#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h> //\_getch\*/

#include <termios.h> //\_getch\*/

void input\_manual();

void input\_file(char \*);

void generator();

int choice1();

int choice2();

void print();

void rows();

void columns();

void block();

void inter();

int sudoku();

int checker();

void solve();

int valid();

//clear screen code for linux

void clrscr(void)

{

system("clear");

}

//holds the screen until any key is pressed

char getch(){

char buf=0;

struct termios old={0};

fflush(stdout);

if(tcgetattr(0, &old)<0)

perror("tcsetattr()");

old.c\_lflag&=~ICANON;

old.c\_lflag&=~ECHO;

old.c\_cc[VMIN]=1;

old.c\_cc[VTIME]=0;

if(tcsetattr(0, TCSANOW, &old)<0)

perror("tcsetattr ICANON");

if(read(0,&buf,1)<0)

perror("read()");

old.c\_lflag|=ICANON;

old.c\_lflag|=ECHO;

if(tcsetattr(0, TCSADRAIN, &old)<0)

perror ("tcsetattr ~ICANON");

printf("%c\n",buf);

return buf;

}

//structure storing the elements missing in each row,column and block

//structure array variable of 10 elements each containing another array of 10 within it

struct abc{

int a[10];

}row[10],col[10],blo[10];

//3-D array storing the indexes of each element of the block according to the block number

int bloc[9][9][2]={{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3}},

{{1,4},{1,5},{1,6},{2,4},{2,5},{2,6},{3,4},{3,5},{3,6}},

{{1,7},{1,8},{1,9},{2,7},{2,8},{2,9},{3,7},{3,8},{3,9}},

{{4,1},{4,2},{4,3},{5,1},{5,2},{5,3},{6,1},{6,2},{6,3}},

{{4,4},{4,5},{4,6},{5,4},{5,5},{5,6},{6,4},{6,5},{6,6}},

{{4,7},{4,8},{4,9},{5,7},{5,8},{5,9},{6,7},{6,8},{6,9}},

{{7,1},{7,2},{7,3},{8,1},{8,2},{8,3},{9,1},{9,2},{9,3}},

{{7,4},{7,5},{7,6},{8,4},{8,5},{8,6},{9,4},{9,5},{9,6}},

{{7,7},{7,8},{7,9},{8,7},{8,8},{8,9},{9,7},{9,8},{9,9}}};

//structure of each node of a sudoku containing its face value, possible elements and block number

struct node{

int face;

int a[10] ;

int blo;

}sudo[10][10];

int empty\_ele=81; //taking number of empty elements initially as 81

int main(){

int ch1;

do{

clrscr();

int i,j;

empty\_ele=81;

for(i=1;i<10;i++)

for(j=0;j<10;j++)

row[i].a[j]=col[i].a[j]=blo[i].a[j]=1;

//menu driven program to take input from the user in three ways

ch1=choice1();

switch(ch1){

//one is through direct manual input from the user on the terminal

case 1:

input\_manual();

clrscr();

if(valid());

else{

printf("\n\n\n\t\tinvalid sudoku\n\n");

break;

}

solve();

break;

//second is through the input taken from a file containing sudoku

case 2:

clrscr();

printf("enter file name: ");

char a[20];

scanf("%s",a);

input\_file(a);

clrscr();

if(valid());

else{

printf("\n\n\tinvalid sudoku\n\n");

break;

}

solve();

break;

//third is the sudoku generator which generates random sudoku

case 3: generator();

clrscr();

printf("\n\nsudoku generated is :\n\n");

print();

printf("do you want to solve this sudoku(y/n)\nenter choice : ");

char x;

scanf("%c",&x);

clrscr();

if (x=='y');

solve();

break;

case 4:

break;

default : printf("wrong choice");

break;

}

}while(ch1!=4);

return 0;

}

//function which executes the major code for solving the sudoku according to the input method preferred by the user

//displays the message when the sudoku is solved

//takes in the users's choice whether he wants to check the solved sudoku

//displays message regarding the sudoku being right or wrong

void solve(){

rows();

columns();

block();

inter();

printf("\n\nsudoku we are going to solve is : \n\n");

print();

getch();

int hold=sudoku();

if (hold==1){

print();

printf("\n\nsudoku is solved !! \ndo you want to check if the sudoku is correct(y/n) \nenter choice : ");

char ch;

scanf("%c",&ch);

scanf("%c", &ch);

if (ch=='y'){

if (checker()){

printf("sum of all rows, columns and blocks is 45\nthus sudoku is solved correctly\n");

getch();

}

else{

printf("sudoku solved is wrong");

getch();

}

}

else if(ch=='n');

else {

printf("wrong choice entered");

}

}

}

//function checking the validity of the given sudoku

int valid(){

int flag,i,j;

for(i=1;i<10;i++){

for(j=1;j<10;j++){

if(sudo[i][j].face>=0 && sudo[i][j].face<=9)

flag=0;

else{

flag=1;

return 0;

}

}

}

return 1;

}

// function to choose what type of input method the user prefers

int choice1(){

printf("enter the method of input:\n\n1)manual input\n\n2)text file input\n\n3)sudoku generator\n\n4)exit\n\nenter choice : ");

int ch;

scanf("%d",&ch);

clrscr();

return ch;

}

// function to choose the difficulty level of the sudoku generator by the user

int choice2(){

printf("enter the difficulty level:\n1)easy\n2)moderate\n3)difficult\n4)evil\n\tenter choice : ");

int ch;

scanf("%d",&ch);

clrscr();

if (ch==1)

return 21;

else if(ch==2)

return 30;

else if(ch==3)

return 38;

else if(ch==4)

return 45;

}

// function which takes input from the user manually directly on the teminal

void input\_manual()

{

printf("enter the given valued of sudoku in their specific indexes\n");

int i,j;

for(i=1;i<10;i++)

{

printf("\n");

for(j=1;j<10;j++){

printf("row :%d ,column :%d =",i,j);

scanf("%d",&sudo[i][j].face);

if(sudo[i][j].face==0)

empty\_ele--; //keeps on decreasing the value of empty\_ele by one when there is a filled value btw 1-9 in

//sudoku

}

}

}

//function which takes in input from user via the file saved

void input\_file(char \*a){

FILE \*fp;

char c;

fp=fopen(a,"r");

int i=1,j=1;

while( !feof(fp)){

c = fgetc(fp);

if(c>='0'&&c<='9'){

sudo[i][j].face=c-48;

if(j==9){

i++;

j=1;

}

else

j++;

}

}

fclose(fp);

for (i = 1; i < 10; ++i){

for (j = 1; j < 10; ++j){

if (sudo[i][j].face!=0)

empty\_ele--;

}

}

}

//printing the sudoku in a appropriate manner

void print(){

int i,j;

for(i=1;i<=3;i++){

for(j=1;j<=3;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=4;j<=6;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=7;j<=9;j++)

printf("%d ",sudo[i][j].face);

printf("\n");

}

printf("\n");

for(i=4;i<=6;i++){

for(j=1;j<=3;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=4;j<=6;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=7;j<=9;j++)

printf("%d ",sudo[i][j].face);

printf("\n");

}

printf("\n");

for(i=7;i<=9;i++){

for(j=1;j<=3;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=4;j<=6;j++)

printf("%d ",sudo[i][j].face);

printf(" ");

for(j=7;j<=9;j++)

printf("%d ",sudo[i][j].face);

printf("\n");

}

}

//finding the missing elements in each row

void rows()

{

int i,j;

for(i=1;i<10;i++)

for(j=1;j<10;j++)

if(sudo[i][j].face!=0)

row[i].a[sudo[i][j].face]=0;

}

//finding the missing elements in each column

void columns()

{

int i,j;

for(i=1;i<10;i++)

for(j=1;j<10;j++)

if(sudo[j][i].face!=0)

col[i].a[sudo[j][i].face]=0;

}

//finding the missing elements in each block and assigning the block number to each node

void block()

{

int i,j;

for(i=1;i<=3;i++){

for(j=1;j<=3;j++){

sudo[i][j].blo=1;

if(sudo[i][j].face!=0)

blo[1].a[sudo[i][j].face]=0;

}

for(j=4;j<=6;j++){

sudo[i][j].blo=2;

if(sudo[i][j].face!=0)

blo[2].a[sudo[i][j].face]=0;

}

for(j=7;j<=9;j++){

sudo[i][j].blo=3;

if(sudo[i][j].face!=0)

blo[3].a[sudo[i][j].face]=0;

}

}

for(i=4;i<=6;i++){

for(j=1;j<=3;j++){

sudo[i][j].blo=4;

if(sudo[i][j].face!=0)

blo[4].a[sudo[i][j].face]=0;

}

for(j=4;j<=6;j++){

sudo[i][j].blo=5;

if(sudo[i][j].face!=0)

blo[5].a[sudo[i][j].face]=0;

}

for(j=7;j<=9;j++){

sudo[i][j].blo=6;

if(sudo[i][j].face!=0)

blo[6].a[sudo[i][j].face]=0;

}

}

for(i=7;i<=9;i++){

for(j=1;j<=3;j++){

sudo[i][j].blo=7;

if(sudo[i][j].face!=0)

blo[7].a[sudo[i][j].face]=0;

}

for(j=4;j<=6;j++){

sudo[i][j].blo=8;

if(sudo[i][j].face!=0)

blo[8].a[sudo[i][j].face]=0;

}

for(j=7;j<=9;j++){

sudo[i][j].blo=9;

if(sudo[i][j].face!=0)

blo[9].a[sudo[i][j].face]=0;

}

}

}

//finding the intersection of row,col and blo for each node -> node..a..

void inter()

{

int i,j,k,p;

for(i=1;i<10;i++)

for(j=1;j<10;j++){

int count=0;

if(sudo[i][j].face==0)

for(k=1;k<10;k++)

if(row[i].a[k]==1 && col[j].a[k]==1 && blo[sudo[i][j].blo].a[k]==1){

count++;

sudo[i][j].a[k]=1;

}

else

sudo[i][j].a[k]=0;

sudo[i][j].a[0]=count; //a[0] stores the number of intersection or the possible values

}

}

// main function for solving the sudoku

int sudoku(){

int i,j,k,p,moves=1,ele,t,count,var,t2,x,y,flag,x1,x2,y1,y2;

printf("\nnumber of empty elements :%d\n",empty\_ele);

printf("\n\tstarting\n\n");

getch();

clrscr();

while(empty\_ele!=0){ //loop continues until all the elements of a sudoku are filled or until any absurd condition does not arises

flag=0; // flag to detect whether a particular part of the code is executed or not

//first logic

//we check the entire intersection or the possible values of each node and find the node where only one possible value exists

for(i=1;i<10;i++){

for (j=1;j<10;j++){

if(sudo[i][j].a[0]==1){ // condition checking the if there is only one possible value of a particular node

flag=1;

// printf("%d",flag);

sudo[i][j].a[0]=0; //since only one value exists, we make its count as 0

empty\_ele--;

for(k=1;k<10;k++){

if(sudo[i][j].a[k]==1){

sudo[i][j].face=k; //assign the single possible value to the face of the sudoku node

printf("\nmove : %d\n\n",moves++);

print();

printf("\n\nelement changed in \n\trow : %d\n\tcol : %d\nnumber of elements left : %d",i,j,empty\_ele);

getch();

clrscr();

sudo[i][j].a[k]=0; //making the element, thus the node inactive

row[i].a[k]=0; // removing the assigned value from the missing elements of that particylar row

col[j].a[k]=0; //removing the assigned value from the missing elements of that particylar column

blo[sudo[i][j].blo].a[k]=0; //removing the assigned value from the missing elements of that particylar block

// making the assigned value inactive for the block containing the node

for(p=0;p<9;p++){

if(sudo[bloc[sudo[i][j].blo-1][p][0]][bloc[sudo[i][j].blo-1][p][1]].a[k]==1 && sudo[bloc[sudo[i][j].blo-1][p][0]][bloc[sudo[i][j].blo-1][p][1]].a[0]>0)

{

sudo[bloc[sudo[i][j].blo-1][p][0]][bloc[sudo[i][j].blo-1][p][1]].a[0]--;

sudo[bloc[sudo[i][j].blo-1][p][0]][bloc[sudo[i][j].blo-1][p][1]].a[k]=0;

}

}

// making the assigned value inactive for the row and column containing the node

for(p =1;p<10;p++){

if(sudo[i][p].a[k]==1 && sudo[i][p].a[0]>0){

sudo[i][p].a[0]--;

sudo[i][p].a[k]=0;

}

if(sudo[p][j].a[k]==1 && sudo[p][j].a[0]>0){

sudo[p][j].a[0]--;

sudo[p][j].a[k]=0;

}

}

break;

}

}

}

}

}

//second logic if the first logic fails, i.e no single possibility exists

//checking the each row, column and block, if there is a number possible at only one node in a entire row,column or block

if (flag==0){

for(i=1;i<10;i++){

for(ele=1;ele<10;ele++){

count=0;

if(row[i].a[ele]==1){// checking whether an element ele possible in a node of row i

for(t=1;t<10;t++){

if(sudo[i][t].a[ele]==1){ // counting the number of times an element ele is possible in the 9 nodes of a row

count++;

var=t;

}

}

// we proceed if the count is 1

if(count==1){

printf("\nmove : %d\n\n",moves++);

// since the element ele is possible only at one index in a row we assign its value to the node

sudo[i][var].face=ele;

print();

printf("\n\nelement changed in \n\trow : %d\n\tcol : %d\nnumber of elements left : %d",i,var,empty\_ele);

getch();

clrscr();

flag=2;

sudo[i][var].a[0]=0;

for(k=1;k<10;k++)

sudo[i][var].a[k]=0;

//removing the assigned element from the row, column and block associated with that node

row[i].a[ele]=0;

col[var].a[ele]=0;

blo[sudo[i][var].blo].a[ele]=0;

empty\_ele--;

//removing the element ele from where ever it is possible in that column

for(t=1;t<10;t++){

if(sudo[t][var].a[ele]==1 && sudo[t][var].a[0]>0){

sudo[t][var].a[0]--;

sudo[t][var].a[ele]=0;

}

//removing the element ele from where ever it is possible in that block

for(t2=0;t2<9;t2++){

if(sudo[bloc[sudo[i][var].blo-1][t2][0]][bloc[sudo[i][var].blo-1][t2][1]].a[ele]==1 && sudo[bloc[sudo[i][var].blo-1][t2][0]][bloc[sudo[i][var].blo-1][t2][1]].a[0]>0)

{

sudo[bloc[sudo[i][var].blo-1][t2][0]][bloc[sudo[i][var].blo-1][t2][1]].a[0]--;

sudo[bloc[sudo[i][var].blo-1][t2][0]][bloc[sudo[i][var].blo-1][t2][1]].a[ele]=0;

}

}

}

}

if(col[i].a[ele]==1){// checking whether an element ele possible in a node of column i

count=0;

for(t=1;t<10;t++){

if(sudo[t][i].a[ele]==1){// counting the number of times an element ele is possible in the 9 nodes of a column

count++;

var=t;

}

}

// we proceed if the count is 1

if(count==1){

printf("\nmove : %d\n\n",moves++);

sudo[var][i].face=ele;

print();

printf("\n\nelement changed in \n\trow : %d\n\tcol : %d\nnumber of elements left : %d",var,i,empty\_ele);

getch();

clrscr();

flag=2;

sudo[var][i].a[0]=0;

for (k=1;k<10;k++)

sudo[var][i].a[k]=0;

// since the element ele is possible only at one index in a row we assign its value to the node

row[var].a[ele]=0;

col[i].a[ele]=0;

blo[sudo[var][i].blo].a[ele]=0;

empty\_ele--;

//removing the element ele from where ever it is possible in the associated row

for(t=1;t<10;t++){

if(sudo[var][t].a[ele]==1 && sudo[var][t].a[0]>0){

sudo[var][t].a[0]--;

sudo[var][t].a[ele]=0;

}

}

//removing the element ele from where ever it is possible in the associated block

for(t2=0;t2<9;t2++){

if(sudo[bloc[sudo[var][i].blo-1][t2][0]][bloc[sudo[var][i].blo-1][t2][1]].a[ele]==1 && sudo[bloc[sudo[var][i].blo-1][t2][0]][bloc[sudo[var][i].blo-1][t2][1]].a[0]>0)

{

sudo[bloc[sudo[var][i].blo-1][t2][0]][bloc[sudo[var][i].blo-1][t2][1]].a[0]--;

sudo[bloc[sudo[var][i].blo-1][t2][0]][bloc[sudo[var][i].blo-1][t2][1]].a[ele]=0;

}

}

}

}

}

}

// checking the each block, if there is a number possible at only one node in a entire block

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

for(ele=1;ele<10;ele++){

count=0;

if(blo[i+1].a[ele]==1){

for(t=0;t<9;t++){

if(sudo[bloc[i][t][0]][bloc[i][t][1]].a[ele]==1){

count++;

x=bloc[i][t][0];

y=bloc[i][t][1];

}

}

if(count==1){

printf("\nmove : %d\n\n",moves++);

sudo[x][y].face=ele;

print();

printf("\n\nelement changed in \n\trow : %d\n\tcol : %d\nnumber of elements left : %d",x,y,empty\_ele);

getch();

clrscr();

flag=2;

sudo[x][y].a[0]=0;

for(k=1;k<10;k++)

sudo[x][y].a[k]=0;

blo[i+1].a[ele]=0;

row[x].a[ele]=0;

col[y].a[ele]=0;

empty\_ele--;

//removing the element ele from each element in a row and column

for(t=1;t<10;t++){

if(sudo[x][t].a[ele]==1 && sudo[x][t].a[0]>0){

sudo[x][t].a[0]--;

sudo[x][t].a[ele]=0;

}

if(sudo[t][y].a[ele]==1 && sudo[t][y].a[0]>0){

sudo[t][y].a[0]--;

sudo[t][y].a[ele]=0;

}

}

}

}

}

}

}

if (flag ==1 || flag==2){

for(i=1;i<10;i++){

for(k=1;k<10;k++){

if(blo[i].a[k]==1){

count=0,x1=0,y1=0,x2,y2;

for(t=0;t<9;t++){

if(sudo[bloc[i-1][t][0]][bloc[i-1][t][1]].a[k]==1){

count++;

if(x1==0 && y1==0){

x1=bloc[i-1][t][0];

y1=bloc[i-1][t][1];

}

else{

x2=bloc[i-1][t][0];

y2=bloc[i-1][t][1];

}

}

}

if(count==2){

if(y1==(y2+1) || y2==(y1+1) && x1==x2){

for(j=1;j<10;j++){

if(j!=y1 && j!=y2){

if(sudo[x1][j].a[k]==1){

sudo[x1][j].a[k]=0;

sudo[x1][j].a[0]--;

}

}

}

}

else if(x1==(x2+1) || x2==(x1+1) && y1==y2){

for(j=1;j<10;j++){

if(j!=x1 && j!=x2){

if(sudo[j][y1].a[k]==1){

sudo[j][y1].a[k]=0;

sudo[j][y1].a[0]--;

}

}

}

}

}

}

}

}

}

if( empty\_ele==0){ // sudoku solved

return 1;

}

else if( flag!=1 && flag !=2){ // unsolvable sudoku

print();

printf("\n\nsudoku is solvable only till this move further hit an trial might work which is out of the scope of this program or any absurd condition has occured due to which this sudoku is unsolvable\n\n");

printf("\nnumber of elements left to be filled are : %d\n\n",empty\_ele);

getch();

return 0;

}

}

}

// function which checks whether the solved sudoku is correct or not

//when the sum of all elements in each row, column and block is 45 the sudoku filled is correct

int checker(){

int i,j,s1,s2,s3,flag;

int r[10],c[10],b[10];

for(i=1;i<10;i++){

s1=0,s2=0,s3=0;

for(j=1;j<10;j++){

s1=s1+sudo[i][j].face;

s2=s2+sudo[j][i].face;

}

r[i]=s1;

c[i]=s2;

}

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

s3=0;

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

s3=s3+sudo[bloc[i][j][0]][bloc[i][j][1]].face;

b[i]=s3;

}

flag=0;

for(i=1;i<10;i++){

if(r[i]==45 && b[i-1]==45 && c[i]==45)

flag=1;

else{

flag=0;

break;

}

}

if(flag==1)

return 1;

else

return 0;

}

//function which randomly generates the sudoku

void generator(){

int c=choice2();

int x,y,i,k=0,flag;

int pos[c][2];

char \*a[]={"a.txt","b.txt","c.txt","d.txt","e.txt","f.txt","g.txt","h.txt","i.txt","j.txt"};//database storing 10 solved sudoku

srand((unsigned) time(NULL));

int r=rand() % 10; // randomly selcting any one of the 10 data bases

printf("\n%s",a[r]);

input\_file(a[r]);

//randomly removing the elements of the selected sudoku database according to the difficulty level chosen by the user

while(c>0){

flag=0;

x=rand()%9;

y=rand()%9;

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

if (x==pos[i][0] && y==pos[i][1])

flag=1;

if (flag==0){

pos[k][0]=x;

pos[k][1]=y;

k++;

sudo[x+1][y+1].face=0;

c--;

}

}

empty\_ele=k;

}















