**BearING**

How to Code the 9 Helper Functions

**Introduction:**

In this pseudocode, I will go through 9 helper functions in the BearING program and describe how their uses and how to code them. All code is assumed to be written in C++.

**Standard advice for each function:**

First, we must declare the function. Each function will have a type that needs to be typed. Use the type of the output in the bulleted list to determine this. Then write the function's name with parameters asking for the input(s) listed in the bulleted lists.

**Functions:**

**is_vowel:**

- Purpose: Checks if the current character under the reading head is a vowel.
- Input: One character
- Output: True if it is a vowel, false if it is not.

To complete the task, we can use a simple if statement to ask if the character input through the argument is “==” (equal) to a, e, i, o, or u. (use || in between each check) In the true section which is the one right after the vowel check, return true. In the false section, (after the else) type return false;.

**Is_consonant:**

- Purpose: Checks if the current character under the reading head is a consonant.
- Input: One character.
- Output: True if it is a consonant, false if it is not.

To complete the task, we can use a simple if statement to ask if the character input through the argument is “!=” (equal) to a, e, i, o, or u. (use || in between each check) In the true section which is the one right after the consonant check, return true. In the false section, (after the else) type return false;.

**ends_with_double_consonant:**

- Purpose: Checks if the string input has two characters and ends with two consonants.
- Input: One string
- Output: Out puts true if the word ends with the same constant twice, false if it does not. Returns false if the word is less than 2 characters.

First, let's find the length of the string using the. length() function provided by C++ and turn it into a local variable for the function. ( We will call this variable wordlength) Too cover all grounds, lets use an ifstatement that simply continues the function if wordlength is >= 2 and returns false if not. Then, we can simply go to word[wordlength – 1] and word[wordlength – 2] to find the last two letters. Use an if statement to compare the words against the is_consonant function and if it is true return true, and if it is false return false.

**ends_with_cvc:**

- Purpose: Checks if the word has three or more characters and ends with a consonant-vowel-consonant pattern.
- Input: One string
- Output: Returns true if the word ends with a consonant-vowel-consonant pattern, false if it does not.

First, let's find the length of the string using the. length() function provided by C++ and turn it into a local variable for the function. ( We will call this variable wordlength) Too cover all grounds, lets use an ifstatement that simply continues the function if wordlength is >= 3 and returns false if not. Similarly to ends_with_double_consonant, we will return true if word[wordlength – 1] and word[wordlength – 3] are consonants and word[wordlength – 2] is a vowel. In any other case return false.

**contains_vowel:**

- Purpose: Checks if the word contains a vowel.
- Input: One String
- Output: Returns true if the word contains a vowel, false if it does not.

Here we will need our first loop! First, let's find the length of the string using the. length() function provided by C++ and turn it into a local variable for the function. ( We will call this variable wordlength) Let's use a while loop to increment through all the characters in the word. In this case I will simply use while( i < wordlength) and  have a simple if statement to return true if the character at i (wordlength[i]) inside of the is_vowel function returns true. If it doesn't simply return false.

**count_consonants_at_front:**

- Purpose: Checks for the amount of consecutive consonants at the front of the word.
- Input: One string
- Output: Outputs the amount of consecutive consonants at the front of the word as an integer.

Once again we will loop through the word length.First, let's find the length of the string using the. length() function provided by C++ and turn it into a local variable for the function. ( We will call this variable wordlength) In this case I will simply use while( i < wordlength) and use a if statement to either increment i if teh current character at i (wordlength[i]) inside of is_consonant is true and return i if it is false.

**count_vowels_at_back:**

- Purpose: Checks for the amount of consecutive vowels at the back of the word.
- Input: One string
- Output: Returns the amount of consecutive vowels at the back of the word as an integer.

Now we will increment backwards with a while loop. To do this let's find the length of the string using the. length() function provided by C++ and turn it into a local variable for the function. ( We will call this variable wordlength)

We will also need to count the vowels so make a variable of count = 0. Now let's set i = wordlength – 1. (I know a for loop is probably better for this but I have already been using while loops) Now set the while loop condition to keep going while i > 0. This means the increment is going to b e i—not i++ so be careful.

Now let's set up an if statement that simply runs the funnction is_vowel with the word at the current index as the argument. If it returns true we will increment i again and add one to count. If it returns false we will return count.

**ends_with:**

- Purpose: Checks if a word's and a suffix's strings are empty, and if not, checks if the word is empty and if the suffix is not empty. Finally, checks if the word ends with the suffix.
- Input: Two strings. The word and the suffix.
- Output: Firstly, returns false if suffix is longer than word. True if both strings are empty, false if the word is empty and the suffix is not, and true if the word ends with the suffix. False if none of the conditions are met.

Before anything, we need to create an if statement that returns false if the length of the suffix is longer than the length of the word. (use the .length() function) Two more if statements wil be needed. For the next one, if the word is equal to zero and the suffix isnt, return false. Then make another that checks if they are both equal to zero, if true, return true. In an else statement that follows, create a loop that increments through the length of suffix and checks the last suffix length amount of letters in the word to

see if they are equal. You can do this by going to the index of the word at the space of the words length, minus the suffix length plus the current index in the suffix fromt the for loop. If any of them aren't equal, return false. Otherwise, return true.

**new_ending:**

- Purpose: Creates a new string from the candidate string by removing the last suffix.length() characters and appending the new suffix.
- Input: A string, a number, a string named candidate, the suffix length, and replacement in that order.
- Output: Returns the new word.

For this function we will assume the suffix length is always less than or equal to the length of the candiate. We do however need to return an empty string if the candidate length is == to 0. In the else statement following, set your new word variable equal to the candiate minus the length of the suffix using the substr() function. Finally, add the the replacement to the the end of the word and return.

Congratulations!