Sudoku

CSC-5
Fall 2016

Introduction

"Sudoku", or originally called *Number Place* is a number-placement puzzle game. It can often be seen on newspapers in the comic section where the crosswords and horoscopes are located. "Sudoku" means "single number" in Japanese. The game originated from Japan. It became popular in 1986 by the Japanese puzzle company *Nikoli*. It then reached the west in 2004 from the efforts of Wayne Gould who made a computer program that produces distinct puzzles.

-https://en.wikipedia.org/wiki/Sudoku

How the Game Works

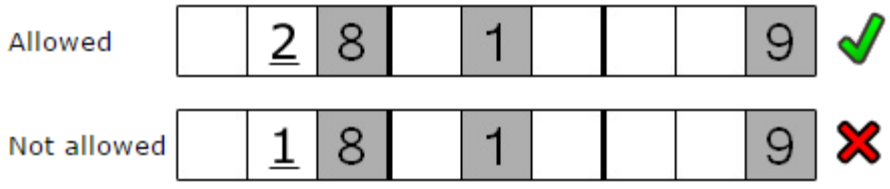
Objective of the game

The objective is to fill a grid table with numbers so that each row, column, and each 3x3 section contains all of the numbers from 1 to 9.

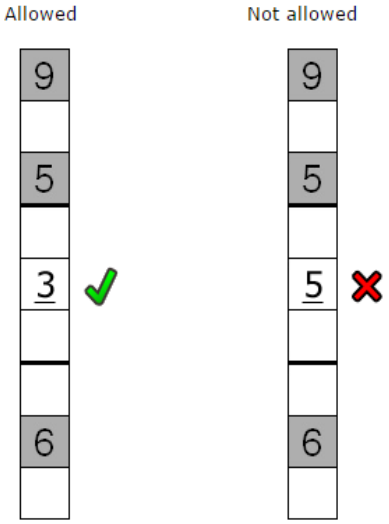
Rules of the Game

1. You can only use numbers from 1 to 9.

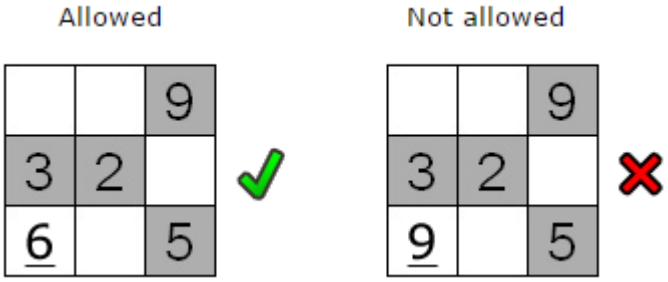
2. Number can only appear ONCE on each row:



3. Number can only appear ONCE on each column:



4. Number can only appear ONCE on each 3x3 section:



5. You CANNOT change the numbers in boxes that are already filled in when you start a game.

6. Keep filling in the empty spots until you complete the table.

-Images from http://www.sudoku.name/rules/en

How to Play

- Starting the game, you can either LOAD a game from a file or make a NEW game with random numbers.
- You can only load a game if there is a save file previously made.
- Starting a new game is always a puzzle with different numbers and combination.
- Your cursor – or where you would fill in a number – is indicated by the "#" sign.
- You can move your cursor by pressing W, A, S, D. (up, left, down, right)
- You can only move your cursor to blank spaces and those that you previously answered.
- You can make changes to your previously answered numbers anytime by moving your cursor back to the same space.
- Every time you enter a number with duplicates in the same row, column, or the 3x3 section, you will see a notification and your answer will not register on the grid.
- You can save the game anytime by pressing "E". The game will not exit.
- You can only save ONE game in ONE file. If you save a game while there is a previously saved file, you will overwrite on the file.
- You can quit or exit the game anytime by pressing "Q". If you did not save your game, you will lose the puzzle and any progress.
- Win by completing the whole table with numbers.

Programming the Game

When I first brain-stormed about how I would program this game I knew from the start that it has to revolve around arrays. At first I wrote my code using TWENTY-SEVEN different arrays. One for each rows, columns, and the nine 3x3 sections. When I got to the part where I have to write a code to check each rows, columns, and sections for duplicate numbers, I then had a thought “Would it not be easier if I made multi-dimension arrays instead of having all 27?”. I realized it would take a lot of time and space passing the arrays into each function.

I re-wrote my code from scratch. This time using only ONE single three-dimensional array. It worked great. I had less codes to write, easier functions, and less effort trying to search for duplicate numbers.

Problems with the Program

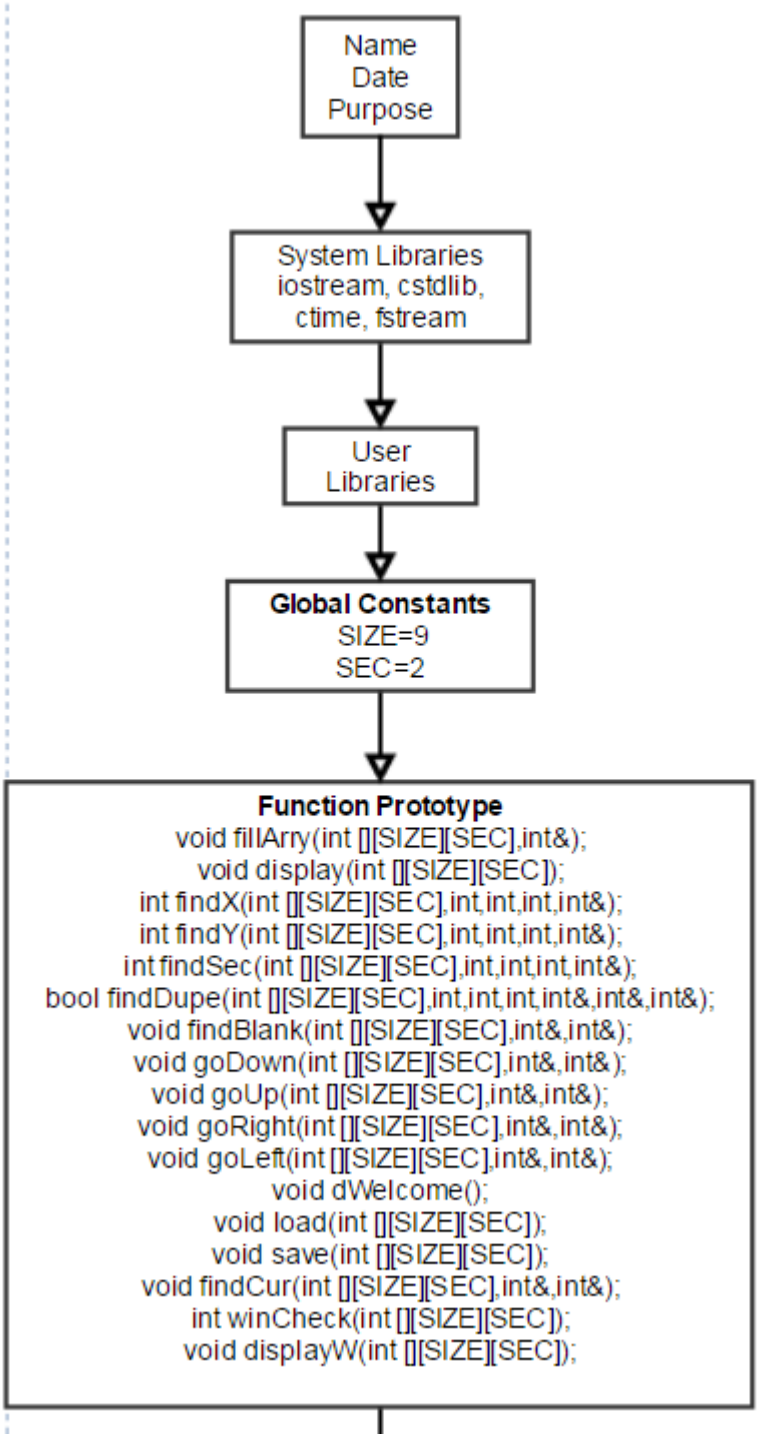
When you play the game, you CANNOT distinguish the boxes that you previously filled in with your answer from the pre-filled boxes when you started the game. Thus, it's hard to move across the grid and it is very difficult to finish the game because you cannot tell which box can be changed. I do have an idea of how I could fix this problem but it would take me a lot of time doing it and I do not have enough time.

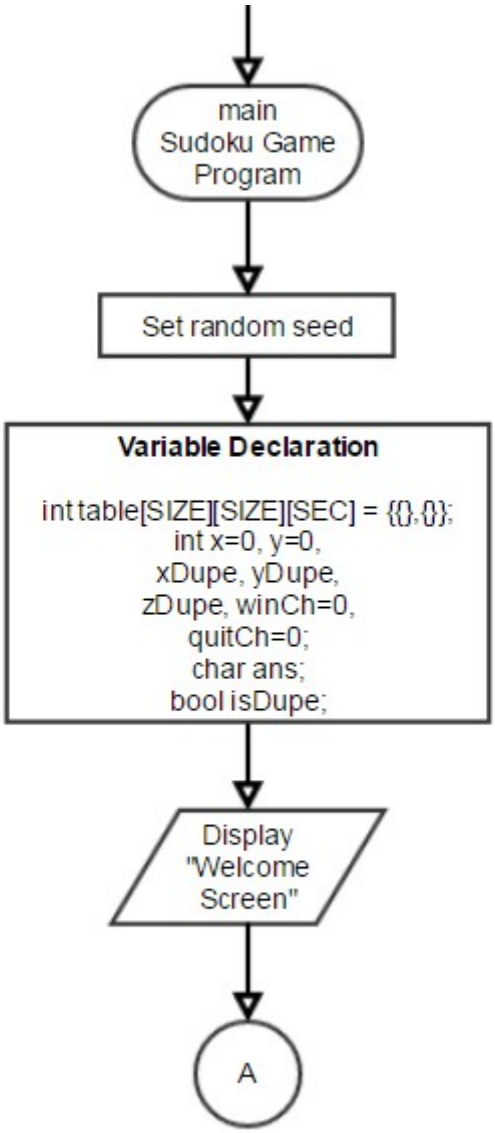
Another problem is with the lines and borders of the grid. The lines are not really connected and so it is a bit hard to see the table. It also makes it hard to see the nine 3x3 sections which are an important aspect of the game.

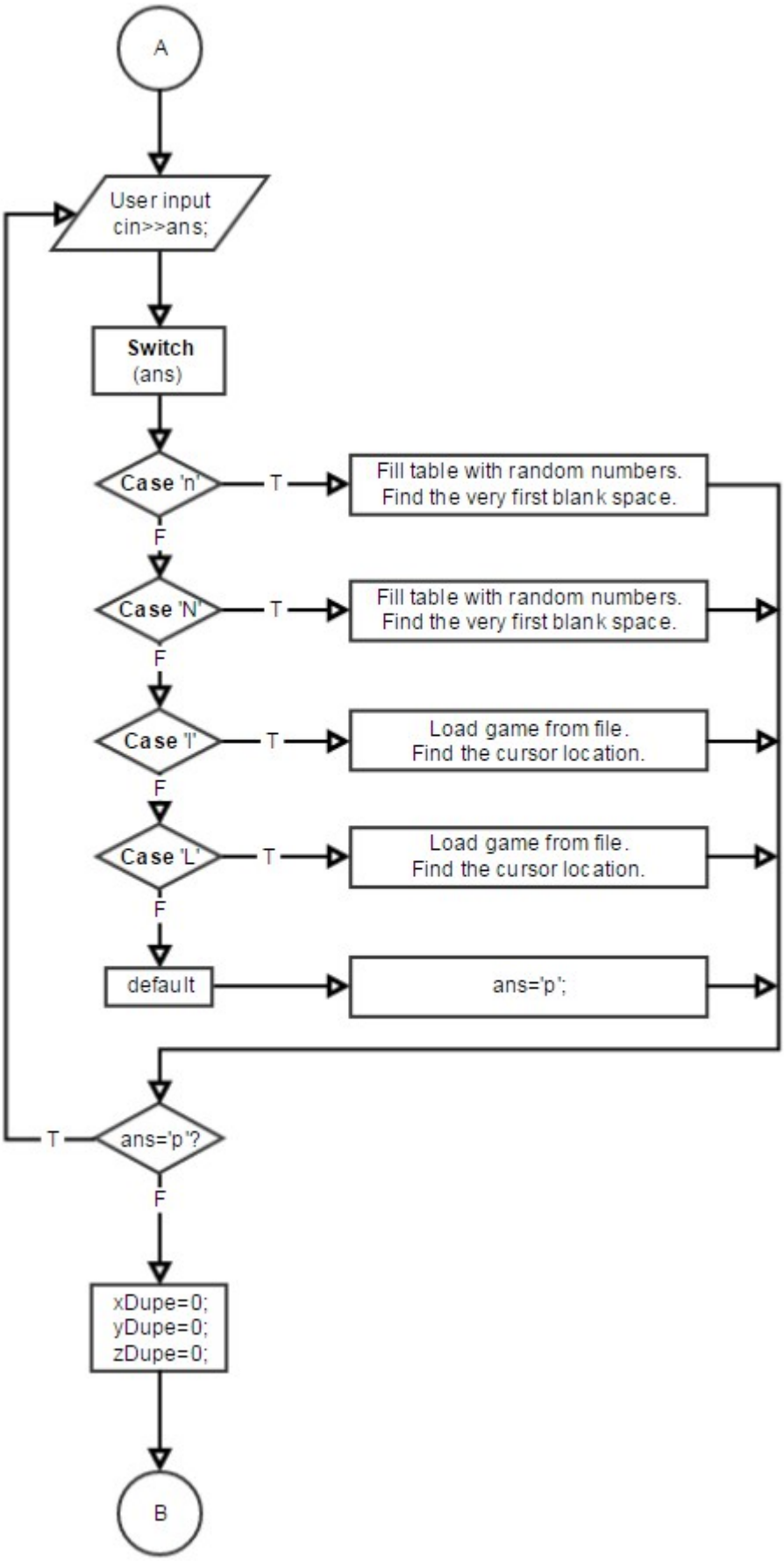
With saving the game, you can only save ONE game. I also have an idea where you can save multiple save files and naming the files by user's input but I did not have time to implement this.

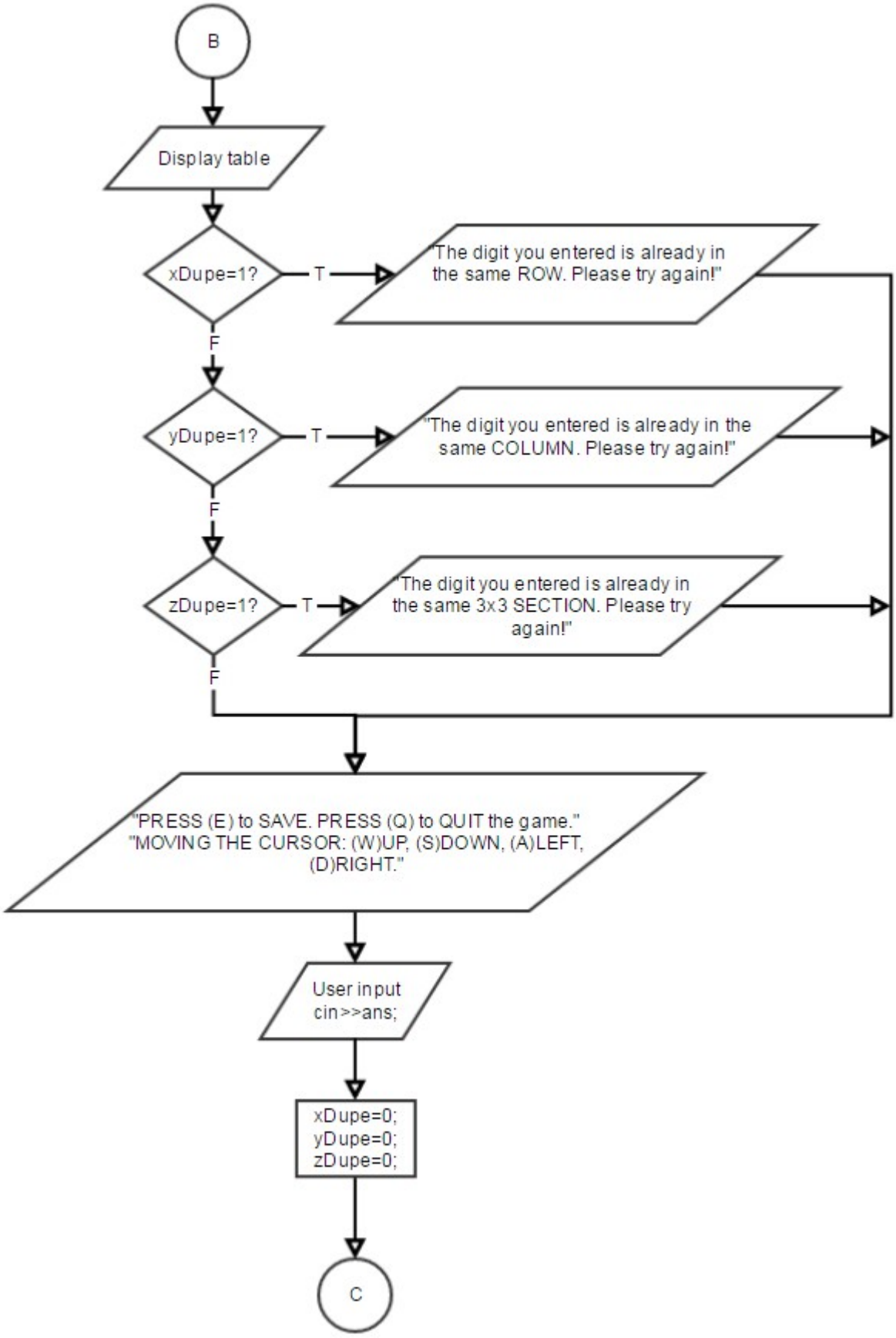
Flowchart

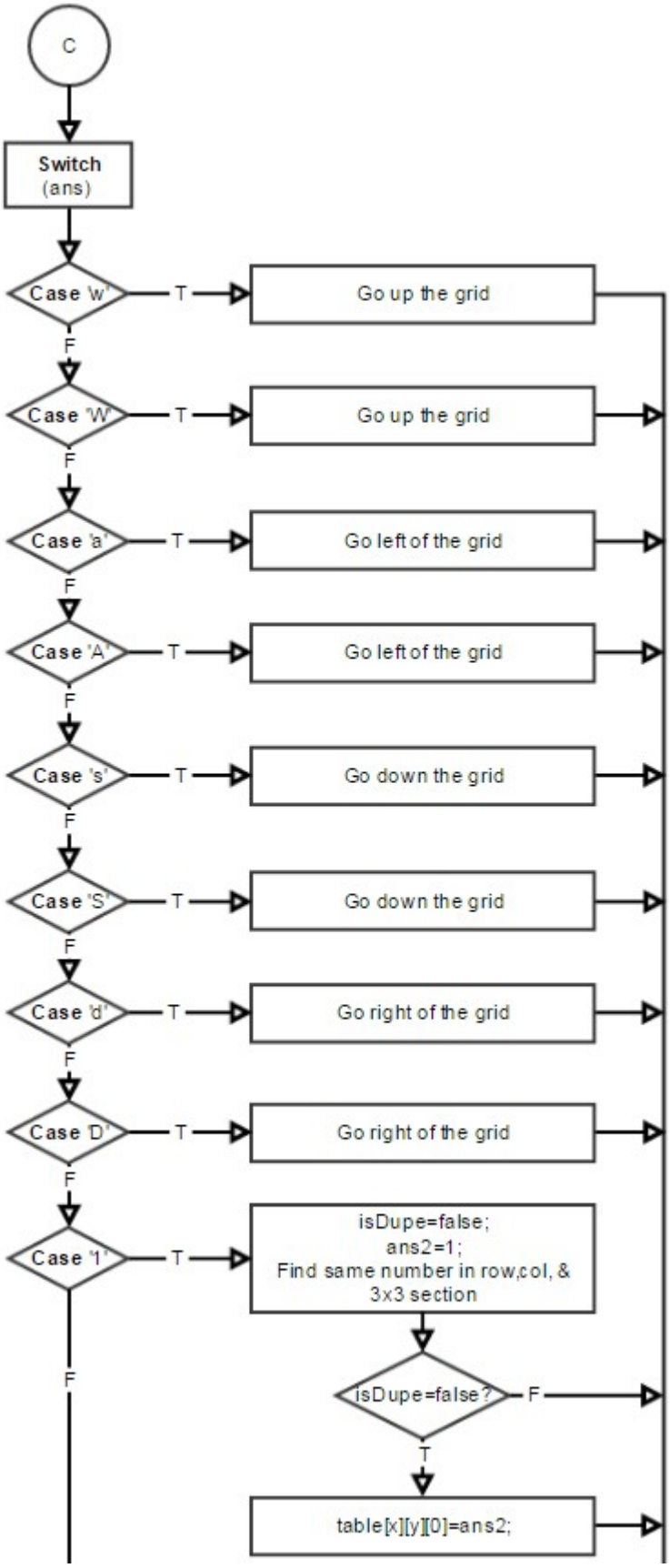
(Oversized flowchart. Broke it up into smaller parts. You can view the whole chart here: https://www.gliffy.com/go/publish/11594493)

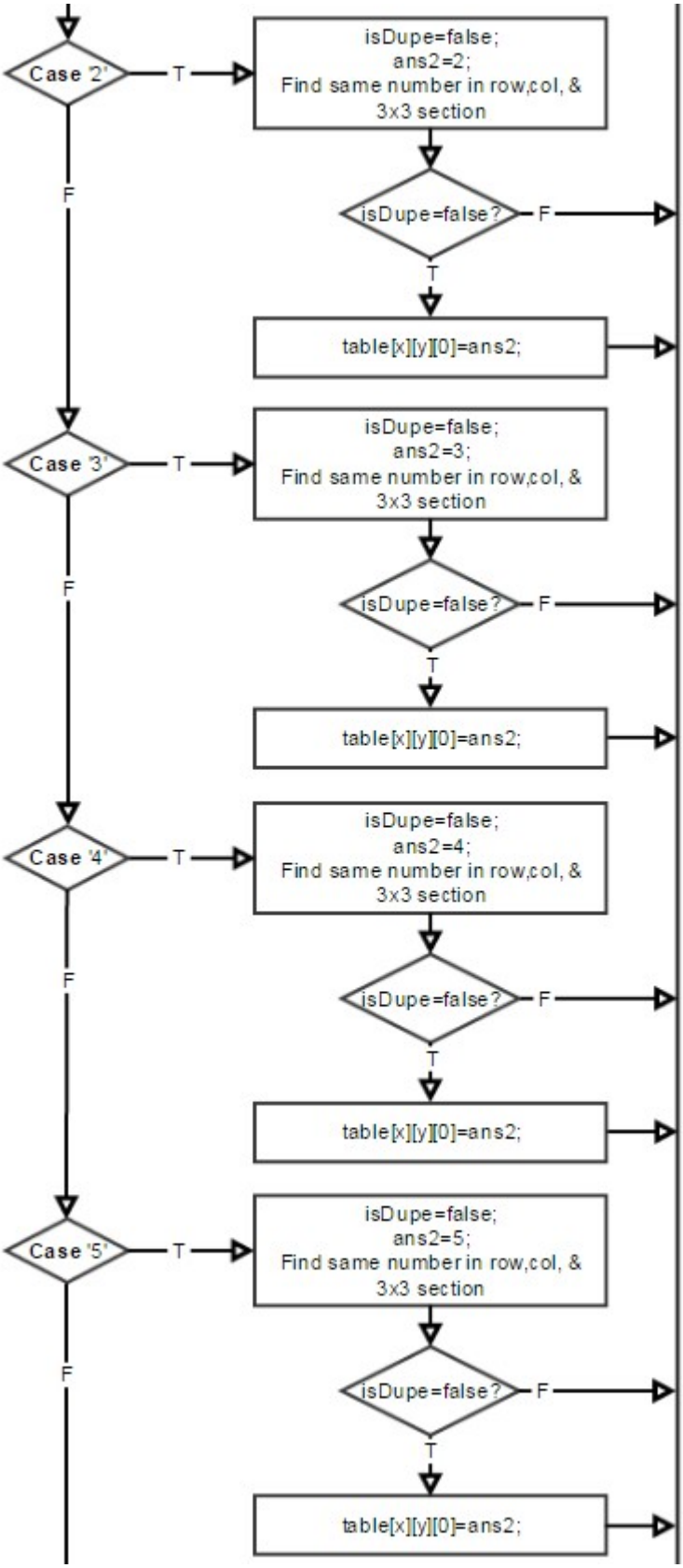


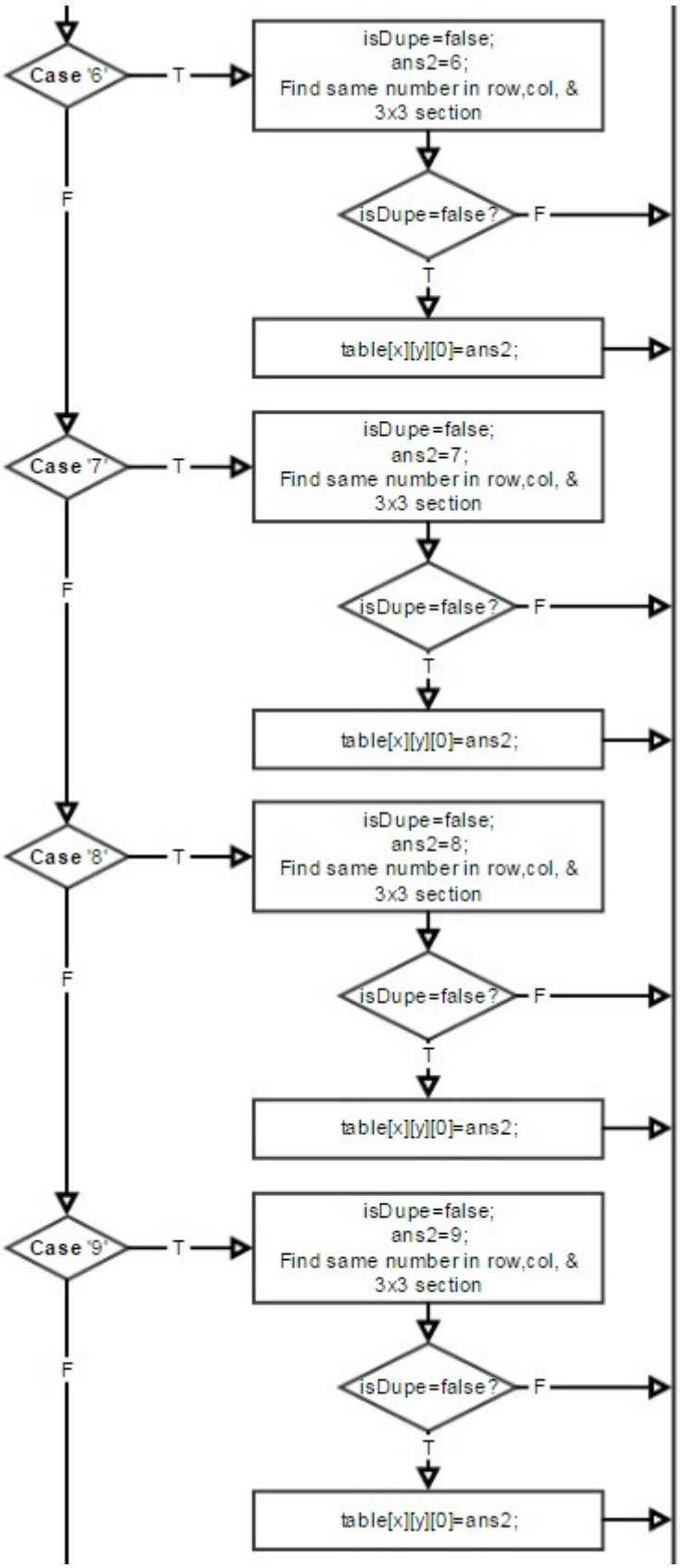

main
Sudoku Game
Program
Set random seed
Variable Declaration
int table[SIZE][SIZE][SEC] = {{},{}};
int x=0, y=0,
xDupe, yDupe,
zDupe, winCh=0,
quitCh=0;
char ans;
bool isDupe;
Display
"Welcome
Screen"
A

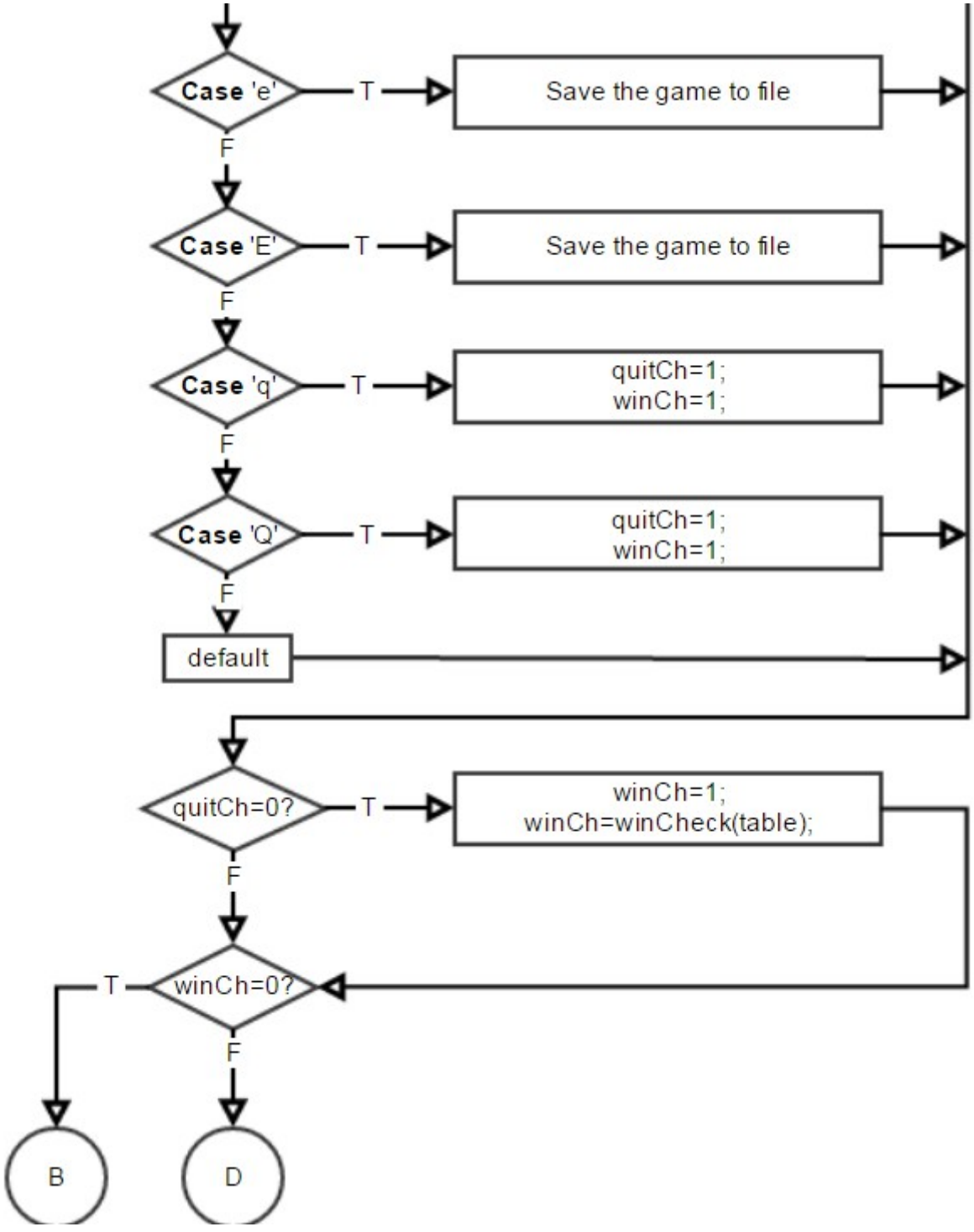

A
User input
cin>>ans;
Switch
(ans)
Case 'n'
T
Fill table with random numbers.
Find the very first blank space.
F
Case 'N'
T
Fill table with random numbers.
Find the very first blank space.
F
Case 'l'
T
Load game from file.
Find the cursor location.
F
Case 'L'
T
Load game from file.
Find the cursor location.
F
default
ans='p';
ans='p'?
T
F
xDupe=0;
yDupe=0;
zDupe=0;
B

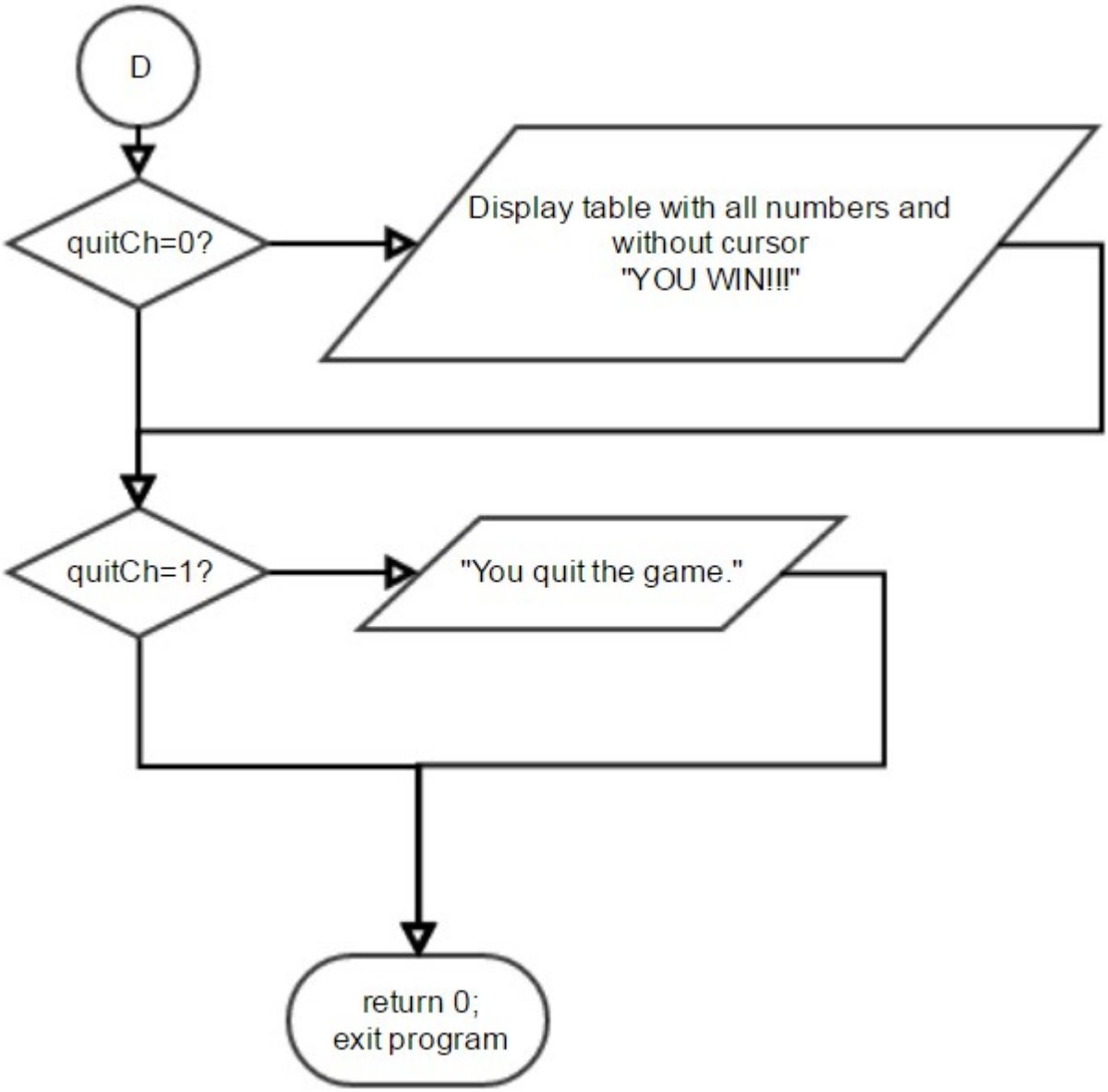

B
Display table
xDupe=1?
T
"The digit you entered is already in the same ROW. Please try again!"
F
yDupe=1?
T
"The digit you entered is already in the same COLUMN. Please try again!"
F
zDupe=1?
T
"The digit you entered is already in the same 3x3 SECTION. Please try again!"
F
"PRESS (E) to SAVE. PRESS (Q) to QUIT the game."
"MOVING THE CURSOR: (W)UP, (S)DOWN, (A)LEFT, (D)RIGHT."
User input
cin>>ans;
xDupe=0;
yDupe=0;
zDupe=0;
C


C
Switch
(ans)
Case 'w'
T
Go up the grid
F
Case 'W'
T
Go up the grid
F
Case 'a'
T
Go left of the grid
F
Case 'A'
T
Go left of the grid
F
Case 's'
T
Go down the grid
F
Case 'S'
T
Go down the grid
F
Case 'd'
T
Go right of the grid
F
Case 'D'
T
Go right of the grid
F
Case '1'
T
isDupe=false;
ans2=1;
Find same number in row,col, &
3x3 section
isDupe=false?
F
T
table[x][y][0]=ans2;
F


Case '2'
T
isDupe=false;
ans2=2;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '3'
T
isDupe=false;
ans2=3;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '4'
T
isDupe=false;
ans2=4;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '5'
T
isDupe=false;
ans2=5;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;


Case '6'
T
isDupe=false;
ans2=6;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '7'
T
isDupe=false;
ans2=7;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '8'
T
isDupe=false;
ans2=8;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;
Case '9'
T
isDupe=false;
ans2=9;
Find same number in row,col, &
3x3 section
F
isDupe=false?
F
T
table[x][y][0]=ans2;


Case 'e'
T
Save the game to file
F
Case 'E'
T
Save the game to file
F
Case 'q'
T
quitCh=1;
winCh=1;
F
Case 'Q'
T
quitCh=1;
winCh=1;
F
default
quitCh=0?
T
winCh=1;
winCh=winCheck(table);
F
T
winCh=0?
F
B
D


D
quitCh=0?
Display table with all numbers and
without cursor
"YOU WIN!!!"
quitCh=1?
"You quit the game."
return 0;
exit program

Constructs & Concepts Utilized

iostream Library

Name	Frequency	Description	Location
cout	47	Output Data	Throughout
cin	2	Input Data	Line 74, 135

cstdlib Library

Name	Frequency	Description	Location
srand()	1	Random # seed	Line 47
rand()	2	Generates rand #	Line 410, 411

ctime Library

Name	Frequency	Description	Location
time	1	Set current time	Line 47

fstream Library

Name	Frequency	Description	Location
savef.open()	2	Open file	Line 332, 364
savef.close()	2	Close file	Line 349, 385

Data Types:

Data Types	Frequency	Location
int	17	Line 28, 29, 30, 41, 50, 52-59, 404, 938-940
char	1	Line 61
bool	4	Line 31, 63, 402
ifstream	1	Line 331
ofstream	1	Line 362

Conditional Statements

Data Types	Frequency	Location
if	4	Line 118, 123, 128, 276, 283
else if	1	Line 289
switch	2	Line 75, 143

Loops

Data Types	Frequency	Location
for	51	Throughout
do-while	2	Line 72, 114

Pseudo Code

Start
Generate random seed
Display welcome screen
Ask user if they want to load a game or start a new one
Get user input

If press 'n'
    Start new. Fill the table with random numbers.
If press 'l”
    Load a game from a file
If input not 'n' nor 'l'
    Repeat the question and ask for input again

Reset the duplicate check (rows, columns, sections)

Main Game
Display the table

If there is duplicate in row
    Display notification
If there is duplicate in column
    Display notification
If there is duplicate in section
    Display notification

Display info reminding user they can save and quit anytime
Get user input

Reset the duplicate check (rows, columns, sections)

If press 'w'
    Move the cursor up the grid
If press 'a'
    Move the cursor left of the grid

If press 's'
    Move the cursor down the grid
If press 'd'
    Move the cursor right of the grid
If press '1'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '2'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '3'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '4'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '5'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '6'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array
If press '7'
    Reset duplicate check
    Assign user's answer to int variable
    Find dupe in row,col,sec
    No duplicate? Then assign answer in the array

If press '8'

Reset duplicate check

Assign user's answer to int variable

Find dupe in row,col,sec

No duplicate? Then assign answer in the array

If press '9'

Reset duplicate check

Assign user's answer to int variable

Find dupe in row,col,sec

No duplicate? Then assign answer in the array

If press 'e'

Save the game

If press 'q'

Quit the game

Check if there are any '0' left in the grid

If no '0'

User win

Else

Repeat to ask for input

Program

```
//System Libraries
#include <iostream>     //Input/Output objects
#include <cstdlib>      //Random Generator
#include <ctime>        //Time
#include <fstream>      //File I/O
using namespace std;    //Name-space used in the System Library

//User Libraries

//Global Constants
const short SIZE=9;                                 //Size of array
const short SEC=2;                                  //Size for our 3x3 Section Array in 3rd Dimension

//Function prototypes
void fillArry(int [][SIZE][SEC],int&);                  //When starting a new game. Fill table with random
numbers
void display(int [][SIZE][SEC]);                        //Display table with digits
int findX(int [][SIZE][SEC],int,int,int,int&);              //Find duplicate digit in ROW
int findY(int [][SIZE][SEC],int,int,int,int&);              //Find duplicate digit in COLUMN
int findSec(int [][SIZE][SEC],int,int,int,int&);            //Find duplicate digit in 3x3 SECTION
bool findDupe(int [][SIZE][SEC],int,int,int,int&,int&,int&);    //Use every time user input's an answer->calls
findX,findY,findSEC functions to find duplicates
void findBlank(int [][SIZE][SEC],int&,int&);                //Find a blank spot to move cursor to
void goDown(int [][SIZE][SEC],int&,int&);                   //Move down the grid
void goUp(int [][SIZE][SEC],int&,int&);                     //Move up the grid
void goRight(int [][SIZE][SEC],int&,int&);                  //Move to the right of grid
void goLeft(int [][SIZE][SEC],int&,int&);                   //Move to the left of grid
void dWelcome();                                    //Display welcome message
void load(int [][SIZE][SEC]);                           //Load game from a file
void save(int [][SIZE][SEC]);                           //Save game from a file
void findCur(int [][SIZE][SEC],int&,int&);                  //Find previous cursor location
int winCheck(int [][SIZE][SEC]);                        //Check if player wins
void displayW(int [][SIZE][SEC]);                       //Display the table with every number

//Execution Begins Here!
int main(int argc, char** argv) {
   //Set random seed
   srand(static_cast<unsigned int>(time(0)));

   //Declaration of Variables
   int     table[SIZE][SIZE][SEC] = {{},{}};               //Our one and only array

   int     x=0,                                    //X-coord of cursor
           y=0,                                    //Y-coord of cursor
           xDupe,                                  //Check if there's duplicate in row
           yDupe,                                  //Check if there's duplicate in col
           zDupe,                                  //Check if there's duplicate in 3x3 Section
           winCh=0,                                //Check if won
           quitCh=0,                               //Check if quit
           ans2;                                   //Store user's answer here if its a digit
```

```
char    ans;                                    //User's main input

bool    isDupe;                                 //Check if there's duplicate in row,col,3x3 sec

//START GAME/WELCOME SCREEN
dWelcome();

cout<<endl<<endl;
cout<<"New Game or Load Game?"<<endl;                   //Ask user if they want to load or new game
cout<<"(New: N, Load: L)"<<endl<<endl;

do
{
   cin>>ans;
   switch(ans)
   {
      case 'n':                                 //If n, start new, fill table with random numbers
            {
               fillArry(table,zDupe);
               findBlank(table,x,y);
               break;
            }
      case 'N':                                 //If N, start new, fill table with random numbers
            {
               fillArry(table,zDupe);
               findBlank(table,x,y);
               break;
            }
      case 'l':                                 //If l, load game from a file
            {
               load(table);
               findCur(table,x,y);
               break;
            }
      case 'L':                                 //If L, load game from a file
            {
               load(table);
               findCur(table,x,y);
               break;
            }
      default:                                  //If not n nor l, assign 'p' to repeat question
      {
         ans='p'; break;
      }
   }
}while(ans=='p');

xDupe=0;                                        //Reset the duplicate check from previous loop
yDupe=0;
zDupe=0;

//MAIN GAME
```

```
   do
   {
      display(table);                              //Display table with numbers

      if(xDupe==1)                                 //If there is duplicate in row, then display warning
      {
         cout<<"The digit you entered is already in the same ROW. Please try again!"<<endl<<endl;
      }

      if(yDupe==1)                                 //If there is duplicate in col, then display warning
      {
         cout<<"The digit you entered is already in the same COLUMN. Please try again!"<<endl<<endl;
      }

      if(zDupe==1)                                 //If there is duplicate in 3x3 Section, then display warning
      {
         cout<<"The digit you entered is already in the same 3x3 SECTION. Please try again!"<<endl<<endl;
      }

      cout<<endl;
      cout<<"PRESS (E) to SAVE. PRESS (Q) to QUIT the game."<<endl;          //A little info to remind they
can save and quit ANY TIME
      cout<<"MOVING THE CURSOR: (W)UP, (S)DOWN, (A)LEFT, (D)RIGHT."<<endl<<endl;

      cin>>ans;                                            //Get user's response

      xDupe=0;                                             //Reset the duplicate check from previous loop
      yDupe=0;
      zDupe=0;

      switch(ans)
      {
         case 'w': goUp(table,x,y); break;          //If pressed w, go up the grid
         case 'W': goUp(table,x,y); break;          //If pressed W, go up the grid
         case 'a': goLeft(table,x,y); break;        //If pressed a, go left of the grid
         case 'A': goLeft(table,x,y); break;        //If pressed A, go left of the grid
         case 's': goDown(table,x,y); break;        //If pressed s, go down the grid
         case 'S': goDown(table,x,y); break;        //If pressed S, go down the grid
         case 'd': goRight(table,x,y); break;       //If pressed d, go right of the grid
         case 'D': goRight(table,x,y); break;       //If pressed D, go right of the grid
         case '1':                              //If user answers 1
               {
                  isDupe=false;                 //Reset duplicate check
                  ans2=1;                       //Assign user's answer to int variable
                  isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
                  if(isDupe==false)             //Only if there is no dupe then we assign answer in the table
                  {
                     table[x][y][0]=ans2;
                  }
                  break;
               }
         case '2':
```

```
        {
          isDupe=false;                //Reset duplicate check
          ans2=2;                      //Assign user's answer to int variable
          isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
          if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
          {
            table[x][y][0]=ans2;
          }
          break;
        }
case '3':
        {
          isDupe=false;                //Reset duplicate check
          ans2=3;                      //Assign user's answer to int variable
          isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
          if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
          {
            table[x][y][0]=ans2;
          }
          break;
        }
case '4':
        {
          isDupe=false;                //Reset duplicate check
          ans2=4;                      //Assign user's answer to int variable
          isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
          if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
          {
            table[x][y][0]=ans2;
          }
          break;
        }
case '5':
        {
          isDupe=false;                //Reset duplicate check
          ans2=5;                      //Assign user's answer to int variable
          isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
          if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
          {
            table[x][y][0]=ans2;
          }
          break;
        }
case '6':
        {
          isDupe=false;                //Reset duplicate check
          ans2=6;                      //Assign user's answer to int variable
          isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
          if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
          {
            table[x][y][0]=ans2;
          }
          break;
```

```
        }
case '7':
        {
           isDupe=false;                //Reset duplicate check
           ans2=7;                      //Assign user's answer to int variable
           isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
           if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
           {
              table[x][y][0]=ans2;
           }
           break;
        }
case '8':
        {
           isDupe=false;                //Reset duplicate check
           ans2=8;                      //Assign user's answer to int variable
           isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
           if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
           {
              table[x][y][0]=ans2;
           }
           break;
        }
case '9':
        {
           isDupe=false;                //Reset duplicate check
           ans2=9;                      //Assign user's answer to int variable
           isDupe=findDupe(table,ans2,x,y,xDupe,yDupe,zDupe);      //Find dupe in row,col,sec
           if(isDupe==false)            //Only if there is no dupe then we assign answer in the table
           {
              table[x][y][0]=ans2;
           }
           break;
        }
case 'e':                           //If pressed e, then save game
        {
           save(table);
           break;
        }
case 'E':                           //If pressed e, then save game
        {
           save(table);
           break;
        }
case 'q':                           //If pressed q, then quit game
        {
           quitCh=1;
           winCh=1;
           break;
        }
case 'Q':                           //If pressed q, then quit game
        {
           quitCh=1;
```

```
                winCh=1;
                break;
            }
        default: break;
      }
      if(quitCh==0)                    //Check if user quit
      {
        winCh=1;
        winCh=winCheck(table);          //Find any '0' all rows, columns. If no '0' then win!
      }
   }while(winCh==0);                   //Repeat until winCh=1

   if(quitCh==0)                       //If did not quit, display win
   {
      displayW(table);

      cout<<"YOU WIN!!!"<<endl;
   }
   else if(quitCh==1)                   //If quit, and did not win, display quit game
   {
      cout<<endl<<endl;
      cout<<"You quit the game."<<endl;
   }
   //Exit Program
   return 0;
}
```

Functions

```
//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  dWelcome  ************************************
//Purpose:  Display the welcome screen: SUDOKU (big letters and borders).
//Inputs:   n/a
//Output:   n/a
//*****************************************************************************
void dWelcome()
{
   cout<<"*********************************************"<<endl;
   cout<<"*********************************************"<<endl;
   cout<<"***  ##  # # ###  ##  # # # # ***"<<endl;
   cout<<"*** # # # # # # # # # # # # ***"<<endl;
   cout<<"*** #   # # # # # # ##  # # ***"<<endl;
   cout<<"***  #  # # # # # # ##  # # ***"<<endl;
   cout<<"***   #  # # # # # # ##  # # ***"<<endl;
   cout<<"***    # # # # # # # ##  # # ***"<<endl;
   cout<<"*** # # # # # # # # # # # # ***"<<endl;
   cout<<"***  ##  ##  ###  ##  # #  ##  ***"<<endl;
   cout<<"*********************************************"<<endl;
   cout<<"*********************************************"<<endl;
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  load  ****************************************
//Purpose:  Load a previous saved game from a file "savefile.dat".
//Inputs:   array->The game table
//Output:   n/a
//*****************************************************************************
void load(int array[][SIZE][SEC])
{
   ifstream savef;
   savef.open("savefile.dat");

   for(int x=0; x<SIZE; x++)
   {
      for(int y=0; y<SIZE; y++)
      {
         savef>>array[x][y][0];
      }
   }

   for(int x=0; x<SIZE; x++)
   {
      for(int y=0; y<SIZE; y++)
      {
         savef>>array[x][y][1];
      }
```

```
   }
   savef.close();
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  save  ****************************************
//Purpose:  Save the game to the file "savefile.dat".
//Inputs:   array->The game table
//Output:   n/a
//*************************************************************************
void save(int array[][SIZE][SEC])
{
   ofstream savef;

   savef.open("savefile.dat");

   for(int x=0; x<SIZE; x++)
   {
      for(int y=0; y<SIZE; y++)
      {
         savef<<array[x][y][0]<<" ";
      }
      savef<<endl;
   }

   for(int x=0; x<SIZE; x++)
   {
      for(int y=0; y<SIZE; y++)
      {
         savef<<array[x][y][1]<<" ";
      }
      savef<<endl;
   }
   cout<<endl<<endl;
   cout<<"GAME SAVED!"<<endl;
   savef.close();
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  fillArry  ************************************
//Purpose:  Fill out the array(or game) with random numbers in random locations(boxes).
//Inputs:   array->The game table
//          zDupe->Value if there's duplicate in the same 3x3 section
//Output:   n/a
//*************************************************************************
void fillArry(int array[][SIZE][SEC],int& zDupe)
{
   short   rn,                          //Random number to fill the game
           rn2;                         //Holds number if we will reveal a slot or not
```

```
bool    isSameX,isSameY;

int     isSameZ;              //Check if it has the same number in rows,col,section

for(int x=0; x<SIZE; x++)
{
   for(int y=0; y<SIZE; y++)                //REPEAT 9 times to fill row
   {
      rn=rand()%SIZE+1;                      //Set random number to fill slot
      rn2=rand()%SIZE+1;

      if(rn2==1 || rn2==2 || rn2==3)
      {
         do
         {
            isSameY=false;
            isSameX=false;
            isSameZ=0;
            for(int x2=0; x2<SIZE; x2++)       //Go through all 9 slots find if same number
            {
               if(array[x][x2][0]==rn)              //If same, set rn to 0 or blank
               {
                  if(rn==9) { rn=1; }
                  else { rn++; }
                  isSameX=true;
               }

               if(isSameX==false)
               {
                  for(int x3=0; x3<SIZE; x3++)
                  {
                     if(array[x3][y][0]==rn)              //If same, set rn to 0 or blank
                     {
                        if(rn==9) { rn=1; }
                        else { rn++; }
                        isSameY=true;
                     }
                  }
               }

               if(isSameY==false)
               {
                  isSameZ=findSec(array,rn,x,y,zDupe);
                  if(isSameZ==1)
                  {
                     if(rn==9) { rn=1; }
                     else { rn++; }
                  }
               }
            }
         }while(isSameX==true || isSameY==true || isSameZ==1);
         array[x][y][0]=rn;
```

```
            array[x][y][1]=rn;
         }
      }
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  display  *************************************
//Purpose:  Display our game board (or table/array) with blanks and player cursor.
//Inputs:   array->The game table
//Output:   n/a
//*******************************************************************************
void display(int array[][SIZE][SEC])
{
   cout<<"************************************"<<endl;
   for(int x=0; x<SIZE; x++)
   {
      cout<<"* ";
      for(int y=0; y<SIZE; y++)
      {
         if(array[x][y][1]==10)
         {
            cout<<"#";
         }
         else if(array[x][y][0]==0)
         {
            if(array[x][y][1]==10) cout<<"#";
            else cout<<" ";
         }
         else cout<<array[x][y][0];
         if (y==2 || y==5 || y==8) cout<<" * ";
         else cout<<" | ";
      }
      cout<<endl;
      if(x==2 || x==5 || x==8) cout<<"************************************"<<endl;
      else cout<<"*---+---+---*---+---+---*---+---+---*"<<endl;
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  displayW  ************************************
//Purpose:  Display our game board (or table/array) all filled and WITHOUT the
//          player cursor.
//Inputs:   array->The game table
//Output:   n/a
//*******************************************************************************
void displayW(int array[][SIZE][SEC])
{
   cout<<"************************************"<<endl;
```

```
   for(int x=0; x<SIZE; x++)
   {
      cout<<"* ";
      for(int y=0; y<SIZE; y++)
      {
         cout<<array[x][y][0];
         if (y==2 || y==5 || y==8) cout<<" * ";
         else cout<<" | ";
      }
      cout<<endl;
      if(x==2 || x==5 || x==8) cout<<"*************************************"<<endl;
      else cout<<"*---+---+---*---+---+---*---+---+---*"<<endl;
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  findBlank  ************************************
//Purpose:  Find the very first "0" or blank box in the array for the player's
//          cursor.
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*************************************************************************
void findBlank(int array[][SIZE][SEC],int& x,int& y)
{
   for(int x1=0; x1<SIZE; x1++)
   {
      for(int y1=0; y1<SIZE; y1++)
      {
         if(array[x1][y1][1]==0)
         {
            array[x1][y1][1]=10;
            x=x1;
            y=y1;
            return;
         }
      }
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  findCur  ************************************
//Purpose:  Find the location of previous player's cursor
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*************************************************************************
```

```
void findCur(int array[][SIZE][SEC],int& x,int& y)
{
   for(int x1=0; x1<SIZE; x1++)
   {
      for(int y1=0; y1<SIZE; y1++)
      {
         if(array[x1][y1][1]==10)
         {
            x=x1;
            y=y1;
            return;
         }
      }
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  goDown  ************************************
//Purpose:  Move the player's cursor down the grid to the next BLANK space
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*********************************************************************************
void goDown(int array[][SIZE][SEC],int& x,int& y)
{
   for(int x1=x; x1<SIZE; x1++)
   {
      if(array[x1][y][1]==0)
      {
         array[x1][y][1]=10;
         array[x][y][1]=0;
         x=x1;
         return;
      }
      if(x1==8)
      {
         if(array[0][y][1]==0)
         {
            array[0][y][1]=10;
            array[x][y][1]=0;
            x=0;
            return;
         }
         x1=0;
      }
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
```

```
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  goUp  ****************************************
//Purpose:  Move the player's cursor up the grid to the next BLANK space
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*************************************************************************
void goUp(int array[][SIZE][SEC],int& x,int& y)
{
   for(int x1=x; x1>=0; x1--)
   {
      if(array[x1][y][1]==0)
      {
         array[x1][y][1]=10;
         array[x][y][1]=0;
         x=x1;
         return;
      }
      if(x1==0)
      {
         if(array[8][y][1]==0)
         {
            array[8][y][1]=10;
            array[x][y][1]=0;
            x=8;
            return;
         }
         x1=8;
      }
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//************************  goRight  *************************************
//Purpose:  Move the player's cursor to the right of the grid to the next BLANK space
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*************************************************************************
void goRight(int array[][SIZE][SEC],int& x,int& y)
{
   for(int y1=y; y1<SIZE; y1++)
   {
      if(array[x][y1][1]==0)
      {
         array[x][y1][1]=10;
         array[x][y][1]=0;
         y=y1;
         return;
```

```
      }
      if(y1==8)
      {
         if(array[x][0][1]==0)
         {
            array[x][0][1]=10;
            array[x][y][1]=0;
            y=0;
            return;
         }
         y1=0;
      }
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//*************************  goLeft  ***************************************
//Purpose:  Move the player's cursor to the left of the grid to the next BLANK space
//Inputs:   array->The game table
//          x->Row of cursor location
//          y->Col of cursor location
//Output:   n/a
//*****************************************************************************
void goLeft(int array[][SIZE][SEC],int& x,int& y)
{
   for(int y1=y; y1>=0; y1--)
   {
      if(array[x][y1][1]==0)
      {
         array[x][y1][1]=10;
         array[x][y][1]=0;
         y=y1;
         return;
      }
      if(y1==0)
      {
         if(array[x][8][1]==0)
         {
            array[x][8][1]=10;
            array[x][y][1]=0;
            y=8;
            return;
         }
         y1=8;
      }
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777 8
//3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

```
//************************  findX  ***************************************
//Purpose:  Look for duplicates across the ROW of cursor
//Inputs:   array->The game table
//          num->The number user answered
//          x->Row of cursor location
//          y->Col of cursor location
//          isX->If there is a duplicate in the row
//Output:   Return 1, if we find a duplicate
//************************************************************************
int findX(int array[][SIZE][SEC],int num,int x,int y,int& isX)
{
   for(int y2=0; y2<SIZE; y2++)
   {
      if(array[x][y2][0]==num)
      {
         cout<<"DUPE! X!"<<endl;
         isX=1;
         return 1;
      }
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777
//3456789012345678901234567890123456789012345678901234567890123456789012345678
//************************  findY  ***************************************
//Purpose:  Look for duplicates across the COLUMN of cursor
//Inputs:   array->The game table
//          num->The number user answered
//          x->Row of cursor location
//          y->Col of cursor location
//          isY->If there is a duplicate in the column
//Output:   Return 1, if we find a duplicate
//************************************************************************
int findY(int array[][SIZE][SEC],int num,int x,int y,int& isY)
{
   for(int x2=0; x2<SIZE; x2++)
   {
      if(array[x2][y][0]==num)
      {
         cout<<"DUPE! Y!"<<endl;
         isY=1;
         return 1;
      }
   }
}


//0000000111111111122222222223333333333444444444455555555556666666666777777777
//3456789012345678901234567890123456789012345678901234567890123456789012345678
//************************  findSec  ***************************************
//Purpose:  Look for duplicates in the same 3x3 SECTION of the cursor
//Inputs:   array->The game table
```

```
//          num->The number user answered
//          x->Row of cursor location
//          y->Col of cursor location
//          isZ->If there is a duplicate in the 3x3 SECTION
//Output:   Return 1, if we find a duplicate
//*************************************************************************
int findSec(int array[][SIZE][SEC],int num,int x,int y,int& isZ)
{
   if(x<3)
   {
      if(y<3)
      {
         for(int x2=0; x2<3; x2++)
         {
            for(int y2=0; y2<3; y2++)
            {
               if(num==array[x2][y2][0])
               {
                  isZ=1;
                  return 1;
               }
            }
         }
      }

      else if(y>2 && y<6)
      {
         for(int x2=0; x2<3; x2++)
         {
            for(int y2=3; y2<6; y2++)
            {
               if(num==array[x2][y2][0])
               {
                  isZ=1;
                  return 1;
               }
            }
         }
      }

      else
      {
         for(int x2=0; x2<3; x2++)
         {
            for(int y2=6; y2<9; y2++)
            {
               if(num==array[x2][y2][0])
               {
                  isZ=1;
                  return 1;
               }
            }
         }
```

```
    }
}
else if(x>2 && x<6)
{
    if(y<3)
    {
        for(int x2=3; x2<6; x2++)
        {
            for(int y2=0; y2<3; y2++)
            {
                if(num==array[x2][y2][0])
                {
                    isZ=1;
                    return 1;
                }
            }
        }
    }

    else if(y>2 && y<6)
    {
        for(int x2=3; x2<6; x2++)
        {
            for(int y2=3; y2<6; y2++)
            {
                if(num==array[x2][y2][0])
                {
                    isZ=1;
                    return 1;
                }
            }
        }
    }

    else
    {
        for(int x2=3; x2<6; x2++)
        {
            for(int y2=6; y2<9; y2++)
            {
                if(num==array[x2][y2][0])
                {
                    isZ=1;
                    return 1;
                }
            }
        }
    }
}
else
{
    if(y<3)
    {
```

```
            for(int x2=6; x2<9; x2++)
            {
               for(int y2=0; y2<3; y2++)
               {
                  if(num==array[x2][y2][0])
                  {
                     isZ=1;
                     return 1;
                  }
               }
            }
         }

         else if(y>2 && y<6)
         {
            for(int x2=6; x2<9; x2++)
            {
               for(int y2=3; y2<6; y2++)
               {
                  if(num==array[x2][y2][0])
                  {
                     isZ=1;
                     return 1;
                  }
               }
            }
         }

         else
         {
            for(int x2=6; x2<9; x2++)
            {
               for(int y2=6; y2<9; y2++)
               {
                  if(num==array[x2][y2][0])
                  {
                     isZ=1;
                     return 1;
                  }
               }
            }
         }
      }
   }

//00000001111111111222222222233333333334444444445555555555666666666777777777 8
//34567890123456789012345678901234567890123456789012345678901234567890123456789 0
//************************  findDupe  ************************************
//Purpose:  The function to run to find duplicates in ROW, COLUMN, and 3x3 SECTION
//Inputs:   array->The game table
//          num->The number user answered
//          x->Row of cursor location
//          y->Col of cursor location
```

```
//          xDupe->If there is a duplicate in the row
//          yDupe->If there is a duplicate in the column
//          zDupe->If there is a duplicate in the 3x3 SECTION
//Output:   Return true, if we find duplicate IN ANY of the 3 (row,column,3x3)
//*************************************************************************
bool findDupe(int array[][SIZE][SEC],int num,int x,int y,int& xDupe,int& yDupe,int& zDupe)
{
   int     isX,
           isY,
           isZ;

   isX=0;
   isY=0;
   isZ=0;

   isX=findX(array,num,x,y,xDupe);
   isY=findY(array,num,x,y,yDupe);
   isZ=findSec(array,num,x,y,zDupe);

   if(isX==1 || isX==1 || isZ==1)
   {
      return true;
   }
}


//000000011111111112222222222333333333344444444455555555556666666666777777777 8
//345678901234567890123456789012345678901234567890123456789012345678901234567890
//*************************   winCheck   ************************************
//Purpose:  Check the entire array if there is any "0" left
//Inputs:   array->The game table
//Output:   Return 0, if there is "0" in the array
//*************************************************************************
int winCheck(int array[][SIZE][SEC])
{
   for(int x=0; x<SIZE; x++)
   {
      for(int y=0; y<SIZE; y++)
      {
         if(array[x][y][0]==0)
         {
            return 0;
         }
      }
   }
}
```