CMSC 202 Spring 2020

Project 1 – Fifteen Game

Assignment: Project 1 – Fifteen Game

Due Date: Tuesday, February 25th at 8:59pm

Value: 80 points

1. Overview

In this project, you will:

- Practice basic C++ syntax including branching structures
- Write a program that calls multiple functions
- Manage a two-dimensional array
- Use simple file input/output

2. Background

The fifteen (15 or 16 puzzle) is a common sliding puzzle that you may remember from your youth. It involves a frame of numbered square tiles in random order with one tile missing. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

Figure 1. Solved 15 Puzzle

For this effort, we are going to create a game board that the user can manipulate in order to get a board that is out of order into a correct order. Additionally, the game will indicate when the game is “solved” by putting the tiles into the correct location.

The game board will be represented by a two-dimensional grid. The actual implementation of how you want to create the grid is up to your design decisions, but you must meet all requirements listed below for full points.

The user will be able to "slide" any tile to an empty location. If the user tries to slide the tile into a location that is already populated or off of the board, the system will indicate that it is not possible. While we can make the grid of any size, we will start with a standard 4 by 4.

		Columns			
		0	**1**	**2**	**3**
Rows	**0**	0,0	0,1	0,2	0,3
	1	1,0	1,1	1,2	1,3
	2	2,0	2,1	2,2	2,3
	3	3,0	3,1	3,2	3,3

Figure 2. Sample Grid

3. Assignment Description

Your assignment is to develop the fifteen game (using text) where the user tries to put each of the tiles into order. On each turn, the user can slide any tile adjacent to a blank into the blank location.

The game continues until all fifteen tiles are in order as shown in figure 1 above. It should then prompt the user for another game.

4. Requirements:

This is a list of the requirements of this application. For this project, it is up to you exactly how you want to implement it. For you to earn all the points, however, you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standards (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. These are the only libraries that you are allowed to use in this project **`<iostream>`, `<ctime>`, `<fstream>`, `<iomanip>`, `<cmath>`, and**

<cstdlib>. You may NOT use **<string>**. You should only use **namespace std**.

- You must use a variety of functions (at least 5) including passing parameters to those functions and returning information from those functions. At least one time, an array (of any size) must be passed to a function (although you may do this more than once).
- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.
- You must use at least one multi-dimensional array in this project.
- When the game is initially started, it should prompt the user if they want to load the "default" puzzle or a puzzle from a file. The default puzzle which can be used for initial testing should look like this:

		Columns			
		0	**1**	**2**	**3**
Rows	**0**	1	2	3	4
	1	5	6	7	8
	2	9	10	11	12
	3	13	14	0	15

Figure 3. Default Puzzle

- Input files start in row 0 and iterate through the columns then move to row 1 and iterate through the columns all the way to row 4. Each line will have data for exactly one cell. There are three test puzzle files (all are solvable) named **proj1_test1.txt**, **proj1_test2.txt**, and **proj1_test3.txt**. We may use different test files and the name of the file should not appear in your code at all.

		Columns			
		0	**1**	**2**	**3**
Rows	**0**	1	2	3	4
	1	5	6	7	8
	2	9	10	11	12
	3	13	14	15	16

Figure 4. Load Order from File

- On a player's turn, the player enters a row and column number. The row and column must start at 1 (unlike the values in the array itself).
- The board will show each of the integers (1-15) or a 0 (representing the blank).
- A tile may only slide into a blank. If a tile is attempted to be slid into a space that already has a value, it will not move there. Additionally, if the tile would leave the board, it will not move either. In both cases, the user should be notified of the illegal move.
- After each slide, the game should check to see if all 15 tiles are in order and the blank is in the bottom right hand corner. If the win condition is met, the game should indicate that the game was won and to re-prompt the user to play again.
- A solved board looks like this:

		Columns			
		0	**1**	**2**	**3**
Rows	**0**	1	2	3	4
	1	5	6	7	8
	2	9	10	11	12
	3	13	14	15	0

Figure 5. Solved 15 Puzzle

- Exit and include a thank you message for the user.
- Specific coding requirements include:
  - Must use at least 5 different functions.
  - Must be able to read a puzzle in the form of a file
  - Must use at least one multidimensional array
  - Must pass at least one array to a function
  - Must not use any global variables (constants are good!)
  - Must use at least one switch statement.
  - Must use input validation (assume the data is the correct type).
  - Must use at least one do..while loop.
  - Must use constants as needed.

5. Recommendations

You are free to implement this with your own functions. While not required, these are some functions that you may want to include:

- Start the Game – Sets up the board based on a file or makes the default scenario (with just two out of order for initial testing).
- Select Tile – user chooses the row and column of the tile to slide into the blank.
- Swap Tile – swaps the value of the chosen tile and the blank
- Check Direction – After the user chooses a tile, checks to see if any of the adjacent tiles are empty. If so, swaps them.
- Print the Board – used to display the entire board formatted with the row and columns visible.

6. Sample Input and Output

For this project, there are three sample files: **`proj1_test1.txt`**, **`proj1_test2.txt`**, and **`proj1_test3.txt`** and they can be copied from my directory to yours.

First navigate to your project 1 directory. Then run this command:

`cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj1/* .`

Note: the line above goes asterisk, space, then period. This should copy all three files into your working directory.

In the sample output below, user input is colored blue for clarity. After compiling and running proj1, here is a sample run using the default board. As you can see, it can be solved with just one move (there is an example of what happens when you choose the blank itself).

```
Welcome to the Fifteen Game
Would you like to load a board?
1. Yes
2. No
2
Loading default
```

```
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|   14|    0|   15|
What is the row (left) and column (top) to slide?
4
3
You chose the blank
That tile cannot slide
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|   14|    0|   15|
No winner
What is the row (left) and column (top) to slide?
4
4
The tile slid left
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|   14|   15|    0|
Congrats you've won!
Play again? (y/n)n
Thank you for playing the game of fifteen!
```

Here is the **`proj1_test2.txt`** run.

```
Welcome to the Fifteen Game
Would you like to load a board?
1. Yes
```

```
2. No
1
What is the file name?
proj1_test2.txt
       1     2     3     4
   1|    0|    2|    3|    4|
   2|    1|    6|    7|    8|
   3|    5|   10|   11|   12|
   4|    9|   13|   14|   15|
What is the row (left) and column (top) to slide?
2
1
The tile slid up
       1     2     3     4
   1|    1|    2|    3|    4|
   2|    0|    6|    7|    8|
   3|    5|   10|   11|   12|
   4|    9|   13|   14|   15|
No winner
What is the row (left) and column (top) to slide?
3
1
The tile slid up
       1     2     3     4
   1|    1|    2|    3|    4|
   2|    5|    6|    7|    8|
   3|    0|   10|   11|   12|
   4|    9|   13|   14|   15|
```

```
No winner
What is the row (left) and column (top) to slide?
4
1
The tile slid up
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|   0|   13|   14|   15|
No winner
What is the row (left) and column (top) to slide?
4
2
The tile slid left
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|    0|   14|   15|
No winner
What is the row (left) and column (top) to slide?
4
3
The tile slid left
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
```

```
   3|   9|   10|   11|   12|
   4|  13|   14|    0|   15|
No winner
What is the row (left) and column (top) to slide?
4
4
The tile slid left
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|   14|   15|    0|
Congrats you've won!
Play again? (y/n)
n
Thank you for playing the game of fifteen!
```

Here is an example validating row/column entries. It re-prompts until the user enters a valid value.

```
Welcome to the Fifteen Game
Would you like to load a board?
1. Yes
2. No
2
Loading default
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
```

```
   4|  13|   14|    0|   15|
What is the row (left) and column (top) to slide?
0
0
What is the row (left) and column (top) to slide?
5
5
What is the row (left) and column (top) to slide?
4
4
Row: 3 Column: 3
The tile slid left
       1     2     3     4
   1|   1|    2|    3|    4|
   2|   5|    6|    7|    8|
   3|   9|   10|   11|   12|
   4|  13|   14|   15|    0|
Congrats you've won!
Play again? (y/n)
```

7. Compiling and Running

To compile your program, enter the following command at the Linux prompt:

`g++ -Wall proj1.cpp -o proj1` (use this command to show warnings – which should be eliminated before turning your code in)

This command runs the GNU C++ compiler (**`g++`**). The option **`-Wall`** instructs the compiler to be verbose in its production of warning messages; the option **`-o proj1`** (hyphen followed by the letter "o", not the digit zero), instructs the compiler to give the executable program the name **`proj1`**. If the program compiles correctly, the executable file **`proj1`** will be created in the current directory. Your project files should have no warnings or errors when turned in.

At the Linux prompt, enter the command **`./proj1`** to run your program. It should look like the sample output provided above.

8. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

Test your code! Make sure that every scenario works. Slide your tiles into each of the corners and make sure they work. Solve a couple of different puzzles.

For this project, you should submit these files to the **`proj1`** subdirectory:

`proj1.cpp` — should include your implementations of the required functions.

Submitting your project has two steps:

1. Set up a symbolic link in your home directory to your submission folder. Execute these commands:
   a. **`cd ~`**
   b. For the next command, copy it exactly – you should not need to modify it at all (**`$USER`** will automatically populate your user name on GL). Also, notice the space after **`$USER`** and before **`~/cs202proj`**.

      ```
      ln -s /afs/umbc.edu/users/j/d/jdixon/pub/cmsc202/$USER ~/cs202proj
      ```

   c. To check that the symbolic link was built successfully, you can type:
      i. **`ls ~/cs202proj`**
      ii. **`It should list proj1, proj1-late1, proj1-late2 through proj5-late2`**

2. Copy the project files into your proj1 folder. Execute these commands:

   a. cd to your project 1 folder. An example might be:

      `cd ~/202/projects/proj1`

   b. **`cp proj1.cpp ~/cs202proj/proj1`**

You can check to make sure that your files were successfully copied over to the submission directory by entering the command

`ls ~/cs202proj/proj1`

You can check that your program compiles and runs in the **`proj1`** directory, but please clean up any **`.o`** and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here.

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj1 folder, it is because the due date has passed. You should be able to see your proj1 files but you can no longer edit or copy the files in to your proj1 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj1-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj1-late2`**
- If it is after 48 hours late, it is too late to be submitted.