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 a function to translate from the 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 integer in the range 0 to $2^{31} - 1$.

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

The new commands required are:

- `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.

Programming Requirements

Declare the NumberGrid class in the header file, and implement its methods in the implementation file. Descriptions of the data members and methods follow. This class will be used to store a rectangular grid of numbers. The numbers will be used to select colors to assign to pixels in an image.

Create NumberGrid.{h,cpp}

Data Members:

- `int`: The height of the grid.
- `int`: The width of the grid.
- `int`: The maximum value allowed in the grid.
- `std::vector<int>`: The grid numbers.

Methods:

- `NumberGrid()`: Initializes the grid to a height of 300, width of 400, max number of 255, and fills the grid with 0s.
- `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.
- `int getHeight( ) const`: Returns the height of the grid.
- `int getWidth( ) const`: Returns the width of the grid.
- `int getMaxNumber( ) const`: Returns the maximum number allowed in the grid.
- `void setGridSize( const int& height, const int& width )`: Sets the height and width of the grid, and resizes the grid storage vector 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.
- `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.
- `const std::vector<int>& getNumbers( ) const`: Returns a reference to the std::vector of grid values.
- `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.
- `bool indexValid( const int& row, const int& column ) const`: Returns true if row and column are both within the range of the grid. Otherwise returns false.
- `bool numberValid( const int& number ) const`: Returns true if the number is non-negative and is no larger than the maximum allowed value.
- `int getNumber( const int& row, const int& column ) const`: Returns a number from the grid, at the position specified by the row and column. If the position is not valid, returns -1.
- `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.

- \texttt{void setPPM(PPM & ppm) const;} 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 of the PPM to 63. Finally, for each pixel in the PPM object, sets the color based on the table below.

<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 allowed in the grid</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, 31, 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>

**Updates to \{ActionData.h, cpp\}**

Additional Data Members:

- \texttt{NumberGrid \*} A number grid pointer.

Updated Methods:

- \texttt{ActionData(std::istream\& is, std::ostream\& os)} Needs to initialize the number grid pointer data member to \texttt{0} (the null pointer).

Additional Methods:

- \texttt{~ActionData();} The destructor. Must \texttt{delete} the number grid pointer, but only if the number grid pointer is \texttt{0}.
- \texttt{NumberGrid\& getGrid();} Returns the dereferenced number grid pointer. For example, if the data member was named \texttt{fred}, this method would return \texttt{\*fred}, or \texttt{\*\(\texttt{this}\rightarrow\texttt{fred}\)}.
- \texttt{void setGrid(NumberGrid \*grid);} If the data member number grid pointer is \texttt{0}, \texttt{delete} it. Then assign the data member number grid pointer to the parameter \texttt{grid}. Note this is copying pointers, not copying the contents pointed to.

**Add New Functions to \{image_menu.h, cpp\}**

The follow functions must be declared and implemented.

- \texttt{void configureGrid(ActionData\& action_data);} Prompt the user for integers “Grid Height? “, “Grid Width? “, and “Grid Max Value? “. Use them to configure the NumberGrid object in the ActionData.
- \texttt{void setGrid(ActionData\& action_data);} Prompt the user for integers “Grid Row? “, “Grid Column? “, and “Grid Value? “. Use them to set a number in the NumberGrid object of ActionData.
- \texttt{void applyGrid(ActionData\& action_data);} Configure the output image using the number grid.

The following functions will require updates to their implementations.

- \texttt{void configureMenu( MenuData\& menu_data )} add the new actions with the names and descriptions listed below.
- \texttt{int imageMenu(std::istream\& is, std::ostream\& os)} After the \{ActionData\} object is created, set the number grid of the action data object to be the pointer to a \texttt{NumberGrid} object allocated from the heap. (Think \texttt{new NumberGrid}.)

**Table of New Commands**

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid</td>
<td>configureGrid</td>
<td>Configure the grid.</td>
</tr>
<tr>
<td>grid-set</td>
<td>setGrid</td>
<td>Set a single value in the grid.</td>
</tr>
<tr>
<td>grid-apply</td>
<td>applyGrid</td>
<td>Use the grid values to set colors in the output image.</td>
</tr>
</tbody>
</table>
Build Requirements

- `make` must build the complete programs named `ppm_menu`, `image_file`, `ascii_image`, and `questions_3`.
- `make` must have the `clean` target.

Additional Documentation

- [C++ Reference](#)
- [Examples from class](#)

Show Off Your Work

To receive credit for this assignment, you must

- complete the unit tests available in CodeGrinder
- 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, diamonds, or jack-o-lantern faces in the grid.