CS 3005: Programming in C++

Paint By Numbers (NumberGrid)

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

This assignment will extend the previous menu based program to allow the user to “paint by numbers”. The user will store a single number per location in a grid. Your program will create an image with the same height and width as the grid, and translate the numbers into colors for the pixels.

For example, every place the user stores the number 3 in the grid will have the same pixel color values in the image.

Our implementation will require 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 (roughly 2 billion).

The `ppm_menu` program will have a few new commands:

- `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

The following files must be updated or created and stored in the `src` directory of your repository.

Create `NumberGrid.{h,cpp}`

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.

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.
- `virtual ~NumberGrid();` This destructor only needs an empty block of code. But, it must exist.
- `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.
- `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.

Number in Grid	Color (R, G, B)
0	(0, 0, 0)
maximum value allowed in the grid	(63, 31, 31)
number % 8 == 0	(63, 63, 63)
number % 8 == 1	(63, 31, 31)
number % 8 == 2	(63, 63, 31)
number % 8 == 3	(31, 63, 31)
number % 8 == 4	(0, 0, 0)
number % 8 == 5	(31, 63, 63)
number % 8 == 6	(31, 31, 63)
number % 8 == 7	(63, 31, 63)

Updates to `ActionData.{h,cpp}`

Additional Data Members:

- `NumberGrid *` A number grid pointer.

Updated Methods:

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

Additional Methods:

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

Update `image_menu.h` and `image_drawing.cpp`

The follow functions must be declared and implemented.

- `void configureGrid(ActionData& action_data);` Prompt the user for integers “Grid Height? “, “Grid Width? “, and “Grid Max Value? “. Use them to configure the existing NumberGrid object in the ActionData. Does not create a new NumberGrid object.
- `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.
- `void applyGrid(ActionData& action_data);` Configure the output image using the number grid.

Update `controllers.cpp`

The following functions will require updates to their implementations.

- `void configureMenu( MenuData& menu_data )` add the new actions with the names and descriptions listed below.
- `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 `NumberGrid` object allocated from the heap. (Think `new NumberGrid`.)

Table of New Commands

Command Name	Function Name	Description
grid	configureGrid	Configure the grid.
grid-set	setGrid	Set a single value in the grid.
grid-apply	applyGrid	Use the grid values to set colors in the output image.

Update `Makefile`

The following commands should work correctly.

- `make hello` - builds the hello program
- `make questions_3` - builds the questions_3 program
- `make ascii_image` - builds the ascii_image program
- `make image_file` - builds the image_file program
- `make ppm_menu` - builds the image_file program
- `make all` - builds all programs
- `make` - builds all programs (same as `make all`)
- `make clean` - removes all .o files, and all executable programs

Additional Documentation

- C++ Reference
- Examples from class

Show Off Your Work

To receive credit for this assignment, you must

- use git to add, commit and push your solution to your repository for this class.
- successfully pass all unit tests and acceptance tests

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.