1806ICT Assignment – Trimester 1, 2017

Carl Humphries
S5084150

Name: Carl Humphries
Student Number: 5084150

Data Section

STRUCT GRID

I found it necessary to use a struct to store other data types to represent the grid. This included a multidimensional array of integers to store all grid elements, an integer for the length of both the rows and column given they are the same size, and two arrays of integers for both the row totals and the column totals. This allowed me to send them through functions and intern made it easier to work with and understand.

NAME	TYPE	DESCRIPTION
length	int	This is to store the length of the grid both rows and columns
grid	int Array(2D)	This stores all the elements in the grid as [row][column] being there index
rowTotals	int Array	Stores the totals for each row to be easily obtained
columnTotals	Int Array	Stores the totals for each column to be easily obtained

readGridFile

There is only two main variables used in reading the file that is the data stream FILE and an input buffer to store the value temporary to be written to the grid.

NAME	TYPE	DESCRIPTION
inputfile	FILE pointer	To point to the data stream of the input file for reading in the grid
inputBuffer	int	To store each value read in order to be stored in the grid

testForComplete

There are only two arrays used to store the totals of all the elements in each row and column. This allows a test to see if they match the full totals to test if the grid is complete.

NAME	TYPE	DESCRIPTION
currentRowTotal	int Array	Stores the row totals at the current time of testing
currentColumnTotal	int Array	Stores the column totals at the current time of testing

General

There is a few variables used through and mainly in for loops these are for indexing arrays. ‘i’ generally will be used in 1D array and ‘i1’/’i2’ will be used in a 2D array.

NAME	TYPE	DESCRIPTION
i	int	Indexing 1D array
i1/i2	int	Indexing 2D array

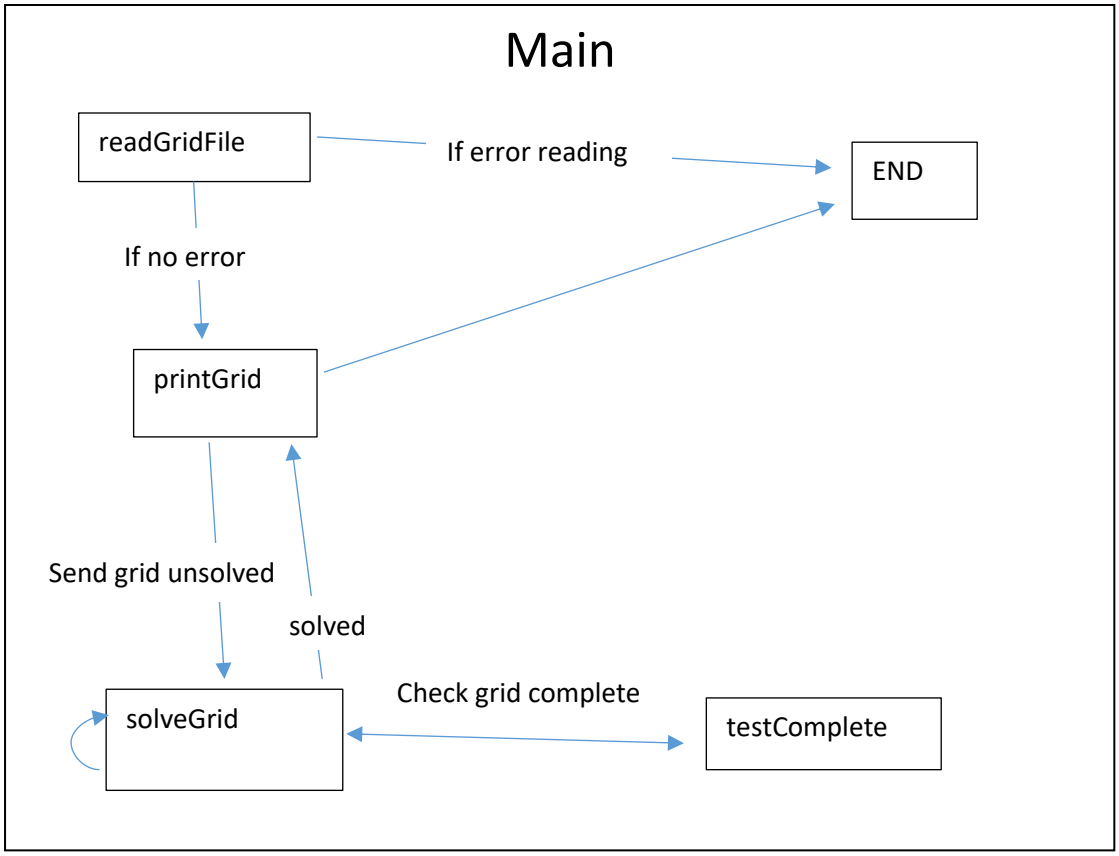
Name: Carl Humphries
Student Number: 5084150

solveGrid

In this function, there are two variables that store the position of the element that is being edited and tested one is row and other is column. In the solving part of the function there is variables to store the amount of -1s in the rows/columns and to store the totals of all non -1 elements. These allow calculations to get rowNumber/columnNumber which is what will be substituted in the element there is also a number variable which just loops from 9-0 to be subbed in.

NAME	TYPE	DESCRIPTION
row/column	int	Store index of the grid
rowCount/columnCount	int	Store the amount of -1's in the row and column
rowTotal/columnTotal	int	Store the total of all values in the row and column
rowNumber/columnNumber	int	Number to be subbed
number	int	Number to be subbed 0-9

Relational Chart For Functions



Name: Carl Humphries
Student Number: 5084150

Functions Sections

NewGRID

This function is to set the length of a new GRID to 0 this initialises it so there is no length. This allows all grid to be uniform and have 0 length to start.

readGridFile

The main use of this function is to read the input file and store in the struct GRID. This is done by reading a file in the format of the length of the grid, then the elements row by row and then the row totals followed by the column totals. It uses the length to alter the loops that store the elements and the totals into the structure. If there is no problem reading the file it returns a 1(true) otherwise it returns a false. It stores in the same struct that is send as a pointer.

printGrid

The purpose of this function is to print the data in the struct GRID in a nicely formatted method. Its done by loops that run through printing elements in blocks if they don't have -1 and if they do it prints a blank block.

testForComplete

This function is to test if a grid is "complete" is does this by testing if the totals of the grid are equal to what is currently in the grid. It calculates the totals of all the elements of the grid in its current state by looping through all positions then compares that to the totals given. If they are equal the grid is solved. Because the solveGrid function only tries values from 0-9 it doesn't need to check if the elements are in the range.

solveGrid

This function is by far the most complex and uses recursion to solve the grid. It loops through each index of the grid and finds a -1 if its not it moves to the next position. It uses different forms of subbing values then sends the altered grid back into the function to test the next value until it reaches the end. If at any point the test return a 0 because a value cannot be subbed or an incorrect value shows up it goes back to the point where it subbed the last and continues to do this until no point where a error occurs ir no problem with any number and or the testcomplete function returns a 1.

The methods it uses are it test to see if it can be solved because it's the only -1 in either column or row. This allows a number to be generated by subtracting all other elements from the total. It checks this generated number against 0-9 to make sure its valid as well as if it's the only -1 in both column and row it compares both generated number to make sure they are correct.

If this doesn't work, then it uses a brute force like method trying every value from 0-9. If any error returns a 0 the last subbed value gets set back to -1 and then goes back to the last position where something was change.

main

the main is mainly for stitching the program together and allowing for it to run. All it does is creates the GRID then sends it to the readGridFile to get the input. If it fails it prints error and stops the program. If success it prints the uncompleted grid. Then runs the solveGrid function and prints the grid when complete.

Name: Carl Humphries
Student Number: 5084150

Algorithms Section

readGridFile

```
if FILE exsist
   READ length
   LOOP until i1 is EQUAL to length
      LOOP until i2 is EQUAL to length
         READ element
         IF element is INTEGER
            STORE at (i1, i2) in grid
         END IF
      END LOOP
   END LOOP
   LOOP until i is EQUAL to length
      READ element
      IF element is INTEGER
         STORE at (i) in rowTotals
      END IF
   END LOOP
   LOOP until i is EQUAL to length
      READ element
      IF element is INTEGER
         STORE at (i) in columnTotals
      END IF
   END LOOP
   CLOSE FILE
   RETURN TRUE
ELSE
   CLOSE FILE
   RETURN FALSE
END IF
```

printGrid

```
LOOP until i is EQUAL to length
   PRINT "___"
END LOOP
PRINT newline
FOR ALL elements in grid
   IF element does not EQUAL -1
      PRINT "|element|"
   ELSE
      PRINT "|  |"
   END IF
   PRINT rowTotal
   PRINT newline
END FOR
LOOP until i is EQUAL to length
   PRINT "___"
END LOOP
PRINT newline
LOOP until i is EQUAL to length
   PRINT "|columnTotal|"
END LOOP
PRINT newline
```

testForComplete

```
FOR ALL elements
   IF element is in range 0-9
      ADD element to currentRowTotal at index1
```


Name: Carl Humphries
Student Number: 5084150

```
      ADD element to currentColumnTotal at index2
    ELSE
      RETURN FALSE
    END IF
  END FOR
  LOOP until i is EQUAL to length
    IF currentRowTotal at i is NOT EQUAL to rowTotal at i OR currentColumnTotal at i is NOT EQUAL to columnTotal at i
      RETURN FALSE
    END IF
  END LOOP
  RETURN TRUE
```

solveGrid

```
IF grid is COMPLETE
  RETURN TRUE
END IF
IF position is out of bounds
  RESTART with CURRENT grid
END IF
IF element at position is EQUAL to -1
  COUNT -1's in current row and column
  IF no other -1's
    CALCULATE number from other numbers and total both row and column
    IF only -1 in row and column
      CHECK row number with column number
      SUB in number
      IF solveGrid(currentGridState) RETURNS TRUE
        RETURN TRUE
      END IF
      SET number back to -1
      RETURN FALSE
    ELSE IF only -1 in row or column
      SUB number from row or column
      IF solveGrid(currentGridState) RETURNS TRUE
        RETURN TRUE
      END IF
      SET number back to -1
      RETURN FALSE
    END IF
  END IF
  LOOP number 0-9
  SUB in number
  IF solveGrid(currentGridState) RETURNS TRUE
    RETURN TRUE
  END IF
  SET number back to -1
  RETURN FALSE
ELSE
  IF solveGrid(currentGridState at next position) RETURNS TRUE
    RETURN TRUE
  END IF
END IF
RETURN FALSE
```


Name: Carl Humphries
Student Number: 5084150

main

```
PRINT “Please enter file name or directory using”
PRINT newline
GET file directory
IF readGridFile(grid, file) RETURNS TRUE
   printGrid(grid)
ELSE
   PRINT “Error”
   END PROGRAM
END IF
IF solveGrid(grid) RETURNS TRUE
   PRINT “Complete”
   PRINT newline
   printGrid(grid)
ELSE
   PRINT FAIL
   END PROGRAM
END IF
END PROGRAM
```

Test Section

readGridFile

To test this, I used the files we were given and ran the program to see if it printed errors. I also ran the print grid function to make sure the data was entered was correct and no miss matches. Before I made it store into the struct I made it print to make sure it was reading the correct line at the correct time. I didn’t come into too many issues with this everything was storing correctly if the file was in the right format.

printGrid

I hard codded the struct to make sure things would be printed correctly I ran through several times because I wanted it to line up nicely. It was mainly trial and error to determine the spacing and the decimal count.

testForComplete

For this I used multiple different test.txt files to make sure if it was unsolved it didn’t return true. I then edited the test.txt file and solved it myself to check if the grid showed up that it was complete when it was. I ended up making about 6 new files of variations of the test.txt files and fixed a bug that it was comparing the current columns to rows not current columns to columns. I found this by printing the totals and what they were comparing to just before they compared.

solveGrid

I tested this section by section using the printf function to make sure it was putting in the correct data I also used my own printGrid function to see where it was up to and where it got stuck. once I had the logic I just had to make sure the correct data was being compared and stored so I just made it print before storing and this helped me to trouble shoot. After I was happy it could solve grids I tested it against all the test files provided. I also made my own files to test if a blank space XbyX scattered or together would be able to be solved and after testing I noticed if its bigger than 4by4 it takes a lot longer. So every space up to 4by4 can be solved but anything over will take some time.

main

I used combination of all the function to test at run time but the main one that came in handy was the printGrid this allowed me to check that everything was working and no errors.