Introduction to Computing

=========================

Assignment 1 Autumn 2017

=========================

For this assignment we will write a program which checks whether

a certain input is a valid nxn sudoku or not.

For our purposes, an nxn sudoku is a collection of numbers

organised in rows, columns and subsquares. We consider each of the rows,

columns, and subsquares to have n elements, with n being a square number

greater than 1: 4, 9, 16 and so on. It follows that a 4x4 sudoku

will contain 16 elements, a 9x9 one 81 elements, and so on.

For instance the following is a 4x4 sudoku that we could expect

in input for our program:

1 2 3 4

4 1 2 76

4 4 4 1

9 4 1 -7

The rows are:

1 2 3 4; 4 1 2 76; 4 4 4 1; 9 4 1 -7

The columns are:

1 4 4 9; 2 1 4 4; 3 2 4 1; 4 76 1 -7

The subsquares are:

1 2 4 1; 3 4 2 76; 4 4 9 4; 4 1 1 -7

We say that an nxn sudoku is \_valid\_ if each of the rows, columns

and subsquares contains all the numbers from 1 to n.

The sudoku above is clearly not a valid one.

The following 9x9 sudoku is a valid one:

8 4 2 5 7 1 9 3 6

3 5 1 8 6 9 7 4 2

7 6 9 2 4 3 1 8 5

5 9 7 6 2 8 3 1 4

6 2 3 1 9 4 5 7 8

4 1 8 3 5 7 2 6 9

9 8 6 7 3 2 4 5 1

1 7 4 9 8 5 6 2 3

2 3 5 4 1 6 8 9 7

File scd.cpp contains an initial draft for the program and

additional instructions. Complete the program and submit

the source file.

The program reads the input and prints it on the screen. It

determines whether the sudoku is a valid one or not and prints

the outcome on the screen.

As you can see, the sudoku is read from a text file: the program

asks the user to enter the name of the file containing the input.

For this to work, the executable and the input text file need to

be in the same directory but this may not work as expected if

you use an IDE to launch the program.

So if this aspect doesn't work as expected please execute the

program from a Terminal.

In order to test the program you will have to create some text

files containing sudokus. When our program requires the name of

the file in input, this needs to be entered including any

extensions (for instance .txt) that are part of the name of the

file.

Some guidelines for this assignment:

- None of the functions, except for the main, should contain user

or file input or output (std::ifstream, std::cin, std::cout,

etc) in their implementation: std::ifstream, std::cin,

std::cout, etc are only allowed in the main.

- All the variables should be declared in the scope of a function

(either the main or some other one): global variables are not

allowed.

- All the loops should be controlled either by the loop condition

or by return. Statements such as switch, break, continue, goto

are not allowed anywhere in the program.

- Compile and test your program on Linux on the lab computers before

submitting in order to make sure you are not using any non-standard features.

- If you use features introduced in the C++11 (or later) version of the standard

you need to specify this in a comment on top of your source file.

#include <iostream>

#include <string>

#include <cmath>

#include <cstdlib>

#include <fstream>

#include <vector>

// do not alter the header inclusion

// do not include any additional headers

// do not alter the function declarations

// do not add any other functions to the program

// see below for more information about these functions

bool valid\_sudoku(const std::vector<int>& g);

bool check\_sequence(const std::vector<int>& v);

int mfind(int n, const std::vector<int>& v);

void extract\_row(int r, const std::vector<int>& in, std::vector<int>& out);

void extract\_col(int c, const std::vector<int>& in, std::vector<int>& out);

void extract\_subsq(int subs, const std::vector<int>& in, std::vector<int>& out);

int xytoi(int x, int y, int rowlen);

// do not alter the main

int main(){

std::vector<int> s;

// we are going to represent the sudoku as a one dimensional vector

// linearised by row, for instance the 4x4 sudoku in the readme will be

// represented as: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7

// begin of block of code reading the sudoku from the text file

std::string filename;

std::cout << "plese enter name of file containing the sudoku" << std::endl;

std::cin >> filename;

std::ifstream infile;

infile.open(filename.c\_str());

if(!infile.is\_open()){

std::cout << "error, can't open input file" << std::endl;

exit(EXIT\_FAILURE);

}

int tmp;

while(infile >> tmp){

s.push\_back(tmp);

}

// end of block of code reading the sudoku from the text file

// we now have the content of the sudoku in vector s

int side = std::sqrt(s.size());

// assuming it is a valid square grid in terms of size

// (if it isn't, the program will not work properly, which is expected)

// printing the content of the sudoku (as a square);

// the sudoku is stored as a one-dimensional vector

// but the vector can be indexed using two indices

// by converting the indices of a two dimensional representation

// into the corresponding one-dimensional index (using our function xytoi)

for(int i = 0; i < side; i++){

for(int j = 0; j < side; j++){

std::cout << s[xytoi(i,j,side)] << " ";

}

std::cout << std::endl;

}

// calling the function checking if the sudoku is a valid one:

bool valid = valid\_sudoku(s);

if(valid){

std::cout << "valid" << std::endl;

}

else{

std::cout << "not valid" << std::endl;

}

return 0;

}

// function xytoi maps two-dimensional indices to a one-dimensional one

// it takes in input:

// - the two indices x and y of a two dimensional data structure

// - the length of the row, rowlen

// it returns:

// - the corresponding one-dimensional index

int xytoi(int x, int y, int rowlen){

return x\*rowlen+y;

}

// function valid\_sudoku

// takes in input:

// - a one dimensional vector representing a sudoku

// returns:

// - a boolean which is true if the sudoku is valid and false if it isn't valid

// write your code where indicated

// i.e. within the two remaining incomplete `for' loops

// do not alter anything else

bool valid\_sudoku(const std::vector<int>& g){

int side = std::sqrt(g.size());

// for each row...

for(int i = 0; i < side; i++){

std::vector<int> row;

extract\_row(i, g, row);

if(!check\_sequence(row)){

return false;

}

}

// for each column...

for(int i = 0; i < side; i++){

// write your code here

}

// for each subsquare...

for(int i = 0; i < side; i++){

// write your code here

}

// if the program execution gets to here it means that

// it hasn't executed a "return false" above

// so the sudoku is valid:

return true;

}

// function mfind

// takes in input:

// - a vector of integers v

// - an integer n

// returns:

// the index of the element n in v, if n is in v

// -1 if n is not in v

// for instance:

// v: 3 2 5; n: 2 | return: 1

// v: 3 1 2; n: -6 | return: -1

// v: 1 1 -12 1 5; n: 5 | return: 4

int mfind(int n, const std::vector<int>& v){

// write your code here

}

// function check\_sequence

// takes in input:

// - a vector of integers v

// returns:

// - true if v contains all the numbers from 1 to v.size() and false otherwise

// for instance:

// v: 3 2 5 | return: false

// v: 3 1 2 | return: true

// v: 1 1 -12 1 5 | return: false

// note that check\_sequence should call mfind

bool check\_sequence(const std::vector<int>& v){

// write your code here

}

// note that all the following functions should call function xytoi

// function extract\_row

// takes in input:

// - a row index r

// - a one dimensional vector `in' representing a sudoku

// provides in output:

// - a vector out containing the r-th row of in

// for example:

// r: 0; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 1 2 3 4

// r: 3; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 9 4 1 -7

void extract\_row(int r, const std::vector<int>& in, std::vector<int>& out){

// write your code here

}

// function extract\_col

// like extract\_row but for columns

// for example:

// c: 0; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 1 4 4 9

void extract\_col(int c, const std::vector<int>& in, std::vector<int>& out){

// write your code here

}

// function extract\_subsq

// like the two functions above but for subsquares

// we consider subsquares to be indexed from left to right

// and then from top to bottom

// for example:

// subs: 0; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 1 2 4 1

// subs: 1; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 3 4 2 76

// subs: 2; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 4 4 9 4

// subs: 3; in: 1 2 3 4 4 1 2 76 4 4 4 1 9 4 1 -7 | out: 4 1 1 -7

void extract\_subsq(int subs, const std::vector<int>& in, std::vector<int>& out){

// write your code here

}