David Tian (dt474) & Rahul Boppana (rsb172)

**Design:**

The first steps we took were to fix any issues with the original program from Assignment 0. For example, we added a check to make sure that every row had the same number of columns as the header. We also changed some malloced strings to arrays and removed their respective free() statements to make things simpler.

Next, we had to find a way to read through directories using C. We used the dirent.h library to open DIR objects on the specified input and output directories. The DIR objects can be iterated with readdir() to generate a directory entity, or dirent, for each file object in the directory. The dirent struct provides all the information we need, including the type of file object and the file name.

The last main topic was processes and forking. We were required to fork when we came across a nested folder within a directory and to fork to sort when we came across a valid csv. A lot of the functions were built into the linux system and c package but it was important to understand how to use them and how to take advantage of their return values as processes cannot communicate with each other. As a general description, every time we forked, we checked which instance we were in (parent vs child) and conducted the respective operations. To avoid fork bombs and the like, we structured the program to wait() every time a fork was called.

**Implementation Basic Overview:**

All of the sorting aspects of the program were carried over from assignment 0 except the process of sorting is now packaged in a method that gets called whenever a compatible csv is discovered. Our main method is now responsible for checking arguments and setting up initial directories. Once the number of arguments are checked, the flags are taken in whatever order they are present and the column and directory values are set. The default is to set the directories to current for both. Then the first process ID is printed and we begin forking. We check each entry in a directory and fork a process that recursively enters another directory by passing in the calculated directory path location or sorts a file. After every fork, the program is basically split up into 2 areas with instructions specified for the child process and instructions for the parent – mainly to wait on the child and check the exist status. It is important to note that we use the exit status to keep track of the total processes to be printed in the end. With nested directories and therefore nested child processes, this allows us to keep track of the lower levels and return that data back to the initial execution instance. Whenever a child is called, the process ID is printed out and on forking errors, the program gracefully halts execution. If a directory value was another directory, the child sets the path to the directory and passes that into the recursive call and the parent waits. If the directory value was a file, the file extension is also checked and the sorting method is called with the file name passed in while the parent again waits for a success or failure status. The sorting method was modified to take a filename and the inputDirPath/outputDirPath values to find the file. The global values are set during each fork to where that fork instance should be looking in the file system.

**Headerfile:**

The headerfile contains a lot of sorting structures and methods carried over from assignment 0. The new additions are global variables to keep track of the process counts and directory paths/names. For example inputDirPath keeps track of the complete path to and location of a given file as does outputDirPath with respect to output. firstProc is simply for the sake of print formatting so the first process isn’t printed with a preceding comma. endsWith is a method to check a file extension or to see if a file has been sorted. All other fields were present in the prior assignment.

**Testing Process:**

**Instructions for Running:**

Please follow the same instructions outline in the Asst1 assignment description. It specifies to run the program using

./scannerCSVsorter -c column\_name [-d input\_directory] [-o output\_directory]

to run the file once compiled.