Comsci 110 Term project Summer 2016

Total score: 100

Due date: Tuesday July 26th 11:59 PM. LATE SUBMISSIONS WILL NOT BE ACCEPTED AT ALL.

Grading: As you are becoming programmers, you are expected to write programs on your own without help. The two exams in this course do not include writing codes and compiling and running using computer. This project has been considered to evaluate your own programming skill as a kind of *exam*. The instructor will not help you with writing the code, compiling, and finding the problems and errors. You are completely responsible for this project. You will do it outside the lecture and lab hours. If there is extra time n the lab you can work on it. After you are done you submit it on D2L before the deadline.

If you have general questions about the project, you will ask it during the lab if there is time.

Copying other students' work is considered as cheating and I have zero tolerance for this. Any cheating will result in 0/100.

A program which does not compile will result in 0/100.

5 points is deducted for anything that is not exactly following the instructions of this handout.

This project is the base of the concept in computer science is called "parsing", which means making it easier to enter email addresses into an email message that is to be sent to a list of recipients, when those recipients are not already in a contacts list.

The project is broken into 3 parts to make it easier for you and guide you proceed step by step.

The required input file is provided. Do not use other input file, unless provided by the instructor. (You can test your code with other simple input files while you are working on the project, but for submission and showing the results you will need to use the provided input file)

What you will need to submit:

-The three source files (codes).

-One word file which has your code and the screenshots of the outputs (like the assignments).

Part1:

Requirements. Write `email1.cpp` (that's "email one"). The program should prompt the user for the input and output filenames, and print the user's choices, and that's all.

The prompts should include the default names for the files. A prompt should look something like this:

"`Enter input filename [default: fileContainingEmails.txt]:` ".

Remember that if the user selects the default for the input filename, then the default for the output filename is "copyPasteMyEmails.txt". Otherwise, if the user types something for the input filename, and does not accept the default, then the default for the output filename is the same as what the user entered for the input filename.

Do NOT type fileContainingEmails.txt or copyPasteMyEmails.txt more than once in your code. It is bad programming practice to repeat values in code. Instead, store otherwise repeated values in variables, and refer to those variables.

Here are some guidelines:

1. If you name the program's source file something other than what is specified, you are not doing this right.
2. If you spell or case the default filenames wrong, you are not doing this right.
3. If the user does not see exactly two prompts every time, you are not doing this right.
4. If I see the character sequence `fileContainingEmails.txt` in your code more than once, you are not doing this right.
5. If I see the character sequence `copyPasteMyEmails.txt` in your code more than once, you are not doing this right.
6. If your prompts do not include the names of the defaults, you are not doing this right.
7. Do not open the input or output files.

It should work like this:

- It should begin with a prompt for the input filename, that also shows the default name, `fileContainingEmails.txt`. If I simply press ENTER, then the input filename should be set by your program to the default name, `fileContainingEmails.txt`.
- Whether I type a filename and press ENTER in response to the prompt, or type nothing and press ENTER, I should get another prompt -- one for the output filename. I should ALWAYS get exactly 2 prompts.
- If I typed a name for the input filename, then whatever I typed also becomes the default for the output filename, instead of `copyPasteMyEmails.txt`. Otherwise, if I just pressed ENTER for the input filename, the default for the output filename is to be `copyPasteMyEmails.txt`.
- If I type something and press ENTER for the output filename, then what I typed should become the output filename. Otherwise, if I just pressed ENTER, the output filename should be set by your program to the default, which will either be `copyPasteMyEmails.txt` or what I typed for the input filename.

Make sure that you understand what is meant by a "default". If an input value has a default value, then (1) the prompt should say what is the default value, and (2) if the user presses ENTER without typing anything first, then the default value should be substituted for the blank that got input. That means the programmer (you) has to test the input value with an if-statement. If the input value is blank, then replace it with the default value. Here's a general algorithm for defaulted console input:

```
  Create a string variable to store the input value, I
  Create a string variable to store the default value, D
```

```
Initialize D to some default value
Print a prompt to the console, including what's stored in D
Read the user's input into I
If I is blank
  set I = D
```

Compile and run the program.

Program I/O. Input: prompt for both the input and output file names, allowing blank for the "default" name.
Output: Print to the console the input and output file names selected, converting any blanks to their corresponding default names. There is NO file input or output in this part.

The output should exactly look like the examples. nothing more, nothing less.

Example. Your program's console I/O should look something like this, with user input in blue (excluding your identifying information):

```
Enter input filename [default: fileContainingEmails.txt]: <enter>
Enter output filename [default: copyPasteMyEmails.txt]: <enter>
Input file: fileContainingEmails.txt
Output file: copyPasteMyEmails.txt
```

...or...

```
Enter input filename [default: fileContainingEmails.txt]: x.txt <enter>
Enter output filename [default: x.txt]: <enter>
Input file: x.txt
Output file: x.txt
```

...or...

```
Enter input filename [default: fileContainingEmails.txt]: <enter>
Enter output filename [default: copyPasteMyEmails.txt]: y.txt <enter>
Input file: fileContainingEmails.txt
Output file: y.txt
```

...or...

```
Enter input filename [default: fileContainingEmails.txt]: x.txt <enter>
Enter output filename [default: x.txt]: y.txt <enter>
Input file: x.txt
Output file: y.txt
```

Part2:

Requirements. Write **`email2.cpp`**. The program should prompt the user for the input and output filenames, open and read the input file, and print to the console the lines that contain the character `@`. If a line contains the character `@` more than once, it should be printed once for each. If a line contains no `@` at all, it should NOT be printed.

Use a loop to process the input file. Adapt the for-loop from the nsaEncoder.cpp program, to traverse each line that you read. In the loop, test each character in the line to see if equals the character`@`, and if it does, print the line to the console, and continue through the rest of the line, looking for more `@`'s.

Here are some guidelines:

1. If you open the output file, you are not doing this right.
2. If you do not have nested loops, you are not doing this right.
3. If you do not process the text a line at a time, you are not doing this right.
4. If you do not use the for-loop from the nsaEncoder.cpp, you are not doing this right.
5. If you use pointers or `char` arrays or C-style coding or C-library functions, except for ones demonstrated in our textbook, you are not doing this right.

Compile and run the program.

Program I/O. The program should prompt the user for the input and output filenames, and then print the input file's lines with the character `@` to the console. There is NO **file output** in this part.

Example. Your program's console I/O should look something like this, with user input in blue (excluding your identifying information):

```
   Enter input filename [default: fileContainingEmails.txt]: x.txt
   Enter output filename [default: x.txt]: y.txt
   <td align="left"><a href="mailto:RBurns@dvc.edu"><IMG src="mail.gif"
alt="RBurns@dvc.edu"></a></td>
   <td align="left"><a href="mailto:RBurns@dvc.edu"><IMG src="mail.gif"
alt="RBurns@dvc.edu"></a></td>
```

Part 3:

Requirements. Write **`email3.cpp`**. The program should prompt the user for the input and output filenames, open and read the input file, and print to the console the valid email addresses as they are found. If a line contains the character `@`, create the variables `s`, `e`, and `hasDot`, and use loops to find their values. If the values for these three indicate that the `@` is inside a valid email address, print to the console that address. Repeat for each `@` in a line.

Do not deal with lists or duplicates at this point.

Here are some guidelines:

1. If you open the output file, you are not doing this right.
2. If your e-loop is NOT after your s-loop, you are not doing this right.
3. If your s- and e-loops are not contained in the code block of an if-statement that tests for the character `@`, you are not doing this right.

Compile and run the program.

Program I/O. The program should prompt the user for the input and output filenames, and then print to the console every valid email address as it is found. There is NO file output in this part.

Example. Your program's console I/O should look something like this, with user input in blue (excluding your identifying information):

```
Enter input filename [default: fileContainingEmails.txt]: x.txt
Enter output filename [default: x.txt]: y.txt
RBurns@dvc.edu
RBurns@dvc.edu
```

Hints

Email addresses consist of the characters A-Z, a-z, 0-9, underscore, dot, hyphen, and plus. Also, they must have exactly one '@' followed by at least one '.'.

The procedure basically involves reading a line from a file as a text variable, and traversing the line of text to find a '@' character. If one is found, its position is saved. Then traverse backwards until an invalid email character found -- that is the position *before* the email address starts. Then traverse forwards from the '@' until an invalid email character is found -- that is the position *after* the email address ends Also count the number of '.'s found as you traverse forwards from '@' -- if any are found, then you can extract a copy of the email address as a substring. Continue from the position after the extracted email address, until no more '@'s are found.

- Read a line from the input file as `string line;`. Traverse it, testing each `line[i]` until you find a '@'. Then look *backwards* in a loop until you find an invalid email address character, or run into the start of `line`. Then look forwards from the same '@' in another loop until you find an invalid email address character, or run into the end of `line`. When looking forwards, be sure to count the number of dots ('.') found -- there needs to be at least one in a valid email address.

- Write a value-returning function for testing a character to see if it is a valid email address character. For example, `bool isValidEmailCharacter(char c)`. In it, test to see if "c" is >='A' and <='Z', or >='a' and <='z', >='0' and <='9', or =='.', or =='-', or =='+'. If it satisfies any of these conditions, return `true`. Otherwise, return `false`.

For all parts consider these for your score deduction:

1. if your program's code does not apply proper alignment and indenting, -5,
2. if your program's code is not commented similarly to the code samples in the textbook, -5,
3. if either or both of the default input and output filenames (fileContainingEmails.txt and copyPasteMyEmails.txt) are not spelled correctly, -5,
4. if your program does not show the default input filename in the prompt, -5,
5. if your program does not show the default output filename correctly in the prompt, when the default input filename is selected, -5,
6. if your program does not show the default output filename correctly in the prompt, when the default input filename is *not* selected, -5,
7. if the user has to do more than to press ENTER to get the default input and/or output filename, -5,
8. if your program does not have exactly two prompts under any circumstances, -5,
9. if any quoted literal string containing a default input or output filename appears in the code more than once, instead of being stored in a variable, -5,
10. if the emails are not listed to the console screen, one per line, -5,