

Due Sunday, September 24 at 6:00 PM

Submit by Friday, September 22 at 6:00 PM for a 5% bonus

Submit by Saturday, September 23 at 6:00 PM for a 2% bonus

For this assignment, the solution to **each problem should be in a separate .cpp file**. Create a new program for each problem within CodeBlocks. Name the file *problem1.cpp* for the first problem and *problem2.cpp* for the second one. Once you have your code running on your virtual machine (VM), you must *submit it to the autograder by zipping all the files* into a single file to be submitted. You **must also submit your code (.zip) to Moodle** to get full credit for the assignment.

Important Note Regarding Zipping Your File: Ensure the .zip is a "flat zip." That is, multi-select the files you wish to zip together and compress them; do not zip the folder they are contained in - this is called a hierarchical zip and COG will not be capable of locating your files. When you double-click on the .zip file, you should be able to see both .cpp files.

Submitting Your Code to the Autograder (COG):

Before you submit your code to COG, make sure it runs on your computer. If it doesn't run on the VM, it won't run on COG. The computer science autograder, known as COG, can be found here: <https://web-cog-csci1300.cs.colorado.edu>

- Login to COG using your identikey and password.
- Select **Assignment 3** from the dropdown.
- Upload your .zip file and click Submit.

Your files within the .zip **must** be named *problem1.cpp* and *problem2.cpp* for the grading script to run. COG will run its tests and display the results in the window below the *Submit* button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

If you are having trouble finding the difference between your output and the output COG is expecting, you can use a diff tool to help you. It will highlight the differences so that they are easy to see.

<https://www.diffchecker.com/diff>

Remember, you must submit your code on COG and Moodle.

Submitting Your Code to Moodle:

You must submit your code to Moodle to get full credit for the assignment, even if the computer science autograder gives you a perfect score.

Please also include the Algorithm as comments in your code submission to describe what each function is doing. Comments at the header of your files should include your name, recitation TA, and the assignment and problem number. **TAs will be checking**

that your code has the required comments and explains the algorithms you have used. If these required comments are missing from your code, your COG score will be reduced by the number of points listed below.

- **10 points for comments at the header of each file in the following format:**

```
// Author: <your name>
// Recitation: <your recitation# and TA>
//
// Assignment 3
// Problem <number>
```

- **10 points for algorithm description**

This is an example of a good algorithm comments. It is taken from the solution for the function **howHot** from Assignment 2. The algorithm is described well and this would receive full points for the algorithm description.

```
/**
 * Algorithm description:
 *   Convert a given temperature in Celcius to a
 *   temperature in Fahrenheit.
 *   Fahrenheit = Celsius * (9/5) + 32
 *   Return Fahrenheit
 */
int howHot(int temperature) {
    int F = temperature * (9.0/5) + 32;
    return F;
}
```

This is an example of a unsatisfactory algorithm description because it does not mention that the conversion is from celsius to fahrenheit, or the mathematical formula being used, therefore the points would be deducted.

```
/**
 * Algorithm description:
 *   Convert a temperature
 */
int howHot(int temperature) {
    int F = temperature * (9.0/5) + 32;
    return F;
}
```

If you would like to see more examples of good algorithm descriptions, refer to the solution for Assignment 2 on Moodle.

Problem Set

For each of the following problems, create a program to solve the problem. Divide the program into separate, but cooperating, functions. Do not write the complete program as one big "chunk" of statements in `main()`.

Problem 1: The Story Generator

In the game Mad-libs, players are asked for parts of speech, such as noun, adjective, or adverb, and those words are plugged into a template sentence to generate a sometimes- funny story.

Menu Function

Write a function called **menu** that is called by `main` when your program starts.

Inside your **menu** function, you should first ask the user which story they want to play. Your question should look like:

```
"Which story would you like to play? Enter the number of the
story (1, 2, or 3) or type q to quit: "
```

If the user types `q` your program should print `"good bye"` and exit the function.

If the user types in an invalid input, for example `"17"` or `"slkerjqoe"`, you should print `"Valid choice not selected."`

If the user types in a valid story number, the **menu** function should call the corresponding story function (**story1**, **story2**, or **story3**).

The user should be allowed to play the game as many times as they like. After printing the new story, your program should ask again which story they would like to play.

```
"Which story would you like to play? Enter the number of the
story (1, 2, or 3) or type q to quit: "
```

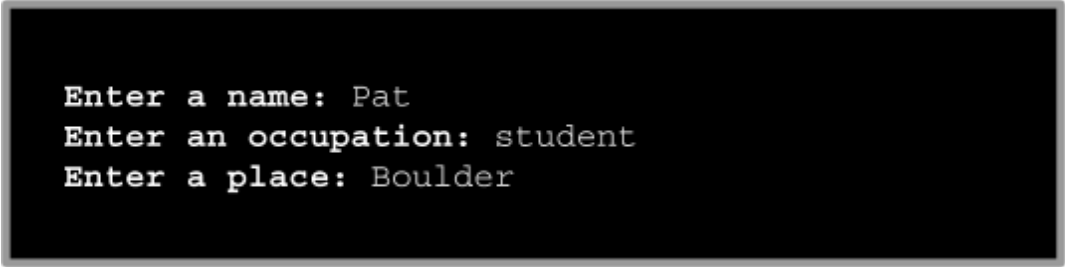
Story Functions

Write 3 functions named **story1**, **story2**, and **story3** that each allow the user to play a simple game of Mad-libs. Each function should store the corresponding story template and ask the user for the missing word types to fill in the template.

For a story template that looks like this:

<NAME> is a <OCCUPATION> who lives in <PLACE>.

The story function should ask for inputs in the following format:



```
Enter a name: Pat
Enter an occupation: student
Enter a place: Boulder
```

Then the story function should print the new sentence with the user entered inputs in place of the word-type placeholders.

Pat is a student who lives in Boulder.

Use the following template for **story1**:

In the book War of the <PLURAL NOUN>, the main character is an anonymous <OCCUPATION> who records the arrival of the <ANIMAL>s in <PLACE>.

Ask for the following inputs for **story1**:

```
"Enter a plural noun: "
"Enter an occupation: "
"Enter an animal name: "
"Enter a place: "
```

Use the following template for **story2**:

<NAME>, I've got a feeling we're not in <STATE/COUNTRY> anymore.

Ask for the following inputs for **story2**:

```
"Enter a name: "
"Enter a state/country: "
```

Use the following template for **story3**:

```
Hello. My name is <FIRST NAME>. You killed my <RELATIVE>.  
Prepare to <VERB>.
```

Ask for the following inputs for **story3**:

```
"Enter a first name: "
```

```
"Enter a relative: "
```

```
"Enter a verb: "
```

IMPORTANT NOTE: User-input to play further must be obtained inside the **menu** function, **not** within your **story1**, **story2**, **story3** and functions. That is, the loop that enables the game to be played indefinitely **must not** be within your story functions - COG will not accept it.

Note: If a function does not take a parameter, you must specify the parameters as void.

e.g. **story1(void)**

For this assignment, the solution to each problem should be in a separate .cpp file. Create a new program for each problem within CodeBlocks. Name the file *problem1.cpp* for the first problem and *problem2.cpp* for the second one. Once you have your code running on your virtual machine (VM), you must submit it to the autograder by zipping all the files into a single file to be submitted. You must also submit your code (.zip) to Moodle to get full credit for the assignment.

Problem 2: Wind Chill

The **wind chill** is the still-air temperature that would have the same cooling effect on exposed human skin as a given combination of temperature and wind speed.

"Windchill." *Merriam-Webster.com*. Merriam-Webster, n.d. Web. 5 Sept. 2017.

For this problem, we will be recreating part of this [wind chill calculator](#) provided by the National Weather Service (NWS).

$$\text{Wind Chill (}^{\circ}\text{F)} = 35.74 + 0.6215T - 35.75(V^{0.16}) + 0.4275T(V^{0.16})$$

Where, T= Air Temperature ($^{\circ}\text{F}$) V= Wind Speed (mph)

Effective 11/01/01

Use this formula for wind chill for your calculations.

Part A

Write a function **windChillCalculator** to determine the wind chill. Your main function should take in user inputs for T (air temperature) and V (wind speed) and pass these values as parameters to your function. After your function calculates the wind chill it should **return** it to main.

To test your code, for T = 30.0 and V = 5.0, you should get a Wind Chill of 24.7268.

Within your main function, print the following cout statement:

```
cout << "The wind chill is " << wind_chill << " degrees F." << endl;
```

HINT: Look at the **pow()** function in the C++ math.h library

Part B

Once you have the wind chill calculation working, create another function **printWindChill**. This function should call **windChillCalculator** and print all the wind chill values for a fixed temperature and variable wind speeds. You will need to write a loop in this function to iterate over the different wind speeds.

The function should receive T (air temperature), low_wind_speed, high_wind_speed, and wind_speed_step as input parameters.

For Example: Let the input parameters be T = 30.0, low_wind_speed = 5.0, high_wind_speed = 8.0, and wind_speed_step = 1.0, your function should print out the wind chill for a temperature of 30 °F and wind speeds starting from 5 mph and ending at 8 mph, incrementing by 1 mph. (5 mph, 6 mph, 7 mph, and 8 mph)

Use the following cout statement to output the values from **within the function**:

```
cout << "The wind chill is " << wind_chill << " degrees F for  
an air temperature of " << T << " degrees F" << " and a wind  
speed of " << V << " mph." << endl;
```

Points to remember when running your code in COG:

These are some things in which COG is very rigid:

1. Students ***must*** include ***void*** for the input parameter of functions that **don't** take input, rather than leaving it blank.

For example,

correct

incorrect

`int some_function (void) {}`

`int some_function () {}`

2. Whenever you are prompting the user for input from ***within*** a **user-defined function**, they must end the prompt with a colon (:).

For example,

`// Inside some_function()`

correct

incorrect

`cout << "Please enter a value for x: ";`

`cout << "Please enter a value for x ";`

3. Answers printed from ***within*** a **user-defined function** must end with a period.

`// Inside some_function()`

correct

incorrect

incorrect

`cout << "The answer is 42.";`

`cout << "The answer is 42!";`

`cout << "The answer is 42";`

4. In addition to #3, answers printed from ***within*** the user-defined function must exactly follow the cout string template given in the writeup.

Challenge Problems: for those who want more coding practice

These problems are for your practice only, DO NOT submit to COG, do not submit to Moodle. You can show your results to the CAs, TAs, and the instructor to validate that you have achieved the correct results.

Challenge #1: Create a function that can search a story for a placeholder ('<' description '>') and prompt the user for the replacement. For example, in the story below, search the string to find the first placeholder "<PLURAL NOUN>" and prompt the

user with “Enter PLURAL NOUN:” to get the response from the user. Place the response into the story in place of the placeholder. Repeat this process until all the placeholders have been replaced.

In the book of the <PLURAL NOUN>, the main character is an anonymous <OCCUPATION> who records the arrival of the <ANIMAL>s in <PLACE>.

Challenge #2: Create a program to generate the chart below:

WIND (mph)	Temperature (° F)												
	35	30	25	20	15	10	5	0	-5	-10	-15	-20	-25
5	32	27	22	16	11	6	0	-5	-10	-15	-21	-26	-31
10	22	16	10	3	-3	-9	-15	-22	-27	-34	-40	-46	-52
15	16	9	2	-5	-11	-18	-25	-31	-38	-45	-51	-58	-65
20	12	4	-3	-10	-17	-24	-31	-39	-46	-53	-60	-67	-74
25	8	1	-7	-15	-22	-29	-36	-44	-51	-59	-66	-74	-81
30	6	-2	-10	-18	-25	-33	-41	-49	-56	-64	-71	-79	-86
35	4	-4	-12	-20	-27	-35	-43	-52	-58	-67	-74	-82	-92
40	3	-5	-13	-21	-29	-37	-45	-53	-60	-69	-76	-84	-92

