Ascii Art – Capstone project in C

CSSE 120 – Introduction to Software Development (Robotics)
Spring 2010-2011

How to begin the Ascii Art project

Proceed as follows, <u>in the order listed</u>.

1. If you have not done so already, learn the project specification – what your project is required to do. See the ***AsciiArtSpecification*** document for that information – we discussed it in class.

   In a nutshell, your project will:

   - Read ASCII artwork from a file that stores the artwork in a simple *compressed* form.
   - Display the artwork in the *uncompressed* form in which it is intended to be viewed.
   - Do some simple manipulations of the artwork.

   It will interact with the user by repeatedly displaying (on the console) a numbered menu of choices. The user enters the number for her choice. Then the program does whatever is required for that choice. This continues until the user selects the QUIT choice.

2. The code that we developed in class today is in your individual repository in a project called

   AsciiArtAgain

   Note: **AsciiArtAgain** and NOT **AsciiArt**.

   Please do all your work in this *new* project and discard the old one. (If you have code in the original **AsciiArt** project that you want to keep, just copy it into the parallel file in **AsciiArtAgain**.)

   The code in **AsciiArtAgain** is similar to the code that we developed in class, ***except in two major ways:***

   - *main* (and functions that *main* calls) is now in a file **main.c**

     Thus, the **AsciiArt.h** and **AsciiArt.c** files contain only structure information and helper functions, akin to the other ***.h*** and ***.c*** file pairs.

   - The ***character*** field of an **ArtTriple** is now just a ***single character***, not an *array* of characters.

   We put each structure definition (**ArtTriple**, **ListOfArtTriples**, **AsciiArt**) into its own file pair because:

   - Each *filename* is the *structure name* with *.h* appended to it, so the structure definitions are easy to find.
   - We teach software development techniques that scale up. For large projects (hundreds or thousands of structures and other such things), separating functionality into separate files is critical for keeping intellectual control of the project.
   - Separating **.h** and **.c** files allows the compiler to run faster in large projects (it does not have to recompile code in files that have not changed).
   - This organization is standard practice in software development in C.

3. Read the code, in the following order:

   - **ArtTriple.h – This defines an ArtTriple:**
     - A ***row*** and ***column*** and the ***character*** to be printed at that row and column.
   - **ArtTriple.c**
   - **ListOfArtTriples.h – This defines a ListOfArtTriples:**
     - An *array* that contains the **ArtTriple**'s that define the artwork in its *compressed* form.
     - The *length* (size) of that array, that is, how many **ArtTriple**'s it currently holds (initially zero).
   - **ListOfArtTriples.c**
   - **AsciiArt.h – This defines an AsciiArt: the collection of information that the program needs to operate on a single piece of ASCII artwork.** For example, the name of the file that contains the artwork in its compressed form, the number of rows and columns in the artwork, and the row/column/character triples that specify the non-space characters to be printed.
   - **AsciiArt.c**
   - **main.c – The *main* function and the functions that it calls, etc.**

   As you read, make sure that you understand:

   - ***Everything*** stored in each of the 3 structure types.
   - The ***make*** and ***print*** helper functions for each structure type – what they do, why they do it, and how they do it.
   - What ***main*** is doing, and why.

   Send questions to csse120-staff@rose-hulman.edu.

   If you run the program, it is currently in an infinite loop – you will fix that next.

4. Proceed according to this iterative enhancement plan, in which you implement, test and debug each stage before proceeding to the next stage.

Stage 1: Implement the *structures* and their helper functions, the *main loop* in ***main***, and the function that *shows the menu*. [This is what we developed together in class and what you were given in the **AsciiArtAgain** project. ***So this stage is done!***]

Except as noted, all your work after Stage 1 will be in main.c.

Stage 2: Currently, the program is in an infinite loop. Augment ***dealWithMenuOnce*** so that it temporarily does ONLY this: It sets the ***quittingTime*** field of the **AsciiArt** structure that it is given to **TRUE**. (But it continues to *return* the art object, as it does now.) After you implement this, the program should print the menu ONCE and quit.

Stage 3: Augment ***dealWithMenuOnce*** so that it does the following part of its specification: It prompts for and inputs the number of the choice from the menu that the user chooses. Temporarily, it also prints that number, so that you can test this stage. (Remove the print when this stage is working correctly.)

Stage 4: ***AFTER*** reading ***ALL*** the instructions for this stage, augment ***dealWithMenuOnce*** so that each user choice (1, 2, … 81, 99) calls a sensibly-named function to (eventually, not yet!) deal with that choice. See, for example, the *enterFilename* function that we already supplied for choice 1. ***First READ and NOTE that:***

- You will move the *quittingTime* code you previously wrote to the case that handles the QUIT choice (99).
- All other choices should implement ***stubs*** for functions that (at this stage) simply print something like "*Function enterFilename is not yet implemented*."
    - Exception: You already have (and should directly call) a function ***printAsciiArt*** that handles the DEBUG (81) choice.
- You should (and often must) provide a ***prototype*** for each of your functions – see the prototypes near the top of **main.c** for examples.
- If the user makes an invalid choice, like 9, simply print a short error message like "*Invalid input. Try again.*"
    - IMPORTANT: You do NOT need to do any "input validation" anywhere in this project except as *explicitly* specified. For example, you can count on the user not entering *blast!* (or any other non-numeric input) as the number she chooses from the menu.

Test that ALL the choices work as intended so far (i.e., the loop continues until **99** which exits the program, **81** prints the default **AsciiArt** object, and the other choices print an appropriate *not yet implemented* message.)

IMPORTANT TECHNICAL NOTE: You are very likely to encounter an error message like:

```
... ld.exe: cannot open output file AsciiArtAgain_Solution.exe: ...
```

This almost certainly means that your program is already running (probably waiting for input). Kill all your running instances (get help as needed) and all should be OK.

Stage 5 begins on the next page.

Stage 5: [Read ALL of this LONG description of Stage 5 before implementing any of it. The code you write here is not terribly long (mine is under 40 lines of code), but subtle – you *must* maintain intellectual control of it.]

In this stage, *make Choice 3 from the menu work correctly*, so that the user can enter ASCII art herself. In particular, if the user selects Choice 3, the program should:

- Prompt for and input from the user the number of *rows* and *columns* in the ASCII art being entered.
- *Repeatedly* prompt for and input from the user:
    - A *row number*
    - A *column number*
    - A *character* to be printed at that row and column

  Stop when the user enters a *negative number* for either the row or the column (and don't include the entry with a negative row or column in the data, of course).

  The row and column numbers should be zero-based. For example, if the number of rows is 5, then the row number can be 0, 1, 2, 3, or 4). You are NOT required to do any input validation here.

  See the **Sample Run** in the **AsciiArtSpecification** document for an example of the user interaction that should run when the user chooses Choice 3 from the menu.
- *Store* the information that the user inputs (per the preceding bullet) in an object that other functions can use.

  To this end, recall that your function that implements Choice 3 should be called from the ***dealWithMenuOnce*** function. The ***dealWithMenuOnce*** function takes (as its sole parameter) an **AsciiArt** object that is intended to hold all the information about the current ASCII artwork. Thus, you need to:
    - Send that **AsciiArt** object to your function that implements Choice 3 (as its sole argument).
    - Store the information obtained from the user (number of rows, number of columns, and a bunch of row/column/character triples) in the appropriate place of the **AsciiArt** object sent to your function.
    - Return that **AsciiArt** object from your function, putting it back into the same variable that was sent to your function, using a statement like:

      ```
      art = enterYourOwn(art)
      ```

      if your function that implements Choice 3 is called **`enterYourOwn`**.

      This is an important statement; be sure you understand exactly ***why*** it is necessary (why can't the art argument just be mutated?) and ***how*** it achieves its goal (updating the **`art`** variable to include the information that the user enters when executing ***enterYourOwn*** in response to the user choosing Choice 3.

  With regard to storing the information in the **AsciiArt** object:
    - The **ArtTriple** and **ListOfArtTriples** structure definitions are complete – you must ***use*** (***call***) them but should ***not modify*** them.

- On the other hand, you ***must modify*** the **AsciiArt** structure definition as you continue to implement this project. In particular, here you must add fields to the **AsciiArt** structure definition to hold the *number of rows* and *number of columns* of the ASCII artwork that Choice 3 inputs.
- Reset the ***length*** of the array in **ListOfArtTriples** to zero, since the user is now entering a new piece of ASCII artwork (other menu choices let the user
- Other than the number of rows and number of columns, the three structures have been set up to be *exactly* what you need. ***Use them wisely, which includes using their `makeXXX` helper functions.***

  Understanding how to store and access information in *structure instances* and *arrays* is a critical learning objective of this project, so be SURE that you COMPLETELY understand *what* you are doing here and *why*. Don't hesitate to ask your instructor questions about this (or any part of this project).

IMPORTANT TECHNICAL NOTES:

- Throughout this program, use ***scanf*** and ***fscanf*** to do input, and do ***NOT*** use their ***%c*** formatting code to input a ***character***. Instead, ***input a string when you want just a character*** and extract and store the first character of the string. See the discussion about *strings* (Thursday in class) for details about how to do this and why it is unwise to do otherwise in this program.
- Use the existing (invaluable, and working!) ***printAsciiArt*** function as you are debugging your code. You'll need to add *printf*'s to the existing code that implements ***printAsciiArt***, for printing the fields that you added to the **AsciiArt** structure.
- The symbol for the logical operator ***OR*** in C is the vertical bar character (on the upper-right of most keyboards).
- It is quite OK, indeed recommended, that you add and use additional helper functions of your own as you do this project. In my implementation,
- Structure assignment, as in the third statement below:

```
ArtTriple a, b;
a.row = ...


b = a;
```

  makes a *copy* of the data. (Details in class Thursday/Friday about why this is important.)

- Pausing and seeing the pop-up information when you type the ***dot*** after a structure object (as in the above example) makes dealing with the structures much easier than it would be otherwise.

Stage 6: Make Choice 4 in the menu work correctly. That is, implement the function that displays the current ASCII art in its uncompressed form (the form in which it is intended to be viewed).

- Recall that the current ASCII art is stored in the AsciiArt object in the ***dealWithMenuOnce*** function. Thus, that object should be sent to and returned from your function that deals with printing the ASCII art.
- You will need a two-dimensional matrix of characters for this, with the number of rows and columns the same as your AsciiArt object indicates. (If the number of rows or columns are negative, that means that there is nothing to print.) You:
    - Declare the matrix (hence allocating storage for it), per the number of rows and columns specified in your AsciiArt object.
    - Loop through the matrix to make it contain all spaces (blanks).
    - Loop through the list of ArtTriple's in your AsciiArt object. For each triple, put its character at the specified row and column in the matrix.
    - Loop through the matrix, row by row, and within each row, column by column, printing its characters.

  If you prefer, you can store the two-dimensional matrix in the AsciiArt object. That uses more storage but runs a bit faster since you don't have to reload the characters in the ArtTriple's when you print the matrix a second (or third…) time. A classic time-space trade-off. Either way is fine (and neither is a lot harder than the other) – it's your choice.

Stage 7: Make Choice 2 in the menu work correctly. That is, implement the function that reads the compressed-version data from a file instead of from a user.

This is quite similar to Stage 6. You can test using the **plane-coords.txt** file that I placed in your AsciiArtAgain project.

Stage 8: Make Choice 1 in the menu work correctly. That is, allow the user to enter than name of the file containing the ASCII art.

Stage 9: Make Choice 7 in the menu work correctly. That is, implement the function that writes the ASCII artwork to a file, in its uncompressed (matrix) form. [More on this Thursday.]

More to come, but if you get all the above done, you are in great shape.