Part E - Polymorphism  
  
**Virtual Functions**

Workshop 8

In this workshop, you are to create an abstract base class and inherit it into two derived classes.

**LEARNING OUTCOMES**

Upon successful completion of this workshop, you will have demonstrated the abilities to

Define a virtual base class

Inherit from a virtual base class

Call derived class functions through a virtual base class call, demonstrating inclusion polymorphism

Reflect on the concepts learned in this workshop

**SUBMISSION POLICY**

The workshop is **due the day before your next scheduled workshop.** Late submissions as always will not be accepted.

**part 1: Vehicle Class**

Design and code a virtual base class named **Vehicle** that holds information about a vehicle. Place your class definition in a header file named **Vehicle.h** and your functions in an implementation file named **Vehicle.cpp**.

Your **Vehicle** design includes information about its **speed**, two **protected** member functions, and three **public** member functions. The design should also include two non-friend helper functions. Here’s the definition for the **Vehicle** class:

* **speed** – a private data member (**double**).
* **double getSpeed() const** – a **protected** member function that returns the speed of a vehicle.
* **void setSpeed(double)** – a **protected** member function that sets the speed of a vehicle to the argument it receives if the speed is greater than zero, otherwise it sets the speed to 0.0.
* **void accelerate()**– a pure virtual **public** member function that increases a vehicle’s speed.
* **void brake()** – a pure virtual **public** member function that decreases a vehicle’s speed.
* **void display(std::ostream&) const** – a pure virtual **public** member function that receives a reference to an **ostream** object and displays the vehicle’s characteristics.
* **void move(Vehicle&)**– a helper operator that will accelerate then brake the Vehicle object passed to it.
* **void show(Vehicle&)**– a helper operator that displays the Vehicle object passed to it.

Derive from the **Vehicle** class a class named **Bus** and a class named **Truck**. Place your class definitions in header files named **Bus.h** and **Truck.h**  respectively, and your functions in implementation files named **Bus.cpp** and **Truck.cpp**  respectively. Include in your design all of the statements and keywords necessary to compile and to run your code successfully under a standard C++ compiler.

Upon instantiation, a **Bus** object may receive no information or may receive 2 values:

* a double for the bus’ speed;
* a integer for the total passengers capacity.

Your **Bus** design includes the following member functions and operators:

* **void accelerate()** – a function that adds 20 to the bus’ speed;
* **void brake()** – a function that subtracts 10 from the bus’ speed;
* **void display(std::ostream&) const** – a query that receives a reference to an **ostream** object and inserts the following into that object:

This bus has total capacity of <capacity> passengers and has a speed of <speed> after acceleration and braking.

Upon instantiation, a **Truck** object may receive no information or may receive 1 or 2 values:

* a double for the truck’s speed;
* a bool defining if the truck has towing capacity or not (with a default value of **false**, if not provided).

Your **Truck** design includes the following member function and operators:

* **void accelerate()** – a function that adds 50 to the truck’s speed;
* **void brake()** – a function that subtracts 20 from the truck’s speed;
* **void display(std::ostream&) const** – a query that receives a reference to an **ostream** object and inserts the following into that object:

This truck is going <speed> after acceleration and braking, and <it can be used to tow / it cannot be used to tow.> (depends on the value of the truck’s **bool**).

The program on the following page uses your **Vehicle, Bus** and **Truck** classes and produces the output shown below:

|  |  |
| --- | --- |
| #include <iostream>  #include "Bus.h"  #include "Truck.h"  int main()  {  Bus cityBus(20, 78);  Truck pickUp(30, true);  move(cityBus);  move(pickUp);  show(cityBus);  show(pickUp);    return 0;  } | This bus has total capacity of 78 passengers and has a speed of 30 after acceleration and braking.  This truck is going 60 after acceleration and braking, and it can be used to tow. |

**Part 2: I/O using virtual base classES**

Add the following pure virtual function and two overloaded non-friend helper operators to the Vehicle class.

* **void set(std::istream&)**– a pure virtual **public** member function that takes in an object of type **istream**.
* **std::istream& operator>>(std::istream&, Vehicle&)**– an extraction operator that calls the Vehicle **set()** function and returns a reference to the **istream**.
* **std::ostream& operator<<(std::ostream&, Vehicle&)** – an insertion operator that calls the **move** helper function on the passed-in vehicle, then calls the vehicle display member function before returning a reference to the **ostream** object.

Add the necessary **set** functions to your **Bus** and **Truck** classes to prompt the user for inputs to set the **Bus** and **Truck** member variables. The program on the following page uses your **Vehicle, Bus and Truck** bonus classes and produces the output shown below:

|  |  |
| --- | --- |
| #include <iostream>  #include "Bus.h"  #include "Truck.h"  int main()  {  Bus cityBus;  Truck pickUp;  std::cin >> cityBus >> pickUp;  std::cout << cityBus << pickUp;    return 0;  } | ***Bold italic represent user inputs***  Bus => total capacity: ***82***  Bus => speed: ***20.2***  Truck => speed: ***32.4***  Truck can tow? (true=1 / false=0): ***1***  This bus has total capacity of 82 passengers and has a speed of 30.2 after acceleration and braking.  This truck is going 62.4 after acceleration and braking, and it can be used to tow. |

**SUBMISSION**

If not on matrix already, upload **Bus.h**, **Bus.cpp,** **Truck.h, Truck.cpp, Vehicle.h, Vehicle.cpp** and **w8.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

**>   
Section SCC and SDD:  
~ronald.burton/submit w8 <ENTER>**