



San Francisco Bay University

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

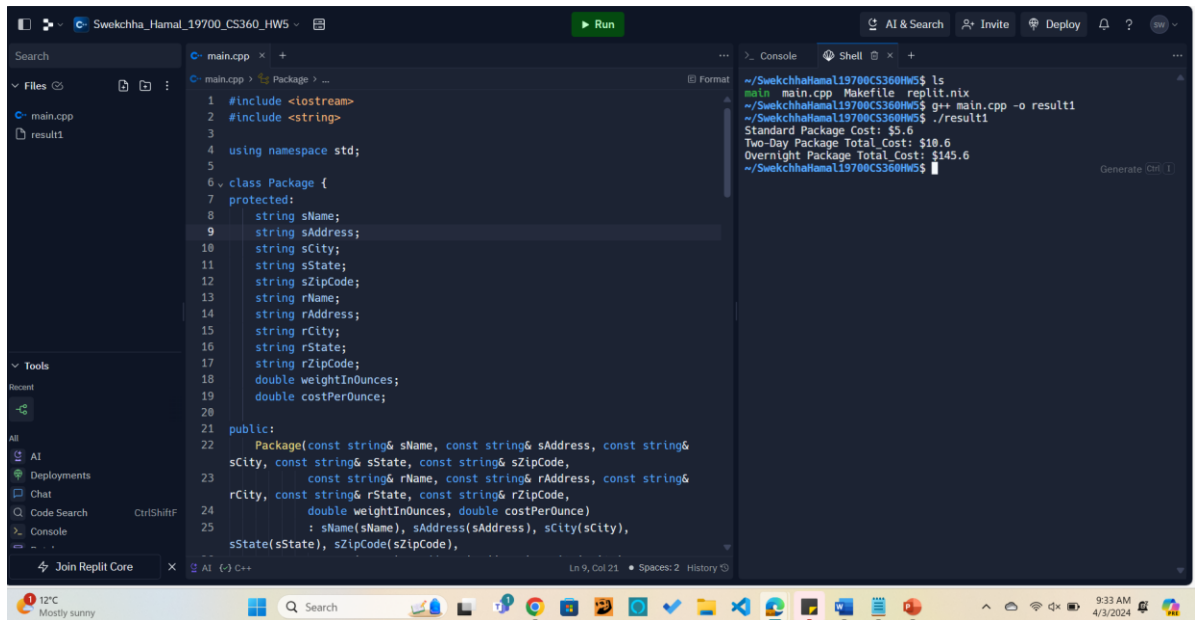
Due day: 4/04/2024

Instruction:

1. Push the answer sheets/source code to Github
2. Please follow the code style rule like programs on handout.
3. Overdue homework assignment submission can't be accepted.
4. Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

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*.



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

class Package {
protected:
    string sName;
    string sAddress;
    string sCity;
    string sState;
    string sZipCode;
    string rName;
    string rAddress;
    string rCity;
    string rState;
    string rZipCode;
    double weightInOunces;
    double costPerOunce;
public:
    Package(const string& sName, const string& sAddress, const string& sCity, const string& sState, const string& sZipCode, const string& rName, const string& rAddress, const string& rCity, const string& rState, const string& rZipCode, double weightInOunces, double costPerOunce) : sName(sName), sAddress(sAddress), sCity(sCity), sState(sState), sZipCode(sZipCode), rName(rName), rAddress(rAddress), rCity(rCity), rState(rState), rZipCode(rZipCode), weightInOunces(weightInOunces), costPerOunce(costPerOunce) {}
};

class TwoDayPackage : public Package {
private:
    double flatFee;
public:
    TwoDayPackage(const string& sName, const string& sAddress, const string& sCity, const string& sState, const string& sZipCode, const string& rName, const string& rAddress, const string& rCity, const string& rState, const string& rZipCode, double weightInOunces, double costPerOunce, double flatFee) : Package(sName, sAddress, sCity, sState, sZipCode, rName, rAddress, rCity, rState, rZipCode, weightInOunces, costPerOunce), flatFee(flatFee) {}
};

int main() {
    Package p1("Standard Package", "123 Main St", "New York", "NY", "10001", "456 Main St", "New York", "NY", "10001", 10, 5.6);
    TwoDayPackage p2("Two-Day Package", "123 Main St", "New York", "NY", "10001", "456 Main St", "New York", "NY", "10001", 10, 5.6, 10.0);
    cout << "Standard Package Total Cost: $" << p1.calculateCost() << endl;
    cout << "Two-Day Package Total Cost: $" << p2.calculateCost() << endl;
    return 0;
}
```

~/SwekchhaHama19700CS360HW5\$ ls
main main.cpp Makefile replit.nix
~/SwekchhaHama19700CS360HW5\$ g++ main.cpp -o result1
~/SwekchhaHama19700CS360HW5\$./result1
Standard Package Cost: \$5.6
Two-Day Package Total Cost: \$10.6
Overnight Package Total Cost: \$145.6
~/SwekchhaHama19700CS360HW5\$

```
sState(sState), sZipCode(sZipCode),  
    rName(rName), rAddress(rAddress), rCity(rCity),  
    rState(rState), rZipCode(rZipCode),  
    weightInOunces(weightInOunces), costPerOunce(costPerOunce) {  
    if (weightInOunces <= 0 || costPerOunce <= 0) {  
        throw invalid_argument("Weight and cost per ounce must be  
positive values.");  
    }  
}  
  
virtual double calculateCost() const {  
    return weightInOunces * costPerOunce;  
}  
};  
  
class TwoDayPackage : public Package {  
private:  
    double flatFee;  
public:  
    TwoDayPackage(const string& sName, const string& sAddress, const  
string& sCity, const string& sState, const string& sZipCode,  
        const string& rName, const string& rAddress, const  
string& rCity, const string& rState, const string& rZipCode,  
        double weightInOunces, double costPerOunce, double  
flatFee)  
        : Package(sName, sAddress, sCity, sState, sZipCode, rName,  
rAddress, rCity, rState, rZipCode,
```

```

        weightInOunces, costPerOunce), flatFee(flatFee) {}
    }

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

class OvernightPackage : public Package {
private:
    double additionalFeePerOunce;

public:
    OvernightPackage(const string& sName, const string& sAddress, const
string& sCity, const string& sState, const string& sZipCode,
                    const string& rName, const string& rAddress, const
string& rCity, const string& rState, const string& rZipCode,
                    double weightInOunces, double costPerOunce, double
additionalFeePerOunce)
        : Package(sName, sAddress, sCity, sState, sZipCode, rName,
rAddress, rCity, rState, rZipCode,
                    weightInOunces, costPerOunce),
        additionalFeePerOunce(additionalFeePerOunce) {}

    double calculateCost() const override {
        return (weightInOunces * costPerOunce) + (weightInOunces *
additionalFeePerOunce);
    }
};

```

```

int main() {
    try {
        Package package("Jasmina", "Thulobharyang", "Kathmandu",
            "Kathmandu", "43567", "Swekchha", "Fernald", "Fremont", "California",
            "94539", 14.0, 0.4);
        TwoDayPackage twoDayPackage("Jasmina", "Thulobharyang",
            "Kathmandu", "Kathmandu", "43567", "Swekchha", "Fernald", "Fremont",
            "California", "94539", 14.0, 0.4, 5.0);
        OvernightPackage overnightPackage("Jasmina", "Thulobharyang",
            "Kathmandu", "Kathmandu", "43567", "Swekchha", "Fernald", "Fremont",
            "California", "94539", 14.0, 0.4, 5.2));

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

    return 0;
}

```

Output:

```

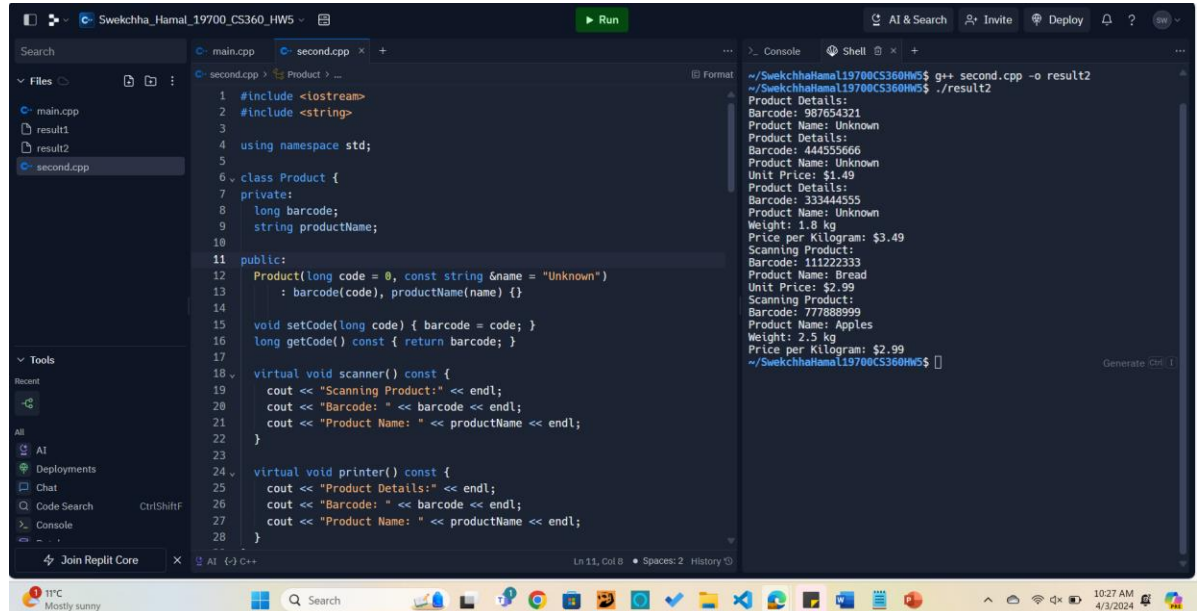
~/SwekchhaHamal19700CS360HW5$ ls
main main.cpp Makefile replit.nix
~/SwekchhaHamal19700CS360HW5$ g++ main.cpp -o result1
~/SwekchhaHamal19700CS360HW5$ ./result1
Standard Package Cost: $5.6
Two-Day Package Total_Cost: $10.6
Overnight Package Total_Cost: $78.4
~/SwekchhaHamal19700CS360HW5$ █

```

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.



The screenshot shows a C++ IDE with a file explorer on the left, a code editor in the center, and a console on the right. The code editor displays the implementation of the *Product* class in *second.cpp*. The class has two private data members: *long barcode* and *string productName*. It includes a constructor *Product(long code = 0, const string &name = "Unknown")* that initializes these members. Public methods include *setCode(long code)*, *getCode()*, *scanner()*, and *printer()*. The *scanner()* method reads input from the console and prints the product details. The *printer()* method prints the product details. The console on the right shows the output of the program, which includes the product details for three different products: a default product, a product with barcode 987654321 and name "Bread", and a product with barcode 777888999, name "Apples", weight 2.5 kg, and price per kilogram \$2.99.

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 class Product {
7 private:
8     long barcode;
9     string productName;
10
11 public:
12     Product(long code = 0, const string &name = "Unknown")
13         : barcode(code), productName(name) {}
14
15     void setCode(long code) { barcode = code; }
16     long getCode() const { return barcode; }
17
18     virtual void scanner() const {
19         cout << "Scanning Product:" << endl;
20         cout << "Barcode: " << barcode << endl;
21         cout << "Product Name: " << productName << endl;
22     }
23
24     virtual void printer() const {
25         cout << "Product Details:" << endl;
26         cout << "Barcode: " << barcode << endl;
27         cout << "Product Name: " << productName << endl;
28     }
29 }
```

```
~/Swkchha_Hamal_19700_CS360_HW5$ g++ second.cpp -o result2
~/Swkchha_Hamal_19700_CS360_HW5$ ./result2
Product Details:
Barcode: 987654321
Product Name: Unknown
Product Details:
Barcode: 444555666
Product Name: Unknown
Unit Price: $1.49
Product Details:
Barcode: 333444555
Product Name: Unknown
Weight: 1.8 kg
Price per Kilogram: $3.49
Scanning Product:
Barcode: 11222333
Product Name: Bread
Unit Price: $2.99
Scanning Product:
Barcode: 777888999
Product Name: Apples
Weight: 2.5 kg
Price per Kilogram: $2.99
~/Swkchha_Hamal_19700_CS360_HW5$
```

- 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.

```
class PrepackedFood : public Product {
private:
    double unitPrice;

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

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

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

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

class FreshFood : public Product {
private:
    double weight;
    double pricePerKilo;
```

```

public:
    FreshFood(long code = 0, const string &name = "Unknown",
              double weightVal = 0.0, double pricePerKg = 0.0)
        : Product(code, name), weight(weightVal), pricePerKilo(pricePerKg)
    {}

    void setWeight(double w) { weight = w; }
    double getWeight() const { return weight; }

    void setPricePerKilo(double price) { pricePerKilo = price; }
    double getPricePerKilo() const { return pricePerKilo; }

    void scanner() const override {
        Product::scanner();
        cout << "Weight: " << weight << " kg" << endl;
        cout << "Price per Kilogram: $" << pricePerKilo << endl;
    }

    void printer() const override {
        Product::printer();
        cout << "Weight: " << weight << " kg" << endl;
        cout << "Price per Kilogram: $" << 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.

```

int main() {

    Product prod1(123456789, "Milk");
    Product prod2;
    prod2.setCode(987654321);
    prod2.printer();

    PrepackedFood packed1(111222333, "Bread", 2.99);
    PrepackedFood packed2;
    packed2.setCode(444555666);
    packed2.setUnitPrice(1.49);
    packed2.printer();

    FreshFood fresh1(777888999, "Apples", 2.5, 2.99);
    FreshFood fresh2;
    fresh2.setCode(333444555);
    fresh2.setWeight(1.8);
    fresh2.setPricePerKilo(3.49);
    fresh2.printer();

    packed1.scanner();
    fresh1.scanner();

    return 0;
}

```

Output:

```

Product Name: Bread
Unit Price: $2.99
Scanning Product:
Barcode: 777888999
Product Name: Apples
Weight: 2.5 kg
Price per Kilogram: $2.99

```