Project 1

<Sudoku>

CIS-17A 48983
Branden Hitt
Date: 10/25/2015

Table of Contents

Introduction

Title: Sudoku

This is a small program that lets a user play a game of Sudoku.

At the beginning of the program, the user is greeted by a message saying “Welcome to Sudoku”. The user is then asked if they are a new player or not. If they are, it loads their save. Otherwise, it creates a file for them. The user is then sent to a main menu where they have 5 choices on where to proceed. Those choices are to 1)Display the instructions on how to play, 2)Display the high score records from previous players, 3)Show the current user's stats, 4)Start a new game, or 5)Exit the program. After completing their choice, the user is then sent back to the menu where they can decide again until they exit the program.

1)Instructions: “There is only one simple rule: Fill in the spaces so that each of the nine rows, each of the nine columns, and each of the nine 3x3 sections contain all the numbers from 1 to 9. There are 3 difficulties to choose from, and the only way to lose is to reach a total of 5 errors on the able before winning. During the game you will be prompted for the index of the row first, followed by the column, and then you input the number you believe to be correct. Good Luck”

Example Solved Sudoku Puzzle:

```
            ABC   DEF   GHI    ← index for input

    a       4 3 5  2 6 9  7 8 1
    b       6 8 2  5 7 1  4 9 3
    c       1 9 7  8 3 4  5 6 2

    d       8 2 6  1 9 5  3 4 7
    e       3 7 4  6 8 2  9 1 5           ← actual puzzle (numbers in grid pattern)
    f       9 5 1  7 4 3  6 2 8

    g       5 1 9  3 2 6  8 7 4
    h       2 4 8  9 5 7  1 3 6
    i       7 6 3  4 1 8  2 5 9

    ^ index for input
```

2)High Score Display: This choice in the menu directs the player to a list that is displayed to them. The list is read from an included file. Each member of the list is shown with these values:

Sample high score:

```
Name                : Lou Costello
Total Games Played  : 20
Easy Games Won      : 5
Medium Games Won : 5
Hard Games Won      : 0
Win Rate            : 50.00%
```

3)Current User's Record: This choice in the menu directs the player to a prompt where they input the file location of their stats. The file is then loaded and displayed:

Example of display:

```
Name                : Branden Hitt
Total Games Played  : 20
Easy Games Won      : 5
Medium Games Won : 5
Hard Games Won      : 10
Win Rate            : 100.00%
```

4)New Game: This choice in the menu begins a new game for the player. The first action in the game is a prompt to decide what difficulty they want to play. The choices are Easy, Medium, and Hard. The difference between the difficulty levels is the amount of "Givens" on the board and how they are spread out. (A given is the numbers that are already on the Sudoku Puzzle from the start.) More givens = easier game.

Sample Easy Puzzle:

```
        ABC    DEF    GHI

a       _ _ _  2 6 _  7 _ 1
b       6 8 _  _ 7 _  _ 9 _
c       1 9 _  _ _ 4  5 _ _

d       8 2 _  1 _ _  _ 4 _
e       _ _ 4  6 _ 2  9 _ _
f       _ 5 _  _ _ 3  _ 2 8

g       _ _ 9  3 _ _  _ 7 4
h       _ 4 _  _ 5 _  _ 3 6
i       7 _ 3  _ 1 8  _ _ _
```

→ The puzzle completed:

```
        ABC    DEF    GHI

a       4 3 5  2 6 9  7 8 1
b       6 8 2  5 7 1  4 9 3
c       1 9 7  8 3 4  5 6 2

d       8 2 6  1 9 5  3 4 7
e       3 7 4  6 8 2  9 1 5
f       9 5 1  7 4 3  6 2 8

g       5 1 9  3 2 6  8 7 4
h       2 4 8  9 5 7  1 3 6
i       7 6 3  4 1 8  2 5 9
```

5)Exit the Program: This choice is exactly what it states. The user is exited out of the program.

Summary

Project size: 1101 lines (with comments)

Number of Variables: ~94

Number of Functions: 22 (+1 for main)

For this project I decided to devise a program that could play games of Sudoku. Throughout the years, I have played many games of Sudoku myself and was curious on how I could make a C++ version of it. In the last C++ class, I chose to make a program for Texas Holdem which turned out to be much more difficult than I had expected, so for this project I was hoping to pick something simpler. Sudoku turned out to be much simpler and in fact may have been too simple to incorporate the new concepts we had covered in class. This forced me to be creative and come up with ways to add them in (such as structures of player details).

The "gameplay" itself is just a simple 2D array that holds numbers and allows the user to input numbers to complete it. The 2D array is compared to another 2D array that holds the completed puzzle. If the user attempted to overwrite one of the original numbers (or givens) then they are stopped and forced to pick another space to edit. The game also keeps track of errors on the board and if the user reaches 5 total errors written on board then they lose. All said and done, the coding behind that did not take me long and the only real problem I ran into was formatting the game to be readable enough to play. Unfortunately, I do not have any experience in GUI's yet so it looks a bit archaic, but I am still happy with how it turned out.

My use of the other concepts we learned in class was mostly implemented in keeping high scores and saving input. For example, the high scores for players and the user are part of structures that are input and output from binary files. In addition, I kept the high scores in an array of structures as practice for use in sending them to functions and utilizing pointers to the array of structures.

Major Concept: Memory allocation:

I utilized memory allocation to hold a 2D dynamic array. The array itself held numbers for the givens in a puzzle. A seperate function swept through the table to find the position of the givens and then input that into a 2D array. The reason it had to be dynamic is because different puzzles have different amounts of givens. The array was then used to check and make sure the givens weren't being edited by the user. After the puzzle is finished, the array is de-allocated.

Major Concept: Functions with structures as input/output:

Structures came in handy for holding player data and high scores of previous players. The functions would output a structure when first creating it. A separate function would use a structure as input to display its current contents. My biggest obstacle here was realizing that structures are not automatically passed by reference. That was a simple fact I overlooked that ended up costing me a lot of time.

Major Concept: Pointers with arrays and arrays of structures:

The biggest problem I had with this project may have been trying to understand how to utilize a pointer to an array of structures since I wanted to use the pointer as a function argument. It was as if I tried to run a marathon before learning how to crawl. Of course, I could not find specifically where in the book it would help me achieve this so it was a great deal of trial and error. It would have been easy to just pass in one of the structures in the array at a time, however I wanted to be sure to know how to just pass in the whole array of structures and access its members from there. Eventually I was able to figure out a way for it to work (as far as I understand) which allowed me to send in the whole array of structures into a function. Most of what I did with pointers throughout this project ended up not being necessary, however I learned a lot about how to use them if I need.

Major Concept: Use of character arrays as well as string objects:

Character arrays were mostly used in my program to hold data in structures however I also utilized them in other instances, such as the beginning greeting sign. Although the player may not see the logic behind what happens to the greeting, I still was able to learn a lot from just testing out the different uses for char arrays. For instance, I learned how to concatenate two of the arrays together, as well as set the first letter of the character array to uppercase. Again, it was difficult to always find a use for things but I was able to learn a good amount of how the different concepts function. As far as strings go, I had already used them a lot in previous programs so I mostly just tried to work with character arrays. I probably could have found more uses for strings and tested out the extra concepts that the chapter detailed at the end though. Still, I feel that I understand the two much better now than before.

Major Concept: Reading and writing to binary files:

This concept ended up being the one that I probably learned the most from and had the most fun with even though it was a pain at times. My biggest struggle with this was deciding that I wanted to pass whole structures into binary files since that would probably be the best way to do it. This meant that I had to understand how binary output works along with outputting a structure in general. My biggest pitfall here was of course not realizing that structures passed into functions need to be pass-by-reference if I was to change them. I spent a lot of time reading in a structure from binary only to read it into a structure that disappeared before I used it for anything else (such as display). Once I got passed this I was able to use the concept correctly and store data into included files. This is something I'm more than certain will come in handy over time.

Future Improvements

Most of what I would want to improve would be display wise. Once I have knowledge in GUIs I might revisit this program and make it more visually appealing. Other than that I might add in an undo button since I never got around to that. That small improvement would get rid of some of the annoyance in the program. Lastly, I would add more puzzles into the program since currently I only included one for each difficulty level.

Simple Program Function Overview

```
main(
        gEdit(); ← edit the greeting's message (practice with c-string's and concepts such as toUpper)
        entRec(); ← fill structure for a new player
        writrec(); ← create a file for a new player containing structure
        readRec(); ← read in record for existing player
        menu(); ← menu that contains choices for the user
)
```

```
menu(
    instruc(); ← displays instruction on how to play the game
    showStr(); ← displays the high scores (an array of structures)
    showRec(); ← displays the player's record
    newGame(); ← starts a new game
    writRec(); ← update file for player stats
)
newGame(
    filTbl(); ← fills the table (2D array) with the correct Sudoku puzzle to be solved
    filKey(); ← fills the key table (another 2D array) with the correct key
    findGiv(); ← finds the givens in a puzzle and sets them in a dynamic array
    //loop until finished
        prntTbl(); ← displays the table
        entNum(); ← allows user to input a number to the puzzle
        check(); ← check the puzzle for errors or win
    //end loop
    destGiv(); ← de-allocates dynamic array used to hold givens
)


entNum(
    assign();  ← assigns a number to the letter input (translation)
    isGiven(); ← check to see if the position is a given or not
)
```

Psuedocode

```
/*
     * File: main.cpp
     * Author: Branden Hitt
     * Created on Oct 13th, 2015, 12:38 PM
     * Last edited: Oct 25,2015, 9:00 PM
     * Purpose: Create a game of Sudoku
     */
     //System Libraries
      //I/O standard
      // string usage
      // string length and string cat
      // file I/O


     //User Libraries
      //holds structure
     //Global Constants
     //Function Prototypes
      //menu for player
      //instructions on how to play
      //show the record of player on file
      //enter the record of player into a structure
      //read the record of a player from a file
      //write the record of a player to a file
      //display the high scores
      //start a new game
      //fill the table with the correct puzzle
      //fill the key with the correct key
      //print sudoku table
      //enter a number into table
      //assign a number to a char input
      //check table for win or errors
      //greetings edit
      //find the givens of the current puzzle
      //prevent editing givens
      //de-allocate the array for givens
     //Execution Begins Here!

     //Declare Variables
     //greeting array length
     //second part to be added
     //greeting message

     //is the player new or returning
     //prompt to be passed
     //structure of user data initialization
     //Greet the user and pull data
```

```
//edit the greeting
//loop and output greeting

//set up player file

//ask if the player is new


// if the player is new, fill the structure
//create the file
  //if a returning player, check for file

  //if no file exists, then create one

  //if a file exists, fill structure with files contents


  //send the player to the main menu


//Exit stage right


//*******menu*******//

 //declare variables
 //choice and repeat
 //boolean for menu skipping
 //size of high score array
 //create an array of structures

 //prompt for menu

     //menu

         //game instructions

         //display the high scores

        //show player stats

        //start a new game
           //record stats
           //write stats to file

           //set to skip to end
        //account for default entry
  //prompt for repeat
  //exit the program
}
//**********how to play***********************//
```

```
    //explain the rules
//************display the structure*******//
   //pause variable

   //loop through array of structures
        //display current structure
   //pause
//*******new Game*******//
 //Declare Variables
 //table dimensions
 //counter for errors in puzzle
 //win condition to exit loop
 //difficulty level
 //counter for givens
 //determined outcome for stats
 //create 2D array for table
 //find difficulty level
 //Create Table to be solved
 //create key
 //find the givens
 //Output
 //user enters in number
 //check for errors or win
 //after exiting current game
    //if win
    //tally which level diff is won
    //give congrats message
 //if loss, record loss

//de-allocate the givens array
//Exit stage right
//*******fill the table*******//
 //declare variables
 //open file according to difficulty level
 //fill table with files contents

 //close file
//*******fill the key*******//
 //declare variables
 //file input
 //open file according to difficulty level
 //fill table with files contents
 //close file
//*******print the table*******//
 //format
 //loop for format and display table
 //keep track of errors on table

//*******enter a number into a table*******//
 //declare variables
 //variables to hold selection
 //variables to hold translated selection
 //boolean for isGiven
  //input row
  //select row (ex: a)
  //input validation
  //assign number to choice (ex: a=0)
  //input col
  //select col (ex: A)
  //input validation
  //assign a number to choice (ex: A=0)
       //check to make sure its not a given
   //enter in the number to table
  //enter in guess
  //input validation
  //enter in guess into table

//*******assign a number to a char*******//
   //'A' returns 0 and so on...
   //default case for incorrect input

//*******check the table for win or errors*******//
  //total errors counter
  //win counter
 //loop to find errors
        //if error is found, increment error
 //set reference variable = current errors
 //if no errors
 //loop and check for win
    //counter for correct answers
  //if all correct, return true

  //if not finished, return false

//*******greeting's editor*******//
  //put both strings together
  //set the first letter to uppercase
//*******find the givens of the current puzzle*******//
  //declare variables
//find how many givens
       //rows = number of givens
   //create the number of rows
 //loop and create the columns
       //fill the array with givens positions
 //return the array
//*******check to make sure player isn't editing givens*******//
  //declare variables
  //loop to check givens
  //return the boolean
//*******de-allocate the array for givens*******//
 //Loop and destroy the columns
```

```
   //Destroy the rows
  //******enter player record*********//
   //declare struct
   //prompt for name in record
   //set all other values to zer0
   //return struct
  //******write player record to a file******//
   //open file
   //write to file
   //close file
  //*******read player record from file*******//
    //create structure
    //open file
    //read file into structure
    //close file
    //return struct
  //*********show player record*********//
    //calculate win rate
    //output all of structure
```

Concepts from the Book (Savitch 9th Ed)

Chapter	Concept	Description	Location
9. Pointers	Records *a	Pointer to structure of arrays	Line 176
	int **arrayG	Initializing a dynamic array	Line 216
	int **a	Pointer to dynamic array	Line 412
	delete [] a[i]	De-allocation of columns in an array	Line 451
	Delete []a	De-allocation of rows in an array	Line 454
10. Characters, C-Strings, string class	isalpha(rowIn)	Check to see if the char is alphabetical	Line 319
	islower(a[0])	Check to see if the char is lowercase	Line 391
	isupper(a[0])	Check to see if the char is uppercase	Line 393
	toupper(a[0])	Set the char to uppercase	Line 392
	char greet[GLENGTH]	A string held in a char array (C-string)	Line 44
	strlen(a)	Gives length of c-string	Line 385
	strcat(a,b)	Concatenates one c-string to another	Line 386

	String prompt	A string object	Line 48
11.Structured Data	struct Records	Structure initialization	Line 12 “PlayerData.h”
	Records temp	Structure Variable	Line 459
	Records hiScore[CHAMPS]	Array of structures initialization	Line 92
	showStr(hiScore,CHAMPS)	Array of structures as a function argument	Line 116
	return temp	returning a structure	Line 469
	Records *a	Pointer to an array of structures	Line 176
12. Advanced File Operations	fstream fin	File stream object	Line 486
	ifstream fin	Input file stream object	Line 265
	ios::in	Input mode	Line 487
	ios::out	Output mode	Line 475
	ios::binary	Binary mode	Line 475
	“user.dat”	Binary File	Line 475
	fin.read	Reads in binary input from a file	Line 489
	fout.write	Writes out binary input to a file	Line 477
	reinterpret_cast<char *>(&p),sizeof(p)	Converts a pointer to one of a char *	Line 477
	fout.fail();	Checks to see if file exists	Line 68

References

1.Textbook – Starting out with C++ (Gaddis, 7th Edition)

2. Sudoku Challenge – Uncle John's Bathroom Reader

Flowcharts

Flowcharts can be found in the Flowcharts folder

Program Code

```
/*
 * File: main.cpp
 * Author: Branden Hitt
 * Created on Oct 13th, 2015, 12:38 PM
 * Last edited: Oct 25,2015, 9:00 PM
 * Purpose: Create a game of Sudoku
 */

//System Libraries
```

```
#include <iostream>//I/O standard
#include <string>// string usage
#include <cstring>// string length and string cat
#include <fstream>// file I/O
using namespace std;

//User Libraries
#include "PlayerData.h"//holds structure
//Global Constants

//Function Prototypes
void menu(Records &);//menu for player
void instruc();//instructions on how to play
void showRec(Records);//show the record of player on file
Records entRec(string);//enter the record of player into a structure
Records readRec();//read the record of a player from a file
void writRec(Records);//write the record of a player to a file
void showStr(Records *,int);//display the high scores
short newGame();//start a new game
void filTbl(int [][9], int, short);//fill the table with the correct puzzle
void filKey(int [][9], int, short);//fill the key with the correct key
void prntTbl(int [][9], int, int &);//print sudoku table
void entNum(int [][9],int **,int);//enter a number into table
short assign(char);//assign a number to a char input
bool check(int [][9],int [][9],int,int &);//check table for win or errors
void gEdit(char [],char [],int);//greetings edit
int **findGiv(int [][9], int &);//find the givens of the current puzzle
bool isGiven(int **,int, int, int);//prevent editing givens
void destGiv(int **,int);//de-allocate the array for givens
//Execution Begins Here!
int main(int argc, char** argv) {
//Declare Variables
const int GLENGTH=25;//greeting array length
const int ILENGTH=9;//second part to be added
char greet[GLENGTH]={'w','E','L','C','O','M','E',' ','T','O',' '};//greeting message
char ing[ILENGTH]={'S','U','D','O','K','U'};
cout<<"*********************************"<<endl;
char prompt1='X';//is the player new or returning
string prompt="What is your name?";//prompt to be passed
Records user;//structure of user data initialization
//Greet the user and pull data
gEdit(greet,ing,GLENGTH);//edit the greeting
for(int i=0;i<GLENGTH;i++){//loop and output greeting
cout<<greet[i];
}
cout<<endl;
//set up player file
do{
cout<<"Before we begin, are you a new player?"<<endl;
cout<<"Enter in Y for yes or N for no:"<<endl;
cin>>prompt1;//ask if the player is new
}while((prompt1!='Y')&&(prompt1!='y')&&(prompt1!='N')&&(prompt1!='n'));
if(prompt1=='Y'||prompt1=='y'){
```

```
user=entRec(prompt);// if the player is new, fill the structure
writRec(user);//create the file
}else{
ofstream fout;
fout.open("user.dat", ios::in);
if(fout.fail()){//if a returning player, check for file
fout.close();
cout<<"No previous user file exists."<<endl;
cout<<"Creating a new file now...."<<endl;
user=entRec(prompt);//if no file exists, then create one
writRec(user);
}else{
fout.close();
user=readRec();//if a file exists, fill structure with files contents
}
}
//send the player to the main menu
cout<<"*********************************"<<endl;
menu(user);
//Exit stage right
return 0;
}
//*******menu*******//
void menu(Records &user){
//declare variables
char cho,repeat;//choice and repeat
bool skip=false;//boolean for menu skipping
const int CHAMPS=5;//size of high score array
//create an array of structures
Records hiScore[CHAMPS]={{{'B','r','a','n','d','e','n',' ','H','i','t','t'},20,5,5,10,100.00},
{{'D','o','n','n','i','e',' ','D','a','r','k','o'},20,5,5,9,95.00},
{{'A','m','e','l','i','e',' ','P','o','u','l','a','i','n'},20,5,5,8,90.00},
{{'A','n','d','r','e','w',' ','R','y','a','n'},20,5,5,7,85.00},
{{'L','o','u',' ','C','o','s','t','e','l','l','o'},20,5,5,0,50.00}
};
//prompt for menu
do{
cout<<"MAIN MENU"<<endl;
cout<<"Enter 1: Instructions on how to play"<<endl;
cout<<" 2: View High Scores"<<endl;
cout<<" 3: View Player Stats"<<endl;
cout<<" 4: Start a game"<<endl;
cout<<" 5: Exit the program"<<endl;
cin>>cho;
cout<<"*********************************"<<endl;
//menu
switch(cho){
case '1':{
instruc();//game instructions
break;
}
case '2':{
//display the high scores
showStr(hiScore,CHAMPS);
break;
}
case '3':{
showRec(user);//show player stats
break;
}
case '4':{
short stats=0;
//start a new game
user.ttlG++;
stats=newGame();
//record stats
if(stats==1)user.easyG++;
if(stats==2)user.mediG++;
if(stats==3)user.hardG++;
//write stats to file
writRec(user);
break;
}
case '5':{
skip=true;//set to skip to end
repeat='X';
break;
}
default:{//account for default entry
cout<<"Invalid Entry"<<endl;
skip=true;
repeat='R';
break;
}
}
//prompt for repeat
cout<<"********************************"<<endl;
if(!skip){
cout<<"Would you like to return to the menu or exit the program?"<<endl;
cout<<"Enter in R to return to the menu or X for exit:"<<endl;
cin>>repeat;
cout<<"********************************"<<endl;
}
}while(repeat=='R'||repeat=='r');
//exit the program
}
//**********how to play***********************//
void instruc(){
//explain the rules
cout<<"There is only one simple rule:"<<endl;
cout<<"Fill in the spaces so that each of the nine rows,"<<endl;
cout<<"each of the nine columns, and each of the nine"<<endl;
```

```
cout<<"3x3 sections contain all the numbers from 1 to 9."<<endl;
cout<<endl;
cout<<"There are 3 difficulty levels to choose from, and the only way to lose"<<endl;
cout<<"is to reach a total of 5 errors on the table before winning."<<endl;
cout<<endl;
cout<<"During the game you will be prompted for index of the row first,"<<endl;
cout<<"followed by the column, and then you input the number you believe"<<endl;
cout<<"to be correct. Good Luck"<<endl;
cout<<endl;
}
//************display the structure*******//
void showStr(Records *a,int s){
char pause;//pause variable
cin.ignore();
for(int i=0;i<s;i++){//loop through array of structures
cout<<endl;//display current structure
cout<<"RANK "<<i+1<<endl;
cout<<"Name : "<<a[i].name<<endl;
cout<<"Total Games : "<<a[i].ttlG<<endl;
cout<<"Easy Games Won : "<<a[i].easyG<<endl;
cout<<"Medium Games Won : "<<a[i].mediG<<endl;
cout<<"Hard Games Won : "<<a[i].hardG<<endl;
cout<<"Win Rate : "<<a[i].winR<<"%"<<endl;
//pause
cout<<"Press the Enter key to continue."<<endl;
cin.get(pause);
}
cout<<endl;
}
//*******new Game*******//
short newGame(){
//Declare Variables
const int DIMEN=9;//table dimensions
int errors=0;//counter for errors in puzzle
bool win=false, loss=false;//win condition to exit loop
short diff=0;//difficulty level
int count=0;//counter for givens
short stats=0;//determined outcome for stats
//create 2D array for table
int table[DIMEN][DIMEN]={};
int tableK[DIMEN][DIMEN]={};
//find difficulty level
cout<<"What difficulty would you like?"<<endl;
cout<<"1) Easy 2) Medium 3)Hard"<<endl;
cin>>diff;
cout<<"*********************************"<<endl;
//Create Table to be solved
filTbl(table,DIMEN,diff);
//create key
filKey(tableK,DIMEN,diff);
//find the givens
```

```
int **arrayG=findGiv(table,count);
//Output
do{
prntTbl(table,DIMEN,errors);
//user enters in number
entNum(table,arrayG,count);
//check for errors or win
win=check(table,tableK,DIMEN,errors);
if(errors>=5) loss=true;
}while(win==false && loss==false);
//after exiting current game
if(win==true){//if win
if(diff==1)stats=1;//tally which level diff is won
if(diff==2)stats=2;
if(diff==3)stats=3;
cout<<endl;
cout<<"******************************"<<endl;
cout<<"*CONGRATS!! YOU HAVE WON!!!*"<<endl;//give congrats message
cout<<"******************************"<<endl;
cout<<endl;
}else{
cout<<"Too many errors. YOU LOSE"<<endl;
stats=4;//if loss, record loss
}
//de-allocate the givens array
destGiv(arrayG,2);
//Exit stage right
return stats;
}
//*******fill the table*******//
void filTbl(int a[][9],int rC, short fNum){
//declare variables
ifstream fin;
//open file according to difficulty level
if(fNum==1) fin.open("puzzle1.txt");
else if(fNum==2) fin.open("puzzle2.txt");
else fin.open("puzzle3.txt");
//fill table with files contents
for(int x=0;x<rC;x++){
for(int y=0;y<rC;y++){
fin>>a[x][y];
}
}
//close file
fin.close();
}
//*******fill the key*******//
void filKey(int a[][9], int rC, short fNum){
//declare variables
ifstream fin;//file input
//open file according to difficulty level
```

```
if(fNum==1) fin.open("key1.txt");
else if(fNum==2) fin.open("key2.txt");
else fin.open("key3.txt");
//fill table with files contents
for(int x=0;x<rC;x++){
for(int y=0;y<rC;y++){
fin>>a[x][y];
}
}
//close file
fin.close();
}
//*******print the table*******//
void prntTbl(int a[][9], int rC,int &err){
cout<<" A B C D E F G H I"<<endl;//format
cout<<endl;
//cout<<" _____________________"<<endl;
for (int x=0;x<rC;x++){
cout<<" ";
for (int y=0;y<rC;y++){//loop for format and display table
if(y==0){
if(x==0)cout<<"a ";
if(x==1)cout<<"b ";
if(x==2)cout<<"c ";
if(x==3)cout<<"d ";
if(x==4)cout<<"e ";
if(x==5)cout<<"f ";
if(x==6)cout<<"g ";
if(x==7)cout<<"h ";
if(x==8)cout<<"i ";
}
if(a[x][y]==0) cout<<"_ ";
else cout<<a[x][y] << " ";
if(y==2||y==5)cout<<" ";
if((x==2&&y==8)||(x==5&&y==8))cout<<endl;
//if(((y+1)==3)||((y+1)==6))cout<<"| ";
}
cout<<endl;
}
cout<<"Total Errors: ("<<err<<")"<<endl;//keep track of errors on table
}
//*******enter a number into a table*******//
void entNum(int a[][9],int **b,int count){
//declare variables
char rowIn='0',colIn='0';//variables to hold selection
short row=10,col=10,guess=10;//variables to hold translated selection
bool gChk=false;//boolean for isGiven
do{
//input row
do{
cout<<"What is the letter of the row (ex: a)"<<endl;
cin>>rowIn;//select row (ex: a)
if(!(isalpha(rowIn)))cout<<"Input needs to be a letter from a-i."<<endl;//input validation
row=assign(rowIn);//assign number to choice (ex: a=0)
if(!(row>=0&&row<=9))cout<<"Invalid Entry"<<endl;
}while(row<0||row>9);
//input col
do{
cout<<"What is the letter of the column (ex: A)"<<endl;
cin>>colIn;//select col (ex: A)
if(!(isalpha(colIn)))cout<<"Input needs to be a letter from A-I."<<endl;//input validation
col=assign(colIn);//assign a number to choice (ex: A=0)
if(!(col>=0&&col<=9))cout<<"Invalid Entry"<<endl;
}while(col<0||col>9);
gChk=isGiven(b,count,row,col);//check to make sure its not a given
if(gChk==true){
cout<<"Cannot edit a Given"<<endl;
cout<<endl;
}
}while(gChk==true);
//enter in the number to table
do{
cout<<"What is the number you wish to input (1-9)"<<endl;
cin>>guess;//enter in guess
if(guess<1||guess>9)cout<<"Invalid Entry"<<endl;//input validation
}while(guess<1||guess>9);
a[row][col]=guess;//enter in guess into table
}
//*******assign a number to a char*******//
short assign(char c){
if(c=='A'||c=='a')return 0;//'A' returns 0 and so on...
else if(c=='B'||c=='b')return 1;
else if(c=='C'||c=='c')return 2;
else if(c=='D'||c=='d')return 3;
else if(c=='E'||c=='e')return 4;
else if(c=='F'||c=='f')return 5;
else if(c=='G'||c=='g')return 6;
else if(c=='H'||c=='h')return 7;
else if(c=='I'||c=='i')return 8;
else return 10;//default case for incorrect input
}
//*******check the table for win or errors*******//
bool check(int a[][9],int b[][9],int rC,int &e){
int ttlE=0;//total errors counter
int winC=0;//win counter
//loop to find errors
for(int x=0;x<rC;x++){
for(int y=0;y<rC;y++){
if(a[x][y]!=b[x][y]){
if(a[x][y]!=0) ttlE++;//if error is found, incriment error
}
}
```

```
}
e=ttlE;//set reference variable = current errors
if(ttlE==0){//if no errors
//loop and check for win
for(int x=0;x<rC;x++){
for(int y=0;y<rC;y++){
if(a[x][y]==b[x][y]) winC++;//counter for correct answers
}
}
if(winC==81)return true;//if all correct, return true
}
return false;//if not finished, return false
}
//*******greeting's editor*******//
void gEdit(char a[],char b[],int c){
//put both strings together
if( c >=(strlen(a)+strlen(b)+1)){
strcat(a,b);
}else{
cout<<"Greetings array is not large enough for both strings."<<endl;
}
//set the first letter to uppercase
if(islower(a[0])){
a[0]=toupper(a[0]);
}else if(isupper(a[0])){
//do nothing
}else{
cout<<"error"<<endl;
}
}
//*******find the givens of the current puzzle*******//
int **findGiv(int p[][9],int &row){
//declare variables
int fill=0,col=2;
//find how many givens
for(int x=0;x<9;x++){
for(int y=0;y<9;y++){
if(p[x][y]!=0)row++;//rows = number of givens
}
}
//cout<<"count : "<<row; <-- test purposes
if(row>0){
//create the number of rows
int **array=new int*[row];
//loop and create the columns
for(int i=0;i<row;i++){
array[i]=new int[col];
}
//fill the array with givens positions
for(int x=0;x<9;x++){
for(int y=0;y<9;y++){
```

```
if(p[x][y]!=0){
//cout<<"x: "<<x<<" y: "<<y<<endl; <-- test purposes
array[fill][0]=x;
array[fill][1]=y;
fill++;
//cout<<"fill: "<<fill<<endl; <-- test purposes
}
}
}
//return the array
return array;
}else{
//do nothing - no givens
cout<<"No Givens in puzzle"<<endl;//error case
}
}
//*******check to make sure player isn't editing givens*******//
bool isGiven(int **b, int count, int row, int col){
//declare variables
bool stop=false;
//loop to check givens
for(int x=0;x<count;x++){
if((row==b[x][0])&&(col==b[x][1])) stop=true;
}
//return the boolean
return stop;
}
//*******de-allocate the array for givens*******//
void destGiv(int **a,int c){
//Loop and destroy the columns
for(int i=0;i<c;i++){
delete [] a[i];
}
//Destroy the rows
delete []a;
}
//******enter player record*********//
Records entRec(string p){
//declare struct
Records temp;
cout<<p<<endl;//prompt for name in record
cin.ignore();
cin.getline(temp.name,30);
temp.ttlG=0;//set all other values to zer0
temp.easyG=0;
temp.mediG=0;
temp.hardG=0;
temp.winR=0.00;
//return struct
return temp;
}
```

```
//******write player record to a file******//
void writRec(Records p){
//open file
fstream fout;
fout.open("user.dat", ios::out | ios::binary);
//write to file
fout.write(reinterpret_cast<char *>(&p),sizeof(p));
//close file
fout.close();
}
//*******read player record from file*******//
Records readRec(){
//create structure
Records temp;
//open file
fstream fin;
fin.open("user.dat", ios::in | ios::binary);
//read file into structure
fin.read(reinterpret_cast<char *>(&temp),sizeof(temp));
//close file
fin.close();
//return struct
return temp;
}
//*********show player record*********//
void showRec(Records p){
//calculate win rate
float winRate;
if(p.ttlG>0)winRate = ((p.easyG)+(p.mediG)+(p.hardG))*100.0f/ p.ttlG;
else winRate=0;
//output all of structure
cout<<"Name : "<<p.name<<endl;
cout<<"Total Games Played : "<<p.ttlG<<endl;
cout<<"Easy Games Won : "<<p.easyG<<endl;
cout<<"Medium Games Won : "<<p.mediG<<endl;
cout<<"Hard Games Won : "<<p.hardG<<endl;
cout<<"Win Rate : "<<winRate<<"%"<<endl;

}
```