دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

5/25/2020

Report Homework 5

Advanced Programming

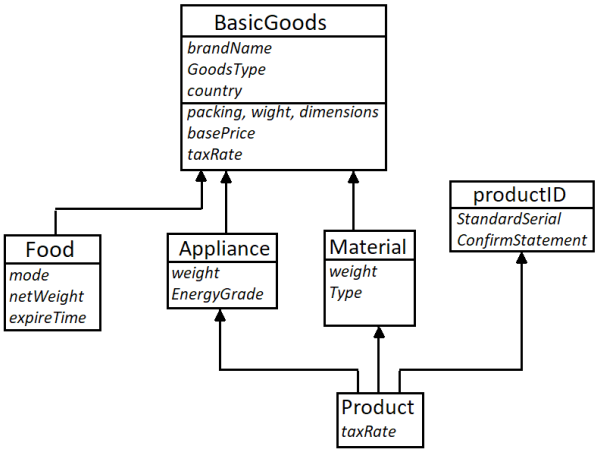
Mohammad Javad Amin (9523008)
AMIRKABIR UNIVERSITY OF TECHNOLOGY

به نام خدا

بخش اول c++

هدف پروژه ؛ طراحی انبارکالا (storehose) و اطلاعات مشتری‌های آن است.

ابتدا بایدکالاها را دسته بندی کنیم . در دسته بندی کالا ها از الگوریتم زیر استفاده می‌کنیم.



همانطورکه در شکل بالا مشخص شده ابتدا باید class Basic Goods که کلاس پایه بوده و بقیه class ها از آن ارث بری می‌کنند را تعریف کنیم.

class Basic Goods :

ابتدا دو enum class به اسم pack_type که مشخص کننده نوع بسته بندی و Goods_type که مشخص کننده نوع کالا است را تعریف و به صورت زیر پیاده سازی می‌کنیم.

```
enum class PACK_TYPE
{
    UNKNOWN,
    RAW,
    CAN,
    NYLON,
    CARTON,
    BARREL
};

enum class GOODS_TYPE
{
    CONSUMABLE,
    DISPOSABLE,
    OBJECT
};
```

member variable های class Basic goods :

- brandName : نام برندکالا
- goodsType : مشخص کننده نوع کالا
- originCountry : اسم کشور مبدا
- packing : مشخص کننده نوع بسته بندی
- weight : وزن بسته بندی
- approxDim : ابعاد بسته بندی
- basePrice : قیمت پایه
- taxRate : مقدار مالیات

```
class BasicGoods
{
protected:
    PACK_TYPE packing;
    double weight;                      //approx weight of packing! not the commodity
    std::shared_ptr< std::array<double, 3> > approxDim;  //dimentions
    double basePrice;                    // value in euro or dollar
    double taxRate;                      // in percent

    void setNewPrice(double price);      //set new base price

private:
    char* brandName;                     //brand name
    GOODS_TYPE goodsType;
    char* originCountry;

public:
    BasicGoods(const std::string& bName, GOODS_TYPE gTyp, double price);  //constructor
    virtual ~BasicGoods();                                                //destructor

    constexpr static double INITIAL_TAX_RATE = 5;

    // getters
    std::string getBrandName(void) const;
    double getBaseprice(void) const;
    GOODS_TYPE getGoodsType(void) const;
    virtual double getWeight(void) const;
    virtual double getVolume(void) const;
    virtual double getEndPrice(void) const;
    virtual double getTaxPrice(void) const;

    // setters
    virtual void setOriginCountry(const std::string& cnt);
    void setAppearance(PACK_TYPE pck, std::shared_ptr< std::array<double, 3> > dimen);

    // operators
    virtual bool operator== (const BasicGoods& in) const ;
    virtual bool operator< (const BasicGoods& in) const;
};
```

member function های class Basic goods :

۱)constructor اصلی را تعریف می‌کنیم که شامل ورودی های زیر است.

- bName : نام برندکالا
- gtype : نوع کالا
- price : قیمت پایه

این تابع به صورت زیر پیاده سازی می‌شود.

```
BasicGoods::BasicGoods(const std::string& bName, GOODS_TYPE gTyp, double price) //constructor
{
    brandName = strdup(bName.c_str());    //convert string to char*

    goodsType = gTyp;                  //save goodsType
    basePrice = price;                //save base price

    originCountry = nullptr;          //initialize country
    taxRate = INITIAL_TAX_RATE;       //initialize tax rate
    packing = PACK_TYPE::UNKNOWN;     //initialize packing
    weight = 0.1;                     //initialize weight
}
```

۲) destructor : که در آن باید متغیر های dynamic را حذف کرد که به صورت زیر پیاده‌سازی می‌شود.

```
BasicGoods::~BasicGoods()        //destructor
{
    free(brandName);
    free(originCountry);
}
```

۳)getBrandname : نام برندکالا را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
std::string BasicGoods::getBrandName(void) const  //get brand name
{
    return brandName;
}
```

۴)getbaseprice : قیمت پایه را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double BasicGoods::getBaseprice(void) const   //get base price
{
    return basePrice;
}
```

۵)getGoodsType : نوع کالا را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
GOODS_TYPE BasicGoods::getGoodsType(void) const  //get goods type
{
    return goodsType;
}
```

۶)getWeight : وزن بسته بندی را برمیگرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double BasicGoods::getWeight(void) const  //get weight
{
    return weight;
}
```

۷)getVolume : حجم را برمی‌گرداند و به صورت زیر حساب می‌شود.

$$volume = approxDim[0] * approxDim[1] * approxDim[2]$$

```
double BasicGoods::getVolume(void) const   //get volume
{
    return (*(approxDim.get()->begin())) * (*(approxDim.get()->begin() + 1)) * (*(approxDim.get()->begin() + 2));
}
```

۸)getEndprice : قیمت پایانی را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$EndPrice = basePrice * (1 + \frac{taxRate}{100})$$

```
double BasicGoods::getEndPrice(void) const   //get end price
{
    return basePrice * (1 + taxRate / 100);
}
```

۹)getTaxPrice : مقدار مالیات را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$TaxPrice = basePrice * \frac{taxRate}{100}$$

```
double BasicGoods::getTaxPrice(void) const   //get tax price
{
    return basePrice * (taxRate / 100);
}
```

۱۰) setorigincountry : ورودی آن نام جدیدکشور مبدا است که جایگزین قبلی می‌شود و به صورت زیر پیاده‌سازی می‌شود.

```
void BasicGoods::setOriginCountry(const std::string& cnt)  //set country
{
    originCountry = strdup(cnt.c_str());          //convert string to char*
}
```

۱۱)setAppearance : که ورودی ها آن نوع جدید بسته بندی و ابعاد جدید بسته بندی است که به صورت زیر پیاده سازی می‌شود.

```
void BasicGoods::setAppearance(PACK_TYPE pck, std::shared_ptr<std::array<double, 3>> dimen)  //set new approxdim and packing
{
    packing = pck;
    approxDim = dimen;
}
```

۱۲)setNewPrice : ورودی آن مقدار جدید base price است و به صورت زیر پیاده‌سازی شود.

```
void BasicGoods::setNewPrice(double price)    //set new base price
{
    basePrice = price;
}
```

۱۳)operator== : در این تابع شرط برابری دو basic goods که brand name و good_type و base price باید یکی باشد ؛ چک می‌شود و به صورت زیر پیاده سازی می‌شود.

```
bool BasicGoods::operator==(const BasicGoods& in) const   //operator ==
{
    if (typeid(in) == typeid(BasicGoods) && goodsType == in.goodsType && *brandName == *in.brandName && basePrice == in.basePrice)
        return true;
    return false;
}
```

۱۴)operator> : که در آن endprice ها مقایسه می‌شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool BasicGoods::operator<(const BasicGoods& in) const  //operator <
{
    if (getEndPrice() < in.getEndPrice())
        return true;
    return false;
}
```

جواب Qusetion1 : اگر قصد داریم که کاربر نتواند object از این class کپی کند باید copy constructor های آن را به صورت protected تعریف میکنیم و اما اگر بخواهیم درکلاس هایی که از آن نیز به ارث میبرند هم نتوان کپی ساخت؛copy constructor ها را به صورت private تعریف می‌کنیم.

جواب Qusetion2 : اگر ما یک object از class derived که در پوینترکلاس base ذخیره شده داشته باشیم وقتی آن را delete کنیم اگر destructor به صورت virtual نباشد فقط destructor کلاسbase صدا زده می‌شود حال در این شرایط اگر درکلاس derived متغییر dynamic باشدکه باید در destructor حذف شود ، در این حالت این متغییر حذف نمی‌شود. ولی اگر destructor کلاس پایه به صورت virtual باشد هر دو destructor صدا زده می‌شود.

: class Food

ابتدا یک enum class به اسم FOOD_MODE که مشخص کننده نوع غذا است را تعریف و به صورت زیر پیاده سازی می‌کنیم.

```
enum class FOOD_MODE   //to classifiy food
{
    LIQUID,
    CREAM,
    SOLID
};
```

این کلاس از class Basic Goods به صورت publicارث بری می‌کند.

member variable های class Food :

- mode : مشخص کننده نوع غذا
- netweight : وزن غذا
- expirationTime : تاریخ مصرف

```
class Food : public BasicGoods
{
private:
    FOOD_MODE mode;
    double netWeight;          //weight of food
    std::time_t expirationTime;

public:

    Food(const std::string& name, double price, double Wei, FOOD_MODE md=FOOD_MODE::SOLID);  //constructor
    ~Food()=default;  //destructor

    constexpr static double FOOD_TAX_SUBSIDE = 3; // subtract this!

    // getters
    double getEndPrice(void) const;           //get end price
    double getTaxPrice(void) const override;  //get tax price
    double getWeight(void) const;             //get total weight
    std::time_t getRemainingExp(void) const;  //get remain expiration time

    // setters
    void setExpiration(std::time_t t);  //set exp
    void setFoodMode(FOOD_MODE md);     //set mode food

    //operator
    bool operator== (const BasicGoods& in) const;
    bool operator< (const BasicGoods& in) const;
    bool operator== (const Food& in) const ;
    bool operator< (const Food& in) const ;

};
```

member function های class Food :

۱)constructor : که شامل سه ورودی است :

- name : نام برندکالا
- price : قیمت پایه
- wei : وزن غذا
- md : مشخص کننده ی نوع غذا که به صورت پیش فرض soild است.

در اینجا ابتدا deligate می‌کنیم به constructor کلاس basic و سپس member variable های کلاس food را مقدار دهی می‌کنیم. توجه شود نوع بسته بندی و ابعاد طبق الگوریتم زیر تعیین می‌شود.

> LIQUID : choose a CAN packing, default dimensions: 0.3, 0.4, (0.2*netWeight)
>
> CREAM : choose a BARREL packing, default dimensions: 0.3, 0.4, (0.3*netWeight)
>
> SOLID : choose a NYLON packing, default dimensions: 0.3, 0.4, (0.7*netWeight)

```
Food::Food(const std::string& name, double price, double netWei, FOOD_MODE md) : BasicGoods{name,GOODS_TYPE::CONSUMABLE ,price} //constructor
{
    //initialize exp
    std::time_t now = std::time(nullptr);
    now += 14 * 24 * 3600;
    expirationTime = now;

    netWeight = netWei;  //initialize weight food

    mode = md;           //save food mode

    //set dimension food and packing
    if (md == FOOD_MODE::LIQUID)  //for liquid
    {
        packing = PACK_TYPE::CAN;
        approxDim = std::make_shared < std::array<double, 3>>(std::array<double, 3>{0.3, 0.4, 0.2 * netWei});
    }
    else if (md == FOOD_MODE::CREAM)  //for cream
    {
        packing = PACK_TYPE::BARREL;
        approxDim = std::make_shared < std::array<double, 3>>(std::array<double, 3>{0.3, 0.4, 0.3 * netWei});
    }
    else if (md == FOOD_MODE::SOLID)  //for solid
    {
        packing = PACK_TYPE::NYLON;
        approxDim = std::make_shared < std::array<double, 3>>(std::array<double, 3>{0.3, 0.4, 0.7 * netWei});
    }
}
```

۲) destructor : به صورت default تعریف می‌شود و باید توجه کرد هرموقع این destructor صدا زده می‌شود destructor کلاس basic نیز صدا زده می‌شود.

۳)getendPrice : قیمت پایانی را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$EndPrice = basePrice * (1 + \frac{taxRate - SUBSIE}{100})$$

```
double Food::getEndPrice(void) const  //get end price
{
    return basePrice * (1 + (taxRate - FOOD_TAX_SUBSIDE) / 100);
}
```

۴) getTaxPrice : مقدار مالیات را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$Tax\ price = basePrice * \frac{taxRate - SUBSIE}{100}$$

```cpp
double Food::getTaxPrice(void) const   //get tax price
{
    return basePrice * ((taxRate - FOOD_TAX_SUBSIDE) / 100);
}
```

۵)getWeight : وزن کل را برمیگرداند و به صورت زیر پیاده‌سازی می‌شود.

```cpp
double Food::getWeight(void) const   //get total weight
{
    return weight + netWeight;
}
```

۶)getRemainExp : زمان باقی مانده تا تاریخ انقضا را برمیگرداند.

```cpp
std::time_t Food::getRemainingExp(void) const   //get remain exp from now
{
    return expirationTime - std::time(nullptr);
}
```

۷)setExpiration : ورودی تاریخ انقضا جدید بوده و آن را جایگزین قبلی می‌کند.

```cpp
void Food::setExpiration(std::time_t t)   //set exp
{
    if (t - std::time(nullptr) >= 0)
        expirationTime = t;
    else
        std::cout << "invalid Time" << std::endl;
}
```

۸)setFoodmode : ورودی نوع جدید غذا بوده و آن را جایگزین قبلی می‌کند.

```cpp
void Food::setFoodMode(FOOD_MODE md)    //set food mode
{
    mode = md;
}
```

۹)operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت

حالت اول این است که ورودی food باشدکه باید شرایط تساوی دو food بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```cpp
bool Food::operator==(const Food& in) const   //operator ==
{
    if (getBrandName()==in.getBrandName() && getGoodsType()==in.getGoodsType() && basePrice==in.basePrice && mode==in.mode)
        return true;
    return false;
}
```

حالت دوم ورودی basic food است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```cpp
bool Food::operator==(const BasicGoods& in) const   //operator ==
{
    if (typeid(in) != typeid(Food))
        return false;
    else
    {
        BasicGoods* temp{ const_cast<BasicGoods*>(&in) };
        Food* temp_2{ static_cast<Food*>(temp) };
        return Food::operator==(*temp_2);
    }
}
```

۱۰)operator< : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت

حالت اول این است که ورودی food باشدکه باید شرایط > دو food بررسی شود(مقایسه تاریخ انقضا) و به صورت زیر پیاده‌سازی می‌شود.

```
bool Food::operator<(const Food& in) const    //opertor <
{
    if (expirationTime<in.expirationTime)
        return true;
    return false;
}
```

حالت دوم ورودی basic food است که در این حالت مقایسه قیمت تمام شده است و به صورت زیر پیاده‌سازی می‌شود.

```
bool Food::operator<(const BasicGoods& in) const   //operator <
{
    if (typeid(in) != typeid(Food))
    {
        if (getEndPrice() < in.getEndPrice())
            return true;
        else
            return false;
    }
    else
    {
        BasicGoods* temp{ const_cast<BasicGoods*>(&in) };
        Food* temp_2{ static_cast<Food*>(temp) };
        return Food::operator<(*temp_2);
    }
}
```

class Apliance :

ابتدا یک enum class به اسم ENERGY_COST که مشخص کننده سطح انرژي است را تعریف و به صورت زیر پیاده سازی می‌کنیم.

```
enum class ENERGY_COST
{
    IRRELEVANT,
    LOW,
    MEDIUM,
    HIGH
};
```

این کلاس از class Basic Goods به صورت publicارث بری می‌کند.

member variable های class Apliance :

- weight : وزن appliance
- energyGrade : سطح انرژي

```cpp
class Appliance : virtual public BasicGoods
{
protected:
    double weight;                          // which is netWeight
    ENERGY_COST energyGrade;                //energy grade

    double getEndPrice(void) const override;  //get end price

public:

    Appliance(const std::string& name, double price, double Wei); //constructor
    ~Appliance() = default;   //destructor


    constexpr static double APPLIANCE_EXTRA_TAX_HIGH = 17;
    constexpr static double APPLIANCE_EXTRA_TAX_MEDIUM = 7;

    using BasicGoods::setNewPrice;   //use setnewPrice for appliance as public
    using BasicGoods::approxDim;     //use approxDim for appliace as public

    //getter
    double getWeight(void) const ;     //get total weight
    double getTaxPrice(void) const;    //get end price

    //setters
    void setWeight(double w);          //set  weight appliance
    void setEnergyGrade(ENERGY_COST ec); //set energy Grade

    //oprator
    bool operator== (const BasicGoods& in) const;
    bool operator==(const Appliance& in);

};
```

member function های class Apliance :

۱)constructor : constructor که شامل سه ورودی است :

- name : نام برندکالا
- price : قیمت پایه
- wei : وزن appliance

در اینجا ابتدا deligate می‌کنیم به constructor کلاس basic و سپس member variable های کلاس appliance را مقدار دهی می‌کنیم.

```cpp
Appliance::Appliance(const std::string& name, double price, double Wei) : BasicGoods{name,GOODS_TYPE::OBJECT,price}  //constructor
{
    weight = Wei;   //save weight appliance
    energyGrade = {};  //initialize energy garade
}
```

۲)destructor : به صورت default تعریف می‌شود و باید توجه کرد هرموقع این destructor صدا زده می‌شود destructor کلاس basic نیز صدا زده می‌شود.

۳)getEndPrice : قیمت پایانی را طبق رابطه زیر حساب کرده و برمی‌گرداند. فقط توجه شود که مقدار extra tax برحسب سطح انرژی متفاوت است.

$$EndPrice = basePrice * (1 + \frac{taxRate + (APPLIANCE.EXTRA.TAX.?)}{100})$$

```cpp
double Appliance::getEndPrice(void) const    //get end price
{
    if (energyGrade == ENERGY_COST::HIGH)   //high
        return basePrice * (1 + (taxRate + APPLIANCE_EXTRA_TAX_HIGH) / 100);
    return basePrice * (1 + (taxRate + APPLIANCE_EXTRA_TAX_MEDIUM) / 100);  //otherwise
}
```

۴) getTaxPrice : مقدار مالیات را طبق رابطه زیر حساب کرده و برمی‌گرداند. فقط توجه شود که مقدار extra tax برحسب سطح انرژی متفاوت است.

$$Tax\ price = basePrice * \frac{taxRate + APPLIANCE.EXRA.TAX?}{100}$$

```cpp
double Appliance::getTaxPrice(void) const    //get tax price
{
    if (energyGrade == ENERGY_COST::HIGH)  //energy high
        return basePrice * ((taxRate + APPLIANCE_EXTRA_TAX_HIGH) / 100);
    return basePrice * ((taxRate + APPLIANCE_EXTRA_TAX_MEDIUM) / 100);  //other wise
}
```

۵) getWeight : وزن کل را برمیگرداند و به صورت زیر پیاده‌سازی می‌شود.

```cpp
double Appliance::getWeight(void) const //get total weight
{
    return BasicGoods::weight + weight;
}
```

۶) setWeight : ورودی ، وزن جدید appliance است که جایگزین قبلی می‌شود. این تابع به صورت زیر پیاده‌سازی می‌شود.

```cpp
void Appliance::setWeight(double w)   //set weight appliance
{
    weight = w;
}
```

۷) setEnergyGrade : ورودی ، سطح جدید انرژی است که جایگزین قبلی می‌شود. این تابع به صورت زیر پیاده‌سازی می‌شود.

```cpp
void Appliance::setEnergyGrade(ENERGY_COST ec)  //set energy grade
{
    energyGrade = ec;
}
```

۸) operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی appliance باشدکه باید شرایط تساوی دو appliance (سطح انرژی و وزن یکسان و شرایط تساوی basic goods)بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```cpp
bool Appliance::operator==(const Appliance& in)
{
    if (getBrandName() == in.getBrandName() && getGoodsType() == in.getGoodsType() && basePrice == in.basePrice
        && energyGrade == in.energyGrade && Appliance::weight==in.weight)
        return true;
    return false;
}
```

حالت دوم ورودی basic food است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool Appliance::operator==(const BasicGoods& in) const  //operator ==
{
    if(typeid(in)!=typeid(Appliance))
        return false;
    else
    {
        BasicGoods* temp{ const_cast<BasicGoods*>(&in) };
        Appliance* temp_2{ dynamic_cast<Appliance*>(temp) };
        return Appliance::operator==(*temp_2);
    }
}
```

جواب Qusetion3 : باید در قسمت public کلاس appliance نوشت :

```
using BasicGoods::setNewPrice;   //use setnewPrice for appliance as public
```

در این حالت برای object های class appliance تابع setNewPrice قابل دسترسی است.

جواب Qusetion4 : باید object این کلاس را به صورت پوینتری در class Basic goods ذخیره کرد و از این روش تابع getEndprice این کلاس قابل دسترسی است مانند زیر :

```
Appliance ag1{ "TV", 300, 5 };
BasicGoods* a{ &ag1 };
std::cout << a->getEndPrice() << std::endl;
```

جواب Qusetion5 : باید در قسمت public کلاس appliance نوشت :

```
using BasicGoods::approxDim;     //use approxDim for appliace as public
```

در این حالت برای object های class appliance متغییر approxDimقابل دسترسی است.

class Material :

ابتدا یک enum class به اسم MAT_TYPE که مشخص کننده نوع material است را تعریف و به صورت زیر پیاده سازی می‌کنیم.

```
enum class MAT_TYPE   //to classifiy material
{
    NORMAL,
    FLAMABLE,
    HAZARDOUS
};
```

این کلاس از class Basic Goods به صورت publicارث بری می‌کند.

member variable های class Material :

- weight : وزن appliance

- matType : مشخص کننده نوع material

```
class Material : virtual public BasicGoods
{
protected:
    double weigth;   //weight material
    MAT_TYPE matType;//material type

public:
    Material(const std::string& name, double price , double weight, MAT_TYPE mt=MAT_TYPE::NORMAL);   //constructor
    ~Material() = default;   //destructor

    constexpr static double MATERIAL_TAX = 1.5;


    // getters
    double getWeight(void) const;   //get total weight
    double getTaxPrice(void) const;  //get tax pice
    double getEndPrice(void) const;  //get end price

    //operator
    bool operator==(const Material& in) const;
    bool operator== (const BasicGoods& in) const;
};
```

member function های class Material :

۱) constructor : که شامل چهر ورودی است :

- name : نام برندکالا
- price : قیمت پایه
- wei : وزن Material
- mt : مشخص کننده نوع material که به صورت پیش فرض normal است.

در اینجا ابتدا deligate می‌کنیم به constructor کلاس basic و سپس member variable های کلاس appliance را مقدار دهی می‌کنیم.

```
Material::Material(const std::string& name, double price, double weight, MAT_TYPE mt) : BasicGoods{name,GOODS_TYPE::CONSUMABLE,price }  //constructor
{
    this->weigth = weight;   //save weight material
    matType = mt;            //save material energy
}
```

۲)destructor : به صورت default تعریف می‌شود و باید توجه کرد هرموقع این destructor صدا زده می‌شود destructor کلاس basic نیز صدا زده می‌شود.

۳)getWeight : وزن کل را برمیگرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double Material::getWeight(void) const  //get total weight
{
    return weigth + BasicGoods::weight;
}
```

۴)getEndPrice : قیمت پایانی را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$EndPrice = basePrice * (1 + \frac{taxRate + ExtraTax}{100})$$

```
double Material::getEndPrice(void) const   //get end price
{
    return basePrice * (1 + (taxRate + MATERIAL_TAX) / 100);
}
```

۵) getTaxPrice : مقدار مالیات را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$Tax\ price = basePrice * \frac{taxRate + EXRA\ TAX}{100}$$

```
double Material::getTaxPrice(void) const   //get tax price
{
    return basePrice * ((taxRate + MATERIAL_TAX) / 100);
}
```

۶)operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی Material باشدکه باید شرایط تساوی دو Material (وزن یکسان و شرایط تساوی basic goods) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool Material::operator==(const Material& in) const  //operator ==
{
    if (getBrandName() == in.getBrandName() && getGoodsType() == in.getGoodsType() && basePrice == in.basePrice && weigth==in.weigth)
        return true;
    return false;
}
```

حالت دوم ورودی basic food است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool Material::operator==(const BasicGoods& in) const
{
    if (typeid(in) != typeid(Material))
        return false;
    else
    {
        BasicGoods* temp{ const_cast<BasicGoods*>(&in) };
        Material* temp_2{ dynamic_cast<Material*>(temp) };
        return Material::operator==(*temp_2);
    }
}
```

class product ID :

member variable های class product ID :

- standardSerial : شماره سریال
- confirmStatment

- countryName : اسم کشور سازنده

```
class productID
{
protected:
    std::string standardSerial;   //serail number
    std::shared_ptr< std::string > confirmStatement; //confirmation satatment
    std::string countryName;   //country name


public:
    productID(std::shared_ptr< std::string >& in, const std::string& ss = {});  //constructor

    void printStatement(void) const;   //print confirmation
    friend std::ostream& operator<<(std::ostream& os, const productID& in); //operator <<
};
```

member function های class product ID :

۱) constructor : شامل دو ورودی :

- in : همان confirmStatment است.
- ss : همان standardSerial است که به صورت پیش فرض خالی است.

```
productID::productID(std::shared_ptr<std::string>& in, const std::string& ss)  //constructor
{
    confirmStatement = in;    //save confirmation statement
    standardSerial = ss;      //save serial number
}
```

۲)printStatment : این تابع confirmStatment در قالب خواسته شده چاپ می‌کند و به صورت زیر پیاده‌سازی می‌شود.

```
void productID::printStatement(void) const     //print confirmation
{
    std::cout << "Legal Confirmation Statement: [" << *confirmStatement << "] , producer country: " << countryName << std::endl;
}
```

۳)operator<<: کار این operator چاپ کردن ویژگی های product ID در قالب خواسته شده است. قابل توجه اینکه خروجی ostream است و prototype را به صورتfriend و در داخل خود classتعریف می‌شودکه باعث می شود به متغیر های private اجازه دسترسی شود و به صورت زیر پیاده سازی می‌شود .

```
std::ostream& operator<<(std::ostream& os, const productID& in)   //operator <<
{
    os << "Legal Confirmation Statement: [" << *(in.confirmStatement) << "]\nproducer country: " << in.countryName << " , serial number: " << in.standardSerial << std::endl;
    return os;
}
```

جواب Qusetion6 : به کلاسی گفته می‌شود که حداقل یکی از method های آن pure virtual باشد.

جواب Qusetion7 : constructorرا در قسمت protected می‌گذاریم.

class product:

این کلاس از class Basic Goods به صورت غیر مستقیم و به صورت مستقیم از prodict ID و material و appliance به صورت publicارث بری می‌کند.

نکته قابل توجه که در این کلاس چون از دوکلاس material و appliance ارث بری می‌کنیم فقط باید یک نسخه از basic goods وجود داشته باشد پس در دوکلاس مذکور basic goods به صورت virtual ارث برده شود.

member variable های class product:

- taxRate : مقدار مالیات product

```
class Product : public Appliance, public Material, public productID
{
private:
    double taxRate;   //tax rate product
public:
    Product(const std::string& brandName, double price, double matWei, shared_str confirmation);   //constructor
    ~Product() = default;  //destructor

    constexpr static double TYPICAL_TAX_RATE = 23;

    //getter
    double getTaxPrice(void) const;   //get tax price
    double getEndPrice() const;   //get end price
    double getWeight(void) const;  //get totol weight

    //setter
    void setTaxRate(double newTax);   //set new tax rate oroduct
    void setOriginCountry(const std::string& cnt);  //set origin country

    //opertator
    bool operator== (const BasicGoods& in) const;
    bool operator== (const Product& in) const;
};
```

member function های class product:

۱) constructor : شامل ۴ ورودی است :

- bname : نام برندکالا
- price : قیمت پایه
- mat wei: وزن Material
- confstate : همان confirmStatment است.

در اینجا ابتدا deligate می‌کنیم به constructor های کلاس basic و material و appliance و سپس member variable کلاس product را مقدار دهی می‌کنیم.

فقط توجه شود که goodType کلاس basic goods ، object است و وزن پیش فرض appliance برابر 0.3 بوده و نوع بسته بندی product (packing)، carton بوده و در آخر ابعاد (approxDim) به صورت $0.5 , 0.8 , 1.2 * (1 + matwei)$ مقداردهی می‌شود.

```
Product::Product(const std::string& brandName, double price, double matWei, shared_str confirmation) : BasicGoods{ brandName,GOODS_TYPE::OBJECT,price }
    ,Appliance{brandName,price,0.3} ,Material{brandName,price,matWei},productID{confirmation}
{   //constructor
    taxRate = TYPICAL_TAX_RATE;

    //set member variables
    BasicGoods::approxDim = std::make_shared < std::array<double, 3>>(std::array<double, 3>{0.3, 0.4, 1.2 * (1 + matWei)});
    BasicGoods::packing = PACK_TYPE::CARTON;
    Appliance::energyGrade = ENERGY_COST::LOW;
}
```

۲)destructor : به صورت default تعریف می‌شود و باید توجه کرد هرموقع این destructor صدا زده می‌شود destructor کلاس basic و material و appliance نیز صدا زده می‌شود.

۳)getWeight : وزن کل را برمیگرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double Product::getWeight(void) const  //get total weight
{
    return BasicGoods::weight + Material::weigth + Appliance::weight;
}
```

۴)getEndPrice : قیمت پایانی را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$EndPrice = basePrice * (1 + \frac{ProductTaxRate + ApplianceExtraTax}{100})$$

```
double Product::getEndPrice() const   //get end price
{
    if (energyGrade == ENERGY_COST::HIGH)
        return BasicGoods::basePrice * (1 + (taxRate + APPLIANCE_EXTRA_TAX_HIGH) / 100);   //for hight energy
    else
        return BasicGoods::basePrice * (1 + (taxRate + APPLIANCE_EXTRA_TAX_MEDIUM) / 100);  //other wise
}
```

۵) getTaxPrice : مقدار مالیات را طبق رابطه زیر حساب کرده و برمی‌گرداند.

$$Tax\ price = basePrice * \frac{ProductTaxRate + Appliance\ EXRA\ TAX}{100}$$

```
double Product::getTaxPrice(void) const  //get tax price
{
    if (energyGrade == ENERGY_COST::HIGH)
        return BasicGoods::basePrice * ((taxRate + APPLIANCE_EXTRA_TAX_HIGH) / 100);
    else
        return BasicGoods::basePrice * ((taxRate + APPLIANCE_EXTRA_TAX_MEDIUM) / 100);
}
```

۶)setTaxRate: ورودی، مقدار جدید taxRate است که جایگزین قبلی می‌شود. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
void Product::setTaxRate(double newTax)  //set tax rate
{
    taxRate = newTax;
}
```

۷) setorigincountry : ورودی آن نام جدیدکشور مبدا است که جایگزین قبلی می‌شود و به صورت زیر پیاده‌سازی می‌شود.

```
void Product::setOriginCountry(const std::string& cnt)  //set origin country
{
    BasicGoods::setOriginCountry(cnt);
    countryName = cnt;
}
```

۸)operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

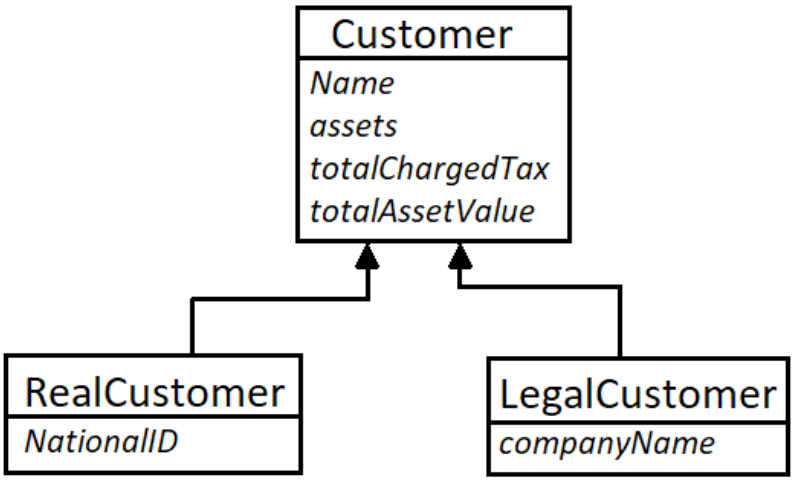
حالت اول این است که ورودی product باشدکه باید شرایط تساوی دو product (شرایط تساوی basic goods) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool Product::operator==(const Product& in) const
{
    if (getBrandName() == in.getBrandName() && getGoodsType() == in.getGoodsType() && getBaseprice() == in.getBaseprice())
        return true;
    return false;
```

حالت دوم ورودی basic food است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool Product::operator==(const BasicGoods& in) const    //operator ==
{
    if (typeid(in) != typeid(Product))
        return false;
    else
        return Product::operator==(in);
}
```

بعد از دسته بندی کالا ها ، به سراغ دسته‌بندی مشتری ها ، طبق الگوریتم زیر می‌رویم.



همانطورکه در شکل بالا مشخص شده ابتدا باید class Customer که کلاس پایه بوده و بقیه class ها از آن ارث بری می‌کنند را تعریف کنیم.

class Customer :

این کلاس abstract بوده و پایه دو کلاس بعدی است.

```cpp
class Customer
{
protected:
    std::string name;      //customer name
    double totalChargedTax;  //total tax price
    std::deque< std::shared_ptr<BasicGoods> > assets;   //all goods
    double assetValue;   //total end price

public:
    explicit Customer(const std::string& name);  //constructor
    virtual ~Customer() = default;               //destructor


    // simple functions
    using sharedBG = std::shared_ptr<BasicGoods>;
    std::string printBasicInfo(void) const;    //print info

    // getter
    double getTotalAsset(void) const;       //get asset value
    double getTotalChargedTax(void) const;  //ger total charged tax
    std::string getName(void) const;        //get name
    std::deque< std::shared_ptr<BasicGoods> > getassets(void);  //all goods

    // importers
    virtual void importNewGoods(std::shared_ptr<BasicGoods>);   //add new good

    template<class Iter>
    void importNewGoods(Iter start, Iter end);  //add new goods

    template <class T>
    void findGoods( std::deque< sharedBG >& list, T comp);  //find specific goods


    template <typename T>
    bool queueGoods(std::priority_queue< sharedBG, std::deque<sharedBG>, Shared_less<BasicGoods> >& Q);  //find same kind goods

    // operators
    virtual bool operator==(const Customer& in) const = 0;
    virtual bool operator<(const Customer& in) const = 0;
};
```

member variable های class Customer :

- name : اسم مشتری
- totalChargedTax : مجموع تمام tax price ها است.
- assets : کالا های هر مشتری
- asssetValue : مجموع endPrice ها است.

member function های class Customer :

۱)constructor : فقط یک ورودی که آن هم اسم مشتری بوده ، دارد و به صورت زیر پیاده‌سازی می‌شود.

```cpp
Customer::Customer(const std::string& name)  //constructor
{
    this->name = name;      //save name
    assetValue = 0;         //initialize asswetValue
    totalChargedTax = 0;    //initialize total charged tax
}
```

۲)destructor : به صورت default تعریف می‌شود

۳)printBasicInfo : این تابع بعضی از اطلاعات مشتری را چاپ می‌کند و به صورت زیر پیاده‌سازی می‌شود.

```
std::string Customer::printBasicInfo(void) const    //print info
{
    std::ostringstream print_InFo{};
    print_InFo << name << " >> No of assets: " << assets.size() << ", Total Assest: " << assetValue << ", Charged Tax: " << totalChargedTax;
    return print_InFo.str();
}
```

۴)getTotalAsset : مقدار asssetValue را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double Customer::getTotalAsset(void) const   //get total asset
{
    return assetValue;
}
```

۵)getTotalChargedTax : مقدار totalChargedTax را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
double Customer::getTotalChargedTax(void) const  //get total charged tax
{
    return totalChargedTax;
}
```

۶)getName: اسم مشتری را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
std::string Customer::getName(void) const   //get name
{
    return name;
}
```

۷)getassets : کل کالا های مشتری را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
std::deque<std::shared_ptr<BasicGoods>> Customer::getassets(void)  //get assets (goods)
{
    return assets;
}
```

۸)importNewGodds :این تابع به دو حالت نوشته می‌شود :

حالت اول) ورودی این تابع پوینتر یک کالا است که میخواهیم به مجموعه کالاهای مشتری اضافه کنیم و در این جا باید مقدار asssetValue و totalChargedTax با توجه به کالای جدید به روز رسانی شود و به صورت زیر پیاده‌سازی می‌شود.

```
void Customer::importNewGoods(std::shared_ptr<BasicGoods> in)    //add new good
{
    assets.push_back(in);                //add new good
    totalChargedTax += in->getTaxPrice(); //update total charged tax
    assetValue += in->getEndPrice();      //update asset value
}
```

حالت دوم)ورودی تابع iteratorیک container است که کالاها در آن قرار دارند و میخواهیم به مجموعه کالاهای مشتری اضافه کنیم و در این جا باید مقدار asssetValue و totalChargedTax با توجه به کالاهای جدید به روز رسانی شود و به صورت زیر پیاده‌سازی می‌شود.

```
template<class Iter>
void Customer::importNewGoods(Iter start, Iter end)   //add new goods
{
    if (start == end)
        assets.push_back(*start);
    for (auto it{ start }; it != end; it++)
        importNewGoods(*it);
}
```

۹)findGoods : کار این تابع پیدا کردن کالاهایی است که یک شرط خاص را ارضا می‌کند و آن کالا ها را در container که در ورودی به صورت رفرنسی داده شده ذخیره می‌کند. در این تابع برای پیدا کردن کالاهای مورد نظر از دستور copy_if استفاده می‌کنیم . این تابع به صورت زیر پیاده‌سازی می‌شود.

```
template<class T>
void Customer::findGoods(std::deque<sharedBG>& list, T comp)  //find specific goods
{
    auto it = std::copy_if(assets.begin(), assets.end(), list.begin(), comp);
    list.resize(std::distance(list.begin(), it));    //resize out put
}
```

برای مقایسه دو کالا باید class shared_less را به صورت زیر تعریف کنیم.

```
template<typename T>
class Shared_less
{

public:
    bool operator()(const std::shared_ptr<T>& a,const std::shared_ptr<T>& b) const {
        return (*a.get() < *b.get());
    }
};
```

۱۰)queueGoods : این تابع کالاهایی که از یک جنس هستند را در container که در ورودی به صورت رفرنسی داده شده ذخیره می‌کند. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
template<typename T>
bool Customer::queueGoods(std::priority_queue<sharedBG, std::deque<sharedBG>, Shared_less<BasicGoods>>& Q)  //find same kind goods
{
    for (size_t i{}; i < assets.size(); i++)
        if (typeid(T) == typeid(*assets[i].get()))
            Q.push(assets[i]);

    if (Q.empty())
        return false;
    return true;
}
```

۱۱)operator== : به صورت pure virtual بوده و باید درکلاس‌هایی که از آن ارث بری می‌شود تعریف شود.

۱۲)operator<< : به صورت pure virtual بوده و باید درکلاس‌هایی که از آن ارث بری می‌شود تعریف شود.

جواب Qusetion8 : باعث تغییر مکان عناصر شده (چون بعضی از عناصر حذف می‌شودند) و دیگر مشکلی که به وجود می‌آید این است که با استفاده مکرر باعث overhead زیاد برای جابهجایی می‌شود.

class Real Customer :

این کلاس به صورت public از costomer ارث می‌برد.

member variable های class Real Customer :

- nationalID

```
class RealCustomer : public Customer
{
private:
    long int nationalID;   //national ID

public:

    RealCustomer(const std::string& name, long int nid=0);   //constructor
    ~RealCustomer() = default;        //destructor

    long int getNationalID(void);    //get national ID

    //operators
    virtual bool operator==(const Customer& in) const override;
    virtual bool operator<(const Customer& in) const override;
    bool operator==(const RealCustomer& in) const;
    bool operator<(const RealCustomer& in) const;
};
```

member function های class Real Customer :

۱) constructor: شامل دو وردی است :

- name : اسم مشتری
- nid : همان nationalID که به صورت پیش فرض صفر است.

در اینجا ابتدا deligate می‌کنیم به constructor کلاس Customer و سپس member variable کلاس Real Customer را مقدار دهی می‌کنیم.

```
RealCustomer::RealCustomer(const std::string& name, long int nid) :Customer{name}  //constructor
{
    nationalID = nid;   //save national ID
}
```

۲)destructor : به صورت default تعریف می‌شود

۳)getNationalID : این تابع مقدار nationalID را برمی‌گرداند و به صورت زیر پیاده‌سازی می‌شود.

```
long int RealCustomer::getNationalID(void)  //get national ID
{
    return nationalID;
}
```

۴) operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی Real Customer باشدکه باید شرایط تساوی دو Real Customer (برابری nationalID و اسم) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool RealCustomer::operator==(const RealCustomer& in) const
{
    if (name == in.name && nationalID == in.nationalID)
        return true;
    return false;
}
```

حالت دوم ورودی legal Customer است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool RealCustomer::operator==(const Customer& in) const
{
    if (typeid(in) != typeid(RealCustomer))
        return false;
    Customer* a{ const_cast<Customer*>(&in) };
    RealCustomer* b = static_cast<RealCustomer*>(a);
    return RealCustomer::operator==(*b);
}
```

۵) operator< : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی Real Customer باشدکه باید شرایط مقایسه دو Real Customer (مقایسه اسم) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool RealCustomer::operator<(const RealCustomer& in) const
{
    if (RealCustomer::operator==(in))
        return false;
    else
        if (name < in.name)
            return true;
    return false;
}
```

حالت دوم ورودی legal Customer است که در این حالت تساوی برقرار است و به صورت زیر پیاده‌سازی می‌شود.

```
bool RealCustomer::operator<(const Customer& in) const
{
    if (typeid(in) == typeid(RealCustomer))
    {
        Customer* a{ const_cast<Customer*>(&in) };
        RealCustomer* b = static_cast<RealCustomer*>(a);
        return RealCustomer::operator<(*b);
    }
    return true;;
}
```

class legal Customer :

این کلاس به صورت public از costomer ارث می‌برد.

member variable های class legal Customer :

- company name : نام کمپانی

```
class LegalCustomer : public Customer
{
private:
    std::string companyName;   //company name

public:
    LegalCustomer(const std::string& corrName, const std::string& compName);  //constructor
    ~LegalCustomer() = default;      //destructor

    //opeartor
    virtual bool operator==(const Customer& in) const override;
    virtual bool operator<(const Customer& in) const override;
    bool operator==(const LegalCustomer& in) const;
    bool operator<(const LegalCustomer& in) const;
};
```

member function های class legal Customer :

۱) constructor : شامل دو ورودی است :

- corrName : اسم مشتری
- comName : نام کمپانی

در اینجا ابتدا deligate می‌کنیم به constructor کلاس Customer و سپس member variable کلاس legal Customer را مقدار دهی می‌کنیم.

```
LegalCustomer::LegalCustomer(const std::string& corrName, const std::string& compName) :Customer{corrName}
{
    companyName = compName;   //save copmany name
}
```

۲)destructor : به صورت default تعریف می‌شود

۳) operator== : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی legal Customer باشدکه باید شرایط تساوی دو legal Customer (برابری نام کمپانی و اسم) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool LegalCustomer::operator==(const LegalCustomer& in) const
{
    if (name == in.name && companyName == in.companyName)
        return true;
    return false;
}
```

حالت دوم ورودی Real Customer است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool LegalCustomer::operator==(const Customer& in) const
{
    if (typeid(in) != typeid(LegalCustomer))
        return false;
    Customer* a{ const_cast<Customer*>(&in) };
    LegalCustomer* b = static_cast<LegalCustomer*>(a);
    return LegalCustomer::operator==(*b);
}
```

۴) operator< : برحسب جنس ورودی آن به دو صورت باید این تابع را نوشت.

حالت اول این است که ورودی legal Customer باشدکه باید شرایط مقایسه دو legal Customer (مقایسه اسم) بررسی شود و به صورت زیر پیاده‌سازی می‌شود.

```
bool LegalCustomer::operator<(const LegalCustomer& in) const
{
    if (LegalCustomer::operator==(in))
        return false;

    if (name < in.name)
        return true;
    return false;
}
```

حالت دوم ورودی Real Customer است که در این حالت تساوی برقرار نیست و به صورت زیر پیاده‌سازی می‌شود.

```
bool LegalCustomer::operator<(const Customer& in) const
{
    if (typeid(in) == typeid(LegalCustomer))
    {
        Customer* temp { const_cast<Customer*>(&in) };
        LegalCustomer* b = static_cast<LegalCustomer*>(temp);
        return LegalCustomer::operator<(*b);
    }
    return false;
}
```

class Storehouse :

هدف این کلاس ذخیره مشتری‌های یک انبار است.

member variable های class Storehouse :

- storeName : نام انبار
- theCustomers : یک set است اطلاعات در مشتری ها را در خود دارد.

```
class Storehouse
{
private:
    std::string storeName;   //store name
    std::set < std::shared_ptr<Customer>, Shared_less<Customer>> theCustomers;  //all customer

public:
    explicit Storehouse(const std::string& name);   //constructor
    Storehouse() = default;   //default constructor

    void newCustomer(std::shared_ptr<Customer> newC );   //add new customer
    std::string printCustomers(void) const;     //print detail of all customer
};
```

member function های class Storehouse :

۱) constructor : شامل یک ورودی است که آن هم نام انبار است و به صورت زیر پیاده‌سازی می‌شود.

```
Storehouse::Storehouse(const std::string& name)   //constructor
{
    storeName = name;   //save name storehouse
}
```

۲) default constructor : به صورت default تعریف می‌شود

۳) newCustomer : کار این اضافه کردن مشتری است که در ورودی به آن داده شده و باید آن را در داخل set ذخیره کند . ابتدا باید چک شود که مشتری ورودی تکراری نباشد و اگر بود باید فقط کالاهای آن به قبلی ها اضافه شود. برای اضافه کردن کالاهای جدید به یک مشتری از تابع importgoods استفاده می‌کنیم. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
void Storehouse::newCustomer(std::shared_ptr<Customer> newC)  //add new customer
{
    bool flag{ false };   //to show new customer added

    for (auto it = theCustomers.begin(); it != theCustomers.end(); it++)  //cheack for repeat customer
    {
        if ((*(*it)) == *newC)   //if find repeated customer
        {
            size_t size{ (*newC).getassets().size() };
            for (size_t i{}; i < size; i++)  //import all new goods
                (*(*it)).importNewGoods((*newC).getassets()[i]);
            flag = true;  //show add new customer
            break;
        }
    }
    if(!flag)  //if not added new customer yet add it
        theCustomers.insert(newC);
}
```

توجه شود قسمت امتیازی نیز انجام شده است

۴) printCustomer : این تابع اطلاعات مشتری‌های انبار را در قالب خواسته شد نمایش می‌دهد. این تابع به صورت زیر پیاده‌سازی می‌شود.

```
std::string Storehouse::printCustomers(void) const   //print detail of all customer
{
    std::ostringstream print_InFo{};
    for (auto it = theCustomers.begin(); it != theCustomers.end(); it++)
    {
        if (typeid(LegalCustomer) == typeid(*(*it)))
            print_InFo << "Legal : ";
        else if (typeid(RealCustomer) == typeid(*(*it)))
            print_InFo << "Real : ";
        print_InFo << (*it).get()->printBasicInfo() << std::endl;
    }
    std::cout << print_InFo.str() << std::endl;

    return print_InFo.str();
}
```

توجه۱ : در ورودی بعضی از تابع ها ؛ دلیل اینکه ورودی را به صورت refernce می‌دهیم این است که باعث افزایش performance و صرفه‌جویی در حافظه می‌شود. و در برخی موارد ورودی را به صورت const هم می‌دهیم که این کار برای جلوگیری از اجازه تغییر در تابع است.

توجه۲: توضیحات مربوط به پیاده سازی الگوریتم ها در داخل کد به صورت کامنت نوشته شده است و در این گزارش کار سعی شده در مورد الگوریتم توابع توضیح داده شود.

نتایج google test به شرح زیر است.

```
[==========] Running 10 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 1 test from ST_Begin
[ RUN      ] ST_Begin.check
[       OK ] ST_Begin.check (0 ms)
[----------] 1 test from ST_Begin (1 ms total)

[----------] 5 tests from ST_Goods
[ RUN      ] ST_Goods.Basic
[       OK ] ST_Goods.Basic (0 ms)
[ RUN      ] ST_Goods.Food
[       OK ] ST_Goods.Food (0 ms)
[ RUN      ] ST_Goods.Appliance
[       OK ] ST_Goods.Appliance (0 ms)
[ RUN      ] ST_Goods.Material
[       OK ] ST_Goods.Material (0 ms)
[ RUN      ] ST_Goods.Product
[       OK ] ST_Goods.Product (0 ms)
[----------] 5 tests from ST_Goods (5 ms total)

[----------] 3 tests from ST_Customer
[ RUN      ] ST_Customer.Reals
[       OK ] ST_Customer.Reals (0 ms)
[ RUN      ] ST_Customer.Legals
[       OK ] ST_Customer.Legals (0 ms)
[ RUN      ] ST_Customer.CUSTOMER
[       OK ] ST_Customer.CUSTOMER (0 ms)
[----------] 3 tests from ST_Customer (3 ms total)

[----------] 1 test from ST_Stoorehouse
[ RUN      ] ST_Stoorehouse.Store
Real : Sadeghi Hariry >> No of assets: 6, Total Assest: 12032.4, Charged Tax: 2462.4
Legal : MehdiAkhavan >> No of assets: 2, Total Assest: 672, Charged Tax: 72

[       OK ] ST_Stoorehouse.Store (1 ms)
[----------] 1 test from ST_Stoorehouse (1 ms total)

[----------] Global test environment tear-down
[==========] 10 tests from 4 test suites ran. (16 ms total)
[  PASSED  ] 10 tests.
```

بخش دوم python

در بخش اول باید یک تابع بنویسیم که اسم فایل عکس و یک تابع به عنوان ورودی می‌گیرد .

ایتدا فایل عکس را خوانده و تبدیل به یک ارایه از byte ها می‌کنیم بعد از آن طبق تابع ورودی تغیرات را بر روی byteها انجام می‌دهیم سپس در آخر فایل جدید را می‌سازیم.

```
def code(filname,fun):
    new_byte=[]
    with open(filname,"rb") as image:
        f=image.read()          #read image
        byte = bytearray(f)     #get byte of image

    new_byte=map(fun,byte)    #maping

    with open(filname,"wb") as new_image:  #write new image
        for i in new_byte:
            new_image.write(bytes([i]))
```

جواب Qusetion9 : اگر دوبارکد را اجرا کنیم به فایل اصلی می‌رسیم و همه چی خنثی می‌شود.(مثل دوبار معکوس کردن است که برابر خودش می‌شود.)

در بخش دوم باید فاکتوریل را با استفاده از dynamic programming پیاده سازی‌کنیم.

یک کلاس به اسم Factorial تعریف میکنم و در آن ()operator را تعریف کرده و برای محاسبه فاکتوریک از الگوریتم بازگشتی استفاده می‌کنیم . توجه شود که مقدار حساب شده در هر مرحله را در یک dictionary ذخیره میکنیم و در مرحله محاسبه فاکتوریل ابتدا چک میکنیم که ایا فاکتوریل مورد نظر قبلا حساب شده یا نه.

```
class Factorial:
    def __init__(self):  #
        self.dict={1:1}

    def __call__(self,number):    #operator()
        return self.calculate(number)   #claculate factorial

    def calculate(self,temp):  #claculate factorial
        for key,val in self.dict.items():  #if factorial has been computed
            if key==temp :
                return val

        # if factorial has not been computed ,compute it!
        if temp==1:
            return 1
        else:
            result=self.calculate(temp-1)*temp
            self.dict.update({temp:result})
            return result
```

در آخر نتایج تست ها:

```
----------------------------------------------------------------------
Ran 2 tests in 0.119s

OK
```

پایان