CS 135 - Computer Science I

Design Assignment 9 (DA9-11/28)

Programming Assignment 10 (PA10-12/05)

As specified in your syllabus, you must turn your assignments in by 6:00 pm on the due date specified. If it is turned in late, but prior to 12:00 midnight the day it is due, credit will be reduced by 50% of the earned score. Any laboratories turned in more than 6 hours late will not earn any credit.

Objectives:

1) You will use a design procedure to develop a well-organized and modular program
2) You will use programming strategies and tools learned across the semester
3) You will use file input and output (I/O) to access and process data from a file
4) You will use one- and two-dimensional arrays to read data from a file, store data for processing, and display results of a process, along with the requisite iterating actions
5) You will use and manipulate c-style strings; **string class/object values are not allowed**
6) You will work in a small group situation to develop a somewhat larger-scale program

Tasks:

The Rogue-Like Game

1) You will be creating a game called **`roguelike.cpp`** that represents a very simple version of more completed programs in this genre. You will have extra credit opportunities to expand the scope of the program at your discretion.

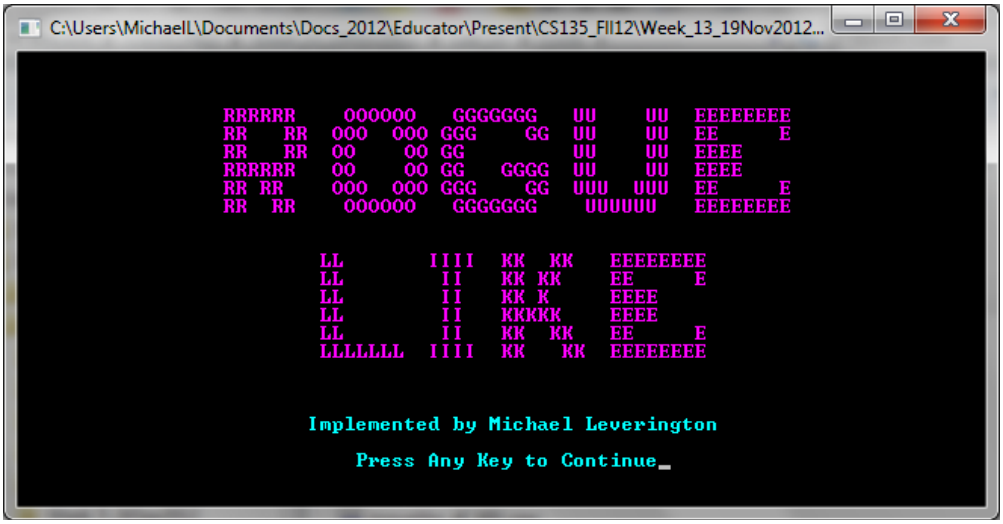
2) You will need to download three files necessary for the layout of the mazes; they are **`highlevel.txt`**, **`midlevel.txt`**, and **`lowlevel.txt`**. These and the example program **`roguelikeEX.exe`** can be found on the WebCampus assignment page as usual. The example program provides both the required components of the program and the extra credit components of the project, which are shown if the 'E' key is pressed at the splash screen. Any other key press will run the program with the basic requirements.

3) As explained later in this document, you will also be working in groups of three. These groups will be randomly chosen as is common in industrial and upper-level academic environments.

4) You will be using the **`formatted_console_io_v18.h`** tools. Any use of **`iostream`** or other tools not found in this header will result in a significant reduction to your program credit.

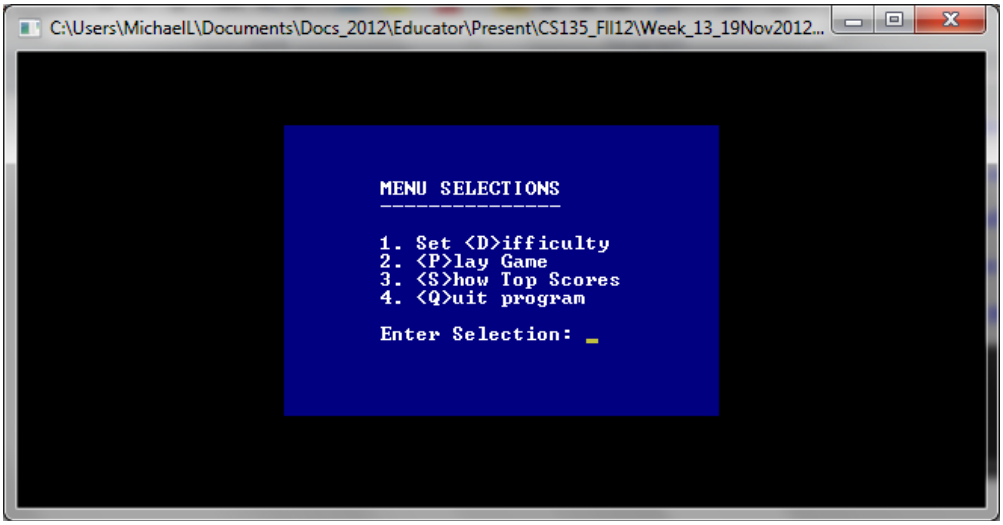
5) You will have slightly more than a week to create the design assignment since part of the assignment period is a holiday. You will then have one week to complete the coding part of the program.

6) The program begins with a splash screen. Your splash screen can be different but it must show something more than just regular text. The example program splash screen shows the name of the game in large letters. It is provided here.

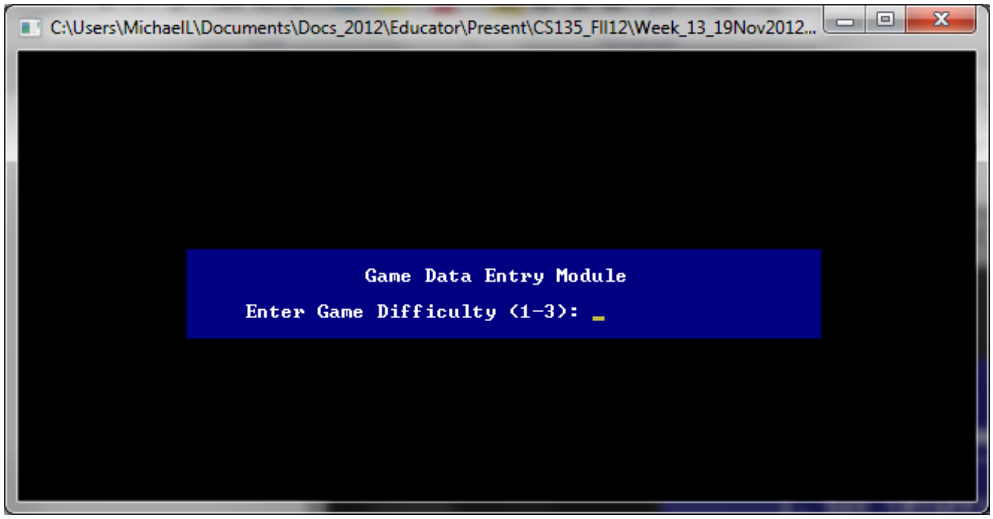


7) Pressing any key will lead to the instruction screen, shown here. Note that the extra credit instruction screen is different, and as mentioned previously in this document, is found by pressing the letter ‘E’ at the splash screen.

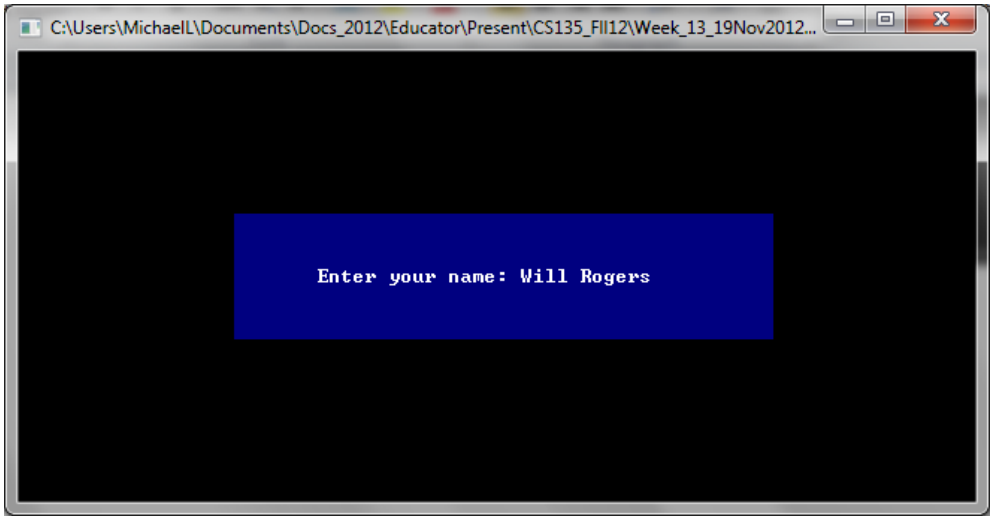
8) Once the instruction screen has been displayed, pressing any key will lead to the program menu. You must use the menu format provided to you in class, and it must be in the main function. Any deviation from this will be cause for credit reduction. The menu is shown here.



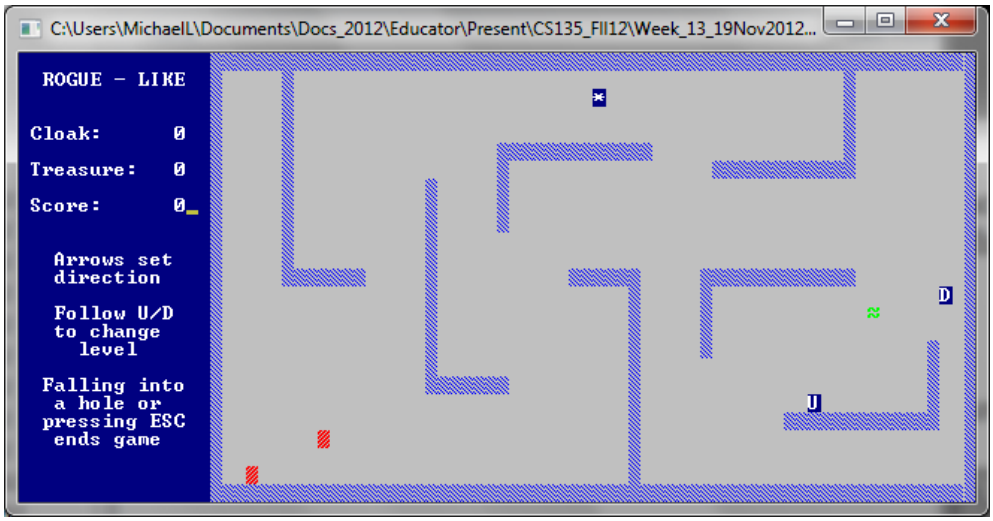
9) The difficulty setting changes the speed of the program. Using the **`waitForInput`** function, the example program settings are 5 for easy difficulty (i.e., user selection 1), 3 for medium difficulty (i.e., user selection 2), and 1 for hard difficulty (i.e., user selection 3). The difficulty setting display is shown here.



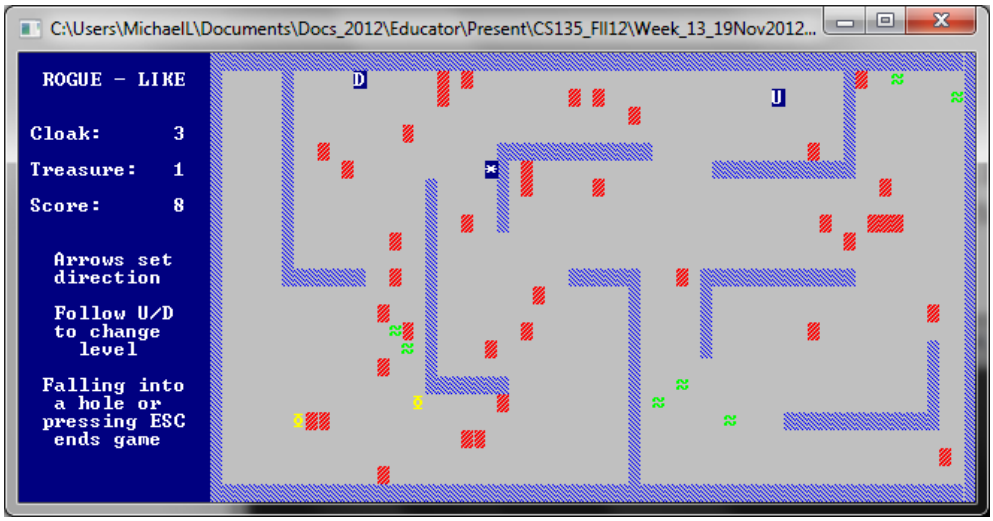
10) When the play game selection is made, the user is prompted for her/his name. This will be used for the score posting later. This operation must be able to take in names with spaces. The name input screen is shown here.



11) Once the name is input, the program starts with the player at the top center of the screen as shown. The objects will pop up randomly as specified later in this document. An example of the playing screen is shown here.



12) Another example of the screen with some points acquired is shown here.



13) The rules of the game are as follows:

a) a red square (ASCII value 178) is a **hole**; these have a 35% chance of appearing on the game board with each cycle*; falling in a hole (without a cloak) ends the game. *A cycle is one complete pass through the game loop. The loop starts, the user input is captured, a switch statement then looks at the user input and implements the appropriate action, and then the loop repeats.

b) a green double tilde (ASCII value 247) is a **cloak**; these have a 10% chance of appearing on the game board with each cycle; having one or more cloaks to throw over holes protects the player from falling into a hole, and keeping a cloak increases your score by 1

c) a yellow phi symbol (ASCII value 232) is **treasure**; this only has a 5% chance of appearing on the game board during each cycle; a treasure increases your score by 5

d) for extra credit, a green omega sign (ASCII value 234) is a **spell**: these also have a 10% chance of appearing on the game board during each cycle; they allow the user to walk through internal walls (ASCII value 176), and keeping a spell increases your score by 1

e) for extra credit, a red double cross (ASCII value 206) is a **monster**; this has a 25% chance of appearing on the game board during each cycle; running into a monster (without a sword) also ends the game

f) for extra credit, a green Y-double-bar character (ASCII value 157) is a **sword**; this has a 10% chance of appearing on the game board during each cycle; having one or more swords in your possession when meeting a monster allows you to slay it (rather than being eaten and stopping the game) and continue playing, and keeping a sword increases your score by 1

g) running into a letter **U** or **D** on the board changes your level to a different playing surface; you can start over on each level and you can change levels as much as you want, unless the holes surround you or the **U** or **D**; note that the U and D are displayed with their own colored background as is the player, which is represented by an asterisk; all three levels must be loaded from the given text files; this will be discussed later in this document; note that the player must always start at the top center location shown

h) all of the above items are randomly placed on the board if their chance percentage allows them to; the chance percentage is simply a random value chosen between 1 and 100 and then tested for equal to or less than the percentage number given; you have a version of this function from a previous program

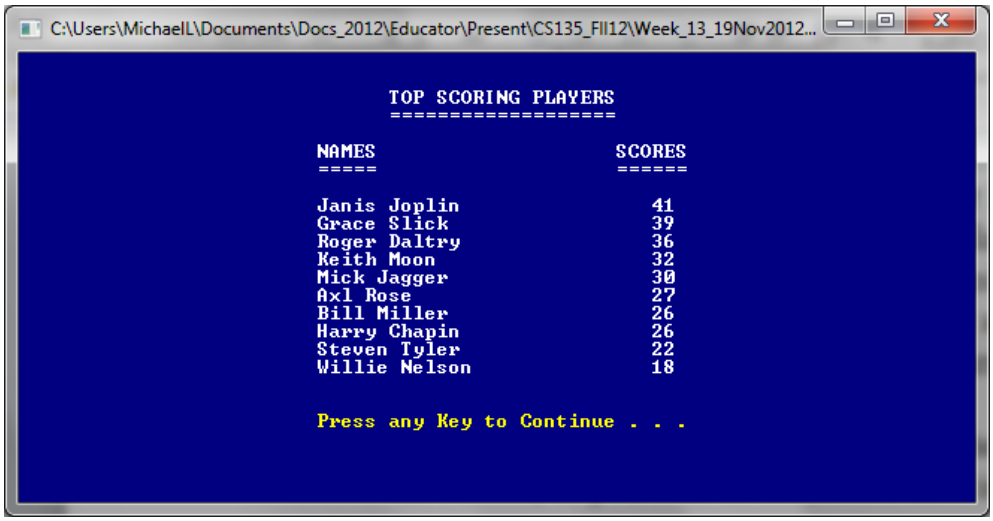
i) the scores are kept on the upper part of the instruction panel, located on the left side of the screen; as you will notice, more data is displayed under the extra credit operation; these scores must be updated real-time meaning they must display the updated scores immediately upon their modification

j) you will also be using a new function called **`printSpecialCharAt`**; it is identical to the **`printCharAt`** function except it takes a **`short`** value representing the extended ASCII set discussed previously in this document; if it appears to be necessary, you can use the **`printSpecialCharAt`** function to display basic ASCII characters but you cannot use **`printCharAt`** to display extended ASCII characters

k) you are strongly urged to play the game several times and take notes before beginning your design process

14) Your program must meet the specifications stated in the previous item, and meet or exceed any other operations found in the non-extra-credit operation of the example game.

15) At any time, the user may check the top ten scores generated from game playing. An example of this screen is shown here.



16) The scoring system must store the name and the data in a file. The scoring system must be able to handle 1) an empty or non-existent file condition, 2) any number of players and their scores, up to ten, and 3) if the list is full, it must drop off the bottom player so that a maximum of ten players is never exceeded.

17) The data files for the levels simply represent several x and y positions in an array where the internal walls (ASCII value 176) are located. The data assumes a 62 (wide) by 23 (height) playing board. You must store these values in a **`short`** array as a **`char`** array would not work for these values. When you do add standard characters to the array, it will store them properly. An example of the data is shown here. Note that the first value is an x location (in the array) and the second value is the associated y location (in the array). This data is found in the middle level data file; it shows that wall objects should be located at locations (5,0), (5,1), (5,2), and (5,3) in the array

```
5 0
5 1
5 2
5 3
```

18) You must use a two-dimensional array to handle all the working data of the program. The indices of the array will not be the same as the on-screen x and y positions, but this issue is easily resolved by using offsets as needed to translate between array location and screen location.

19) For your reference, standard characters have an ASCII value between 32 and 126 for the printable characters and the extended array which includes the special characters you will be using extends from 128 to 254. You may learn more about this using the ASCII table as needed. Remember that these values should NEVER be found in your code; however, it is very appropriate to store either ASCII values or characters in a global constant.

20) Programming Hints:

a) due to the many components that need consistency throughout your program, you will be wise to use global constants; the example program uses roughly 80 of them

b) you will need to test all three text files before starting the game play; your program must show an error if any of the files are not found, and stop game play from continuing; you should remove one of the text files from your folder and see how the program responds to this condition

c) your program design strategies will be challenged with this program; you need to spend a significant amount of time in design before starting on code; in addition, while you will likely divide up the work among your team members, you will be much wiser to work regularly and frequently together to implement the design; if this is done well, it will be easy to develop the functions independently; as a reference, the example program has more than 25 functions although some of them are needed for the extra credit components which you may or may not implement

d) your Design Assignment steps 1 through 4 should go easily; you only have to provide the menu structure in the **`main`** function although you must make sure it is complete; the challenges will be developing the supporting function with emphasis on the play-game function; this function should take advantage of several other functions and you should work in your team to completely and thoroughly design this function before you go on to the others

e) remember that you may only use c-style strings on this assignment; any use of the **`string`** class/object will incur credit reduction; also note that we may not have time in class to thoroughly discuss the sorting operations necessary for your score file management, but as you know, there is information on that in the online reference

21) Extra Credit:

a) addition of the spell operation allows the player to walk through internal walls as long as there are spells available, and not walk through screens if none are available; hitting a spell symbol increments the spell count, walking through a wall decrements the spell counter. Credit: up to 2 points/4% of the available PA grade

b) requires spell operation to work, then addition of the sword and monster operations as specified previously requires that you can kill a monster with the sword and decrement the sword count, hitting a sword increments the sword count; the game must end if the player hits a monster without a sword in her/his possession. Credit: up to 4 points/8% of the available PA grade

c) for either of these additions, you must add the information to the initially-displayed instruction screen, and you must add the scoring information to the left instruction panel; you must also keep track of the sword/spell count and manage incrementing and decrementing it/them appropriately

d) also note that your program must meet all of the non-extra-credit requirements before extra credit will be considered

23) Create a screen shot of each of the screens (9 in all):
    a) splash screen
    b) instruction screen with extra credit items as appropriate
    c) main menu
    d) display of the game screen: 1) first started before player moves, 2) with all score items showing some value, 3) when the game ends
    e) display of score file: 1) empty, 2) with 5 – 7 names, and 3) full

24) Also provide the three score files for the three score display conditions specified in item d) of the previous item

TEAM OPERATIONS

25) You will be working in teams of 3, or in some cases 2. Your team members will be assigned to you by your TA. Team assignments are not negotiable; please do not ask your TA to change this.

26) You must designate a team leader who will be responsible for uploading the DA and the PA. However, it is still your responsibility to make sure that your team leader meets her or his responsibilities. Failure to upload the assignments, or uploading them incorrectly will likely result in the assignment not being graded.

27) In the Design Assignment function specifications, one extra line will be added that indicates the person responsible (i.e., the "Developer") for specification and implementation of that function. An example function specification must look like the following

```
/*
Name: openAnalysisFile
Process: clear and open file
Input: file name (char [])
Output/passed: file stream object (ifstream)
Dependencies: ifstream I/O tools
Developer: Roger Ramjet
*/
void openAnalysisFile( char fName[ MAX_STR_LEN ], ifstream &inf );
```

28) Note that there may not be a significant difference between your DA and your PA in terms of functions called and implemented. Adding, deleting, or significantly modifying* more than three functions will drive a credit reduction. *Based on your TAs judgement.

29) With your **Design Assignment**, your team must also turn in a separate paper indicating your team names, your leader, and how you have broken out the functions that are necessary for the program. In other words, if you have 27 functions, your paper might look like the following:

    Team leader: Sandra Smith
    Other members: Bill Rogers, Ralph Majors

    Functions for Sandra Smith:
    func1, func2, func3, etc.

    Functions for Bill Rogers:
    func10, func11, func12, etc.

    Functions for Ralph Majors
    func24, func25, func26, etc.

30) Note that only one copy of your work will be uploaded by the team leader; this is explained in the "Turning in" part later in this document.

31) For your **Programming Assignment**, each student will also turn in a very simple evaluation sheet in Word (.doc) form (no other file form is acceptable). This sheet must include your name, and your evaluation of yourself and your team members. The information on your forms will be used as part of the grading process. An example is shown here:

My Name: Bill Rogers	My contribution to the PA: 35%
My Team Leader: Sandra Smith	Her contribution to the PA: 35%
Other Team Member: Ralph Majors	His contribution to the PA: 30%

Turning in your Design Assignment:

Information:

Week: 13
Laboratory:13
Design Assignment: 9
Due Date: 11/28, 6:00 pm

To turn in – by the team leader:

The first five steps of the Six Step Programming Process, including:

1. roguelike_s1.cpp
2. roguelike_s2.cpp
3. roguelike_s3.cpp
4. roguelike_s4.cpp
5. roguelike_s5.cpp
6. paper with function

Upload these as separate files. Note that following the instructions for uploading your work is critical; you will lose points if you do not follow these instructions. Do not upload any files or in any format other than that specified here. **Also note that if the uploading format is not implemented correctly, your Design Assignment will not be graded.**

1) The team leader will upload the separate files, and then must place the following in the "Assignment Materials" area:

Team Leader: <team leader name>
Other members: <other members name>
Section Number: <your section number>
TA Name: <your TA name>

2) The other one or two members will NOT attach anything to their upload action, but must still place the following in their "Assignment Materials" area and upload the following:

My Name: <member name>
Team Leader Name: <team leader name>
Other Member Name: <other member name>
Section Number: <your section number>
TA Name: <your TA name>

Following these instructions is critical to the appropriate grading of your work. If it is not done correctly, your DA may not be graded.

Again remember that the team leader must upload the materials but it is the responsibility of all team members to make sure this process gets completed and on time.

Turning in your Programming Assignment:

Information:

Week: 14
Laboratory: 14
Programming Assignment: 10
Due Date: 12/05, 6:00 pm

To turn in:

1. The Word file containing the following:

    a. There should be at least nine screen shots for the programs as specified in item #23 previously in this document

    b. Remember to clearly annotate every displayed result

2. The executable file:
    a. roguelike.exe (roguelike_s6.exe is also acceptable)

3. The source code file:
    a. roguelike_6.cpp

4. At least <u>three</u> score files as specified in item #24 previously in this document

5. The evaluation document specified in item #31 previously in this document

These files <u>must</u> be compressed and uploaded as one zip file. To do this, select all of the required files, right click on them, and select "Send To", then select "Compressed (zipped) Folder". **<u>Also note that if the uploading format is not implemented correctly, your Programming Assignment will not be graded.</u>**

Once the folder is created, it will be placed in the same folder in which you are working. Change the name of the zipped folder to "LastnameFirstname_PAX" (where 'X' is the number of the Programming Assignment) as shown in the following example: "LeveringtonMichael_PA3" (no quotes). After you have renamed the zipped folder, double click on it to verify that it has all the files it is supposed to have.

1) The team leader will upload the compressed file and her or his evaluation form, and then must place the following in the "Assignment Materials" area:

Team Leader: <team leader name>
Other members: <other members name>
Section Number: <your section number>
TA Name: <your TA name>

2) The other one or two members will upload ONLY their evaluation forms (in a Word (.doc) file), and will place the following in their “Assignment Materials” area:

My Name: <member name>
Team Leader Name: <team leader name>
Other Member Name: <other member name>
Section Number: <your section number>
TA Name: <your TA name>

Following these instruction is critical to the appropriate grading of your work. If it is not done correctly, your PA may not be graded.

Again remember that the team leader must upload the materials, and the team members must upload the evaluation forms, but it is the responsibility of all team members to make sure that all the materials get uploaded correctly and on time.

Note that following the instructions for uploading your work is critical; you will lose points if you do not follow these instructions. Do not upload any files or in any format other than that specified in this document.

For any other information on how to turn in Programming Assignments, refer to the "How to Turn in Programming Assignments" in the "General Course Information" folder