S	U	D	O	K	U			
		S	O	L	V	E	R	
				B	Y			
	A	B	H	I	S	H	E	K
	S	R	I	K	A	N	T	H

The Sudoku Solver is a programme specifically tailored to solve any valid Sudoku.
It is capable of solving any Sudoku you find in your day to day life.

Abhishek Srikanth
Global Indian International School, Queenstown
Class: 12 - A

Contents

- **About the programming tool used**
  - Language
  - Compiler software
  - Compiler
- **About the Software**
- **Factors used**
- **Sudoku solver – user guide**
- **System requirements**
- **Design requirements**
- **Algorithms**
  - Box()
  - setposib()
  - singletons()
  - 1-4-7 co-ordinate system
  - backup()
  - rfrequency()
  - guess()
  - view()
  - Input of Sudoku in main() function
- **Flowchart**
- **Source code**
- **Outputs**
- **Acknowledgement**
- **Bibliography**

About the programming tool used

Language:

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features. Developed by Bjarne Stroustrup starting in 1979 at Bell Labs, it adds object oriented features, such as classes, and other enhancements to the C programming language. Originally named C with Classes, the language was renamed C++ in 1983, as a pun involving the increment operator.

C++ is one of the most popular programming languages and is implemented on a wide variety of hardware and operating system platforms. As an efficient compiler to native code, its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games. Several groups provide both free and proprietary C++ compiler software, including the GNU Project, Microsoft, Intel and Embarcadero Technologies.

Compiler software:

Code::Blocks is a free and open source, cross-platform IDE which supports multiple compilers including GCC and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C and C++. It can also be used for creating ARM, AVR, D, *DirectX*, FLTK, Fortran, *GLFW*, *GLUT*, GTK+, Irrlicht, Lightfeather, MATLAB, *OGRE*, *OpenGL*, Qt, SDL, SFML, *STL*, SmartWin and wx programs and applications, although in some cases installing third-party SDKs or frameworks is necessary.

Compiler:

The **GNU Compiler** Collection includes front ends for C, C++, C#, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

About the Software

The Sudoku Solver is a programme specifically designed to solve ANY valid Sudoku puzzle.

The Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids contain all of the digits from 1 to 9. An additional constraint on the contents of individual regions is that the same single integer may not appear twice in the same 9×9 playing board row or column or in any of the nine 3×3 sub-regions of the 9×9 playing board.

The puzzle was popularized in 1986 by the Japanese puzzle company Nikoli, under the name Sudoku, meaning *single number*. It became an international hit in 2005.

There a total of 6,670,903,752,021,072,936,960 possible permutations. This number is equal to 9! × 722 × 27 × 27,704,267,971, the last factor of which is prime. The result was derived through logic and “**brute force computation**."

Factors used

- ✓ Use of arrays
- ✓ Use of Classes
- ✓ Presence of Validation checks
- ✓ Use of nested loops
  - ➢ greatest depth of for loops = 6
  - ➢ lines 340 to 395 in rfrequency();
- ✓ Use of if - else if ladder
- ✓ File streaming
- ✓ Use of following header files :
  - ➢ iostream
  - ➢ fstream
  - ➢ conio.h
- ✓ use of complicated algorithms

Sudoku solver - user guide

During input:
  -> Only numbers {1-9} and 'return - enter' key is accepted
  -> Any other character will not be recognised
  -> If you make a mistake, after input you may choose to re-enter
      -> during YES or NO choice, only {y,Y,n,N} are accepted
      -> all others will be considered as YES
  -> Entering an invalid Sudoku will make programme run to infinity

After input wait for programme to solve

Once solution has been displayed, you may choose to save the answer

A YES or NO {y,Y,n,N} choice is presented
    -> incorrect entry is treated as a NO
    -> if YES, solution saved in "solutions.txt"
          -> user may press any key to proceed forward

System requirements

- ❖ RAM: Minimum 450 KB, Suggested 520 KB or more.
- ❖ Hard Disk: Minimum 1MB.
- ❖ It works on Windows only.
- ❖ Need Keyboard for input.

Design requirements

1. Should solve any valid Sudoku
2. Should give accurate and correct solutions
3. Should solve the Sudoku quickly
4. Should be easy to repair and update
5. Should use little memory
6. Should be user friendly

Box() : constructor

```
# Sets the following #
val = 0

nposib = 0

posib[1-9]=[1-9]
```

setposib() : function to set possibilities

```
for( i : 1 to 9)

    for( j : 1 to 9)

        If sudoku[i][j]'s value is not a ZERO

            .Run Loop to remove SUDOKU[i][j]'s value as a posib from corresponding row and column

            .look for corresponding 3x3 box by 1-4-7 co-ordination system*

            .remove sudoku[i][j]'s value as a posib from corresponding row and column
```

singletons() : function to find singletons

```
for( i : 1 to 9)

    for( j : 1 to 9)

        if sudoku[i][j] has nposib==1

            .find that posib and set val as that posib

            # as a value has been changed, possibilities in each cell changes #
```

.Run Loop to remove SUDOKU[i][j]'s value as a posib from corresponding row and column

.look for corresponding 3x3 box by 1-4-7 co-ordination system*

.remove sudoku[i][j]'s value as a posib from corresponding row and column

.goto start of the function because a value had been changed => another singleton might now be present

1-4-7 co-ordination system:

Look at value of:

i,i+1,i-1 and find whether any are 1,4,7

j,j+1,j-1 and find whether any are 1,4,7

we get pair like (1,1)(1,4)(7,4)...

this represents 3x3 matrix required

backup() : function that backup's sudoku[][]

```
for( i : 1 to 9)
    for( j : 1 to 9)
        .save[][].val = sudoku[][].val
        .save[][].nposib = sudoku[][].nposib
        .save[][].posib[1 to 9] = sudoku[][].posib[1 to 9]
```

setsudoku() : function that resets sudoku[][]

```
for( i : 1 to 9)
        for( j : 1 to 9)
                .sudoku[][].val = save[][].val
                .sudoku[][].nposib = save[][].nposib
                .sudoku[][].posib[1 to 9] = save[][].posib[1 to 9]
```

rfrequency() : function that solves based on frequency distribution

```
.call setposib()
.call singletons()


.counter = 0                # flag to see if changes were made or not
for( i : 1 to 9)
        .freq[9]={0,0,0...0}
        for( j : 1 to 9)
                .if no value present in cell then add posib's values to freq[]
        .n = 0
        for( k : 1 to 9)     # to scan through freq
                if freq[k]==1
                        .n = k+1
                        .break
        if(n!=0)
                .counter = 1
                for( j : 1 to 9)
                        if(sudoku[i][j] has possiblities)
                                if(sudoku[i][j].posib[n-1]!=0)
```

```
                    .set value to cell as that possibility
if(counter!=0)
    .call setposib()
    .call singletons()
    .goto top of function


#                                                          #
#  Similar code for column wise and 3x3 matrix wise solution   #
# in case of 3x3 matrix, we run on following 3x3 matrices:      #
#          (1,1)->(1,4)->(1,7)->(4,1)  ...->...  (7,4)->(7,7)          #
#                                                          #
```

guess() : function that guesses remaining solution

```
for( i : 1 to 9)
    for( j : 1 to 9)
        .first unsolved cell has its values set as the first possibility for matrix sudoku[][]
        .break;
.call rfrequency()
for( i : 1 to 9)
    for( j : 1 to 9)
        if(sudoku is not solved yet)
            if(solution is further impossible to get)
```

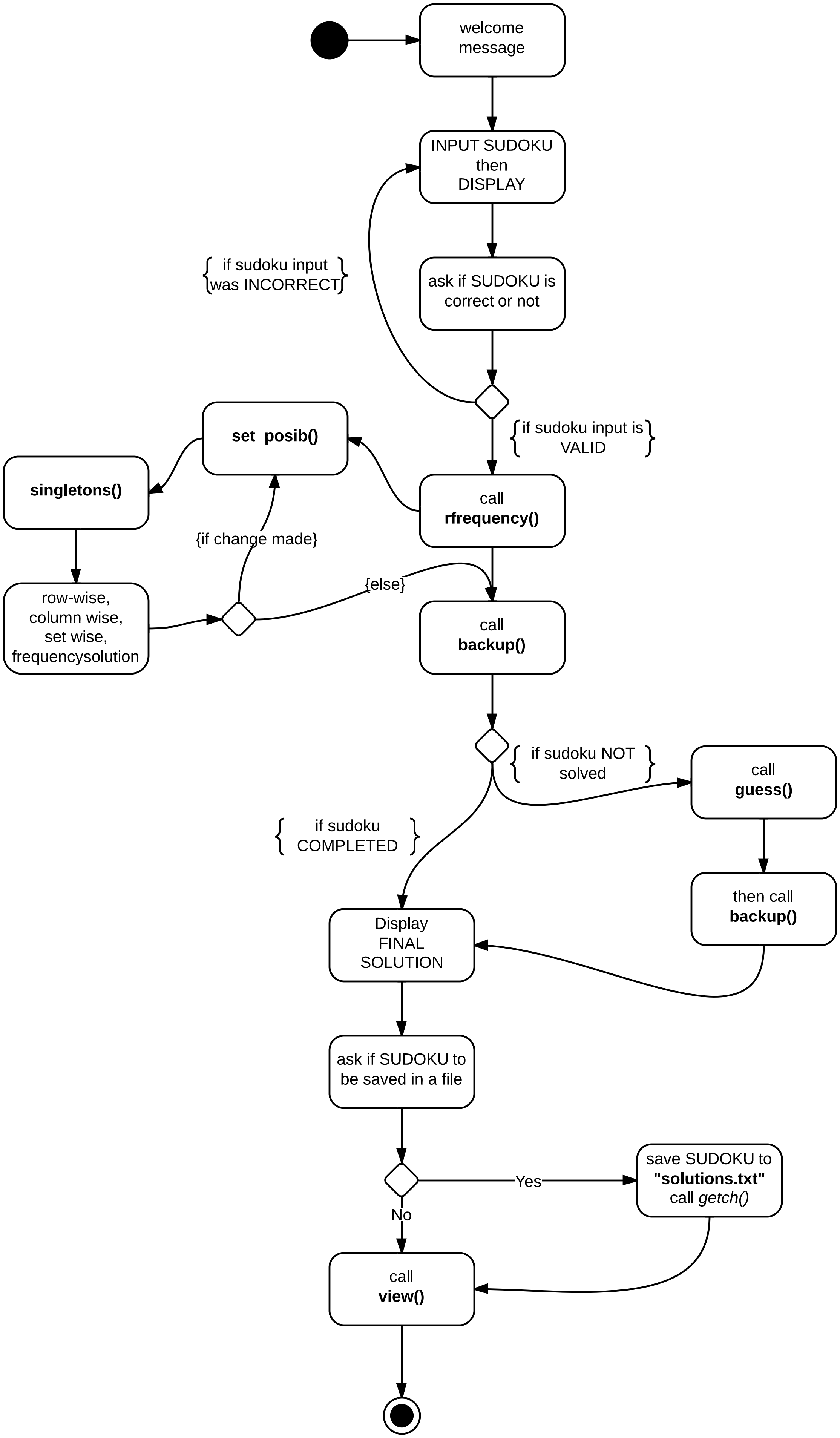
```
                    .call setsudoku()                    # to reset Sudoku #

                .goto starting of function


    #                                                                    #

    #   if Sudoku is incorrect, program runs into infinity over here   #

    #                                                                    #
```

view() : function to display final message

```
.print 'SUDOKU SOLVER By ABHISHEK'

     #  in fancy font  #
```

INPUT OF SUDOKU IN MAIN FUNCTION

```
    char ch;

    for( i : 1 to 9)

        for( j : 1 to 9)

            .Get character

            if(it is a valid character)

                .set the value of corresponding cell of Sudoku

                .display the input character

            else

                .goto input of character


#                                                     #
#   function displays ' * ' as guidelines for user to input    #

#                                                                          #
```

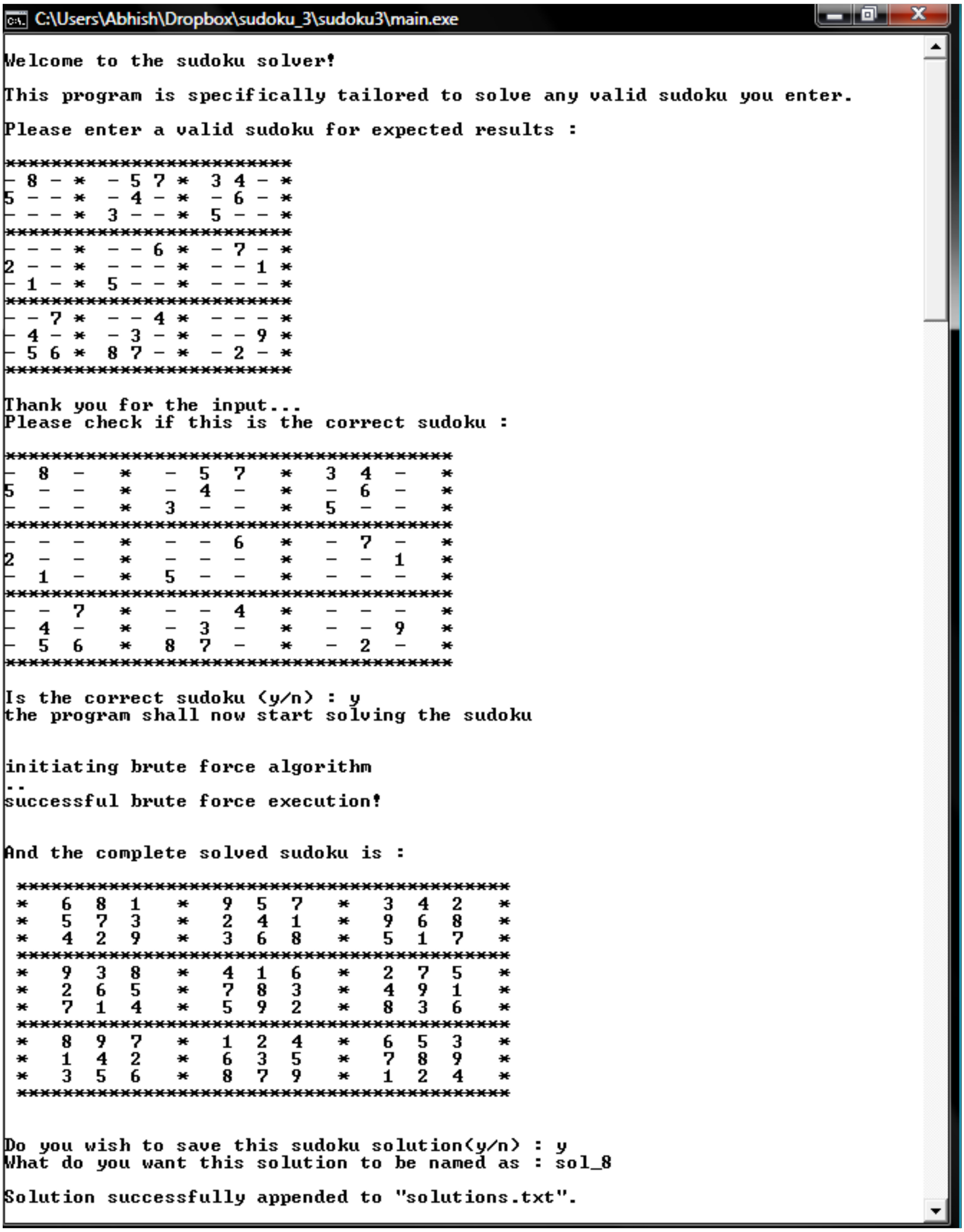

welcome message
INPUT SUDOKU then DISPLAY
ask if SUDOKU is correct or not
{ if sudoku input was INCORRECT }
{ if sudoku input is VALID }
set_posib()
singletons()
call rfrequency()
row-wise, column wise, set wise, frequencysolution
{if change made}
{else}
call backup()
{ if sudoku NOT solved }
call guess()
then call backup()
{ if sudoku COMPLETED }
Display FINAL SOLUTION
ask if SUDOKU to be saved in a file
Yes
save SUDOKU to "solutions.txt" call getch()
No
call view()

```
/**

    Sudoku Solver
        Abhishek Srikanth
        Class 12 - A
        Global Indian Int'l School

**/

#include <iostream>
#include <fstream>
#include <conio.h>

using namespace std;

class box
{
    public:
    int val;
    int nposib;
    int posib[9];
    box()   // sets val to ZERO, posib from 1-9
    {
        val = 0;
        nposib = 9;
        for(int i = 0; i < 9; ++i)
        {
            posib[i] = i + 1;
        }
    }
};
box sudoku[9][9];
box save[9][9];


void set_posib()
{
    // "Will now set posibilities for each value";

    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            if(sudoku[i][j].val!=0)     // IF SUDOKU[i][j] HAS A REAL VALUE
            {

                // Loop removes SUDOKU[i][j]'s value as a posib from corresponding row and column
                for(int m = 0; m < 9; ++m)
                {

                       if(sudoku[i][m].posib[ sudoku[i][j].val - 1 ] != 0)
                        {
                            sudoku[i][m].posib[ sudoku[i][j].val - 1 ] = 0;
                            sudoku[i][m].nposib--;      // change no. of possibilities
                        }
                        if(sudoku[m][j].posib[ sudoku[i][j].val - 1 ] != 0)
                        {
                            sudoku[m][j].posib[ sudoku[i][j].val - 1 ] = 0;
                            sudoku[m][j].nposib--;
                        }
                }

                /* Below body to find center point of corresponding quadrant */
                int Ci=-1,Cj=-1;
                if(i+1 == 1 || i+1 == 4 || i+1 == 7)
                    Ci = i+1;
                else if(i==1 || i==4 || i==7)
                    Ci = i;
                else if(i-1 == 1 || i-1 == 4 || i-1 == 7)
                    Ci = i-1;

                if(j+1 == 1 || j+1 == 4 || j+1 == 7)
                    Cj = j+1;
                else if(j==1 || j==4 || j==7)
                    Cj = j;
                else if(j-1 == 1 || j-1 == 4 || j-1 == 7)
                    Cj = j-1;
                /* Center point of quardrant is denoted by 'Ci' and 'Cj'    */

                for(int m = Ci-1; m < Ci+2; ++m)
                {
                    for(int n = Cj-1; n < Cj+2; ++n)
                    {
                        if(sudoku[m][n].posib[ sudoku[i][j].val -1 ] != 0)
                            {
                                sudoku[m][n].posib[ sudoku[i][j].val -1 ] = 0;
                                sudoku[m][n].nposib--;
                            }
                    }
                }
                /* Above nested loop accesses all values present in quadrant */

                sudoku[i][j].nposib = 0;
                for(int r = 0; r < 9; ++r)
                {
                    sudoku[i][j].posib[r] = 0;
                }
            }

        }
    }
}

void singletons()
{
    start_cuz_values_have_changed:
    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            if(sudoku[i][j].nposib == 1) // if only 1 possibility is present
            {
                for(int k = 0; k < 9; ++k)  // scan through posibilities
                {
                    if(sudoku[i][j].posib[k]!=0)  // If 'k'th possibility is NONZERO
                    {
                        sudoku[i][j].val = sudoku[i][j].posib[k];   // set value to that possibility
                        sudoku[i][j].nposib=0;                      // set number of possibilities to ZERO
                        for(int r = 0; r < 9; ++r)
                            sudoku[i][j].posib[r] = 0;

                        break;                                      // exit scanning possibilities
                    }
                }

            // To eliminate that value from corresponding ROW, COL, QUADRANT :

                    // Loop removes SUDOKU[i][j]'s value as a posib from corresponding row and column
                    for(int m = 0; m < 9; ++m)
                    {
                       if(sudoku[i][m].posib[ sudoku[i][j].val - 1 ] != 0)
                        {
                            sudoku[i][m].posib[ sudoku[i][j].val - 1 ] = 0;
                            sudoku[i][m].nposib--;
                        }
                        if(sudoku[m][j].posib[ sudoku[i][j].val - 1 ] != 0)
                        {
                            sudoku[m][j].posib[ sudoku[i][j].val - 1 ] = 0;
                            sudoku[m][j].nposib--;
                        }
                    }

                    /* Below body to find center point of corresponding quadrant */
                    int Ci=-1,Cj=-1;
                    if(i+1 == 1 || i+1 == 4 || i+1 == 7)
                        Ci = i+1;
                    else if(i==1 || i==4 || i==7)
                        Ci = i;
                    else if(i-1 == 1 || i-1 == 4 || i-1 == 7)
                        Ci = i-1;

                    if(j+1 == 1 || j+1 == 4 || j+1 == 7)
                        Cj = j+1;
                    else if(j==1 || j==4 || j==7)
                        Cj = j;
                    else if(j-1 == 1 || j-1 == 4 || j-1 == 7)
                        Cj = j-1;
                    /* Center point of quardrant is denoted by 'Ci' and 'Cj'    */

                    for(int m = Ci-1; m < Ci+2; ++m)
                    {
                        for(int n = Cj-1; n < Cj+2; ++n)
                        {
                            if(sudoku[m][n].posib[ sudoku[i][j].val -1 ] != 0)
                            {
                                sudoku[m][n].posib[ sudoku[i][j].val -1 ] = 0;
                                sudoku[m][n].nposib--;
                            }
                        }
                    }
                    /* Above nested loop accesses all values present in quadrant */

                goto start_cuz_values_have_changed;     // goes only if a value has been set
            }
        }
    }
}

void backup()
{
/**
    This set of code simply backs up
    the current sudoku so that guessing
    can be done                             **/

    // this results in save[][] being the same as sudoku
    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            save[i][j].val = sudoku[i][j].val;
            save[i][j].nposib = sudoku[i][j].nposib;
            for(int m = 0; m < 9; ++m)
                save[i][j].posib[m] = sudoku[i][j].posib[m];
        }
    }
}
```

```
void setsudoku()
{
/**
    This set of code simply resets
    the modulated sudoku so that guessing
    can be done                             **/

    // this results in sudoku[][] being the same as save[][]
    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            sudoku[i][j].val = save[i][j].val;
            sudoku[i][j].nposib = save[i][j].nposib;
            for(int m = 0; m < 9; ++m)
                sudoku[i][j].posib[m] = save[i][j].posib[m];
        }
    }
}

void rfrequency()
{
    set_posib();
    singletons();   // directly calls these functions, hence eliminating the need to call them in main()

    rowcheck:
    int counter = 0;    // to check whether row has frequency change or not
    // row-wise
    for(int i = 0; i < 9; ++i) // traversers from row 1-9
    {
        int freq[9] = {0,0,0,0,0,0,0,0,0};
        for(int j = 0; j < 9; ++j)
        {
            if(sudoku[i][j].nposib!=0) // if the values is not set already
                for(int k = 0; k < 9; ++k)
                {

                    if(sudoku[i][j].posib[k]!=0)    // and if the possibility is non-zero
                    {
                        freq[k]++;
                    }
                }
        }
        int n = 0;
        for(int k = 0; k < 9; ++k)
        {
            if(freq[k]==1)
            {
                n = k+1; // n holds value of number with 1 frequency
                break;
            }
        }
        // If number with 1 frequency exists
        if(n!=0)
        {
            ++counter;
            for(int j = 0; j < 9; ++j)  // for every element in that row
            {
                if(sudoku[i][j].nposib!=0)          // if value is already not present
                    if(sudoku[i][j].posib[n-1]!=0)  // cuz that is value with frequency 1
                    {
                        // set val
                        sudoku[i][j].val = n;
                        sudoku[i][j].nposib = 0;
                        for(int m = 0; m < 9; ++m)
                        {
                            sudoku[i][j].posib[m] = 0;
                        }
                        break;
                    }
            }
        }
    }
    if(counter!=0)
    {
        set_posib();    // if change has been made, call set_posib
        singletons();   // call singleton function, set singletons again cuz some new ones may be formed!
        goto rowcheck;  // restart row wise check
    }

    /**   ONCE ALL ROWS HAVE BEEN SET , START WORKING ON COLUMNS  **/

    int counter2 = 0;
    // col-wise
    for(int i = 0; i < 9; ++i) // traversers from col 1-9
    {
        int freq2[9] = {0,0,0,0,0,0,0,0,0};
        for(int j = 0; j < 9; ++j)
        {
            for(int k = 0; k < 9; ++k)
            {
                if(sudoku[j][i].nposib!=0) // if the values is not set already
                    if(sudoku[j][i].posib[k]!=0)    // and if the possibility is non-zero
                    {
                        freq2[k]++;
                    }
            }
        }
        int n2 = 0;
        for(int k = 0; k < 9; ++k)
        {
            if(freq2[k]==1)
            {
                n2 = k+1; // n2 holds value of number with ! frequency
                break;
            }
        }
        // If number with 1 frequency exists
        if(n2!=0)
        {
            ++counter2;
            for(int j = 0; j < 9; ++j)
            {
                if(sudoku[j][i].nposib!=0)          // if value is already not present
                    if(sudoku[j][i].posib[n2-1]!=0)  // cuz that is value with frequency 1
                    {
                        // set val
                        sudoku[j][i].val = n2;
                        sudoku[j][i].nposib = 0;
                        for(int m = 0; m < 9; ++m)
                        {
                            sudoku[j][i].posib[m] = 0;
                        }
                        break;
                    }
            }
        }
    }
    if(counter2!=0)
    {
        set_posib();    // if change has been made, call set_posib
        singletons();   // call singleton function, set singletons again cuz some new ones may be formed!
        goto rowcheck;  // restart row wise check
    }

    /**   ONCE ALL COLS HAVE BEEN SET , START WORKING ON QUADRANTS  **/

    // quardrant - vise
    int counter3 = 0;

    // note that the loop only gives i = j= {1,4,7} which are quadrant centers
    for(int i = 1; i < 8; i+=3)
    {
        for(int j = 1; j < 8; j+=3)
        {
            // for every box henceforth
            int freq3[9] = {0,0,0,0,0,0,0,0,0};
            for(int Ci = i-1; Ci<=i+1; ++Ci)
            {
                for(int Cj = j-1; Cj<=j+1; ++Cj)
                {
                    if(sudoku[Ci][Cj].nposib!=0)    // if the value has not been determined
                    {
                        for(int k = 0; k < 9; ++k)
                        {
                            if(sudoku[Ci][Cj].posib[k]!=0)  // if 'k'th posib exists,
                                freq3[k]++;
                        }
                    }
                }
            }
            int n3 = 0;
            for(int k = 0; k < 9; ++k)
            {
                if(freq3[k]==1)
                {
                    n3=k+1;
                    break;
                }
            }

            if(n3!=0)       // if a frequency 1 value exists
            {
                ++counter3;
                for(int Ci = i-1; Ci<=i+1; ++Ci)
                {
                    for(int Cj = j-1; Cj<=j+1; ++Cj)
                    {
                        // every element in the quadrant
                        for(int k = 0; k < 9; ++k)
                        {
                            if(sudoku[Ci][Cj].posib[n3-1] != 0) // if required box is located
                            {
                                sudoku[Ci][Cj].val = n3;
                                sudoku[Ci][Cj].nposib = 0;

                                for(int r = 0; r < 9; ++r)
                                    sudoku[Ci][Cj].posib[r] = 0;
                                break;

                            }
                        }
                    }
                }
            }
        }
    }
    if(counter3!=0)
    {
        set_posib();    // if change has been made, call set_posib
        singletons();   // call singleton function, set singletons again cuz some new ones may be formed!
        goto rowcheck;  // restart row wise check
    }
}


void guess()
{
    cout << "initiating brute force algorithm \n";
    starting:
    int row=-1,col=-1,val=0;
    for(int i =0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
        // goes through every element
            if(sudoku[i][j].val==0)
            {
                row=i;
                col=j;
                for(int k = 0; k < 9; ++k)
                {
                    if(sudoku[i][j].posib[k]!=0)
                    {
                        val = sudoku[i][j].posib[k];
                        sudoku[i][j].val = val;
                        sudoku[i][j].nposib=0;
                        goto loop_stop;
                    }
                }
            }
        }
    }
    loop_stop:

    for(int k = 0; k<9; ++k)
        sudoku[row][col].posib[k]=0;

    rfrequency();

    // sudoku with a guess has been solved
    // loop then runs to see if it worked

    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
        // for very element in the sudoku

            if(sudoku[i][j].val == 0)   // if not solved
            {
                // if no solution is possible
                // then make changes to save[][]
                // resetsudoku according to change
                if(sudoku[i][j].nposib==0)
                {
                    save[row][col].posib[val-1]=0;
                    save[row][col].nposib-=1;
                    save[row][col].val=0;       // just incase
                    setsudoku();
                }
                cout << '.';
                goto starting;
            }
```

```cpp
        }
    }
    cout << "\nsuccessful brute force execution!\n";

}


// the final message!
void view()
{
    cout << endl << endl;
    cout <<" ######  ##     ## ########   #######  ##    ## ##     ## "<< endl;
    cout <<"##    ## ##     ## ##     ## ##     ## ##   ##  ##     ## "<< endl;
    cout <<"##       ##     ## ##     ## ##     ## ##  ##   ##     ## "<< endl;
    cout <<" ######  ##     ## ##     ## ##     ## #####    ##     ## "<< endl;
    cout <<"      ## ##     ## ##     ## ##     ## ##  ##   ##     ## "<< endl;
    cout <<"##    ## ##     ## ##     ## ##     ## ##   ##  ##     ## "<< endl;
    cout <<" ######   #######  ########   #######  ##    ##  ####### "<< endl;
    cout << endl;
    cout <<" ######   #######  ##       ##     ## ######## ######## "<< endl;
    cout <<"##    ## ##     ## ##       ##     ## ##       ##     ## "<< endl;
    cout <<"##       ##     ## ##       ##     ## ##       ##     ## "<< endl;
    cout <<" ######  ##     ## ##       ##     ## ######   ######## "<< endl;
    cout <<"      ## ##     ## ##        ##   ##  ##       ##   ## "<< endl;
    cout <<"##    ## ##     ## ##         ## ##   ##       ##    ## "<< endl;
    cout <<" ######   #######  ########    ###    ######## ##     ## "<< endl;
    cout << endl
         << endl
         << endl
         << "              BBBB       \n"
         << "              B   B      \n"
         << "              BBBB  y  y \n"
         << "              B   B y  y \n"
         << "              BBBB   yyy \n"
         << "                      y \n"
         << "                   yyy  \n"
         << endl
         << endl
         << endl
         << "    #                                                " << endl
         << "   # #   #####  #    # #  ####  #    # ###### #    # " << endl
         << "  #   #  #    # #    # # #      #    # #      #   #  " << endl
         << " #     # #####  ###### #  ####  ###### #####  ####   " << endl
         << " ####### #    # #    # #      # #    # #      #  #   " << endl
         << " #     # #    # #    # # #    # #    # #      #   #  " << endl
         << " #     # #####  #    # #  ####  #    # ###### #    # " << endl
         << endl
         << endl
         << endl;

}

int main()
{
    cout << endl;
    cout << "Welcome to the sudoku solver! \n";
    cout << endl;
    cout << "This program is specifically tailored to solve any valid sudoku you enter.\n";
    b:
    cout << endl;
    cout << "Please enter a valid sudoku for expected results : \n\n";
    char ch;
/********************************* INPUT **********************************************/

    cout << "*************************" << endl;
    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            a:
            ch = getch();
            if(ch > '0' && ch <= '9')
            {
                sudoku[i][j].val = (int)ch - 48;
                cout << sudoku[i][j].val;
            }
            else if(ch=='\n' || ch=='\r')
            {
                sudoku[i][j].val = 0;
                cout << "-";
            }
            else
            goto a;
            if((j+1)%3==0)
                cout << " * ";
            cout << " ";
        }
        cout << endl;
        if((i+1)%3==0)
            cout << "*************************" << endl;
    }

/***************************** DISPLAY ***********************************************/

    cout << "\nThank you for the input..." << endl;
    cout << "Please check if this is the correct sudoku : \n\n";

    cout << "***************************************" << endl;
    for(int i = 0; i < 9; ++i)
    {
        for(int j = 0; j < 9; ++j)
        {
            if(sudoku[i][j].val!=0)
                cout << sudoku[i][j].val << "  ";
            else cout << "-  ";

            if((j+1)%3==0)
                    cout << " *   ";
        }
        cout << endl;
        if((i+1)%3==0)
            cout << "***************************************" << endl;
    }
    cout << endl << "Is the correct sudoku (y/n) : " ;
    cin >> ch;
    if(ch=='N' || ch == 'n')
        goto b;
    else
    cout << "the program shall now start solving the sudoku \n\n";

/*********************************** SOLUTION **************************************/

    rfrequency();
    backup();

    cout << endl;

/****************** Call for guessing ***************************/

for(int i = 0; i < 9; ++i)
{
    for(int j = 0; j < 9; ++j)
    {
        if(save[i][j].val==0)
        {
            guess();
            backup();
            goto loop_term;
        }
    }
}
loop_term:

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

        cout << "\n\nAnd the complete solved sudoku is : \n\n";

    // display after brute force solution
        cout << " *******************************************" << endl;
        for(int i = 0; i < 9; ++i)
        {
            cout << " *   ";
            for(int j = 0; j < 9; ++j)
            {
                if(sudoku[i][j].val!=0)
                    cout << sudoku[i][j].val << "  ";
                else cout << "-  ";
                if((j+1)%3==0)
                    cout << " *   ";
            }
            cout << endl;
            if((i+1)%3==0)
                cout << " *******************************************" << endl;
        }

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

    cout << "\n\nDo you wish to save this sudoku solution(y/n) : ";
    cin >> ch;
    if(ch=='y' || ch == 'Y')
    {
        ofstream solution("solutions.txt", ios_base::app | ios::out);
        cout << "What do you want this solution to be named as : ";
        char puzzle_name[10];
        cin >> puzzle_name;
        solution << endl << puzzle_name << endl;
        solution << "***************************************\n";
        for(int i = 0; i < 9; ++i)
        {
            for(int j = 0; j < 9; ++j)
            {
                solution << sudoku[i][j].val << "  ";
                if((j+1)%3==0)
                    solution << " *   ";
            }
            solution << endl;
            if((i+1)%3==0)
                solution << "***************************************\n";
        }

        solution.close();
        cout << endl
             << "Solution successfully appended to \"solutions.txt\"."
             << endl << endl;
        getch();
    }

    cout << "\n\n\n\n\nThank you for using this programme and i hope it impressed you!\n\n\n" << endl <<
```

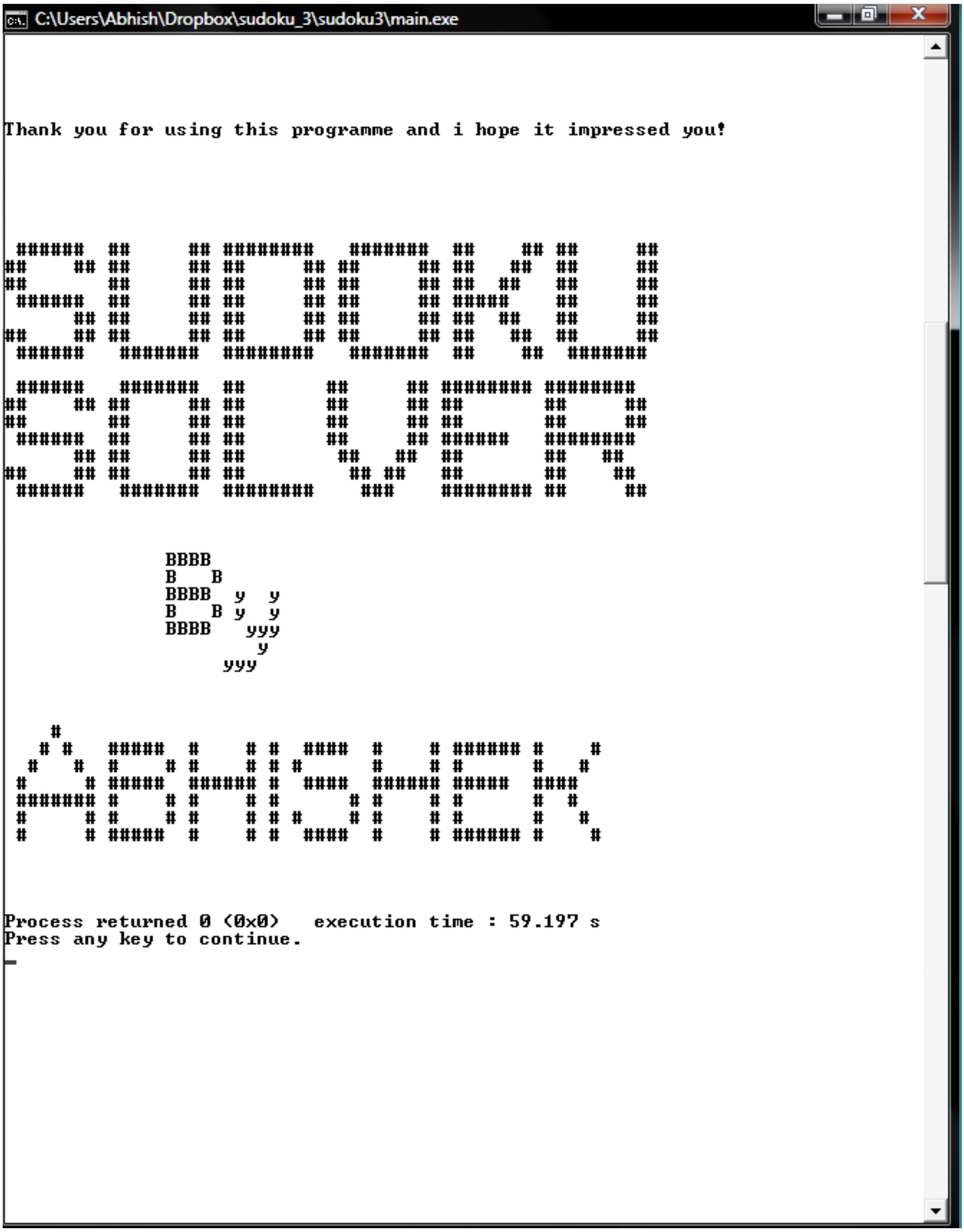
```
endl;
    view();

    return 0;
}
```



```
Welcome to the sudoku solver!

This program is specifically tailored to solve any valid sudoku you enter.

Please enter a valid sudoku for expected results :

*************************
- 8 - *  - 5 7 *  3 4 - *
5 - - *  - 4 - *  - 6 - *
- - - *  3 - - *  5 - - *
*************************
- - - *  - - 6 *  - 7 - *
2 - - *  - - - *  - - 1 *
- 1 - *  5 - - *  - - - *
*************************
- - 7 *  - - 4 *  - - - *
- 4 - *  - 3 - *  - - 9 *
- 5 6 *  8 7 - *  - 2 - *
*************************

Thank you for the input...
Please check if this is the correct sudoku :

***************************************
-  8  -   *   -  5  7   *   3  4  -   *
5  -  -   *   -  4  -   *   -  6  -   *
-  -  -   *   3  -  -   *   5  -  -   *
***************************************
-  -  -   *   -  -  6   *   -  7  -   *
2  -  -   *   -  -  -   *   -  -  1   *
-  1  -   *   5  -  -   *   -  -  -   *
***************************************
-  -  7   *   -  -  4   *   -  -  -   *
-  4  -   *   -  3  -   *   -  -  9   *
-  5  6   *   8  7  -   *   -  2  -   *
***************************************

Is the correct sudoku (y/n) : y
the program shall now start solving the sudoku


initiating brute force algorithm
..
successful brute force execution!


And the complete solved sudoku is :

 *******************************************
 *   6  8  1   *   9  5  7   *   3  4  2   *
 *   5  7  3   *   2  4  1   *   9  6  8   *
 *   4  2  9   *   3  6  8   *   5  1  7   *
 *******************************************
 *   9  3  8   *   4  1  6   *   2  7  5   *
 *   2  6  5   *   7  8  3   *   4  9  1   *
 *   7  1  4   *   5  9  2   *   8  3  6   *
 *******************************************
 *   8  9  7   *   1  2  4   *   6  5  3   *
 *   1  4  2   *   6  3  5   *   7  8  9   *
 *   3  5  6   *   8  7  9   *   1  2  4   *
 *******************************************


Do you wish to save this sudoku solution(y/n) : y
What do you want this solution to be named as : sol_8

Solution successfully appended to "solutions.txt".
```

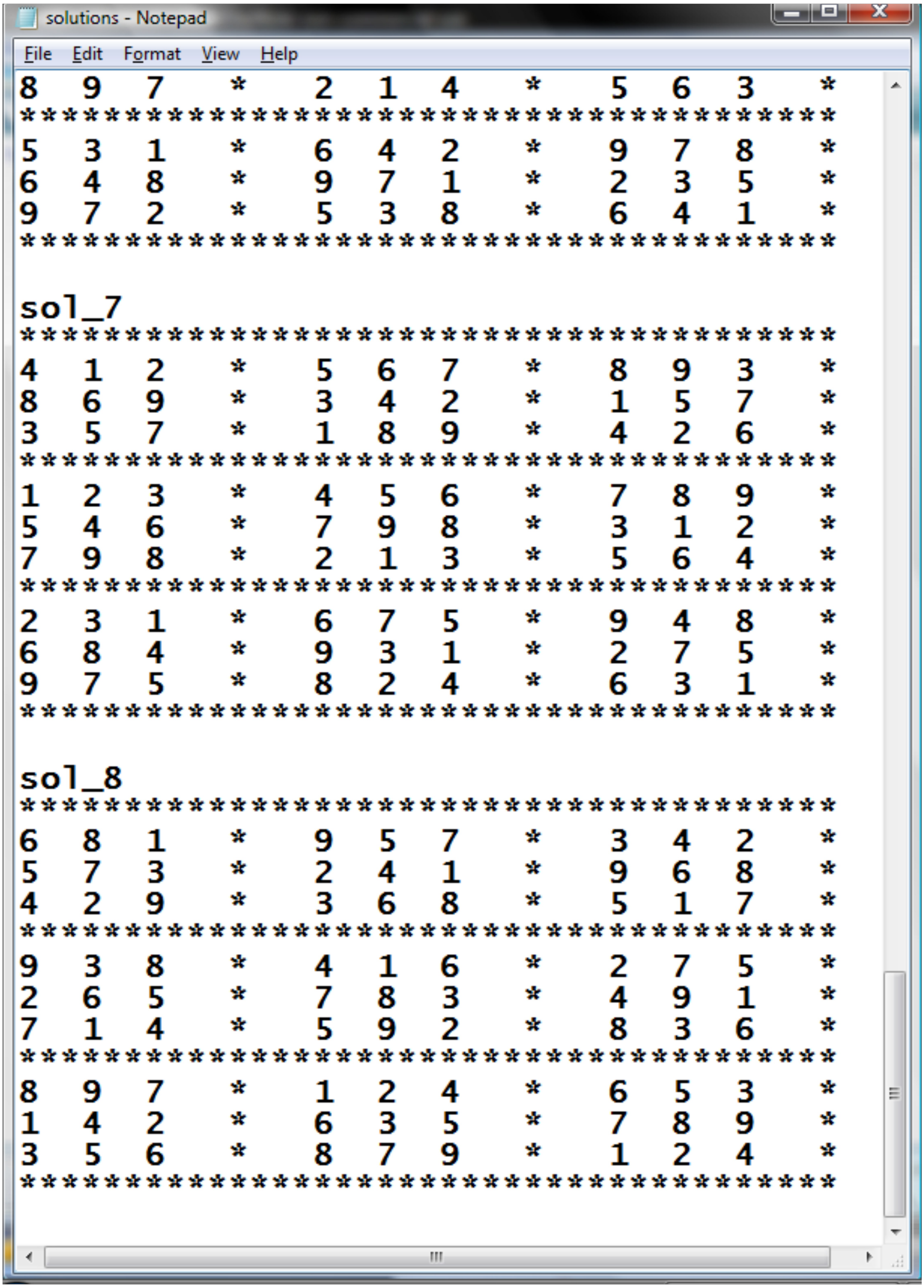
This is the first output screen shot.
Sudoku question taken from www.websudoku.com

Solution displayed
Solution saved as “sol_8” in “solutions.txt”



```
Thank you for using this programme and i hope it impressed you!




 ######  ##     ## ########   #######  ##    ## ##     ##
##    ## ##     ## ##     ## ##     ## ##   ##  ##     ##
##       ##     ## ##     ## ##     ## ##  ##   ##     ##
 ######  ##     ## ##     ## ##     ## #####    ##     ##
      ## ##     ## ##     ## ##     ## ##  ##   ##     ##
##    ## ##     ## ##     ## ##     ## ##   ##  ##     ##
 ######   #######  ########   #######  ##    ##  #######

 ######   #######  ##       ##     ## ######## ########
##    ## ##     ## ##       ##     ## ##       ##     ##
##       ##     ## ##       ##     ## ##       ##     ##
 ######  ##     ## ##       ##     ## ######   ########
      ## ##     ## ##        ##   ##  ##       ##   ##
##    ## ##     ## ##         ## ##   ##       ##    ##
 ######   #######  ########    ###    ######## ##     ##


              BBBB
              B   B
              BBBB  y  y
              B   B y  y
              BBBB   yyy
                      y
                   yyy


    #
   # #   #####  #    # #  ####  #    # ###### #    #
  #   #  #    # #    # # #      #    # #      #   #
 #     # #####  ###### #  ####  ###### #####  ####
 ####### #    # #    # #      # #    # #      #  #
 #     # #    # #    # # #    # #    # #      #   #
 #     # #####  #    # #  ####  #    # ###### #    #



Process returned 0 (0x0)   execution time : 59.197 s
Press any key to continue.
```

Final Thank you message
and
Project title



```
8  9  7   *   2  1  4   *   5  6  3   *
***************************************
5  3  1   *   6  4  2   *   9  7  8   *
6  4  8   *   9  7  1   *   2  3  5   *
9  7  2   *   5  3  8   *   6  4  1   *
***************************************

sol_7
***************************************
4  1  2   *   5  6  7   *   8  9  3   *
8  6  9   *   3  4  2   *   1  5  7   *
3  5  7   *   1  8  9   *   4  2  6   *
***************************************
1  2  3   *   4  5  6   *   7  8  9   *
5  4  6   *   7  9  8   *   3  1  2   *
7  9  8   *   2  1  3   *   5  6  4   *
***************************************
2  3  1   *   6  7  5   *   9  4  8   *
6  8  4   *   9  3  1   *   2  7  5   *
9  7  5   *   8  2  4   *   6  3  1   *
***************************************

sol_8
***************************************
6  8  1   *   9  5  7   *   3  4  2   *
5  7  3   *   2  4  1   *   9  6  8   *
4  2  9   *   3  6  8   *   5  1  7   *
***************************************
9  3  8   *   4  1  6   *   2  7  5   *
2  6  5   *   7  8  3   *   4  9  1   *
7  1  4   *   5  9  2   *   8  3  6   *
***************************************
8  9  7   *   1  2  4   *   6  5  3   *
1  4  2   *   6  3  5   *   7  8  9   *
3  5  6   *   8  7  9   *   1  2  4   *
***************************************
```

Sol_8 is the solution to question

Previous Sudoku answers for other Sudoku puzzles

Acknowledgement

I would like to acknowledge the following for the success of my programme:

- My parents and teachers for their support
- Various books and the internet for the vast information available
- *www.websudoku.com* for the huge database of unsolved puzzles used for testing
- Code::blocks for a wonderful and user friendly open source, cross-platform IDE.

Bibliography

1. http://en.wikipedia.org/wiki/C%2B%2B
2. http://en.wikipedia.org/wiki/Code::Blocks
3. http://en.wikipedia.org/wiki/Sudoku
4. http://gcc.gnu.org/
5. http://stackoverflow.com/questions/7023071/number-of-possible-sudoku-puzzles
6. http://www.sciencebuddies.org/engineering-design-process/design-requirements-examples.shtml
7. http://www.eddaardvark.co.uk/sudokusolver.html
8. http://www.websudoku.com
9. http://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20