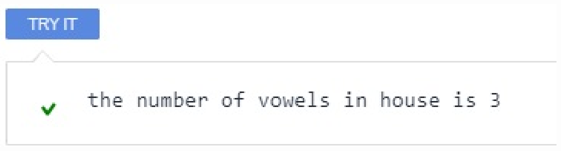
Encapsulation Lab 1

This lab will focus on building on the idea of a “black box.” The term “black box” is used to describe a system in which the internal workings are hidden from the user. In fact, the user is not expected to know how the system works; they only need to know how to use it. Encapsulation allows you to create classes that are black boxes. Previously, we created a class `Words` that took in a string and printed the number of vowels in that string.



```
the number of vowels in house is 3
```

.guides/img/Encapsulation/BlackBox3

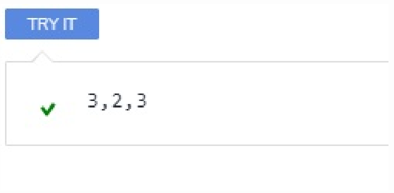
In this lab, we are going to create a function that gives the user the ability to put in a vector of strings and our program will print out the number of vowels in each string within that vector.

```
  //add code below this line

    vector<string> list = {"house", "cake", "pancake"};
    Words vowels(list);
    vowels.CoutStrings();

  //add code above this line
```

If you run the code above successfully in `main`, you should expect to see `3,2,3` as the output.



```
3,2,3
```

.guides/img/Encapsulation/BlackBox4

In this scenario, we are going to modify our previous `Words` class so that it can include a vector. All of the following code will go into the `class definitions` field.

```
class Words {
  private:
    vector<string> list_of_words;
};
```

We don’t need to reinvent the wheel; we previously worked on a few functions that told us how many vowels exist in a specific string. We are going to again re-use those functions. These are functions that the user does not need to interact with. Thus, they can be `private`, essentially making them act as part of the “black box.” Now, add the following `private` functions into the `Words` class.

```
bool IsVowel(char ch) {
  ch = toupper(ch);
  return (ch=='A' || ch=='E' || ch=='I' ||
          ch=='O' || ch=='U');
}

int CountVowels(string str, int n) {
  if (n == 1) {
    return IsVowel(str[n-1]);
  }
  return CountVowels(str, n-1) + IsVowel(str[n-1]);
}
```

Now that all of the data that we don’t want the user to manipulate are properly encapsulated as `private`, we can start working on our `public` members such as the constructor `Words` and the function `CoutStrings`. In the example above, we can see our constructor takes in one attribute.

```
public:
  Words(vector<string>& n){
    list_of_words = n;
  }
```

Another piece of information that we have from the `main` function is that the function `CoutStrings` is used. This function has no arguments, but is accessible to the user. This function also has several other tasks:

1. It iterates through a given vector
2. Counts the number of vowels in each given string
3. Creates a vector storing the vowel sizes
4. Prints out the output in a defined fashion

```
void CoutStrings() {
   vector<int> vowel_sizes;
   int size;
   for (auto a : list_of_words) {
     size = CountVowels(a, a.length());
     vowel_sizes.push_back(size);
   }
   for (int i = 0; i < vowel_sizes.size(); i++) {
     if (i == vowel_sizes.size()-1) {
       cout << vowel_sizes.at(i) << endl;
     }
     else {
       cout << vowel_sizes.at(i) << ',';
     }
   }
 }
```

▼ **Full Code**

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Words {
  public:
    Words(vector<string>& n){
      list_of_words = n;
    }

    void CoutStrings() {
       vector<int> vowel_sizes;
       int size;
       for (auto a : list_of_words) {
         size = CountVowels(a, a.length());
         vowel_sizes.push_back(size);
       }
       for (int i = 0; i < vowel_sizes.size(); i++) {
         if (i == vowel_sizes.size()-1) {
           cout << vowel_sizes.at(i) << endl;
         }
         else {
           cout << vowel_sizes.at(i) << ',';
         }
       }
    }

  private:
    vector<string> list_of_words;

    bool IsVowel(char ch) {
      ch = toupper(ch);
      return (ch=='A' || ch=='E' || ch=='I' ||
              ch=='O' || ch=='U');
    }

    int CountVowels(string str, int n) {
      if (n == 1) {
        return IsVowel(str[n-1]);
      }
      return CountVowels(str, n-1) + IsVowel(str[n-1]);
    }
};

//add class definitions above this line


int main() {

  //add code below this line

    vector<string> list = {"house", "cake", "pancake"};
    Words vowels(list);
    vowels.CoutStrings();

  //add code above this line

  return 0;

}
```

Challenge

Can you modify the code so that it prints the number of non-vowels instead of vowels? For example, `vector<string> list_of_words = {"house", "cake", "pancake"};` will have the following output: `2,2,4`.

▼ **Possible Solution**

```
#include <iostream>
```

```
#include <vector>
using namespace std;

//add class definitions below this line

class Words {
  public:
    Words(vector<string>& n){
      list_of_words = n;
    }

    void CoutStrings() {
       vector<int> counts;
       int size;
       for (auto a : list_of_words) {
         size = CountNonVowels(a, a.length());
         counts.push_back(size);
       }
       for (int i = 0; i < counts.size(); i++) {
         if (i == counts.size()-1) {
           cout << counts.at(i) << endl;
         }
         else {
           cout << counts.at(i) << ',';
         }
       }
     }

  private:
    vector<string> list_of_words;

    bool IsNonVowel(char ch) {
      ch = toupper(ch);
      return !(ch=='A' || ch=='E' || ch=='I' ||
               ch=='O' || ch=='U');
    }

    int CountNonVowels(string str, int n) {
      if (n == 1) {
        return IsNonVowel(str[n-1]);
      }
      return CountNonVowels(str, n-1) + IsNonVowel(str[n-
        1]);
    }
};

//add class definitions above this line
```

```
int main() {

  //add code below this line

    vector<string> list = {"house", "cake", "pancake"};
    Words nonvowels(list);
    nonvowels.CoutStrings();

  //add code above this line

  return 0;

}
```

Encapsulation Lab 2

In this next lab, we'll continuing building our `Words` class to include a getter and setter. These should be `public` to enable the `main` function to call them.

▼ **Full Code**

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Words {
  public:
    Words(vector<string>& n){
      list_of_words = n;
    }

    void CoutStrings() {
       vector<int> vowel_sizes;
       int size;
       for (auto a : list_of_words) {
         size = CountVowels(a, a.length());
         vowel_sizes.push_back(size);
       }
       for (int i = 0; i < vowel_sizes.size(); i++) {
         if (i == vowel_sizes.size()-1) {
           cout << vowel_sizes.at(i) << endl;
         }
         else {
           cout << vowel_sizes.at(i) << ',';
         }
       }
     }

  private:
    vector<string> list_of_words;

    bool IsVowel(char ch) {
      ch = toupper(ch);
      return (ch=='A' || ch=='E' || ch=='I' ||
              ch=='O' || ch=='U');
    }

    int CountVowels(string str, int n) {
      if (n == 1) {
        return IsVowel(str[n-1]);
      }
      return CountVowels(str, n-1) + IsVowel(str[n-1]);
    }
};

//add class definitions above this line


int main() {

  //add code below this line

    vector<string> list = {"house", "cake", "pancake"};
    Words vowels(list);
    vowels.CoutStrings();

  //add code above this line

  return 0;

}
```

First, we should determine what attribute to get and set. We currently have only one attribute, `list_of_words`, which is used to store the list of strings. However, what if we want to get a particular string? Or perhaps, set a particular string at an index?

To be able to return a string from the vector, we'll need to specify an index or position. This means our getter function will need to take in an integer parameter. However, there is a limitation to the index we can specify. We cannot specify an index position that does not exist. Therefore, we'll need to validate the given parameter before returning anything.

```
string GetString(int i) {
  if (i >= list_of_words.size()) {
    return "No string exists at this index.";
  }
  return list_of_words.at(i);
}
```

`GetString` checks to see if the index exists before it returns a string within the vector. If the index does not exist, `No string exists at this index.` will be returned to the user. Otherwise, the function will return the string at the specified index.

Next, let's create our setter. This setter function will first check to see if the index exists, then it will set a particular string specified by the user into that index. Therefore, we'll need two parameters for our setter.

```
void SetString(string s, int i) {
  if (i >= list_of_words.size()) {
    cout << "No string exists at this index." << endl;
  }
  list_of_words.at(i) = s;
}
```

Let's test the code with the following commands in `main`.

```
  //add code below this line

  vector<string> list = {"house", "cake", "pancake"};
  Words words(list);
  cout << words.GetString(0) << endl;
  words.SetString("mouse", 0);
  cout << words.GetString(0) << endl;

  //add code above this line
```

After the vector is accepted as a parameter, the `GetString` function is able to return the string specified at index `0`. This is why `house` is printed. Then, `SetString` takes in `mouse` as a parameter and replaces the word at index `0` with it. `GetString` is then called again. Since `mouse` has replaced `house`, `mouse` is printed during the second call.

▼ **Updated Code**

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Words {
  public:
    Words(vector<string> n){
      list_of_words = n;
    }

    string GetString(int i) {
      if (i >= list_of_words.size()) {
        return "No string exists at this index.";
      }
      return list_of_words.at(i);
    }

    void SetString(string s, int i) {
      if (i >= list_of_words.size()) {
        cout << "No string exists at this index." << endl;
      }
      list_of_words.at(i) = s;
    }

    void CoutStrings() {
       vector<int> vowel_sizes;
       int size;
       for (auto a : list_of_words) {
         size = CountVowels(a, a.length());
         vowel_sizes.push_back(size);
       }
       for (int i = 0; i < vowel_sizes.size(); i++) {
         if (i == vowel_sizes.size()-1) {
           cout << vowel_sizes.at(i) << endl;
         }
         else {
           cout << vowel_sizes.at(i) << ',';
         }
       }
     }

  private:
    vector<string> list_of_words;

    bool IsVowel(char ch) {
      ch = toupper(ch);
      return (ch=='A' || ch=='E' || ch=='I' ||
              ch=='O' || ch=='U');
    }

    int CountVowels(string str, int n) {
      if (n == 1) {
        return IsVowel(str[n-1]);
      }
      return CountVowels(str, n-1) + IsVowel(str[n-1]);
    }
```

```
};

//add class definitions above this line


int main() {

  //add code below this line

  vector<string> list = {"house", "cake", "pancake"};
  Words words(list);
  cout << words.GetString(0) << endl;
  words.SetString("mouse", 0);
  cout << words.GetString(0) << endl;

  //add code above this line

  return 0;

}
```

Lab Challenge

Lab Challenge

Problem
Write a class named Person that has attributes string name, int age, and string occupation. These attributes should be private. Create getters and setters for each attribute following C++ conventions.

▼ **Given Code**

```cpp
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line



//add class definitions above this line


int main() {

  //DO NOT EDIT CODE BELOW THIS LINE

  Person p("Citra Curie", 16, "student");
  cout << p.GetName() << endl;
  p.SetName("Rowan Faraday");
  cout << p.GetAge() << endl;
  p.SetAge(18);
  cout << p.GetOccupation() << endl;
  p.SetOccupation("plumber");
  cout << p.GetName() << endl;
  cout << p.GetAge() << endl;
  cout << p.GetOccupation() << endl;

  //DO NOT EDIT CODE ABOVE THIS LINE

  return 0;

}
```

Requirements
* Declare the instance `Person("Citra Curie", 16, "student")`
* The function `GetName()` returns `Citra Curie`
* The function `SetName("Rowan Faraday")` changes the name attribute to “Rowan Faraday”
* The function `GetAge()` returns 16
* The function `SetAge(18)` changes the age attribute to 18
* The function `GetOccupation()` returns `student`
* The function `SetOccupation("plumber")` changes the occupation attribute to “plumber”
* **DO NOT EDIT** the specified code in the `main` function or you may not receive credit for your work!

Expected Output

```
Citra Curie
16
student
Rowan Faraday
18
plumber
```

Testing Your Code
Use the button below to test your code before submitting it for evaluation.