Homework 1: Hello World and Translating Python Code to C++ CSCI 60

Due 11:59pm Friday, September 26, 2025 (Part A) and Wednesday, October 1, 2025 (Part B) Krehbiel

Overview. This first programming assignment is split into two parts with two different submission deadlines. Part A is just to make sure your development environment is up and running and to give me a little bit of information about you. If your on-time submission compiles, runs, and meets the below specifications, you will get full credit. Part B has you write a program to play a version of the recently popular game Wordle. Your program will select a random secret word from the provided wordbank and then solicit and process several user guesses. The goal is to focus on C++ syntax, so there's very little problem-solving this week.

Part A - Instructions. You have no starter files for Part A, but you may just download `hello.cpp` to get the boilerplate code. If you do, be sure to rename it `helloworld.cpp`. Modify the main function to output two paragraphs of text to the console.

- The only include directive you need is `iostream` and it's a good idea to use the `std` namespace. (These are already included in `hello.cpp`.)
- Separate your paragraphs with `endl`. Use `\t` if you want to indent the second paragraph.
- Use multiple `cout` commands and/or chained insertion operators << spread across separate lines to make sure that your source code doesn't go past 80 characters per line.
- In the first paragraph, tell me your name, your hometown, your year in school, and something you enjoy outside of academics.
- In the second paragraph, tell me about your background with computer science, what you see as your strengths and weaknesses in programming so far, and what study strategies you are planning on to help you do well in CSCI 60.
- Include a file comment at the top of the file. Your comment should include the following:

  A description of the program, e.g., "A simple program that outputs text for HW1 Part A."

  A statement that you know where this file lives on your computer and you compiled and ran it by command line as done in class to confirm there are no compilation or runtime errors.

  Your name and the date.

  A description of any outside resources used, or a statement that you didn't use any.

Submit your program file named $\boxed{\texttt{helloworld.cpp}}$ to Gradescope (Camino will have a link) before the Friday night deadline. An autograder will tell you if your submission meets the technical specifications, and it will not work if you use a different filename or have an error in your code. You may submit as many times before the deadline as you like; only your last submission will be graded.

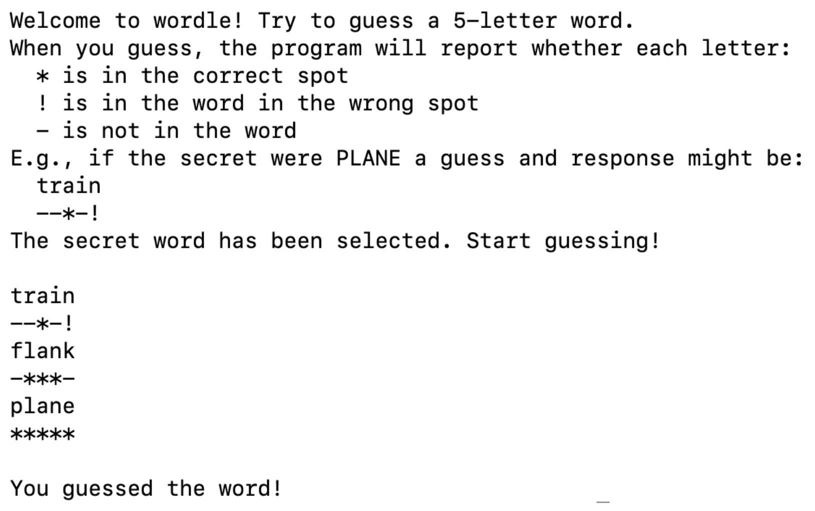
Part B - Starter files. You have three starter files for Part B. They include a file `wordle.py` that fully implements this program in Python. You do not have to modify, run, or even open this Python program; it just serves as a reference for you to make the programming task more straightforward.

Your C++ implementation that you submit for Part B will go in `wordle.cpp`. Open the starter file and note that it includes a pre-implemented function that handles the file-reading necessary to pick a random word; that means you don't have to do this!

Finally, a plaintext file `wordbank.txt` contains a lexicon of words that your program will read and use to pick a secret word. You don't need to do anything for this file except put it in the same folder on your computer as `wordle.cpp` so you can test your game.

Review class notes, the starter code, and relevant sections of Chapters 1-5 to finish implementing the Wordle game; the specifications and a description if how to break the problem into manageable tasks are detailed below. Use the Python implementation as much or as little as it is useful to you. Upload your completed `wordle.cpp` file to Gradescope by the Wednesday night deadline. The rest of this handout has additional descriptions about how the game works and splits the task into milestones with detailed suggestions about how to approach your assignment.

Part B - Milestone 1: Understand the game and run the starter code. Before writing your own code, make sure your files are organized and that you can run the code that has been provided to you so you can get familiar with how your final program should work. Once the starter files are downloaded in the same directory, navigate there and run the programs. If you have Python on your machine (it's fine if you don't), you can play the game as follows:



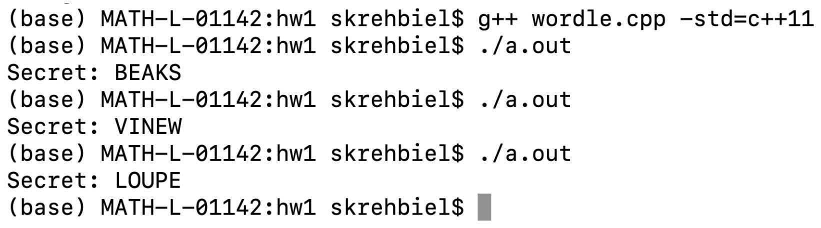
```
Welcome to wordle! Try to guess a 5-letter word.
When you guess, the program will report whether each letter:
  * is in the correct spot
  ! is in the word in the wrong spot
  - is not in the word
E.g., if the secret were PLANE a guess and response might be:
  train
  --*-!
The secret word has been selected. Start guessing!

train
--*-!
flank
-***-
plane
*****

You guessed the word!
```

The objective is to guess a secret 5-letter word, and each guess gives information about the secret word. For each letter in the guessed word, the program should report whether the letter is or is not in the secret word and where it is or is not in the correct position if it's in the word. The user can make up to six guesses. There are a few differences between this game and the popular version owned by the New York Times:

- This is a purely text-based game, so we don't have nice colors to convey the information.
- If you are used to Wordle, you'll know that if the secret word is `PLANE` and you guess `HELLO`, only one of the L characters will be highlighted, indicating that there is only one L in the secret word. For simplicity, we process each letter individually, so our program would respond with `-!!!-`, indicating that an `E` in `HELLO` is present in the secret word but not in the second spot, and an `L` is present in the secret word, but not in the third or fourth spot.
- The game should be case-insensitive, so a guess `hello` will be processed the same as `hElLo`, but you may assume that users don't enter non-letter characters. E.g., whatever your program says if the user guesses `Code$` is fine.
- Your program must check that the user enters a 5-character guess and reprompt until they do, but the program shouldn't check that the word is in the word bank. E.g., `PYTHON` would not be accepted as a guess because it is 6 letters, but `CppPy` would because it is 5 letters even though it isn't a valid word.
- The word bank we are using includes some pretty funky words, which makes it harder. Finally, your program picks a random word every time you run it rather than changing the secret once per day, so you can play as much as you want!

After you're familiar with the working Python code (not necessarily the syntax), check out `wordle.cpp` and scan the code that is already there. The file-reading is already done for you, but looking at the code might help you start to see differences and similarities between Python and C++ syntax. In particular, note that Python has a nice built-in `upper()` function but in C++ we have to use a loop to go through and capitalize each word bank letter character-by-character. You will have to do this with user guesses for Milestone 3, too!

Make sure you can run the code by first compiling with `g++` and then executing with `./a.out`. Currently the main file just calls the pre-written `getWord()` function and prints the returned string to the console. You'll eventually remove the `cout` statement, but first run the code a few times to see some random words.



```
(base) MATH-L-01142:hw1 skrehbiel$ g++ wordle.cpp -std=c++11
(base) MATH-L-01142:hw1 skrehbiel$ ./a.out
Secret: BEAKS
(base) MATH-L-01142:hw1 skrehbiel$ ./a.out
Secret: VINEW
(base) MATH-L-01142:hw1 skrehbiel$ ./a.out
Secret: LOUPE
(base) MATH-L-01142:hw1 skrehbiel$ 
```

Part B - Milestone 2: Define stub functions. Define stub functions analogous to the other helper functions in `wordle.py`. To do this, first write a function declaration *above* the `main` function that will call it. Note that unlike in Python, you have to declare the type of value returned by a function and the type of its arguments. Specifying these types before the function is used is the reason we declare functions. Then *below* the `main` function, define the function, i.e., specify what code gets executed. For a stub function, you just want something that compiles. For example, your entire function body could just be `return "GUESS";`. Finally, call your function from `main` and output its return value to make sure it works as expected. Once you're clear on the syntax for function declaration, definition, and calling, start to implement the logic!

Part B - Milestone 3: Do all the real work! Implement all the functionality present in `wordle.py`. Testing regularly makes it easier to find bugs, so compile and run your code every few lines you write.

Part B - Milestone 4: Test extensively. Make sure all your prompts are exactly as in the Python implementation, and make sure that your code is well-documented and follows good readability guidelines before submitting. Most of your grade is based on the functionality of your code, but part of it is based on good style, with some requirements and guidance provided below.

Submission guidelines. The following guidelines apply to all submissions and can be used as a checklist.

1. Make sure you are submitting all the correct files with correct filenames.
2. Each file needs a comment with a brief file description, your name/date, and a collaboration statement.
3. Comment your code throughout development, and scan it a final time before submitting to remove unused code and comments. Make readability improvements based on C++ style guidelines, including:

   good documentation (short program description and comments for functions and complex code),

   sensible decomposition for efficiency and modularity (usually $\leq 10 - 15$ lines per function),

   appropriately scoped variables (only constants should be global), and

   descriptive variable names (use camelCase).
4. Check that your code compiles and runs correctly before you submit, and reread the *entire* assignment handout to make sure you've met all specifications.
5. Your last submission is the only one that will be graded, so include all necessary files there.