#include <stdio.h>

#include <stdlib.h> //random function

#include <linux/types.h>

#include <time.h>

#include <assert.h>

int sudoku[9][9];

int num[9];

int duplicate[9][9];

int level = 2;

typedef unsigned char bool;

#define FALSE 0

#define TRUE 1

#define PRINT\_AS\_9X9\_GRID 0

void GenerateData(void);

void fill\_first\_line(void);

void fill\_first\_column(void);

void fill\_last\_line(void);

bool JudgeData(int line, int column, int is\_subject, int (\*array)[9]);

bool is\_line\_ok(int line, int is\_subject, int (\*array)[9]);

bool is\_column\_ok(int column, int is\_subject, int (\*array)[9]);

bool is\_matrix\_ok(int line, int column, int is\_subject, int (\*array)[9]);

int get\_random\_number(int line, int column, int \*rand\_number);

int get\_random\_range(int in\_line, int in\_column);

void print\_answer();

void print\_menu();

void print\_option();

void select\_level();

void start\_game();

void generate\_subject(int hide\_number, int \*hide\_item);

void display\_subject();

int main()

{

int i, j;

//int sel;

char sel;

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

{

//initial the number array;

num[i] = i + 1;

//printf("%d ", num[i]);

//initial the array of sudoku

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

{

sudoku[i][j] = 0;

}

}

//printf("\n");

srand((uint)time(NULL));

//fill the first line of the sudoku

fill\_first\_line();

fill\_first\_column();

//fill the other part of the sudoku

GenerateData();

//fill the last line

fill\_last\_line();

/\*

#if PRINT\_AS\_9X9\_GRID

printf("-------------------------------------\n");

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

{

printf("|");

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

{

//printf("%d ", sudoku[i][j]);

printf(" %d ", sudoku[i][j]);

//if(0 == (j + 1) % 3 )

{

printf("|");

}

}

printf("\n");

//if(0 == (i + 1) % 3 )

{

printf("-------------------------------------\n");

}

}

#else

printf("-------------------------\n");

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

{

printf("| ");

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

{

printf("%d ", sudoku[i][j]);

if(0 == (j + 1) % 3 )

{

printf("| ");

}

}

printf("\n");

if(0 == (i + 1) % 3 )

{

printf("-------------------------\n");

}

}

#endif

//\*/

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

{

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

{

duplicate[i][j] = sudoku[i][j];

}

}

do

{

print\_menu();

//scanf("%d", &sel);

sel = getchar();

switch(sel)

{

case '1' : //start game

start\_game();

break;

case '2' : //select level

getchar(); // clear CR

select\_level();

printf("level is %d\n", level);

break;

case '3' : //show answer;

print\_answer();

break;

case '4' : //exit

printf("Goodbye!\n");

break;

default:

printf("input error!\n\n");

if('\n' != sel)

{

getchar(); // clear CR

}

break;

}

//printf("sel is %d\n", sel);

}while('4' != sel);

return 1;

}

void GenerateData()

{

int i, j, k;

bool ret;

int number;

int rand\_num;

int count;

int cnt;

ret = TRUE;

//srand((uint)time(NULL));

for(i = 1; i < 8; i++)

{

count = 0;

cnt = 0;

for(j = 1; j < 9; j++)

{

number = sudoku[i][j]; //store the previous data

sudoku[i][j] = get\_random\_number(i, j, &rand\_num);

ret = JudgeData(i, j, 0, sudoku);

//printf("ran = %d, num = %d \n", rand\_num, sudoku[i][j]);

count++;

if(FALSE == ret)

{

/\*

2 3 6 4 8 7 1 5 9

4 1 8 5 6 9 7 2 3

9 7 5 2 3 1 4 6 8

3 8 1 6 2 4 5 9 7

6 2 7 1 5 3 8 4 9

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

ran = 0, num = 9

sudoku[0][8] = 9, sudoku[4][8] = 9 col NG!

\*/

//go into deed loop.

sudoku[i][j] = number; //restore the previous data

if(count > 8)

{

//printf("count > 8\n");

for(k = 1; k < 9; k++)//reset the the current whole line except column 0

{

sudoku[i][k] = 0;

}

j = 0; //recompute the whole line.

count = 0;

cnt++;

if(cnt > 3)

{

/\*

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

9 8 7 6 5 4 3 2 1 ------------------0

7 3 2 8 5 9 6 1 4

4 8 6 7 1 2 3 5 9

1 5 9 4 6 3 7 8 2

5 7 1 2 9 4 8 6 3

2 4 3 6 8 7 1 9 5

8 9 9 0 0 0 0 0 0

9 0 0 0 0 0 0 0 0

3 0 0 0 0 0 0 0 0

6 0 0 0 0 0 0 0 0

go into dead loop. 6 and 9 are present in column 2, so the line 5 can never be filled success.

\*/

//printf("cnt > 3\n");

if(i > 1)

{

for(k = 1; k < 9; k++)//reset the rest column of the current line

{

sudoku[i - 1][k] = 0;

}

i -= 2; //recompute the previous line.

}

else //i = 1

{

for(k = 1; k < 9; k++)//reset the rest column of the current line

{

sudoku[i][k] = 0;

}

i = 0;

}

}

}

else

{

if(j > 1)

{

k = j - 1;

j = j - 2;

}

else

{

k = j;

j = 0;

}

for(; k < 9; k++)//reset the rest column of the current line

{

sudoku[i][k] = 0;

}

}

}

else

{

number = num[rand\_num];

num[rand\_num] = num[8 - j]; //switch data to the last not used one

num[8 - j] = number;

//count = 0; the loop may go into dead loop.

}

}

}

}

void fill\_first\_line()

{

int i, j, temp;

//printf("first line: ");

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

{

j = (uint)(random() \* (9.0 - i) / (RAND\_MAX + 0.0)); // random number [0, 9);

temp = num[j];

num[j] = num[8 - i]; //switch data to the last not used one

num[8 - i] = temp;

sudoku[0][i] = temp;

//printf("%d ", sudoku[0][i]);

}

//printf("\n");

/\*

printf("num array: ");

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

printf("%d ", num[i]);

printf("\n");

\*/

}

void fill\_first\_column()

{

int i, j, temp;

int range;

//reinitial the number array

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

{

num[i] = i + 1;

}

//delete sudoku[0][0] from number array

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

{

if(num[i] == sudoku[0][0])

{

//printf("i= %d, sudoku[0][0] = %d\n", i, sudoku[0][0]);

num[i] = num[8];

num[8] = sudoku[0][0];

break;

}

}

//check delete data

/\*

printf("delete data: ");

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

{

printf("%d ", num[i]);

}

printf("\n");

//\*/

temp = 0;

for(i = 1; i < 9; i++)

{

range = get\_random\_range(i, 0);

//if(range > 0)

{

j = (uint)(random() \* (range + 0.0) / (RAND\_MAX + 0.0)); // random number [0, 9);

temp = num[j];

num[j] = num[8 - i]; //switch data to the last not used one //鏁扮粍琚€掕繃鏉ュ瓨浜嗐€傘€傘€?

num[8 - i] = temp;

sudoku[i][0] = temp;

}

}

}

void fill\_last\_line()

{

int i, j, temp;

int total;

total = 0;

temp = 0;

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

{

total += num[i];

}

for(j = 1; j < 9; j++)

{

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

{

temp += sudoku[i][j];

}

sudoku[8][j] = total - temp;

temp = 0;

}

}

int get\_random\_number(int line, int column, int \*rand\_number)

{

int j, temp;

int range;

assert(line < 9);

assert(column < 9);

temp = 0;

range = get\_random\_range(line, column);

//if(range > 0)

{

j = (uint)(random() \* (range + 0.0) / (RAND\_MAX + 0.0)); // random number [0, 9);

temp = num[j];

\*rand\_number = j;

}

return temp;

}

bool is\_line\_ok(int line, int is\_subject, int (\*array)[9])

{

int column;

bool ret;

int i;

assert(line < 9);

ret = TRUE;

/\*

if(1 == is\_subject)

{

for(column = 0; column < 8; column++)

{

for(i = column + 1; i < 9; i++)

{

if((duplicate[line][column] == duplicate[line][i]) && (duplicate[line][column] != 0))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], line, i, sudoku[line][i]);

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, line + 1, i + 1);

ret = FALSE;

goto exit\_function;

}

}

}

}

else

{

for(column = 0; column < 8; column++)

{

for(i = column + 1; i < 9; i++)

{

if((sudoku[line][column] == sudoku[line][i]) && (sudoku[line][column] != 0))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], line, i, sudoku[line][i]);

ret = FALSE;

goto exit\_function;

}

}

}

}

\*/

for(column = 0; column < 8; column++)

{

for(i = column + 1; i < 9; i++)

{

if((array[line][column] == array[line][i]) && (array[line][column] != 0))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], line, i, sudoku[line][i]);

if(1 == is\_subject)

{

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, line + 1, i + 1);

}

ret = FALSE;

goto exit\_function;

}

}

}

exit\_function:

return ret;

}

bool is\_column\_ok(int column, int is\_subject, int (\*array)[9])

{

int line;

bool ret;

int i;

assert(column < 9);

ret = TRUE;

/\*

if(1 == is\_subject)

{

for(line = 0; line < 8; line++)

{

for(i = line + 1; i < 9; i++)

{

if((duplicate[line][column] == duplicate[i][column]) && (duplicate[line][column]) != 0)

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, column, sudoku[i][column]);

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, i + 1, column + 1);

ret = FALSE;

goto exit\_function;

}

}

}

}

else

{

for(line = 0; line < 8; line++)

{

for(i = line + 1; i < 9; i++)

{

if((sudoku[line][column] == sudoku[i][column]) && (sudoku[line][column]) != 0)

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, column, sudoku[i][column]);

ret = FALSE;

goto exit\_function;

}

}

}

}

\*/

for(line = 0; line < 8; line++)

{

for(i = line + 1; i < 9; i++)

{

if((array[line][column] == array[i][column]) && (array[line][column]) != 0)

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, column, sudoku[i][column]);

if(1 == is\_subject)

{

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, i + 1, column + 1);

}

ret = FALSE;

goto exit\_function;

}

}

}

exit\_function:

return ret;

}

bool is\_matrix\_ok(int line, int column, int is\_subject, int (\*array)[9])

{

int startline, endline;

int startcolumn, endcolumn;

int i, j;

bool ret;

assert(line < 9);

assert(column < 9);

ret = TRUE;

startline = line/3 \* 3;

endline = startline + 3;

startcolumn = column/3 \* 3;

endcolumn = startcolumn + 3;

//printf("line = %d, column = %d \n", line, column);

//printf("sl = %d, el = %d \n", startline, endline);

//printf("sc = %d, ec = %d \n", startcolumn, endcolumn);

/\*

if(1 == is\_subject)

{

for(i = startline; i < endline; i++)

{

for(j = startcolumn; j < endcolumn; j++)

{

if((duplicate[i][j] == duplicate[line][column]) && (duplicate[line][column] != 0) && (i != line))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, j, sudoku[i][j]);

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, i + 1, j + 1);

ret = FALSE;

goto exit\_function;

}

}

}

}

else

{

for(i = startline; i < endline; i++)

{

for(j = startcolumn; j < endcolumn; j++)

{

if((sudoku[i][j] == sudoku[line][column]) && (sudoku[line][column] != 0) && (i != line))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, j, sudoku[i][j]);

ret = FALSE;

goto exit\_function;

}

}

}

}

\*/

for(i = startline; i < endline; i++)

{

for(j = startcolumn; j < endcolumn; j++)

{

if((array[i][j] == array[line][column]) && (array[line][column] != 0) && (i != line))

{

//printf("sudoku[%d][%d] = %d, sudoku[%d][%d] = %d ", line, column, sudoku[line][column], i, j, sudoku[i][j]);

if(1 == is\_subject)

{

printf("(%d, %d) is the same as (%d, %d)!\n", line + 1, column + 1, i + 1, j + 1);

}

ret = FALSE;

goto exit\_function;

}

}

}

exit\_function:

return ret;

}

bool JudgeData(int line, int column, int is\_subject, int (\*array)[9])

{

bool ret;

assert(line < 9);

assert(column < 9);

ret = TRUE;

ret = is\_line\_ok(line, is\_subject, array);

if(ret == FALSE)

{

//printf("line NG!\n");

goto exit\_function;

}

ret = is\_column\_ok(column, is\_subject, array);

if(ret == FALSE)

{

//printf("col NG!\n");

goto exit\_function;

}

ret = is\_matrix\_ok(line, column, is\_subject, array);

if(ret == FALSE)

{

//printf("mtr NG!\n");

goto exit\_function;

}

exit\_function:

/\*

if(ret == FALSE)

{

int i, j;

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

{

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

{

printf("%d ", sudoku[i][j]);

}

printf("\n");

}

printf("--\n");

}

\*/

return ret;

}

int get\_random\_range(int in\_line, int in\_column)

{

int i, j, k;

int count;

int range\_array[9];

int getdata;

int startline, endline;

int startcolumn, endcolumn;

int temp[9];

int tmp;

//printf("get\_random\_range\n");

assert(in\_line < 9);

assert(in\_column < 9);

startline = in\_line/3 \* 3;

endline = startline + 3;

startcolumn = in\_column/3 \* 3;

endcolumn = startcolumn + 3;

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

{

range\_array[i] = i + 1;

temp[i] = 10;

}

count = 0;

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

{

getdata = 0;

//strip the data in line

for(i = 0; i < in\_line + 1; i++)

{

if(range\_array[k] == sudoku[i][in\_column])

{

//printf("line data: i = %d, col = %d, k = %d, data = %d, a[k]=%d\n", i, in\_column, k, sudoku[i][in\_column], range\_array[k]);

getdata = 1;

temp[k] = range\_array[k];

//temp[count] = range\_array[k];

range\_array[k] = 10;

count++;

break;

}

}

if(getdata == 1)

{

continue;

}

//strip the data in column

for(j = 0; j < in\_column + 1; j++)

{

if(range\_array[k] == sudoku[in\_line][j])

{

//printf("col data: line = %d, j = %d, k = %d, data = %d, a[k]=%d\n", in\_line, j, k, sudoku[in\_line][j], range\_array[k]);

getdata = 1;

temp[k] = range\_array[k];

//temp[count] = range\_array[k];

range\_array[k] = 10;

count++;

break;

}

}

if(getdata == 1)

{

continue;

}

//strip the data in the 3\*3 matrix

for(i = startline; i < endline; i++)

{

for(j = startcolumn; j < endcolumn; j++)

{

if((sudoku[i][j] == range\_array[k]))

{

//printf("mtr data: i=%d, j = %d, k = %d, data = %d, a[k] = %d\n", i, j, k, sudoku[i][j], range\_array[k]);

getdata = 1;

temp[k] = range\_array[k];

//temp[count] = range\_array[k];

range\_array[k] = 10;

count++;

break;

}

}

if(1 == getdata)

{

break;

}

}

}

for(i = 0; i < 8; i++)

{

for(j = i + 1; j < 9; j++)

{

if(range\_array[i] > range\_array[j])

{

tmp = range\_array[i];

range\_array[i] = range\_array[j];

range\_array[j] = tmp;

}

//\*

if(temp[i] < temp[j])

{

tmp = temp[i];

temp[i] = temp[j];

temp[j] = tmp;

}

//\*/

}

}

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

{

if(10 != range\_array[i])

{

num[i] = range\_array[i];

}

else

{

num[i] = temp[i];

}

//printf("%d ", num[i]);

}

//printf("------------------%d\n", (9 - count));

return (9 - count);

}

void print\_answer()

{

int i, j;

printf("-------------------------\n");

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

{

printf("| ");

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

{

printf("%d ", sudoku[i][j]);

if(0 == (j + 1) % 3 )

{

printf("| ");

}

}

printf("\n");

if(0 == (i + 1) % 3 )

{

printf("-------------------------\n");

}

}

}

void print\_menu()

{

printf("please input the number to select the item: (1 ~ 3)\n1. start game\n2. select level\n3. show answer\n4. exit\n");

}

void select\_level()

{

//int sel;

char sel;

printoption:

print\_option();

//scanf("%d", &sel);

sel = getchar();

switch(sel)

{

case '1' :

case '2' :

case '3' :

level = sel - '0';

break;

default:

printf("input error!\n\n");

if('\n' != sel)

{

getchar(); // clear CR

}

goto printoption;

break;

}

getchar(); //clear CR

}

void print\_option()

{

printf("please select the level: (1 ~ 3)\n1. easy\n2. normal(default)\n3. hard\n");

}

void start\_game()

{

int hide\_number; // total hide number is 5, 10, 15.

int \*hide\_index; //array of hide item. hide item index is between 0 and 80.

int i, j;

uint line, column, number;

bool ret;

hide\_number = level \* 5;

hide\_index = (int \*)malloc(hide\_number \* sizeof(int));

//init the array.

for(i = 0; i < hide\_number; i++)

{

hide\_index[i] = 90;

}

for(i = 0; i < hide\_number; i++)

{

hide\_index[i] = (uint)(random() \* (81 + 0.0) / (RAND\_MAX + 0.0)); // random number [0, 81);

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

{

if(hide\_index[j] == hide\_index[i])

{

i--;

break;

}

}

}

//print the array;

/\*

printf("\n");

for(i = 0; i < hide\_number; i++)

{

printf("%d ", hide\_index[i]);

}

printf("\n");

\*/

generate\_subject(hide\_number, hide\_index);

display\_subject();

printf("Please input the item as line,column,number. \nFor example 1,8,5 means fill the 1st line, the 8th column with number 5.\nLine, column and number must be greater than 0 and less than 10.\nInput 0,0,0 to quit. input: ");

while(1)

{

scanf("%d,%d,%d", &line, &column, &number);

getchar();//clear CR

if(0 == line && 0 == column && 0 == number)

{

break;

}

if(1 > line || line > 9)

{

printf("input line number error. please input again: ");

continue;

}

if(1 > column || column > 9)

{

printf("input column number error. please input again: ");

continue;

}

if(0 != duplicate[line - 1][column -1])

{

printf("(%d, %d) is not blank. please input again: ", line, column);

continue;

}

duplicate[line - 1][column - 1] = number;

ret = JudgeData((line - 1), (column - 1), 1, duplicate);

if(FALSE == ret)

{

duplicate[line - 1][column - 1] = 0;

}

else

{

hide\_number--;

display\_subject();

if(0 == hide\_number)

{

printf("Congratulations!\n\n");

break;

}

}

printf("please input again: ");

}

free(hide\_index);

}

void generate\_subject(int hide\_number, int \*hide\_item)

{

int i;

int index;

for(i = 0; i < hide\_number; i++)

{

index = hide\_item[i];

duplicate[index / 9][index % 9] = 0;

}

}

void display\_subject()

{

int i, j;

printf("\n 1 2 3 4 5 6 7 8 9\n");

printf(" -------------------------\n");

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

{

printf("%d | ", (i + 1));

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

{

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

{

printf(" ");

}

else

{

printf("%d ", duplicate[i][j]);

}

if(0 == (j + 1) % 3 )

{

printf("| ");

}

}

printf("\n");

if(0 == (i + 1) % 3 )

{

printf(" -------------------------\n");

}

}

}