****

[**CS-2006**](https://classroom.google.com/c/NTM4MDQ5NzM1Mzgw) **Operating System**

**Project Report**

**Project By:**

**Rabia Mustafa:** 20i-2853

**Azka Asim:** 20i-2478

For this project phase 1, we have analyzed that when a row has an invalid entry, the output will tell the

* The invalid entry
* Row number having invalid entry
* The position of the invalid entry in that row.

For the columns, if a column have invalid entry, than the output will show

* The invalid entry
* Column number having invalid entry
* The index of column having invalid entry

For the sub boxes the invalid entry is considered only when a digit is repeated. The first occurrence will be valid and the repeated (second) occurrence will be invalid. The output will depict:

* The invalid entry
* The index within the box of invalid entry
* The column and row number having invalid entry

1. **Pseudocode For Phase 1:**

initialize main array[9][9]

/\*{

{6, 2, 4, 7, 2, 9, 1, 8, 7},

{5, 1, 9, 7, 2, 10, 6, 3, 4},

{8, 3, 7, 11, 1, 4, 2, 9, 5},

{1, 4, 3, 8, 6, 5, 7, 2, 9},

{9, 5, 8, 2, 4, 7, 3, 6, 1},

{7, 6, 2, 3, 9, 1, 4, 5, 8},

{3, 7, 1, 9, 5, 6, 8, 4, 2},

{4, 9, 6, 1, 8, 2, 5, 7, 3},

{2, 8, 5, 4, 7, 3, 9, 1, 6}};\*/

structure rowcheck/columncheck

integer rownumber

structure BoxCheck

integer starting row

integer starting column

integer resultarray

function validrow

// initialize resultarray with 0

//RN=rownumber; CN=columnnumber

//Get values of rownumber/columnnumber from Structure created

//locking the other threads

//outerloop

for i in RN

//innerloop

for j in columns //9

//reading row whose validity is to be checked

//resultarray counts the valid/invalid enteries

//enteriesarray stores the enteries obtained

//unlock the mutex

function validcolumn

// initialize resultarray with 0

//RN=rownumber; CN=columnnumber

//Get values of rownumber/columnnumber from Structure created

//locking the other threads

//outerloop

for i in rownumber //9

//innerloop

for j in CN //for inner columns

//reading row whose validity is to be checked

//resultarray counts the valid/invalid enteries

//enteriesarray stores the enteries obtained

//unlock the mutex

function ValidBox

// initialize resultarray with 0

//Get values of rownumber/columnnumber from Structure created (Box check)

//mutex locking

for i in RN

//innerloop

for j in CN //for inner columns

//reading row whose validity is to be checked

//resultarray counts the valid/invalid enteries

//enteriesarray stores the enteries obtained

//unlocking mutex

**Main()**

**//Checking validity for row numbers**

for i in Rows

Create pthread (call validrow function())

join pthread

//checks for invalid enteries

//prints the location and invalid entry value along with thread id

for i 1 to 9

//

for k to 9

//if count in result array>1 than it will print value and position

for i to 9

//checks the enteries array

//if value <1 and value>9 than it will return the position and value

//if invalid enteries count 0 than valid row

//else invalid count++

//cancel the thread

**//Checking validity for column numbers**

for i in Column

Create pthread (call validrow function())

join pthread

//checks for invalid enteries

//prints the location and invalid entry value along with thread id

for i 1 to 9

//

for k to 9

//if count in result array>1 than it will print value and position

for i to 9

//checks the enteries array

//if value <1 and value>9 than it will return the position and value

//if invalid enteries count 0 than valid column

//else invalid count++

//cancel the thread

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ONLY FOR 1 BOX\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\*\*\*\*\*\*\*\*\*\*\*SAME FOR ALL THE REST OF 8 BOXES EXCEPT THE Rows NUMBER AND COLUMN NUMBERS ACCORDING TO EACH BOX\*\*\*\*\*\*\*\*\*\*/**

**//Checking validity for Boxes**

//creating threads for the validity of boxes

for i to 9

if resultarray is equal to 1

tempboxnum<-1 //box validity is true

else

resultarray not equals to 0 then

print invalid entry

print rows and columns for invalid enteries

// for values greater than 9 and smaller than 0

//if greater than 9 and smaller than 0

//box invalid position and location is printed

else

it is a valid box

//also printing the invalid enteries ;if any by comparing the values

//printing the numbers that are invalid; the count of invalid enteries

//cancelling box thread

1. **Pseudocode For Phase 2:**

// Declare A structure for rowcheck/ columncheck

//Declare A structure for ValueCheck

// MainArray // that will be actual puzzle

function setInvalid

//if row validity and box validity and column validity is 0

//place that particular missing value in that row or column

//increment invalid entries, threadcount and moves

function Check

// intialize rowvalidtyArray ColumnValidtyArray and BoxValidtyArray wit zero

// get values Row Column Values from structure

for i=RN

for j=0 to columns

// get values for desire row

for i=0 to rows

for j=CN

// get values for desire Column

//Sub Box 1

// applying condition on RN and CN

for loop to iterate over rows

inner loop to iterate over columns

//Get value of desire box in checkBoxArray

// Same process goes for sub box 2 to 9

for loop i=0 to 9

// making RowvalidtyArray array

for loop i=0 to 9

// making ColumnvalidtyArray array

for loop i=0 to 9

// making BoxvalidtyArray array

for loop on RowvalidtyArray

// if any value in row validty array is not equal to 1 it will generate rowWarning

for loop on ColumnvalidtyArray

// if any value in Columnvalidty array is not equal to 1 it will generate rowWarning

for loop on BoxvalidtyArray

// if any value in BoxvalidtyArray array is not equal to 1 it will generate rowWarning

if boxwarning rowwarning and columnwarning is equal to one

thread will be created

it will call setInvalid function //it will place the correct values at correct position

**Main()**

//printing original puzzle

//outer and inner for loop to call the check function for all the enteries of puzzle

//getting thread id doing this check

than

//print the replaced puzzle.

1. **Concepts of Operating Systems Implemented:**

In this project we implemented the concepts of threads. By declaring functions for the threads as well as by getting the ids of the thread that is working on what part of the program.

We implemented the concept of mutex lock and semaphores so that the function of each thread is synchronized according to the execution of the program. For example if 1 thread is executing, the other threads will wait for the thread to get executed and will only function once the thread is unlocked.

The concept of joinable thread was used so that once output of 1 thread is obtained then the other thread will be executed.

1. **Codes Implemented:**

**Phase I:**

Code was quite long hence attached the .cpp file.

**Phase II:**

#include <iostream>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <cstring>

#include <thread>

#include <fstream>

#include <semaphore.h>

using namespace std;

int moves = 0;

int \*checkRowArray = new int[9];

int \*checkColumnArray = new int[9];

int \*checkBoxArray = new int[9];

int \*RowvalidtyArray = new int[9];

int invaildentriesCount = 0;

int \*BoxvalidtyArray = new int[9];

int \*ColumnvalidtyArray = new int[9];

int boxWarning = 0, columnWarning = 0, rowWarning = 0;

int RN, CN, valuetobereplaced; // struct

int threadCount = 0;

sem\_t sem;

struct rowcheck

{

// int columnnumber;

int rownumber;

};

struct ValueCheck

{

int Row;

int Column;

};

int MAINARRAY[9][9] = {

{6, 2, 4, 5, 3, 4, 4, 8, 7},

{5, 1, 9, 7, 2, 8, 6, 3, 4},

{8, 3, 7, 6, 1, 4, 2, 9, 5},

{1, 4, 3, 8, 6, 5, 7, 2, 9},

{9, 5, 8, 2, 4, 7, 3, 6, 1},

{7, 6, 2, 3, 9, 1, 4, 5, 8},

{3, 7, 1, 9, 1, 6, 8, 4, 2},

{4, 9, 6, 1, 8, 2, 5, 7, 3},

{2, 8, 5, 4, 7, 3, 9, 1, 6}

};

void \*setInvalid(void \*vargp)

{

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

{

if (RowvalidtyArray[i] == 0 && BoxvalidtyArray[i] == 0 && ColumnvalidtyArray[i] == 0)

{

cout << "Value to be changed is: " << MAINARRAY[RN][CN] << " at Row: " << RN + 1 << " at Column: " << CN + 1 << endl;

valuetobereplaced = i + 1;

MAINARRAY[RN][CN] = valuetobereplaced;

moves++;

threadCount++;

invaildentriesCount++;

}

}

return 0;

}

void \*Check(void \*vargp)

{

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

{

RowvalidtyArray[i] = 0;

ColumnvalidtyArray[i] = 0;

BoxvalidtyArray[i] = 0;

}

// row column box

boxWarning = 0, columnWarning = 0, rowWarning = 0;

ValueCheck \*ptr = (ValueCheck \*)vargp;

RN = (\*ptr).Row;

CN = (\*ptr).Column;

RN = RN - 1;

CN = CN - 1;

for (int i = RN; i <= RN; i++)

{

// adding values of rows

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

{

checkRowArray[j] = MAINARRAY[i][j];

}

}

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

{ // adding values of Columns

for (int j = CN; j <= CN; j++)

{

checkColumnArray[i] = MAINARRAY[i][j];

}

}

// Sub Box1

if (RN <= 2 && CN <= 2)

{

int tempcount = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box2 horizontal

if ((RN <= 2) && (CN > 2 && CN < 5))

{

int tempcount = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 3; j < 6; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box3 horizontal

if ((RN <= 2) && (CN > 5 && CN < 9))

{

int tempcount = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 6; j < 9; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box4 verticle

if ((RN > 2 && RN < 6) && (CN <= 2))

{

int tempcount = 0;

for (int i = 3; i < 6; i++)

{

for (int j = 0; j < 3; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box5 horizontal

if ((RN > 2 && RN < 6) && (CN > 2 && CN < 5))

{

int tempcount = 0;

for (int i = 3; i < 6; i++)

{

for (int j = 3; j < 6; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box6 horizontal

if ((RN > 2 && RN < 6) && (CN > 5 && CN < 9))

{

int tempcount = 0;

for (int i = 3; i < 6; i++)

{

for (int j = 6; j < 9; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box7 verticle

if ((RN > 6 && RN <= 8) && (CN <= 2))

{

int tempcount = 0;

for (int i = 6; i < 9; i++)

{

for (int j = 0; j < 3; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box 8 horizontal

if ((RN > 6 && RN <= 8) && (CN > 2 && CN < 5))

{

int tempcount = 0;

for (int i = 6; i < 9; i++)

{

for (int j = 3; j < 6; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

// Sub Box9 horizontal

if ((RN > 6 && RN <= 8) && (CN > 5 && CN < 9))

{

int tempcount = 0;

for (int i = 6; i < 9; i++)

{

for (int j = 6; j < 9; j++)

{

checkBoxArray[tempcount] = MAINARRAY[i][j];

tempcount++;

}

}

}

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

{ // checking desire rows

RowvalidtyArray[checkRowArray[i] - 1] += 1;

}

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

{ // checking desire Columns

ColumnvalidtyArray[checkColumnArray[i] - 1] += 1;

}

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

{ // checking desire Columns

BoxvalidtyArray[checkBoxArray[i] - 1] += 1;

}

int valuetobereplaced = 0;

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

{

if (RowvalidtyArray[i] != 1)

{ // we have to do changes

rowWarning = 1;

}

}

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

{

if (ColumnvalidtyArray[i] != 1)

{

columnWarning = 1;

}

}

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

{

if (BoxvalidtyArray[i] != 1)

{

boxWarning = 1;

}

}

if (boxWarning == 1 && columnWarning == 1 && rowWarning == 1)

{

// actual value will be changed here

pthread\_t callSetThread;

pthread\_create(&callSetThread, NULL, setInvalid, NULL);

pthread\_join(callSetThread, NULL);

cout<<"Threads setting the invalid values: "<<pthread\_self()<<" at location "<<RN+1<<" and "<<CN+1<<endl;

//cout<<endl;

// sem\_wait(&sem) ;

// threadCount++;

// cout<<"threadCount="<<threadCount;

// sem\_post(&sem) ;

}

return 0;

}

int main()

{

cout << "Original Puzzlee: " << endl;

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

{

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

{

cout << MAINARRAY[i][j] << " ";

}

cout << endl;

}

// Checking the validty Of Matrix

// ///////////

// /////////

pthread\_t thread\_id1;

struct ValueCheck p1;

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

{

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

{

// sem\_wait(&sem) ;

p1 = {i + 1, j + 1};

pthread\_create(&thread\_id1, NULL, Check, (void \*)&p1);

pthread\_join(thread\_id1, NULL);

// sem\_post(&sem) ;

cout<<"Threads checking Invalid positions: "<<pthread\_self()<<" at location "<<i<<" and "<<j<<endl;

}

}

cout << "\nInvalid entries count: " << invaildentriesCount << endl;

cout << "\nmoves count: " << moves << endl;

cout << "Thread count: " << threadCount << endl;

cout<<"\nReplaced Matrix: "<<endl;

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

{

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

{

cout << MAINARRAY[i][j] << " ";

}

cout << endl;

}

}

1. **System Specifications:**

There are not any special specifications to be used for the execution of this project.

Pthread.h, semaphore.h, iostream, stdlib, unistd.h are the basic libraries to be included in the file for execution.

1. **Implementation of these concepts/this logic in some other scenario:**

As the implementation is checking the validity of the digits and also the validity of board of sudoku, the same concepts can be applied to check the valid move or to interpret the safe place for Queen in the game of chess. The concepts of phase 1 can help to validate whether the moves of king, rooks, bishop and knights are valid or not.

Phase 2 will help to swap/ create a valid move when there is an invalid movement.