CSC 230 – Project #1

Due: Friday, Feb. 23rd, before midnight

Due Date:
11:59pm, Feb 23nd, 2017

Project Details:

Write a program to search words in a matrix (imagine it is a 2-dimenional array). This program will find the words listed in command line and highlight the words with color. On Canvas, there are several matrices for testing. For example, inside 0505matrix file, you will read:

```
5 5
u r a q o
f t c n j
k r h p r
e a v o t
z h g a h
```

Here, the first 5 is the number of rows, the second 5 is the number of columns. If the C++ program name is *words* and we want to search tcnj and go in this matrix. We can type the following command

```
%words tcnj go < 0505matrix
```

```
u r a q o
f t c n j
k r h p r
e a v o t
z h g a h
```

As you can see, tcnj and go are red colored.

If you want to search this project is hard inside file 1520matrix. You can type:

```
%words this project is hard < 1520matrix
```

Similarly, to search you enjoy this project inside 2020matrix, you can type:

```
%words you enjoy this project < 2020matrix
```

How to print a letter with color: Whenever we print out some text in terminal, the color of the text itself is called foreground color, the color of the text background is called background color. In the file "colormod.h", we defined a Setting class, which allows you to change the foreground color and background color. In this project, you just need to change the foreground color.

Please note once you change either background or foreground color, the following text will use the same setting until you **reset** it or change the colors again. It is a good idea that you change the color before printing the text, then change the color back to default value after the text print is done.

An example C++ program to output Hello World in Red is listed in the following. In this example, "red" object sets color to red. "def" object sets color to default. In this project, please do NOT modify colormod.h, just use it.

Code:

```
#include "colormod.h" // namespace Color
#include <iostream>
using namespace std;
int main() {
    Color::Setting red(Color::FG_RED);
    Color::Setting def(Color::FG_DEFAULT);
    cout << "This ->" << red << "Hello, world!" << def << "<- is red." << endl;

}
```

Before starting write a code: Think about how a human being can solve this problem. How to search the all possibilities? Do not rush to write code before figuring out the whole idea. Usually, try to figure out the solution of some simplified the case. For this project, you can first think about the situation that it has one search word, and the program searches from one position with one direction. After figuring out the specific situation, try to solve it in a more general situation.

How to search ONE word in the matrix: Your program should scan the whole matrix. If the program is visiting [i][j] element now, it will check whether word match the strings on [i][j] element's left side, right side, upward, downward, and four diagonal directions. If the program finds a match, it will mark a separating matrix to keep the record. This separating matrix can be a Boolean array or char array. Please keep in mind that the input matrix may have multiple matches for one word. Your result must show all the matches.

Should I use two-dimensional array or something else? In the lecture we will compare 2D array with other choices. In general, we strongly suggest you use vector in this project. Vector will make coding way much easier than 2D array. The following is an example how to define two-dimensional vector, which is a replacement of traditional two-dimensional array.

```
void init(vector< vector<char> > &twoD) {
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++)
      twoD[i][j] = -1;
  }
}
```

```
vector< vector<char> > searchMatrix;
searchMatrix.resize(x);
for (int i=0; i<x; i++) {
    searchMatrix[i].resize(y);
}
….
init(searchMatrix);
```

How to search multiple words in the matrix: Once your program can search one word, you can use a for loop to go over each word typed at the command line. Note all the words you type will be saved to argv[]. You can use **argc** to check how many words typed. The words you typed are stored in array, from **argv[1]** to **argv[argc-1]**. The following code shows how to use argc and argv[]. Please note argv is an array, the elements of the array are char pointers. The argv[0] is the program name itself. For example, if you type

./a.out a b c

The output will be

./a.out

a

b

c

```
#include <iostream>
using namespace std;

int main(int arg, char *argv[]){

  for(int i = 0; i < arg; i++)
    cout << argv[i] << endl;

}
```

How to read the matrix into memory: Just like what we do in lab1, you should use cin and nested for loops to read the data.

How many functions do I need? It is up to you. It depends on how you will divide functions among methods. But that does not mean you can/should just define one function, that is main(). If you do so, believe me, it is hard to do debugging. I prefer you define one function that will check whether the word match the string starting from [i][j] at **one** direction (up, down, left,

right, diagonals). If there is match, another method will record this information into a separating matrix.

Do NOT make one method do too many things!!!!!

Anything else I should know about the method that checks match? If you are checking the string starting at [i][j] on right direction, your program will check letter by letter to the right side. Make sure that your program will NOT move over the bounds of the matrix. In other words, the match checking must be always inside the matrix.

Do NOT think about the code. Instead, think about as a human being how to solve it, in a simplified situation. Then think about how to solve a more general case. Repeat this process until you have a solution for the original problem. Write down your solution (algorithm) on paper.

NEVER ever write code before you have a scrap paper with your solution on it. NEVER try to figure out your idea after you start typing. It will save you a lot of time.

Please build your code incrementally. Start coding with almost nothing inside, then add several lines, compile it, check it. If the code looks fine, add several more lines, compile it, check it. Repeat this process until the job is done.

When you decide which part should be implemented first, consider the simplest function of the project. Implement the simple ones first, handle the more complicated ones later. As human being, we prefer to solve simple questions first. The same rule applies to programming.

Never ever write down the whole code, then do the debugging. Remember we build a building piece-by-piece, floor-by-floor. No one builds all the floors of a building at the same time. It won't work!!!