CS 2630: Data Structures

Fall 2021

Program 3 (30 points)

Due: Sunday 11/7 before 11:00pm

Objectives

After this program assignment, students should be able to:

- Create a C++ Windows Form Application
- Familiarize themselves with new programming concepts (GUI elements, etc.) through independent study
- Implement a recursive solution for finding a path through a maze

Introduction

For this programming assignment, you will create a GUI-based maze program. Given a starting point in a maze, you are to determine if there is a way out. You **MUST** do this recursively. You will display all cells "checked" by the recursion.

The input for your maze program will be one of several data files provided to you on Canvas. Keep in mind that some of these files are "good" and some are purposely made "bad" (bad mazes do not adhere to the rules given below). These data files contain a two-dimensional (2D) array describing the maze. The first line of data in the maze file contains the number of columns and rows for the maze. Each subsequent line in the data file contains each row of the maze as a series of characters, according to the following format: `'O'` (open path), `'+'` (dead end or trap), or `'E'` (exit from the maze). More precisely, the data file adheres to the following format:

```
width  height
<maze data, row-wise>
```

For example,

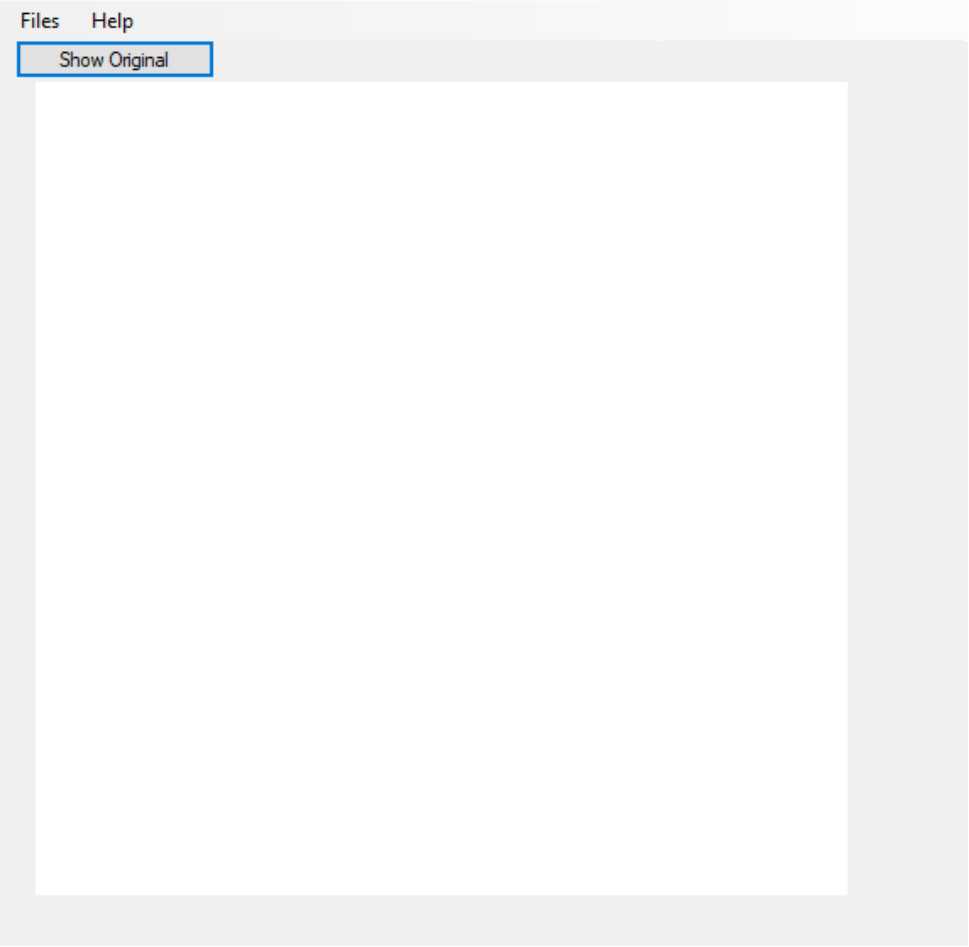
```
10  12
OO+E+OO+++
O++O+O+OOO
OOOOOO+O+O
+++++O++OO
OOO+OOO+O+
O+O+O+++O+
O+O+OOO+OO
++O+++O++O
O+OOOOO++O
O+O++O+OOO
++O++O+OO+
OOO+OOO+O+
```

A player of your game may move vertically or horizontally in any direction that contains an `'O'`; you may not move into a cell with a `'+'`. If you move into the cell with an `'E'`, you are free! You may not move diagonally.

You will use a `panel` and Graphics routines, such as `FillRectangle` to represent the maze.

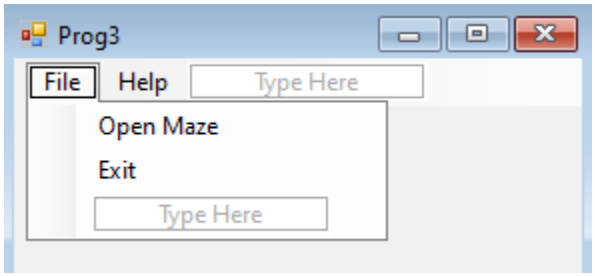
Program Design – Getting Started

You have been given an executable version of Program 3 on Canvas. Download and run the program, the main screen looks like the screenshot shown below.



Experiment with the executable program and the given maps, your version of Program 3 should behave similarly to the executable program.

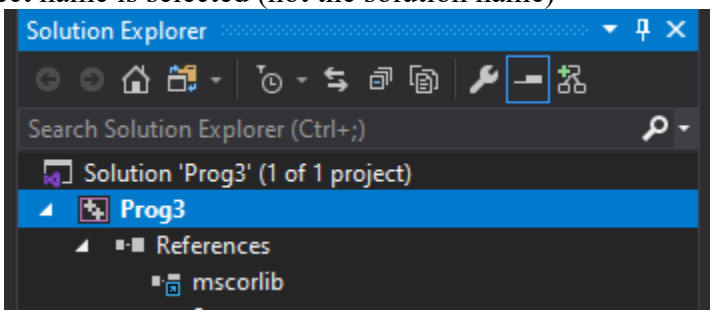
To start, refresh your memory on C++ Windows Form development by reviewing Lab 4. You will create Program 3 according to the tutorial given in Lab 4. After you have a basic Empty CLR project completed, add a `MenuStrip` to the main form. Add the following options to the `MenuStrip`:



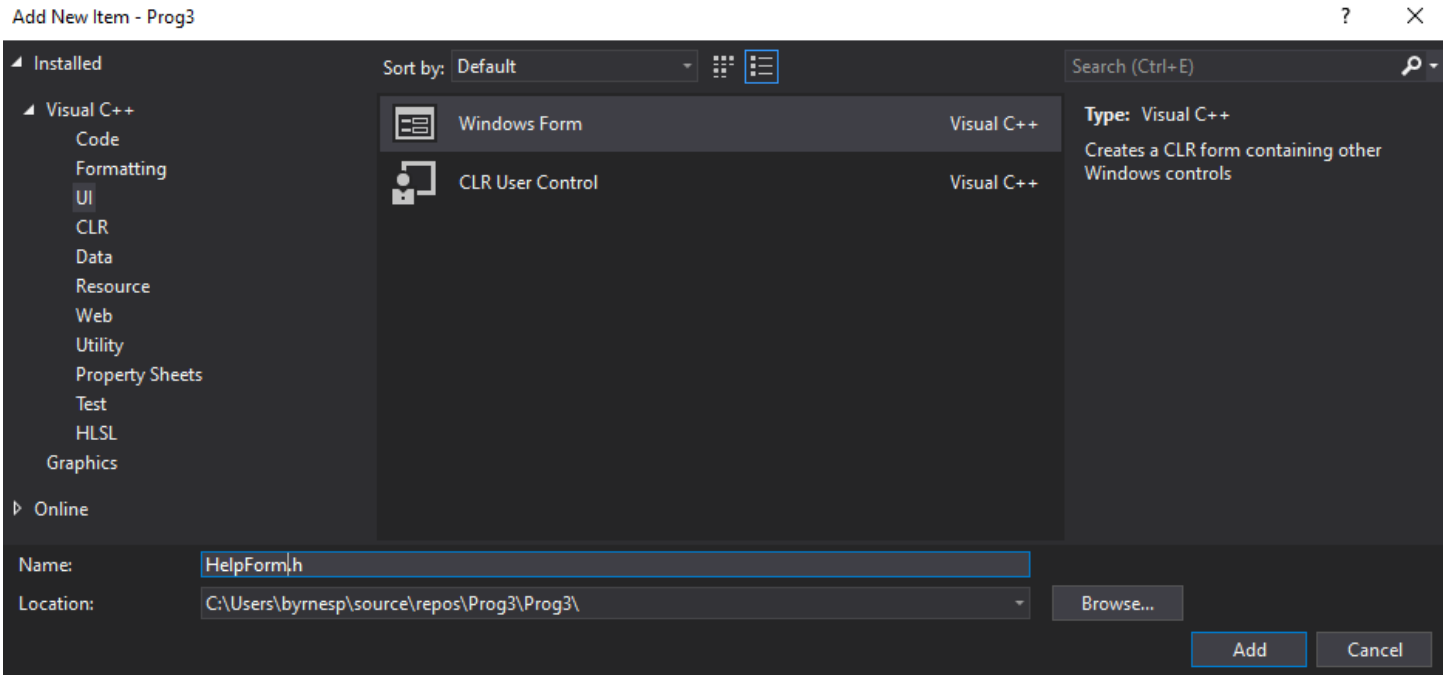
- File
  - Contains two sub-options: “Open Maze” and “Exit.”
- Help
  - When selected, a Help Form is shown to the user containing game instructions

Don't add any event handlers yet to the `MenuStrip` yet, let's add the Help Form together and then get to a good starting point for completing Program 3. Add a new form to your program:

- Make sure your project name is selected (not the solution name)

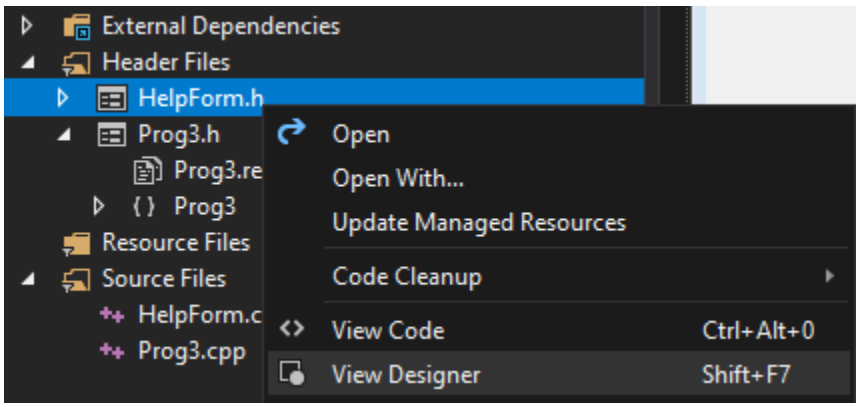


- Now, select Project → Add New Item… → UI → Windows Form
- Call the Windows Form "HelpForm.h"



- Finally, select Add.
- Remember, any time you add a form, you will get an error. Ignore the error, save all your project files, close the solution, and then re-open your project. As discussed in Lab 4, this is a well-known bug in Visual Studio that has not been fixed (nor does it seem to be a high priority for Microsoft to do so).

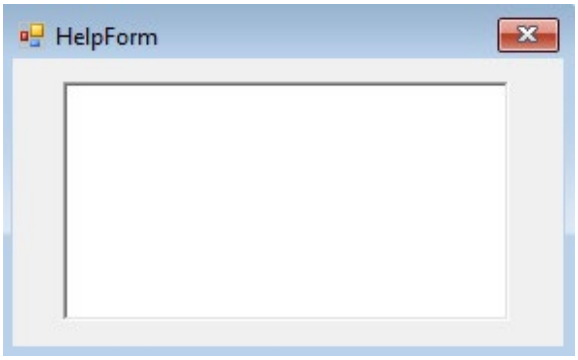
Once your project is re-opened, right-click on `HelpForm.h` and open the View Designer:



Now, select the Help Form and edit its properties to disable the minimize and maximize buttons.

MaximizeBox	False
MinimizeBox	False

Add a `RichTextBox` to the Help Form and make things look neat and symmetrical. Also, find the `ReadOnly` property and set it to "`True`" to prevent users for typing text into the box.



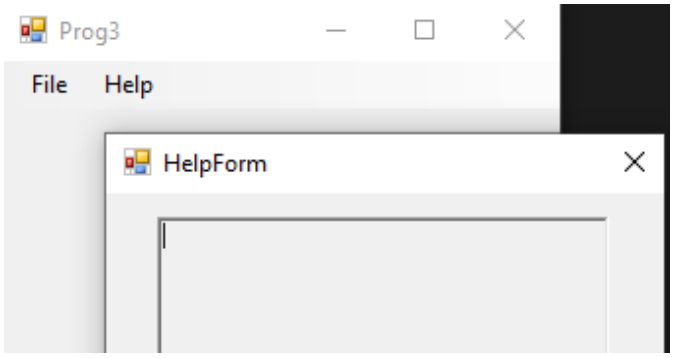
We'll add text to the `RichTextBox` after we add an event handler for handling the "Help" selection on the `MenuStrip`. Go back to your main form and double-click on the word "Help" in your `MenuStrip`. This will add an event handler to your main form that is called when the user selects "Help" on the toolbar. Add the following code to the new event handler:

```
122     private: System::Void helpToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) {
123         HelpForm^ f = gcnew HelpForm(); // display help form
124         f->ShowDialog();
125     }
```

Notice that `HelpForm` is underlined in red. This is because we haven't added `HelpForm.h` to our main form header file yet. Scroll up to the top of your main form code and add the following line:

```
1   #pragma once
2   #include "HelpForm.h"
3
4   namespace Prog3 {
```

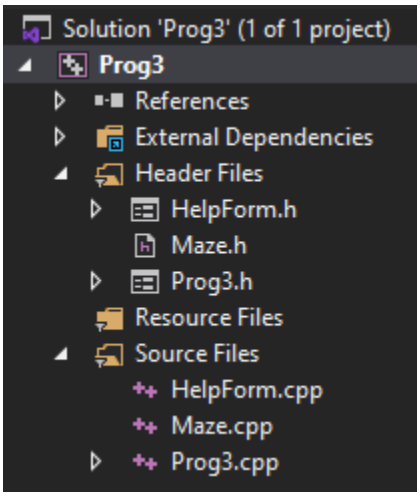
Build and run your project to test whether the Help Form appears when you select "Help" on the toolbar.



Adding the appropriate text to the Help Form is left to you. As a hint, you can add the text to `HelpForm.h`.

Program Design – The Maze Class

For Program 3, we will add a `Maze` class to our CLR project to separate functionality related to the Maze from the basic Windows Form the user interacts with. To do this, add a Maze specification file (`Maze.h`) and implementation file (`Maze.cpp`) to your project. Your Solution Explorer window should look something like the following:



Let's connect the Maze class appropriately to our Windows Form application and then discuss some constraints on the project before leaving the rest of the solution to you. Open `Maze.h` and add the following code to get started:

```
1   #pragma once
2   #include <vcclr.h>  // Needed for gcroot declaration: allows use of GC class in non-GC class
3   #include <fstream>  // needed for file reading of Mazes
4
5   using namespace std;    // this NEEDS to be first so as not to cause conflicts
6   using namespace System;
7   using namespace System::Drawing;
8   using namespace System::Windows::Forms;
9
10  class Maze { };
```

Before we describe the requirements for the `Maze` class, open `Maze.cpp` and add the following code:

```
1   #include "Maze.h"       // Maze class specification
2
```

Now, let's add a pointer for the Maze class in our main form header file, along with a `Panel` for the maze to be drawn on. First, open the code for your main form header file and include the `Maze` class specification file, along with the `fstream` header.

```
1   #pragma once
2   #include "HelpForm.h"
3   #include "Maze.h"
4   #include <fstream>
```

Now, open the main form in View Designer and find a `Panel` in the toolbox. Add the Panel to your form and size it to take up most of the form. You can adjust the size later to polish your design (you can also adjust it in code when you determine the size of the loaded maze).

Once the panel is added to your form, go back to your main form header code and add a `Maze *` as a private member variable to your form.

```
41      private: Maze* maze;
42      private: System::Windows::Forms::MenuStrip^ menuStrip1;
43      private: System::Windows::Forms::ToolStripMenuItem^ fileToolStripMenuItem;
44      private: System::Windows::Forms::ToolStripMenuItem^ openMazeToolStripMenuItem;
45      private: System::Windows::Forms::ToolStripMenuItem^ exitToolStripMenuItem;
46      private: System::Windows::Forms::ToolStripMenuItem^ helpToolStripMenuItem;
47      private: System::Windows::Forms::Panel^ panel1;
48      protected:
```

The `panel1` variable at line `47` is the new `Panel` we just added in the View Designer. We'll use this to draw a loaded maze onto the GUI. Finally, make sure to initialize your `Maze *` to nullptr.

```
20      public:
21          Prog3(void)
22          {
23              InitializeComponent();
24              //
25              //TODO: Add the constructor code here
26              //
27          }
```

```
20      public:
21          Prog3(void)
22          {
23              InitializeComponent();
24              maze = nullptr;
25          }
```

Now that we have the basic program design, use the following specification for your Maze class:

```
class Maze
{
public:
      //---------------------------------------------------------------
      // Maze constructor
      // Reads in the map.
      // Sets valid to true if there are no invalid characters found and
      //    the size of the map is valid
      //---------------------------------------------------------------
      Maze(Panel^ drawingPanel, ifstream& ifs);

      // Maze destructor - clean up dynamically allocated 2D arrays!
      ~Maze();

      //---------------------------------------------------------------
      // Returns true if the maze is valid, false otherwise
      //---------------------------------------------------------------
      bool IsValid() const { return valid; }

      //---------------------------------------------------------------
      // Returns true if the exit of the maze was found, false otherwise
      //---------------------------------------------------------------
      bool IsFree() const { return free; }

      //---------------------------------------------------------------
      // First checks if the mouse clicked a deadend or the exit and
      // returns if either is true.
      // Otherwise, it establishes the starting cell, colors it red,
      // and calls RecSolve with the row and column of the clicked
```

```
    // cell as parameters.
    //-----------------------------------------------------------------
    void Solve(int xPixel, int yPixel);

    //-----------------------------------------------------------------
    // Calls Show with the original maze 2d array as parameter
    //-----------------------------------------------------------------
    void ShowOriginal() { Show(orig); }

    //-----------------------------------------------------------------
    // Calls Show with the solved maze 2d array as parameter
    //-----------------------------------------------------------------
    void ShowSolved() { Show(solved); }

private:
    static const int CELLSIZE = 16;
    static const int MAXSIZE = 30;

    static const char OPEN = 'O';
    static const char DEADEND = '+';
    static const char EXIT = 'E';
    static const char START = 'S';
    static const char VISITED = 'X';

    int width, height;          // width and height of the maze
    bool free;                  // Did RecSolve reach the exit?
    bool valid;                 // Is Maze Valid?

    gcroot<Panel^> panel;       // Panel on which to show the Maze.

    // start with statically allocated arrays!
    char orig[MAXSIZE][MAXSIZE];   // Original Maze Data
    char solved[MAXSIZE][MAXSIZE]; // "Solved" Maze Data

    // Once your program is working - comment the statically
    // allocated arrays above, and un-comment the pointers below.
    // You will need to dynamically allocate the 2D arrays like
    // we discussed in class - keep in mind, once the arrays are
    // correctly allocated, they may be used exactly like the
    // statically allocated arrays above - that means, very little
    // of your working code needs to change, just add dynamic allocation
    // in the right place!
    // char** orig;
    // char** solved;

    //-----------------------------------------------------------------
    // Solves the maze recursively.
    // First, as a base case, it checks if the current cell is the exit.
    // If the cell is not the exit, it then checks if any adjacent cells
    // are the exit if the exit is adjacent, RecSolve is called with
    // the location of the exit as parameters.
    // Otherwise, it checks for adjacent open cells and makes calls
```

```
    // to RecSolve to move to said open cells.
    //-------------------------------------------------------------------
    void RecSolve(int row, int col);

    //-------------------------------------------------------------------
    // Displays the given data as a maze with deadends represented by
    // black squares, open cells by white squares, and the exit by
    // a green square.
    //-------------------------------------------------------------------
    void Show(char cells[][MAXSIZE]);

    // when you are ready to use dynamic allocation of the 2D arrays
    // uncomment this version of Show and comment the one above!
    //void Show(char** cells);

};
```

Other than the changes related to switching from statically allocated 2D arrays to dynamically allocated 2D arrays, you may not modify the specification file. Implement each function and modify the main form to properly interact with the main class when the user clicks the mouse on the maze panel. Figuring out how to properly use the GUI to trigger functionality in your code is part of Program 3. This is non-trivial, start early!

Program Requirements, Additional Information, and Hints

Now that you have a basic project created and the specification for the Maze class, keep the following requirements in mind for Program 3.

1. You must use `Maze.h` without modification (except for the 2D array changes when you are ready!).
    a. However, you may modify `Maze.h` if you add extra credit to your project AND properly document the change and why it was made. You will lose points if you modify the Maze specification without documenting why in your comments with an appropriate explanation and what extra credit functionality these changes provide.

    b. Extra credit will be evaluated by me on a case-by-case basis, don't ask what constitutes extra credit, I will evaluate what I see documented and award points as I deem worthy. If you think of something cool, try and implement it!

2. The implementation for the Maze class function `RecSolve` must be recursive. The recursion is similar to the "flood fill" strategy. Think about it this way:

    You have 4 friends stuck with you in the maze. You tell them that you will check if the present position is the exit; if not, you send each one off in a different direction to find the exit while you stand guard in the present position so that no one returns to this position and checks it again (after all, you already did your part by checking it and you know that it is not the way out).

3. The main form can't be any bigger than `1024 x 768` - it must run on my machine without me changing the graphics setup. I suggest making it only big enough to display the maximum `30 x 30` maze.

4. You must add an `OpenFileDialog` to your main form for the user to browse for a Maze data file when selecting "Open Maze" from the toolbar.

a. Remember, there are several data files on Canvas, some good and some bad. Your program must recognize that a maze data file is not valid and inform the user (Put up a `MessageBox`).
b. Set `InitialDirectory` Property of the Open File Dialog to `"."`.
c. To use a File Open Dialog (e.g., in the File→Open Maze handler):

```
if ( openFileDialog1->ShowDialog() == ::DialogResult::OK )
{
   // Do something
}
```

d. The filename is returned via the `FileName` property of the `OpenFileDialog`.
e. Copy the characters over one by one, using the following pseudocode:

Declare: `char filename[1024];`

Then:

loop through the `openFileDialog1->FileName->Length` characters of `openFileDialog1->FileName`, copying them to `filename`.
Make sure you NULL-terminate `filename`.

Then:

```
ifstream ifs;
ifs.open(filename);
maze = new Maze( panel1, ifs );
ifs.close();
```

5. I expect that you will use .NET help for the GUI components. We will not be covering these in class. Like what you were required to do in CS/SE 2430, when you use class libraries, you need to use the help!

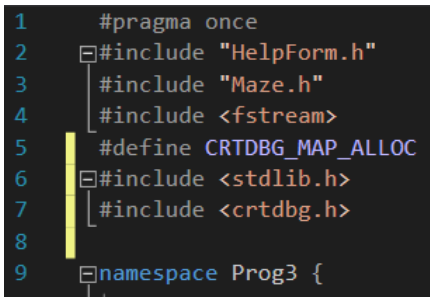
6. The starting cell will be chosen by clicking on it. Your program must have an event handler for `MouseDown` on the main form. In the event handler, you can read the coordinates of the mouse click and pass them to your Maze class.

7. Your program must have the minimum functionality that the sample Prog3.exe on Canvas has.

8. You must have a help screen, that must be a "Form", not a `MessageBox`. If you've followed the instructions so far, then we already created and added this form together.

9. Put the following in your main form's H file, right after the other all other `#include`'s:

```
#define CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```



10. Make an "Exit" event handler for the main form where you:

a. Call `delete` on your `Maze *` (Make sure it's a valid pointer first!)
b. Add the line: `_CrtDumpMemoryLeaks();`
c. Add the line: `this->Close();` (this will close the application)

- Note that you must also `delete` your maze pointer every time you make a new one!!

Hints

1. There are several base cases for the recursion.

2. You can set the size of the panel inside the Maze class use the `Width` and `Height` properties.

3. Remember about `ifstream`. To use it, be sure to:
   ```
   #include <fstream>
   using namespace std;    // Can cause problems if in wrong place
   ```

   If you followed the instructions in this document, then you've already included `<fstream>` and "`using namespace std`" in the right places!

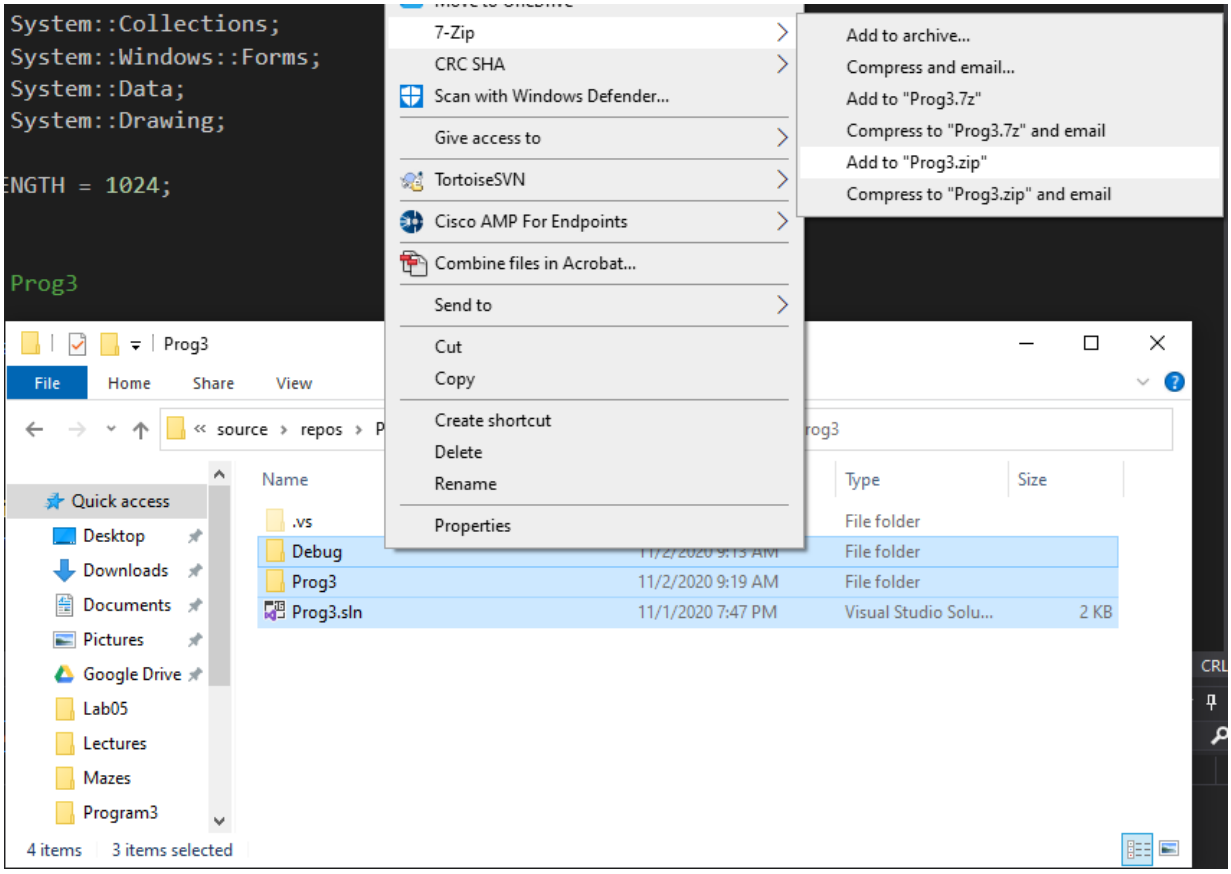
4. You can hide and show .NET components at run time using their `Hide()` and `Show()` methods.

5. I suggest you do all "showing" in the paint methods of the form and panel. Make, set, and use form state variables to do this.

6. To draw, you'll need to create a `Graphics^` object. You can use this object's `FillRectangle` function to draw squares on the maze panel: it needs a font and colored brushes to do this.

Submission

You must submit the entire project folder as a compressed file to Canvas. However, make sure to only add the content of your project folder as follows:



The hidden folder "`.vs`" is **HUGE** and contains information I don't need. **DO NOT add this to your zip file**. For example, the zip'ed file I created above is less than 1MB. If you clean your solution before compressing your project, then your submission should not be much bigger (if at all) than that. **DO NOT submit a 200MB file**, it will take me forever to download student submissions for grading and **you will lose points for doing so**.

When you are sure you have a working version of your program, submit the Final version to Canvas. Any submissions after the due date will be assessed a 25% penalty per day late (-7.5 points / day).

Rubric

Criteria	**Possible Points**
Working Help Form	2
Maze class implementation (besides RecSolve)	5
Recursive RecSolve implementation -Exit found when clicking on Maze (or not found, if no solution)	10
File open dialog implementation – Maze data loaded and properly determined to be valid / invalid	4
Initial drawing of Maze after loading a Maze file	3
Drawing of explored Maze space after clicking on Maze	4
Original Maze is shown after clicking "Show Original" - Or message displayed indicating user needs to load a Maze	2
Documented and working extra credit functionality	+ **?**
Total	30 + **?**