NIU CSCI 340 GRADE-O-MATIC ASSIGNMENT

ASSOCIATIVE CONTAINERS - INI FILES AND SUDOKU SOLVER

INTRODUCTION

This assignment gives you an opportunity to work with both of the main STL associative container types.

- `std::map` will be used to contain configuration info loaded from a file in the INI file format.
- `std::set` will be used to store the remaining possibilities for each cell in a Sudoku grid solver

This assignment will use the `grid_row_major` class you will have written in a previous assignment. If you don't have a working version from the last assignment, then you will need to implement the portions you need to use in this program. It will not be tested again separately.

YOUR TASK

You are responsible for implementing all of the following functions:

- For the INI file functionality. The declarations for these are found in `iniparse.h` (which you should not alter), and you will be implementing the functions in `iniparse.cc`.
  - `add_ini_section(config, section)`
  - `remove_ini_section(config, section)`
  - `get_ini_key(config, section, key);`
  - `set_ini_key(config, section, key, value)`
  - `remove_ini_key(config, section, key)`
  - `read_ini(input, verbosity)`
  - `write_ini(ost, config);`
  - `print_ini(ost, config);`
- For the Sudoku solver. These are declared in `sudoku.h` (which you should not alter), and you are expected to provde the implementations in the file `sudoku.cc`.
  - `initialize_grid(grid)`
  - `set_sudoku_cell_known(grid, row, col, solution)`
  - `set_sudoku_cell_unknown(grid, row, col)`
  - `remove_sudoku_option(grid, row, col, value)`
  - `handle_row_for_cell(grid, row, col)`
  - `handle_col_for_cell(grid, row, col)`
  - `handle_subgrid_for_cell(grid, row, col)`
  - `load_sudoku_grid(filename, grid)`
  - `print_sudoku_grid(ost, grid, unknown, impossible)`

WHAT IS AN INI FILE?

INI files are files whose data was written in the INI file format. This format (INI is short for initialization) was commonly used to store configuration information for DOS and Windows applications, but it's been used for many things over the years, and it's a good application for our associative containers.

The idea is that there are several sections, each of which can contain several keys, with associated values. Here's an example of one:

```
; This is an INI file -- if the first non-whitespace character is a ;, it's a comment
[section1]
key1=value1
key2=value2

[section2]
laugh=haha
smile=:-)
```

The lines `[section1]` and `[section2]` are section headers. They indicate that the keys that follow will belong to the section named.

The `key=value` lines are declaring that a key (whose name is on the left hand side of `=`) exists in the most recently named section, and its value should be set to whatever's on the right hand side of the `=`.

So, in the above file, the section `section1` has two keys, `key1` and `key2`. The section `section2` has two keys, `laugh` and `smile`.

Comments start with `;`, and are ignored. Leading and trailing whitespace are ignored.

WHAT IS SUDOKU?

Sudoku is a common puzzle game where the player is presented a 9×9 grid, divided into 9, 3×3 subgrids. In this grid, cells can be filled with numbers $1 - -9$ or left blank to begin.

The goal of the puzzle is to fill any remaining blanks with the numbers $1 - -9$ such that:

- No number appears twice on the same row.
- No number appears twice on the same column.
- No number appears twice in the same subgrid.

We'll be writing a program that will use these constraints to automatically solve most Sudoku puzzles that are solveable.

WHAT IS `INI_CONFIG`?

The type `INI_CONFIG` is used to store the configuration data. The default type to be used is `map<string, map<string, string>>`.

The inner map links string keynames to their string values, and each will hold all of the keys for a particular section.

The outer map allows you to look up the appropriate inner map (based on the string section name) for a given section's keys.

WHAT IS `SUDOKUGRID`?

The type you see above, `SUDOKUGRID`, is a `typedef` for the type you are using to handle your Sudoku grid, whose data will consist of a 9×9 grid containing `std::set<int>` in each cell. These sets will be filled the with the numbers $1 - -9$ that are still possible for that cell in the puzzle.

In `settings.h`, I set up the default type for `SUDOKUGRID` as `grid_row_major<std::set<int>>`, which makes the test program store the grid of remaining possibilities using the class you coded for the grids assignment.

FUNCTIONS REQUIRED FOR INI PARSING

`add_ini_section(config, section)`

- `config` - The `INI_CONFIG` to add the section to.
- `section` - A string containing the name of the section to add.

If the section named already exists, do nothing. If not, create a new empty map to hold the keys the section may contain later and add it into the `INI_CONFIG` outer map using the section name as the key.

`remove_ini_section(config, section)`

- `config` - The `INI_CONFIG` to remove the section from.
- `section` - A string containing the name of the section to remove.

If the section named already exists, remove it and all of its keys. Return the number of sections actually removed.

`get_ini_key(config, section, key);`

- `config` - The `INI_CONFIG` to retrieve the key from.
- `section` - A string containing the name of the section containing the key desired.
- `key` - A string containing the name of the key whose value is desired.

If the given key exists in the given section, return its value, otherwise return an empty string.

`set_ini_key(config, section, key, value)`

- `config` - The `INI_CONFIG` to set the key in.
- `section` - A string containing the name of the section containing the key desired.
- `key` - A string containing the name of the key whose value should be set.
- `value` - A string for the new value of the key.

If the given key exists in the given section, set its value to `value`. If it did not previously exist, add it.

`remove_ini_key(config, section, key)`

- `config` - The `INI_CONFIG` to remove the key from.
- `section` - A string containing the name of the section containing the key.
- `key` - A string containing the name of the key to remove.

If the given key exists in the given section, remove it. Return the number of keys actually removed.

`read_ini(input, verbosity)`

- `input` - an input stream to read from. The data read will be interpreted as a file in the INI format.
- `verbosity` - The verbosity level – controls how much debug information will be shown
  - `0` - no debug information, be quiet and load
  - `1` - print when sections begin and when keys are detected
  - `2` - all of the debug information from `1` **and** print the line number and contents for each non-empty, non-comment line found

This function will start with an empty `INI_CONFIG`, then read all available data from the `input` stream a line at a time. Each line is interpreted in the INI file format:

- Lines with only whitespace (spaces or tabs) are empty and should be ignored
- Lines where the first non-whitespace character is `';'` are to be interpreted as comments, and ignored.
- Lines where the first non-whitespace character is `'['` are to be interpreted as heading naming the section that future keys are to be added into. The section name continues either until a `']'` (correct) or the end of the line (invalid but we'll deal with it).
- Non-comment lines of the form `key=value` are interpreted to mean that there is a key named `key` in the current section with value `value`. All of the values are to be kept as strings, and any conversion will happen later. The key will be the part before the `'='` and the value will be everything from the character after the `'='` until the end of the line, but with no leading or trailing whitespace kept.
- Lines that are not empty, are not comments, do not name a section, and do not set a key's value are invalid, and should be ignored.

Key names, section names, and values may exist in the file with whitespace around them, but they should have any leading or trailing whitespace removed from them before storage in your configuration.

This function returns the `INI_CONFIG` built up from the input file. All of the detected keys should have values in the maps for their corresponding sections. No empty sections should be kept.

`write_ini(ost, config);`

- `ost` - output stream to write the INI data to
- `config` the `INI_CONFIG` we are writing to the output stream

This function will output all of the keys found in `config` in such a way that the output could be used with `read_ini` to restore the same config (except for any sections that contain no keys).

`print_ini(ost, config);`

- `ost` - output stream to print to
- `config` the `INI_CONFIG` whose info we would like to print

This function prints the configuration data from `config` with one key per line, in the format `section_name.key_name = "value"`. Use a tab character and not a space between the key name and the equal sign so things will tend to line up more neatly.

If `config` is empty, print `"Configuration is empty. Nothing to print\n"` to `cout`. If any of the sections are empty, print a message in the format `Section "section_name" is empty\n`.

Print out an additional newline at the end of printing all of the keys.

FUNCTIONS REQUIRED FOR SUDOKU SOLVING

`initialize_grid(grid)` I've provided an implementation of this if you are using your `row_major_grid` from the previous assignment. If you're using something else, then this function is an opportunity to do any initialization that is necessary to prepare an object of the the type you chose for `SUDOKUGRID`.

`set_sudoku_cell_known(grid, row, col, solution)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're marking
- `col` - the column of the cell we're marking
- `solution` - the value we're marking for the cell

Marks the Sudoku cell in row `row` and column `col` as known by clearing the set of everything but the known answer, `solution`.

`set_sudoku_cell_unknown(grid, row, col)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're marking
- `col` - the column of the cell we're marking

Marks the Sudoku cell in row `row` and column `col` as unknown by filling the set with all possibilies ($1 - -9$).

`remove_sudoku_option(grid, row, col, value)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're removing an option from
- `col` - the column of the cell we're removing an option from

This removes the given value from the set of remaining possibilities for the cell at in the given row and column.

`handle_row_for_cell(grid, row, col)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're handling
- `col` - the column of the cell we're handling

If the cell at (row,col) is known (only one possibility remaining), remove its value from the possibilities of all of the *other* cells in the same row. If unknown, do nothing for now. The function should return the number of possibilities that were actually removed.

`handle_col_for_cell(grid, row, col)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're handling
- `col` - the column of the cell we're handling

If the cell at (row,col) is known (only one possibility remaining), remove its value from the possibilities of all of the *other* cells in the same column. If unknown, do nothing for now. The function should return the number of possibilities that were actually removed.

`handle_subgrid_for_cell(grid, row, col)`

- `grid` - the `SUDOKUGRID` to use
- `row` - the row of the cell we're handling
- `col` - the column of the cell we're handling

If the cell at (row,col) is known (only one possibility remaining), remove its value from the possibilities of all of the *other* cells in the same 3×3 subgrid. If unknown, do nothing for now. The function should return the number of possibilities that were actually removed.

load_sudoku_grid(filename, grid)

- `filename` - filename of the grid file that's storing our Sudoku grid
- `grid` - the `SUDOKUGRID` that will be populated based on the input data

This function reads in the Sudoku data from a file. The data is stored in the same format as was used to store files for the `grid_row_major`. The values will be integers from $0-9$. 0 will indicate that the cell was blank, and needs to be solved for. The numbers $1-{-}9$ indicate the known value for the cell.

Remember that the numbers in the file are integers, but your `SUDOKUGRID` contains sets which contain the values that remain possible for each field.

If the file cannot be opened, print an error message and return `false`.

If the dimensions of the grid in the input file are not 9×9, print an error message and return `false`.

If the file is successfully loaded, the function should return `true`.

print_sudoku_grid(ost, grid, unknown, impossible)

- `ost` - output stream to print to
- `grid` - `SUDOKUGRID` containing the grid information
- `unknown` - The character used for a cell with an unknown value. The default is `' '` (space).
- `impossible` - The character used when a cell has no remaining possibilities. The default is `'x'`

This function prints the grid to the `ost` output stream in a neat format.

- each cell has one space before its value and one space after
- each 3×3 subgrid will be divided by vertical lines drawn with `|` and horizontal lines drawn with `-`.

If a cell is known, its known value should be displayed. If it's unknown, the `unknown` character should be printed, and if there are no remaining possibilities, display the `impossible` character.

```
with unknown = ' '                  with unknown = '?'
 5  3    |    7    |                 5  3  ? | ?  7  ? | ?  ?  ?
 6       | 1  9  5 |                 6  ?  ? | 1  9  5 | ?  ?  ?
    9  8 |         |    6            ?  9  8 | ?  ?  ? | ?  6  ?
---------|---------|---------       ---------|---------|---------
 8       |    6    |       3         8  ?  ? | ?  6  ? | ?  ?  3
 4       | 8     3 |       1         4  ?  ? | 8  ?  3 | ?  ?  1
 7       |    2    |       6         7  ?  ? | ?  2  ? | ?  ?  6
---------|---------|---------       ---------|---------|---------
    6    |         | 2  8            ?  6  ? | ?  ?  ? | 2  8  ?
         | 4  1  9 |       5         ?  ?  ? | 4  1  9 | ?  ?  5
         |    8    |    7  9         ?  ?  ? | ?  8  ? | ?  7  9
```

NOTES

Your work should be done in `grid.h`, `sudoku.cc`, and `iniparse.cc`. Do not alter the other files that were provided with the assignment.

You should feel free to create whatever files you want to test things locally, including writing your own simple programs to test small parts on their own. I actually *encourage* you to write unit test programs for yourself, but they should not be a part of your submission.

TESTING

There are a number of testing programs included. These will be used to evaluate the functionality of your implementations of the required functions. Typing `make` will attempt to compile them all, and will succeed to the degree it can with whatever you have implemented at that point.

The table below has a list of the tests available, and they are shown in order from least complex to most complex. It will be easier for you if you work on them in order.

Order	Test	Purpose
1	`test1`	Interactive test for the functions that add/remove/change things in an `INI_CONFIG`.
2	`test2`	Loads an INI file into an `INI_CONFIG`. Prints out all of the keys in the `INI_CONFIG`.
3	`test3`	Loads a Sudoku grid from a file. Prints it out immediately.
4	`test4`	Tests portions of the Sudoku solver without loading in a Sudoku file.
5	`test5`	Loads a Sudoku grid from a file. Prints it before and after solving.
6	`test6`	Loads a Sudoku grid from a file. Reads some parameters from an INI file that control its behavior.

The expected output using default values is contained in the `*.refout` files in the `output/` directory to allow you to test locally. The tests used for grading on the Grade-o-Matic may use other parameters, and any differences in the output of your program versus the reference will be reported.

HOW TO SUBMIT

Submission will be done through the NIU Grade-o-Matic Autograder.

GRADING CONSIDERATIONS

- Does your implementation work? Does it compile? How does the output compare to the reference program when given the same input?
- Did you indent your code?
  - Indentation aids in the readability of source code, and if you're not indenting your code blocks, the grader will legitimately dislike you for it. I'm authorizing them to mark you off if you subject them to reading that.
- Did you document your code?
  - You need a docbox at the top of every one of the files you're required to change including:
    - Your name
    - Your zid
    - Your course section
    - A description of what the program does
  - You should add a docbox for every function that you implement, explaining what it does and what each parameter is for.
  - Add other comments inside your code blocks describing what you're doing and why.
  - The use of `doxygen` style comments is encouraged, but not required.