CSCI 1300 Introduction to Computer Programming

Instructors: Knox

Assignment 3

Due Friday, Feb 10, by 12:30 pm

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 **MUST BE A FLAT ZIP**.  After submitting your code to COG, you **must also submit your code (.zip)** to Moodle to get full credit for the assignment.

**Submitting Your  Code to the Autograder:**

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 do not get your assignment to run on COG before the assignment deadline, you will have the option of scheduling an interview grade with your TA to get a grade for the assignment. We’ll talk more about scheduling the interview in lecture and recitation. Even if you do get the assignment to run on COG, you can schedule the interview if you just want to talk about the assignment and get feedback on your implementation.

**Submitting Your Code to Moodle:**

You must submit your code to Moodle to get full credit for the assignment, even if COG gives you a perfect score and your Moodle grade displays a score of 100%. If your .zip file is not uploaded to the Moodle, your score will retroactively be reduced to account for the absence of the files.

Please also include comments in your code submission to describe what your code 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 you code has comments.

**Problems:**

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().

1. The Story Generator

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

For this problem, write a function named “**madLibs**” that plays a simple game of Mad-libs.

void madLibs(void)

Your program needs to store the story template in a string variable and ask the user for parts of speech to fill in the template.  Use the following template:

“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>. Needless to say, havoc reigns as the <ANIMAL>s continue to <VERB> everything in sight, until they are killed by the common <NOUN>.”

When the programmer starts, you should first ask the user if they want to play a game. Your question should look like:

“Do you want to play a game? (y) or (n)”

The user types *y* if they want to play and *n* if they don’t. If the user answers *y*, then your program should ask for entries for each of the missing word types in your template, such as:

“Enter a plural:”

“Enter an occupation:”

“Enter an animal:”

“Enter a place:”

“Enter another animal:”

“Enter a verb:”

“Enter a noun:”

Build a new string variable that includes the user entries in place of the word-type placeholders and print the variable to show the user the new sentence.

If the user types “n” when asked if they want to play, your program should print “good bye” and exit.

The user should be allowed to play the game as many times as they like.   After printing the new sentence, your program should ask if they would like to play again, and if they answer *y*,  your program should repeat the game, beginning at the spot where you ask the user to enter word types.

*Note: The COG grading script will test with different inputs for the same template. Once you have your code running on COG, you are welcome to explore other approaches to generating the template, such as reading it in from a file, or having multiple templates that can be selected randomly. Doing so will make it possible to build a much more complex, and interesting, game.*

2. Solar Energy

Solar Energy (measured in KWh) generated by a solar panel depends on its area, efficiency, solar radiation of the location, and performance ratio.

Equations to calculate solar energy output is shown here:

E = A\*r\*H\*Pr

where E is the energy generated annually by the solar panel in kWh.

There are several other noteworthy parameters in these equations:

A is area of the solar panel in meter square

r is the solar panel efficiency in percentage

H is the average solar radiation in kWh/m2

Pr is the performance ratio (coefficient of loss), it’s usually between 0.5 and 0.9 with a default of 0.75

* 1. Write a function “**energyCalculator**” to calculate peak energy output. The function should take in user inputs for A, r, H, and default value for Pr and return the value of **AVERAGE** annual energy. To test your code, if you use *A = 20, r = 15% , H= 1250 as inputs and assume Pr= 0.75*, you should get *2812.5 KWh*.

Use the following cout statement to output the value in main function:

cout << Solar Energy << ”KWh” << endl;

* 1. Once you have the peak energy output calculation working, create another function **“printAverageEnergy”** that prints all the average energy values for efficiency values from 0.10 to 0.35 by 0.5 step. The function should take user input for A, r, H.

Use the following cout statement to output the values in function:

Cout << “Solar Energy for” << Pr value << “KWh” << endl

* 1. If it's given that one house uses 901 kWh per year of energy, then calculate the number of houses that one solar panel can support for a year. Create a function **“getNoOfHousesStarted”** that takes in peak energy of solar panel and energy required by one house and then returns the number of houses it will support. You need to take user input for A, r, H and default value for Pr and calculate the peak energy as done in point ‘a’.

Use the following cout statement to output the value in main function:

       cout << “No. of houses: “ << ‘house number in int’ << endl;

Note: Redundant code is discouraged.