

```

1  /**
2
3      Sudoku Solver
4          Abhishek Srikanth
5          Class 12 - A
6          Global Indian Int'l School
7
8  **/
9
10 #include <iostream>
11 #include <fstream>
12 #include <conio.h>
13
14 using namespace std;
15
16 class box
17 {
18     public:
19     int val;
20     int nposib;
21     int posib[9];
22     box()    // sets val to ZERO, posib from 1-9
23     {
24         val = 0;
25         nposib = 9;
26         for(int i = 0; i < 9; ++i)
27         {
28             posib[i] = i + 1;
29         }
30     }
31 };
32 box sudoku[9][9];
33 box save[9][9];
34
35
36 void set_posib()
37 {
38     // "Will now set possibilities for each value";
39
40     for(int i = 0; i < 9; ++i)
41     {
42         for(int j = 0; j < 9; ++j)
43         {
44             if(sudoku[i][j].val!=0)    // IF SUDOKU[i][j] HAS A REAL VALUE
45             {
46
47                 // Loop removes SUDOKU[i][j]'s value as a posib from corresponding row and column
48                 for(int m = 0; m < 9; ++m)
49                 {
50
51                     if(sudoku[i][m].posib[ sudoku[i][j].val - 1 ] != 0)
52                     {
53                         sudoku[i][m].posib[ sudoku[i][j].val - 1 ] = 0;
54                         sudoku[i][m].nposib--;    // change no. of possibilities
55                     }
56                     if(sudoku[m][j].posib[ sudoku[i][j].val - 1 ] != 0)
57                     {
58                         sudoku[m][j].posib[ sudoku[i][j].val - 1 ] = 0;
59                         sudoku[m][j].nposib--;
60                     }
61                 }
62
63                 /* Below body to find center point of corresponding quadrant */
64                 int Ci=-1,Cj=-1;
65                 if(i+1 == 1 || i+1 == 4 || i+1 == 7)
66                     Ci = i+1;

```

```

67     else if(i==1 || i==4 || i==7)
68         Ci = i;
69     else if(i-1 == 1 || i-1 == 4 || i-1 == 7)
70         Ci = i-1;
71
72     if(j+1 == 1 || j+1 == 4 || j+1 == 7)
73         Cj = j+1;
74     else if(j==1 || j==4 || j==7)
75         Cj = j;
76     else if(j-1 == 1 || j-1 == 4 || j-1 == 7)
77         Cj = j-1;
78     /* Center point of quadrant is denoted by 'Ci' and 'Cj' */
79
80     for(int m = Ci-1; m < Ci+2; ++m)
81     {
82         for(int n = Cj-1; n < Cj+2; ++n)
83         {
84             if(sudoku[m][n].posib[ sudoku[i][j].val - 1 ] != 0)
85             {
86                 sudoku[m][n].posib[ sudoku[i][j].val - 1 ] = 0;
87                 sudoku[m][n].nposib--;
88             }
89         }
90     }
91     /* Above nested loop accesses all values present in quadrant */
92
93     sudoku[i][j].nposib = 0;
94     for(int r = 0; r < 9; ++r)
95     {
96         sudoku[i][j].posib[r] = 0;
97     }
98 }
99
100 }
101 }
102 }
103
104 void singletons()
105 {
106     start_cuz_values_have_changed:
107     for(int i = 0; i < 9; ++i)
108     {
109         for(int j = 0; j < 9; ++j)
110         {
111             if(sudoku[i][j].nposib == 1) // if only 1 possibility is present
112             {
113                 for(int k = 0; k < 9; ++k) // scan through possibilities
114                 {
115                     if(sudoku[i][j].posib[k]!=0) // If 'k'th possibility is NONZERO
116                     {
117                         sudoku[i][j].val = sudoku[i][j].posib[k]; // set value to that possibility
118                         sudoku[i][j].nposib=0; // set number of possibilities to ZERO
119                         for(int r = 0; r < 9; ++r)
120                             sudoku[i][j].posib[r] = 0;
121
122                         break; // exit scanning possibilities
123                     }
124                 }
125
126                 // To eliminate that value from corresponding ROW, COL, QUADRANT :
127
128                 // Loop removes SUDOKU[i][j]'s value as a posib from corresponding row and column
129                 for(int m = 0; m < 9; ++m)
130                 {
131                     if(sudoku[i][m].posib[ sudoku[i][j].val - 1 ] != 0)
132                     {

```

```

133         sudoku[i][m].posib[ sudoku[i][j].val - 1 ] = 0;
134         sudoku[i][m].nposib--;
135     }
136     if(sudoku[m][j].posib[ sudoku[i][j].val - 1 ] != 0)
137     {
138         sudoku[m][j].posib[ sudoku[i][j].val - 1 ] = 0;
139         sudoku[m][j].nposib--;
140     }
141 }
142
143 /* Below body to find center point of corresponding quadrant */
144 int Ci=-1,Cj=-1;
145 if(i+1 == 1 || i+1 == 4 || i+1 == 7)
146     Ci = i+1;
147 else if(i==1 || i==4 || i==7)
148     Ci = i;
149 else if(i-1 == 1 || i-1 == 4 || i-1 == 7)
150     Ci = i-1;
151
152 if(j+1 == 1 || j+1 == 4 || j+1 == 7)
153     Cj = j+1;
154 else if(j==1 || j==4 || j==7)
155     Cj = j;
156 else if(j-1 == 1 || j-1 == 4 || j-1 == 7)
157     Cj = j-1;
158 /* Center point of quardrant is denoted by 'Ci' and 'Cj' */
159
160 for(int m = Ci-1; m < Ci+2; ++m)
161 {
162     for(int n = Cj-1; n < Cj+2; ++n)
163     {
164         if(sudoku[m][n].posib[ sudoku[i][j].val -1 ] != 0)
165         {
166             sudoku[m][n].posib[ sudoku[i][j].val -1 ] = 0;
167             sudoku[m][n].nposib--;
168         }
169     }
170 }
171 /* Above nested loop accesses all values present in quadrant */
172
173 goto start_cuz_values_have_changed; // goes only if a value has been set
174 }
175 }
176 }
177 }
178
179 void backup()
180 {
181 /**
182     This set of code simply backs up
183     the current sudoku so that guessing
184     can be done **/
185
186     // this results in save[][] being the same as sudoku
187     for(int i = 0; i < 9; ++i)
188     {
189         for(int j = 0; j < 9; ++j)
190         {
191             save[i][j].val = sudoku[i][j].val;
192             save[i][j].nposib = sudoku[i][j].nposib;
193             for(int m = 0; m < 9; ++m)
194                 save[i][j].posib[m] = sudoku[i][j].posib[m];
195         }
196     }
197 }
198

```

```

199 void setsudoku()
200 {
201     /**
202     This set of code simply resets
203     the modulated sudoku so that guessing
204     can be done **/
205
206     // this results in sudoku[i][j] being the same as save[i][j]
207     for(int i = 0; i < 9; ++i)
208     {
209         for(int j = 0; j < 9; ++j)
210         {
211             sudoku[i][j].val = save[i][j].val;
212             sudoku[i][j].nposib = save[i][j].nposib;
213             for(int m = 0; m < 9; ++m)
214                 sudoku[i][j].posib[m] = save[i][j].posib[m];
215         }
216     }
217 }
218
219 void rfrequency()
220 {
221     set_posib();
222     singletons(); // directly calls these functions, hence eliminating the need to call them in main()
223
224     rowcheck:
225     int counter = 0; // to check whether row has frequency change or not
226     // row-wise
227     for(int i = 0; i < 9; ++i) // traversers from row 1-9
228     {
229         int freq[9] = {0,0,0,0,0,0,0,0,0};
230         for(int j = 0; j < 9; ++j)
231         {
232             if(sudoku[i][j].nposib!=0) // if the values is not set already
233                 for(int k = 0; k < 9; ++k)
234                 {
235
236                     if(sudoku[i][j].posib[k]!=0) // and if the possibility is non-zero
237                     {
238                         freq[k]++;
239                     }
240                 }
241         }
242         int n = 0;
243         for(int k = 0; k < 9; ++k)
244         {
245             if(freq[k]==1)
246             {
247                 n = k+1; // n holds value of number with 1 frequency
248                 break;
249             }
250         }
251         // If number with 1 frequency exists
252         if(n!=0)
253         {
254             ++counter;
255             for(int j = 0; j < 9; ++j) // for every element in that row
256             {
257                 if(sudoku[i][j].nposib!=0) // if value is already not present
258                     if(sudoku[i][j].posib[n-1]!=0) // cuz that is value with frequency 1
259                     {
260                         // set val
261                         sudoku[i][j].val = n;
262                         sudoku[i][j].nposib = 0;
263                         for(int m = 0; m < 9; ++m)
264                         {

```

```

265         sudoku[i][j].posib[m] = 0;
266     }
267     break;
268 }
269 }
270 }
271 }
272 if(counter!=0)
273 {
274     set_posib(); // if change has been made, call set_posib
275     singletons(); // call singleton function, set singletons again cuz some new ones may be formed!
276     goto rowcheck; // restart row wise check
277 }
278
279 /**  ONCE ALL ROWS HAVE BEEN SET , START WORKING ON COLUMNS  */
280
281 int counter2 = 0;
282 // col-wise
283 for(int i = 0; i < 9; ++i) // traversers from col 1-9
284 {
285     int freq2[9] = {0,0,0,0,0,0,0,0,0};
286     for(int j = 0; j < 9; ++j)
287     {
288         for(int k = 0; k < 9; ++k)
289         {
290             if(sudoku[j][i].nposib!=0) // if the values is not set already
291                 if(sudoku[j][i].posib[k]!=0) // and if the possibility is non-zero
292                 {
293                     freq2[k]++;
294                 }
295             }
296         }
297         int n2 = 0;
298         for(int k = 0; k < 9; ++k)
299         {
300             if(freq2[k]==1)
301             {
302                 n2 = k+1; // n2 holds value of number with ! frequency
303                 break;
304             }
305         }
306         // If number with 1 frequency exists
307         if(n2!=0)
308         {
309             ++counter2;
310             for(int j = 0; j < 9; ++j)
311             {
312                 if(sudoku[j][i].nposib!=0) // if value is already not present
313                     if(sudoku[j][i].posib[n2-1]!=0) // cuz that is value with frequency 1
314                     {
315                         // set val
316                         sudoku[j][i].val = n2;
317                         sudoku[j][i].nposib = 0;
318                         for(int m = 0; m < 9; ++m)
319                         {
320                             sudoku[j][i].posib[m] = 0;
321                         }
322                         break;
323                     }
324             }
325         }
326     }
327 if(counter2!=0)
328 {
329     set_posib(); // if change has been made, call set_posib
330     singletons(); // call singleton function, set singletons again cuz some new ones may be formed!

```

```

331     goto rowcheck; // restart row wise check
332 }
333
334 /**  ONCE ALL COLS HAVE BEEN SET , START WORKING ON QUADRANTS  **/
335
336 // quadrant - wise
337 int counter3 = 0;
338
339 // note that the loop only gives i = j= {1,4,7} which are quadrant centers
340 for(int i = 1; i < 8; i+=3)
341 {
342     for(int j = 1; j < 8; j+=3)
343     {
344         // for every box henceforth
345         int freq3[9] = {0,0,0,0,0,0,0,0,0};
346         for(int Ci = i-1; Ci<=i+1; ++Ci)
347         {
348             for(int Cj = j-1; Cj<=j+1; ++Cj)
349             {
350                 if(sudoku[Ci][Cj].nposib!=0) // if the value has not been determined
351                 {
352                     for(int k = 0; k < 9; ++k)
353                     {
354                         if(sudoku[Ci][Cj].posib[k]!=0) // if 'k'th posib exists,
355                             freq3[k]++;
356                     }
357                 }
358             }
359         }
360         int n3 = 0;
361         for(int k = 0; k < 9; ++k)
362         {
363             if(freq3[k]==1)
364             {
365                 n3=k+1;
366                 break;
367             }
368         }
369         if(n3!=0) // if a frequency 1 value exists
370         {
371             ++counter3;
372             for(int Ci = i-1; Ci<=i+1; ++Ci)
373             {
374                 for(int Cj = j-1; Cj<=j+1; ++Cj)
375                 {
376                     // every element in the quadrant
377                     for(int k = 0; k < 9; ++k)
378                     {
379                         if(sudoku[Ci][Cj].posib[n3-1] != 0) // if required box is located
380                         {
381                             sudoku[Ci][Cj].val = n3;
382                             sudoku[Ci][Cj].nposib = 0;
383
384                             for(int r = 0; r < 9; ++r)
385                                 sudoku[Ci][Cj].posib[r] = 0;
386                             break;
387                         }
388                     }
389                 }
390             }
391         }
392     }
393 }
394 }
395 }
396 if(counter3!=0)

```

```

397     {
398         set_posib();    // if change has been made, call set_posib
399         singletons();   // call singleton function, set singletons again cuz some new ones may be formed!
400         goto rowcheck;  // restart row wise check
401     }
402 }
403
404
405 void guess()
406 {
407     cout << "initiating brute force algorithm \n";
408     starting:
409     int row=-1,col=-1,val=0;
410     for(int i =0; i < 9; ++i)
411     {
412         for(int j = 0; j < 9; ++j)
413         {
414             // goes through every element
415             if(sudoku[i][j].val==0)
416             {
417                 row=i;
418                 col=j;
419                 for(int k = 0; k < 9; ++k)
420                 {
421                     if(sudoku[i][j].posib[k]!=0)
422                     {
423                         val = sudoku[i][j].posib[k];
424                         sudoku[i][j].val = val;
425                         sudoku[i][j].nposib=0;
426                         goto loop_stop;
427                     }
428                 }
429             }
430         }
431     }
432     loop_stop:
433
434     for(int k = 0; k<9; ++k)
435         sudoku[row][col].posib[k]=0;
436
437     rfrequency();
438
439     // sudoku with a guess has been solved
440     // loop then runs to see if it worked
441
442     for(int i = 0; i < 9; ++i)
443     {
444         for(int j = 0; j < 9; ++j)
445         {
446             // for every element in the sudoku
447
448             if(sudoku[i][j].val == 0)    // if not solved
449             {
450                 // if no solution is possible
451                 // then make changes to save[][]
452                 // resetsudoku according to change
453                 if(sudoku[i][j].nposib==0)
454                 {
455                     save[row][col].posib[val-1]=0;
456                     save[row][col].nposib-=1;
457                     save[row][col].val=0;          // just incase
458                     setsudoku();
459                 }
460                 cout << '.';
461                 goto starting;
462             }

```

```

463     }
464 }
465 cout << "\nsuccessful brute force execution!\n";
466
467 }
468
469
470 // the final message!
471 void view()
472 {
473     cout << endl << endl;
474     cout << " #####  ##      ## #####  #####  ##      ##  ##      ## " << endl;
475     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
476     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
477     cout << " #####  ##      ##      ##      ##      ##      ##      ## " << endl;
478     cout << "      ##      ##      ##      ##      ##      ##      ##      ## " << endl;
479     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
480     cout << " #####  #####  #####  #####  ##      ##      ##      ## " << endl;
481     cout << endl;
482     cout << " #####  #####  ##      ##      ##      #####  #####  " << endl;
483     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
484     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
485     cout << " #####  ##      ##      ##      ##      ##      #####  ##### " << endl;
486     cout << "      ##      ##      ##      ##      ##      ##      ##      ## " << endl;
487     cout << "##      ##      ##      ##      ##      ##      ##      ## " << endl;
488     cout << " #####  #####  #####  ###      #####  ##      ##      ## " << endl;
489     cout << endl
490     << endl
491     << endl
492     << "          BBBB      \n"
493     << "          B   B      \n"
494     << "          BBBB y  y \n"
495     << "          B   B y  y \n"
496     << "          BBBB   yy \n"
497     << "              y \n"
498     << "              yyy \n"
499     << endl
500     << endl
501     << endl
502     << "      #                                " << endl
503     << "      # # ##### #      # # ##### #      # ##### #      # " << endl
504     << "      # # #      # #      # #      #      # #      #      # " << endl
505     << "      #      # ##### ##### # ##### ##### ##### ##### " << endl
506     << "      ##### #      # #      # #      # #      # #      # " << endl
507     << "      #      # #      # #      # #      # #      #      # " << endl
508     << "      #      # ##### #      # # ##### #      # ##### #      # " << endl
509     << endl
510     << endl
511     << endl;
512 }
513
514
515 int main()
516 {
517     cout << endl;
518     cout << "Welcome to the sudoku solver! \n";
519     cout << endl;
520     cout << "This program is specifically tailored to solve any valid sudoku you enter.\n";
521     b:
522     cout << endl;
523     cout << "Please enter a valid sudoku for expected results : \n\n";
524     char ch;
525     /***** INPUT *****/
526
527     cout << "*****" << endl;
528     for(int i = 0; i < 9; ++i)

```



```

529 {
530     for(int j = 0; j < 9; ++j)
531     {
532         a:
533         ch = getch();
534         if(ch > '0' && ch <= '9')
535         {
536             sudoku[i][j].val = (int)ch - 48;
537             cout << sudoku[i][j].val;
538         }
539         else if(ch=='\n' || ch=='\r')
540         {
541             sudoku[i][j].val = 0;
542             cout << "- ";
543         }
544         else
545             goto a;
546         if((j+1)%3==0)
547             cout << " * ";
548         cout << " ";
549     }
550     cout << endl;
551     if((i+1)%3==0)
552         cout << "*****" << endl;
553 }
554
555 /***** DISPLAY *****/
556
557 cout << "\nThank you for the input..." << endl;
558 cout << "Please check if this is the correct sudoku : \n\n";
559
560 cout << "*****" << endl;
561 for(int i = 0; i < 9; ++i)
562 {
563     for(int j = 0; j < 9; ++j)
564     {
565         if(sudoku[i][j].val!=0)
566             cout << sudoku[i][j].val << " ";
567         else cout << "- ";
568
569         if((j+1)%3==0)
570             cout << " * ";
571     }
572     cout << endl;
573     if((i+1)%3==0)
574         cout << "*****" << endl;
575 }
576 cout << endl << "Is the correct sudoku (y/n) : " ;
577 cin >> ch;
578 if(ch=='N' || ch == 'n')
579     goto b;
580 else
581     cout << "the program shall now start solving the sudoku \n\n";
582
583 /***** SOLUTION *****/
584
585 rfrequency();
586 backup();
587
588 cout << endl;
589
590 /***** Call for guessing *****/
591
592 for(int i = 0; i < 9; ++i)
593 {
594     for(int j = 0; j < 9; ++j)

```

```

595     {
596         if(save[i][j].val==0)
597         {
598             guess();
599             backup();
600             goto loop_term;
601         }
602     }
603 }
604 loop_term:
605
606 /*****/
607
608     cout << "\n\nAnd the complete solved sudoku is : \n\n";
609
610     // display after brute force solution
611     cout << " ****" << endl;
612     for(int i = 0; i < 9; ++i)
613     {
614         cout << " * ";
615         for(int j = 0; j < 9; ++j)
616         {
617             if(sudoku[i][j].val!=0)
618                 cout << sudoku[i][j].val << " ";
619             else cout << "- ";
620             if((j+1)%3==0)
621                 cout << " * ";
622         }
623         cout << endl;
624         if((i+1)%3==0)
625             cout << " ****" << endl;
626     }
627
628 /*****/
629
630     cout << "\n\nDo you wish to save this sudoku solution(y/n) : ";
631     cin >> ch;
632     if(ch=='y' || ch == 'Y')
633     {
634         ofstream solution("solutions.txt", ios_base::app | ios::out);
635         cout << "What do you want this solution to be named as : ";
636         char puzzle_name[10];
637         cin >> puzzle_name;
638         solution << endl << puzzle_name << endl;
639         solution << "*****\n";
640         for(int i = 0; i < 9; ++i)
641         {
642             for(int j = 0; j < 9; ++j)
643             {
644                 solution << sudoku[i][j].val << " ";
645                 if((j+1)%3==0)
646                     solution << " * ";
647             }
648             solution << endl;
649             if((i+1)%3==0)
650                 solution << "*****\n";
651         }
652
653         solution.close();
654         cout << endl
655             << "Solution successfully appended to \"solutions.txt\"."
656             << endl << endl;
657         getch();
658     }
659
660     cout << "\n\n\n\nThank you for using this programme and i hope it impressed you!\n\n\n" << endl <<

```

```
endl;  
661     view();  
662  
663     return 0;  
664 }
```