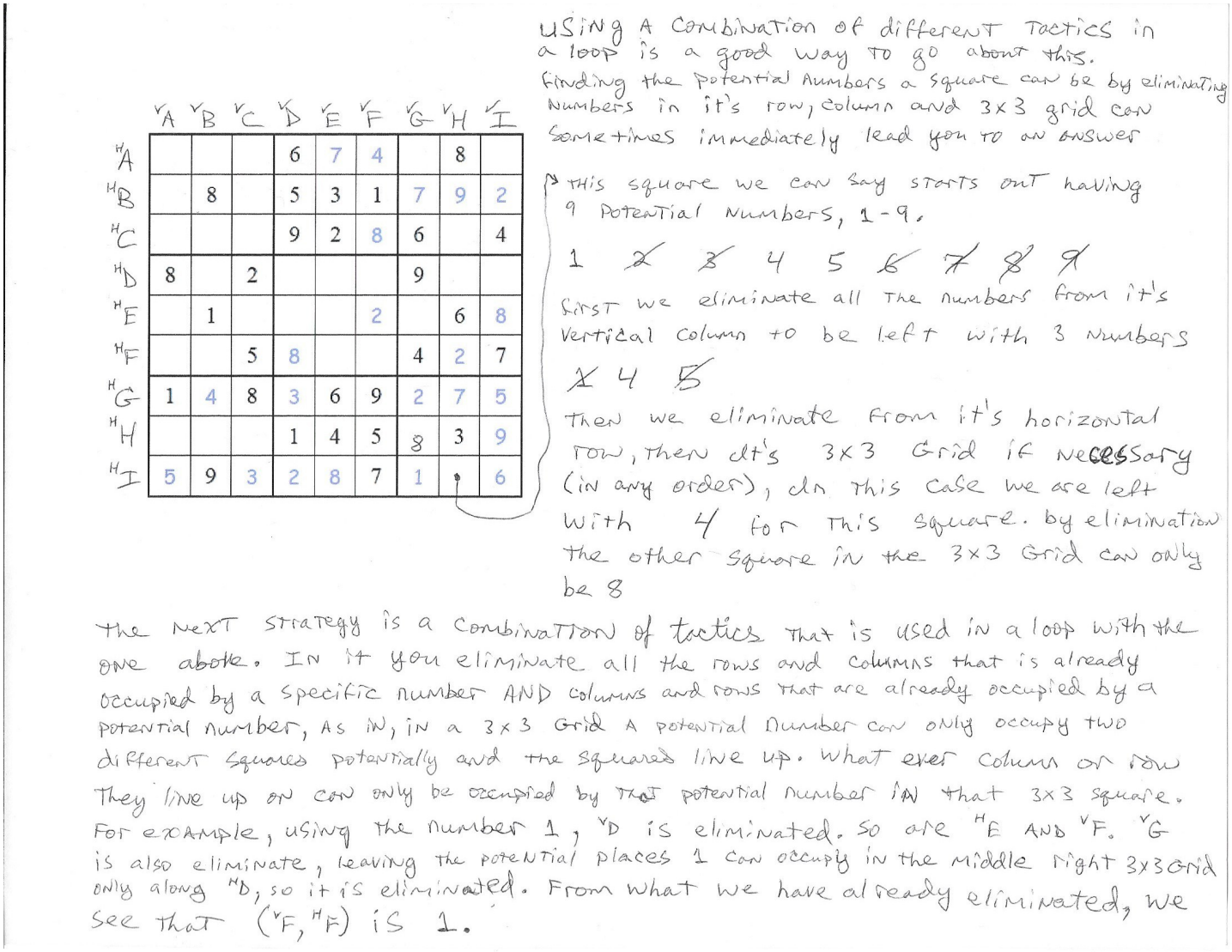
This Sudoku solver I wrote will solve any Sudoku puzzle. First it will try to solve the puzzle by certain logic methods. If that fails, it will then use brute force to solve it, no matter which Sudoku puzzle it is.

Please zoom in if it's hard to read.



If these tactics fail to solve the puzzle, brute force is then applied until the puzzle is solved. Brute force will solve any puzzle. At each empty square, a randomly selected potential number of that square is taken and tested to see if it was already used in the row, column or 3x3 grid. If it was not used, it is added to the square and the program goes to the next empty square. If it was used, the program picks another randomly selected potential number of that square and tests it. If all the potential numbers of an empty square are already taken, a new brute force attempt is started. Brute force is applied until the puzzle is solved.

A page of sample output, followed by the program's source code, follows.

```
-------------------------
~ 1¦ ¦7 ~ 6¦9¦  ~  ¦ ¦  ~
-------------------------
~ 9¦ ¦  ~ 3¦5¦  ~ 1¦7¦  ~
=========================
43
=========================
~  ¦4¦9 ~ 8¦ ¦5 ~  ¦1¦  ~
-------------------------
~  ¦ ¦1 ~ 4¦ ¦6 ~ 9¦ ¦  ~
-------------------------
~  ¦8¦  ~ 9¦2¦1 ~  ¦ ¦4 ~
=========================
~  ¦1¦  ~ 2¦ ¦9 ~  ¦ ¦5 ~
-------------------------
~  ¦ ¦  ~ 1¦ ¦  ~  ¦9¦  ~
-------------------------
~ 8¦9¦  ~ 5¦ ¦  ~  ¦4¦1 ~
=========================
~ 5¦ ¦  ~ 7¦1¦  ~  ¦2¦9 ~
-------------------------
~ 1¦ ¦7 ~ 6¦9¦  ~  ¦ ¦  ~
-------------------------
~ 9¦ ¦  ~ 3¦5¦  ~ 1¦7¦  ~
=========================
42
Could not solve!
Oh, no!

Brute force being applied, please wait...

=========================
~ 3¦4¦9 ~ 8¦7¦5 ~ 2¦1¦6 ~
-------------------------
~ 2¦5¦1 ~ 4¦3¦6 ~ 9¦8¦7 ~
-------------------------
~ 7¦8¦6 ~ 9¦2¦1 ~ 5¦3¦4 ~
=========================
~ 4¦1¦3 ~ 2¦8¦9 ~ 7¦6¦5 ~
-------------------------
~ 6¦7¦5 ~ 1¦4¦3 ~ 8¦9¦2 ~
-------------------------
~ 8¦9¦2 ~ 5¦6¦7 ~ 3¦4¦1 ~
=========================
~ 5¦3¦8 ~ 7¦1¦4 ~ 6¦2¦9 ~
-------------------------
~ 1¦2¦7 ~ 6¦9¦8 ~ 4¦5¦3 ~
-------------------------
~ 9¦6¦4 ~ 3¦5¦2 ~ 1¦7¦8 ~
=========================
Brute force for the win!

Number filled at start: 24

Process returned 0 (0x0)   execution time : 0.361 s
Press any key to continue.
```

```
/**
  Name: Sudoku
  Copyright:
  Author: Jeremy Alexandre
  Date:
  Description: Sudoku solver that will solve any Sudoku puzzle. Input is
  received through infile stream, from a file named "data.txt". The format
  of the input file is space for an empty square and number for a square with
  that number.

  First it uses certain logic methods to try to solve the puzzle. If that fails,
  it uses brute force to solve the puzzle.

  Here is an example "data.txt" file, copy from after open quotation mark to
  before close quotation. Don't forget to add spaces so you have 9 sections
  to a row:

"53  7
6  195
 98    6
8   6   3
4  8 3  1
7   2   6
 6    28
   419  5
    8  79"

  The following is an example of a puzzle that will cause brute force to be
  applied:

"   74
    1 563
2    94
9     31
  6   8
 32     7
  82    4
371 9
    57   "

*/

#include <cstdlib>
#include <iostream>
#include <cctype>
#include <fstream>
#include <ctime>

using namespace std;

ifstream infile;    //creates an instream file



/*****************************************************************************/
/**

 Fills the array from infile, placing numbers for numbers and zeros for
 spaces. When ever a space is encountered, toSolve also get incremented.
 Function then returns true. If there are missing numbers or spaces, sends
 the row and number of missing numbers and/or spaces to cout and returns
 false. If there is an invalid character, sends the invalid character to
 cout and returns false.

**/
/*****************************************************************************/
```

```
67
68
69
70  bool fillBoard(int arr[][9], int &toSolve)
71  {
72      int c;
73
74      infile.open("data.txt");    //links infile to
75      //data.txt
76
77      //tests to see if infile is open
78      if ( !infile.is_open() )
79      {
80          cout << endl << "ERROR: unable to open infile" << endl;
81          system ("PAUSE");
82          exit(1);
83      }
84
85
86      /// Fills the array from infile, placing numbers for numbers and zeros for
87      /// spaces. When ever a space is encountered, toSolve also get incremented.
88      /// If there are missing numbers or spaces, sends the row and
89      /// number of missing numbers and/or spaces to cout and returns false.
90      /// If there is an invalid character, sends the invalid character to cout
91      /// and returns false.
92      for (int j = 0; j < 9; j++)
93      {
94          for (int k = 0; k < 9; k++)
95          {
96              c = infile.get();
97              if (c >= '1' && c <= '9')
98              {
99                  arr[j][k] = c - 48;
100             }
101             else if (c == ' ')
102             {
103                 arr[j][k] = 0;
104                 toSolve++;
105             }
106             else if ((c == 10) || (c == -1))
107             {
108                 cout << "Row " << j + 1 << " missing " << 9 - k
109                      << " number(s) and/or space(s)\n";
110                 return false;
111             }
112             else
113             {
114                 cout.put(c) << endl;
115                 cout << "invalid character." << endl;
116                 return false;
117             }
118         }// end for (int k = 0; k < 9; k++)
119
120         /// Ignore the endl character
121         infile.ignore();
122
123     }// end for (int j = 0; j < 9; j++)
124
125     return true;
126 }// end bool fillBoard(int arr[][9], int &toSolve)
127
128
129
130 /*****************************************************************************/
131 /**
132
```

```
 Receives the array filled with the puzzle and attempts to solve it using
 logic. If an answer to a square is discovered it is added to the array. If
 the puzzle is solved, returns true. If the puzzle cannot be solved using
 these logic methods, returns false.

**/
/*****************************************************************************/



bool solve(int arr[][9], int &leftToSolve)
{
    bool change, potential[9], elimination, gridPotential[9][9];
    int potentialLeft;

    void display(int [][9]);
    void squareEliminator(bool [][9], int, int);
    void actualAddAndElimPotentialElim(int arr[][9], bool gridPotential[][9],
                                       bool &elimination, bool &change,
                                       int &leftToSolve, int &i);



    do
    {
        change = false;

        for (int j = 0; j < 9; j++)
        {
            for (int k = 0; k < 9; k++)
            {
                if (arr[j][k] == 0)
                {
                    /// Before any numbers are eliminated, there are 9 potential
                    /// numbers, the numbers 1-9
                    for (int i = 0; i < 9; i++)
                        potential[i] = true;

                    potentialLeft = 9;


                    /// Checks the row of the square to see which numbers where
                    /// already used and therefore not a potential number for
                    /// the square
                    for (int i = 0; i < 9; i++)
                    {
                        if (arr[j][i] > 0)
                        {
                            potential[arr[j][i] - 1] = false;

                            potentialLeft--;
                        }// end if (arr[j][i] > 0)

                    }// end for (int i = 0; i < 9; i++)



                    /// Checks the column of the square to see which numbers
                    /// where already used and therefore not a potential number
                    /// for the square. It also checks to make sure the number
                    /// has not already been eliminated.
                    for (int i = 0; i < 9; i++)
                    {
                        if (arr[i][k] > 0)
                            if (potential[arr[i][k] - 1] == true)
                            {
```

```
                        potential[arr[i][k] - 1] = false;

                        potentialLeft--;
                    }// end if (arr[i][k] > 0)

                }// end for (int i = 0; i < 9; i++)



                /// This section checks within the 3 x 3 grid of the square
                /// being tested to see which numbers are taken, and
                /// therefore not a potential number for the square.
                /// As before, it also checks to make sure the number
                /// has not already been eliminated.

                /// Top left 3x3 grid
                if (j <= 2 && k <=2)
                {
                    for (int m = 0; m < 3; m++)
                        for (int n = 0; n < 3; n++)
                            if (arr[m][n] > 0)
                                if (potential[arr[m][n] - 1] == true)
                                {
                                    potential[arr[m][n] - 1] = false;

                                    potentialLeft--;
                                }
                }
                /// Top center 3x3 grid
                else if (j <= 2 && (k >= 3 && k <= 5))
                {
                    for (int m = 0; m < 3; m++)
                        for (int n = 3; n < 6; n++)
                            if (arr[m][n] > 0)
                                if (potential[arr[m][n] - 1] == true)
                                {
                                    potential[arr[m][n] - 1] = false;

                                    potentialLeft--;
                                }
                }
                /// Top right 3x3 grid
                else if (j <= 2 && (k >= 6 && k <= 8))
                {
                    for (int m = 0; m < 3; m++)
                        for (int n = 6; n < 9; n++)
                            if (arr[m][n] > 0)
                                if (potential[arr[m][n] - 1] == true)
                                {
                                    potential[arr[m][n] - 1] = false;

                                    potentialLeft--;
                                }
                }
                /// Middle left 3x3 grid
                else if ((j >= 3 && j <= 5) && k <=2)
                {
                    for (int m = 3; m < 6; m++)
                        for (int n = 0; n < 3; n++)
                            if (arr[m][n] > 0)
                                if (potential[arr[m][n] - 1] == true)
                                {
                                    potential[arr[m][n] - 1] = false;

                                    potentialLeft--;
                                }
```

```
                    }
                    /// Middle center 3x3 grid
                    else if ((j >= 3 && j <= 5) && (k >= 3 && k <= 5))
                    {
                        for (int m = 3; m < 6; m++)
                            for (int n = 3; n < 6; n++)
                                if (arr[m][n] > 0)
                                    if (potential[arr[m][n] - 1] == true)
                                    {
                                        potential[arr[m][n] - 1] = false;

                                        potentialLeft--;
                                    }
                    }
                    /// Middle right 3x3 grid
                    else if ((j >= 3 && j <= 5) && (k >= 6 && k <= 8))
                    {
                        for (int m = 3; m < 6; m++)
                            for (int n = 6; n < 9; n++)
                                if (arr[m][n] > 0)
                                    if (potential[arr[m][n] - 1] == true)
                                    {
                                        potential[arr[m][n] - 1] = false;

                                        potentialLeft--;
                                    }
                    }
                    /// Bottom left 3x3 grid
                    else if ((j >= 6 && j <= 8) && k <=2)
                    {
                        for (int m = 6; m < 9; m++)
                            for (int n = 0; n < 3; n++)
                                if (arr[m][n] > 0)
                                    if (potential[arr[m][n] - 1] == true)
                                    {
                                        potential[arr[m][n] - 1] = false;

                                        potentialLeft--;
                                    }
                    }
                    /// Bottom center 3x3 grid
                    else if ((j >= 6 && j <= 8) && (k >= 3 && k <= 5))
                    {
                        for (int m = 6; m < 9; m++)
                            for (int n = 3; n < 6; n++)
                                if (arr[m][n] > 0)
                                    if (potential[arr[m][n] - 1] == true)
                                    {
                                        potential[arr[m][n] - 1] = false;

                                        potentialLeft--;
                                    }
                    }
                    /// Bottom right 3x3 grid
                    else
                    {
                        for (int m = 6; m < 9; m++)
                            for (int n = 6; n < 9; n++)
                                if (arr[m][n] > 0)
                                    if (potential[arr[m][n] - 1] == true)
                                    {
                                        potential[arr[m][n] - 1] = false;

                                        potentialLeft--;
                                    }
                    }
```

```
331
332
333                     if (potentialLeft == 1)
334                     {
335                         for (int i = 0; i < 9; i++)
336                             if(potential[i] == true)
337                             {
338                                 arr[j][k] = i + 1;
339
340                                 change = true;
341
342                                 leftToSolve--;
343
344                                 display(arr);
345
346                                 cout << leftToSolve << endl;
347                             }
348                     }
349
350
351                 }// end if (arr[j][k] == 0)
352
353             }//end for (int k = 0; k < 9; k++)
354
355         }// end for (int j = 0; j < 9; j++)
356
357
358
359         for (int i = 1; i <= 9; i++)
360         {
361
362             /// Initialize all the squares to false if they are occupied
363             /// and true if they are empty
364             for (int m = 0; m < 9; m++)
365                 for (int n = 0; n < 9; n++)
366                     if (arr[m][n] > 0)
367                     {
368                         gridPotential[m][n] = false;
369                     }
370                     else
371                     {
372                         gridPotential[m][n] = true;
373                     }
374
375
376
377
378             for (int j = 0; j < 9; j++)
379                 for (int k = 0; k < 9; k++)
380                 {
381
382                     if (arr[j][k] == i)
383                     {
384                         /// Eliminates the column, row and 3x3 grid the
385                         /// number occupies
386                         squareEliminator(gridPotential, j, k);
387
388
389
390                         /// This row has already been eliminated, start at the
391                         /// beginning of the next row. k is negative one
392                         /// because it will be incremented to 0 at the start of
393                         /// it's for loop. If j equals 8, incrementing j would
394                         /// cause the next set of test parameters to be
395                         /// 9 and 0 which is outside the range of the array
396                         if (j != 8)
```

```
                        {
                            k = -1;
                            j++;
                        }
                    }// end if (arr[j][k] == i)
                }


            do
            {
                elimination = false;

                actualAddAndElimPotentialElim(arr, gridPotential, elimination,
                                              change, leftToSolve, i);

            }
            while (elimination);

        }// end for (int i = 1; i <= 9; i++)


    }
    while (change);

    if (leftToSolve == 0)
    {
        cout << "Solved!" << endl;
        return true;
    }
    else
    {
        cout << "Could not solve!" << endl;
        return false;
    }

}// end bool solve(int arr[][9], int &leftToSolve)



/*****************************************************************************/
/**

 Displays the puzzle.

**/
/*****************************************************************************/



void display(int arr[][9])
{
    cout << "=========================" << endl;

    for (int j = 0; j < 9; j++)
    {
        cout << "~ ";
        for (int k = 0; k < 9; k++)
        {
            if (arr[j][k] == 0)
                cout << " ";
            else
                cout << arr[j][k];
            if ((k % 3) != 2)
                cout << "|";
            else if (k != 8 )
                cout << " ~ ";
```

```
            else
                cout << " ~";
        }// end for (int k = 0; k < 9; k++)

        cout << endl;

        if ((j % 3) != 2)
            cout << "-------------------------" << endl;
        else cout << "=========================" << endl;

    }// end for (int j = 0; j < 3; j++)


}



/*****************************************************************************/
/**

 Eliminates the column, row and 3x3 square the number occupies

**/
/*****************************************************************************/



void squareEliminator(bool gridPotential[][9], int j, int k)
{
    /// Eliminates both the column and row the number
    /// occupies
    for (int m = 0; m < 9; m++)
    {
        gridPotential[j][m] = false;
        gridPotential[m][k] = false;
    }

    /// Eliminates the 3x3 grid the number occupies

    /// Top left 3x3 grid
    if (j <= 2 && k <=2)
    {
        for (int m = 0; m < 3; m++)
            for (int n = 0; n < 3; n++)
                gridPotential[m][n] = false;
    }
    /// Top center 3x3 grid
    else if (j <= 2 && (k >= 3 && k <= 5))
    {
        for (int m = 0; m < 3; m++)
            for (int n = 3; n < 6; n++)
                gridPotential[m][n] = false;
    }
    /// Top right 3x3 grid
    else if (j <= 2 && (k >= 6 && k <= 8))
    {
        for (int m = 0; m < 3; m++)
            for (int n = 6; n < 9; n++)
                gridPotential[m][n] = false;
    }
    /// Middle left 3x3 grid
    else if ((j >= 3 && j <= 5) && k <=2)
    {
        for (int m = 3; m < 6; m++)
            for (int n = 0; n < 3; n++)
                gridPotential[m][n] = false;
```

```
    }
    /// Middle center 3x3 grid
    else if ((j >= 3 && j <= 5) && (k >= 3 && k <= 5))
    {
        for (int m = 3; m < 6; m++)
            for (int n = 3; n < 6; n++)
                gridPotential[m][n] = false;
    }
    /// Middle right 3x3 grid
    else if ((j >= 3 && j <= 5) && (k >= 6 && k <= 8))
    {
        for (int m = 3; m < 6; m++)
            for (int n = 6; n < 9; n++)
                gridPotential[m][n] = false;
    }
    /// Bottom left 3x3 grid
    else if ((j >= 6 && j <= 8) && k <=2)
    {
        for (int m = 6; m < 9; m++)
            for (int n = 0; n < 3; n++)
                gridPotential[m][n] = false;
    }
    /// Bottom center 3x3 grid
    else if ((j >= 6 && j <= 8) && (k >= 3 && k <= 5))
    {
        for (int m = 6; m < 9; m++)
            for (int n = 3; n < 6; n++)
                gridPotential[m][n] = false;
    }
    /// Bottom right 3x3 grid
    else
    {
        for (int m = 6; m < 9; m++)
            for (int n = 6; n < 9; n++)
                gridPotential[m][n] = false;
    }
}

/*****************************************************************************/
/**

 If there is only one potential space a number can occupy within a 3x3 grid,
 adds that number to the array and deletes that numbers row, column and 3x3
 grid from the list of potential squares. If there are 2 potential spaces
 within a 3x3 grid and they line up on a row or column, that number can only
 occupy that row or column within that 3x3 grid, therefore that row or column
 is eliminated.

**/
/*****************************************************************************/

void actualAddAndElimPotentialElim(int arr[][9], bool gridPotential[][9],
                                   bool &elimination, bool &change,
                                   int &leftToSolve, int &i)
{
    void display(int [][9]);
    void squareEliminator(bool [][9], int, int);

    int potentialSpaces = 0;


    /// Top left 3x3 grid
    for (int m = 0; m < 3; m++)
        for (int n = 0; n < 3; n++)
            if (gridPotential[m][n] == true)
            {
```

```
595                 potentialSpaces++;
596             }
597
598     if (potentialSpaces == 1)
599     {
600         for (int m = 0; m < 3; m++)
601             for (int n = 0; n < 3; n++)
602                 if (gridPotential[m][n] == true)
603                 {
604                     arr[m][n] = i;
605
606                     change = true;
607
608                     leftToSolve--;
609
610                     squareEliminator(gridPotential, m, n);
611
612                     elimination = true;
613
614                     display(arr);
615
616                     cout << leftToSolve << endl;
617
618                 }
619
620     }// end if (potentialSpaces == 1)
621     else if (potentialSpaces == 2)
622     {
623         potentialSpaces = 0;
624
625         /// If the 2 potential spaces line up in a row within a
626         /// 3x3 grid, that row is eliminated because it is already
627         /// taken by the number, i, in that 3x3 grid
628         for (int m = 0; m < 3; m++)
629         {
630             for (int n = 0; n < 3; n++)
631             {
632                 if (gridPotential[m][n] == true)
633                 {
634                     potentialSpaces++;
635
636                     if (potentialSpaces == 2)
637                     {
638                         for (int j = 0; j < 9; j++)
639                         {
640                             gridPotential[m][j] = false;
641
642                             elimination = true;
643                         }// end for (int j = 0; j < 9; j++)
644                     }// end if (potentialSpaces == 2)
645                 }// end if (gridPotential[m][n] == true)
646             }
647
648             potentialSpaces = 0;
649         }
650
651
652
653         /// If the 2 potential spaces line up in a column within a
654         /// 3x3 grid, that column is eliminated because it is
655         /// already taken by the number, i, in that 3x3 grid
656         for (int n = 0; n < 3; n++)
657         {
658             for (int m = 0; m < 3; m++)
659             {
660                 if (gridPotential[m][n] == true)
```

```
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)


    potentialSpaces = 0;


    /// Top center 3x3 grid
    for (int m = 0; m < 3; m++)
        for (int n = 3; n < 6; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 0; m < 3; m++)
            for (int n = 3; n < 6; n++)
                if (gridPotential[m][n] == true)
                {
                    arr[m][n] = i;

                    change = true;

                    leftToSolve--;

                    squareEliminator(gridPotential, m, n);

                    elimination = true;

                    display(arr);

                    cout << leftToSolve << endl;

                }

    }// end if (potentialSpaces == 1)
    else if (potentialSpaces == 2)
    {
        potentialSpaces = 0;

        /// If the 2 potential spaces line up in a row within a
        /// 3x3 grid, that row is eliminated because it is already
        /// taken by the number, i, in that 3x3 grid
        for (int m = 0; m < 3; m++)
        {
```

```
            for (int n = 3; n < 6; n++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[m][j] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }



        /// If the 2 potential spaces line up in a column within a
        /// 3x3 grid, that column is eliminated because it is
        /// already taken by the number, i, in that 3x3 grid
        for (int n = 3; n < 6; n++)
        {
            for (int m = 0; m < 3; m++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Top right 3x3 grid
    for (int m = 0; m < 3; m++)
        for (int n = 6; n < 9; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
```

```
793         {
794             for (int m = 0; m < 3; m++)
795                 for (int n = 6; n < 9; n++)
796                     if (gridPotential[m][n] == true)
797                     {
798                         arr[m][n] = i;
799 
800                         change = true;
801 
802                         leftToSolve--;
803 
804                         squareEliminator(gridPotential, m, n);
805 
806                         elimination = true;
807 
808                         display(arr);
809 
810                         cout << leftToSolve << endl;
811 
812                     }
813 
814         }// end if (potentialSpaces == 1)
815         else if (potentialSpaces == 2)
816         {
817             potentialSpaces = 0;
818 
819             /// If the 2 potential spaces line up in a row within a
820             /// 3x3 grid, that row is eliminated because it is already
821             /// taken by the number, i, in that 3x3 grid
822             for (int m = 0; m < 3; m++)
823             {
824                 for (int n = 6; n < 9; n++)
825                 {
826                     if (gridPotential[m][n] == true)
827                     {
828                         potentialSpaces++;
829 
830                         if (potentialSpaces == 2)
831                         {
832                             for (int j = 0; j < 9; j++)
833                             {
834                                 gridPotential[m][j] = false;
835 
836                                 elimination = true;
837                             }// end for (int j = 0; j < 9; j++)
838                         }// end if (potentialSpaces == 2)
839                     }// end if (gridPotential[m][n] == true)
840                 }
841 
842                 potentialSpaces = 0;
843             }
844 
845 
846 
847             /// If the 2 potential spaces line up in a column within a
848             /// 3x3 grid, that column is eliminated because it is
849             /// already taken by the number, i, in that 3x3 grid
850             for (int n = 6; n < 9; n++)
851             {
852                 for (int m = 0; m < 3; m++)
853                 {
854                     if (gridPotential[m][n] == true)
855                     {
856                         potentialSpaces++;
857 
858                         if (potentialSpaces == 2)
```

```
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Middle left 3x3 grid
    for (int m = 3; m < 6; m++)
        for (int n = 0; n < 3; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 3; m < 6; m++)
            for (int n = 0; n < 3; n++)
                if (gridPotential[m][n] == true)
                {
                    arr[m][n] = i;

                    change = true;

                    leftToSolve--;

                    squareEliminator(gridPotential, m, n);

                    elimination = true;

                    display(arr);

                    cout << leftToSolve << endl;

                }

    }// end if (potentialSpaces == 1)
    else if (potentialSpaces == 2)
    {
        potentialSpaces = 0;

        /// If the 2 potential spaces line up in a row within a
        /// 3x3 grid, that row is eliminated because it is already
        /// taken by the number, i, in that 3x3 grid
        for (int m = 3; m < 6; m++)
        {
            for (int n = 0; n < 3; n++)
            {
                if (gridPotential[m][n] == true)
                {
```

```
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[m][j] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }



        /// If the 2 potential spaces line up in a column within a
        /// 3x3 grid, that column is eliminated because it is
        /// already taken by the number, i, in that 3x3 grid
        for (int n = 0; n < 3; n++)
        {
            for (int m = 3; m < 6; m++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Middle center 3x3 grid
    for (int m = 3; m < 6; m++)
        for (int n = 3; n < 6; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 3; m < 6; m++)
            for (int n = 3; n < 6; n++)
                if (gridPotential[m][n] == true)
```

```
                {
                    arr[m][n] = i;

                    change = true;

                    leftToSolve--;

                    squareEliminator(gridPotential, m, n);

                    elimination = true;

                    display(arr);

                    cout << leftToSolve << endl;

                }

    }// end if (potentialSpaces == 1)
    else if (potentialSpaces == 2)
    {
        potentialSpaces = 0;

        /// If the 2 potential spaces line up in a row within a
        /// 3x3 grid, that row is eliminated because it is already
        /// taken by the number, i, in that 3x3 grid
        for (int m = 3; m < 6; m++)
        {
            for (int n = 3; n < 6; n++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[m][j] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }



        /// If the 2 potential spaces line up in a column within a
        /// 3x3 grid, that column is eliminated because it is
        /// already taken by the number, i, in that 3x3 grid
        for (int n = 3; n < 6; n++)
        {
            for (int m = 3; m < 6; m++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;
```

```
                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Middle right 3x3 grid
    for (int m = 3; m < 6; m++)
        for (int n = 6; n < 9; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 3; m < 6; m++)
            for (int n = 6; n < 9; n++)
                if (gridPotential[m][n] == true)
                {
                    arr[m][n] = i;

                    change = true;

                    leftToSolve--;

                    squareEliminator(gridPotential, m, n);

                    elimination = true;

                    display(arr);

                    cout << leftToSolve << endl;

                }

    }// end if (potentialSpaces == 1)
    else if (potentialSpaces == 2)
    {
        potentialSpaces = 0;

        /// If the 2 potential spaces line up in a row within a
        /// 3x3 grid, that row is eliminated because it is already
        /// taken by the number, i, in that 3x3 grid
        for (int m = 3; m < 6; m++)
        {
            for (int n = 6; n < 9; n++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
```

```
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[m][j] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }



        /// If the 2 potential spaces line up in a column within a
        /// 3x3 grid, that column is eliminated because it is
        /// already taken by the number, i, in that 3x3 grid
        for (int n = 6; n < 9; n++)
        {
            for (int m = 3; m < 6; m++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Bottom left 3x3 grid
    for (int m = 6; m < 9; m++)
        for (int n = 0; n < 3; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 6; m < 9; m++)
            for (int n = 0; n < 3; n++)
                if (gridPotential[m][n] == true)
                {
                    arr[m][n] = i;

                    change = true;
```

```
1189
1190                    leftToSolve--;
1191
1192                    squareEliminator(gridPotential, m, n);
1193
1194                    elimination = true;
1195
1196                    display(arr);
1197
1198                    cout << leftToSolve << endl;
1199
1200                }
1201
1202    }// end if (potentialSpaces == 1)
1203    else if (potentialSpaces == 2)
1204    {
1205        potentialSpaces = 0;
1206
1207        /// If the 2 potential spaces line up in a row within a
1208        /// 3x3 grid, that row is eliminated because it is already
1209        /// taken by the number, i, in that 3x3 grid
1210        for (int m = 6; m < 9; m++)
1211        {
1212            for (int n = 0; n < 3; n++)
1213            {
1214                if (gridPotential[m][n] == true)
1215                {
1216                    potentialSpaces++;
1217
1218                    if (potentialSpaces == 2)
1219                    {
1220                        for (int j = 0; j < 9; j++)
1221                        {
1222                            gridPotential[m][j] = false;
1223
1224                            elimination = true;
1225                        }// end for (int j = 0; j < 9; j++)
1226                    }// end if (potentialSpaces == 2)
1227                }// end if (gridPotential[m][n] == true)
1228            }
1229
1230            potentialSpaces = 0;
1231        }
1232
1233
1234
1235        /// If the 2 potential spaces line up in a column within a
1236        /// 3x3 grid, that column is eliminated because it is
1237        /// already taken by the number, i, in that 3x3 grid
1238        for (int n = 0; n < 3; n++)
1239        {
1240            for (int m = 6; m < 9; m++)
1241            {
1242                if (gridPotential[m][n] == true)
1243                {
1244                    potentialSpaces++;
1245
1246                    if (potentialSpaces == 2)
1247                    {
1248                        for (int j = 0; j < 9; j++)
1249                        {
1250                            gridPotential[j][n] = false;
1251
1252                            elimination = true;
1253                        }// end for (int j = 0; j < 9; j++)
1254                    }// end if (potentialSpaces == 2)
```

```
1255                }// end if (gridPotential[m][n] == true)
1256            }
1257
1258            potentialSpaces = 0;
1259        }
1260
1261    }// end else if (potentialSpaces == 2)
1262
1263
1264
1265    potentialSpaces = 0;
1266
1267
1268
1269    /// Bottom center 3x3 grid
1270    for (int m = 6; m < 9; m++)
1271        for (int n = 3; n < 6; n++)
1272            if (gridPotential[m][n] == true)
1273            {
1274                potentialSpaces++;
1275            }
1276
1277    if (potentialSpaces == 1)
1278    {
1279        for (int m = 6; m < 9; m++)
1280            for (int n = 3; n < 6; n++)
1281                if (gridPotential[m][n] == true)
1282                {
1283                    arr[m][n] = i;
1284
1285                    change = true;
1286
1287                    leftToSolve--;
1288
1289                    squareEliminator(gridPotential, m, n);
1290
1291                    elimination = true;
1292
1293                    display(arr);
1294
1295                    cout << leftToSolve << endl;
1296
1297                }
1298
1299    }// end if (potentialSpaces == 1)
1300    else if (potentialSpaces == 2)
1301    {
1302        potentialSpaces = 0;
1303
1304        /// If the 2 potential spaces line up in a row within a
1305        /// 3x3 grid, that row is eliminated because it is already
1306        /// taken by the number, i, in that 3x3 grid
1307        for (int m = 6; m < 9; m++)
1308        {
1309            for (int n = 3; n < 6; n++)
1310            {
1311                if (gridPotential[m][n] == true)
1312                {
1313                    potentialSpaces++;
1314
1315                    if (potentialSpaces == 2)
1316                    {
1317                        for (int j = 0; j < 9; j++)
1318                        {
1319                            gridPotential[m][j] = false;
1320
```

```
                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }



        /// If the 2 potential spaces line up in a column within a
        /// 3x3 grid, that column is eliminated because it is
        /// already taken by the number, i, in that 3x3 grid
        for (int n = 3; n < 6; n++)
        {
            for (int m = 6; m < 9; m++)
            {
                if (gridPotential[m][n] == true)
                {
                    potentialSpaces++;

                    if (potentialSpaces == 2)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            gridPotential[j][n] = false;

                            elimination = true;
                        }// end for (int j = 0; j < 9; j++)
                    }// end if (potentialSpaces == 2)
                }// end if (gridPotential[m][n] == true)
            }

            potentialSpaces = 0;
        }

    }// end else if (potentialSpaces == 2)



    potentialSpaces = 0;



    /// Bottom right 3x3 grid
    for (int m = 6; m < 9; m++)
        for (int n = 6; n < 9; n++)
            if (gridPotential[m][n] == true)
            {
                potentialSpaces++;
            }

    if (potentialSpaces == 1)
    {
        for (int m = 6; m < 9; m++)
            for (int n = 6; n < 9; n++)
                if (gridPotential[m][n] == true)
                {
                    arr[m][n] = i;

                    change = true;

                    leftToSolve--;

                    squareEliminator(gridPotential, m, n);
```

```
1387
1388                    elimination = true;
1389
1390                    display(arr);
1391
1392                    cout << leftToSolve << endl;
1393
1394                }
1395
1396    }// end if (potentialSpaces == 1)
1397    else if (potentialSpaces == 2)
1398    {
1399        potentialSpaces = 0;
1400
1401        /// If the 2 potential spaces line up in a row within a
1402        /// 3x3 grid, that row is eliminated because it is already
1403        /// taken by the number, i, in that 3x3 grid
1404        for (int m = 6; m < 9; m++)
1405        {
1406            for (int n = 6; n < 9; n++)
1407            {
1408                if (gridPotential[m][n] == true)
1409                {
1410                    potentialSpaces++;
1411
1412                    if (potentialSpaces == 2)
1413                    {
1414                        for (int j = 0; j < 9; j++)
1415                        {
1416                            gridPotential[m][j] = false;
1417
1418                            elimination = true;
1419                        }// end for (int j = 0; j < 9; j++)
1420                    }// end if (potentialSpaces == 2)
1421                }// end if (gridPotential[m][n] == true)
1422            }
1423
1424            potentialSpaces = 0;
1425        }
1426
1427
1428
1429        /// If the 2 potential spaces line up in a column within a
1430        /// 3x3 grid, that column is eliminated because it is
1431        /// already taken by the number, i, in that 3x3 grid
1432        for (int n = 6; n < 9; n++)
1433        {
1434            for (int m = 6; m < 9; m++)
1435            {
1436                if (gridPotential[m][n] == true)
1437                {
1438                    potentialSpaces++;
1439
1440                    if (potentialSpaces == 2)
1441                    {
1442                        for (int j = 0; j < 9; j++)
1443                        {
1444                            gridPotential[j][n] = false;
1445
1446                            elimination = true;
1447                        }// end for (int j = 0; j < 9; j++)
1448                    }// end if (potentialSpaces == 2)
1449                }// end if (gridPotential[m][n] == true)
1450            }
1451
1452            potentialSpaces = 0;
```

```
        }

    }// end else if (potentialSpaces == 2)


}// end void actualAddAndElimPotentialElim(int arr[][9],
//                                        bool gridPotential[][9],
//                                        bool &elimination, bool &change,
//                                        int &leftToSolve)


/*****************************************************************************/
/**

 Applies brute force to the array to solve it. Will solve any puzzle.

 At each empty square, takes a randomly selected potential number of
 that square and tests to see if it was already used in the row,
 column or 3x3 grid. If it was not used, adds it to the square and goes
 to the next empty square. If it was used, picks another randomly
 selected potential number of that square and tests it. If all the
 potential numbers of an empty square are already taken, a new brute
 force attempt is started. Brute force is applied until the puzzle
 is solved.

**/
/*****************************************************************************/


bool bruteForce(int arr[][9])
{
    int numberOfPotentials[9][9], potentialNumbers[9][9][9], randomPotential,
        arrCopy[9][9], numbersTriedCount;
    bool gridPotential[9][9], repeat, usedNumbers[9], taken, run;

    void squareEliminator(bool [][9], int, int);
    void display(int [][9]);


    for (int m = 0; m < 9; m++)
        for (int n = 0; n < 9; n++)
        {
            numberOfPotentials[m][n] = 0;
        }



    /// Takes each number and checks for all the rows, columns and 3x3 grids
    /// that are already occupied by it on the board. It then adds that
    /// number to the next available slot of all other squares in a 3
    /// dimensional array and keeps track of how many potential numbers are on
    /// each square.
    for (int i = 1; i <= 9; i++)
    {

        /// Initialize all the squares to false if they are occupied
        /// and true if they are empty
        for (int m = 0; m < 9; m++)
            for (int n = 0; n < 9; n++)
                if (arr[m][n] > 0)
                {
                    gridPotential[m][n] = false;
```

```
1519                }
1520                else
1521                {
1522                    gridPotential[m][n] = true;
1523                }
1524
1525
1526
1527
1528            for (int j = 0; j < 9; j++)
1529                for (int k = 0; k < 9; k++)
1530                {
1531
1532                    if (arr[j][k] == i)
1533                    {
1534                        /// Eliminates the column, row and 3x3 grid the
1535                        /// number occupies
1536                        squareEliminator(gridPotential, j, k);
1537
1538
1539
1540                        /// This row has already been eliminated, start at the
1541                        /// beginning of the next row. k is negative one
1542                        /// because it will be incremented to 0 at the start of
1543                        /// it's for loop. If j equals 8, incrementing j would
1544                        /// cause the next set of test parameters to be
1545                        /// 9 and 0 which is outside the range of the array
1546                        if (j != 8)
1547                        {
1548                            k = -1;
1549                            j++;
1550                        }
1551                    }// end if (arr[j][k] == i)
1552                }
1553
1554
1555
1556
1557            /// Adds the number to the next available slot on each square that was
1558            /// not part of a row, column or 3x3 grid that was already occupied
1559            /// by the number and increments the count for each square the number
1560            /// is added to
1561            for (int j = 0; j < 9; j++)
1562                for (int k = 0; k < 9; k++)
1563                    if (gridPotential[j][k] == true)
1564                    {
1565                        potentialNumbers[j][k][numberOfPotentials[j][k]] = i;
1566
1567                        numberOfPotentials[j][k]++;
1568                    }
1569
1570
1571
1572        }// end for (int i = 1; i <= 9; i++)
1573
1574        cout << endl;
1575        cout << "Brute force being applied, please wait..." << endl << endl;
1576
1577
1578
1579        /// Seed random number generator with current time
1580        srand(time(0));
1581
1582
1583
1584        /// At each empty square, takes a randomly selected potential number of
```

```
1585        /// that square and tests to see if it was already used in the row,
1586        /// column or 3x3 grid. If it was not used, adds it to the square and goes
1587        /// to the next empty square. If it was used, picks another randomly
1588        /// selected potential number of that square and tests it. If all the
1589        /// potential numbers of an empty square are already taken, a new brute
1590        /// force attempt is started. Brute force is applied until the puzzle
1591        /// is solved.
1592        for (int j = 0; j < 9; j++)
1593        {
1594            for (int k = 0; k < 9; k++)
1595            {
1596
1597                /// Initializes arrCopy to arr's state at the start of each brute
1598                /// force attempt
1599                if (j == 0 && k == 0)
1600                {
1601                    for (int m = 0; m < 9; m++)
1602                        for (int n = 0; n < 9; n++)
1603                        {
1604                            arrCopy[m][n] = arr[m][n];
1605                        }
1606
1607                }// end if (j == 0 && k == 0)
1608
1609
1610
1611                if (arrCopy[j][k] == 0)
1612                {
1613                    for (int m = 0; m < 9; m++)
1614                    {
1615                        usedNumbers[m] = false;
1616                    }
1617
1618                    repeat = false;
1619
1620                    numbersTriedCount = 0;
1621
1622                    do
1623                    {
1624
1625                        do
1626                        {
1627                            randomPotential =
1628                                potentialNumbers
1629                                [j][k][rand() % numberOfPotentials[j][k]];
1630
1631                            if (usedNumbers[randomPotential - 1] == false)
1632                            {
1633                                repeat = false;
1634
1635                                run = true;
1636
1637                                usedNumbers[randomPotential - 1] = true;
1638
1639                                numbersTriedCount++;
1640
1641
1642                            }// end if (usedNumbers[randomPotential - 1] == false)
1643                            else
1644                            {
1645                                repeat = true;
1646
1647                                /// Ends this brute force attempt
1648                                if (numbersTriedCount == numberOfPotentials[j][k])
1649                                {
1650                                    j = 0;
```

```
1651
1652                                k = -1;
1653
1654                                /// Breaks out of the do while loops
1655
1656                                repeat = false;
1657
1658                                run = false;
1659
1660                                taken = false;
1661                            }
1662                        }
1663
1664                    }
1665                    while(repeat);
1666
1667
1668                    if (run)
1669                    {
1670
1671
1672                        taken = false;
1673
1674                        /// Checks the row of the square to see if the number
1675                        /// was already used
1676                        for (int i = 0; i < 9; i++)
1677                        {
1678                            if (arrCopy[j][i] == randomPotential)
1679                            {
1680                                taken = true;
1681                            }// end if (arrCopy[j][i] == randomPotential)
1682
1683                        }// end for (int i = 0; i < 9; i++)
1684
1685
1686                        if (!taken)
1687                        {
1688
1689                            /// Checks the column of the square to see if the
1690                            /// number was already used
1691                            for (int i = 0; i < 9; i++)
1692                            {
1693                                if (arrCopy[i][k] == randomPotential)
1694                                {
1695                                    taken = true;
1696
1697                                }// end if (arrCopy[i][k] == randomPotential)
1698
1699                            }// end for (int i = 0; i < 9; i++)
1700
1701
1702
1703                            if (!taken)
1704                            {
1705
1706                                /// This section checks within the 3 x 3 grid
1707                                /// of the square being tested to see if the
1708                                /// number was already used
1709
1710                                /// Top left 3x3 grid
1711                                if (j <= 2 && k <=2)
1712                                {
1713                                    for (int m = 0; m < 3; m++)
1714                                        for (int n = 0; n < 3; n++)
1715                                            if
1716                                            (arrCopy[m][n] == randomPotential)
```

```
                    {
                        taken = true;

                    }
                }
                /// Top center 3x3 grid
                else if (j <= 2 && (k >= 3 && k <= 5))
                {
                    for (int m = 0; m < 3; m++)
                        for (int n = 3; n < 6; n++)
                            if
                            (arrCopy[m][n] == randomPotential)
                            {
                                taken = true;

                            }
                }
                /// Top right 3x3 grid
                else if (j <= 2 && (k >= 6 && k <= 8))
                {
                    for (int m = 0; m < 3; m++)
                        for (int n = 6; n < 9; n++)
                            if
                            (arrCopy[m][n] == randomPotential)
                            {
                                taken = true;

                            }
                }
                /// Middle left 3x3 grid
                else if ((j >= 3 && j <= 5) && k <=2)
                {
                    for (int m = 3; m < 6; m++)
                        for (int n = 0; n < 3; n++)
                            if
                            (arrCopy[m][n] == randomPotential)
                            {
                                taken = true;

                            }
                }
                /// Middle center 3x3 grid
                else if ((j >= 3 && j <= 5) &&
                         (k >= 3 && k <= 5))
                {
                    for (int m = 3; m < 6; m++)
                        for (int n = 3; n < 6; n++)
                            if
                            (arrCopy[m][n] == randomPotential)
                            {
                                taken = true;

                            }
                }
                /// Middle right 3x3 grid
                else if ((j >= 3 && j <= 5) &&
                         (k >= 6 && k <= 8))
                {
                    for (int m = 3; m < 6; m++)
                        for (int n = 6; n < 9; n++)
                            if
                            (arrCopy[m][n] == randomPotential)
                            {
                                taken = true;

                            }
```

```
                                }
                                /// Bottom left 3x3 grid
                                else if ((j >= 6 && j <= 8) && k <=2)
                                {
                                    for (int m = 6; m < 9; m++)
                                        for (int n = 0; n < 3; n++)
                                            if
                                            (arrCopy[m][n] == randomPotential)
                                            {
                                                taken = true;

                                            }
                                }
                                /// Bottom center 3x3 grid
                                else if ((j >= 6 && j <= 8) &&
                                         (k >= 3 && k <= 5))
                                {
                                    for (int m = 6; m < 9; m++)
                                        for (int n = 3; n < 6; n++)
                                            if
                                            (arrCopy[m][n] == randomPotential)
                                            {
                                                taken = true;

                                            }
                                }
                                /// Bottom right 3x3 grid
                                else
                                {
                                    for (int m = 6; m < 9; m++)
                                        for (int n = 6; n < 9; n++)
                                            if
                                            (arrCopy[m][n] == randomPotential)
                                            {
                                                taken = true;

                                            }
                                }



                                /// If the number is not taken in the row,
                                /// column, or 3x3 grid, it is placed on
                                /// the square
                                if (!taken)
                                {
                                    arrCopy[j][k] = randomPotential;
                                }


                            }// end if (!taken)

                        }// end if (!taken)

                    }// end if (run)

                }
                while (taken);

            }// end if (arrCopy[j][k] == 0)

        }// end for (int k = 0; k < 9; k++)
    }// end for (int j = 0; j < 9; j++)

    display(arrCopy);

```

```
1849        return true;
1850
1851    }// end bool bruteForce(int arr[][9])
1852
1853    /*****************************************************************************/
1854
1855    int main()
1856    {
1857        int board[9][9], numToSolve = 0, startingFilled;
1858
1859        bool fillBoard(int [][9], int &);
1860        bool solve(int [][9], int &);
1861        bool bruteForce(int [][9]);
1862
1863
1864        if(!fillBoard(board, numToSolve))
1865        {
1866            cout << "what?" << endl;
1867        }
1868        else
1869        {
1870            startingFilled = 81 - numToSolve;
1871
1872            if(!solve(board, numToSolve))
1873            {
1874                cout << "Oh, no!" << endl;
1875
1876                if (bruteForce(board))
1877                    cout << "Brute force for the win!" << endl << endl;
1878            }
1879
1880            cout << "Number filled at start: " << startingFilled << endl;
1881        }
1882
1883
1884        return 0;
1885    }
```