Puzzle Solver Assignment 1806ICT

### Note to marker

My solution reads in data using the following code:

FILE \* pointer;

int size;

// file reads here. Change according to your data name and location.

pointer = fopen("../test1.txt", "r");

if(pointer == NULL)

{

printf("Open operation failed.");

return 1;

}

When you test data you will need to be aware of this fopen("../test26.txt", "r") found in getSize() function on line 130 and also in fillArray() function on line 150 of my code solution.

# Data

## Variables

**Int size**

Size is taken as the first data from the test.txt file. Size will be used as a parameter for size for all arrays and array operation.

**Int solved**

Solved appears in main and also in solveArray() function and specialCase() function. In each case solved is used a variable which tests if puzzled is solved with the value of 1 equalling solved.

**Int myArray[size][size]**

myArray will take initial puzzle data from test.txt file using fscanf() within a nested loop. Pointer to myArray is passed to solveArray() function to be solved. Again it will be passed to specialCase() if puzzle turns out to be more difficult. When and if myArray is passed to specialCase() it will be passed updated to the point the solveArray() could achieve. The rest of the work to solve will be done in specialCase().

**Int copyArray[size][size]**

Initially memory is allocated for copyArray with using malloc() along with myArray but it is only used if solveArray() cannot solve the puzzle. In specialCase() copyArray is sent to createCopy() function to copy myArray, in it’s current partly solved state, into copyArray. Then copyArray is used to test values along with solveArray() function to the puzzle. If a solution is found it is then put into myArray and solved.

**Int rowTotal[size]**

A 1D array that hold row totals obtained from test.txt file.

**Int colTotal[size]**

A 1D array that hold column totals obtained from test.txt file.

# Functions

createCopy

Main

getSize

fillArray

printUnsolvedArray

solveArray

checkTotals

specialCase

## Flow Chart

printSolvedArray

(1)

**Int getSize()**

Returns size, read as the first data from test.txt with **fscanf()**.Size will be used as a parameter for size for all arrays and array operation throughout the complete program.

(2)

**void fillArray(int \*\*array, int \*rowTotal, int \*colTotal, int count)**

fillArray takes as parameters pointers to myArray, rowTotals and colTotal and also int size. The first data point in test.txt file is skipped by assigning it to x. Then nested loops, with **size** as limit, are use to fill myArray. Once size is reach, size+1 fills rowTotal and size+2 fills column totals. All data are obtained from test.txt using fscanf().

(3)

**void printUnsolvedArray(int \*\*array, int \*rowTotal, int \*colTotal, int size)**

Once myArray, rowTotal, colTotal are filled with data printUnsolvedArray() takes the 3 arrays mentioned ass parameters along with int size to print to console. The function uses nested loops and printf() to print unsolved array then row totals and then column totals.

(4)

**int solveArray(int \*\*myArray, int \*rowTotal, int \*colTotal, int size)**

This function takes as parameters pointers to myArray, rowTotals and colTotal and also int size. This is where the simple algorithm is used to solve the puzzle. 13 out of the 15 examples given in our assignment specs are solved here. The algorithm is explained below. If myArray is solved function returns 1 and if not after 20 loops of trying my program will move on to specialCase() to attempt to solve there.

(5)

**int specialCase(int \*\*myArray, int \*\*copyArray, int \*rowTotal, int \*colTotal, int size)**;

If puzzle is not solved in solveArray() mayArray will be sent here. This function takes as parameters pointers to myArray, copyArry, rowTotals and colTotal and also int size. Firstly important variables are created and set to 0, solved, check1 and check2. All of these variables need to equal 1 for the puzzle to be solved.

myArray comes here partly solved from solveArray(). A copy of myArray’s current state is put into copyArray through the function call of createCopy() described below. A loop tests a value between 0 and 9 in copyArray at the first location of a -1. With each guess check1 and check2 need to equal 1 for the puzzle to be solved. copyArray is sent with the guess value back to solveArray() to see if it solves. If it solved 1 is returned and stored in check1. Check2 will only be true if checkTotals() returns 1. CheckTotals() takes copyArray which has just been solved in solveArray() and uses nested loops to check if totals match up. If totals match up check2 receives a 1 and program moves onto next step. If totals don’t match a copyArray is refilled with myArray in its unsolved state to test a new value until a solution is found. Once solution is found the correct value is put into myArray and sent to solveArray() to be solved with the simple algorithm described below.

(6)

**void createCopy(int \*\*myArray, int \*\*copyArray, int size)**

Takes pointers to myArry and copyArray along with int size to genersate copies of myArray when needed. This function is used in specialCase() function only the case where myArray is not solved in solveArray() function. How it is used is described more in the description for specialCase() function. Copying is done with nested loops.

(7)

**int checkTotals(int \*\*copyArray, int \*rowTotal, int \*colTotal, int size)**

Checks row and column totals, returns 1 if true.

CheckTotals() takes copyArray which has just been solved in solveArray() and uses nested loops to check if totals match up.

(8)

**void printSolvedArray(int \*\*array, int size)**

This function takes myArray a soon as it is solved and size to print to console using nested loops.

# Algorithm

The simple algorithm is to look through a row and if only one -1 is found sum the values in the row and replace the -1 with the difference of the row sum and the row total. Loops through all rows doing this then through all columns using column totals. Continue going from rows to columns like this until solved. In some cases the simple algorithm could not solve such as when a row and a column are left with two -1’s in each. After 20 tries unsolved my special case algorithm will start. My special case solution algorithm is as follows. Use a copy array to test guess values in one of the -1’s and test if arrays solves and also if row and column totals are correct. Repeat until correct guess value is found. Once correct guess value is found use it in the real array and solve using the simple algorithm.

# Testing

I copy the 15 examples provided into test.txt files and also all the examples in the assignment document plus a few more. In total I tested 28 sets of data. My data files are saved as test1.txt, test2.txt, test3.txt …………. test28.txt. I found 2 of the given example data not to be solved with the simple algorithm, test11.txt and test16.txt. I also generated another like this myself test25.txt. The three I just mentioned were solved in the special case algorithm. All test data I used are provided in the appendix. Luke said to me that the test data will be no more difficult than the examples given so I am confident all given data set can be solved by my solution.

I spent a lot of time debugging my solution as it grew to include functions and solve for the special case. I tested for trying to read from a file that doesn’t exist and also when data is not integers or the wrong integer (test26.txt and test27.txt). In Both cases message is printed and program is stopped. Also if test data puzzle is to difficult for my solution a massage will be displayed and program will stop running. Such is the case for test28.txt.

## Test Data

Test1

3

3 9 5

8 1 -1

-1 -1 3

17 9 11

11 18 8

test2

3

3 -1 -1

1 2 2

6 -1 5

16 5 13

10 11 13

test3

3

5 -1 -1

-1 7 -1

8 -1 0

14 19 11

22 15 7

test4

4

1 1 -1 9

-1 7 2 2

3 -1 1 2

6 8 5 -1

19 12 6 27

11 16 16 21

test5

4

-1 -1 7 2

3 1 9 2

-1 2 4 -1

-1 7 6 7

21 15 16 22

13 15 26 20

test6

4

-1 7 3 5

-1 -1 -1 -1

6 0 9 5

3 -1 4 9

23 20 20 16

24 15 16 24

test7

5

5 9 2 -1 -1

-1 -1 2 9 0

8 3 3 -1 6

6 1 3 0 -1

3 4 0 -1 5

26 21 23 14 21

25 24 10 27 19

test8

5

1 3 5 -1 -1

5 3 -1 -1 9

5 2 1 7 -1

0 -1 0 0 8

-1 5 2 2 -1

21 27 19 10 23

17 15 9 25 34

test9

6

1 -1 1 -1 -1 5

-1 1 2 6 4 5

6 3 3 3 0 5

1 5 8 2 -1 -1

9 6 9 2 1 3

-1 4 0 1 4 9

24 25 20 26 30 25

31 23 23 21 23 29

test10

6

9 6 6 -1 1 8

1 4 8 -1 8 -1

-1 0 3 9 6 9

-1 -1 1 8 3 -1

6 8 -1 1 -1 9

8 9 7 2 -1 -1

35 32 27 19 41 35

27 27 33 31 32 39

test11

6

-1 0 0 6 0 -1

-1 7 6 -1 8 -1

5 9 6 -1 -1 2

1 -1 -1 -1 7 8

-1 6 1 9 3 3

2 7 3 1 3 -1

10 32 29 31 27 18

14 33 24 23 27 26

test12

3 7 3 5 4 0 2 -1

0 7 2 5 1 8 -1 4

5 -1 3 0 9 5 5 6

8 4 1 0 6 6 9 -1

9 5 -1 6 -1 -1 3 -1

8 -1 0 0 2 4 -1 8

-1 4 3 5 6 6 3 2

3 1 8 0 1 -1 -1 2

25 28 38 35 41 31 31 30

38 35 21 21 37 38 37 32

test13

8

1 -1 4 5 0 7 6 -1

1 8 2 4 -1 5 6 -1

0 6 2 4 6 1 6 -1

8 9 1 9 3 4 -1 0

1 -1 -1 -1 0 -1 -1 3

-1 8 2 2 6 8 1 4

4 0 5 4 8 7 6 -1

-1 4 0 2 6 6 4 -1

27 33 26 36 18 37 41 38

30 44 20 30 29 39 33 31

test14

8

8 5 6 3 4 6 -1 2

1 7 1 5 8 4 -1 6

6 1 -1 1 6 7 -1 9

-1 -1 4 4 5 -1 4 -1

6 -1 4 3 5 7 0 4

1 4 2 3 6 1 -1 -1

2 0 5 -1 -1 -1 0 7

8 4 0 2 9 -1 8 5

34 39 39 28 38 32 35 44

34 32 29 25 52 43 29 45

test15

10

9 -1 4 1 3 -1 0 0 9 7

8 3 6 8 7 -1 5 2 6 -1

4 7 6 7 0 1 6 9 8 -1

2 3 4 7 -1 5 0 4 9 2

-1 2 1 -1 8 2 1 1 -1 3

-1 6 4 9 -1 7 -1 0 6 8

2 3 -1 1 -1 -1 0 4 9 1

9 8 3 2 4 1 6 3 5 7

5 5 8 4 0 7 3 7 -1 9

1 3 6 5 2 3 5 -1 0 2

42 56 54 42 27 61 32 48 50 32

53 47 42 44 42 45 33 35 55 48

test16

4

-1 -1 -1 1

9 0 -1 -1

1 3 -1 -1

0 8 6 -1

7 10 14 19

10 12 18 10

test17

5

-1 7 -1 5 -1

7 -1 5 -1 2

2 2 4 -1 6

-1 -1 7 5 4

5 4 4 4 -1

20 20 20 19 18

17 15 25 25 15

test18

3

2 4 -1

1 -1 1

-1 8 -1

8 9 16

8 9 16

test19

4

0 0 -1 4

2 9 4 -1

-1 8 3 -1

0 -1 0 0

11 24 19 9

2 26 14 21

test20

4

-1 -1 -1 1

9 0 -1 -1

1 3 -1 -1

0 8 6 -1

7 10 14 19

10 12 18 10

test21

5

-1 -1 0 -1 7

0 4 5 -1 4

7 1 -1 9 8

0 5 0 -1 -1

3 -1 -1 1 8

15 15 29 22 23

15 14 17 22 36

test22

-1 6 9 -1 8

6 2 -1 -1 0

2 6 -1 9 0

4 -1 -1 2 -1

1 -1 9 4 7

27 16 26 15 26

15 21 38 19 17

test23

6

5 -1 -1 2 6 -1

-1 5 4 9 6 0

2 2 1 -1 2 4

-1 5 8 -1 2 9

-1 6 -1 3 6 6

6 2 6 -1 -1 4

31 24 11 35 30 27

28 23 28 18 30 31

test24

6

-1 5 -1 -1 -1 9

1 2 0 9 -1 1

-1 6 9 0 7 8

8 9 3 8 -1 2

7 -1 6 -1 1 -1

7 -1 2 5 5 4

41 18 30 34 30 25

28 29 29 33 28 31

test25

6

0 0 0 -1 0 -1

-1 7 6 -1 8 -1

5 9 6 -1 -1 2

1 -1 -1 -1 7 8

-1 6 1 9 3 3

2 7 3 1 3 -1

10 32 29 31 27 18

14 33 24 23 27 26

test26

3

hello

how

are

you

doing

test27

11

3 9 5

8 1 -1

-1 -1 3

17 9 11

11 18 8

test28

6

-1 0 0 -1 0 -1

-1 7 -1 -1 8 -1

5 -1 6 -1 -1 2

1 -1 -1 -1 7 8

-1 6 1 -1 3 3

2 7 -1 1 3 -1

10 32 29 31 27 18

14 33 24 23 27 26