

Project Starter Doc: Spelling Bee

- [1. Read the Project Doc](#)
- [2. Create Example Guesses and Outputs](#)
- [3. Formatting Guesses](#)
- [4. Checking for invalid guesses](#)
- [5. Checking for Pangrams](#)
- [6. Using Functions](#)
- [7. Play multiple rounds](#)
- [8. Meet with a TA](#)

1. Read the [Project Doc](#)

It is very important that you understand all aspects of this project. Read through the document at least twice so you have a full understanding of what is being asked.

2. Create Example Guesses and Outputs (5 pts)

Like in the Rock-Paper-Scissors ELO Rating Lab, we will start by coming up with some example inputs and outputs.

Consider the case where the spelling bee letters are as follows: **WRAMIGN**

The required letter is **M**

Scenario	Example Guess	Score
The guess does not have enough letters		
The guess does not use the required letter		
The guess uses an invalid letter		
The guess is valid with 4 letters		
The guess is valid with 5 or more letters		
The guess is a pangram		

3. Formatting Guesses (5 pts)

For this problem, your starter code will handle getting the spelling bee letters. However, the starter code letters are all capitalized.

When asking the user for their guess, you will be expected to handle capital and lowercase letters. When a user types in a guess with lowercase letters, your code should convert all letters to uppercase **and store them in a variable**. You should use the [toupper](#) function to do this.

```
string guess = ...; // assume this has data
string uppercaseGuess; // create this using the for loop
for (int i = 0; i < guess.length(); i++) {
    // convert every char to an uppercase letter
}
```

4. Checking for invalid guesses (10 pts)

Part of why this project seems complex is that we need to check for lots of different things. Let's break down the problem by first focusing on invalid guesses.

Assume we have the following variables

```
string guess;          // Stores the user's guess for each round
string validLetters;  // Stores the valid letters
char requiredLetter; // Stores the required letter
```

a. How can we check if the guess is not long enough?

```
if (
    // Print: this is invalid
}
```

There are a couple checks that will require us to see if a character is part of a string. To do this, I've included a helper function in your code that we made together in lab. Look at this function

```
bool containsChar(string word, char letter) {
    for (int i = 0; i < word.length(); i++) {
        char currentLetter = word.at(i);
        if (currentLetter == letter) {
            return true;
        }
    }
    return false;
}
```

You should use this function to check if a string of letters has the required letter.

b. How can we check if the guess has the required letter?

Hint: you can call the containsChar function

```
if (
)
```

```
// Print: this is invalid  
}
```

c. How can we check if the guess uses only valid letters?

Hint: you can call the containsChar function

```
for (int i = 0; i < guess.length(); i++)
```

Note: if you google how to find if a string contains a char, the C++ string function [find](#) will likely pop up. For this lab, you should not use the find function. Instead, use the **containsChar** function in your project code.

5. Checking for Pangrams (5 pts)

Read the project spec to understand how scoring works in Spelling Bee. A pangram uses every valid letter at least once and is worth extra points. A pangram is worth the number of letters in the guess + 7 extra points.

How can we check if a guess is a pangram?

```
// Your code here
```

6. Using Functions (10 pts)

You will make the following functions in order to complete this assignment:

- bool isValidGuess(string guess, string validLetters, char required letter)
- string getGuess(string validLetters, char required letter)
- bool isPangram(string guess, string validLetters)
- int scoreGuess(string guess, string validLetters)

The **getGuess** function should ask the user for input until their guess is valid. You should use the **isValidGuess** function to check if their guess is valid. Once the guess is valid, the **getGuess** function can return the guess.

Write your code for the getGuess function.

Hint: you should call the **isValidGuess** function

```
// Write code for the getGuess function - it does not need to be perfect  
string getGuess(string validLetters, char required letter) {  
  
    return????  
}
```

The **scoreGuess** function should return the score of the word. Reference your examples from part 1 to understand scoring. You should use the **isPangram** function to check if the guess gets extra points.

Write your code for the scoreGuess function.

Hint: you should call the **isPangram** function

```
// Write code for the getGuess function - it does not need to be perfect
int scoreGuess(string guess, string validLetters) {
    return ****
}
```

What do we want to do for every round of the game? Which functions should we call?

Hint: you will want to call the **getGuess** and **scoreGuess** functions.

```
// Call the functions in order of the game flow
```

7. Play multiple rounds (5 pts)

So far, we have a solid plan for how to deal with one guess. But now we want the user to keep making guesses until they want to stop. We know they want to stop when they enter the guess “END”

How would you set up your while loop in your main method?

Hint: Use your work from part 6 to add code to the inside of the while loop

```
// Keep playing the game while...
string guess;
while (<What condition should go here?>) {
    // What do we do for every round of the game?
}
```

8. Meet with a TA

Once you finish this starter doc, please book an appointment with a TA to go over your ideas. **Appointments must happen by the end of day Wednesday, November 12th** to get credit. Meet early so you can start coding!