

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.

	A	B	C	D	E	F	G	H	I
A				6	7	4		8	
B		8		5	3	1	7	9	2
C				9	2	8	6		4
D	8		2				9		
E		1				2		6	8
F			5	8			4	2	7
G	1	4	8	3	6	9	2	7	5
H				1	4	5	8	3	9
I	5	9	3	2	8	7	1		6

USING A COMBINATION of different TACTICS in a loop is a good way to go about this. Finding the potential numbers a square can be by eliminating numbers in it's row, column and 3x3 grid can sometimes immediately lead you to an answer.

→ this square we can say starts out having 9 potential numbers, 1-9.

1 2 3 4 5 6 7 8 9

First we eliminate all the numbers from it's vertical column to be left with 3 numbers

~~1~~ 4 5

Then we eliminate from it's horizontal row, then it's 3x3 Grid if necessary (in any order), in this case we are left with 4 for this square. by elimination the other square in the 3x3 Grid can only be 8

The next strategy is a combination of tactics that is used in a loop with the one above. In it you eliminate all the rows and columns that is already occupied by a specific number AND columns and rows that are already occupied by a potential number, as in, in a 3x3 Grid a potential number can only occupy two different squares potentially and the squares line up. What ever column or row they line up on can only be occupied by that potential number in that 3x3 square. For example, using the number 1, D is eliminated. so are E and F. G is also eliminated, leaving the potential places 1 can occupy in the middle right 3x3 grid only along H, so it is eliminated. From what we have already eliminated, we see that (F, H) is 1.

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: 17 ~ 6:9: ~ 1: 1 ~
~ 9: 1 ~ 3:5: ~ 1:7: ~
=====
43
=====
~ 14:9 ~ 8: 15 ~ 1:1 ~
~ 1: 11 ~ 4: 16 ~ 9: 1 ~
~ 18: ~ 9:2:1 ~ 1: 14 ~
=====
~ 1:1 ~ 2: 19 ~ 1: 15 ~
~ 1: 1 ~ 1: 1 ~ 19: ~
~ 8:9: ~ 5: 1 ~ 14:1 ~
=====
~ 5: 1 ~ 7:1: ~ 12:9 ~
~ 1: 17 ~ 6:9: ~ 1: 1 ~
~ 9: 1 ~ 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.

```

1  /**
2   Name: Sudoku
3   Copyright:
4   Author: Jeremy Alexandre
5   Date:
6   Description: Sudoku solver that will solve any Sudoku puzzle. Input is
7   received through infile stream, from a file named "data.txt". The format
8   of the input file is space for an empty square and number for a square with
9   that number.
10
11   First it uses certain logic methods to try to solve the puzzle. If that fails,
12   it uses brute force to solve the puzzle.
13
14   Here is an example "data.txt" file, copy from after open quotation mark to
15   before close quotation. Don't forget to add spaces so you have 9 sections
16   to a row:
17
18   "53  7
19   6  195
20   98    6
21   8    6  3
22   4  8 3  1
23   7    2  6
24   6      28
25     419  5
26      8  79"
27
28   The following is an example of a puzzle that will cause brute force to be
29   applied:
30
31   "    74
32     1 563
33   2    94
34   9    31
35     6    8
36   32    7
37   82    4
38 371 9
39     57   "
40
41  */
42
43  #include <cstdlib>
44  #include <iostream>
45  #include <cctype>
46  #include <fstream>
47  #include <ctime>
48
49  using namespace std;
50
51  ifstream infile;    //creates an instream file
52
53
54
55  /*****
56  /**
57
58   Fills the array from infile, placing numbers for numbers and zeros for
59   spaces. When ever a space is encountered, toSolve also get incremented.
60   Function then returns true. If there are missing numbers or spaces, sends
61   the row and number of missing numbers and/or spaces to cout and returns
62   false. If there is an invalid character, sends the invalid character to
63   cout and returns false.
64
65  **/
66  /*****/

```

```

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

```

```

133     Receives the array filled with the puzzle and attempts to solve it using
134     logic. If an answer to a square is discovered it is added to the array. If
135     the puzzle is solved, returns true. If the puzzle cannot be solved using
136     these logic methods, returns false.
137
138     */
139     /*****
140
141
142
143     bool solve(int arr[][9], int &leftToSolve)
144     {
145         bool change, potential[9], elimination, gridPotential[9][9];
146         int potentialLeft;
147
148         void display(int [][][9]);
149         void squareEliminator(bool [][][9], int, int);
150         void actualAddAndElimPotentialElim(int arr[][9], bool gridPotential[][9],
151                                             bool &elimination, bool &change,
152                                             int &leftToSolve, int &i);
153
154
155
156         do
157         {
158             change = false;
159
160             for (int j = 0; j < 9; j++)
161             {
162                 for (int k = 0; k < 9; k++)
163                 {
164                     if (arr[j][k] == 0)
165                     {
166                         // Before any numbers are eliminated, there are 9 potential
167                         // numbers, the numbers 1-9
168                         for (int i = 0; i < 9; i++)
169                             potential[i] = true;
170
171                         potentialLeft = 9;
172
173
174                         // Checks the row of the square to see which numbers where
175                         // already used and therefore not a potential number for
176                         // the square
177                         for (int i = 0; i < 9; i++)
178                         {
179                             if (arr[j][i] > 0)
180                             {
181                                 potential[arr[j][i] - 1] = false;
182
183                                 potentialLeft--;
184                             } // end if (arr[j][i] > 0)
185                         } // end for (int i = 0; i < 9; i++)
186
187
188
189                         // Checks the column of the square to see which numbers
190                         // where already used and therefore not a potential number
191                         // for the square. It also checks to make sure the number
192                         // has not already been eliminated.
193                         for (int i = 0; i < 9; i++)
194                         {
195                             if (arr[i][k] > 0)
196                             {
197                                 if (potential[arr[i][k] - 1] == true)
198                                 {

```

```

199         potential[arr[i][k] - 1] = false;
200
201         potentialLeft--;
202     } // end if (arr[i][k] > 0)
203
204 } // end for (int i = 0; i < 9; i++)
205
206
207
208 /// This section checks within the 3 x 3 grid of the square
209 /// being tested to see which numbers are taken, and
210 /// therefore not a potential number for the square.
211 /// As before, it also checks to make sure the number
212 /// has not already been eliminated.
213
214 /// Top left 3x3 grid
215 if (j <= 2 && k <= 2)
216 {
217     for (int m = 0; m < 3; m++)
218     for (int n = 0; n < 3; n++)
219     if (arr[m][n] > 0)
220     if (potential[arr[m][n] - 1] == true)
221     {
222         potential[arr[m][n] - 1] = false;
223
224         potentialLeft--;
225     }
226 }
227 /// Top center 3x3 grid
228 else if (j <= 2 && (k >= 3 && k <= 5))
229 {
230     for (int m = 0; m < 3; m++)
231     for (int n = 3; n < 6; n++)
232     if (arr[m][n] > 0)
233     if (potential[arr[m][n] - 1] == true)
234     {
235         potential[arr[m][n] - 1] = false;
236
237         potentialLeft--;
238     }
239 }
240 /// Top right 3x3 grid
241 else if (j <= 2 && (k >= 6 && k <= 8))
242 {
243     for (int m = 0; m < 3; m++)
244     for (int n = 6; n < 9; n++)
245     if (arr[m][n] > 0)
246     if (potential[arr[m][n] - 1] == true)
247     {
248         potential[arr[m][n] - 1] = false;
249
250         potentialLeft--;
251     }
252 }
253 /// Middle left 3x3 grid
254 else if ((j >= 3 && j <= 5) && k <= 2)
255 {
256     for (int m = 3; m < 6; m++)
257     for (int n = 0; n < 3; n++)
258     if (arr[m][n] > 0)
259     if (potential[arr[m][n] - 1] == true)
260     {
261         potential[arr[m][n] - 1] = false;
262
263         potentialLeft--;
264     }

```

```

265     }
266     /// Middle center 3x3 grid
267     else if ((j >= 3 && j <= 5) && (k >= 3 && k <= 5))
268     {
269         for (int m = 3; m < 6; m++)
270             for (int n = 3; n < 6; n++)
271                 if (arr[m][n] > 0)
272                     if (potential[arr[m][n] - 1] == true)
273                     {
274                         potential[arr[m][n] - 1] = false;
275
276                         potentialLeft--;
277                     }
278     }
279     /// Middle right 3x3 grid
280     else if ((j >= 3 && j <= 5) && (k >= 6 && k <= 8))
281     {
282         for (int m = 3; m < 6; m++)
283             for (int n = 6; n < 9; n++)
284                 if (arr[m][n] > 0)
285                     if (potential[arr[m][n] - 1] == true)
286                     {
287                         potential[arr[m][n] - 1] = false;
288
289                         potentialLeft--;
290                     }
291     }
292     /// Bottom left 3x3 grid
293     else if ((j >= 6 && j <= 8) && k <= 2)
294     {
295         for (int m = 6; m < 9; m++)
296             for (int n = 0; n < 3; n++)
297                 if (arr[m][n] > 0)
298                     if (potential[arr[m][n] - 1] == true)
299                     {
300                         potential[arr[m][n] - 1] = false;
301
302                         potentialLeft--;
303                     }
304     }
305     /// Bottom center 3x3 grid
306     else if ((j >= 6 && j <= 8) && (k >= 3 && k <= 5))
307     {
308         for (int m = 6; m < 9; m++)
309             for (int n = 3; n < 6; n++)
310                 if (arr[m][n] > 0)
311                     if (potential[arr[m][n] - 1] == true)
312                     {
313                         potential[arr[m][n] - 1] = false;
314
315                         potentialLeft--;
316                     }
317     }
318     /// Bottom right 3x3 grid
319     else
320     {
321         for (int m = 6; m < 9; m++)
322             for (int n = 6; n < 9; n++)
323                 if (arr[m][n] > 0)
324                     if (potential[arr[m][n] - 1] == true)
325                     {
326                         potential[arr[m][n] - 1] = false;
327
328                         potentialLeft--;
329                     }
330     }

```

```

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)

```



```

397         {
398             k = -1;
399             j++;
400         }
401     } // end if (arr[j][k] == i)
402 }
403
404
405     do
406     {
407         elimination = false;
408
409         actualAddAndElimPotentialElim(arr, gridPotential, elimination,
410                                     change, leftToSolve, i);
411
412     }
413     while (elimination);
414
415 } // end for (int i = 1; i <= 9; i++)
416
417 }
418 while (change);
419
420 if (leftToSolve == 0)
421 {
422     cout << "Solved!" << endl;
423     return true;
424 }
425 else
426 {
427     cout << "Could not solve!" << endl;
428     return false;
429 }
430
431 } // end bool solve(int arr[][9], int &leftToSolve)
432
433
434
435
436 /*****
437 **
438
439 Displays the puzzle.
440
441 **/
442 /*****
443
444
445
446 void display(int arr[][9])
447 {
448     cout << "===== " << endl;
449
450     for (int j = 0; j < 9; j++)
451     {
452         cout << "~ ";
453         for (int k = 0; k < 9; k++)
454         {
455             if (arr[j][k] == 0)
456                 cout << " ";
457             else
458                 cout << arr[j][k];
459             if ((k % 3) != 2)
460                 cout << "| ";
461             else if (k != 8)
462                 cout << " ~ ";

```

```

463         else
464             cout << " ~";
465     } // end for (int k = 0; k < 9; k++)
466
467     cout << endl;
468
469     if ((j % 3) != 2)
470         cout << "-----" << endl;
471     else cout << "===== " << endl;
472
473 } // end for (int j = 0; j < 3; j++)
474
475 }
476
477
478
479
480 /*****
481 **
482
483     Eliminates the column, row and 3x3 square the number occupies
484
485 **/
486 /*****
487
488
489
490 void squareEliminator(bool gridPotential[][9], int j, int k)
491 {
492     /// Eliminates both the column and row the number
493     /// occupies
494     for (int m = 0; m < 9; m++)
495     {
496         gridPotential[j][m] = false;
497         gridPotential[m][k] = false;
498     }
499
500     /// Eliminates the 3x3 grid the number occupies
501
502     /// Top left 3x3 grid
503     if (j <= 2 && k <= 2)
504     {
505         for (int m = 0; m < 3; m++)
506             for (int n = 0; n < 3; n++)
507                 gridPotential[m][n] = false;
508     }
509     /// Top center 3x3 grid
510     else if (j <= 2 && (k >= 3 && k <= 5))
511     {
512         for (int m = 0; m < 3; m++)
513             for (int n = 3; n < 6; n++)
514                 gridPotential[m][n] = false;
515     }
516     /// Top right 3x3 grid
517     else if (j <= 2 && (k >= 6 && k <= 8))
518     {
519         for (int m = 0; m < 3; m++)
520             for (int n = 6; n < 9; n++)
521                 gridPotential[m][n] = false;
522     }
523     /// Middle left 3x3 grid
524     else if ((j >= 3 && j <= 5) && k <= 2)
525     {
526         for (int m = 3; m < 6; m++)
527             for (int n = 0; n < 3; n++)
528                 gridPotential[m][n] = false;

```

```

529     }
530     /// Middle center 3x3 grid
531     else if ((j >= 3 && j <= 5) && (k >= 3 && k <= 5))
532     {
533         for (int m = 3; m < 6; m++)
534             for (int n = 3; n < 6; n++)
535                 gridPotential[m][n] = false;
536     }
537     /// Middle right 3x3 grid
538     else if ((j >= 3 && j <= 5) && (k >= 6 && k <= 8))
539     {
540         for (int m = 3; m < 6; m++)
541             for (int n = 6; n < 9; n++)
542                 gridPotential[m][n] = false;
543     }
544     /// Bottom left 3x3 grid
545     else if ((j >= 6 && j <= 8) && k <= 2)
546     {
547         for (int m = 6; m < 9; m++)
548             for (int n = 0; n < 3; n++)
549                 gridPotential[m][n] = false;
550     }
551     /// Bottom center 3x3 grid
552     else if ((j >= 6 && j <= 8) && (k >= 3 && k <= 5))
553     {
554         for (int m = 6; m < 9; m++)
555             for (int n = 3; n < 6; n++)
556                 gridPotential[m][n] = false;
557     }
558     /// Bottom right 3x3 grid
559     else
560     {
561         for (int m = 6; m < 9; m++)
562             for (int n = 6; n < 9; n++)
563                 gridPotential[m][n] = false;
564     }
565 }
566
567 /*****
568 **
569
570 If there is only one potential space a number can occupy within a 3x3 grid,
571 adds that number to the array and deletes that numbers row, column and 3x3
572 grid from the list of potential squares. If there are 2 potential spaces
573 within a 3x3 grid and they line up on a row or column, that number can only
574 occupy that row or column within that 3x3 grid, therefore that row or column
575 is eliminated.
576
577 **/
578 /*****
579
580 void actualAddAndElimPotentialElim(int arr[][9], bool gridPotential[][9],
581                                   bool &elimination, bool &change,
582                                   int &leftToSolve, int &i)
583 {
584     void display(int [][][9]);
585     void squareEliminator(bool [][][9], int, int);
586
587     int potentialSpaces = 0;
588
589
590     /// Top left 3x3 grid
591     for (int m = 0; m < 3; m++)
592         for (int n = 0; n < 3; n++)
593             if (gridPotential[m][n] == true)
594                 {

```

```

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)

```

```

661         {
662             potentialSpaces++;
663
664             if (potentialSpaces == 2)
665             {
666                 for (int j = 0; j < 9; j++)
667                 {
668                     gridPotential[j][n] = false;
669
670                     elimination = true;
671                 } // end for (int j = 0; j < 9; j++)
672             } // end if (potentialSpaces == 2)
673             } // end if (gridPotential[m][n] == true)
674         }
675
676         potentialSpaces = 0;
677     }
678
679 } // end else if (potentialSpaces == 2)
680
681
682
683 potentialSpaces = 0;
684
685
686
687 /// Top center 3x3 grid
688 for (int m = 0; m < 3; m++)
689     for (int n = 3; n < 6; n++)
690         if (gridPotential[m][n] == true)
691         {
692             potentialSpaces++;
693         }
694
695 if (potentialSpaces == 1)
696 {
697     for (int m = 0; m < 3; m++)
698         for (int n = 3; n < 6; n++)
699             if (gridPotential[m][n] == true)
700             {
701                 arr[m][n] = i;
702
703                 change = true;
704
705                 leftToSolve--;
706
707                 squareEliminator(gridPotential, m, n);
708
709                 elimination = true;
710
711                 display(arr);
712
713                 cout << leftToSolve << endl;
714
715             }
716
717 } // end if (potentialSpaces == 1)
718 else if (potentialSpaces == 2)
719 {
720     potentialSpaces = 0;
721
722     /// If the 2 potential spaces line up in a row within a
723     /// 3x3 grid, that row is eliminated because it is already
724     /// taken by the number, i, in that 3x3 grid
725     for (int m = 0; m < 3; m++)
726     {

```

```

727     for (int n = 3; n < 6; n++)
728     {
729         if (gridPotential[m][n] == true)
730         {
731             potentialSpaces++;
732
733             if (potentialSpaces == 2)
734             {
735                 for (int j = 0; j < 9; j++)
736                 {
737                     gridPotential[m][j] = false;
738
739                     elimination = true;
740                 } // end for (int j = 0; j < 9; j++)
741             } // end if (potentialSpaces == 2)
742         } // end if (gridPotential[m][n] == true)
743     }
744
745     potentialSpaces = 0;
746 }
747
748
749
750 /// If the 2 potential spaces line up in a column within a
751 /// 3x3 grid, that column is eliminated because it is
752 /// already taken by the number, i, in that 3x3 grid
753 for (int n = 3; n < 6; n++)
754 {
755     for (int m = 0; m < 3; m++)
756     {
757         if (gridPotential[m][n] == true)
758         {
759             potentialSpaces++;
760
761             if (potentialSpaces == 2)
762             {
763                 for (int j = 0; j < 9; j++)
764                 {
765                     gridPotential[j][n] = false;
766
767                     elimination = true;
768                 } // end for (int j = 0; j < 9; j++)
769             } // end if (potentialSpaces == 2)
770         } // end if (gridPotential[m][n] == true)
771     }
772
773     potentialSpaces = 0;
774 }
775
776 } // end else if (potentialSpaces == 2)
777
778
779
780 potentialSpaces = 0;
781
782
783
784 /// Top right 3x3 grid
785 for (int m = 0; m < 3; m++)
786     for (int n = 6; n < 9; n++)
787         if (gridPotential[m][n] == true)
788         {
789             potentialSpaces++;
790         }
791
792 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)

```

```

859         {
860             for (int j = 0; j < 9; j++)
861             {
862                 gridPotential[j][n] = false;
863
864                 elimination = true;
865             } // end for (int j = 0; j < 9; j++)
866         } // end if (potentialSpaces == 2)
867     } // end if (gridPotential[m][n] == true)
868 }
869
870     potentialSpaces = 0;
871 }
872
873 } // end else if (potentialSpaces == 2)
874
875
876
877 potentialSpaces = 0;
878
879
880
881 /// Middle left 3x3 grid
882 for (int m = 3; m < 6; m++)
883     for (int n = 0; n < 3; n++)
884         if (gridPotential[m][n] == true)
885         {
886             potentialSpaces++;
887         }
888
889 if (potentialSpaces == 1)
890 {
891     for (int m = 3; m < 6; m++)
892         for (int n = 0; n < 3; n++)
893             if (gridPotential[m][n] == true)
894             {
895                 arr[m][n] = i;
896
897                 change = true;
898
899                 leftToSolve--;
900
901                 squareEliminator(gridPotential, m, n);
902
903                 elimination = true;
904
905                 display(arr);
906
907                 cout << leftToSolve << endl;
908
909             }
910
911 } // end if (potentialSpaces == 1)
912 else if (potentialSpaces == 2)
913 {
914     potentialSpaces = 0;
915
916     /// If the 2 potential spaces line up in a row within a
917     /// 3x3 grid, that row is eliminated because it is already
918     /// taken by the number, i, in that 3x3 grid
919     for (int m = 3; m < 6; m++)
920     {
921         for (int n = 0; n < 3; n++)
922         {
923             if (gridPotential[m][n] == true)
924             {

```



```

925         potentialSpaces++;
926
927         if (potentialSpaces == 2)
928         {
929             for (int j = 0; j < 9; j++)
930             {
931                 gridPotential[m][j] = false;
932
933                 elimination = true;
934             } // end for (int j = 0; j < 9; j++)
935         } // end if (potentialSpaces == 2)
936     } // end if (gridPotential[m][n] == true)
937 }
938
939 potentialSpaces = 0;
940 }
941
942
943
944 /// If the 2 potential spaces line up in a column within a
945 /// 3x3 grid, that column is eliminated because it is
946 /// already taken by the number, i, in that 3x3 grid
947 for (int n = 0; n < 3; n++)
948 {
949     for (int m = 3; m < 6; m++)
950     {
951         if (gridPotential[m][n] == true)
952         {
953             potentialSpaces++;
954
955             if (potentialSpaces == 2)
956             {
957                 for (int j = 0; j < 9; j++)
958                 {
959                     gridPotential[j][n] = false;
960
961                     elimination = true;
962                 } // end for (int j = 0; j < 9; j++)
963             } // end if (potentialSpaces == 2)
964         } // end if (gridPotential[m][n] == true)
965     }
966
967     potentialSpaces = 0;
968 }
969
970 } // end else if (potentialSpaces == 2)
971
972
973
974 potentialSpaces = 0;
975
976
977
978 /// Middle center 3x3 grid
979 for (int m = 3; m < 6; m++)
980     for (int n = 3; n < 6; n++)
981         if (gridPotential[m][n] == true)
982         {
983             potentialSpaces++;
984         }
985
986 if (potentialSpaces == 1)
987 {
988     for (int m = 3; m < 6; m++)
989         for (int n = 3; n < 6; n++)
990             if (gridPotential[m][n] == true)

```



```

1057
1058             elimination = true;
1059             } // end for (int j = 0; j < 9; j++)
1060             } // end if (potentialSpaces == 2)
1061             } // end if (gridPotential[m][n] == true)
1062         }
1063
1064         potentialSpaces = 0;
1065     }
1066
1067 } // end else if (potentialSpaces == 2)
1068
1069
1070
1071 potentialSpaces = 0;
1072
1073
1074
1075 /// Middle right 3x3 grid
1076 for (int m = 3; m < 6; m++)
1077     for (int n = 6; n < 9; n++)
1078         if (gridPotential[m][n] == true)
1079             {
1080                 potentialSpaces++;
1081             }
1082
1083 if (potentialSpaces == 1)
1084 {
1085     for (int m = 3; m < 6; m++)
1086         for (int n = 6; n < 9; n++)
1087             if (gridPotential[m][n] == true)
1088                 {
1089                     arr[m][n] = i;
1090
1091                     change = true;
1092
1093                     leftToSolve--;
1094
1095                     squareEliminator(gridPotential, m, n);
1096
1097                     elimination = true;
1098
1099                     display(arr);
1100
1101                     cout << leftToSolve << endl;
1102                 }
1103 }
1104
1105 } // end if (potentialSpaces == 1)
1106 else if (potentialSpaces == 2)
1107 {
1108     potentialSpaces = 0;
1109
1110     /// If the 2 potential spaces line up in a row within a
1111     /// 3x3 grid, that row is eliminated because it is already
1112     /// taken by the number, i, in that 3x3 grid
1113     for (int m = 3; m < 6; m++)
1114     {
1115         for (int n = 6; n < 9; n++)
1116         {
1117             if (gridPotential[m][n] == true)
1118             {
1119                 potentialSpaces++;
1120
1121                 if (potentialSpaces == 2)
1122                 {

```

```

1123         for (int j = 0; j < 9; j++)
1124         {
1125             gridPotential[m][j] = false;
1126
1127             elimination = true;
1128         } // end for (int j = 0; j < 9; j++)
1129     } // end if (potentialSpaces == 2)
1130 } // end if (gridPotential[m][n] == true)
1131 }
1132
1133     potentialSpaces = 0;
1134 }
1135
1136
1137
1138     /// If the 2 potential spaces line up in a column within a
1139     /// 3x3 grid, that column is eliminated because it is
1140     /// already taken by the number, i, in that 3x3 grid
1141     for (int n = 6; n < 9; n++)
1142     {
1143         for (int m = 3; m < 6; m++)
1144         {
1145             if (gridPotential[m][n] == true)
1146             {
1147                 potentialSpaces++;
1148
1149                 if (potentialSpaces == 2)
1150                 {
1151                     for (int j = 0; j < 9; j++)
1152                     {
1153                         gridPotential[j][n] = false;
1154
1155                         elimination = true;
1156                     } // end for (int j = 0; j < 9; j++)
1157                 } // end if (potentialSpaces == 2)
1158             } // end if (gridPotential[m][n] == true)
1159         }
1160
1161         potentialSpaces = 0;
1162     }
1163
1164 } // end else if (potentialSpaces == 2)
1165
1166
1167
1168     potentialSpaces = 0;
1169
1170
1171
1172     /// Bottom left 3x3 grid
1173     for (int m = 6; m < 9; m++)
1174     for (int n = 0; n < 3; n++)
1175         if (gridPotential[m][n] == true)
1176         {
1177             potentialSpaces++;
1178         }
1179
1180     if (potentialSpaces == 1)
1181     {
1182         for (int m = 6; m < 9; m++)
1183         for (int n = 0; n < 3; n++)
1184             if (gridPotential[m][n] == true)
1185             {
1186                 arr[m][n] = i;
1187
1188                 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)

```



```

1321         elimination = true;
1322     } // end for (int j = 0; j < 9; j++)
1323     } // end if (potentialSpaces == 2)
1324     } // end if (gridPotential[m][n] == true)
1325 }
1326
1327     potentialSpaces = 0;
1328 }
1329
1330
1331
1332     /// If the 2 potential spaces line up in a column within a
1333     /// 3x3 grid, that column is eliminated because it is
1334     /// already taken by the number, i, in that 3x3 grid
1335     for (int n = 3; n < 6; n++)
1336     {
1337         for (int m = 6; m < 9; m++)
1338         {
1339             if (gridPotential[m][n] == true)
1340             {
1341                 potentialSpaces++;
1342
1343                 if (potentialSpaces == 2)
1344                 {
1345                     for (int j = 0; j < 9; j++)
1346                     {
1347                         gridPotential[j][n] = false;
1348
1349                         elimination = true;
1350                     } // end for (int j = 0; j < 9; j++)
1351                 } // end if (potentialSpaces == 2)
1352             } // end if (gridPotential[m][n] == true)
1353         }
1354
1355         potentialSpaces = 0;
1356     }
1357
1358     } // end else if (potentialSpaces == 2)
1359
1360
1361     potentialSpaces = 0;
1362
1363
1364
1365     /// Bottom right 3x3 grid
1366     for (int m = 6; m < 9; m++)
1367     for (int n = 6; n < 9; n++)
1368     if (gridPotential[m][n] == true)
1369     {
1370         potentialSpaces++;
1371     }
1372
1373
1374     if (potentialSpaces == 1)
1375     {
1376         for (int m = 6; m < 9; m++)
1377         for (int n = 6; n < 9; n++)
1378         if (gridPotential[m][n] == true)
1379         {
1380             arr[m][n] = i;
1381
1382             change = true;
1383
1384             leftToSolve--;
1385
1386             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;

```



```

1453     }
1454
1455     } // end else if (potentialSpaces == 2)
1456
1457
1458
1459 } // end void actualAddAndElimPotentialElim(int arr[][9],
1460 //                                     bool gridPotential[][9],
1461 //                                     bool &elimination, bool &change,
1462 //                                     int &leftToSolve)
1463
1464
1465
1466 /*****
1467 **
1468
1469 Applies brute force to the array to solve it. Will solve any puzzle.
1470
1471 At each empty square, takes a randomly selected potential number of
1472 that square and tests to see if it was already used in the row,
1473 column or 3x3 grid. If it was not used, adds it to the square and goes
1474 to the next empty square. If it was used, picks another randomly
1475 selected potential number of that square and tests it. If all the
1476 potential numbers of an empty square are already taken, a new brute
1477 force attempt is started. Brute force is applied until the puzzle
1478 is solved.
1479
1480 **/
1481 /*****
1482
1483
1484
1485 bool bruteForce(int arr[][9])
1486 {
1487     int numberOfPotentials[9][9], potentialNumbers[9][9][9], randomPotential,
1488         arrCopy[9][9], numbersTriedCount;
1489     bool gridPotential[9][9], repeat, usedNumbers[9], taken, run;
1490
1491     void squareEliminator(bool [][][9], int, int);
1492     void display(int [][][9]);
1493
1494
1495     for (int m = 0; m < 9; m++)
1496         for (int n = 0; n < 9; n++)
1497         {
1498             numberOfPotentials[m][n] = 0;
1499         }
1500
1501
1502
1503
1504     /// Takes each number and checks for all the rows, columns and 3x3 grids
1505     /// that are already occupied by it on the board. It then adds that
1506     /// number to the next available slot of all other squares in a 3
1507     /// dimensional array and keeps track of how many potential numbers are on
1508     /// each square.
1509     for (int i = 1; i <= 9; i++)
1510     {
1511
1512         /// Initialize all the squares to false if they are occupied
1513         /// and true if they are empty
1514         for (int m = 0; m < 9; m++)
1515             for (int n = 0; n < 9; n++)
1516                 if (arr[m][n] > 0)
1517                 {
1518                     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             if (arrCopy[j][k] == 0)
1611             {
1612                 for (int m = 0; m < 9; m++)
1613                 {
1614                     usedNumbers[m] = false;
1615                 }
1616
1617                 repeat = false;
1618
1619                 numbersTriedCount = 0;
1620
1621                 do
1622                 {
1623
1624                     do
1625                     {
1626                         randomPotential =
1627                             potentialNumbers
1628                                 [j][k][rand() % numberOfPotentials[j][k]];
1629
1630                         if (usedNumbers[randomPotential - 1] == false)
1631                         {
1632                             repeat = false;
1633
1634                             run = true;
1635
1636                             usedNumbers[randomPotential - 1] = true;
1637
1638                             numbersTriedCount++;
1639
1640
1641                         } // end if (usedNumbers[randomPotential - 1] == false)
1642                     } else
1643                     {
1644                         repeat = true;
1645
1646                         /// Ends this brute force attempt
1647                         if (numbersTriedCount == numberOfPotentials[j][k])
1648                         {
1649                             j = 0;
1650

```

[illegible]

```

1717         {
1718             taken = true;
1719         }
1720     }
1721 }
1722 /// Top center 3x3 grid
1723 else if (j <= 2 && (k >= 3 && k <= 5))
1724 {
1725     for (int m = 0; m < 3; m++)
1726         for (int n = 3; n < 6; n++)
1727             if
1728                 (arrCopy[m][n] == randomPotential)
1729             {
1730                 taken = true;
1731             }
1732     }
1733 }
1734 /// Top right 3x3 grid
1735 else if (j <= 2 && (k >= 6 && k <= 8))
1736 {
1737     for (int m = 0; m < 3; m++)
1738         for (int n = 6; n < 9; n++)
1739             if
1740                 (arrCopy[m][n] == randomPotential)
1741             {
1742                 taken = true;
1743             }
1744     }
1745 }
1746 /// Middle left 3x3 grid
1747 else if ((j >= 3 && j <= 5) && k <= 2)
1748 {
1749     for (int m = 3; m < 6; m++)
1750         for (int n = 0; n < 3; n++)
1751             if
1752                 (arrCopy[m][n] == randomPotential)
1753             {
1754                 taken = true;
1755             }
1756     }
1757 }
1758 /// Middle center 3x3 grid
1759 else if ((j >= 3 && j <= 5) &&
1760         (k >= 3 && k <= 5))
1761 {
1762     for (int m = 3; m < 6; m++)
1763         for (int n = 3; n < 6; n++)
1764             if
1765                 (arrCopy[m][n] == randomPotential)
1766             {
1767                 taken = true;
1768             }
1769     }
1770 }
1771 /// Middle right 3x3 grid
1772 else if ((j >= 3 && j <= 5) &&
1773         (k >= 6 && k <= 8))
1774 {
1775     for (int m = 3; m < 6; m++)
1776         for (int n = 6; n < 9; n++)
1777             if
1778                 (arrCopy[m][n] == randomPotential)
1779             {
1780                 taken = true;
1781             }
1782     }

```

```

1783     }
1784     /// Bottom left 3x3 grid
1785     else if ((j >= 6 && j <= 8) && k <= 2)
1786     {
1787         for (int m = 6; m < 9; m++)
1788             for (int n = 0; n < 3; n++)
1789                 if
1790                     (arrCopy[m][n] == randomPotential)
1791                     {
1792                         taken = true;
1793                     }
1794     }
1795     /// Bottom center 3x3 grid
1796     else if ((j >= 6 && j <= 8) &&
1797             (k >= 3 && k <= 5))
1798     {
1799         for (int m = 6; m < 9; m++)
1800             for (int n = 3; n < 6; n++)
1801                 if
1802                     (arrCopy[m][n] == randomPotential)
1803                     {
1804                         taken = true;
1805                     }
1806     }
1807     /// Bottom right 3x3 grid
1808     else
1809     {
1810         for (int m = 6; m < 9; m++)
1811             for (int n = 6; n < 9; n++)
1812                 if
1813                     (arrCopy[m][n] == randomPotential)
1814                     {
1815                         taken = true;
1816                     }
1817     }
1818     }
1819     }
1820     }
1821     }
1822     }
1823     /// If the number is not taken in the row,
1824     /// column, or 3x3 grid, it is placed on
1825     /// the square
1826     if (!taken)
1827     {
1828         arrCopy[j][k] = randomPotential;
1829     }
1830     }
1831     }
1832     }
1833     } // end if (!taken)
1834     }
1835     } // end if (!taken)
1836     }
1837     } // end if (run)
1838     }
1839     }
1840     while (taken);
1841     }
1842     } // end if (arrCopy[j][k] == 0)
1843     }
1844     } // end for (int k = 0; k < 9; k++)
1845     } // end for (int j = 0; j < 9; j++)
1846     }
1847     display(arrCopy);
1848     }

```

```

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     if(!fillBoard(board, numToSolve))
1864     {
1865         cout << "what?" << endl;
1866     }
1867     else
1868     {
1869         startingFilled = 81 - numToSolve;
1870
1871         if(!solve(board, numToSolve))
1872         {
1873             cout << "Oh, no!" << endl;
1874
1875             if (bruteForce(board))
1876                 cout << "Brute force for the win!" << endl << endl;
1877         }
1878
1879         cout << "Number filled at start: " << startingFilled << endl;
1880     }
1881
1882     return 0;
1883 }
1884
1885

```