Lab 1

Lab 1: Defining Your Own Object

When defining an object, there are a few criteria that must be met in order for an object to be successfully created.

```
class ClassName {
  public:
    type attribute1;
    type attribute2;
};
```

1. All objects are created inside classes. The first step is use the keyword `class`.
2. Give the class a name which is used to construct an object.
3. The keyword `public:` is needed in order to provide access to object attributes that you define later on.
4. Give your object as many attributes as needed in order to make full use of your object.
5. All attributes go within curly braces `{}` after the class name declaration. Make sure to end the closing curly brace with a semi-colon `;`.

Student Object

Let's define an object called `Student`. When we think of a student, some information or attributes that we might associate with students are:
* Name
* Student ID number
* Major
* GPA
* Age
* Year of graduation

Once we've narrowed down what attributes we want associated with our object, we can start defining them.

```
class Student {
  public:
    string name;
    int ID;
    string major;
    double GPA;
    int age;
    int YOG;
};
```

Creating the Object

Once the class has been established, we can create the object by calling on the object's class and giving the object a name in the `main` function.

```
Student andy;
andy.name = "Andy";
andy.ID = 123456;
andy.major = "Computer Science";
andy.GPA = 3.45;
andy.age = 22;
andy.YOG = 2021;
```

Printing the Object's Attributes

It is not sufficient to simply say `cout << andy;` in order to print the attributes associated with `andy`. Instead, we must use **dot notation** to specify what attribute of `andy` we want to output. To print `andy`'s ID for example, use `cout << andy.ID;`. Or to print `andy`'s major, use `cout << andy.major;`. Add the following to the existing code and click the `TRY IT` button to see what is printed about `andy`.

```
cout << andy.name << " is " << andy.age;
cout << " years old and is graduating with a degree in ";
cout << andy.major << " in " << andy.YOG << ".";
```

Lab 2

Lab 2: Building a Constructor

Defining an object and then having to use dot notation to assign values to it every time can be a long and difficult task. To help alleviate that issue, we can build a **constructor** to help us create an object with specified attributes.

A Default Constructor

A constructor works similarly to a function in that you can define parameters within the constructor. Then when it's time to call the constructor, you can just simply give it the appropriate arguments and the object can be made. Let's return to our `Student` object.

```
//add class definitions below this line

class Student {
  public:
    string name;
    int ID;
    string major;
    double GPA;
    int age;
    int YOG;

  Student() {
    name = "Andy";
    ID = 123456;
    major = "Computer Science";
    GPA = 3.45;
    age = 22;
    YOG = 2021;
  }
};

//add class definitions above this line

int main() {

  //add code below this line

  Student mary;
  cout << mary.name << "'s student ID is " << mary.ID << "." <<
        endl;

  mary.name = "Mary";
  mary.GPA = 3.78;

  cout << mary.name << " has a current GPA of " << mary.GPA <<
        "." << endl;

  //add code above this line

  return 0;

}
```

Before, we had to use dot notation to assign values to our object. But having the constructor, we can build it in a way that it will have default values when the object is created. However, notice how the object `mary` has all of the attributes of `andy`.

Constructors with Parameters

The default constructor makes all objects an `andy` object when they are built. To change the attributes of the object, we can still use dot notation (i.e. `mary.name = "Mary";`). However, most `Students` are unique and to have to reassign value every time a default constructor is used can still be a small challenge.

To make the constructor more flexible, we can give it parameters. A constructor with parameters works similarly to a user-defined function in which you provide the parameter types and the user simply has to provide the arguments.

```
//add class definitions below this line

class Student {
  public:
    string name;
    int ID;
    string major;
    double GPA;
    int age;
    int YOG;

  Student(string n, int id, string m, double g, int a, int y) {
    name = n;
    ID = id;
    major = m;
    GPA = g;
    age = a;
    YOG = y;
  }
};

//add class definitions above this line

int main() {

  //add code below this line

  Student andy("Andy", 123456, "Computer Science", 3.45, 22,
        2021);
  Student mary("Mary", 456789, "Mathematics", 3.78, 21, 2022);

  cout << mary.name << " is a student in the " << mary.major <<
        " department." << endl;
  cout << mary.name << " is a junior while " << andy.name << "
        is a senior." << endl;

  //add code above this line


  return 0;

}
```

A constructor with parameters enables the user to decide what attributes to assign right when the object is created. The user just has to provide those attributes as arguments in parentheses (i.e. `Student mary("Mary", 456789, "Mathematics", 3.78, 21, 2022);`).

Lab Challenge

Lab Challenge

Create the variable `dog1`, and instantiate it as an object of the `Dog` class. This dog's name is `Marceline` and she is a `German Shepherd`. Create the variable `dog2` and make it a **copy** of `dog1`. `dog2` should be named `Cajun` and have the breed `Belgian Malinois`.

Your goal for this assignment is to design the class `Dog` so that the above can be implemented successfully.

Expected Output

Test your code by printing the `name` and `breed` of each dog to make sure they fulfill the requirements above. Most importantly, the third print statement will print `false`.

```
Marceline German Shepherd
Cajun Belgian Malinois
false
```

DO NOT CHANGE the existing code in `main`.

```
Dog dog1("Marceline", "German Shepherd");
Dog dog2 = dog1;
dog2.name = "Cajun";
dog2.breed = "Belgian Malinois";

cout << dog1.name << " " << dog1.breed << endl;
cout << dog2.name << " " << dog2.breed << endl;
if (dog1.name == dog2.name && dog1.breed == dog2.breed) {
  cout << boolalpha << true;
}
else {
  cout << boolalpha << false;
}
```