**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Project 1\*\*\*\*\*\*\*\*\*\*\*\*\***

Title

**Sudoku**

Course

**CSC-17A**

Section

**46097**

Due Date

**July 17, 2019**

Author

**\*\*\*\*\*\*\*\*\*\*\*\*\*Tyler Stevens\*\*\*\*\*\*\*\*\*\*\*\***

Table of Contents

Front Page Pg.1

Table of Contents Pg.1

Introduction Pg.3

Summary Pg.3

Description Pg.4

Pseudocode Pg.6

Flowchart Pg.12

Variables Pg.13

Concepts Pg.15

References Pg.16

Program Pg.16

**Introduction**

Project 1 Sudoku explores the classic game of Sudoku written entirely in C++. Imitating the whole Sudoku board with numbers 0 through 9 and symbols from the keyboard to draw the

![A close up of a keyboard

Description automatically generated]()familiar table. Solving Sudoku puzzles can be a significant challenge, the rules for traditional solution finding are quite straight forward. Each row, column, and nonet can contain each number, 1 through 9 and zeros represent blank spaces, exactly once.

The sum of all numbers in any nonet, row, or column must match the

small number printed in its corner. For traditional Sudoku puzzles

featuring the numbers 1 to 9, this sum is equal to 45. This project is important because it demonstrates how useful and versatile the C++ language is. The tools and concepts required for building this game will stay with me a throughout my software engineering career.

**Summary**

This game was coded entirely in C++ and uses various techniques learned in class to simulate the game of Sudoku. From utilizing pointers, structures, and advanced file operations, they all come together to run the game. The meat of the program is found in the *main.cpp* file and contains 370 lines of code. There is one user library I created in a header file named *Array2D.h*

Which contains the two structures needed for the program and is comprised of 25 lines of code. Inside the file you will two structures named *DynAry2* and *player* which contains a nested *DynAry2* object. There are many text files used in the program that are written to and read from. These files include all three Sudoku tables of difficulty, the game rules and a list of past ![A screenshot of a cell phone

Description automatically generated]()players with the score they received from playing.

There are roughly thirty-two variables used in the and eight functions used in the project. All in all, this project was not extremely challenging, but still hard and took many hours to complete. The idea itself is very simple, just create a 2d array puzzle and have the user enter an index and input value then validate that input. No that wasn’t hard the hard part was implementing all ideas from the project check list. I am sure there are many instances of “you didn’t have to do that”, but it was done to demonstrate my understanding of the constructs we have learned in the course. For instance, I write the score to the file in binary for really no reason other than to show I can. Overall the hardest part of the project was demonstrating my pointer prowess and being able to make pointer to pointer to structure of pointer etc. Finally, I don’t know exactly how many hours I spent on written the code, but every time I sat down and worked, I had a cup of coffee and 3 hours blocked out for solely programming. In fact, that is what I am doing as I finish this write up.

**Description**

The program opens to the rules of the game loaded from a text file named *instructions.txt*. The user is prompted to the choose difficulty (easy, medium, and hard) and to enter their name. From there the board will be loaded based on input and the game begins inside a while loop. The user inputs row followed by a space, the column followed by a space, and what number they want to input. The input is sent to a function that determines whether it meets the rules of Sudoku. There are three lives in total and each time the player inputs an invalid number they lose a life. The program will exit the game play loop when either the player loses all three lives or successfully completes the board. After which the users name and score, based on life counter, will be sent to a text file named *scores.txt* and displayed on screen. ![A screenshot of a cell phone

Description automatically generated]()![A screenshot of a cell phone

Description automatically generated]()

**Pseudocode**

**Enter main function**

Display rules

**Declare Variables and Initialize**

* chances for uses to try a value
* rows
* pointer to a pointer array for sudoku matrix
* fills structure from pointer array
* the score the user starts off with
* Two variable play state that determines loop execution
* users name

Input to variable name from user

print lives with cout statement

Print the board by calling function prnStrc(array2D);

**Enter game loop meat**

while(if game state one is true continue to execute) {

Pass array to test function and set it to game state one

* If game state one returns true, print out *true*
* If game state one returns *false*, print call and subtract user score by one

Pass array to win or lose function and set it to game state two

* If game state two is *false,* player wins the game and loop is terminated
* If game state two is *true*, player while loop continues

Use System clear screen to clear the screen and display updated lives and Sudoku Board.

}

End of while loop

Print out Game-over message

**Determine Score**

* If strikes are equal to two score is a *B*
* If strikes are equal to one score is a *C*
* If strikes are equal to zero score is a *F*
* Otherwise default initialization score of an *A* is recorded

Call function *add2File* and pass the name and score.

Call function *file* and pass *“Scores”* as a string to display past players name and score

Call function *destroy* to deallocated dynamically created memory from arrays

**Exit main function**

**Enter \*\*fill2D(row size, column size) function**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function \*\*fill2D. This function dynamically creates \*

//a 2d pointer array and fill it with contents of a file. \*

//It is filled with either a easy, medium, or hard Sudoku board. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Declare variables**

* Dynamically allocated to pointer to pointer array with row size nine
* File object
* Choice if difficulty
* An enum object with values easy, medium, and hard

**Determine difficulty from user input**

* Display that 1 = easy, 2 = medium, and 3 = hard
* User inputs choice
* Use a switch statement with enum cases to open corresponding files
* Default case is enum hard and will load the hardest Sudoku board

**Verify if file could be opened**

* If file could not open display error message and exit function.

Create and fill the second dimension of the pointer to pointer dynamic array with column size 9

**Fill array from file chosen by user**

* Loop through index of 2D array and file is from file object
* If data cannot be put into index from file display error message and where it accorded in 2D array. Also, exit function because there is no need to continue

Return filled array to function call

**Enter \*filStrc(2D pointer array, row of 9, column of 9)**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function \*filStrc. This function creates a dynamically allocated \*

//pointer objects to store Sudoku boards from a 2d pointer array. As well ` \*

//as the size of the rows and columns or the board. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dynamically allocate array

Point object to row variable in structure and set it equal to 9

Point object to column variable in structure and set it equal to 9

Fill Structures array variable with 2D array

Return structure array

***Enter* prnStrc(Dynaimic 2D structure array)**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function prnStrc. This function prints contents from a \*

//structure and formats it to prints out board like sudoku game \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Display column index**

* Print out a zero
* Use for loop print out column index
* If index is equal to 2, 5 or 8 print a space

**Display board**

* Use for loop to loop three 2D array and print contents
* For the horizontal line if row index is equal to 3,6 or 9 print a line of characters
* For vertical line if column index is equal to 3, 6 or 9 print out vertical line with character “|”
* If column index is zero display row index

Exit void function

***Enter function destroy(Dynamic 2Dpointer structure array)***

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function destroy. This function deallocates \*

//memory created from dynamically allocated arrays. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Deallocate memory**

* Use a for loop to loop through index and delete each
* If index reaches the size or the rows exit for loop
* Delete object array and just object

Exit void function

***Enter bool test2(Dynamic 2Dpointer structure array)***

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function test2. This function verifies if the user \*

//index input and number play by the rules of Sudoku. Which are \*

//there can be no repeats of number through the row, column, or in the 3x3 \*

//boxes. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Declare variables**

* Rows
* Columns
* Users input for use on board
* Starts for loop at the beginning of 3x3 table for the row index
* Starts for loop at the beginning of 3x3 table for column index

Input variable row, column and choice

To find the right index for row beginning in 3x3 box, use row index minus row index modulus 3

To find the right index for column beginning in 3x3 box, use column index minus column index modulus 3

**Check if cell is empty**

* If array at row and column that does not equal to zero return *false*

**Check for repeats in column**

* Use for loop to loop through array only change the column index
* If match is found return false otherwise continue through loop until column index is equal to 9

**Check for repeats in row**

* Use for loop to loop through array only change the row index
* If match is found return false otherwise continue through loop until row index is equal to 9

**Check for repeats inside 3x3 box**

* Nested for loop starting at 0 for three increments for each
* If match is found from comparing the array with input return *false*

If rules are met set value at user input for row and column equal to users input

Return *true*

***Enter function add2File(name, score)***

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of \*addFile. This function receives the name of the user and their \*

//overall score. A nested dynamic structure pointer object is created to store \*

//these values into a file. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Declare variables**

* Dynamically allocated *Player* object
* File Object
* nested structure for enter username and score from dynamically allocated *Player* object

**Input name and score to nested structure**

* user->use->score=s;
* user->use->name=n;

O**pen file with in, append and binary modes and verify if it was a success**

* If file was not opened print error message and exit function

Write name to file

Write score in binary to file

Exit void function

***Enter file(name of file)***

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function prnScore. This function copies data from file \*

//into a char array. After that the contents of the array are displays \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Declare Variables**

* Array size set to one thousand
* Character array with size one thousand
* Size of file set to zero
* Index set to zero

**Open file with string sent to function plus the .txt extension**

* If file is successfully opened get data from file and put it inside array while it doesn’t encounter an end of file marker
* Increment index and size every while loop success

**Display output**

* Set counter equal to zero and loop through as long as the counter is less than the size of the file
* Increment counter every success of outputting place in array

***Enter winOrLose(Dynamic 2Dpointer structure array)***

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Definition of function winOrLose. Determine if player \*

//has filled in board with at least one life \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Scan for blank spaces**

* Nested for loop that increment through rows and columns
* If array does not have any blank spaces return *true* meaning the player completed the board
* Otherwise return false

**FlowchartA close up of text on a white background

Description automatically generated**

**Variables**

**Main**

int strikes=3; //chances for uses to try a value

int rows=9; //rows

int \*\*dynAry=fill2D(rows,cols); //pointer to a pointer array for sudoku matrix

DynAry2 \*array2D=filStrc(dynAry,rows,cols);//fills structure from pointer array array

char score='A'; //the score the user starts off with

bool stat1,stat2=true; //play state

string name; //users name

**fill2D**

int \*\*a=new int\*[row]; //sudoku array

ifstream dataFile; //file object

int choice; //for choosing difficulty

enum funTime {zero,easy,medium,hard}; //difficulties

a[i]=new int[col]; //dynamic array

**fillStrc**

DynAry2 \*d2=new DynAry2; //dynamic memory allocation

**Test2**

int x, //rows

y, //columns

z, //users input for board

bx, //starts for loop at beginning of 3x3 matrix row

by; //starts for loop at beginning of 3x3 matrix column

**add2File**

Player \*user=new Player; //create object from struc

ofstream inFile; //file object

user->use=new DynAry2; //nested struc for enter user name and score

**file**

int arraysize = 1000; //array size

char myArray[arraysize]; //character array with size 1000

int size = 0; //initial file size

int i = 0; //initial index

**Concepts**



**References**

*Starting out with C++ from control structures through object*s. 8th edition, Tony Gaddis

Dr. Mark Lehr <https://github.com/ml1150258/2019_Summer_CSC_CIS_17A/blob/master/Class/Array_of_Structure/main.cpp>

**Program**

[**https://github.com/TS2535824/StevensTyler\_CSC-17A\_46097/tree/master/Projects/Sudoku**](https://github.com/TS2535824/StevensTyler_CSC-17A_46097/tree/master/Projects/Sudoku)