p2B: Spiral Traversal of a Square of Numbers

Due Feb 22 by 10pm **Points** 60 **Submitting** a file upload **Available** until Feb 22 at 10:33pm

This assignment was locked Feb 22 at 10:33pm.

- This project must be done individually.
  - *It is academic misconduct to share your work with others in any form including posting it on publicly accessible web sites, such as GitHub.*
  - *It is academic misconduct for you to copy or use some or all of a program that has been written by someone else.*
- All work for this project is to be done on the **CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities)** . You're encouraged to remotely login using the ssh command in the terminal app on Macs, or on Windows using an ssh client such as MobaXterm.
- All projects in this course are graded on **CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities)** . To receive credit, make sure your code runs as you expect on these machines.

Learning GOALS

The purpose of this assignment is to continue practice writing C programs and gain more experience working in this low-level, non-object oriented language. After completing both parts A and B of this project, you should be comfortable with pointers, arrays, address arithmetic, structures, command-line arguments, and file I/O in C. For part B, you will also be working with structures and file I/O.

OVERVIEW

For this assignment you will be writing a program **traverse_spiral.c,** which will traverse a given square of integers in a spiral pattern and output the result. Like p2A, this project will require you to work with dynamically allocated 2D arrays, and you are **not allowed** to use indexing (e.g., `array[i][j]`) to access the **2D array** that is used to represent the square. Instead, you are required to use address arithmetic and dereferencing to access them. Submitting a solution using indexing to access the 2D square will result in a **significant deduction of points**. You may use indexing to access any 1D arrays used by your program.

You're welcome to develop the solution in two phases. First, code a solution that uses indexing. Once you have that solution working, you can replace indexing with pointer arithmetic before final testing and submission. You're strongly encouraged to use incremental development to code your solution rather than coding the entire solution followed by debugging that entire code. Incremental development adds code in small increments. After each increment is added, you test it to work as desired before adding the next increment of code. Bugs in your code are easier to find since they are more likely to be in the new increment rather than the other code you've already tested.

SPECIFICATIONS

You are to develop your solution using the skeleton code in the file **traverse_spiral.c** found on the CS Linux computers at:

```
/p/course/cs354-skrentny/public/code/p2/traverse_spiral.c
```

The program **traverse_spiral.c** is run as follows:

```
./traverse_spiral <input_filename> <output_filename>
```

Where <input_filename> is the name of the file that contains the matrix to be traversed and <output_filename> contains the results of traversal. The format of the input file will be as follows:

- The first line will contain a positive integer *n* for the size of the square. You may assume the size will range from 0 to 100.
- Every line after that represents a row in the square, starting with the first row. There will be *n* such lines where each line has n numbers(columns) separated by commas.

So for instance, a file containing a square of size 4 that follows the format will look like the following:

```
4
16,15,14,13
12,11,10,9
8,7,6,5
4,3,2,1
```

Your program should write **each value of the spiral traversal followed by a space** to the output file. For the above example, the following should be written to the output file as a single line:

```
16 15 14 13 9 5 1 2 3 4 8 12 11 10 6 7
```

Two sample input files, named traverse1.txt and traverse2.txt, are provided in the directory as shown in the example below:

```
/p/course/cs354-skrentny/public/code/p2/traverse1.txt
```

These test files are not meant to cover all possible cases. We'll use several secret test files to evaluate your program's correctness.

The program should read and parse the input file to construct a matrix using the struct provided in the skeleton code. You will need to dynamically allocate memory for the 2D array. Check **hints** section to see how to parse the input.

To solve this problem, one way is to think of the matrix as comprising of rings and iterate from the outermost ring to the next inner ring. For example, the outermost ring consists of first row, last column, last row and then first column. Similarly, the next ring consists of second row, second last column, second last row, second column and so on. Keep iterating these rings until the matrix is exhausted. While traversing each ring, it is important to use appropriate bounds for that particular layer. You have been given **hints** in the skeleton code regarding this.

The sample run below shows the expected behavior of the program:

```
[skrentny@jimbo] (77)$ ./traverse_spiral
Usage: ./traverse_spiral <input_filename> <output_filename>
[skrentny@jimbo] (78)$ ./traverse_spiral non-existent-file.txt output-file.txt
Cannot open file for reading.
[skrentny@jimbo] (79)$ cat traverse1.txt
4
16,15,14,13
12,11,10,9
8,7,6,5
4,3,2,1
[skrentny@jimbo]] (80)$ ./traverse_spiral traverse1.txt traverse1-output.txt
[skrentny@jimbo] (81)$ cat traverse1-output.txt
16 15 14 13 9 5 1 2 3 4 8 12 11 10 6 7
```

HINTS

Using library functions is something you will do a lot when writing programs in C. Each library function is fully specified in a manual page. The man command is very useful for learning the parameters a library function takes, its return value, detailed description, etc. For example, to view the manual page for fopen, you would issue the command “man fopen”. If you are having trouble using man, the same manual pages are also available online. You will need these library functions to write this program. You do not need to use all of these functions since a couple of them are just different ways to do the same thing. We encourage you to refer to our code in the p2A code skeleton for an example of file I/O.

- **fopen()** to open the file.
- **malloc()** to allocate memory on the heap
- **free()** to free up any dynamically allocated memory
- **fgets()** to read each input from a file. fgets can be used to read input from the console as well, in which case the file is stdin, which does not need to be opened or closed. An issue you need to consider is the size of the buffer. Choose a buffer that is reasonably large enough for the input.
- **fscanf()/scanf():** Instead of fgets() you can also use the fscanf()/scanf() to read input from a file or stdin. Since this allows you to read formatted input you might not need to use strtok() to parse the input.
- **fclose()** to close the file when done.
- **printf()** to display results to the screen.
- **fprintf()** to write the integers to a file.
- **atoi()** to convert the input which is read in as a C string into an integer
- **strtok()** to tokenize a string on some delimiter. In this program the input file for a square has every row represented as values delimited by a comma. See here **(http://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm)** for an example on how to use strtok to tokenize a string.

When opening a file for reading using fopen, make sure you specify the correct mode (read or write) for which you are opening the file.

REQUIREMENTS

- Your program must use address arithmetic and dereferencing to access the 2D array representing the square.
- Your program must use the **->** operator when accessing the fields of a structure via a pointer variable.
- Your program must follow style guidelines as given in the **Style Guide**.
- Your program must follow commenting guidelines as given in the **Commenting Guide**.
- Your programs should operate exactly as the sample run above.
- We will compile your programs with

```
gcc -Wall -m32 -std=gnu99
```

  on the Linux lab machines. So, your programs must compile there, and without warnings or errors.
- Your program must print an error message, as shown in the sample run above, and then call exit(1) if the user invokes the program incorrectly (for example, without any arguments, or the incorrect number of arguments).
- Your program must check return values of library functions that use return values to indicate errors, e.g., malloc(), fopen(), and fclose(). Handle errors by displaying an appropriate error message and then calling exit(1).
- Your program must free up all dynamically allocated memory at the end of the program. For the 2D array you would need to free up all allocated memory in the reverse order of how you allocated it, i.e., don't just free the pointer to the array of arrays.
- Your program must close any opened files when you are done with them.

SUBMITTING Your Work

Submission has been enabled.
Submit the following **source file** under Project p2B in Assignments on Canvas before the deadline:

1. traverse_spiral.c

It is your responsibility to ensure your submission is complete with the correct file names having the correct contents. The following points will seem obvious to most, but we've found we must explicitly state them otherwise some students will request special treatment for their carelessness:

- **You will only receive credit for the files that you submit.** You will not receive credit for files that you do not submit. Forgetting to submit, not submitting all the listed files, or submitting executable files or other wrong files will result in you losing credit for the assignment.
- **Do not zip, compress, submit your files in a folder, or submit each file individually.** Submit only the text files as listed as a single submission.
- **Make sure your file names exactly match those listed.** If you resubmit your work, Canvas will modify the file names by appending a hyphen and a number (e.g., traverse_spiral-1.c) and these Canvas modified names are accepted for grading.

Repeated Submission: You may resubmit your work repeatedly until the deadline has passed. **We strongly encourage you to use Canvas as a place to store a back up copy of your work.** If you resubmit, you must resubmit all of your work rather than uploading just the files you have updated.

p2 (1)

Criteria	Ratings			Pts
1. Compiles without warnings or errors.	**6.0 pts** **No warnings or errors**	**0.0 pts** **error or at least 6 warnings**		6.0 pts
2. Follows the style and commenting guide.	**6.0 pts** **Full Marks**	**0.0 pts** **No Marks**		6.0 pts
3. Checks return values of library functions. Must check: malloc(), fopen(), fclose()	**6.0 pts** **Checks all**	**4.0 pts** **Checks some**	**0.0 pts** **No Marks**	6.0 pts
4. Displays the usage error message if argc is incorrect.	**3.0 pts** **Full Marks**	**0.0 pts** **No Marks**		3.0 pts
5. Frees all dynamically allocated memory.	**5.0 pts** **Frees all**	**3.0 pts** **Frees some**	**0.0 pts** **No Marks**	5.0 pts
6. Closes all opened files.	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**		1.0 pts
7. Uses address arithmetic instead of indexing a[][] on the 2D array square.	**9.0 pts** **Full Marks**	**5.0 pts** **Uses both**	**0.0 pts** **No Marks**	9.0 pts
8. Run test t6	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**		5.0 pts
9. Run test t3	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**		5.0 pts
10. Run test t4	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**		2.0 pts
11. Run test t5	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**		2.0 pts
12. Check get_square_size	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**		5.0 pts
13. Check write_to_file	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**		5.0 pts

Criteria	Ratings	Pts
		Total Points: 60.0