Pastebin Link: <https://paste.ubuntu.com/p/5XgZS7R76R/>

*/\**  
 *Input: Char\*\* str (A 2D character array containing the initial*  
 *configuration of the board)*  
  
 *Define EMPTY\_CELL as the character that defines an empty cell in str*  
  
 *Call sudoku(char\*\* str) to initialize the sudoku structure*  
  
 *Call solve() to find a solution to the puzzle (May return any solution*  
 *if not unique)*  
  
 *solve() returns true if a solution exits, false otherwise.*  
 *If true is returned, var[][] will contain the values of each cell*  
  
 *Call print() to print the solved puzzle (Adjust according to problem statement)*  
*\*/*  
  
#define EMPTY\_CELL '.'  
**typedef** pair <**int**, **int**> pii;  
#define XX first  
#define YY second  
**struct** sudoku{  
 **int** var[15][15];  
 **int** dom[15][15], pdom[15][15];  
 pii pos[15][15];  
 **int** pcnt[15];  
  
 **int** group[10][10]={  
 {0, 0, 0, 1, 1, 1, 2, 2, 2},  
 {0, 0, 0, 1, 1, 1, 2, 2, 2},  
 {0, 0, 0, 1, 1, 1, 2, 2, 2},  
 {3, 3, 3, 4, 4, 4, 5, 5, 5},  
 {3, 3, 3, 4, 4, 4, 5, 5, 5},  
 {3, 3, 3, 4, 4, 4, 5, 5, 5},  
 {6, 6, 6, 7, 7, 7, 8, 8, 8},  
 {6, 6, 6, 7, 7, 7, 8, 8, 8},  
 {6, 6, 6, 7, 7, 7, 8, 8, 8}  
 }; *//Sub-sections that cannot have the same number are marked with same grouo[][] value*  
  
 **inline** **int** set\_bit(**int** mask, **int** pos){ **return** (mask | (1<<pos)); }  
 **inline** **int** reset\_bit(**int** mask, **int** pos){ **return** (mask & ~(1<<pos)); }  
 **inline** **bool** check\_bit(**int** mask, **int** pos){ **return** (mask & (1<<pos))!=0; }  
  
  
 **inline** **void** save(**int** pdom[15][15]){  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 pdom[i][j]=dom[i][j];  
 }  
 }  
 }  
  
 **inline** **void** color\_var(**int** px, **int** py, **int** color){  
 var[px][py]=color;  
 pii tmp;  
 **for**(**int** i=0; i<9; i++) dom[px][i]=reset\_bit(dom[px][i], color);  
 **for**(**int** i=0; i<9; i++) dom[i][py]=reset\_bit(dom[i][py], color);  
 **for**(**int** i=0; i<9; i++){  
 tmp=pos[group[px][py]][i];  
 dom[tmp.XX][tmp.YY]=reset\_bit(dom[tmp.XX][tmp.YY], color);  
 }  
 dom[px][py]=0;  
 dom[px][py]=set\_bit(dom[px][py], color);  
 }  
  
 **inline** **void** restore(**int** pdom[15][15]){  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 dom[i][j]=pdom[i][j];  
 }  
 }  
 }  
  
 **inline** pii findNextPair(){ */\*(-1, -1) if invalid,*  
 *(-2,-2) if solved,*  
 *(x, y) if (x, y) is most constrained \*/*  
 **bool** sflag=**true**, fg=**false**;  
 pii mn;  
 **int** cnt;  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 cnt=\_\_builtin\_popcount(dom[i][j]);  
 **if**(cnt==0){  
 **return** pii(-1,-1);  
 }  
 **if**(cnt!=1) {  
 sflag=**false**;  
 **if**(!fg || cnt<(\_\_builtin\_popcount(dom[mn.XX][mn.YY]))){  
 fg=**true**;  
 mn=pii(i, j);  
 }  
 }  
 **else**{  
 **for**(**int** k=1; k<=9; k++){  
 **if**(check\_bit(dom[i][j], k)){  
 color\_var(i, j, k);  
 **break**;  
 }  
 }  
 }  
 }  
 }  
 **if**(sflag) **return** pii(-2, -2);  
 **return** mn;  
 }  
  
 sudoku(){}  
 sudoku(**char** str[15][15]){  
 memset(pcnt, 0, **sizeof**(pcnt));  
 **int** val=0;  
 **for**(**int** i=1; i<=9; i++){  
 val=set\_bit(val, i);  
 }  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 pos[group[i][j]][pcnt[group[i][j]]]=pii(i, j);  
 pcnt[group[i][j]]++;  
 dom[i][j]=val;  
 }  
 }  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 **if**(str[i][j]!=EMPTY\_CELL){  
 color\_var(i, j, str[i][j]-'0');  
 }  
 }  
 }  
 }  
  
 **bool** backtrack(**int** cx, **int** cy){  
 **int** pdom[15][15];  
 save(pdom);  
 pii tmp;  
 **for**(**int** i=1; i<=9; i++){  
 **if**(check\_bit(dom[cx][cy], i)){  
 color\_var(cx, cy, i);  
 tmp=findNextPair();  
 **if**(tmp==pii(-2, -2)) **return** **true**;  
 **else** **if**(tmp==pii(-1, -1)) restore(pdom);  
 **else**{  
 **if**(backtrack(tmp.XX, tmp.YY)) **return** **true**;  
 restore(pdom);  
 }  
 }  
 }  
 **return** **false**;  
 }  
  
 **bool** solve(){ *//Returns true if solution exists, false otherwise*  
 pii nxt=findNextPair();  
 **if**(nxt==pii(-2, -2)) **return** **true**;  
 **if**(nxt==pii(-1, -1)) **return** **false**;  
 **return** backtrack(nxt.XX, nxt.YY);  
 }  
  
 **void** print(){  
 **for**(**int** i=0; i<9; i++){  
 **for**(**int** j=0; j<9; j++){  
 printf("%c", (**char**)(var[i][j]+'0'));  
 }  
 printf("**\n**");  
 }  
 }  
}sudo;