Project 2

Sudoku Version 2.0

Author
Omar Hernandez

Due Date
December 12, 2016

Class
CSC-5 48101

Index

1.1 | Game

Sudoku is a puzzle game where the objective is to fill the grid with numbers 1-4 or 1-9 depending on the size of the grid. Each column, row, and 3x3 square (subgrid or block) of the puzzle needs to be filled with different numbers in order to complete it. While it can be played with an empty grid where the player fills it up to their liking as long as they follow the rules, conventionally, a few numbers are placed to make the solutions more specific—what makes it a puzzle. The difficulty of the puzzle depends on how many numbers it starts out with and their placement. Generally, less difficult puzzles have more numbers because with more numbers, it is easier to make connections, while more difficult puzzles have less, making it harder to make the connections necessary to completing the grid.

1.2 | Method

It is basic Sudoku. The program prompts the user to select what type of grid they want to play: 4x4 or 9x9. Depending on what is picked, it fills the grid with numbers of the appropriate range bounded by rules of Sudoku—making sure columns, rows, and subgrids are filled with different numbers. To put it simply, it first completes a puzzle. Afterwards, the player selects the difficulty they want to play. In this program, difficulty determines how many cells to empty from the completed puzzle. The higher the difficulty, the more the algorithm will empty. The user then gets to select the tile they want to add a number to. Any other character that is not used in the puzzle will be negated by the program with exception to X and S as they are part of the menu—input validation. If the user gets stuck and is not able to add any more numbers to the puzzle, they will have to input X to exit out. If X is inputted, the array filled correctly is displayed to the player. When all the squares are filled in (only possible if each row and column together have different numbers), the program outputs the completed puzzle and displays “Victory!” to the user.

You can save and load previous puzzles with this program. This is only for 9x9 puzzles as they can be quite time consuming depending on the difficulty and person. As 4x4 puzzles are very easy and quick to finish, I felt no need to go through the efforts of making the function work for it as well. To save a puzzle while you are working on one,

input S. Saving is bundled with exiting in this program, so if one decides to save, they will immediately be taken to the end of the program. Version 2.0 includes a leaderboard that keeps the names of previous players as well as their scores. When the player completes a 9x9 puzzle, he/she will be prompted to add their name to the hall of fame. If their name already exists in the hall of fame, their score is incremented. The hall of fame is also sorted from greatest to least. As such, the players who have completed the most puzzles are going to be sorted to the top.

1.3 | Future Implementations

Previously, this project was completed without using arrays. Now that we have covered arrays in class and can therefore use them in our projects, I have accomplished what would normally be a complete nightmare with the older methods without it being too arduous and messy.

After Version 2.0 Submission:

Expanded Hall of Fame: As of now, it only stores the names and scores of people who have completed the puzzle. Ideally, it would also store the difficulty of the puzzles they have completed. Alternatively, a points system could be done, where higher difficulties provide more points.

Other Challenges: There are many derivatives of Sudoku that are possible to include into the program. An example of it would be Hyper Sudoku where more boundaries are created. Many of the variants are possible with a bit of tweaking of the randomization algorithm in the program.

Code Efficiency: A few of the functions, with a little more work, could be combined into one to make the code look cleaner.

After Version 1.1 Submission:

(V2.0✓) Erasure: Perhaps what is most lacking is the need to make corrections. Before this is done, I would like to learn about methods that would allow for the differentiation of text, such as colors, so the player can easily distinguish between the numbers they have inputted with the numbers the game initiated with. To determine whether a number can be corrected, the randomization could set the numbers from 1 to 4 as opposed to 49 to 53 (ASCII). In the display function, the numbers would be static cast to a short. This would allow for comparisons between the numbers the player set and the numbers

that the program set. This idea has been forgone for until the next implementation as this would raise numerous integer type issues without extensive infrastructure changes to accommodate for the new datatype.

(V2.0✓) Grid Size Selection: For future implementations, the plan is to give the user the option to do 9x9 puzzles while keeping 4x4 puzzles. A menu will be added to prompt the user to selecting their desired puzzle type.

After Version 1.0 Submission:

(V1.1✓) (Implemented) Better Randomization: The way it is right now, it gets the job done, although if played extensively, one may start to notice a pattern in the placement of numbers. ~~What would have been better is if I bounded a switch into a loop where the switch's variable would have a randomized value modded to the amount of numbers needed. Every case would then call a function that checks if the square already has a value or not.~~ The fourth and final implementation before the first submission uses the random function to determine where the first numbers will be placed. The amount of numbers it places ranges from 1 (extremely unlikely) to 8. It still needs improvement as it only does this with two numbers in an effort to prevent it from creating unsolvable puzzles. Putting more thought into this, what I could have done is make the program fill up a grid with the boundaries of a Sudoku game (no numbers vertically, horizontally, or in the square). After that, it would randomly pick squares to convert to 32 (ASCII code for space, " "). Of all the things left to do in this program, a better randomization algorithm tops the list as most wanted along with the ability to make corrections. As of version 1.1, randomization works as desired. More information provided in section 1.4.

(V1.1✓) (Implemented) Difficulty Select Menu: This implementation has no difficulty selection with exception to pre-made puzzles gathered from the Internet. In future builds, the 4 by 4 puzzle will be included alongside more difficulty puzzles such as the standard 9 by 9 and perhaps even the 16 by 16. As mentioned before, 9 by 9 is entirely possible being limited to the concepts learned this far, but with the methods used in this program, it would be pretty messy.

1.4 | Development

Total lines: ~700

Number of Variables: 20+

Functions: ~14

Version 2.0: Version 2.0 makes use of arrays, two dimensional in particular, and vectors to accomplish the most important tasks. Arrays have greatly facilitated many of the processes that previously required very messy and long coding. With the new capabilities, I was able to create a 4x4 and 9x9 puzzle generator which gets a few cells emptied out depending on the difficulty. It also outputs the solution if one decides to input X while playing the game, a feature that was absent in this project's first submission. Along with that, players can now finally edit the contents of a cell as long as it was not set by the program. The numbers that were added by the program are bolded in the display to make it differentiable. Finally, it also now stores the names of the people who have completed the puzzles (9x9 in particular) as well as the amount of times they have completed the puzzle. For that, vectors were used, and they allow for way more than enough entries to be stored.

Version 1.1: The fifth implementation fixes all randomization irregularities where the puzzles can be completely randomly generated. Rather than filling in random areas as in previous builds of this project, it now fills up the grid properly with random numbers and erases random tiles, number depending on the difficulty selected. Not only is the randomization much, much better, but, as the computer completes the puzzle first so to speak, all randomly generated puzzles can be completed. Difficulty selection for the randomly generated puzzles has also been added and determines how many tiles are emptied. Version 1.1 also records all attempts and completions.

Version 1.0: The code presented in this write up is my fifth implementation. The first implementation did not use functions at all. It is fully functional, but with functions, the code looks much nicer. The second implementation forewent the save/load function, but I eventually decided in including it because it was an opportunity to apply more of the concepts we have learned this far. The fourth implementation uses a better randomization algorithm and adds pre-made puzzles as another mode the user can select in the main menu.

2.1 | C++ Constructs

Chapter	Concept	Code/Application
2	Parts of C Program	✓
	cout	Instructions, display puzzle
	#include	*#include <iostream>*
	Variables and Literals	*const short SIZE=9*
	Identifiers	*char menu*
	Integer Data Types	*short gridT, int n*
	char	*menus*
	strings	*string wchGrid*
	Booleans	*bool is9x9*
	Variable Assignments	*gridT=9;*
	Arithmetic Operators	*x=rand()%(gridT);*
	comments	*//else "4x4", set grid variables to suit 4x4*
	constants	*const short SIZE=9;*
3	cin	*cin>>upperC;*
	Mathematical Expressions	*digit=rand()%9+49; //numbers 1-9u*
	Mixing Datatypes	*x=rand()%(gridT);*
	Type Castings	*srand(static_cast<int>(time(0)));*
*	Multiple Assignments	*n=i=20*
	iomanip	*setw(50)*
	char and string objects	*getline(cin, wchGrid);*
	cmath	*puzDiv=sqrt(gridT);*
4	Relational Operators	*i<SIZE*
	Independent, single line If	*if(upperC!='X'&&upperC!='S')cin>>lowerC;*
	Independent, multi-line If	*if(upperC=='S'||lowerC=='S'){ … }*
	if/else	game menu (menu used to fill cell)
	Nested Ifs	game menu (menu used to fill cell)
	if/else if	*else if(upperC=='X'||lowerC=='X')*
	Flags	*if(isSolvd&&is9x9)addRec();*
	Logical Operators	*else if(upperC=='X'||lowerC=='X')*
*	Numeric Ranges with Logical Operators	*while(dif<=0&&dif>5)*
*	Menu	game menu (menu used to fill cell)
	Input Validation	*while(wchGrid!="4x4"&&wchGrid!="9x9");*
	Switch/Case	*switch(menuIn)*
	String Comparison	*wchGrid!="4x4"&&wchGrid!="9x9"*
	Conditional Operator	*cout<<((isSolvd)?"Victory!":"Play again!")*
5	Increment/Decrement	*i++*
	Validating with Loops	*while(isDone==false)*
	Counters	*for(int n=0; n<SIZE; n++)*
	do while	*do{ … } while(isRep && counter<=30);*

	for loop	*for(int n=0; n<SIZE; n++)*
	Nested Loops	*scanning two dimensional arrays*
	File I/O	*inFile.open("prev.dat");*
6	Prototypes	*rndPuz(char [][SIZE], bool);*
	Passing by Value	*prntScr(scores, names, index, names.size());*
	Return/Void	*return false;* *void rndPuz(char [][SIZE], bool);*
	Static Local Variables	*static short puzDiv;*
	Pass by reference	*void markSrt(vector<int> &,vector<int> &,int);*
	Default Arguments	*bool isSaved(char [][SIZE], char [][SIZE], char [][SIZE], bool=true);*
	Exit	*exit(0)*
7	1 Dimensional Arrays	*a[indx[i]]*
	Parallel Arrays	*dfaults[SIZE][SIZE]*
	Function Arguments	*void rndPuz(char [][SIZE], bool);*
	2 Dimensional Arrays	*puzzle[SIZE][SIZE];*
	2D Array Functions	*void rndPuz(char [][SIZE], bool);*
	Vectors	*void prntScr(vector <int> &a, vector<string> &names,vector <int> &indx,int SIZE)*
8	Search	Ensuring input to grid is unique in column, row, and block
	Sort	Mark sort leaderboard
	Vectors Search/Sort	Mark sort leaderboard
9	Getting Address of Variable	*isPlybl=isLoadd(puzzle, dfaults, cmplte, &is9x9)*
	Pointers as Function Parameters	*bool isLoadd(… bool *is9x9)*

2.2 | Outside References

http://www.cplusplus.com/

- **cin.clear()** – clear error state of cin
- **\e[1m** – bold output, make values placed by program differentiable.

http://en.cppreference.com/w/

- **unsetf(ios_base::skipws)** – stop ifstream from skipping white space (reading empty cells in saved file)

3.1 || Working Product

Start

```
Sudoku!

                                A B   C D
                                - -   - -
                                ? ? | ? ? |  a
                                ? ? | ? ? |  b
                                - -   - -
                                ? ? | ? ? |  c
                                ? ? | ? ? |  d

Input P to create puzzle and play, L to load saved
puzzle, or B to display leaderboard.
```

Preferences

```
Play 4x4 or 9x9?
9x9
Select a difficulty (1-5), 1 being the easiest, 5 being the hardest
5

                          A B C   D E F   G H I
                          - - - - - - - - - - -
                            2 7 |       |       |  a
                          4     |     5 |     9 |  b
                            9   | 6     |       |  c
                          - - - - - - - - - - -
                            6   |   4   |   7   |  d
                                | 7     |       |  e
                            4   |       |       |  f
                          - - - - - - - - - - -
                                | 3   9 | 8 5   |  g
                          2   8 |   7   |   3   |  h
                              9 |       |       |  i
```

Load puzzle from file, bold numbers that were set by program.

```
Input P to create puzzle and play, L to load saved
puzzle, or B to display leaderboard.
L

                          A B C   D E F   G H I
                          - - - - - - - - - - -
                          6 2 1 | 3 8 4 | 7 9 5 |  a
                          7 5 3 | 9 6 1 | 2 8 4 |  b
                          9 4 8 | 2 7 5 | 3 1 6 |  c
                          - - - - - - - - - - -
                          8 3 2 | 7 9 6 | 5 4 1 |  d
                          1 9 7 | 4 5 2 | 8 6 3 |  e
                          4 6 5 | 1 3 8 | 9 7 2 |  f
                          - - - - - - - - - - -
                          5 1 4 | 8 2 7 | 6 3 9 |  g
                          3 8 6 | 5 4 9 | 1   7 |  h
                          2 7 9 | 6 1 3 | 4 5 8 |  i


Input column and row you want to edit.
Input S to save progress and exit, X to give up.
```

Give up and show solution.

```
  A B C   D E F   G H I
- - - - - - - - - - - -
      |   2   |       |  a
5   8 |   3   |       |  b
  2   |     7 |       |  c
- - - - - - - - - - - -
6     | 2     | 9     |  d
      |   8   |       |  e
  5 2 |       |       |  f
- - - - - - - - - - - -
      | 4     |   2 6 |  g
  6   | 3   5 |   4   |  h
      |       | 1   3 |  i


Input column and row you want to edit.
Input S to save progress and exit, X to give up.
X
Solution:
  A B C   D E F   G H I
- - - - - - - - - - - -
3 9 7 | 5 2 6 | 4 8 1 |  a
5 4 8 | 9 3 1 | 7 6 2 |  b
1 2 6 | 8 4 7 | 3 5 9 |  c
- - - - - - - - - - - -
6 7 3 | 2 5 4 | 9 1 8 |  d
9 1 4 | 6 8 3 | 2 7 5 |  e
8 5 2 | 1 7 9 | 6 3 4 |  f
- - - - - - - - - - - -
7 3 1 | 4 9 8 | 5 2 6 |  g
2 6 9 | 3 1 5 | 8 4 7 |  h
4 8 5 | 7 6 2 | 1 9 3 |  i

Play again!
```

Enter your name to the Hall of Fame

```
Input S to save progress and exit, X to give up.
Hh 2
Input number to fill this cell. (1-9)
  A B C   D E F   G H I
- - - - - - - - - - - -
6 2 1 | 3 8 4 | 7 9 5 |  a
7 5 3 | 9 6 1 | 2 8 4 |  b
9 4 8 | 2 7 5 | 3 1 6 |  c
- - - - - - - - - - - -
8 3 2 | 7 9 6 | 5 4 1 |  d
1 9 7 | 4 5 2 | 8 6 3 |  e
4 6 5 | 1 3 8 | 9 7 2 |  f
- - - - - - - - - - - -
5 1 4 | 8 2 7 | 6 3 9 |  g
3 8 6 | 5 4 9 | 1 2 7 |  h
2 7 9 | 6 1 3 | 4 5 8 |  i


Victory!
Enter your name to the hall of fame.
Omar
Hall of Fame:
Name--Total Puzzles Completed
Omar--15
Teddy--8
Smith--7
John--5
John--5
Malik--2
Adams--2
c--1
Leste--1
Alex--1
Lester--1
Quincy--1
```

4x4 Puzzle

```
  A B   C D
- -   - -
      |     |  a
      |   4 |  b
- -   - -
1     | 2 3 |  c
3 2 |       |  d


Input column and row you want to edit.
Input X to give up.
```

3.2 | Pseudocode

Main

Input P to generate random puzzle and play, L to load saved puzzle, or B to display scores.

If P inputted,

Ask user to enter grid type (4x4 or 9x9)

If 4x4,

Set is9x9 to false.

If 9x9,

Set is9x9 to true.

Call Randomize Puzzle function to randomize puzzle array.

Copy contents of puzzle array to another array—(puzzle) complete array.

Call Difficulty Select function.

Empty cells according to difficulty.

Copy contents of puzzle array (now with empty spots) to defaults array.

Set isPlayable flag to true to escape main menu and go into the game.

If L inputted,

Call Load Puzzle function to do necessary checks and load from file.

If no file was loaded,

Set isPlayable flag to false.

Else

Set isPlayable to true.

If B inputted,

Call Leaderboard function to display names of other players & scores.

While isPlayable is false. If false, prompt user to menu.

Call Display Puzzle Function to display grid.

Input A-I, a-i to select column and row. Input S to save puzzle and exit. Input X to give up and show completed grid.

If S inputted,

Call Save Puzzle function to save progress and quit.

If X inputted,

Set isDone to true;

Set isSolvd to false;

Call Display Puzzle function and display (puzzle) completed array.

If A-I, a-I inputted,

Call Check Input function.

Check if array is filled.

If array is not filled,

Set isDone to false.

If array is complete,

Set isSolvd to true.

While isDone is false.

If isSolvd is true,

Output “Victory!”
Call Add Record function to prompt user to add their name to the hall of fame.
End Program

Display Puzzle
Output values from passed in array until all is displayed.
Compare first array with second array.
If value of first array is equal to second array,
Make bold the next value.

Random Puzzle
Empty array passed in.
For each value in array,
Set counter to 0.
Call Get Random Number function to return a number within the rules of Sudoku.
While counter is greater than 30 (cannot be completed by program.)

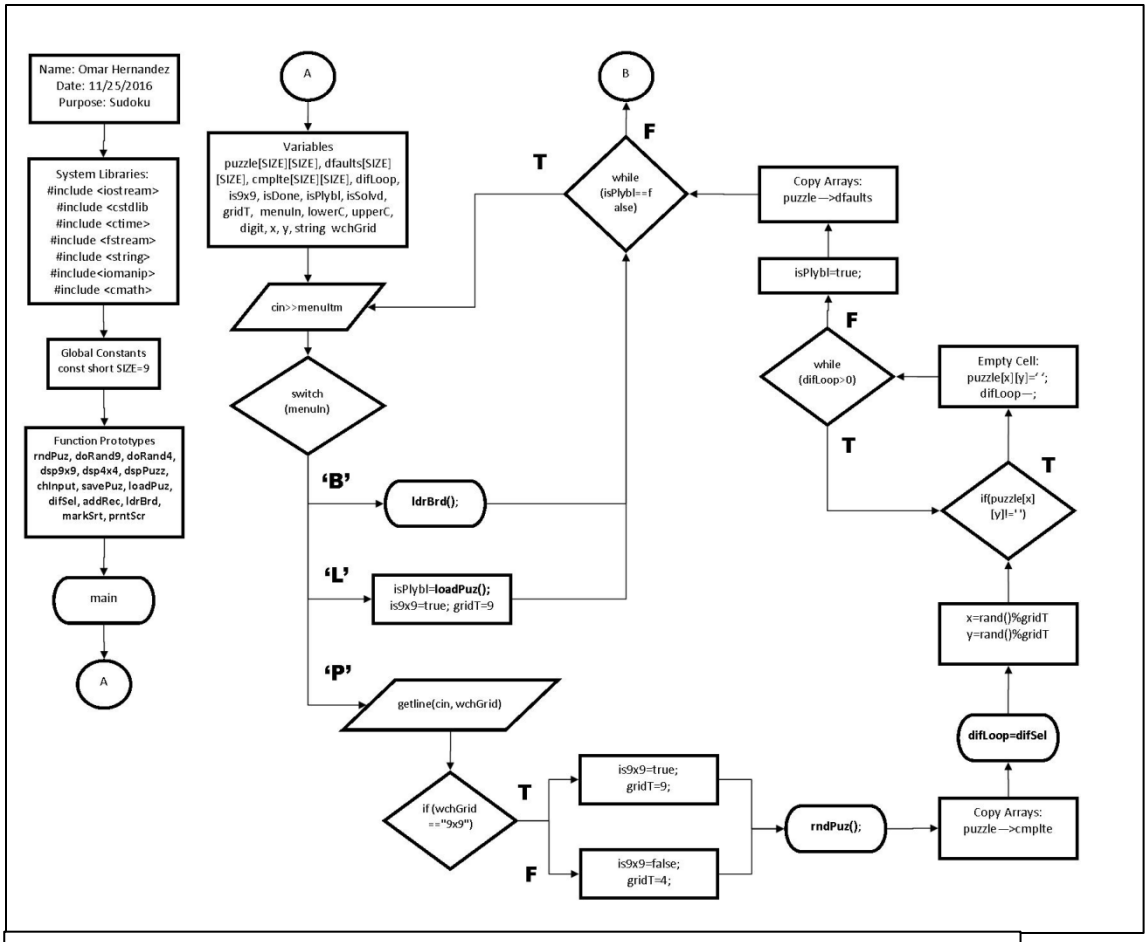
Get Random Number
Generate random number from 1-9 or 1-4 depending on is9x9 flag.
If value exists in the same column,
Set isRepetition to true.
If value exists in the same row,
Set IsRepetition to true.
If value exists in the same block,
Set isRepetition to true.
Increment counter.
While isRepetition is set to true & counter<=30.
Return generated number.

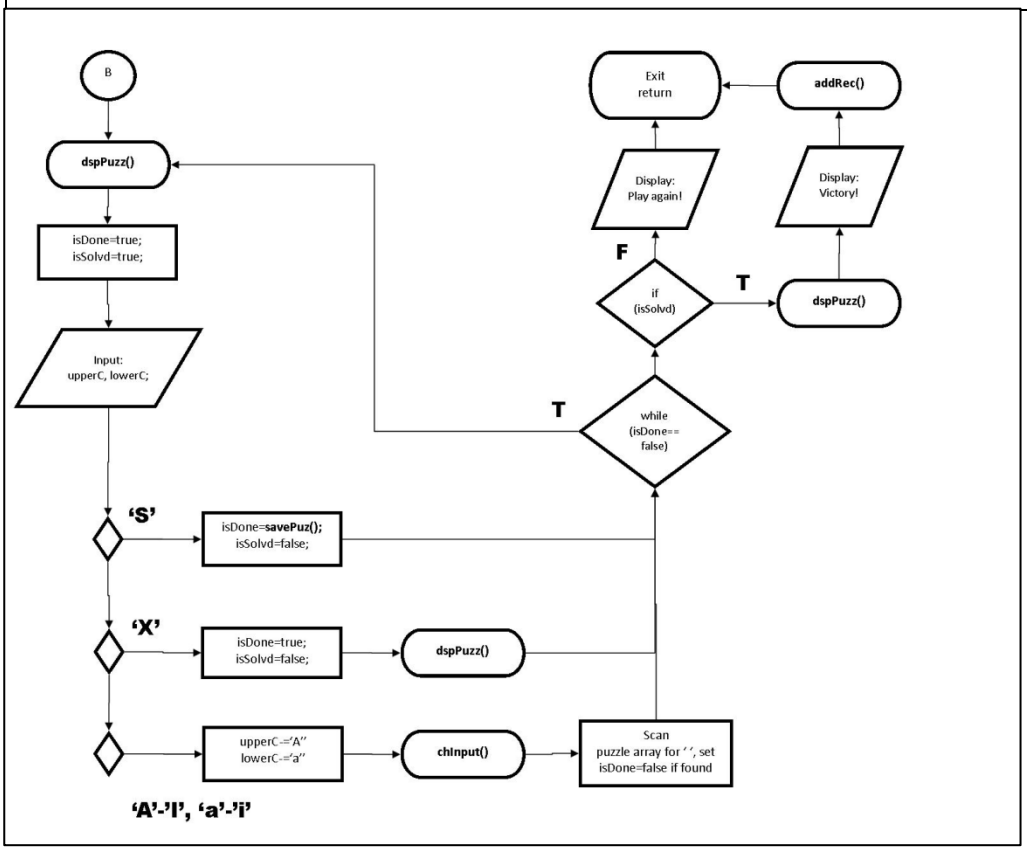
Check Input
If value user wants to change was set by the program,
Return.
Input number to add to cell selected.
If value exists in the same row,
Return.
If value exists in the same column,
Return.
If value exists in the same square,
Return.
Set cell to inputted number.

Difficulty Select
Input difficulty level from 1-5 to play.
Return difficulty selected.

3.3 Flowcharts

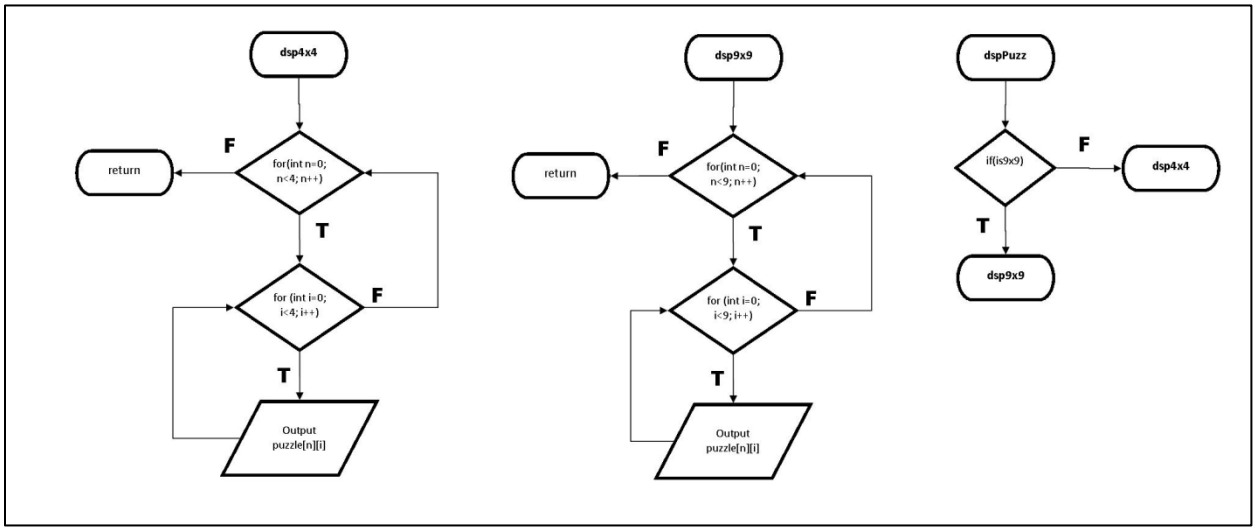
Main



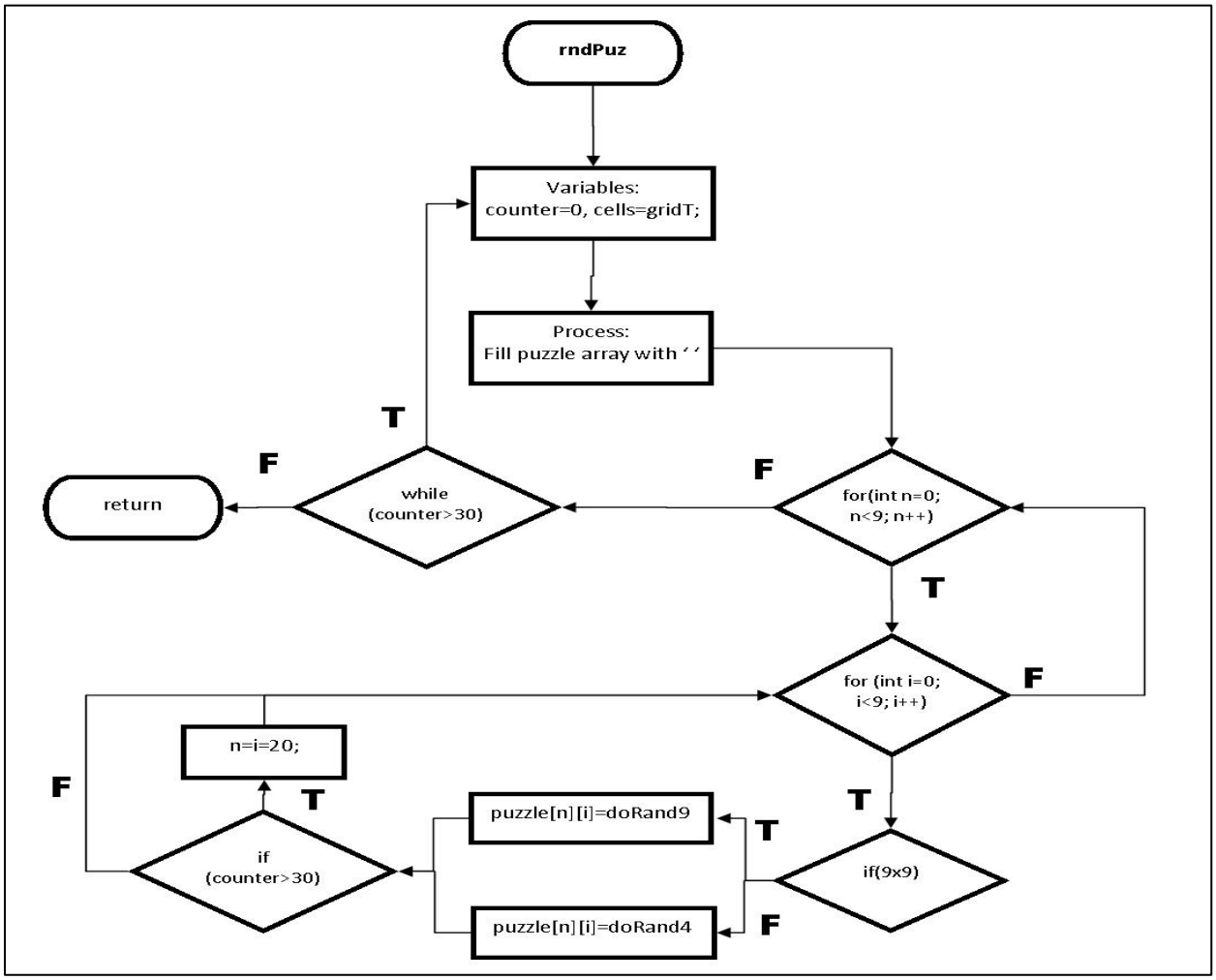


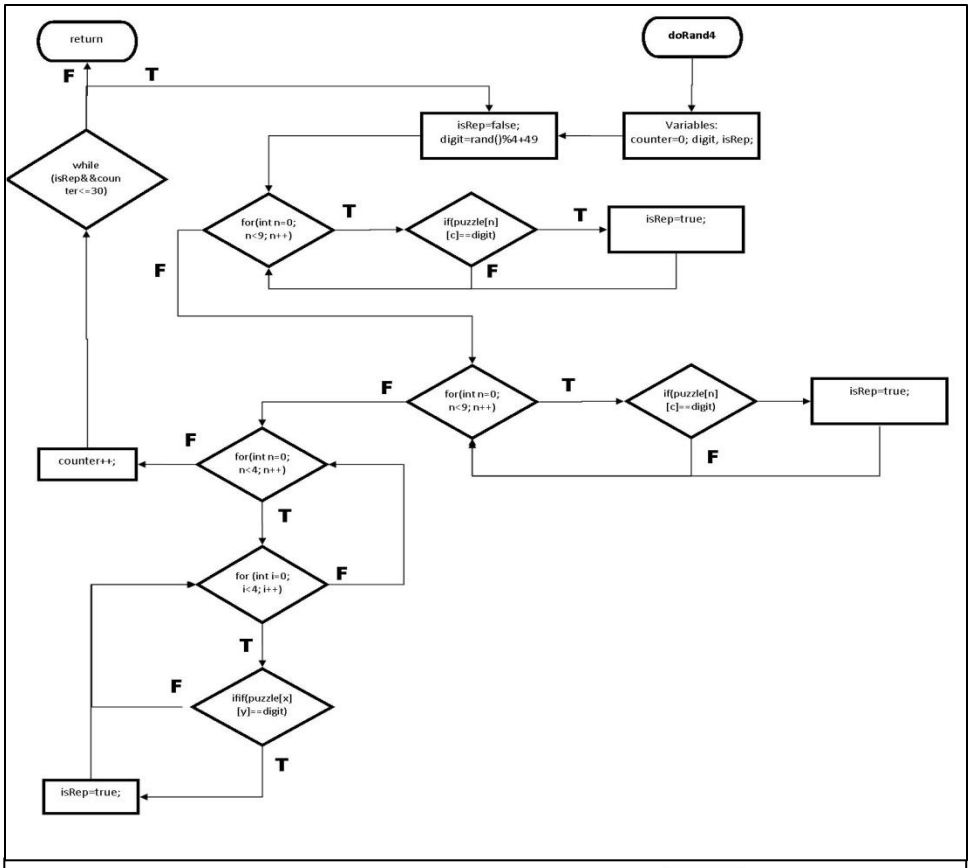
Functions

Display Array

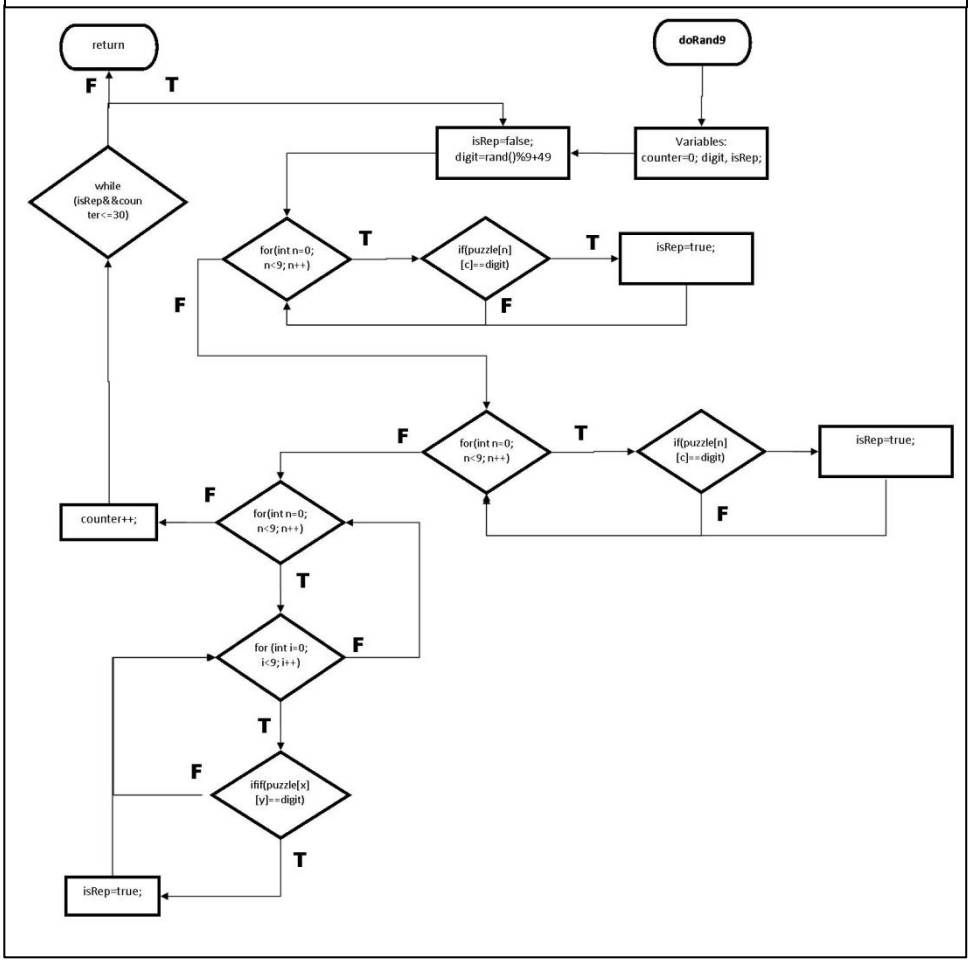


Generate Random Puzzle



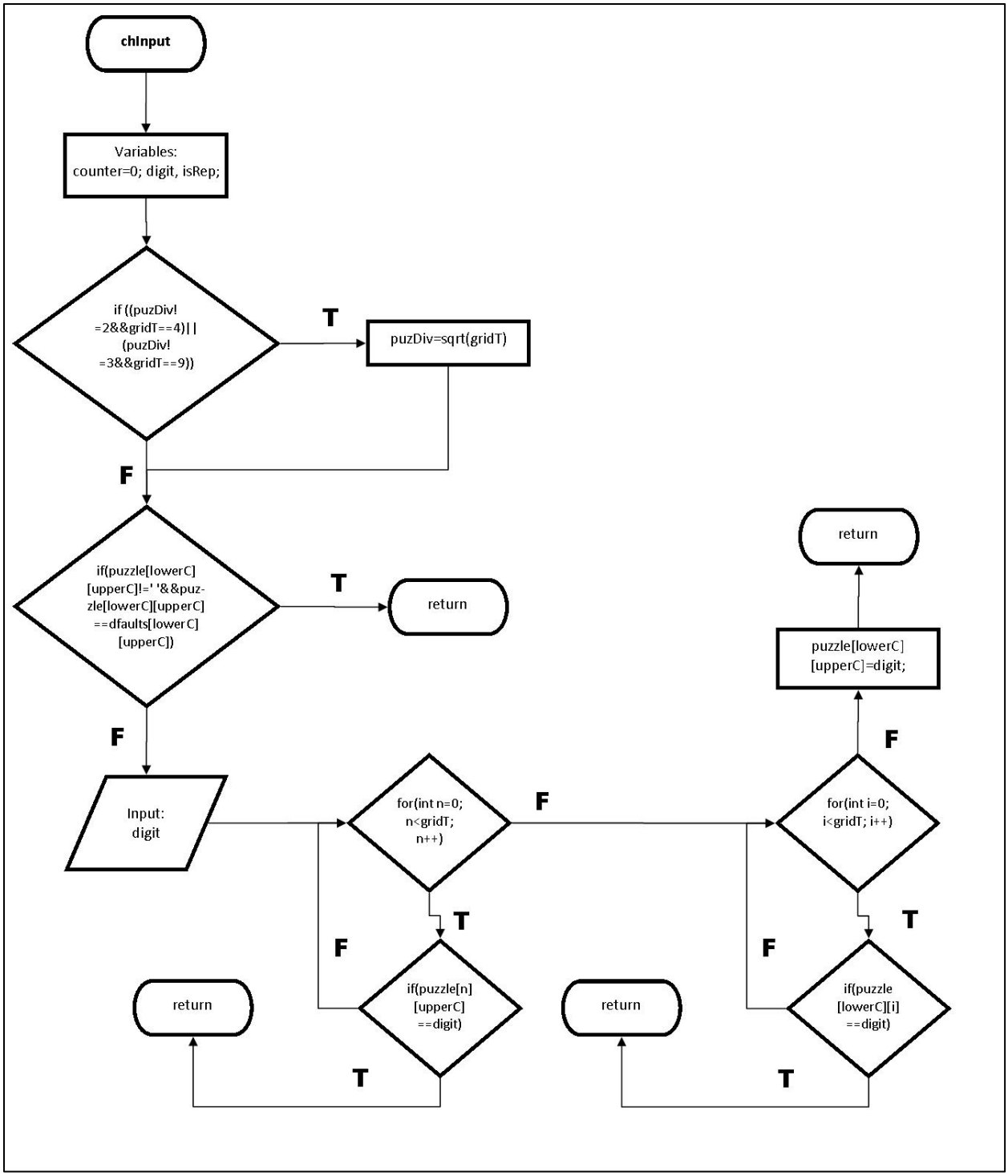


Fill 4x4 Grid With Random Numbers



Fill 9x9 Grid With Random Numbers

Check Input for Grid
(input from player)



3.4 || Code

```
1  /*
2     File:   main
3     Author: Omar Hernandez
4     Created on November 28, 2016, 10:00 PM
5     Purpose:  Sudoku
6   */
7
8  //System Libraries
9  #include <iostream>   //Input/Output objects
10 #include <cstdlib>    //Random number generator
11 #include <ctime>      //rand
12 #include <fstream>    //File I/O
13 #include <string>     //String Library
14 #include <iomanip>    //formatting
15 #include <cmath>      //square root function
16 #include <vector>    //vectors for rankings
17 using namespace std;  //Name-space used in the System Library
18
19 //User Libraries
20 //#include "Victors.h"
21
22 //Global Constants
23 short const SIZE=9;     //2 dimensional array. As it is Sudoku, same amount of rows and columns
24
25 //Function prototypes
26 void rndPuz(char [][SIZE], bool);
27 char doRand9(char [][SIZE], int, int, short &);
28 char doRand4(char [][SIZE], int, int, short &);
29 void dsp9x9(char [][SIZE], char [][SIZE]);
30 void dsp4x4(char [][SIZE], char [][SIZE]);
31 void dspPuzz(char [][SIZE], bool, char[][SIZE]);
32 void chInput(char [][SIZE], char [][SIZE], char, char, short);
33 bool isSaved(char [][SIZE], char [][SIZE], char [][SIZE], bool=true);
34 bool isLoadd(char [][SIZE], char [][SIZE], char [][SIZE], bool *);
35 short difSel(bool);
36 void addRec();
37 void ldrBrd();
38 void markSrt(vector<int> &,vector<int> &,int);
39 void prntScr(vector <int> &, vector<string> &,vector <int> &,int);
40 void escape();
41
```

```
//Execution Begins Here!
int main(int argc, char** argv) {
    //set random number seed
    srand(static_cast<int>(time(0)));
    //declare variables
    char    puzzle[SIZE][SIZE];
    char    dfaults[SIZE][SIZE];   //values that have been left by randomization; parallel array
    char    cmplte[SIZE][SIZE];    //store completed puzzle for reference
    short   difLoop;      //difficulty
    bool    is9x9, isDone, isPlybl, isSolvd; //is 9x9 or 4x4, is done with the puzzle, is playable,
is solved
    short   gridT;      //4x4 or 9x9
    char    menuIn;     //menu input
    char    lowerC, upperC, digit;    //number submit
    char    x, y;       //used in algorithm that determines which cells to edit
    string  wchGrid;    //difficulty select, which grid (4x4 or 9x9)

    //intro
    //fill intro with ? for intro
    for(int n=0; n<SIZE; n++){
        for(int i=0; i<SIZE; i++){
            puzzle[n][i]='?';
        }
    }

    cout<<"Sudoku!"<<endl;
    dspPuzz(puzzle, is9x9, dfaults);    //display puzzle with question marks for intro

    do{
        cout<<"Input P to create puzzle and play, L to load saved"<<endl
            <<"puzzle, B to display leaderboard, or E to exit."<<endl;
        cin>>menuIn;

        switch(menuIn){
            case'P':{
                //puzzle preference selection
                //choose 4x4 or 9x9 by inputting a string
                cout<<"Play 4x4 or 9x9?"<<endl;
                do{
                    cin.clear();
                    getline(cin, wchGrid);
                }while(wchGrid!="4x4"&&wchGrid!="9x9");

```

```
84                  //if input matches "9x9", set grid variables to suit 9x9
85                  if (wchGrid=="9x9"){
86                      is9x9=true;
87                      gridT=9;
88                  }
89                  //else "4x4", set grid variables to suit 4x4
90                  else {
91                      is9x9=false;
92                      gridT=4;
93                  }
94
95                  rndPuz(puzzle, is9x9);
96                  //store array to complete to give solution to player's that concede
97                  for(int n=0; n<SIZE; n++){
98                      for(int i=0; i<SIZE; i++){
99                          cmplte[n][i]=puzzle[n][i];
100                     }
101                 }
102                 difLoop=difSel(is9x9);//difficulty select do-while loop
103
104                 //determine which cells to empty
105                 do{
106                     //x -> row, y -> column
107                     x=rand()%(gridT);
108                     y=rand()%(gridT);
109
110                     //if cells are already empty, do loop again
111                     if(puzzle[x][y]!=' '){
112                         puzzle[x][y]=' ';
113                         difLoop--;
114                     }
115                 }while(difLoop>0);
116
117                 //store array as is to be able to distinguish between values added by the
118                 //player and the values generated by the program
119                 for(int n=0; n<SIZE; n++){
120                     for(int i=0; i<SIZE; i++){
121                         dfaults[n][i]=puzzle[n][i];
122                     }
123                 }
124                 isPlybl=true;
125                 break;
126             }
```

```
127             case 'B':ldrBrd(); break;
128             case 'L': {
129                 isPlybl=isLoadd(puzzle, dfaults, cmplte, &isPlybl);
130                 //set grid type
131                 is9x9=true;
132                 gridT=9;
133                 break;
134             }
135             case 'E':{
136                 escape();
137             }
138         }
139     }while(isPlybl==false);
140
141
142
143     //now play the game
144     do{
145         dspPuzz(puzzle, is9x9, dfaults);
146         isSolvd=false;
147         isDone=false;        //escape from the loop
148         cout<<endl;
149         cout<<"Input column and row you want to edit."<<endl
150             <<"Input"<<((is9x9)?" S to save progress and exit, ":" ")<<"X to give up."<<endl;
151         cin.clear();
152         cin>>upperC;
153         if(upperC!='X'&&upperC!='S')cin>>lowerC;
154
155         //save puzzle and exit
156         if(upperC=='S'||lowerC=='S'){
157             isDone=isSaved(puzzle, dfaults, cmplte, is9x9);
158             isSolvd=false;
159         }
160
161         //exit puzzle and give solution
162         else if(upperC=='X'||lowerC=='X'){
163                 isDone=true;
164                 isSolvd=false;
165                 cout<<"Solution: "<<endl;
166                 dspPuzz(cmplte, is9x9, cmplte);
167         }
168
169         else if ((upperC>='A'&&upperC<='I')&&(lowerC>='a'&&lowerC<='i')){
```

```
            //set column and row input appropriate value for loop
            upperC-='A';
            lowerC-='a';

            chInput(puzzle, dfaults, upperC, lowerC, gridT);
            isSolvd=true;

            isDone=true;
            for(int n=0; n<gridT; n++){
                for(int i=0; i<gridT; i++){
                    if(puzzle[n][i]==' ')isDone=false;
                }
            }
        }
    }while(isDone==false);

    if(isSolvd)dspPuzz(puzzle, is9x9, dfaults);
    cout<<((isSolvd)?"Victory!":"Play again!")<<endl;

    if(isSolvd&&is9x9)addRec();

    //Exit Program
    return 0;
}

//****************************** Leaderboard *********************************
//Purpose: read in names and scores of previous players and display
//Inputs:   Inputs to the function here -> Description, Range, Units
//  string and score vectors to allow for storage of contents in file
//Output:   Outputs to the function here -> Description, Range, Units
//  display names and scores, sorted
//****************************************************************************
void ldrBrd(){
    //read names from file
    ifstream in;
    in.open("names.dat");
    string data;
    //Victors victor;

    vector<string> names;
    vector<int>    scores;


```

```
213     if(in){
214         while(in>>data) {
215             names.push_back(data);
216         }
217     }
218     in.close();
219 
220     //display score
221     in.open("score.dat");
222     int points;
223     if(in){
224         while(in>>points){
225             scores.push_back(points);
226         }
227     }
228 
229     //create index to facilitate sorting
230     vector <int> index(names.size());
231     for(int i=0;i<scores.size();i++){
232         index[i]=i;
233     }
234 
235     //sort
236     markSrt(scores, index, names.size());
237 
238     //print sorted scores
239     prntScr(scores, names, index, names.size());
240 }
241 
242 //****************************** Add Record **********************************
243 //Purpose: add record or increment if it is a new entry, store to file
244 //Inputs:   Inputs to the function here -> Description, Range, Units
245 //  total, name, isIn
246 //Output:   Outputs to the function here -> Description, Range, Units
247 //  store new entry to file
248 //****************************************************************************
249 void addRec(){
250     int total=0;
251     string name;
252     bool isIn;  //vector push flag
253 
254     cout<<"Enter your name to the hall of fame."<<endl;
255     do{
```

```
256         getline(cin, name);
257     }while(name.empty());  //check if input is empty
258 
259     vector<string> names;
260     vector<int>    scores;
261 
262     //read names from file
263     ifstream in;
264     in.open("names.dat");
265     string data;
266     if(in){
267         while(in>>data) {
268             names.push_back(data);
269         }
270     }
271     in.close();
272 
273     //display score
274     in.open("score.dat");
275     int points;
276     if(in){
277         while(in>>points){
278             scores.push_back(points);
279         }
280     }
281 
282     //search vector to find if name is already registered
283     for(int i=0;i<names.size(); i++){
284         if(names[i]==name){
285             isIn=true;
286             scores[i]++;
287         }
288     }
289 
290     //push.back or add new entry
291     if(!isIn){
292         total++;
293         names.push_back(name);
294         scores.push_back(total);
295     }
296 
297     //set index 0 - size of score vector
298     vector <int> index(names.size());
```

```
    for(int i=0;i<scores.size();i++){
        index[i]=i;
    }

    markSrt(scores, index, names.size());

    prntScr(scores, names, index, names.size());

    //store names into file
    ofstream out;
    out.open("names.dat");
    for(int i=0;i<names.size(); i++){
        out<<names[i]<<endl;
    }
    out.close();
    //store scores into file
    out.open("score.dat");
    for(int i=0;i<scores.size(); i++){
        out<<scores[i]<<endl;
    }
    out.close();
}

//****************************** Print Score *********************************
//Purpose: print scores from greatest to least
//Inputs:   Inputs to the function here -> Description, Range, Units
//  names vector, scores vector, and size of vectors
//Output:   Outputs to the function here -> Description, Range, Units
//  display scores
//*****************************************************************************
void prntScr(vector <int> &a, vector<string> &names,vector <int> &indx,int SIZE){
    //from greatest to smallest
    cout<<"Hall of Fame:\nName--Total Puzzles Completed"<<endl;
    for(int i=0;i<SIZE;i++){
        cout<<names[indx[(SIZE-1-i)]]<<"--";
        cout<<a[indx[(SIZE-1-i)]]<<endl;
    }
    cout<<endl;
}

//****************************** Mark Sort ***********************************
//Purpose: sort indexes that lets us sort from greatest to least
//Inputs:   Inputs to the function here -> Description, Range, Units
```

```
342 //  score and index vectors -- the variables to account for storage
343 //Output:   Outputs to the function here -> Description, Range, Units
344 //  return indexes sorted
345 //*****************************************************************************
346 void markSrt(vector <int> &a,vector <int> &indx,int SIZE){
347     for(int i=0;i<SIZE-1;i++){
348         for(int j=i+1;j<SIZE;j++){
349             if(a[indx[i]]>a[indx[j]]){
350                 indx[i]=indx[i]^indx[j];
351                 indx[j]=indx[i]^indx[j];
352                 indx[i]=indx[i]^indx[j];
353             }
354         }
355     }
356 }
357 
358 //****************************** Load Puzzle *********************************
359 //Purpose: load puzzle
360 //Inputs:   Inputs to the function here -> Description, Range, Units
361 //  puzzle, dfaults, cmplte -> puzzle and boundaries
362 //Output:   Outputs to the function here -> Description, Range, Units
363 //  array filled with contents from save file
364 //*****************************************************************************
365 bool isLoadd(char puzzle [SIZE][SIZE], char dfaults [SIZE][SIZE], char cmplte [SIZE][SIZE], bool
    *isPlybl){
366     ifstream inFile;        //load from file
367     inFile.open("prev.dat");
368 
369     if(inFile){
370         inFile.unsetf(ios_base::skipws);    //skip white space
371         //inFile>>is9x9;
372         *isPlybl=true;
373 
374         for(int n=0; n<SIZE; n++){
375             for(int i=0; i<SIZE; i++){
376                 inFile>>puzzle[n][i];
377             }
378         }
379 
380 
381         for(int n=0; n<SIZE; n++){
382             for(int i=0; i<SIZE; i++){
383                 inFile>>dfaults[n][i];
```

```
            }
        }

        for(int n=0; n<SIZE; n++){
            for(int i=0; i<SIZE; i++){
                inFile>>cmplte[n][i];
            }
        }

        inFile.close();
    }
    else {
        //if no load puzzle, set menu=x so it continues the loop
        cout<<endl<<"No saved puzzle."<<endl;
        *isPlybl=false;
    }
}

//****************************** Save Puzzle *********************************
//Purpose: save Puzzle
//Inputs:   Inputs to the function here -> Description, Range, Units
//  puzzle, defaults, complete -> progress, values set by program, and completed
//Output:   Outputs to the function here -> Description, Range, Units
//  store arrays to file
//*****************************************************************************
bool isSaved(char puzzle [][SIZE], char dfaults [][SIZE], char cmplte [][SIZE], bool is9x9){
    if(!is9x9) return false;

    ofstream out;
    out.open("prev.dat");

    //out<<is9x9;

    for(int n=0; n<SIZE; n++){
        for(int i=0; i<SIZE; i++){
            out<<puzzle[n][i];
        }
    }

    for(int n=0; n<SIZE; n++){
        for(int i=0; i<SIZE; i++){
            out<<dfaults[n][i];
        }
```

```
    }

    for(int n=0; n<SIZE; n++){
        for(int i=0; i<SIZE; i++){
            out<<cmplte[n][i];
        }
    }

    out.close();
    cout<<"Saved!"<<endl;
    return true;
}

//****************************** Difficulty Select **************************
//Purpose: allow difficulty selection
//Inputs:   Inputs to the function here -> Description, Range, Units
//  is9x9 -> if true, erase more values
//Output:   Outputs to the function here -> Description, Range, Units
//  return number of cells to empty
//*****************************************************************************
short difSel(bool is9x9){
    short dif;

    //difficulty select do-while loop
    do{
        cout<<"Select a difficulty (1-5), 1 being the easiest, 5 being the hardest."<<endl;
        cin>>dif;
    }while(dif<=0&&dif>5);

    //empty cells depending on difficulty selected
    if (is9x9) return dif*12;
    else return dif*2;
}

//****************************** Check Sudoku Input **************************
//Purpose: check Sudoku input
//Inputs:   Inputs to the function here -> Description, Range, Units
//  puzzle, dfaults, upperC, lowerC, gridT -> validate entry
//Output:   Outputs to the function here -> Description, Range, Units
//  if value is in accordance with Sudoku rules, change cell in puzzle array
//*****************************************************************************
void chInput(char puzzle[][SIZE], char dfaults[][SIZE], char upperC, char lowerC, short gridT){
    //get the square root of the grid type for use in algorithm that scans square
```

```
    //if 4x4, each square has two columns and rows
    //if 9x9, each square has three columns and rows
    char digit;
    static short puzDiv;


    if ((puzDiv!=2&&gridT==4)||(puzDiv!=3&&gridT==9)) puzDiv=sqrt(gridT);

    //check if cell is empty or if it is a value placed by the computer
    if(puzzle[lowerC][upperC]!=' '&&puzzle[lowerC][upperC]==dfaults[lowerC][upperC]){
        cout<<"The cell you want to edit has a value set by the program."<<endl;
        return;
    }

    //player submits a value to be added to cell
    cout<<"Input number to fill this cell. (1-"<<gridT<<")"<<endl;
    //input validation
    do{
        cin>>digit;
    }while(digit<'1'||digit>(gridT+'0'));

    //check columns
    for(int n=0; n<gridT; n++){
        if(puzzle[n][upperC]==digit){
            cout<<"The value already exists in the same column."<<endl;
            return;
        }
    }

    //ensure no duplicates exist in row
    for(int i=0; i<gridT; i++){
        if(puzzle[lowerC][i]==digit){
            cout<<"The value already exists in the same row."<<endl;
            return;
        }
    }

    //check if there are any of the same values in the square
    //x -> beginning row of square
    //y -> beginning column of square
    //n & i for the loop
    for(int x=lowerC-lowerC%puzDiv, n=0; n<puzDiv; x++, n++){
        for(int y=upperC-upperC%puzDiv, i=0; i<puzDiv; y++, i++){
```

```
513             if(puzzle[x][y]==digit) {
514             cout<<"The value already exists in the same square."<<endl;
515             return;
516             }
517         }
518     }
519 
520     puzzle[lowerC][upperC]=digit;
521 }
522 
523 //****************************** Randomize Puzzle 9x9 ************************
524 //Purpose: fill 9x9 array with random numbers within Sudoku boundaries
525 //Inputs:   Inputs to the function here -> Description, Range, Units
526 //  puzzke, r, c, counter, isRep, digit -> puzzle and boundaries
527 //Output:   Outputs to the function here -> Description, Range, Units
528 //  cell with random
529 //*****************************************************************************
530 char doRand9(char puzzle[][SIZE], int r,int c, short & counter){
531     bool isRep;
532     char digit;
533 
534     counter=0;
535 
536     do{
537         isRep=false;
538         digit=rand()%9+49;  //numbers 1-9u
539 
540         //ensure no duplicates exist in column
541         for(int n=0; n<SIZE; n++){
542             if(puzzle[n][c]==digit) isRep=true;
543         }
544 
545         //ensure no duplicates exist in row
546         for(int i=0; i<SIZE; i++){
547             if(puzzle[r][i]==digit) isRep=true;
548         }
549 
550         //check if there are any of the same values in the square
551         //x -> beginning row of square
552         //y -> beginning column of square
553         //n & i for the loop
554         for(int x=r-r%3, n=0; n<3; x++, n++){
555             for(int y=c-c%3, i=0; i<3; y++, i++){
```

```
                if(puzzle[x][y]==digit) isRep=true;
            }
        }
        if(isRep) counter++;
    }while(isRep && counter<=30);

    //dspPuzz(puzzle, true);    //display puzzle with random numbers

    return digit;
}

//****************************** Randomize Puzzle 4x4 ************************
//Purpose: fill 4x4 array with random numbers within Sudoku boundaries
//Inputs:   Inputs to the function here -> Description, Range, Units
//  puzzke, r, c, counter, isRep, digit -> puzzle and boundaries
//Output:   Outputs to the function here -> Description, Range, Units
//  cell with random
//*****************************************************************************
char doRand4(char puzzle[][SIZE], int r,int c, short & counter){
    bool isRep;
    char digit;

    counter=0;

    do{
        isRep=false;
        digit=rand()%4+49;  //numbers 1-9u

        //ensure no duplicates exist in column
        for(int n=0; n<4; n++){
            if(puzzle[n][c]==digit) isRep=true;
        }

        //ensure no duplicates exist in row
        for(int i=0; i<4; i++){
            if(puzzle[r][i]==digit) isRep=true;
        }

        //check if there are any of the same values in the square
        //x -> beginning row of square
        //y -> beginning column of square
        //n & i for the loop
        for(int x=r-r%2, n=0; n<2; x++, n++){
```

```
599             for(int y=c-c%2, i=0; i<2; y++, i++){
600                 if(puzzle[x][y]==digit) isRep=true;
601             }
602         }
603         if(isRep) counter++;
604     }while(isRep && counter<=30);
605 
606     //dspPuzz(puzzle, true);    //display puzzle with random numbers
607 
608     return digit;
609 }
610 
611 //****************************** Randomize Puzzle ****************************
612 //Purpose: fill Sudoku array within the boundaries of the game
613 //Inputs:   Inputs to the function here -> Description, Range, Units
614 //  puzzle, is9x9 -> array to change, if 9x9 or not
615 //Output:   Outputs to the function here -> Description, Range, Units
616 //  arrays
617 //****************************************************************************
618 void rndPuz(char puzzle[][SIZE], bool is9x9){
619     short counter;
620     short cells;
621 
622     if(is9x9) cells=9;
623     else cells=4;
624 
625     do{
626         counter=0;  //used to escape puzzles computer cannot complete
627         //fill array with spaces
628         for(int n=0; n<SIZE; n++){
629             for(int i=0; i<SIZE; i++){
630                 puzzle[n][i]=' ';
631             }
632         }
633 
634         //complete Sudoku puzzle
635         for(int n=0; n<cells; n++){
636             for (int i=0; i<cells; i++){
637                 //9x9 fill
638                 if (is9x9) puzzle[n][i]=doRand9(puzzle, n, i, counter);
639                 //4x4 fill
640                 else puzzle[n][i]=doRand4(puzzle, n, i, counter);
641                 if(counter>30) n=i=20;      //multiple assignment
```

```
642             }
643         }
644     }while(counter>30);
645 }
646 
647 //****************************** Display Puzzle ******************************
648 //Purpose: determine whether to display 4x4 or 9x9
649 //Inputs:   Inputs to the function here -> Description, Range, Units
650 //  is9x9 -> if true, display 9x9 grid, else display 4x4
651 //Output:   Outputs to the function here -> Description, Range, Units
652 //  branch to appropriate function
653 //****************************************************************************
654 void dspPuzz(char puzzle[][SIZE], bool is9x9, char dfaults[][SIZE]){
655     if (is9x9) dsp9x9(puzzle, dfaults);
656     else dsp4x4(puzzle, dfaults);
657 }
658 
659 //****************************** Display 9x9 *********************************
660 //Purpose: Display 9x9 puzzle
661 //Inputs:   Inputs to the function here -> Description, Range, Units
662 //  puzzle, rows, cols -> puzzle itself
663 //Output:   Outputs to the function here -> Description, Range, Units
664 //  display 9x9 puzzle
665 //****************************************************************************
666 void dsp9x9(char puzzle[][SIZE], char dfaults[][SIZE]){
667     char temp='a';      //display navigation character for rows
668     cout<<setw(50)<<"A B C   D E F   G H I";    //column display index
669 
670     for(int n=0; n<SIZE; n++){
671         if(n%3==0) cout<<setw(52)<<endl<<"- - - - - - - - - - - -";
672         cout<<setw(33)<<endl;
673         for (int i=0; i<SIZE; i++){
674             cout<<((puzzle[n][i]==dfaults[n][i])?"\e[1m":"\e[0m")
675                 <<puzzle[n][i]<<((i%3==2)?"\e[0m | ":" ");
676         }
677         cout<<"\e[0m "<<temp++;  //row display index
678     }
679 
680     cout<<endl<<endl;   //create two lines after display
681 }
682 
683 //****************************** Display 4x4 *********************************
684 //Purpose: Display 4x4 puzzle
```

```
685 //Inputs:   Inputs to the function here -> Description, Range, Units
686 //  puzzle, rows, cols -> puzzle itself
687 //Output:   Outputs to the function here -> Description, Range, Units
688 //  display 4x4 puzzle
689 //*******************************************************************************
690 void dsp4x4(char puzzle[][SIZE], char dfaults[][SIZE]){
691     char temp='a';      //display navigation character for rows
692     cout<<setw(38)<<"A B   C D";    //column display index
693 
694     for(int n=0; n<4; n++){
695         if(n%2==0) cout<<setw(38)<<endl<<"- -   - -";
696         cout<<setw(33)<<endl;
697         for (int i=0; i<4; i++){
698             cout<<((puzzle[n][i]==dfaults[n][i])?"\e[1m":"\e[0m")
699                 <<puzzle[n][i]<<((i%2==1)?"\e[0m | ":" ");
700         }
701         cout<<"\e[0m "<<temp++;  //row display index
702     }
703 
704     cout<<endl<<endl;   //create two lines after display
705 }
706 
707 //****************************** Escape ************************************
708 //Purpose: option to exit out of program if user decides to just display score
709 //Inputs:   Inputs to the function here -> Description, Range, Units
710 //Output:   Outputs to the function here -> Description, Range, Units
711 //*******************************************************************************
712 void escape(){
713     cout<<"Come back later!"<<endl;
714     exit(0);
715 }
```