Number Grid

A two-dimensional grid of values can be represented in several different ways in software. One way is to use a one-dimensional array and use function to translate from two-dimensional coordinates to the one-dimensional index. This is like the way you have been storing three-dimensional data in the PPM class.

Assignment

In this assignment you will create a class to store and manage a two dimensional grid of integers. The number grid will use a height and width to manage the dimensions of the data. When a value is read or written, a row and a column must be specified to uniquely identify the value. Values may only be in the range 0 through a maximum configured value. The maximum configured value can be any non-negative integer.

You will also extend the ppm_menu program to add a few new commands.

The new commands required are:

- grid: Configure a grid.
- grid-set: Set a single value in the grid.
- grid-apply: Use the grid values to set colors in the output image.

Potential Session

```
# To run all of the commands from a script, throwing away the prompts
$ ./ppm_menu < ppm_menu_assignment_07_sample_session_grid.txt >> /dev/null
$ ls -l *.ppm
  -rw-r--r-- 1 cgl cgl 236 Feb 27 07:47 sample-grid-image.ppm
```

Programming Requirements

Below, the functions and methods may have a symbol CG? before them, where ? is a number. This indicates which Code Grinder step requires the function or method to be implemented.

Create **NumberGrid.h**

The **NumberGrid** class must store the following information:

- height of the grid
- width of the grid
- maximum allowed value in the grid
- a std::vector of integers.

The following methods must be created in the **NumberGrid** class.

- CG1 NumberGrid();
- CG1 NumberGrid(const int& height, const int& width);
- CG1 int getHeight() const;
- CG1 int getWidth() const;
- CG1 int getMaxNumber() const;
- CG1 void setGridSize(const int& height, const int& width);
- CG1 void setMaxNumber(const int& number);
- CG1 const std::vector<int>& getNumbers() const;
- CG1 int index(const int& row, const int& column) const;
- CG1 bool indexValid(const int& row, const int& column) const;
- CG1 bool numberValid(const int& number) const;
- CG1 int getNumber(const int& row, const int& column) const;
- CG1 void setNumber(const int& row, const int& column, const int& number);
- CG1 void setPPM(PPM& ppm) const;

Create **NumberGrid.cpp**

The following methods must be implemented for the **NumberGrid** class.

- **CG1 NumberGrid();** Initializes the grid to a height of 300, width of 400, max number of 255, and fills the grid with 0s.
- **CG1 NumberGrid( const int& height, const int& width );** Initializes the grid to the height specified, width specified, max number of 255, and fills the grid with 0s.
- **CG1 int getHeight( ) const;** Returns the height of the grid.
- **CG1 int getWidth( ) const;** Returns the width of the grid.
- **CG1 int getMaxNumber( ) const;** Returns the maximum number allowed in the grid.
- **CG1 void setGridSize( const int& height, const int& width );** Sets the height and width of the grid, and resizes the grid storage correctly. Only makes any of these changes if the height and width are both at least 2. The state of the grid values after the resize is undefined.
- **CG1 void setMaxNumber( const int& number );** Change the maximum value allowed in the grid. Only make changes if the new maximum allowed value is at least 0. The state of grid values that are larger than the new maximum allowed value is undefined.
- **CG1 const std::vector< int >& getNumbers( ) const;** Returns a reference to the std::vector of grid values.
- **CG1 int index( const int& row, const int& column ) const;** Returns the index in the grid value vector calculated from the formula: row times width plus column.
- **CG1 bool isValid( const int& row, const int& column ) const;** Returns true if row and column are both within the range of the grid. Otherwise returns false.
- **CG1 bool numberValid( const int& number );** Returns true if the number is non-negative and is no larger than the maximum allowed value.
- **CG1 int getNumber( const int& row, const int& column );** Returns a number from the grid, at the position specified by the row and column. If the position is not valid, returns -1.
- **CG1 void setNumber( const int& row, const int& column, const int& number );** Sets a number in the grid, at the position specified by the row and column. The value is specified by the number parameter. Only makes a change if the position is valid and the number is valid.
- **CG1 void setPPM( PPM& ppm );** Configures the meta data of the PPM object so that the height and width match that of the number grid. Sets the maximum color value to 63. Finally, for each pixel in the PPM object, sets the color based on the following table:

<table>
<thead>
<tr>
<th>Number in Grid</th>
<th>Color (R, G, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>maximum value</td>
<td>(63, 31, 31)</td>
</tr>
<tr>
<td>number % 8 == 0</td>
<td>(63, 63, 63)</td>
</tr>
<tr>
<td>number % 8 == 1</td>
<td>(63, 31, 31)</td>
</tr>
<tr>
<td>number % 8 == 2</td>
<td>(63, 63, 31)</td>
</tr>
<tr>
<td>number % 8 == 3</td>
<td>(31, 63, 31)</td>
</tr>
<tr>
<td>number % 8 == 4</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>number % 8 == 5</td>
<td>(31, 63, 63)</td>
</tr>
<tr>
<td>number % 8 == 6</td>
<td>(31, 31, 63)</td>
</tr>
<tr>
<td>number % 8 == 7</td>
<td>(63, 31, 63)</td>
</tr>
</tbody>
</table>

**Update image_menu.h**

Add the following function declarations to the file.

- **CG2 void configureGrid( std::istream& is, std::ostream& os, NumberGrid& grid );**
- **CG2 void setGrid( std::istream& is, std::ostream& os, NumberGrid& grid );**
- **CG2 void applyGrid( std::istream& is, std::ostream& os, NumberGrid& grid, PPM& dst );**

Modify the function declarations:

- **CG2 void takeAction( std::istream& is, std::ostream& os, const std::string& choice, PPM& input_image1, PPM& input_image2, PPM& output_image, NumberGrid& grid );**

**Update image_menu.cpp**

This file must include the implementations for the new functions declared in image_menu.h.

- **CG2 void configureGrid( std::istream& is, std::ostream& os, NumberGrid& grid );** Prompt the user for integers “Grid Height? ”, “Grid Width? ”, and “Grid Max Value? ”. Use them to configure the grid.
- **CG2 void setGrid( std::istream& is, std::ostream& os, NumberGrid& grid );** Prompt the user for integers “Grid Row? ”, “Grid Column? ”, and “Grid Value? ”. Use them to set a number in the grid.
- **CG2 void applyGrid( std::istream& is, std::ostream& os, NumberGrid& grid, PPM& dst );** Set the PPM
The following functions will require updates to their functionality and/or declarations.

- CG2 `void showMenu(std::ostream& os);` Add to the menu to include the following messages: “grid), Configure the grid.”, “grid-set) Set a single value in the grid.”, “grid-apply) Use the grid values to set colors in the output image.”.
- CG2 `void takeAction(std::istream& is, std::ostream& os, const std::string& choice, PPM& input_image1, PPM& input_image2, PPM& output_image, NumberGrid& grid);` Add to the recognized commands to recognize the new actions in the menu, and take the correct action.
- CG2 `int imageMenu(std::istream& is, std::ostream& os);` Add a declaration of a `NumberGrid` object that is passed to `takeAction`.

**Update ppm_menu.cpp**

No changes are required for `ppm_menu.cpp`.

**Update Makefile**

This file must include the rules to build the program `ppm_menu`. A developer must be able to use the command `make` to compile all necessary files and link them to the executable program `ppm_menu`. Additionally, add the `clean` target that has no dependencies, but will remove any `.o` files and `ppm_menu`.

**Build Requirements**

- `make` must build the complete program named `ppm_menu`
- `make ppm_menu` must build the complete program named `ppm_menu`
- `make ppm_menu.o` must compile `ppm_menu.cpp`
- `make image_menu.o` must compile `image_menu.cpp`
- `make PPM.o` must compile `PPM.cpp`
- `make NumberGrid.o` must compile `NumberGrid.cpp`
- `make clean` must remove all `.o` files and `ppm_menu`

**Additional Documentation**

- C++ Reference
- Examples from class
- Sample Session Input File

**Sample PPM Images**

- Sample Output

**Show Off Your Work**

To receive credit for this assignment, you must

- complete the unit tests available in CodeGrinder
- zip the source code (.cpp and .h files) and the Makefile and upload to the Canvas submission system
- use git to add, commit and push your solution to your repository for this class.

Additionally, the program must build, run and give correct output.

**Extra Challenges (Not Required)**

- Create functions that assign numbers to many grid locations at the same time. For example, you could make boxes, circles and diamonds in the grid.