```
NAME:
    SIMPLE CSV SORTER

DEVELOPERS:
    Justin Chan
    Forrest Smith

OVERVIEW:
    simpleCSVsorter.c will sort a csv file (comma separated value
file) by a given column. The entries are sorted via mergesort.
    This program works for all general csv files (extra credit 1).
    Extra credit 2 is included for the data analysis

GOALS:
    To sort ANY csv file of any size, length, and category.
    To run with smallest space and time complexity.

SOLUTION:
    Our merge sort functions takee in values to be sorted along with a
format specifier. This is in order to select a proper comparator for
integers, strings (char*), or doubles.
    Values are passed in via row structs which hold string arrays of
each entry per row. We select each entry of a row via the column
number by matching column received via commandline arguments. After a
lookup of the column, we determine if the column is entirely strings,
doubles, or integers, and then we are able to select our comparator.
    Then the row structs are sorted based on the targeted entries, and
the new CSV is saved in a file.

HOW TO USE:
    To compile, open command line and cd into the directory holding
"simpleCSVsorter.c", "simpleCSVsorter.h", and "mergesort.c", and a
makefile. Type "make", then type "cat [filename].csv | ./
simpleCSVsorter -c [column]" where "filename" is the name of the CSV
to be sorted and "column" is the category to be sorted by which
appears in the CSV.

    Upon completion of the program, a CSV will be created with the
content sorted as specified by the user.

---------

ORGANIZATION:
    - enum _format{STRING, INTEGER, DOUBLE} format: Holds enum formats
for easy comparison when determening a column's type to sort by.
    - struct _Row: char** entries which holds the array of strings of
entries for all the columns
    - struct _Header: char** titles which holds the array of strings
of the titles from row 0. format* type is an array which holds the
enum format for the column
```

ASSUMPTIONS:
- Create generalized version from the start to work with any CSV file.
- Some of the movie_metadata.csv entries had printed characters that we assumed were spaces but actually were not. We had to account for this special character in our code.
- Initially looking at the data, there are empty/null entries. We made sure to sort these to the top as instructed.
- If everything in a column is a number but meant to be a string (eg. movie names as only numbers), it should be sorted as an number anyway. If there is a majority numerical, but with strings, then they should be sorted as strings.
- We assume we will only ints(+/-), doubles(+/-), or string and won't get pi, e, complex numbers, scientific notation.
- Empty files

DIFFICULTIES:
- Initial difficulties invovled storing the entries into character arrays. The solution was to dynamically allocate space for singly and doubly pointed arrays to hold arrays of strings and arrays of row objects holding arrays of strings.
- Further challenges ivolved the best way to read from the input. One solution was to use read(), but using getline() and tokenizing it manually was much easier to execute.
- Determening how to store the type of the column and storing column header/data. We decided to createa single column object which refers to each entry of the first row of the CSV. The struct we created would hold the string entry and the type via enum of {INTEGER, STRING, DOUBLE}
- Handling negative numbers dealt with via extra edge cases

TESTING PROCEDURE:
1) The testing procedure involved first printing out each line's string array after it was parsed and tokenized to ensure the file was read properly.
2) Then, creating each row object and inserting them all into the Rows array, I printed out each one to ensure all the entries were properly separated and copied correctly.
3) For merge sort, we hard coded test cases into row structs and sorted them.