into sublists. You then repeat this process with recursion until you have broken up

the linked list completely. Then we merge the sublists into larger sublists and here is where we sort the nodes. We keep on combining sublists into larger, sorted sublists with recursion until you make the whole linked list in sorted order.

Printing Out Data

While one of us was working on the mergesort for linked list, the other partner started working on the output. Throughout most of out testing we used the CSV file given below because it takes into account many different situations we can run into with the movie_metadata.csv file.

```
food,calories,fat
"soup,ggg",200,12
pie   ,500,25
   celery,-10,0
```

The CSV file above looks a bit weird but that is intentional so we can test cases such as extra white spaces and quotation marks. We also used this to test our stringtokenizer method. Once the data was put into the linked list, we made a function to print the linked list and the rows corresponding to it. This function took in a head pointer as a parameter and printing the list so that it can be directed to stdout. This function was also used to print the sorted linked list.

Final Output

Once the mergesort for linked lists was finished, we outputted the data with the printSortedList( ) function. We had a couple bug errors with the node pointers but eventually we were able to get it to output correctly. Then we redirected stdout to a CSV file and opened it in excel to check that it was displaying correctly. The CSV file sorts based on the column name without errors.