CSE 1384 - Functions/Files
Lab 4

Objectives:

- Continue practicing past concepts
- Practice using functions
- Practice using files

Assignment:

This week's assignment involves creating a program that takes in a Mad Libs template as a text file and prompts the user for the appropriate items. The program should then display the completed story to the user.

There are **three MadLib text files provided for you** to use in this program, all following the same format: `accident.txt`, `shopping.txt`, and `trip.txt`. Each file consists of only one line of text that contains the entire story. Each "blank" is denoted by the all caps word REPLACE. Each MadLib then has a corresponding prompt file: `accident_labels.txt`, `shopping_labels.txt`, and `trip_labels.txt`. If you're using `accident.txt`, you should use `accident_labels.txt` to find out what word type is needed to replace a blank in the MadLib. If a given MadLib has 14 instances of the word REPLACE, then the corresponding labels file will have 14 types of words on individual lines.

Your program should have three functions:

- main
  - Parameters: none
  - Return values: 0
  - Purpose: Receive the name of the MadLib from the user (accident, shopping, or trip). Loop validate the files existing. Read in your MadLib and label files into separate string vectors: MadLibs and Labels. The MadLib files have the entire story on one line and the Labels files have each prompt on a separate line. Take this into consideration (a line → label item, each word → MadLib item).

    Then, call your generateMadLib function with your vectors. Display the completed story after processing.
- generateMadLib
  - Parameters: MadLibs and label vectors (by reference)
  - Return values: none
  - Purpose: Iterate through the MadLibs vector checking each item. If it contains REPLACE, ask the user for an input using the labels vector to determine what type of prompt it is – determine what article (a/an) to use for your given label by calling the getArticle function. For the first REPLACE, use labels index 0, and so on. Modify the MadLibs vector to contain the user input (make sure you can accept spaces in the input!)

- getArticle
    - Parameters: the word
    - Return values: “a” or “an”
    - Purpose: Take in a label word. Look at the first character of the string and determine if it’s a vowel – return “an” if it is, “a” if it isn’t. Methodology from here of how you do so is up to you.

Hints:

For files:

You are not entering the entire file name here! You’re entering the MadLib name. So, you won’t enter “accident.txt” if you wanted that file, you’d enter “accident”. You’re then going to be able to open both the MadLib file and the label file using this string as a basis. Here’s an example:

```
string file;

string MadLibFile = file + ".txt"
string labelFile = file + "_labels.txt"
```

This should ensure the corresponding labels file is always referenced correctly.

For using the labels vector in generateMadLib:

Your iteration loop should be going item by item in the MadLib vector to check each word. Consider using an integer outside of this to keep track of which label you’re at. Start at 0. Only increment the label counter whenever you’ve replaced a word (or it’ll get too big). You cannot use the same index for the MadLib and labels as they won’t be the same size or placement.

Milestones:

You must receive a check-off for each item to receive full credit on the lab. You may show the milestones incrementally, or receive a check off for multiple in one demo depending on your progress.

Examples of each milestone are included.

Milestone 1:

File reading. Must show:

- File received by user input
    - Does NOT rely on extension for both file types
- File reading successful for both files
    - Shown via outputting the file contents
- Error checking to loop until a proper file is input

My example separated the file contents through a tab character (“`\t`”) to fit everything more gracefully, but yours can use basic newlines.

(Excuse the text wrapping in my console causing some words to be segmented weirdly)

```
Enter in the name of your mad lib (do not include extension): shop
Invalid file. Please try again.

Enter in the name of your mad lib (do not include extension): shopping

Madlibs file:
REPLACE needed  to      go      shopping.       He      and     his     REPLACE REPLACE to      the     REPLACE .       They
   wanted  to      buy       a       REPLACE REPLACE but     the     manager said    that    she     didn't  have    any.
Then    they    went    to      a       REPLACE   but     it      didn't  have    any     either. Finally they    went    to
     the     mall    and     bought  REPLACE and     REPLACE .They     were    very    REPLACE when    they    returned
  home    and     began   REPLACE .

Labels file:
male name       family member   verb    store   adjective       noun    store   plural noun     plural noun     adjective
   present participle
Files successfully read.
```

Outputting the file contents directly is just how this milestone is shown to work. You’ll remove the output directly later and route contents in your vectors instead.

Milestone 2:

generateMadLibs function – loops through the vector, having an input for each item at the replacement value. Must show:

- Loop process that asks for an input for each REPLACE in the vector
- Direct file output from milestone 1 shouldn’t be here at this point, they should be in vectors

```
Enter in the name of your mad lib (do not include extension): shopping

Files successfully read.

Enter replacement value: dfg
Enter replacement value: dfg
Enter replacement value: jjt
Enter replacement value:
```

Milestone 3:

Incorporate the labels of the MadLibs into your generateMadLib function. Must show:

- Customized input that shows what value is desired
- (customized articles aren't required here – you could use "a" for every label)

```
Enter in the name of your mad lib (do not include extension): shopping

Files successfully read.

Enter in a male name: Devin
Enter in a family member: husband
Enter in a verb: run
Enter in a store: Walmart
Enter in a adjective:
```

Milestone 4:

Incorporate the generateArticle function into your label output. Must show:

- Labels have a/an based on vowel status (more fluid things such as "an hour" need not be accommodated)

```
Enter in the name of your mad lib (do not include extension): shopping

Files successfully read.

Enter in a male name: Devin
Enter in a family member: husband
Enter in a verb: run
Enter in a store: Walmart
Enter in an adjective: small
Enter in a noun:
```

Milestone 5:

Finishing generateMadLib and going back to main to display the finished madLib. Must show:

- All prompts run – for at least two stories
- Story displayed with proper user input replacement – for at least two stories
- Multi word items displayed correctly (if the user entered in "Devin Blake" for a prompt, it took both and incorporated it into a story)

(Excuse the text wrapping in my console causing some words to be segmented weirdly)

```
Enter in the name of your mad lib (do not include extension): stuff
Invalid file. Please try again.

Enter in the name of your mad lib (do not include extension): accident

Files successfully read.

Enter in a male name: Devin
Enter in a present participle: sleeping
Enter in an adjective: dumb
Enter in a noun: dog
Enter in an adjective: gray
Enter in a noun: book
Enter in a body part: leg
Enter in an adjective: hairy
Enter in a noun: couch
Enter in a noun: desk
Enter in a noun: can
Enter in a time period: month

One day Devin was sleeping with his dumb dog . Suddenly he crashed into a gray book and fell down. He b
roke his leg so he ran to the hospital. The doctor took some X-Rays and discovered that he was hairy .
The doctor put on a couch , gave him a desk and recommended that he take can . Six month later he was h
ealthy and playing once again.
Press <RETURN> to close this window...
```

BONUS / Honors Student Credit:

If you're an honors student, you ***must*** complete this portion. If you do not, it will be counted against you.

If you're not an honors student, you may complete this section for an additional 10 bonus points.

In the main function, after you're done outputting the completed MadLib, ask the user if they'd like to save their story (validate for yes/no input in a loop!). If they say yes, receive a file name input and then open that file in write mode. Write the contents of the MadLib to the file (make sure the words are separated by spaces and not newlines!).

```
Enter in the name of your mad lib (do not include extension): trip

Files successfully read.

Enter in a name: Maggie
Enter in a name: Nubbins
Enter in a place: home
Enter in a plural noun: books
Enter in a plural noun: penguins
Enter in an adjective: cute
Enter in a vehicle: helicopter
Enter in a noun: trash
Enter in an occupation: librarian
Enter in a female name: Wendy
Enter in an adjective: slow
Enter in a noun: lamp

Maggie and Nubbins decided to travel to home . They gathered their books and penguins and put them in t
heir cute helicopter . An hour later their trash exploded. They were stuck. Luckily a librarian named W
endy passed by. She helped them fix their vehicle and gave them a slow lamp to help them. They finally
arrived at their destination and had a great time.

Would you like to save this story? (yes/no) dfg
Invalid answer. Try again.

Would you like to save this story? (yes/no) yes
What would you like to name the file? testing.txt
Press <RETURN> to close this window...
```

Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                          NetID: <your netID>
 Date: <current date>                       Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Deliverables:

- C++ code (.cpp file) you used to demo your program
    - If the code submit CANNOT achieve the demo, you will receive a 0.

Point Breakdown:
(100 points total)
A submission that hasn't received a check-off will not be considered for grading.

- 25pts - milestone 1
- 15pts - milestone 2
- 10pts - milestone 3
- 15pts - milestone 4
- 25pts - milestone 5
- 10pts - programming style *
- Honors / BONUS (dependent on Honors status):
    - Honors: 10pts penalty for not completing
    - BONUS: 10pts added for completing

* Programming style includes good commenting, variable nomenclature, good whitespace, etc.