

```

#include <iostream>
#include<stdlib.h>
#include<time.h>
#include<fstream>
#include<simplecpp>
#include<stdio.h>
#include<simplecpp>
#include<conio.h>
#include<ctime>

using namespace std;

// UNASSIGNED is used for empty cells in sudoku grid


// N is used for size of Sudoku grid. Size will be NxN
#define N 9


// This function finds an entry in grid that is still empty
bool FindUnassignedLocation(int grid[N][N], int &row, int &col);


// Checks whether it will be legal to assign num to the given row,col
bool isSafe(int grid[N][N], int row, int col, int num);


int sudokugenerator(int grid[][9],int choice);

int q=0;

/* Takes a partially filled-in grid and attempts to assign values to
all unassigned locations in such a way to meet the requirements
for Sudoku solution (non-duplication across rows, columns, and boxes) */


time_t timer1,timer2;

bool SolveSudoku(int grid[N][N])

```

```

{
    q++;
    int row, col, c=0;
    label:
    c++;
    if(c==2)
        return false;
    // If there is no unassigned location, we are done
    if (!FindUnassignedLocation(grid, row, col))
        return true; // success!

    // consider digits 1 to 9
    for (int num = 1; num <= 9; num++)
    {
        // if looks promising
        if (isSafe(grid, row, col, num))
        {
            // make tentative assignment
            grid[row][col] = num;

            if (SolveSudoku(grid))
                return true;

            // failure, unmake & try again
            grid[row][col] = 0;
        }
    }
}

goto label;

```

```
    return false;

    // this triggers backtracking
}
```

```
/* Searches the grid to find an entry that is still unassigned. If
   found, the reference parameters row, col will be set the location
   that is unassigned, and true is returned. If no unassigned entries
   remain, false is returned. */
```

```
bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
    for (row = 0; row < 9; row++)
        for (col = 0; col < 9; col++)
            if (grid[row][col] == 0)
                return true;
    return false;
}
```

```
/* Returns a boolean which indicates whether any assigned entry
   in the specified row matches the given number. */
```

```
bool UsedInRow(int grid[9][9], int row, int num)
{
    for (int col = 0; col < 9; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

```
/* Returns a boolean which indicates whether any assigned entry
   in the specified column matches the given number. */
```

```
bool UsedInCol(int grid[9][9], int col, int num)
```

```
{  
    for (int row = 0; row < 9; row++)  
        if (grid[row][col] == num)  
            return true;  
    return false;  
}
```

```
/* Returns a boolean which indicates whether any assigned entry  
within the specified 3x3 box matches the given number. */
```

```
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num)
```

```
{  
  
    for (int row = 0; row < 3; row++)  
        for (int col = 0; col < 3; col++)  
            if (grid[row+boxStartRow][col+boxStartCol] == num)  
                return true;  
    return false;  
}
```

```
/* Returns a boolean which indicates whether it will be legal to assign  
num to the given row,col location. */
```

```
bool isSafe(int grid[9][9], int row, int col, int num)
```

```
{  
    /* Check if 'num' is not already placed in current row,  
    current column and current 3x3 box */  
    if(q==5000)  
    {
```

```

        return false;
    }

    bool flag=!UsedInRow(grid, row, num) &&
        !UsedInCol(grid, col, num) &&
        !UsedInBox(grid, row - row%3 , col - col%3, num);
    if(flag==false)
        return false;
    else
        return true;
}

```

/* A utility function to print grid in inCanvas() window and produce exit button on printing*/

```

void printGrid(int grid[N][N])
{
    int i,j;
    Rectangle r(220 , 220, 360, 360); r.setColor(GREEN);

    Line lv1(80,40 , 80,400);           //displaying grid
    Line lv2(120,40 , 120,400);
    Line lv3(160,40 , 160,400); lv3.setColor(GREEN);
    Line lv4(200,40 , 200,400);
    Line lv5(240,40 , 240,400);
    Line lv6(280,40 , 280,400); lv6.setColor(GREEN);
    Line lv7(320,40 , 320,400);
    Line lv8(360,40 , 360,400);

    Line lh1(40,80 , 400,80);
    Line lh2(40,120 , 400,120);
    Line lh3(40,160 , 400,160); lh3.setColor(GREEN);
}

```

Line lh4(40,200 , 400,200);

Line lh5(40,240 , 400,240);

Line lh6(40,280 , 400,280); lh6.setColor(GREEN);

Line lh7(40,320 , 400,320);

Line lh8(40,360 , 400,360);

Rectangle b1(60,440,40,40); Text t1(60,440,"1");

Rectangle b2(100,440,40,40); Text t2(100,440,"2");

Rectangle b3(140,440,40,40); Text t3(140,440,"3");

Rectangle b4(180,440,40,40); Text t4(180,440,"4");

Rectangle b5(220,440,40,40); Text t5(220,440,"5");

Rectangle b6(260,440,40,40); Text t6(260,440,"6");

Rectangle b7(300,440,40,40); Text t7(300,440,"7");

Rectangle b8(340,440,40,40); Text t8(340,440,"8");

Rectangle b9(380,440,40,40); Text t9(380,440,"9");

Rectangle b10(440,440,60,40); Text t10(440,440,"DELETE");

Rectangle restart(460,50,60,40); Text t11(460,50, "RESET");

Rectangle submit(460,100,60,40); Text t12(460,100,"SUBMIT");

Text *d;

for(i=0;i<9;i++)

{

for(j=0;j<9;j++)

{

switch(grid[i][j])

{

case 1:d=new Text(60+40*j,60+40*i,"1");break;

case 2:d=new Text(60+40*j,60+40*i,"2");break;

case 3:d=new Text(60+40*j,60+40*i,"3");break;

case 4:d=new Text(60+40*j,60+40*i,"4");break;

```

        case 5:d=new Text(60+40*j,60+40*i,"5");break;
        case 6:d=new Text(60+40*j,60+40*i,"6");break;
        case 7:d=new Text(60+40*j,60+40*i,"7");break;
        case 8:d=new Text(60+40*j,60+40*i,"8");break;
        case 9:d=new Text(60+40*j,60+40*i,"9");break;
    }
    d->setColor(BLUE);
}
}
Rectangle cont(460,150,60,40); Text t13(460,150,"EXIT");
repeat(10)
{
    int clickPos = getClick();
    int cx = clickPos/65536;
    int cy = clickPos % 65536;
    if(cx<=490 && cx>=430 && cy<=160 && cy>=140)
    {
        break;
    }
}

}

//used to open file and read the unsolved sudoku in the grid
int sudokugenerator(int grid[][9])
{
    int i,j,r,k;
    int sample[9][9];
    double s;
    char str[82];

```

```

time(&timer2);          //generating random numbers

FILE *fp;

system("cls");          //clears the screen

s=difftime(timer2,timer1);

r=(int)s%10;

Rectangle r1(220,220,160,40);    //displays the rectangle for difficulty level

Text game(220,220,"1.EASY");


Rectangle r2(220,260,160,40);

Text game2(220,260,"2.MEDIUM");


Rectangle r3(220,300,160,40);

Text game3(220,300,"3.HARD");


int clickpos3=getClick();        //gets position of mouseclick

int cx3=clickpos3/65536;

int cy3=clickpos3%65536;

if(cx3>=220-160/2 && cx3<=220+160/2 && cy3>=220-40/2 && cy3<=220+40/2)

{

    switch(r)                //opening file

    {

        case 1:

            fp=fopen("sudoku1.txt","r");

            break;

        case 2:

            fp=fopen("sudoku2.txt","r");

            break;

        case 3:

            fp=fopen("sudoku3.txt","r");

```



```

        break;
case 4:
    fp=fopen("sudoku4.txt","r");
    break;
case 5:
    fp=fopen("sudoku5.txt","r");
    break;
case 6:
    fp=fopen("sudoku6.txt","r");
    break;
case 7:
    fp=fopen("sudoku7.txt","r");
    break;
case 8:
    fp=fopen("sudoku8.txt","r");
    break;
case 9:
    fp=fopen("sudoku9.txt","r");
    break;
    }
}
else if(cx3>=220-160/2 && cx3<=220+160/2 && cy3>=260-40/2 && cy3<=260+40/2)
{
    switch(r)
    {
        case 1:
            fp=fopen("sudoku10.txt","r");
            break;
        case 2:

```

```

        fp=fopen("sudoku11.txt","r");
        break;
case 3:
        fp=fopen("sudoku12.txt","r");
        break;
case 4:
        fp=fopen("sudoku13.txt","r");
        break;
case 5:
        fp=fopen("sudoku14.txt","r");
        break;
case 6:
        fp=fopen("sudoku15.txt","r");
        break;
case 7:
        fp=fopen("sudoku16.txt","r");
        break;
case 8:
        fp=fopen("sudoku17.txt","r");
        break;
case 9:
        fp=fopen("sudoku18.txt","r");
        break;
    }

}

```

```

else if(cx3>=220-160/2 && cx3<=220+160/2 && cy3>=300-40/2 && cy3<=300+40/2)

```

```

{

```

```
switch(r)
{
    case 1:
        fp=fopen("sudoku19.txt","r");
        break;
    case 2:
        fp=fopen("sudoku20.txt","r");
        break;
    case 3:
        fp=fopen("sudoku21.txt","r");
        break;
    case 4:
        fp=fopen("sudoku22.txt","r");
        break;
    case 5:
        fp=fopen("sudoku23.txt","r");
        break;
    case 6:
        fp=fopen("sudoku24.txt","r");
        break;
    case 7:
        fp=fopen("sudoku25.txt","r");
        break;
    case 8:
        fp=fopen("sudoku26.txt","r");
        break;
    case 9:
        fp=fopen("sudoku27.txt","r");
        break;
```

```

    }
}
if(fp==NULL)        //if file does not exists
{
    cout<<"\nFile does not exists";
}
else
{
    k=0;
    fread(str,1,81,fp);
    for(i=0;i<9;i++)
    {
        for(j=0;j<9;j++)
        {
            sample[i][j]=(int)str[k]-48;
            k++;
        }
    }
    for(i=0;i<9;i++)
    {
        for(j=0;j<9;j++)
        {
            grid[i][j]=sample[i][j];
        }
    }
}

return 0;
}

```

```

/* Driver Program to test above functions */

int main()
{

    time(&timer1);

    initCanvas();                //displaying the window

    int grid[N][N],sample[9][9] ,row,column;

    int i,j,choice,cx,cy,cx5,cy5,ch,mark2;

    bool flag=true;

    do

    {

        system("cls");

        cout<<"*****\n";

        cout<<"*1.Sudoku Generator          *\n";

        cout<<"*2.Sudoku Auto solver          *\n";

        cout<<"*****\n";

        cin>>choice;

        if(choice==1)

        {

            system("cls");

            cout<<"*****\n";

            cout<<"*1.TAKE YOUR TIME AND SOLVE          *\n";

            cout<<"*2.TIME LIMITED SOLVING          *\n";

            cout<<"*****\n";

            cin>>ch;

```

```

time_t timerstart,timerend;    //TIME objects(for start and stop)

double seconds;

sudokugenerator(grid);        //function call to grid

//int
grid[9][9]={0,0,0,5,0,0,0,0,0},{0,0,0,0,1,0,0,0,0},{5,0,0,8,0,2,9,1,0},{0,0,0,0,5,0,0,0,0},{0,0,0,0,0,0,0,0,4},{
0,0,0,0,0,0,0,6,5},{0,0,3,4,0,9,0,0,0},{0,6,0,0,0,8,2,0,0},{2,9,0,7,0,0,0,0,0}};

//displaying the grid on initCanvas();

Rectangle r(220 , 220, 360, 360); r.setColor(GREEN);


Line lv1(80,40 , 80,400);

Line lv2(120,40 , 120,400);

Line lv3(160,40 , 160,400); lv3.setColor(GREEN);

Line lv4(200,40 , 200,400);

Line lv5(240,40 , 240,400);

Line lv6(280,40 , 280,400); lv6.setColor(GREEN);

Line lv7(320,40 , 320,400);

Line lv8(360,40 , 360,400);


Line lh1(40,80 , 400,80);

Line lh2(40,120 , 400,120);

Line lh3(40,160 , 400,160); lh3.setColor(GREEN);

Line lh4(40,200 , 400,200);

Line lh5(40,240 , 400,240);

Line lh6(40,280 , 400,280); lh6.setColor(GREEN);

Line lh7(40,320 , 400,320);

Line lh8(40,360 , 400,360);


Rectangle b1(60,440,40,40); Text t1(60,440,"1");

Rectangle b2(100,440,40,40); Text t2(100,440,"2");

```

```

Rectangle b3(140,440,40,40); Text t3(140,440,"3");
Rectangle b4(180,440,40,40); Text t4(180,440,"4");
Rectangle b5(220,440,40,40); Text t5(220,440,"5");
Rectangle b6(260,440,40,40); Text t6(260,440,"6");
Rectangle b7(300,440,40,40); Text t7(300,440,"7");
Rectangle b8(340,440,40,40); Text t8(340,440,"8");
Rectangle b9(380,440,40,40); Text t9(380,440,"9");
Rectangle b10(440,440,60,40); Text t10(440,440,"DELETE");
Rectangle restart(460,50,60,40); Text t11(460,50, "RESET");
Rectangle submit(460,100,60,40); Text t12(460,100,"SUBMIT");
for(i=0;i<9;i++)
    {
        for(j=0;j<9;j++)
        {
            sample[i][j]=grid[i][j];
        }
    }
Text *m;
int marks=0;
for(i=0;i<9;i++)          //displaying the sudoku opened from files
{
    for(j=0;j<9;j++)
    {
        if(grid[i][j]!=0)
        { marks=marks-1;
          m=new Text(60+j*40,60+40*i,sample[i][j]);
        }
    }
}

```

```

time(&timerstart);          //starting the timer

repeat(100)                 //repeating upto 100 mouseclicks
{
    Text *t;
    int clickPos = getClick(); //getting mouseclick position
    cx = clickPos/65536;
    cy = clickPos % 65536;
    //program to reset the elements entered by user
    if(cx<=490 && cx>=430 && cy<=60 && cy>=40)
    {
        for(i=80; i<440;i+=40)
        {
            for(j=80;j<440;j+=40)
            {
                t=new Text(i-20,j-20," ");
            }
        }
        for(i=0;i<9;i++)
        {
            for(j=0;j<9;j++)
            {
                if(sample[i][j]!=0)
                {
                    m=new Text(60+j*40,60+40*i,sample[i][j]);
                }
            }
        }
    }
}

```



```
//CALCULATING MARKS
```

```
if(cx<=490 && cx>=430 && cy<=110 && cy>=90)
```

```
{
```

```
    q=0;
```

```
    flag=SolveSudoku(sample);
```

```
    mark2=0;
```

```
    for(i=0;i<9;i++)
```

```
    {
```

```
        for(j=0;j<9;j++)
```

```
        {
```

```
            if(sample[i][j]==grid[i][j])
```

```
            {
```

```
                marks++;
```

```
                mark2++;
```

```
            }
```

```
        }
```

```
    }
```

```
    system("cls");
```

```
    cout<<"*****\n";
```

```
    cout<<"*SCORES="<<marks<<"          *\n";
```

```
    cout<<"*****\n";
```

```
    if(mark2==81)
```

```
    {
```

```
        cout<<"*****\n";
```

```
        cout<<"*CONGRATS!! YOU HAVE SOLVED IT CORRECTLY*\n";
```

```
        cout<<"*****\n";
```

```
    }
```

```
    break;
```

```

}
for(i=80; i<440;i+=40)
{
    for(j=80; j<440;j+=40)
    {
        if((i-20)-40/2<= cx && cx<= (i-20)+40/2 && (j-20)-40/2 <= cy && cy <= (j-20)+40/2)
        {
            Rectangle r4(i-20,j-20 , 30,30);
            int clickPos2=getClick();
            int cx1= clickPos2/65536;
            int cy1=clickPos2 % 65536;
            if(cx1>=410&&cx1<=470&&cy1>=420&&cy1<=460) //DELETE element from box
            {
                t=new Text(i-20,j-20," ");
                for(i=0;i<9;i++)
                {
                    for(j=0;j<9;j++)
                    {
                        if(sample[i][j]!=0)

                            m=new Text(60+j*40,60+40*i,sample[i][j]);

                    }
                }
            }
        }
    }
    //printing the element into grid
    for(int k=80; k<440; k+=40)
    {

```

```

(440)+40/2)
        if((k-20)-40/2<= cx1 && cx1<= (k-20)+40/2 && cy1>=(440)-40/2 && cy1 <=
        {
            column=(i-40)/40-1;
            row=(j-40)/40-1;
            switch((k/40)-1)
            {

                case 1: t=new Text(i-20,j-20,"1"); grid[row][column]=1;break;
                case 2: t=new Text(i-20,j-20,"2"); grid[row][column]=2;break;
                case 3: t=new Text(i-20,j-20,"3"); grid[row][column]=3;break;
                case 4: t=new Text(i-20,j-20,"4"); grid[row][column]=4;break;
                case 5: t=new Text(i-20,j-20,"5"); grid[row][column]=5;break;
                case 6: t=new Text(i-20,j-20,"6"); grid[row][column]=6;break;
                case 7: t=new Text(i-20,j-20,"7"); grid[row][column]=7;break;
                case 8: t=new Text(i-20,j-20,"8"); grid[row][column]=8;break;
                case 9: t=new Text(i-20,j-20,"9"); grid[row][column]=9;break;
            }
        }
    }
}

//time limited solving
if(ch==2)
{

    time(&timerend);
    seconds = difftime(timerend,timerstart);
    if(seconds==300)

```

```

{
    Rectangle time1(220,220,160,40);

    Text time(220,220,"Your time is up");

    q=0;

    flag=SolveSudoku(sample);

    mark2=0;

    for(i=0;i<9;i++)
    {
        for(j=0;j<9;j++)
        {
            if(sample[i][j]==grid[i][j])
            {
                marks++;

                mark2++;

            }
        }
    }

    system("cls");

    cout<<"*****\n";

    cout<<"*SCORES="<<marks<<"          *\n";

    cout<<"*****\n";

    if(mark2==81)
    {
        cout<<"*****\n";

        cout<<"*CONGRATS YOU HAVE SOLVED IT CORRECTLY *\n";

        cout<<"*****\n";

    }

    break;

```

```

    }

    }

}

//take your time and solve
if(ch==1)
{
    time(&timerend);
    seconds = difftime(timerend,timerstart);
    cout<<"*****\n";
    cout<<"*TIME TAKEN="<<seconds<<"      *\n";
    cout<<"*****\n";

}

printGrid(sample);

}

```

```

//auto solver
    else if(choice==2)
    {
        flag=true;
        //printing grid
        Rectangle r(220 , 220, 360, 360); r.setColor(GREEN);

        Line lv1(80,40 , 80,400);
    }
}

```

Line lv2(120,40 , 120,400);
Line lv3(160,40 , 160,400); lv3.setColor(GREEN);
Line lv4(200,40 , 200,400);
Line lv5(240,40 , 240,400);
Line lv6(280,40 , 280,400); lv6.setColor(GREEN);
Line lv7(320,40 , 320,400);
Line lv8(360,40 , 360,400);

Line lh1(40,80 , 400,80);
Line lh2(40,120 , 400,120);
Line lh3(40,160 , 400,160); lh3.setColor(GREEN);
Line lh4(40,200 , 400,200);
Line lh5(40,240 , 400,240);
Line lh6(40,280 , 400,280); lh6.setColor(GREEN);
Line lh7(40,320 , 400,320);
Line lh8(40,360 , 400,360);

Rectangle b1(60,440,40,40); Text t1(60,440,"1");
Rectangle b2(100,440,40,40); Text t2(100,440,"2");
Rectangle b3(140,440,40,40); Text t3(140,440,"3");
Rectangle b4(180,440,40,40); Text t4(180,440,"4");
Rectangle b5(220,440,40,40); Text t5(220,440,"5");
Rectangle b6(260,440,40,40); Text t6(260,440,"6");
Rectangle b7(300,440,40,40); Text t7(300,440,"7");
Rectangle b8(340,440,40,40); Text t8(340,440,"8");
Rectangle b9(380,440,40,40); Text t9(380,440,"9");
Rectangle b10(440,440,60,40); Text t10(440,440,"DELETE");
Rectangle restart(460,50,60,40); Text t11(460,50, "RESET");
Rectangle submit(460,100,60,40); Text t12(460,100,"SUBMIT");

```

for(i=0;i<9;i++)
{
    for(j=0;j<9;j++)
    {
        grid[i][j]=0;
    }
}

//repeating for 100 mouseclicks
repeat(100)
{

    Text *t;
    int clickPos = getClick();
    cx = clickPos/65536;
    cy = clickPos % 65536;
    //reseting the grid
    if(cx<=490 && cx>=430 && cy<=60 && cy>=20)
    {
        for(i=80; i<440;i+=40)
        {
            for(j=80;j<440;j+=40)
            {
                t=new Text(i-20,j-20," ");
            }
        }
    }

    for(i=80; i<440;i+=40)
    {

```

```

for(j=80; j<440;j+=40)
{
    if((i-20)-40/2<= cx && cx<= (i-20)+40/2 && (j-20)-40/2 <= cy && cy <= (j-20)+40/2)
    {
        Rectangle r4(i-20,j-20 , 30,30);
        int clickPos2=getClick();
        int cx1= clickPos2/65536;
        int cy1=clickPos2 % 65536;
        //deleting the element from grid
        if(cx1>=410&&cx1<=470&&cy1>=420&&cy1<=460)
        {
            t=new Text(i-20,j-20," ");
            continue;
        }
        //inputting in the grid
        for(int k=80; k<440; k+=40)
        {
            if((k-20)-40/2<= cx1 && cx1<= (k-20)+40/2 && cy1>=(440)-40/2 && cy1 <=
(440)+40/2)
            {
                column=((i-40)/40)-1;
                row=((j-40)/40)-1;
                switch((k/40)-1)
                {

                    case 1: t=new Text(i-20,j-20,"1"); grid[row][column]=1;break;
                    case 2: t=new Text(i-20,j-20,"2"); grid[row][column]=2;break;
                    case 3: t=new Text(i-20,j-20,"3"); grid[row][column]=3;break;
                    case 4: t=new Text(i-20,j-20,"4"); grid[row][column]=4;break;
                }
            }
        }
    }
}

```



```

        case 5: t=new Text(i-20,j-20,"5"); grid[row][column]=5;break;
        case 6: t=new Text(i-20,j-20,"6"); grid[row][column]=6;break;
        case 7: t=new Text(i-20,j-20,"7"); grid[row][column]=7;break;
        case 8: t=new Text(i-20,j-20,"8"); grid[row][column]=8;break;
        case 9: t=new Text(i-20,j-20,"9"); grid[row][column]=9;break;
    }

}

}

}

}

}

if(cx<=490 && cx>=430 && cy<=110 && cy>=90)
{

    if (SolveSudoku(grid) != false)
    {
        printGrid(grid);
        break;
    }

    system("cls");

    cout<<"*****\n";

    cout<<"*SOLUTION DOES NOT EXISTS      *\n";

    cout<<"*****\n";

```

```

        q=0;

        flag=false;

    }

    if(flag==false)
    {
        flag=true;
        break;
    }

}

}

//deleting the grid
Text *t;

    for(i=80; i<440;i+=40)
    {
        for(j=80;j<440;j+=40)
        {
            t=new Text(i-20,j-20,"  ");
        }
    }

Rectangle continu(220,220,240,40);

Text contin(220,220,"DO YOU WANT TO CONTINUE");

    int clickpos5=getClick();

    cx5=clickpos5/65536;

    cy5=clickpos5%65536;

```

```
}while(cx5>=220-160/2 && cx5<=220+160/2 && cy5>=220-40/2 && cy5<=220+40/2);//EXIT  
    return 0;  
}
```