San Francisco Bay University

CS360 - Programming in C and C++
Homework Assignment #5

Due day: 4/04/2024

Replit link: **https://replit.com/@TASMITA-TANJIMT/CSAssignment5**

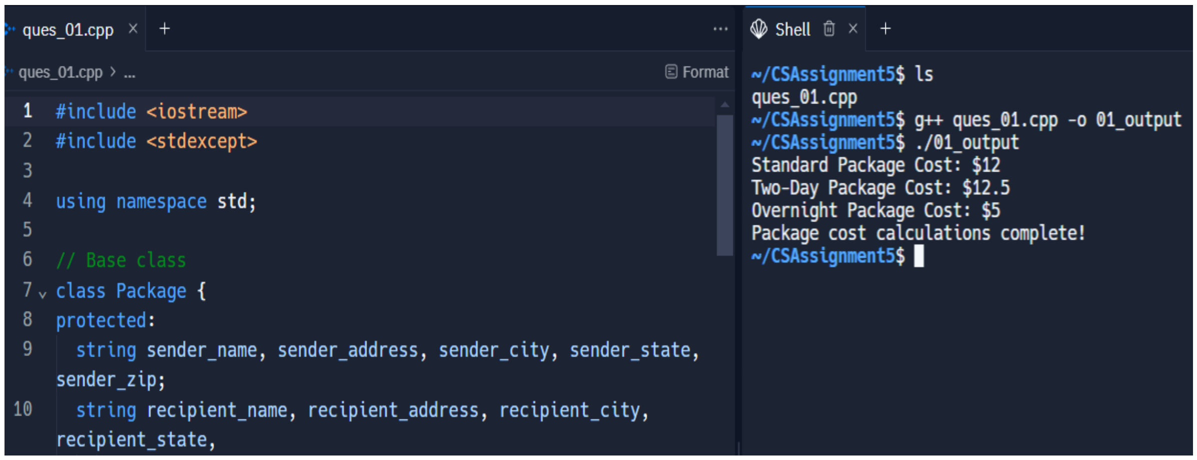
GITHUB LINK: **https://github.com/tasmita0131/CS_Assignment_-5**

1. Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages. Use class *Package* as the base class of the hierarchy, then include classes *TwoDayPackage* and *OvernightPackage* that derive from *Package*.

   Base class *Package* should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. *Package*'s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. *Package* should provide a *public* member function *calculateCost* that returns a *double* indicating the cost associated with shipping the package. *Package's calculateCost* function should determine the cost by multiplying the weight by the cost per ounce. Derived class *TwoDayPackage* should inherit the functionality of base class *Package*, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. *TwoDayPackage's* constructor should receive a value to initialize this data member. *TwoDayPackage* should redefine member function *calculateCost* so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class *Package's calculateCost* function. Class *OvernightPackage* should inherit directly from class *Package* and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. *OvernightPackage* should redefine member function *calculateCost* so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost. Write a *main* program that creates objects of each type of *Package* and tests member function *calculateCost*.

   CODE:

```
#include <iostream>
#include <stdexcept>

using namespace std;

// Base class
class Package {
protected:
  string sender_name, sender_address, sender_city, sender_state, sender_zip;
  string recipient_name, recipient_address, recipient_city, recipient_state,
      recipient_zip;
  double weight; // in ounces
  double cost_per_ounce;

public:
  Package(string sn, string sa, string sc, string ss, string sz, string rn,
          string ra, string rc, string rs, string rz, double w, double cpo)
      : sender_name(sn), sender_address(sa), sender_city(sc), sender_state(ss),
        sender_zip(sz), recipient_name(rn), recipient_address(ra),
        recipient_city(rc), recipient_state(rs), recipient_zip(rz), weight(w),
        cost_per_ounce(cpo) {
    if (weight <= 0 || cost_per_ounce <= 0) {
      throw invalid_argument(
          "Weight and cost per ounce must be positive values.");
    }
  }

  virtual double calculateCost() const { return weight * cost_per_ounce; }
};

// Derived class for TwoDayPackage
class TwoDayPackage : public Package {
private:
  double flat_fee;

public:
  TwoDayPackage(string sn, string sa, string sc, string ss, string sz,
                string rn, string ra, string rc, string rs, string rz, double w,
                double cpo, double ff)
      : Package(sn, sa, sc, ss, sz, rn, ra, rc, rs, rz, w, cpo), flat_fee(ff) {}

  double calculateCost() const override {
    return Package::calculateCost() + flat_fee;
  }
};

// Derived class for OvernightPackage
class OvernightPackage : public Package {
private:
  double additional_fee_per_ounce;

public:
  OvernightPackage(string sn, string sa, string sc, string ss, string sz, string rn, string ra, string rc,
string rs, string rz, double w, double cpo, double afo) :
 Package(sn, sa, sc, ss, sz, rn, ra, rc, rs, rz, w, cpo),
        additional_fee_per_ounce(afo) {}

  double calculateCost() const override {
    return weight * (cost_per_ounce + additional_fee_per_ounce);
  }
};

int main() {
  try {
    Package package("Alice", "123 Main St", "Anytown", "Anystate", "12345",
                "Bob", "456 Second St", "Othertown", "Otherstate", "67890",
                16, 0.75);

    TwoDayPackage twoDayPackage("Charlie", "789 Third St", "Sometown",
                                "Somestate", "24680", "Diana", "1011 Fourth St",
                                "Newtown", "Newstate", "13579", 10, 0.75, 5.00);

    OvernightPackage overnightPackage(
        "Eve", "1213 Fifth St", "Oldtown", "Oldstate", "97531", "Frank",
        "1415 Sixth St", "Losttown", "Loststate", "86420", 5, 0.75, 0.25);

    cout << "Standard Package Cost: $" << package.calculateCost() << endl;
    cout << "Two-Day Package Cost: $" << twoDayPackage.calculateCost() << endl;
    cout << "Overnight Package Cost: $" << overnightPackage.calculateCost()
         << endl;
  } catch (const invalid_argument &e) {
    cerr << "Error: " << e.what() << endl;
  }

  cout << "Package cost calculations complete!" << endl;

  return 0;
}
```



```
#include <iostream>
#include <stdexcept>

using namespace std;

// Base class
class Package {
protected:
  string sender_name, sender_address, sender_city, sender_state,
sender_zip;
  string recipient_name, recipient_address, recipient_city,
recipient_state,
```

```
~/CSAssignment5$ ls
ques_01.cpp
~/CSAssignment5$ g++ ques_01.cpp -o 01_output
~/CSAssignment5$ ./01_output
Standard Package Cost: $12
Two-Day Package Cost: $12.5
Overnight Package Cost: $5
Package cost calculations complete!
~/CSAssignment5$
```

2. A supermarket chain has asked you to develop an automatic checkout system. All products are identifiable by means of a barcode and the product name. Groceries are either sold in packages or by weight. Packed goods have fixed prices. The price of groceries sold by weight is calculated by multiplying the weight by the current price per kilo.

   Develop the classes needed to represent the products first and organize them hierarchically. The *Product* class, which contains generic information on all products (barcode, name, etc.), can be used as a base class.

   a. **The *Product* class contains two data members of type *long* used for storing barcodes and the product name. Define a constructor with parameters for both data members. Add default values for the parameters to provide a default**

constructor for the class. In addition to the access methods *setCode()* and *getCode()*, also define the methods *scanner()* and *printer()*. For test purposes, these methods will simply output product data on screen or read the data of a product from the keyboard.

CODE OF a:

```
#include <iostream>
#include <string>

using namespace std;

class Product {
protected:
    long barcode;
    string name;

public:
    // Constructor with default values for barcode and name
    Product(long barcode = 0, const string& name = "") : barcode(barcode), name(name) {}

    // Access methods for the barcode
    void setCode(long newBarcode) { barcode = newBarcode; }
    long getCode() const { return barcode; }

    // Method for reading product data from keyboard
    void scanner() {
        cout << "Enter barcode: ";
        cin >> barcode;
        cin.ignore(); // Ignore newline after numeric input to allow getline to work for name input
        cout << "Enter product name: ";
        getline(cin, name);
    }

    // Method for printing product data to screen
    void printer() const {
        cout << "Product Barcode: " << barcode << ", Product Name: " << name << endl;
    }
};
```

b. **The next step involves developing special cases of the *Product* class. Define two classes derived from *Product, PrepackedFood* and *FreshFood*. In addition to the product data, the *PrepackedFood* class should contain the unit price and the *FreshFood* class should contain a weight and a price per kilo as data members.**

In both classes define a constructor with parameters providing default-values for all data members. Use both the base and member initializer.

Define the access methods needed for the new data members. Also redefine the methods *scanner()* and *printer()* to take the new data members into consideration.

CODE OF b:

PrepackedFood Class:

```
class PrepackedFood : public Product {
private:
    double unitPrice; // Price of the product per unit

public:
    // Constructor with parameters for all data members, providing default values
    PrepackedFood(long barcode = 0, const string& name = "", double unitPrice = 0.0)
        : Product(barcode, name), unitPrice(unitPrice) {}

    // Access methods for the unit price
    void setUnitPrice(double newUnitPrice) { unitPrice = newUnitPrice; }
    double getUnitPrice() const { return unitPrice; }

    // Override scanner to include unit price input
    void scanner() override {
        Product::scanner(); // Call base class scanner to input barcode and name
        cout << "Enter unit price: ";
        cin >> unitPrice;
    }

    // Override printer to include unit price output
    void printer() const override {
        Product::printer(); // Call base class printer to print barcode and name
        cout << "Unit Price: $" << unitPrice << endl;
    }
};
```

FreshFood Class:

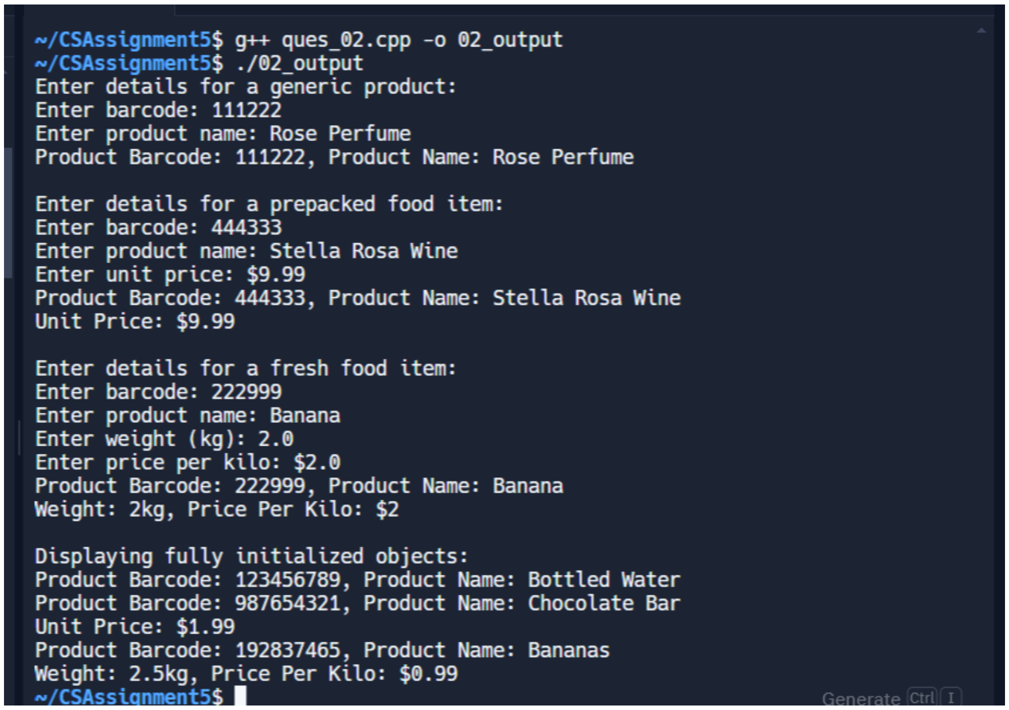
```
class FreshFood : public Product {
private:
    double weight; // Weight of the product
    double pricePerKilo; // Price of the product per kilo

public:
    // Constructor with parameters for all data members, providing default values
    FreshFood(long barcode = 0, const string& name = "", double weight = 0.0, double pricePerKilo =
0.0)
        : Product(barcode, name), weight(weight), pricePerKilo(pricePerKilo) {}

    // Access methods for weight and price per kilo
    void setWeight(double newWeight) { weight = newWeight; }
    double getWeight() const { return weight; }
    void setPricePerKilo(double newPricePerKilo) { pricePerKilo = newPricePerKilo; }
    double getPricePerKilo() const { return pricePerKilo; }
    // Override scanner to include weight and price per kilo input
    void scanner() override {
        Product::scanner(); // Call base class scanner to input barcode and name
        cout << "Enter weight (kg): ";
        cin >> weight;
        cout << "Enter price per kilo: $";
        cin >> pricePerKilo;
    }

    // Override printer to include weight and price per kilo output
    void printer() const override {
        Product::printer(); // Call base class printer to print barcode and name
        cout << "Weight: " << weight << "kg, Price Per Kilo: $" << pricePerKilo << endl;
    }
};
```

c. **Test the various classes in a *main* function that creates two objects each of the types Product, *PrepackedFood* and *FreshFood*. One object of each type is fully initialized in the object definition. Use the default constructor to create the other object. Test the *get* and *set* methods and the *scanner()* method and display the products on screen.**

CODE FOR c:

```
int main() {
    // Default constructor objects
    Product productDefault;
    PrepackedFood prepackedFoodDefault;
    FreshFood freshFoodDefault;

    // Fully initialized objects
    Product productFull(123456789, "Bottled Water");
    PrepackedFood prepackedFoodFull(987654321, "Chocolate Bar", 1.99);
    FreshFood freshFoodFull(192837465, "Bananas", 2.5, 0.99);

    // Testing scanner method on default constructor objects
    cout << "\nEnter details for a generic product:" << endl;
    productDefault.scanner();
    cout << "\nEnter details for a prepacked food item:" << endl;
    prepackedFoodDefault.scanner();
    cout << "\nEnter details for a fresh food item:" << endl;
    freshFoodDefault.scanner();

    // Testing printer method for all objects
    cout << "\nDisplaying all product details:" << endl;
    cout << "Generic Product (Default):" << endl;
    productDefault.printer();
    cout << "Prepacked Food (Default):" << endl;
    prepackedFoodDefault.printer();
    cout << "Fresh Food (Default):" << endl;
    freshFoodDefault.printer();
    cout << "\nGeneric Product (Full):" << endl;
    productFull.printer();
    cout << "Prepacked Food (Full):" << endl;
    prepackedFoodFull.printer();
    cout << "Fresh Food (Full):" << endl;
    freshFoodFull.printer();

    return 0;
}
```

OUTPUT OF THE CODE:



```
~/CSAssignment5$ g++ ques_02.cpp -o 02_output
~/CSAssignment5$ ./02_output
Enter details for a generic product:
Enter barcode: 111222
Enter product name: Rose Perfume
Product Barcode: 111222, Product Name: Rose Perfume

Enter details for a prepacked food item:
Enter barcode: 444333
Enter product name: Stella Rosa Wine
Enter unit price: $9.99
Product Barcode: 444333, Product Name: Stella Rosa Wine
Unit Price: $9.99

Enter details for a fresh food item:
Enter barcode: 222999
Enter product name: Banana
Enter weight (kg): 2.0
Enter price per kilo: $2.0
Product Barcode: 222999, Product Name: Banana
Weight: 2kg, Price Per Kilo: $2

Displaying fully initialized objects:
Product Barcode: 123456789, Product Name: Bottled Water
Product Barcode: 987654321, Product Name: Chocolate Bar
Unit Price: $1.99
Product Barcode: 192837465, Product Name: Bananas
Weight: 2.5kg, Price Per Kilo: $0.99
~/CSAssignment5$
```

ENTIRE CODE:

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

// Base class Product
class Product {
protected:
    long barcode;
    string name;

public:
    Product(long barcode = 0, const string& name = "Unnamed") : barcode(barcode), name(name) {}

    void setCode(long newBarcode) { barcode = newBarcode; }
    long getCode() const { return barcode; }
    virtual void scanner() {
        cout << "Enter barcode: ";
        cin >> barcode;
        cin.ignore(); // To prepare for next getline input
        cout << "Enter product name: ";
        getline(cin, name);
    }

    virtual void printer() const {
        cout << "Product Barcode: " << barcode << ", Product Name: " << name << endl;
    }
};

// Derived class PrepackedFood
class PrepackedFood : public Product {
private:
    double unitPrice;

public:
    PrepackedFood(long barcode = 0, const string& name = "Unnamed", double unitPrice = 0.0)
        : Product(barcode, name), unitPrice(unitPrice) {}

    void setUnitPrice(double newUnitPrice) { unitPrice = newUnitPrice; }
    double getUnitPrice() const { return unitPrice; }

    void scanner() override {
        Product::scanner();
        cout << "Enter unit price: $";
        cin >> unitPrice;
    }

    void printer() const override {
        Product::printer();
        cout << "Unit Price: $" << unitPrice << endl;
    }
};

// Derived class FreshFood
class FreshFood : public Product {
private:
    double weight;
    double pricePerKilo;

public:
    FreshFood(long barcode = 0, const string& name = "Unnamed", double weight = 0.0, double
pricePerKilo = 0.0)
        : Product(barcode, name), weight(weight), pricePerKilo(pricePerKilo) {}

    void setWeight(double newWeight) { weight = newWeight; }
    double getWeight() const { return weight; }
    void setPricePerKilo(double newPricePerKilo) { pricePerKilo = newPricePerKilo; }
    double getPricePerKilo() const { return pricePerKilo; }
    void scanner() override {
        Product::scanner();
        cout << "Enter weight (kg): ";
        cin >> weight;
        cout << "Enter price per kilo: $";
        cin >> pricePerKilo;
    }

    void printer() const override {
        Product::printer();
        cout << "Weight: " << weight << "kg, Price Per Kilo: $" << pricePerKilo << endl;
    }
};

// Main function to test the classes
int main() {
    // Default constructor objects
    Product productDefault;
    PrepackedFood prepackedFoodDefault;
    FreshFood freshFoodDefault;

    // Fully initialized objects
    Product productFull(123456789, "Bottled Water");
    PrepackedFood prepackedFoodFull(987654321, "Chocolate Bar", 1.99);
    FreshFood freshFoodFull(192837465, "Bananas", 2.5, 0.99);

    // Testing scanner and printer for default constructor objects
    cout << "Enter details for a generic product:" << endl;
    productDefault.scanner();
    productDefault.printer();

    cout << "\nEnter details for a prepacked food item:" << endl;
    prepackedFoodDefault.scanner();
    prepackedFoodDefault.printer();

    cout << "\nEnter details for a fresh food item:" << endl;
    freshFoodDefault.scanner();
    freshFoodDefault.printer();

    // Displaying fully initialized objects
    cout << "\nDisplaying fully initialized objects:" << endl;
    productFull.printer();
    prepackedFoodFull.printer();
    freshFoodFull.printer();

    return 0;
}
```