```cpp
/* SUDOKU SOLVER:
~Nitin Rohit

This is a code which I have written from scratch without any internet aid which can be used to solve
sudoku of any valid dimension, which we can change using the variables which I will explain. In this
code I have used basic user defined functions. I'll will explain how the code works at each required
part so that you don't have a hard time understading a simple code. */

#include<iostream>
using namespace std;

// r can be any integer and s should be the square of it, where s is the dimension of sudoku.
const int r=3,s=9;

// This function gets the value of sudoku elements from user using a 2-d loop.
void getSudoku(int (*grid)[s])
    {
        cout<<"Enter the values of sudoku:\n";
        for(int i=0;i<s;i++)
        for(int j=0;j<s;j++)
        cin>>grid[i][j];
    }

/* Value-array: I have defined a 3-d array which stores 1s and 0s for each element of the sudoku grid.
It stores 1 if the a number have a possibility to occur at that box if not possible then 0 is assigned. */

/* This function fills the value-array for the existing elements with 1 for the number present and 0s
for rest of the numbers. */
void fill(int (*value)[s][s], int (*grid)[s],int i, int j)
    {
        for(int k=0;k<s;k++)
        value[i][j][k]=0;
        value[i][j][grid[i][j]-1]=1;
    }

// Intialising the value-array with ones as all numbers can be a potential candidate for an empty sudoku.
void intialisation1(int (*value)[s][s])
    {
        for(int i=0;i<s;i++)
        for(int j=0;j<s;j++)
        for(int k=0;k<s;k++)
        value[i][j][k]=1;
    }

// Intialising the sum array with zeros as sum is intially zero for all elements.
void intialisation2(int (*sum)[s])
    {
        for(int i=0;i<s;i++)
        for(int j=0;j<s;j++)
            sum[i][j]=0;
    }
```

```
51
52 // This fucntion returns the value of numbers of boxs left unsolved so that loop can
   run until then.
53 int endfun(int (*grid)[s])
54     {
55         int end=0;
56         for(int i=0;i<s;i++)
57         for(int j=0;j<s;j++)
58            if(!grid[i][j])
59               end=end+1;
60
61          return end;
62     }
63
64 /* This function assigns the value of value-array using the sudoku logic,i.e, all
   elements in row,
65 column and box should be unique simulataneously */
66 void getValue(int (*value)[s][s],int (*grid)[s])
67     {
68         for(int i=0;i<s;i++)
69         for(int j=0;j<s;j++)
70         if(grid[i][j])
71             fill(value,grid,i,j);
72         else
73         {
74             for(int m=0;m<s;m++)
75                 if(grid[m][j])
76                     value[i][j][grid[m][j]-1]=0;
77             for(int n=0;n<s;n++)
78                 if(grid[i][n])
79                     value[i][j][grid[i][n]-1]=0;
80             for(int x=(i/r)*r;x<(i/r)*r+r;x++)
81             for(int y=(j/r)*r;y<(j/r)*r+r;y++)
82                  if(grid[x][y])
83                      value[i][j][grid[x][y]-1]=0;
84         }
85
86     }
87
88 /* As the name suggests it gives the sum of all elements along the 3rd dimension of
   the value-array,
89 we will use this  array values to find weather a element can occur there with
   assurity. The value-array
90 gives the numbers which can occur or not, so a number can occur there if no other
   number can't occur
91 there apart from itself. Which means that if the sum is one then it has a unique
   solution.*/
92 void getSum(int (*value)[s][s],int (*sum)[s])
93     {
94         intialisation2(sum);
95         for(int i=0;i<s;i++)
96         for(int j=0;j<s;j++)
97         for(int k=0;k<s;k++)
98             sum[i][j]=value[i][j][k]+sum[i][j];
99
100    }
101
102 // This function gives the sum of particular row of all the elements value-array in
   3rd dimension
103 void getHsum(int *hsum, int (*value)[s][s],int i)
```

```
104     {
105         for(int k=0;k<s;k++)
106             hsum[k]=0;
107 
108         for(int k=0;k<s;k++)
109         for(int l=0;l<s;l++)
110             hsum[k]=hsum[k]+value[i][l][k];
111     }
112 
113 // This function gives the sum of particular column of all the elements value-array
    in 3rd dimension
114 void getVsum(int *vsum, int (*value)[s][s],int j)
115     {
116         for(int k=0;k<s;k++)
117             vsum[k]=0;
118 
119         for(int k=0;k<s;k++)
120         for(int l=0;l<s;l++)
121             vsum[k]=vsum[k]+value[l][j][k];
122     }
123 
124 // This function gives the sum of particular box of all the elements value-array in
    3rd dimension
125 void getBsum(int *bsum, int (*value)[s][s],int i,int j)
126     {
127         for(int k=0;k<s;k++)
128             bsum[k]=0;
129 
130         for(int k=0;k<s;k++)
131         for(int l=0;l<r;l++)
132         for(int m=0;m<r;m++)
133             bsum[k]=bsum[k]+value[i+l][j+m][k];
134     }
135 
136 /*  So this fucntion checks if the square has a unique solution according to the
    usual sudoku rule,i.e,
137 if only a single element is being common(row, column and box) to that square then
    that is the solution.
138 It first uses the sum array to check if there exists a unique solution(sum value of
    that box will be one
139 as only it is unique solution), if yes then finds that number and updates the sudoku
    array along with
140 the value-array.  */
141 void intersection(int (*value)[s][s], int (*grid)[s],int (*sum)[s])
142     {
143         getSum(value,sum);
144 
145         for(int i=0;i<s;i++)
146         for(int j=0;j<s;j++)
147            if(sum[i][j]==1)
148               for(int k=0;k<s;k++)
149                   if(value[i][j][k])
150                      {
151                          grid[i][j]=k+1;
152                          getValue(value,grid);
153                          break;
154                      }
155     }
156 
```

```
157 /* As the name suggests the function uses the concept of complementary. This
    function checks if a number
158 can't occur in other squares except for one square, if so the number has to come in
    that square. This
159 has to be checked along row, column and box. So we use the sum array along the
    required direction to check
160 if that unique solution exists by equating it to one as done in previous funciont.
    */
161   void complement(int (*value)[s][s], int (*grid)[s],int *hsum,int *vsum,int *bsum)
162       {
163           for(int i=0;i<s;i++)
164               {
165                   getHsum(hsum,value,i);
166                   for(int x=0;x<s;x++)
167                       if(hsum[x]==1)
168                          for(int y=0;y<s;y++)
169                              if(value[i][y][x]==1)
170                                 {
171                                     grid[i][y]=x+1;
172                                     getValue(value,grid);
173                                 }
174               }
175
176           for(int j=0;j<s;j++)
177               {
178                   getVsum(vsum,value,j);
179                   for(int x=0;x<s;x++)
180                       if(vsum[x]==1)
181                          for(int y=0;y<s;y++)
182                              if(value[y][j][x]==1)
183                                 {
184                                     grid[y][j]=x+1;
185                                     getValue(value,grid);
186                                 }
187               }
188
189           for(int i=0;i<s;i=i+r)
190           for(int j=0;j<s;j=j+r)
191               {
192                   getBsum(bsum,value,i,j);
193                   for(int x=0;x<s;x++)
194                       if(bsum[x]==1)
195                          for(int y=0;y<r;y++)
196                          for(int z=0;z<r;z++)
197                              if(value[y+i][z+j][x]==1)
198                                 {
199                                     grid[y+i][z+j]=x+1;
200                                     getValue(value,grid);
201                                 }
202               }
203       }
204
205   // This function can display a 2-d array of dimension same as of sudoku
206   void display(int (*grid)[s])
207      {
208          for(int i=0;i<s;i++)
209          {
210             for(int j=0;j<s;j++)
211             cout<<grid[i][j]<<" ";
212             cout<<"\n";
```

```
213         }
214     }
215 
216 
217 // Now we just have to call the functions in main function accordingly and run the
    code
218   int main()
219     {
220         int value[s][s][s],grid[s][s],sum[s][s],hsum[s],vsum[s],bsum[s],i,j,k,end;
221         getSudoku(grid);
222         cout<<"\n";
223         intialisation1(value);
224         end=endfun(grid);
225         getValue(value,grid);
226 
227         while(end)
228         {
229             intersection(value,grid,sum);
230             complement(value,grid,hsum,vsum,bsum);
231             end=endfun(grid);
232         }
233 
234         display(grid);
235         return 0;
236     }
237 
238 // THANK YO
```