

Class Operations

- Pointer and References with class types
- Class destructor
- Dynamic Memory Allocation within a class
- Copy constructor
- Limit access to a class
- Nested class

Pointers and References to Class Objects

- Pointers and references to class objects are key features of OOP
- Three basic contexts to use a pointer to a class
 - Calling functions using the dereference operator, ->
 - As an argument to a function
 - As a data member of a class

How to pass parameters?

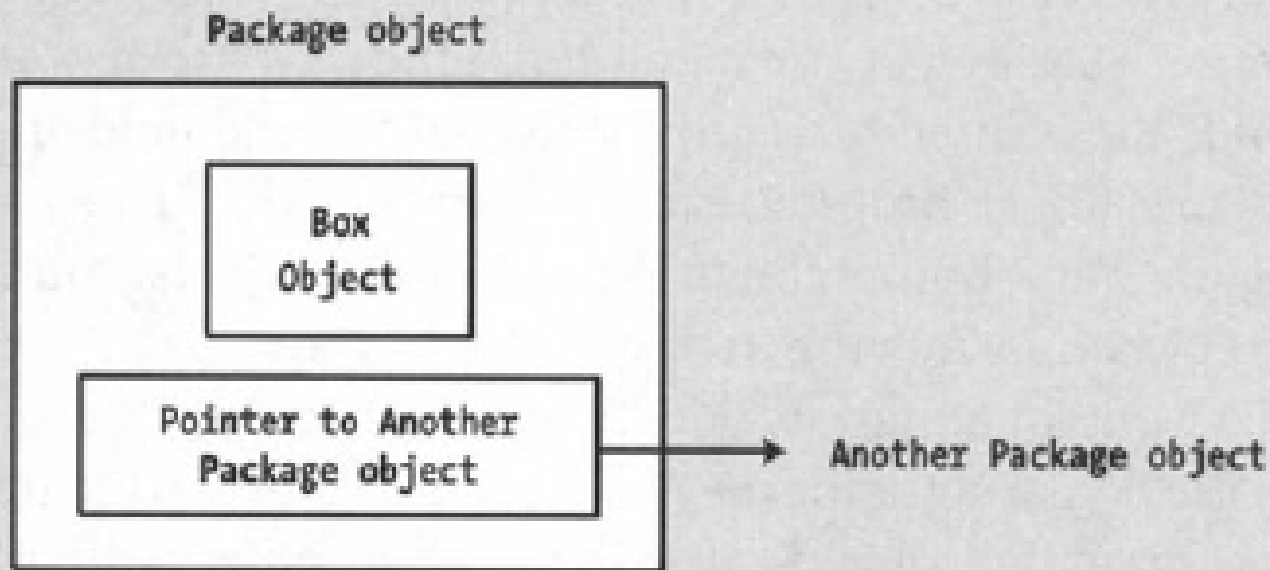
- Function parameters
 - Small size of object (pass by value, pointer or reference)
 - Array object (pass by pointer)
 - struct object (pass by either pointer or reference)
 - class object (pass by reference)

Container class

- A class is defined to contain other classes in private members

Pointers as data members

- The contents of a Package object

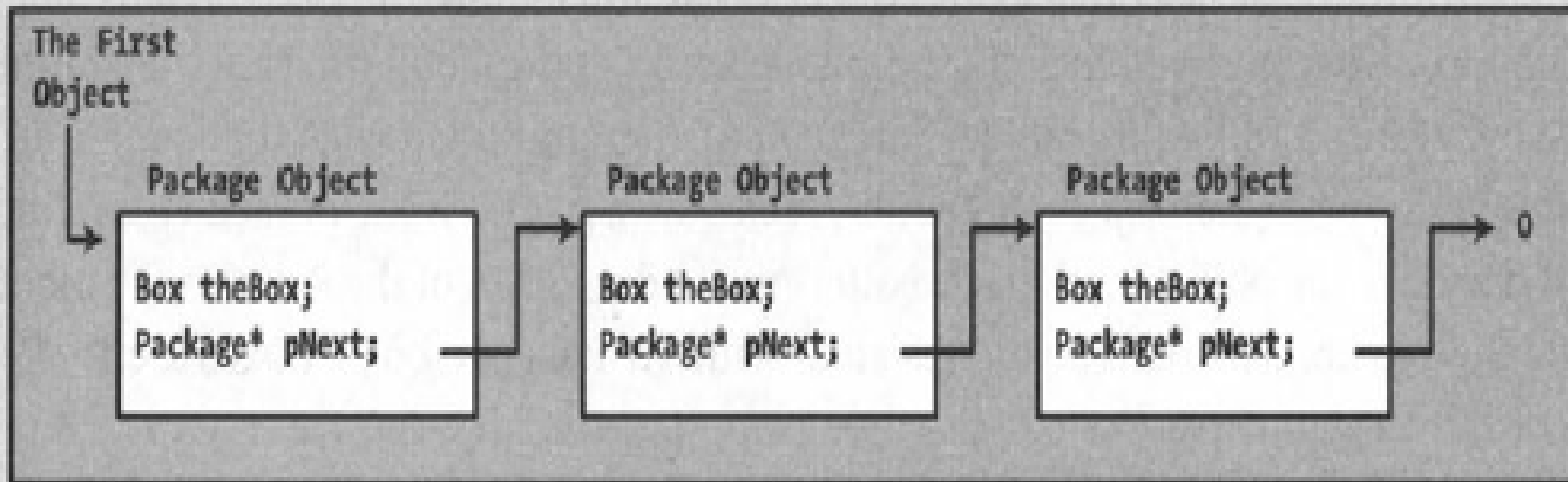


A Package object can contain a Box object, and can be linked to another Package object that contains another Box object.

TruckLoad

- A linked list of three objects

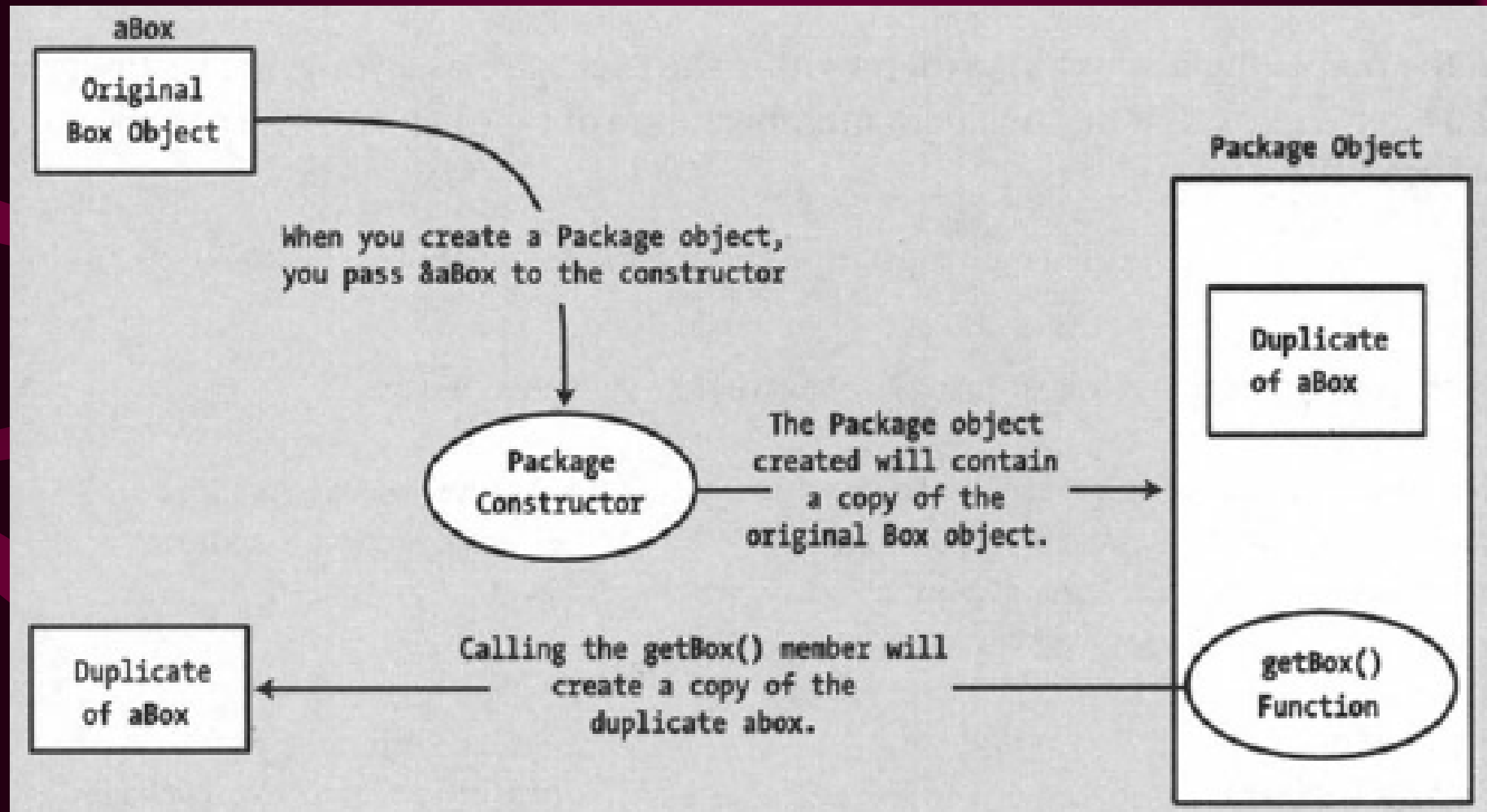
TruckLoad Object



Defining the Package Class

[illegible]

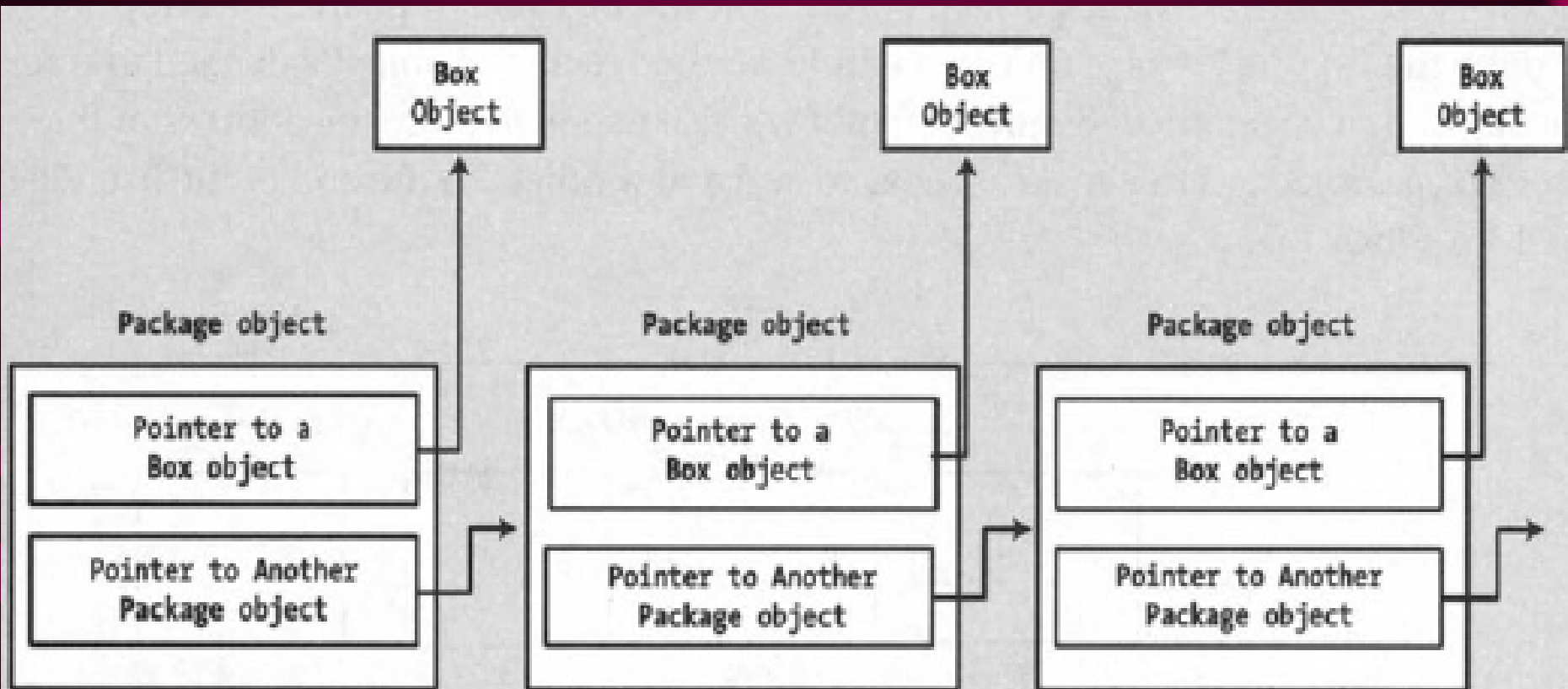
Copies of Box objects everywhere



Revised Package Class

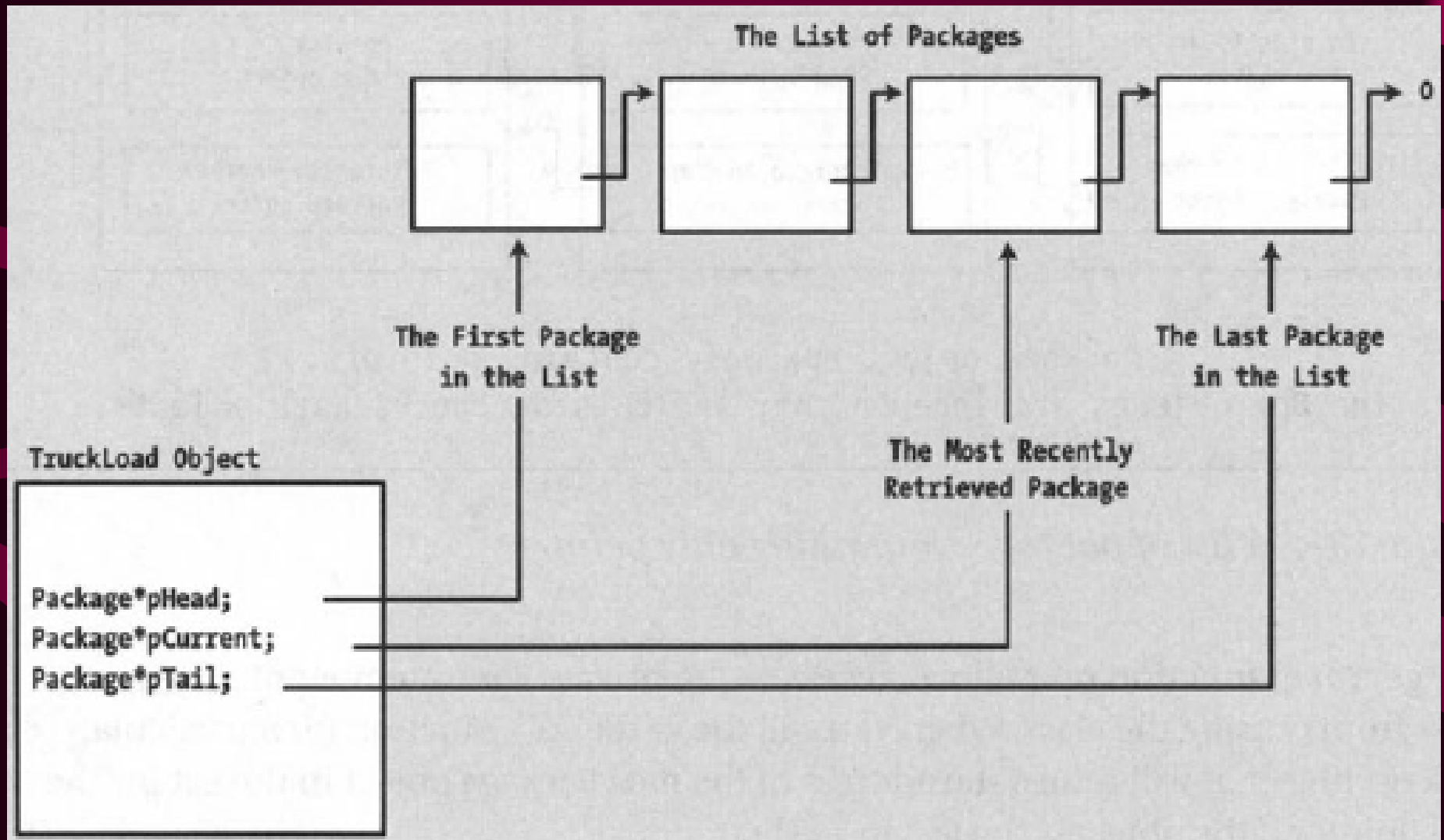
[illegible]

A list of packages containing only pointers



A Package object now only contains pointers.
The Box objects are independent, and outside the Package objects.

Defining the TruckLoad Class



Implementing the TruckLoad Class

```
TruckLoad::TruckLoad(Box* pBox, int count) {  
    pHead = pTail = pCurrent = 0;  
  
    if((count > 0) && (pBox != 0))  
        for(int i = 0 ; i<count ; i++)  
            addBox(pBox+i);  
}
```

Defining the Member Functions

```
void TruckLoad::addBox(Box* nBox) {
    Package* pPackage = nBox;
    Box* TruckLoad::getNextBox() {
        if(pCurrent)
            pCurrent = pCurrent->getNext();
        else
            pCurrent = pHead;
        pTail->setNext(pCurrent);
        return pCurrent ? pCurrent->getBox() : 0;
    }
    pHead = pPackage;
    pTail = pPackage;
}
```

```
Box* TruckLoad::getFirstBox() {
    pCurrent = pHead;
    return pCurrent->getBox();
}
```

- Program 13.1

Three problems in prog13.1

- The Package class is accessible to all, even though one only use this class in the context of the TruckLoad class
- The duplication of a TruckLoad object
- A very serious memory leak with memory management

- A **nested class** definition in a **class**
- One class **TruckLoad** (TruckLoad) and the **TruckLoad** others

```

class TruckLoad {
public:
    TruckLoad(Box* pBox = 0, int count = 1);

    Box* getFirstBox();
    Box* getNextBox();
    void addBox(Box* pBox);

private:
    // Class defining a list element
    class Package {
    public:
        Box* pBox;
        Package* pNext;
        void setNext(Package* pPackage);
        Package(Box* pNewBox);
    };

    Package* pHead;
    Package* pTail;
    Package* pCurrent;
};

```



```

void TruckLoad::addBox(Box* pBox) {
    Package* pPackage = new Package(pBox);

    if(pHead)
        pTail->pNext = pPackage;
    else
        pHead = pPackage;
    pTail = pPackage;
}

```

```

Box* TruckLoad::getFirstBox() {
    pCurrent = pHead;
    return pCurrent->pBox;
}

```

```

Box* TruckLoad::getNextBox() {
    if(pCurrent)
        pCurrent = pCurrent->pNext;
    else
        pCurrent = pHead;

    return pCurrent ? pCurrent->pBox : 0;
}

```

- Revised program 13

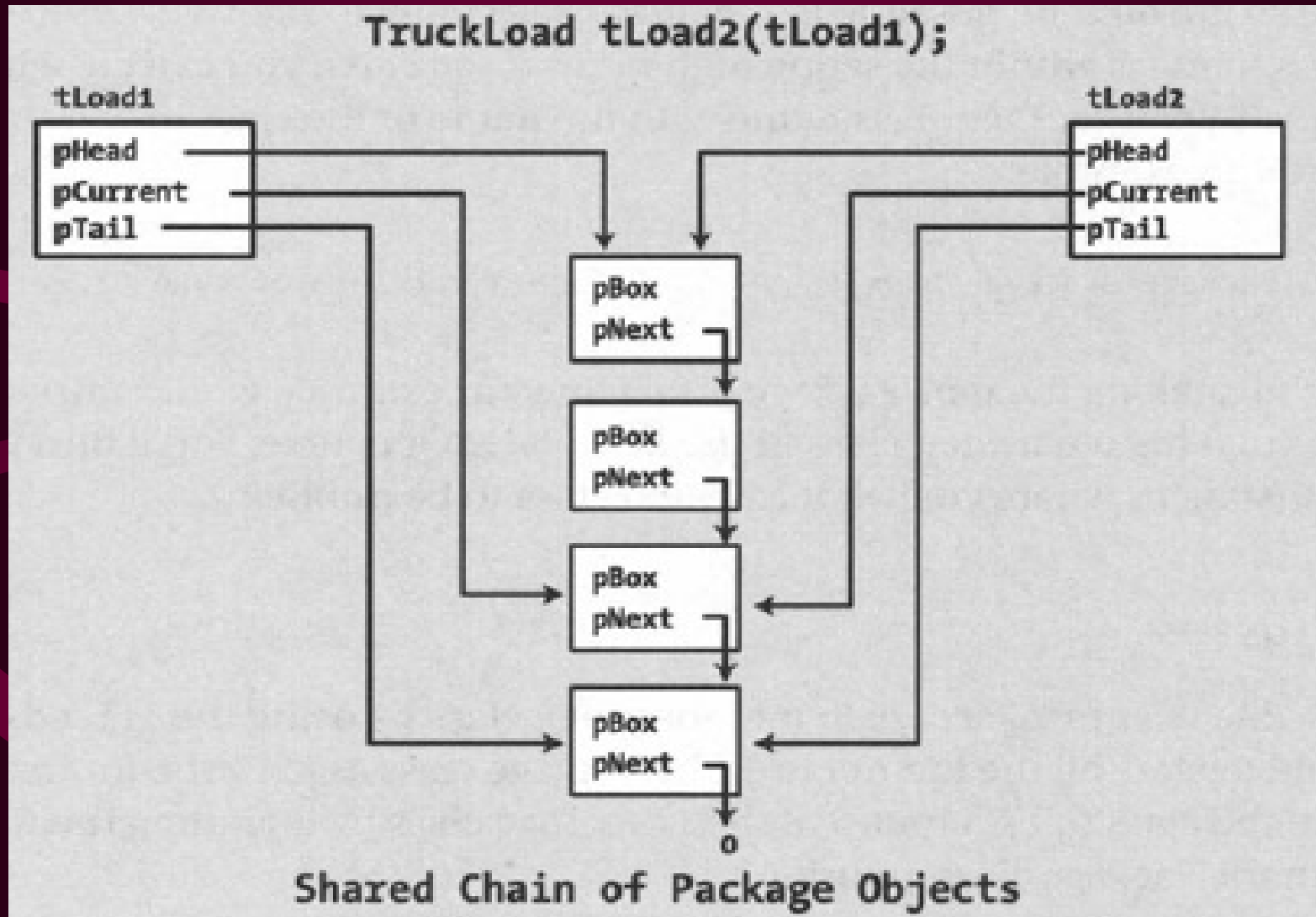
Nested Classes with Public Access Specifiers

```
TruckLoad::Package aPackage(aBox);    // Define a variable of type Package
```

The Copy Constructor

- Copy one object from an existent one
 - `TruckLoad tLoad2(tLoad1);`
- If copy constructor is not defined then compiler would create a default one, but no practical space is created for the pointer member which can cause memory problem
- A self-defined copy constructor is necessary when pointer members are declared in a class

Duplicating an object using the default copy constructor



Self-defined Copy Constructor

```
TruckLoad::TruckLoad(const TruckLoad& load)
{
    pHead = pTail = pCurrent = 0;
    if (load.pHead) return;
    // save address for new chain
    Package* pTemp = load.pHead;
    do
    { addBox(pTemp->pBox);}
    while (pTemp = pTemp->pNext);
}
```

- Revised program 13.2

Dynamic Memory Allocation Within an Object

- Constructor without 'new', preset destructor executed automatically
- Constructor with 'new', destructor should self-defined it by 'delete'
- The default destructor

```
• Rev TruckLoad::~~TruckLoad() {  
    // Code to destroy the object  
}
```

Implementing a Destructor

```
TruckLoad::~~TruckLoad() {  
    cout << "TruckLoad destructor called." << endl;  
    while(pCurrent = pHead->pNext) {  
        delete pHead;                // Delete the previous  
        pHead = pCurrent;            // Store address of next  
    }  
    delete pHead;                    // Delete the last  
}
```

References in Classes

```
TruckLoad* TruckLoad::getFirstBox() {  
    pCurrent = pHead;  
    if (pCurrent != 0)  
        return &pCurrent->rBox;  
}
```

```
Package* TruckLoad::getNextBox() {  
    do {  
        if(pCurrent)  
            pCurrent = pCurrent->pNext;  
        else  
            pCurrent = pHead;  
    } while (pCurrent != 0);  
}
```

Saves address of next box

Assign address of next box

- Return the first box in the load
return pCurrent ? &pCurrent->rBox : 0;