ECE 251
Project 2
Read from, Sort, Write to

Minh-Thai Nguyen, Sophie Jaro

April 2019

1 Program Specifications

This program uses ARM assembler for Raspberry Pi to read from a file with maximum 100 lines of 32-bit integers, sort the numbers, then write the sorted array to a new file.

Implementing the project consisted of the following two parts: reading from the input file, sorting the numbers in the file, and writing to a new output file.

The program required scanf, printf, fscanf, fopen, and fclose to read from and write to files.

2 Architecture

First, necessary C std library functions like scanf, printf, fscanf, fopen, and fclose were imported to the program. Then memory was allocated for variables like the prompt messages; the input and output formats for scanf, printf, fscanf, fprintf, and fopen; the file pointer; the array to store the integers; the size of the array; the error messages; and the return variable.

The file reading portion of the program uses 6 functions: get_infile, init_read, read_line, open_file, close_file and eof. get_infile prompts the user for an input file and stores the name of the file into a variable called file_name. init_read starts the reading process by calling open_file and telling it to use the read mode, while also initializing the iterator of the of the array. open_file uses fopen to open the file in read mode with the file specified by its name. fopen takes in two parameters, the file name and the mode, and returns the pointer to the file in r1. If the file specified by the user does not exist the program will output an error saying: "ERROR: Could not find the input file." If the program successfully opens the file, it will begin to read data from the file and store it into the array. The reading and storing occurs in the read_line loop function. This loop keeps reading lines using fscanf until fscanf doesn't read anything, storing the value in each line into a specific index of an array. The

point where fscanf doesn't read a value is marked by fscanf returning a -1. The fscanf function takes in three parameters: the pointer to the file, the input format, and the memory address to store the variable. These parameters are put passed through r0, r1, and r2, respectively. If there are more than 100 lines, the program will throw an error: "ERROR: File is too big." Otherwise, the program will continue to eof. The function eof stores the size of the array into a variable called size_of_array and closes the file by calling close_file. close_file uses fclose to close the file specified by the variable "file".

An insertion sort algorithm was implemented for the sort portion of the project.First, the following variables were initiated: two pointers to the first value in the array and one pointer to the final value of the array. Two pointers to the beginning were necessary because one would serve as a bound while the other serves as an iterator (i). The insertion sort works by picking the i^{th} element of an array, comparing it to the preceding $0...i-1$ elements of the array, and deciding whether to swap it with one of these preceding element based on the output of the comparison. A swap will occur unless one of the following two conditions is met. First, if the address of the element is less than the address of the first element, no swap is necessary. The next (or $i + 1^{th}$) element in the array is then picked to be sorted. Second, if the element is greater than or equal to whichever $0...i - 1^{th}$ element it is being compared to, no swap is necessary. In this way, the program sorts the elements in ascending order. The sort ends when the final element in the array has been picked and sorted. This final condition is tested by comparing the address of the i^{th} element (stored in r2) to the address of the final element (stored in r1).

The output file writing portion consists of 5 functions: get_outfile, init_write, write_line, open_file, and close_file. get_outfile prompts the user for the name of an output file and stores the input into a variable called "file_name". init_write starts the writing process itself by calling open_file in write mode, which uses fopen to open the file specified in the variable "file". init_write also initializes the variables needed to loop through the sorted array and write each value to the user's specified output file. write_line is the looping function that writes each value of the sorted array to a line in the output file. It does this by passing the appropriate parameters to fprintf. The function fprintf takes the parameters of a file pointer, an output format, and a value to write the value to the file. These parameters are passed through r0, r1, and r2 for fprintf. After an element is written to the file, the index of the array is increased and the next element is written. This continues until the end of the array is reached.

3 Design Choices and Difficulties

The core challenge of this project was reading from and writing to files. This would have been extremely difficult without the printf, scanf, fscanf, fopen, and fclose functions. These functions made inputting and outputting data possible. Throughout most of the program, conditional checking that tested to see if the loop would continue were favored over those that ended the loop because it

allowed for more efficient code. Instead of having a line that branched to a function, the program could instead just move to the next function.