Assignment 0

Do before Assignment 1. Not for marks

In this assignment, you will make a simple word matching program. In this program, the user enters a word with one or more letters replaced by underscores (`'_'`) and the program prints a list of matching words in the dictionary. This process repeats until the user enters `"q"` as a word.

Suppose for example that the dictionary contains:

```
a
act
apple
box
breeze
cat
cot
cow
cut
dog
elf
freeze
gem
```

An example run of the program using this dictionary is given below. The exact input and output format are shown, with user input in bold face.

```
Welcome to the Word Match program!

Enter a word with underscores: c_t
cat
cot
cut

Enter a word with underscores: __ee_e
breeze
freeze

Enter a word with underscores: abcd_fgh

Enter a word with underscores: q

Goodbye!
```

The purpose of this assignment is to review the topics covered in CS 110, including strings, loops, arrays, functional decomposition, and file input. For Part A, you will write a `main` function to read in a word entered by the user and then print it out again. For Part B, you will find matching words. For Part C, you will change the interface to allow the user to test multiple words. For Part D, you will load in a much longer list of words from a file.

Part A: Read in a Word from the User

In Part A, you will create in a simple program that reads in a word from the user and print it again.

Perform the following steps:

1. Create a new project in the programming environment of your choice.
    - If you are using Visual Studio, go to File > New Project…. Select Win32 Console Application, give it a name (such as `Assignment0`), a location, and make sure "Create Directory for Solution" is NOT checked. Click OK and a different popup window will appear. Click the "Next >" button on the popup window, and make sure "Console application" is selected. For the options below, check "Empty project" and uncheck all the others (greyed out is OK too).
2. Create a new file named `Main.cpp`.
    - If you are using Visual Studio, use the Solution Explorer to select Project > Add New Item…. Choose `C++ File` and give it the name `Main.cpp`.
3. At the top of the file, add an `#include` for `iostream`, which is used for screen input and output:

    ```
    #include <iostream>
    ```

    Also include `string`, which, not surprisingly, is used for strings.
4. Add a `using` directive for the `std` namespace:

    ```
    using namespace std;
    ```
5. Add a `main` function that prints "`Hello World!`".

    ```
    int main ()
    {
          cout << "Hello World!" << endl;

          return 0;  // main function should end with this
    }
    ```
6. Compile and run your program. From now on, do this regularly.
    - If you are using Visual Studio, you can compile and run your program by pressing the green arrow (or `[F5]`). However, the program will run in a new window which closes as soon as it finishes, so you don't get to see the output. One way around this is to use the Debug > Start Without Debugging (`[CTRL]+[F5]`) command instead. Alternatively, you can include `cstdlib` and add a call to `system("PAUSE");` immediately before the return statement.
7. Change the message to say "`Welcome to the Word Match program!`".
8. Then print a blank line, followed by "`Enter a word with underscores: `" without a newline. We refer to a line like this as a ***prompt*** to the user.

9. Create a `string` variable and read the word the user typed into it:

```
string user_word;
cin >> user_word;
```

10. Print out the word you read in:

```
cout << "You entered \"" << user_word << "\"" << endl;
```

Note that `\"` is used to put a quotation mark inside a string.

Part B: Find Matching Words

In Part B, you will add a list of words and print the ones that match the one the user entered.

Perform the following steps:

11. Add an array of three `string`s at the beginning of your `main` function:

```
string dictionary[4] =
{
      "a",
      "act",
      "apple",
      "box",
};
```

12. Instead of printing the user word, print all the words in the dictionary using a `for` loop:

```
for(unsigned int i = 0; i < 4; i++)
{
      cout << dictionary[i] << endl;
}
```

    The program should now print `a`, `act`, `apple`, and `box` on four separate lines. The variable `i` is called a ***loop counter*** because it counts how many times the loop has run.

13. Add a constant to represent the number of words in the dictionary. It should be declared before the array.

```
const unsigned int DICTIONARY_SIZE = 4;
```

    It is traditional to name constants in `ALL_CAPITALS_WITH_UNDERSCORES`. This makes it easy to see at a glance which identifiers are constants and which are variables.

14. Change the array size and maximum value for the loop counter to use the `DICTIONARY_SIZE` constant instead of the literal `4`. The program should run as before.

15. Add more words to your dictionary: `"breeze"`, `"cat"`, `"cot"`, `"cow"`, `"cut"`, `"dog"`, `"elf"`, `"freeze"`, and `"gem"`. Increase the `DICTIONARY_SIZE` constant to `13` match.

16. Add an `if` statement inside the `for` loop to only print words from the dictionary if they are they are the same length as the user word:

```
{
      if(user_word.length() == dictionary[i].length())
      {
            cout << dictionary[i] << endl;
      }
}
```

    The "length" of a `string` is the number of characters in it. Test your program with a few words of different lengths as input.

17. Add another check to make sure that the user word has a length greater than `0`. Join the two checks in one `if` statement using `&&` (meaning "and").

18. Add another `if` statement inside the first `if` statement to make sure that the first character is the same in both words.

```
{
      if(user_word.at(0) == dictionary[i].at(0))
      {
            cout << dictionary[i] << endl;
      }
}
```

    Don't ever check the value of a character of a `string` unless you already know that the string has at least that many characters. If the character doesn't exist, the program will crash. Remember that characters are numbered starting at `0`, so to look at character `n`, the length of the string must be strictly greater than `n`.

19. Add another check to the inner `if` statement to allow the first character of the user word to be an underscore (`'_'`). Join the two checks in one `if` statement using `||` (meaning "or"). A single character is shown in a C++ program in single quotes, whereas a string is shown in double quotes.

20. Add another `for` loop around the inner `if` to check every character of the two words instead of just the first character. Make sure to use a new variable name (such as `j`) for the loop counter. Increase the indenting of the inner `if` statement. Use `j` instead of `0` in three places (two added in step 18 and one added in step 19). Test your program. It should produce a lot of output.

The program is producing too much output because it is printing the dictionary word whenever any letter matches. Instead, it should print the word once if all of the letters match. To fix this, we will use a variable named `matched` to keep track of how many characters match between the user word and the dictionary word. For example, if the user word is `_at` and the dictionary word is `cut`, then `matched` should eventually get a value of `2` (one for the `_` matching `c` and the other for `t`

matching `t`). We will use a loop to go through all the letters in the user word. It should increment the `matched` variable if the letter in the user word matches the corresponding letter in the dictionary word. After the end of the loop, if the value in the `matched` variable is the same as the length of the dictionary word, then we will print the word.

21. Start by declaring a variable named `matched` of type `unsigned int` immediately before the inner `for` loop and giving it an initial value of `0`.

22. Move the following statement to a point immediately after the inner `for` loop:
```
cout << dictionary[i] << endl;
```

23. Inside the `if` statement in the inner `for` loop, increase the `matched` variable by 1 whenever the characters match. Three legal syntaxes to do so are as follows (pick any one):
```
matched = matched + 1;
matched += 1;
matched++;
```

24. Add an `if` statement after the inner `for` loop that checks whether the number of matching characters is the same as the length of the word. You can check the length of either the user word or the dictionary word, since we already know they are the same. Move the `cout` command to print the dictionary word (i.e., the one mentioned in step 22) to inside this `if`. Test the program. It should now print the right matching words.

Part C: Checking Multiple Words

In Part C, you will improve the interface of your program so that it will check multiple words instead of just one.

Perform the following steps:

*Steps 25 to 31 will reorganize the program without changing its behaviour, which is called **refactoring** the program. This will make it easier to add to.*

25. Create a new function named `checkWord` after the `main` function:

```
// print dictionary_word if all its characters match
// those in user_word
void checkWord (string user_word, string dictionary_word)
{
      ...  // replace this line in step 27
}
```

Notice that when you define a function, you put the types with the names of the parameters.

26. Add a function prototype for `checkWord` above the `main` function but below the `#includes` and the `using` statement.

```
void checkWord (string user_word, string dictionary_word);
```

In CS 115, we will show the names of the parameters in the prototype exactly the same as in the top line of the function itself. (Some CS 110 instructors suggest leaving out the names of the parameters.)

27. Copy the contents of the outer `for` loop (i.e., everything inside of the `for` loop but not the first line with "`for`" or the braces `{}`) to replace the three dots (**. . .**) line in the `checkWord` function. Also update the indenting.

    - In many Windows programs intended for editing plain text, you can indent text by selecting it and pressing `[TAB]`. Decreasing the indent might be `[SHIFT]+[TAB]` or `[BACKSPACE]`. You can also often go to a specific line by number using `[CTRL]+[G]`.

28. Change every occurrence of `dictionary[i]` in the `checkWord` function to `dictionary_word`, which is the name of the parameter in this function.

29. Replace the contents of the outer `for` loop in your `main` function with a call to the `checkWord` function:

```
// check every word in dictionary for a match to user_word
for (unsigned int i = 0; i < DICTIONARY_SIZE; i++)
{
      checkWord(user_word, dictionary[i]);
}
```

    Notice that when you call a function, you do <u>not</u> put the types with the names of the parameters. Test your program, it should work as before.

30. Add a new function named `checkDictionary` before the `checkWord` function. It should check the user word against every word in an array of strings. Move the `for` loop from the `main` function to this new function. The function prototype should be:

```
void checkDictionary (string user_word,
                      string dictionary[],
                      unsigned int dictionary_size);
```

    - Note that the array in the function prototype does not have a size. You can put a size, but the compiler just ignores it, so it is less confusing if you leave it out.

31. Change your main function to call the `checkDictionry` function where it used to execute the `for` loop. You will need three arguments in the call. To pass the array as an argument, just give its name with no brackets. It is fine to pass a constant as an argument. The program should work as before.

Steps 32 to 35 will change the program to handle a series of input words instead of only one.

32. Add an `if` to your `main` function that checks whether the user entered a word other than "`q`". You can compare strings with `!=`, just as you do with numbers. The `if` should control the call to `checkDictionary`.

- Recall that in C++, a character is surrounded by single quotes (e.g. `'q'`), while a string is surrounded by double quotes (e.g. `"q"`). The two are not interchangeable. Here we need to compare to a string.

33. After the call to `checkDictionary` in the `if` statement in the main function, add code to print a blank line, prompt the user for a new word, and read the new word. Instead of declaring a new variable, read the new word into the same `user_word` variable as before.

34. Replace your `if` statement with a `while` loop. The syntax is the same as an `if` statement, except that it says `while` instead of `if`. Whereas an `if` statement runs once and then stops, a `while` loop will keep running over and over as long as the check is still true.

35. After the end of the `while` loop in the `main` function, print a blank line followed by `"Goodbye!"`. Test your program thoroughly with input such as `_ _eeze`, `c_ _`, and `_ _ _`.

Part D: Load Words from a File

In Part D, you will load the dictionary from a file.

Perform the following steps:

36. Download the `shortdictionary.txt` file from UR Courses.

37. Add an `#include` for `fstream`, near the beginning of your `Main.cpp` file. The `fstream` library is used for file input and output.

38. After the declaration of the dictionary array, add and initialize a variable to read from a file:

```
ifstream fin;
fin.open("shortdictionary.txt");
```

39. Check to see whether the file was opened correctly:

```
if(!fin)
{
      cout << "Error: Could not open file" << endl;
}
```

    The most common reason for a file to not open correctly is because it doesn't exist. This often means that you spelled the name wrong or you did not match upper/lower case or you put the file in the wrong folder. The program should work as before.

40. Add a `for` loop to read values from the file into the `dictionary` array. Use `DICTIONARY_SIZE` to specify the loop count. The syntax for reading from a file is the same as for reading user input, except that you use `fin` instead of `cin`. If your loop counter is named `i`, then you want to read into `dictionary[i]`.

41. After reading in the dictionary, close the file using the `close` function. The syntax is the same as for the `open` function, except that the `close` function takes no parameters.

42. Remove the array initialization for the dictionary array, but keep the declaration itself. The program should continue to run as before. If you have any problems, try printing the dictionary words as you read them.

43. Download the `dictionary.txt` file from UR Courses and load that instead. It contains `58110` words, so you will have to adjust `DICTIONARY_SIZE`. Compile and run the program. What happens?

44. If your program behaves the same way as mine, it crashes at start-up with an incomprehensible error message (e.g., "stack overflow"). The crash happens because the dictionary array took more memory than is available for ***local variables*** (i.e. variables declared in functions). We can fix this by making the dictionary a ***global variable*** (i.e. a variable shared between all functions). To do so, move the declaration of `dictionary` to above the start of the `main` function. You will also have to move `DICTIONARY_SIZE` so that it is still above `dictionary`.

45. Compile and run your program. It should run as before, except that it prints many more words.

46. Try entering a word with uppercase letters, such as `C_t`. No words will appear. This is because C++ is ***case-sensitive***, which means it treats uppercase and lowercase letters as different. All the words in the dictionary are in lowercase, so none of them match.

47. Add an `#include` for the `cctype` library to your program. Add a `for` loop at the beginning of the `checkDictionary` function to apply the `tolower` function to every character in the `user_word` variable. Not surprisingly, this function converts letters to lowercase. The key line will be something like this:

```
user_word.at(i) = tolower(int(user_word.at(i)));
```

48. Try entering a word with uppercase letters, such as `C_t`, again. This time it should work. Test your program as you like. You are done!