sodoku\_main.c

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

/\*

Author= Ziv Belahsan 201567278 Yogev laks=200344729

Project=Exercise 1

Using -

checker.c

create\_output\_file.c

solver.c

text\_to\_mat.c

all of these file are rediricted via "sodoku.h" header.

Description- The main function of project sodoku.

the main function recieves the command line arguments and initiats the required operation

\*/

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

#include "sodoku.h"

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : main

Input arguments: int argc - number of arguments sent from command line

char \* argv[] - array of strings recieved from command line

return: int OK - program run without errors

ERROR - program failed

Description- The main function of project sodoku.

the main function recieves the command line arguments and initiats the required operation

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int main(int argc,char \*argv[])

{

FILE \*output\_file = NULL;

int mat[MAT\_SIZE][MAT\_SIZE];// create 2D matrix for storing the numbers from the imput file

int solver\_or\_checker\_flag =-1;//flag to check or solve operation

if ((argc < INSUFFICIENT\_ARGUMENTS)||(argc > MAX\_ARGUMENTS\_NUMBER)) //check number of arguments from main

{

printf("ARGUMENTS NUMBER INCORRECT\n");

return(ERROR);

}

solver\_or\_checker\_flag = atoi(argv[1]);// get flag for solver/checker from command line

if(text\_to\_mat(argv[2],mat) == ERROR)//convert the input file to a matrix

return ERROR;

if (argc == MAX\_ARGUMENTS\_NUMBER)// we have got an output file name from main

output\_file = create\_output\_file(argv[3]); //create\_output\_file with argv[3] name

else

{

create\_output\_file\_name(argv[2]);// reformat input file name

output\_file = create\_output\_file(argv[2]);// creates a file for writing with the new file name

}

if(solver\_or\_checker\_flag)//0 - solver, 1 - checker

main\_checker(mat,output\_file);

else

main\_solver(mat,output\_file);

fclose(output\_file);

}

Sodoku.h

#ifndef SODOKU\_H

#define SODOKU\_H

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define OK 0

#define ERROR 1

#define ARR\_SIZE 10

#define MAT\_SIZE 9

#define SUB\_MAT\_SIZE 3

#define CHAR\_TO\_DIGIT 48

#define INSUFFICIENT\_ARGUMENTS 2

#define MAX\_ARGUMENTS\_NUMBER 4

#define ONLY\_ONE\_OPTION 1

typedef int opt\_array[ARR\_SIZE];

int text\_to\_mat(char\* input\_file\_name,int arr[MAT\_SIZE][MAT\_SIZE]);

int main\_solver(int mat[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file);

void main\_checker(int arr[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file);

FILE\* create\_output\_file(char\* output\_file\_name);

void create\_output\_file\_name(char\* output\_file\_name);

#endif

text\_to\_mat.c

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

/\*

Author= Ziv Belahsan 201567278 Yogev laks=200344729

Project=Exercise 1

Using - sodoku.h

Description - parsing a text file into 2D array and returning it to main fuction

\*/

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

#include "sodoku.h"

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : text\_to\_mat

Input arguments: char\* input\_file\_name - the name of the input file

int arr[MAT\_SIZE][MAT\_SIZE] - pointer to 2D array to fill accordingly

return: int OK - function run without errors

ERROR - function failed

Description- The function is parsing a text file into 2D array and fills the 2D array and returns

it to the main fuction

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int text\_to\_mat(char\* input\_file\_name,int arr[MAT\_SIZE][MAT\_SIZE])

{

FILE \*fp = NULL;

int digit=0 ,i=0, j=0;

fp = fopen (input\_file\_name,"r+");

if (fp == NULL)

{

printf("FILE NOT FOUND \n");

return ERROR;

}

while (! feof(fp)) //while we havnt reached end of file

{

if(isdigit(digit))

arr[i][j++] = digit - CHAR\_TO\_DIGIT; // convert to digit from ascii value

else if(digit == '.')

arr[i][j++] = 0;

digit = fgetc(fp); // get the next char from stream file.

}

fclose(fp);

return OK;

}

create\_output\_file.c

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

/\*

Author= Ziv Belahsan 201567278 Yogev laks=200344729

Project=Exercise 1

Using - sodoku.h

Description - handles the output file

\*/

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

#include "sodoku.h"

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : create\_output\_file

Input arguments: char\* output\_file\_name - the name of the output file

return: FILE\* fp - A pointer to an output file

the function kills the program if the output file cannot be opened

Description- The function is creating an output file according to output\_file\_name and returns

a pointer to it

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

FILE\* create\_output\_file(char\* output\_file\_name)

{

FILE \*fp;

fp = fopen (output\_file\_name,"w");

if (fp == NULL)

{

printf("FILE NOT FOUND \n");// if cannot open file - kill program

exit(1) ;

}

return fp;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : create\_output\_file\_name

Input arguments: char\* output\_file\_name - the name of the output file

return: None

Description- The function reformat the input file name to get the correct output file name

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void create\_output\_file\_name(char\* output\_file\_name)

{

output\_file\_name=strtok(output\_file\_name,".");

strcat(output\_file\_name,"\_sol.txt");

}

Checker.c

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

/\*

Author= Ziv Belahsan 201567278 Yogev laks=200344729

Project=Exercise 1

Using - sodoku.h

Description - checks a given solution for a sodoku

\*/

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

#include "sodoku.h"

int gErrorFound=0; //global variable that indicates first orrurance of errors

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : error\_found

Input arguments: FILE \*output\_file - The output file handler

return: None

Description- The function prints the error header only once according to the global Error flag

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void error\_found(FILE \*output\_file)

{

if(!gErrorFound)

{

gErrorFound =1;

fprintf(output\_file,"Found errors in given Sudoku puzzle.\nThe errors are:\n");

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : check\_rows

Input arguments: int arr[MAT\_SIZE][MAT\_SIZE] - The input matrix

FILE \*output\_file - The output file handler

return: None

Description- The function prints the errors that are found in the rows

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void check\_rows(int arr[MAT\_SIZE][MAT\_SIZE],FILE \*output\_file)

{

int i,j,k,tmp;

for (i=0; i<MAT\_SIZE; i++)

{

for(j=0; j<MAT\_SIZE; j++)

{

tmp = arr[i][j];

for(k=j+1;k<MAT\_SIZE;k++)

{

if(arr[i][k] == tmp)

{

error\_found(output\_file);

fprintf(output\_file,"Line error: digit %d appears at (%d,%d) and (%d,%d)\n",arr[i][k],j+1,i+1,k+1,i+1);

}

}

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : check\_colomn

Input arguments: int arr[MAT\_SIZE][MAT\_SIZE] - The input matrix

FILE \*output\_file - The output file handler

return: None

Description- The function prints the errors that are found in the colomns

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void check\_colomn(int arr[MAT\_SIZE][MAT\_SIZE],FILE \*output\_file)

{

int i,j,k,tmp;

for (i=0; i<MAT\_SIZE; i++)

{

for(j=0; j<MAT\_SIZE; j++)

{

tmp = arr[j][i];

for(k=j+1;k<MAT\_SIZE;k++)

{

if(arr[k][i] == tmp)

{

error\_found(output\_file);

fprintf(output\_file,"Column error: digit %d appears at (%d,%d) and (%d,%d)\n",arr[k][i],i+1,j+1,i+1,k+1);

}

}

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : check\_box

Input arguments: int arr[MAT\_SIZE][MAT\_SIZE] - The input matrix

FILE \*output\_file - The output file handler

return: None

Description- The function gets a subgrid and prints the errors that are found in it.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void check\_box(int arr[MAT\_SIZE][MAT\_SIZE],int row,int col,FILE \*output\_file)

{

int i,j,x,y;

int repeating\_numbers\_array[ARR\_SIZE]={0};

for(i=0;i<SUB\_MAT\_SIZE;i++)

{

for(j=0;j<SUB\_MAT\_SIZE;j++)

{

for(x=0;x<SUB\_MAT\_SIZE;x++)

{

for(y=0;y<SUB\_MAT\_SIZE;y++)

{

if(x!=i || y!=j)//make sure that we don't double check a value (i,j) and (j,i)

if(arr[i][j] == arr[x][y])

if(! repeating\_numbers\_array[arr[i][j]])//repeating\_numbers\_array managers the already found repeating numbers

{

error\_found(output\_file);

fprintf(output\_file,"Subgrid error: digit %d appears at (%d,%d) and (%d,%d)\n",arr[i][j],j+col,i+row,y+col,x+row);

repeating\_numbers\_array[arr[i][j]]++;

}

}

}

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : check\_Subgrid

Input arguments: int arr[MAT\_SIZE][MAT\_SIZE] - The input matrix

FILE \*output\_file - The output file handler

return: None

Description- The function calls the check\_box function with the needed sub grid in order to find

the errors in it.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void check\_Subgrid(int arr[MAT\_SIZE][MAT\_SIZE],FILE \*output\_file)

{

int row ,col,k=0;

for(row =0 ; row < MAT\_SIZE ; row +=3)

{

for(col = 0; col <MAT\_SIZE; col+=3)

{

check\_box((int (\*)[MAT\_SIZE])&arr[row][col],row+1,col+1,output\_file);

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : main\_checker

Input arguments: int arr[MAT\_SIZE][MAT\_SIZE] - The input matrix

FILE \*output\_file - The output file handler

return: None

Description- The function calls sub routines in order the go over the input matrix and check it.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void main\_checker(int arr[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file)

{

int row =0, col=0;

int backet[MAT\_SIZE]={0};

check\_rows(arr,output\_file);

check\_colomn(arr,output\_file);

check\_Subgrid(arr,output\_file);

if(!gErrorFound)

fprintf(output\_file,"No errors found in given Sudoku puzzle.");

}

Solver.c

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

/\*

Author= Ziv Belahsan 201567278 Yogev laks=200344729

Project=Exercise 1

Using - sodoku.h

Description - solves the sodoku if possible

\*/

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

#include "sodoku.h"

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : fill\_row\_optional\_numbers

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - 2D array containing the current sodoku matrix

arr[ARR\_SIZE] - an array of flags determing the available row values for the current cell

arr[ARR\_SIZE] is sent with all the values in it initialized to '1'

row - The current row number

return: None

Description- The function use the arr[ARR\_SIZE] to indicate which values are already filled in the current row

it goes over the specific row at the matrix and zeroes the value in the arr[ARR\_SIZE] index if it is

present at the 2D matrix row

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void fill\_row\_optional\_numbers(int mat[MAT\_SIZE][MAT\_SIZE],int arr[ARR\_SIZE],int row)

{

int i;

for(i=0; i<MAT\_SIZE; i++)

{

if(arr[mat[row][i]])

{

arr[mat[row][i]]--;

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : fill\_column\_optional\_numbers

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - 2D array containing the current sodoku matrix

arr[ARR\_SIZE] - an array of flags determing the available column values for the current cell

arr[ARR\_SIZE] is sent with all the values in it initialized to '1'

col - The current column number

return: None

Description- The function use the arr[ARR\_SIZE] to indicate which values are already filled in the current column

it goes over the specific column at the matrix and zeroes the value in the arr[ARR\_SIZE] index if it is

present at the 2D matrix column

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void fill\_column\_optional\_numbers(int mat[MAT\_SIZE][MAT\_SIZE],int arr[ARR\_SIZE],int col)

{

int i;

for(i=0; i<MAT\_SIZE; i++)

{

if(arr[mat[i][col]])

{

arr[mat[i][col]]--;

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : fill\_subgrid\_optional\_numbers

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - 2D array containing the current sodoku matrix

arr[ARR\_SIZE] - an array of flags determing the available sub grid values for the current cell

arr[ARR\_SIZE] is sent with all the values in it initialized to '1'

return: None

Description- The function use the arr[ARR\_SIZE] to indicate which values are already filled in the current sub grid

it goes over the specific sub grid at the matrix and zeroes the value in the arr[ARR\_SIZE] index if it is

present at the 2D matrix sub grid

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void fill\_subgrid\_optional\_numbers(int mat[MAT\_SIZE][MAT\_SIZE],int arr[ARR\_SIZE])

{

int i,j;

for(i=0;i<SUB\_MAT\_SIZE;i++)

{

for(j=0;j<SUB\_MAT\_SIZE;j++)

{

if(arr[mat[i][j]])

{

arr[mat[i][j]]--;

}

}

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : fill\_subgrid\_optional\_numbers

Input arguments: int arr\_row[ARR\_SIZE] - 1D array contaning the available row values for the current cell

arr\_col[ARR\_SIZE] - 1D array contaning the available column values for the current cell

arr\_subgrid[ARR\_SIZE] - 1D array contaning the available sub grid values for the current cell

return: int \*total\_optional\_numbers\_for\_cell - a dinamic allocated array indicating all the available value in the

sudoko matrix.

Description- The function allocates a dinamic array for each cell that was previosly empty, that array indicates

which values are optional for that cell according to the row, column and sub grid.

the first cell of the dinamic array - arr[0]- contains the number of available values.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int \*total\_optional\_numbers\_for\_cell(int arr\_row[ARR\_SIZE],int arr\_col[ARR\_SIZE],int arr\_subgrid[ARR\_SIZE])

{

int i,count =0,flag=1;

int \*final\_arr;

final\_arr = (int\*) malloc(ARR\_SIZE\*sizeof(int));

if(final\_arr ==NULL)

exit(ERROR);

for(i=1 ;i<ARR\_SIZE; i++)

{

final\_arr[i] = arr\_row[i] & arr\_col[i] & arr\_subgrid[i];

if(final\_arr[i])

count ++;

}

final\_arr [0] = count;

for(i=0;i<ARR\_SIZE;i++)

{

if(final\_arr[i])

flag =0;

}

if(flag)

exit(ERROR); //No optional values available for the current cell , no go situation

return final\_arr;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : return\_single\_option

Input arguments: int arr[ARR\_SIZE] - an array of flags determing the available values for the current cell

return: int i - the single value at the array or OK if the array is empty

Description- The function returned the single value available for the current cell

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int return\_single\_option(int arr[ARR\_SIZE])

{

int i;

for(i = 1; i<ARR\_SIZE; i++)

if(arr[i])

return i;

return OK;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : init\_arr

Input arguments: int arr[ARR\_SIZE] - 1D array

num - Initial value

return: None

Description- The function initialize all the values in the array to num

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void init\_arr(int arr[ARR\_SIZE],int num)

{

int i;

for (i=0;i<ARR\_SIZE;i++)

arr[i] =num;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : found\_in\_array

Input arguments: int k - The value to be searched

arr[ARR\_SIZE] - 1D array

return: int OK - if the value is found

ERROR - if the value is not found

Description- The function search for k in the array and returns OK if it is there

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int found\_in\_array(int k,int arr[ARR\_SIZE])

{

if(arr[k])

return OK;

return ERROR;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : find\_single\_in\_sub\_grid

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - The value to be searched

row - Current row

col - Current column

int \*opt\_mat[MAT\_SIZE][MAT\_SIZE]- The matrix of all the available option

return: int flag - If the cell had his value changed

Description- The function operates when there is more than one value available for each cell in a sub grid

it goes over the cells, and finds if there is a value that can be placed only in one cell.

If such a case happens it place the value in the cell.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int find\_single\_in\_sub\_grid(int mat[MAT\_SIZE][MAT\_SIZE],int \*opt\_mat[MAT\_SIZE][MAT\_SIZE],int row, int col)

{

int arr[ARR\_SIZE];

int i,j,k;

int flag =0;

init\_arr(arr,0);

for(i=0;i<SUB\_MAT\_SIZE;i++)

{

for(j=0;j<SUB\_MAT\_SIZE;j++)

{

if(mat[i][j] == 0)

{

for(k=1;k<ARR\_SIZE;k++)

{

if(opt\_mat[i+row][j+col][k])

arr[k]++;

}

}

}

}

for(k=1;k<ARR\_SIZE;k++)

{

if(arr[k] == 1)

{

for(i=0;i<SUB\_MAT\_SIZE;i++)

{

for(j=0;j<SUB\_MAT\_SIZE;j++)

{

if(mat[i][j] == 0)

{

if(!found\_in\_array(k,opt\_mat[i+row][j+col]))

{

mat[i][j] = k;

flag=1;

free(opt\_mat[i+row][j+col]);

}

}

}

}

}

}

return flag;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : fprint\_mat

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - The complete sodoku matrix

FILE \* output\_file - The output file handler

return: None

Description- The function prints the matrix to a output file according to exercise the instructions

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

void fprint\_mat(int mat[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file)

{

int i,j;

for(i=0;i<MAT\_SIZE; i++)

{

for(j=0;j<MAT\_SIZE; j++)

{

if ((j %SUB\_MAT\_SIZE == 0) && (j>1))

fprintf(output\_file," |");

fprintf(output\_file," %d",mat[i][j]);

}

if ((i %SUB\_MAT\_SIZE == 2) && (i>1) && (i<8))

fprintf(output\_file,"\n-------+-------+-------");

fprintf(output\_file,"\n");

}

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : check\_if\_done

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - The complete sodoku matrix

\*opt\_mat[MAT\_SIZE][MAT\_SIZE]- The option matrix

FILE \* output\_file - The output file handler

return: int OK if the matrix is solved ERROR else.

Description- The function checks if the mtrix is full, if it is not full it prints a message to the output file

furthermore, the function frees any leftover dinamic allocated arrays.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int check\_if\_done(int mat[MAT\_SIZE][MAT\_SIZE],int \*opt\_mat[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file)

{

int i,j,error\_flag = 0;

for(i=0;i<MAT\_SIZE; i++)

{

for(j=0;j<MAT\_SIZE; j++)

{

if(mat[i][j] == 0)

{

if(!error\_flag)

{

fprintf(output\_file,"Sudoku puzzele is too hard for me to solve");

error\_flag ++;

}

free(opt\_mat[i][j]);

}

}

}

if(error\_flag)

return ERROR;

return OK;

}

/\*oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO

function name : main\_solver

Input arguments: int mat[MAT\_SIZE][MAT\_SIZE] - The complete sodoku matrix

FILE \* output\_file - The output file handler

return: int OK if the matrix is solved ERROR else.

Description- The function calls the sub routines in order to solve the sudoko.

oOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoOoO\*/

int main\_solver(int mat[MAT\_SIZE][MAT\_SIZE],FILE \* output\_file)

{

int row,col;

int arr\_row[ARR\_SIZE],arr\_col[ARR\_SIZE],arr\_subgrid[ARR\_SIZE];

int flag =1;

int \*options\_matrix[MAT\_SIZE][MAT\_SIZE];

while(flag)//flag goes up if there is any change in the matrix in a single loop

{

flag = 0;

// START - fill options per cell

for(row = 0; row< MAT\_SIZE; row ++)

{

for(col = 0; col<MAT\_SIZE; col++)

{

init\_arr(arr\_row,1);

init\_arr(arr\_col,1);

init\_arr(arr\_subgrid,1);

if(0 == mat[row][col])//the cell needs to be filled

{

fill\_row\_optional\_numbers(mat,arr\_row,row);

fill\_column\_optional\_numbers(mat,arr\_col,col);

fill\_subgrid\_optional\_numbers((int (\*)[MAT\_SIZE])

&mat[(row/SUB\_MAT\_SIZE)\*SUB\_MAT\_SIZE][(col/SUB\_MAT\_SIZE)\*SUB\_MAT\_SIZE],arr\_subgrid);

//this function recives a sub matrix according to the cell cordinations

options\_matrix[row][col] = total\_optional\_numbers\_for\_cell(arr\_row,arr\_col,arr\_subgrid);

}

}

}

// END - fill options per cell

//START - fill cells with only one option

for(row=0; row< MAT\_SIZE; row ++)

{

for(col=0; col<MAT\_SIZE; col++)

{

if(mat[row][col] == 0)

{

if(\*options\_matrix[row][col] == ONLY\_ONE\_OPTION)// if there is only one option than set it as the value of the cell

{

mat[row][col] = return\_single\_option(options\_matrix[row][col]);

flag =1;

free(options\_matrix[row][col]);

}

}

}

}//if flag is zero at this point either we finnished or we have a problem of at least 2 options available for a cell

//END - fill cells with only one option

//START - try to fill cells with more than one option

if(!flag)

{

for(row = 0; row< MAT\_SIZE; row+=3)

{

for(col = 0; col<MAT\_SIZE; col+=3)

{

if(find\_single\_in\_sub\_grid((int (\*)[MAT\_SIZE])&mat[row][col],options\_matrix,row,col))

flag=1;

}

}

}

//END - try to fill cells with more than one option

}//End of while

if(check\_if\_done(mat,options\_matrix,output\_file) == ERROR)

return ERROR;

fprint\_mat(mat,output\_file);//prints matrix to the output file

return OK;

}