Computing 2 Homework Problems

Problem Set 1 (Classes `BikeSaddle`, `Bike`, and their Composition)

In what follows below, you are asked to work on the following two classes in the order listed below. You also need to write them in this order in your cpp file:

1. `BikeSaddle`
2. `Bike`

Your tasks are:

1. Implement the definition and all its member functions of class `BikeSaddle` as shown in the UML class diagram depicted in Figure 1.
2. Implement the definition and all its member functions of class `Bike` as shown in the UML class diagram depicted in Figure 2.
3. Write a stand-alone function `void display(Bike bike)` that outputs the properties of the bike object. This function is already provided for the most part below. The goal is to demonstrate to you that enums are very helpful in a switch/case environment.
4. Using the `Bike` class, write a main function, that produces the following output. The class `BikeSaddle` is not yet needed for the main function. This completes the first part of the tutorial.

```
Bike properties:
Bike type: unknown type
Bike id: 0

Bike properties:
Bike type: cruiser bike
Bike id: 4711
```

5. In the second part of this exercise, we want to add a `BikeSaddle` object to our `Bike` class as displayed in Figure 3. Please implement the additional member functions related to `BikeSaddle` as shown in the UML diagram. Keep in mind that the `BikeSaddle` class has no default constructor, and this is a situation where an initialization list is needed for the constructor of the `Bike` class.
6. Now, expand the stand-alone function `void display(Bike bike)` such that it also outputs the attributes `color_` and `manufacturer_` of the `BikeSaddle` object which is part of the `Bike` class.
7. Finally, make sure that the main function outputs what is shown below.

```
Bike properties:
Bike type: no type specified yet
Bike id: 0
Saddle color:
Saddle manufacturer:

Bike properties:
Bike type: cruiser bike
Bike id: 4711
Saddle color: black
Saddle manufacturer: Airo
```

Additional Information

The UML class diagram for the class `BikeSaddle` is

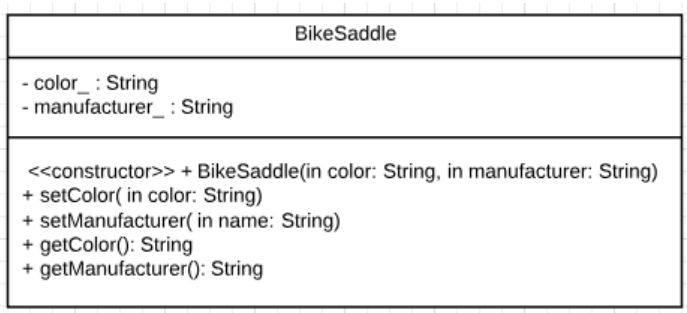

Figure 1: UML Class Diagram for class `BikeSaddle`. Note that the class `BikeSaddle` has no default constructor.

The class `Bike` will be expanded as we move along. We start with a simple version of it shown in the UML class diagram below (see Figure 2).

Among others, you find the type `BikeType` there. It is an `enum` defined as follows:

```
enum BikeType { kUnknown, kRoadBike, kTouringBike, kMountainBike,
kCruiser, kCityBike, kBmxBike, kFoldingBike, kTandem, kTricycle};
```

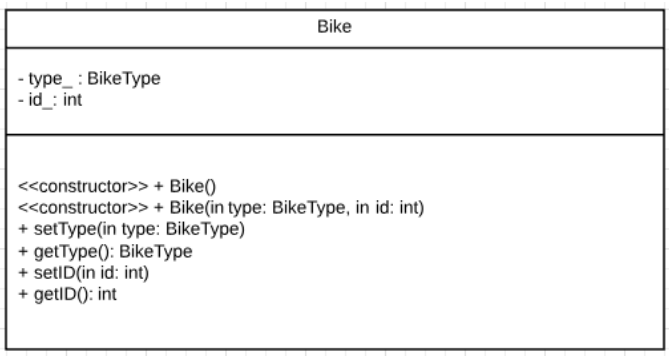

Figure 2: First UML Diagram for class Bike. The type BikeType is an enum.

Here is the definition of the stand-alone function `void display(Bike bike)` which you need to generate the output in the `main` function (tasks 3 and 4 above):

```
void display(Bike bike)
{
      cout << "\nBike properties: " << endl;
      cout << "Bike type: ";
      //your code here
      {
         case kUnknown:      cout << "unknown type" << endl; break;
         case kRoadBike:     cout << "road bike" << endl; break;
         case kTouringBike:  cout << "touring bike" << endl; break;
         case kMountainBike: cout << "mountain bike" << endl; break;
         case kCruiser:      cout << "cruiser bike" << endl; break;
         case kCityBike:     cout << "city bike" << endl; break;
         case kBmxBike:      cout << "BMX bike" << endl; break;
         case kFoldingBike:  cout << "folding bike" << endl; break;
         case kTandem:       cout << "folding bike" << endl; break;
```

```
      case kTricycle:     cout << "tricycle" << endl; break;
      default:
        cout << "no valid bike type!" << endl;
   }
   cout << /* your code here */ << endl;

   return;
}
```

Below, you find the UML diagram for the `Bike` class that now has an additional data element of type `BikeSaddle`. This relationship is called **composition**. In UML, it is depicted as a binary association decorated with a filled black diamond at the aggregate (whole) end. The numbers represent multiplicity. Here, a bike has one saddle, and each saddle belongs to exactly one bike.

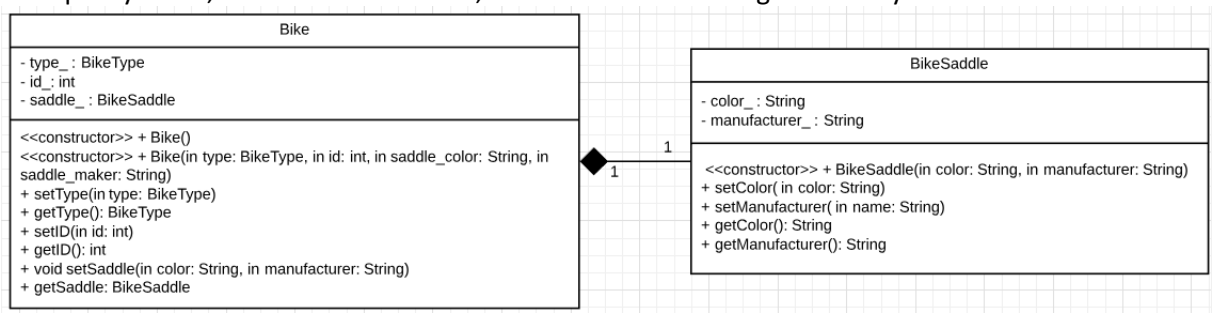

Figure 3: The UML diagram for an expanded class `Bike`. It has with a data member of type `BikeSaddle`. This is called composition.

Here is the definition of the stand-alone function `void display(Bike bike)` which you need to generate the output in the `main` function (tasks 6 and 7 above):

```
void display(Bike bike)
{
   cout << "\nBike properties: " << endl;
   cout << "Bike type: ";
   //your code here
   {
      case kUnknown:      cout << "unknown type" << endl; break;
      case kRoadBike:     cout << "road bike" << endl; break;
      case kTouringBike:  cout << "touring bike" << endl; break;
      case kMountainBike: cout << "mountain bike" << endl; break;
      case kCruiser:      cout << "cruiser bike" << endl; break;
      case kCityBike:     cout << "city bike" << endl; break;
      case kBmxBike:      cout << "BMX bike" << endl; break;
      case kFoldingBike:  cout << "folding bike" << endl; break;
      case kTandem:       cout << "folding bike" << endl; break;
      case kTricycle:     cout << "tricycle" << endl; break;
      default:
        cout << "no valid bike type!" << endl;
   }
   cout << "Bike id: " << /* your code here */ << endl;
   cout << "Saddle color: " << /* your code here */ << endl;
   cout << "Saddle manufacturer: " << /* your code */ << endl;

   return;
}
```