

Module 5: Inheritance

Prof. Tran Minh Triet

Acknowledgement

❖ Slides

- Course CS202: Programming Systems
Instructor: MSc. Karla Fant,
Portland State University
- Course CS202: Programming Systems
Instructor: Dr. Dinh Ba Tien,
University of Science, VNU-HCMC
- Course DEV275: Essentials of Visual Modeling with
UML 2.0
IBM Software Group

Outline

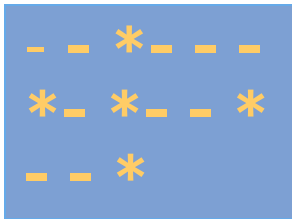
- ❖ Introduction to inheritance
- ❖ Hierarchical organization of concepts
- ❖ Types of inheritance
- ❖ Derived class
 - Constructor
 - Destructor
 - Copy constructor & assignment operator

Introduction

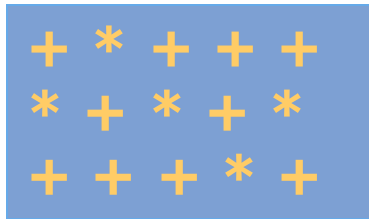
- ❖ A new concept does not come alone. When we introduce a class of car or employee for example. It may lead us to describe:
 - wheel, engine, driver etc.
 - Or: manager, director...
- ❖ To model them, we can use class. However, how can we model the relationship between them?
- ❖ In addition, re-usability of existing classes is one of the features of OOP.

Inheritance

A



B

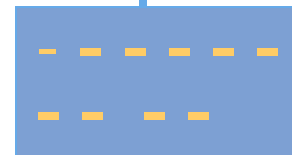


* common attribs/functions
- only in A
+ only in B

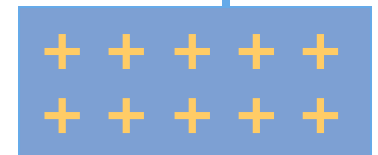
C



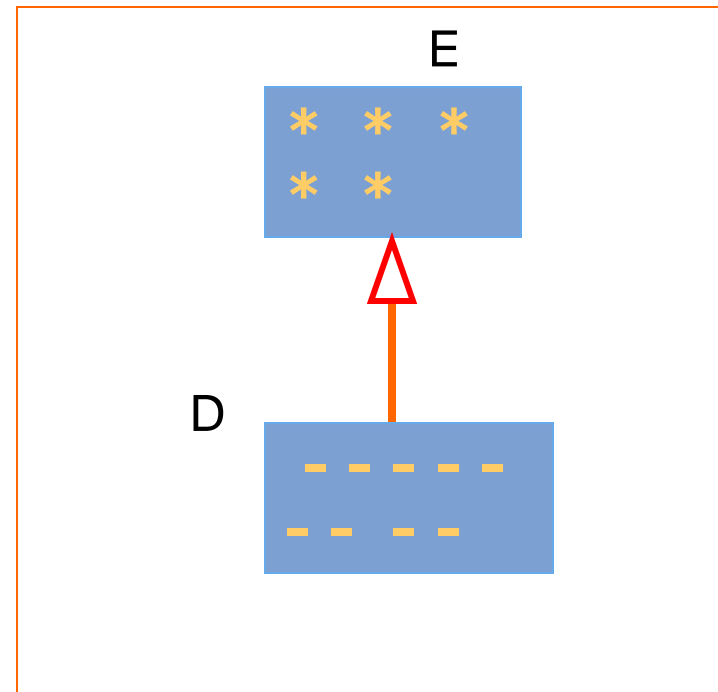
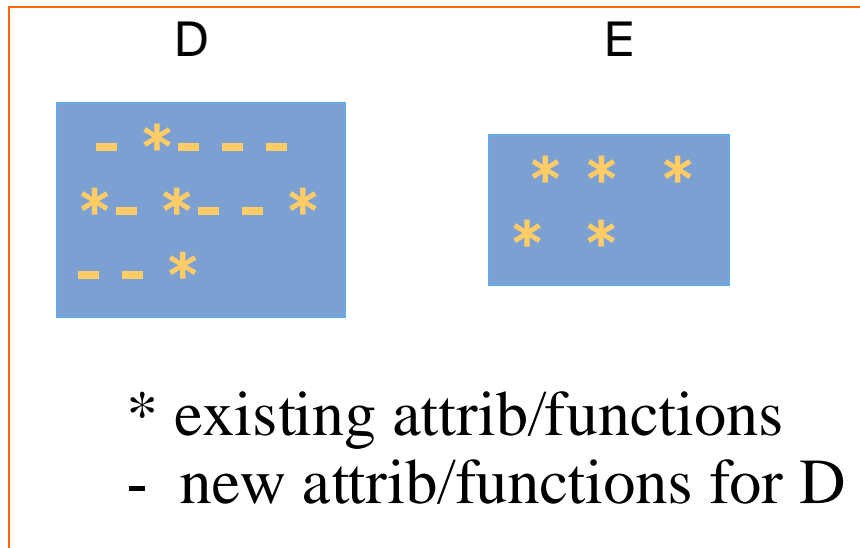
A



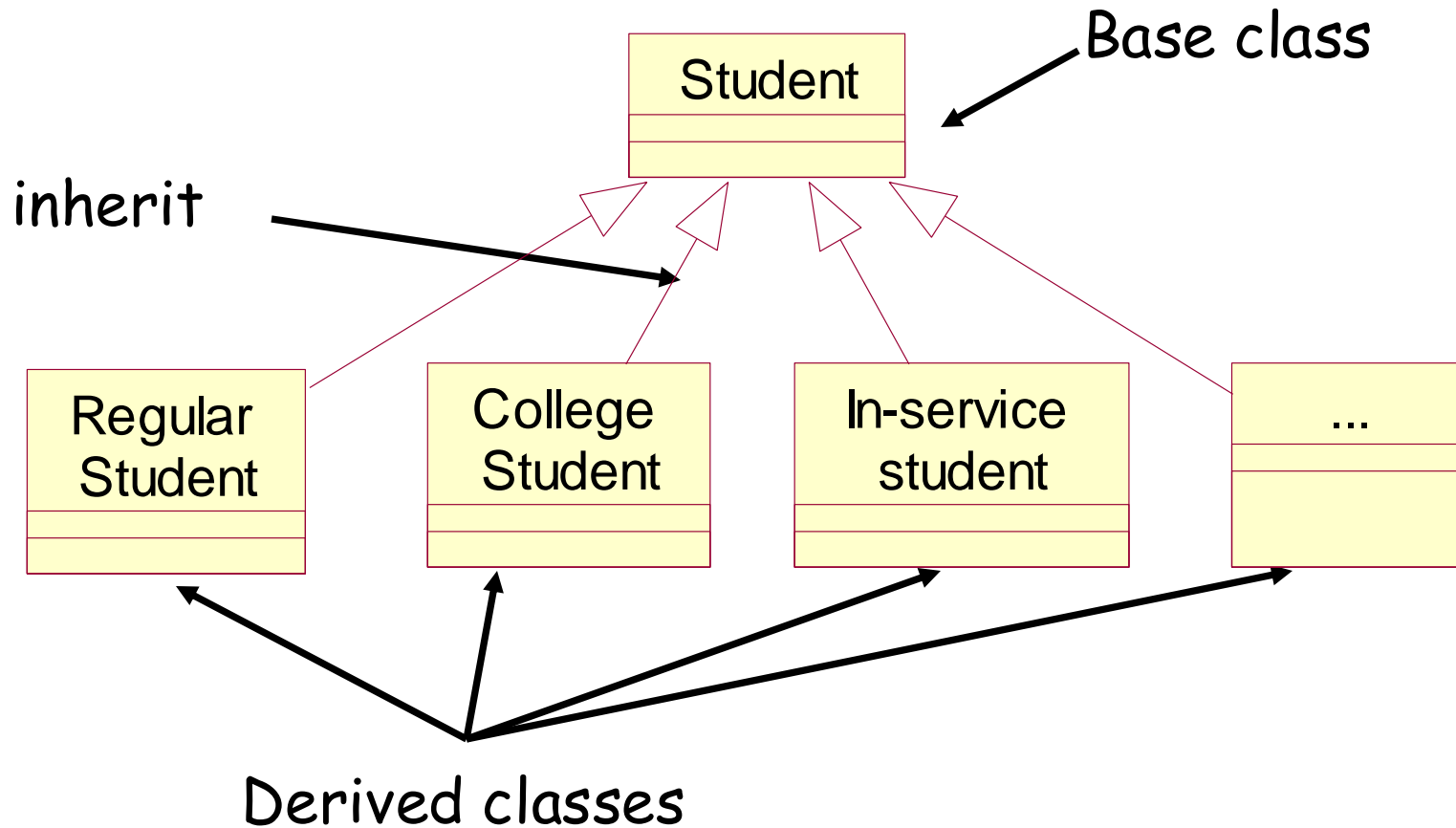
B



Inheritance



An example



Classify the following types of units into groups and subgroups



Barbarian-Axe Swinger



Catapult



Balloon



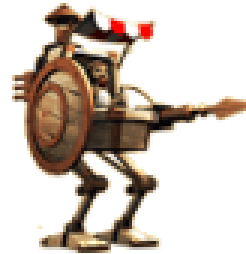
Doctor



Cook



Musketeer



Steam Giant



Swordsman



Ram



Gyrocopter



Slinger



Spearman



Phalanx



Cannon



Archer

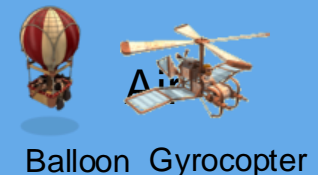
Hierarchical organization of concepts

- ❖ A group of concepts is divided into sub-groups according to some criterion.
- ❖ Example: Types of units are classified according to their means of transportation

GroundUnit



AirUnit



You should use
only one criterion at a time
to classify a group of concepts.



Barbarian-Axe Swinger



Catapult



Gyrocopter



Balloon



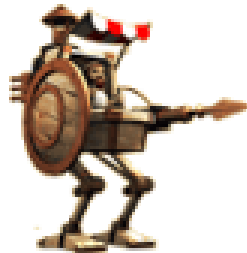
Cook



Musketeer



Swordsman



Steam Giant



Doctor



Ram

Ground Units

Ground units VS Air units

Air Units



Archer



Slinger



Spearman



Phalanx



Cannon

Hierarchical organization of concepts

- ❖ Concepts are recursively classified into subgroups

GroundUnit



Swordsman



Phalanx



Slinger



Musketeer



Archer



Spearman



Barbarian-Axe Swinger



Doctor



Cook



Ram



Steam Giant



Catapult



Cannon

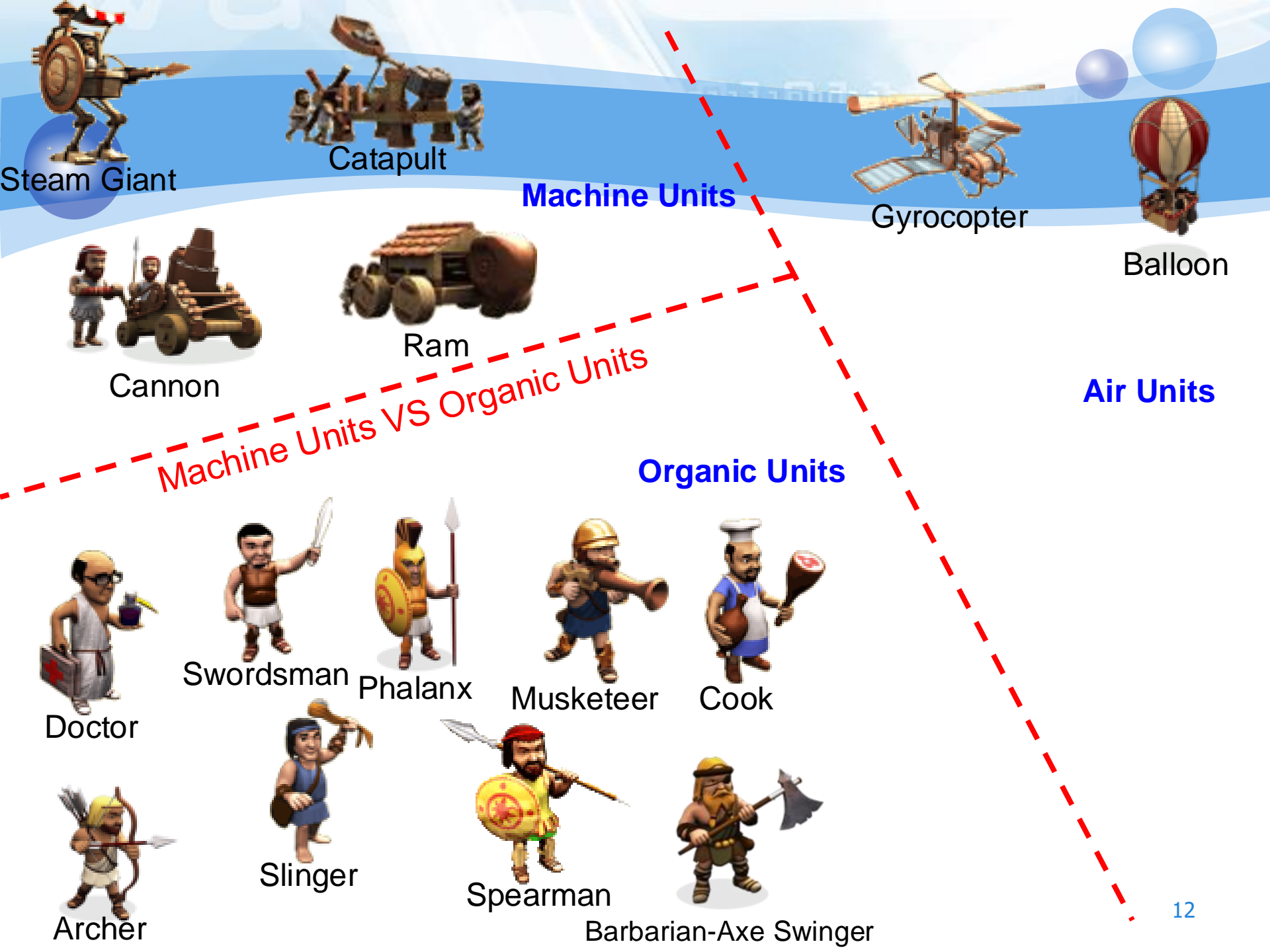
AirUnit

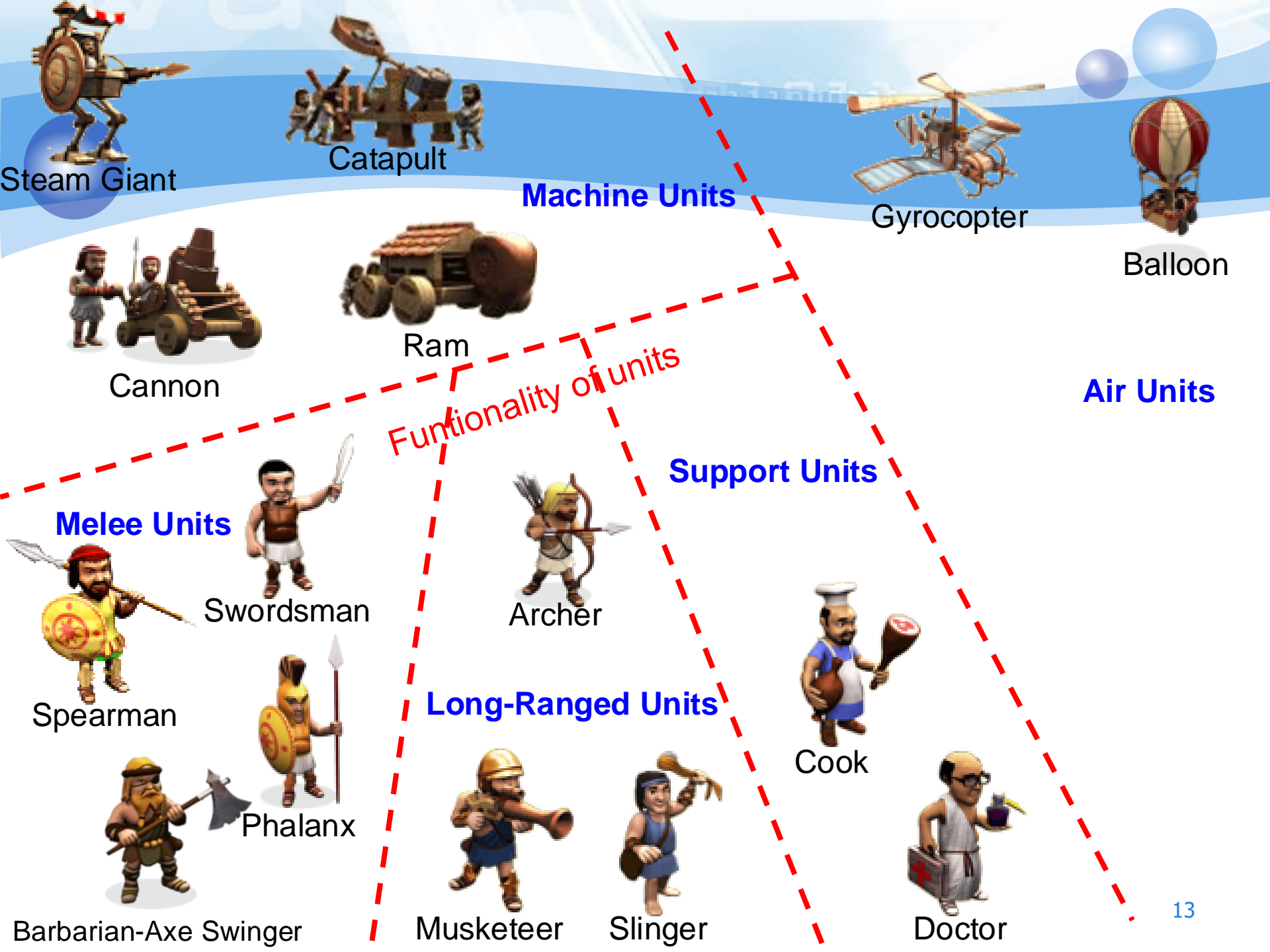


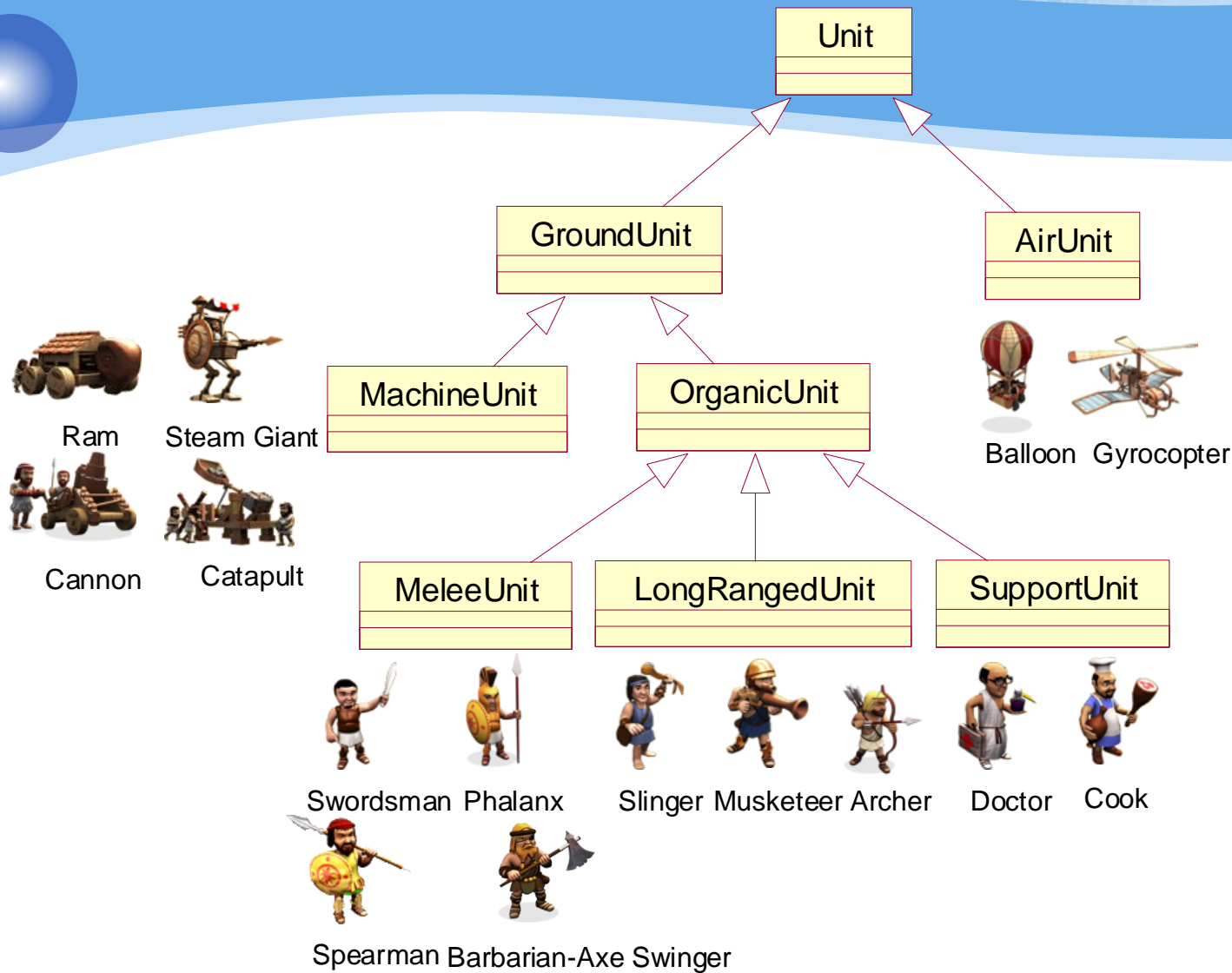
Balloon



Gyrocopter







It is possible to have different ways
to create logical hierarchical organization of concepts



Balloon



Gyrocopter



Cook



Doctor



Ram



Catapult



Cannon



Slinger



Archer



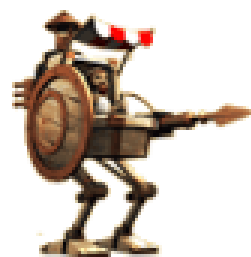
Barbarian-Axe Swinger



Phalanx



Spearman



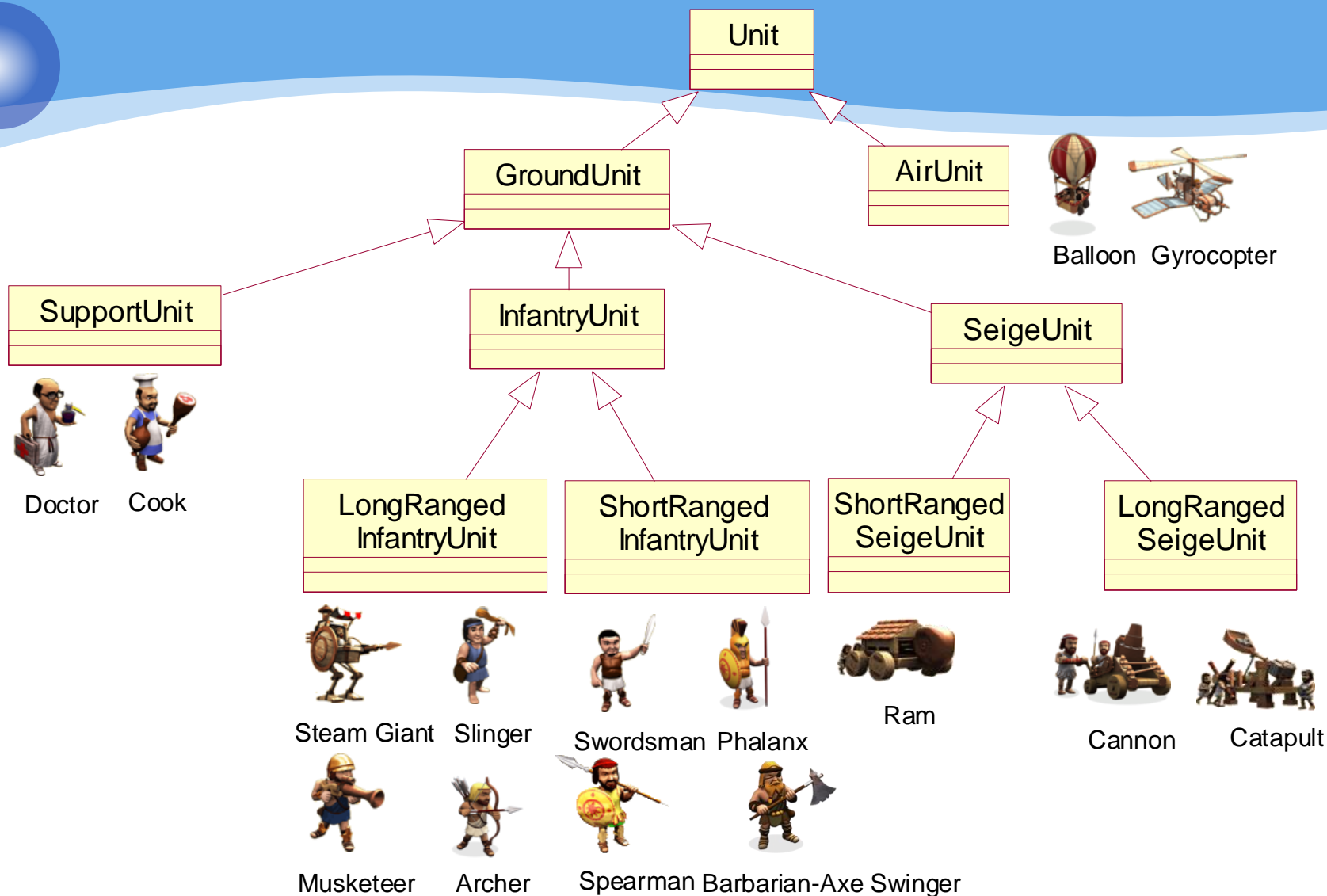
Steam Giant



Musketeer



Swordsman¹⁵



It is possible to have different ways
to create logical hierarchical organization of concepts

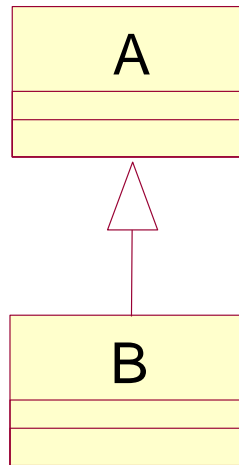
Syntax

❖ In C++, the inheritance is described as:

```
class <Derived Class> : <TypeOfInheritance> <Base class>
{
    public:
        <public attributes/functions>

    private:
        <private attributes/functions>
};
```

Inheritance: notes



- ❖ Attributes/functions in **public** of A will become attributes/functions in B
- ❖ **Private** attributes/functions of A will be part of B but it is only accessible via **public** or **protected** functions of A

protected keyword

- ❖ **protected** attributes/functions of A is accessible from the derived class B but not from outside

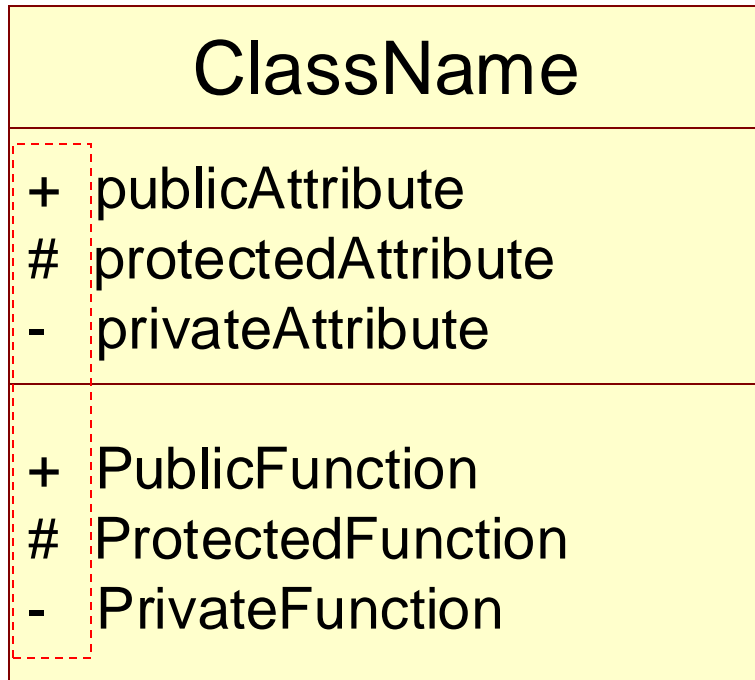
```
class A
{
    public:
        ...
    protected:
        int t;
};
```

```
class B: public A
{
    public:
        void Test()
        {
            cout << t;
        }
        ...
};
```

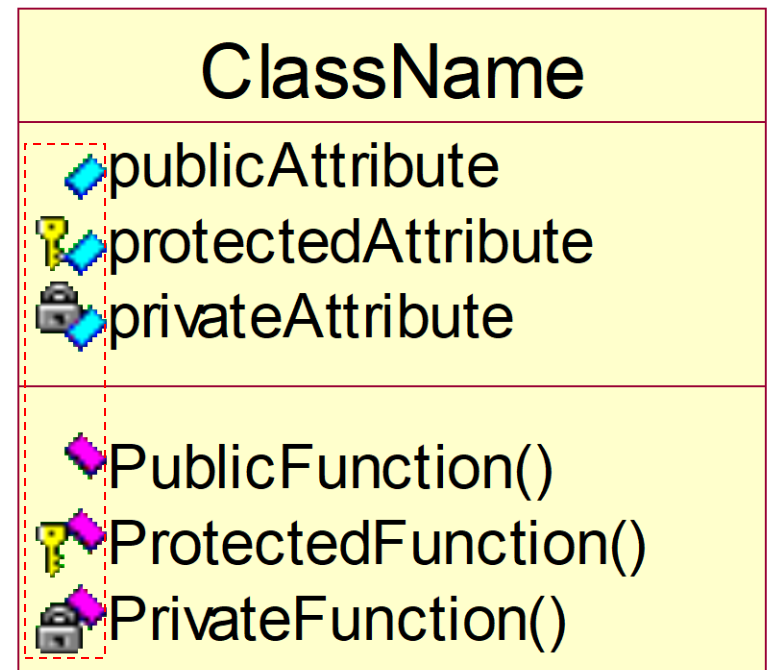
Protected
attribute
of A

UML Notation of Visibility

- ❖ Visibility of an attribute/function can be **public**, **protected**, or **private**



or



Types of inheritance (in C++)

There are 3 types of inheritance in C++:

- ❖ **public** inheritance
- ❖ **protected**
- ❖ **private**

Notes: from now on, if there is no mention of what type of inheritance, it means public inheritance

Types of inheritance (in C++)

- ❖ **public**: **public** and **protected** of the **base class** become **public** and **protected** of the **derived class**.
- ❖ **protected**: **public** and **protected** of the **base class** become **protected** of the **derived class**.
- ❖ **private**: **public** and **protected** of the **base class** become **private** of the **derived class**.

Types of inheritance (in C++)

class B: public A



Public
attributes/functions

Protected
attributes/functions

Private
attributes/functions



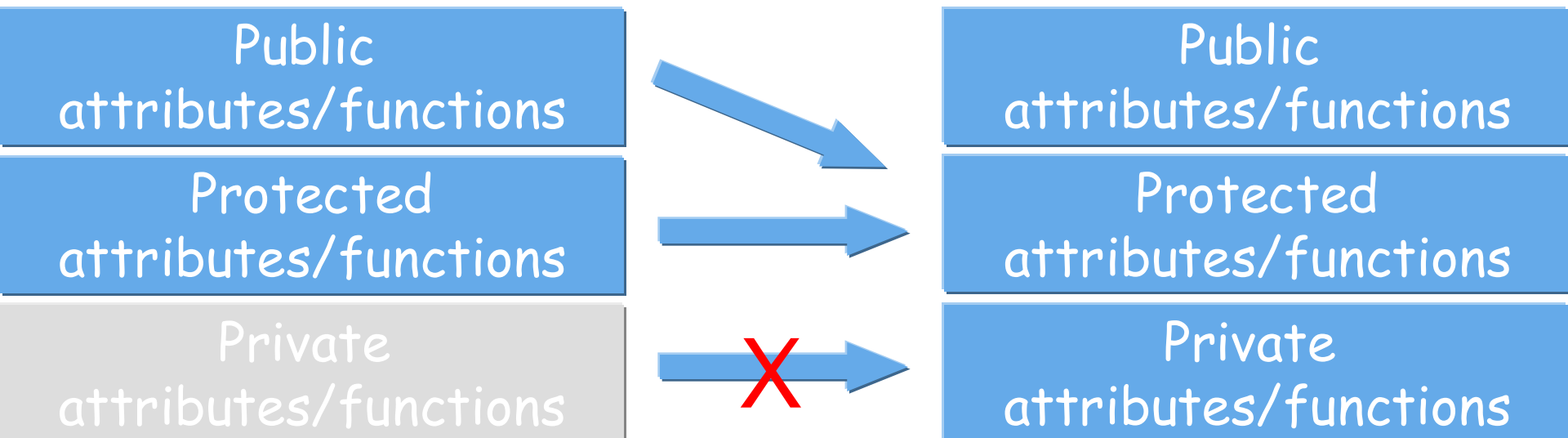
Public
attributes/functions

Protected
attributes/functions

Private
attributes/functions

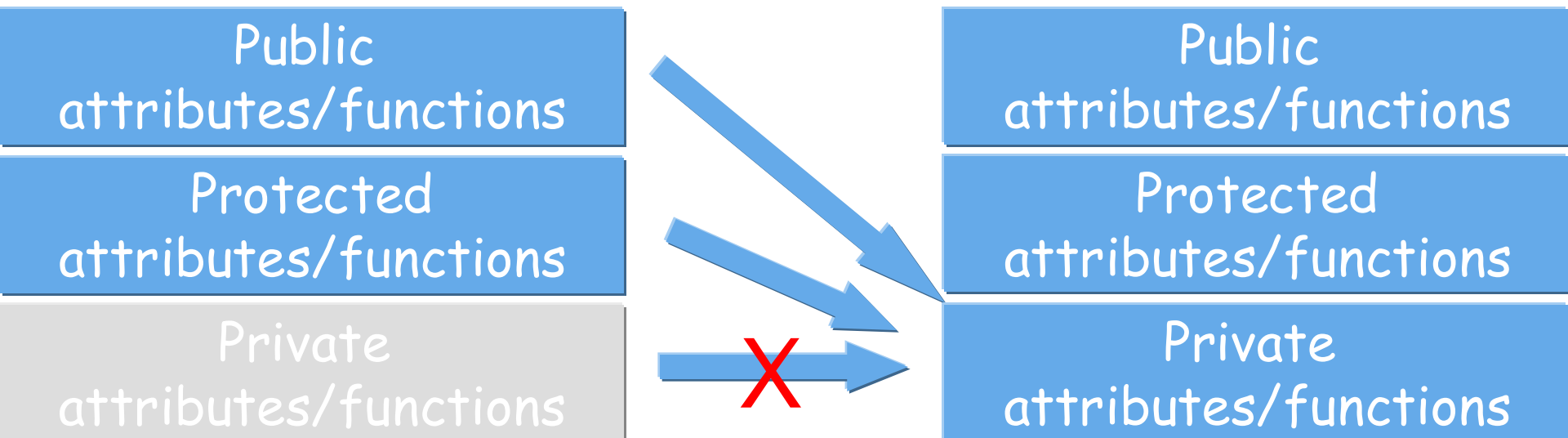
Types of inheritance (in C++)

class B: **protected** A

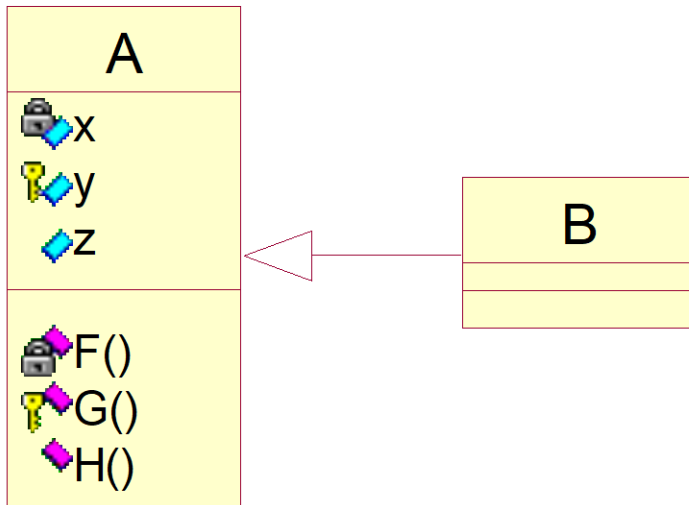


Types of inheritance (in C++)

class B: private A



Types of inheritance (in C++)

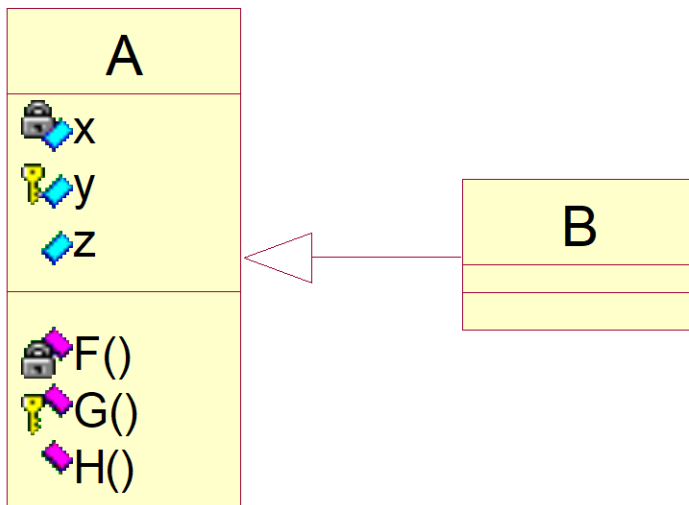


❖ Class B has:

- Attribute y (protected)
- Attribute z (public)
- Function G (protected)
- Function H (public)

Type of inheritance: **public**

Types of inheritance (in C++)

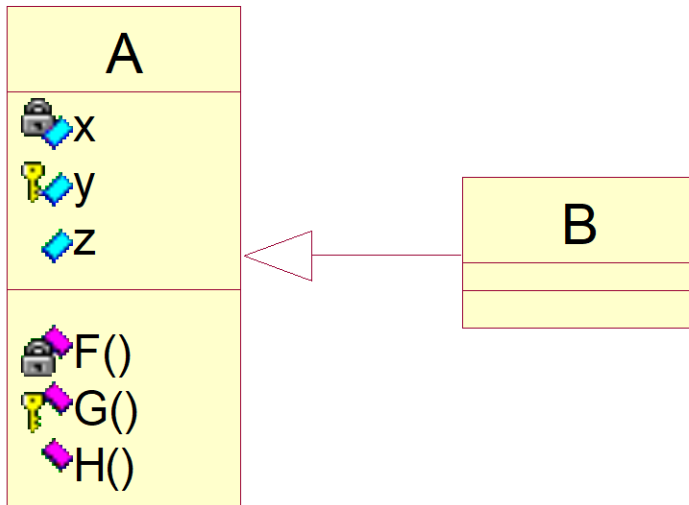


❖ Class B has:

- Attribute y (protected)
- Attribute z (**protected**)
- Function G (protected)
- Function H (**protected**)

Type of inheritance: **protected**

Types of inheritance (in C++)



❖ Class B has:

- Attribute y (**private**)
- Attribute z (**private**)
- Function G (**private**)
- Function H (**private**)

Type of inheritance: **private**

Inheritance: member functions

- ❖ Member functions of the **base class** are inherited in the **derived class**, **except**:
 - Constructors
 - Destructors
 - Assignment operators
- ❖ Notes: private member functions of the base class are inherited but only accessible via other public/protected functions of A

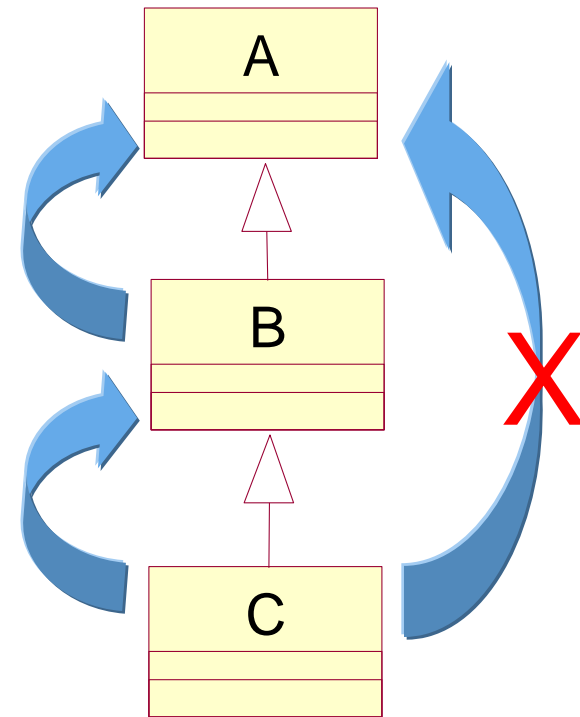
Constructors in inheritance

- ❖ When a new object of a **derived class** is created
 - The **constructor** of the **base class** is invoked first.
 - Then, the **constructor** of the **derived class** is invoked.
 - In the **constructor** of the **derived class**, we can specify which **constructor** of the **base class** is called. Otherwise, the **default constructor** of the **base class** will be invoked.

Constructors in inheritance

Notes:

- ❖ The **constructor** of the **derived class** is able to specify the **constructor** of the **immediate base class** to be called.

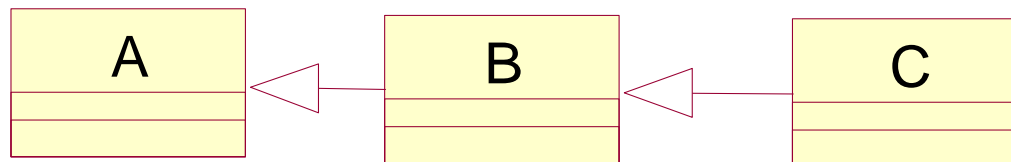
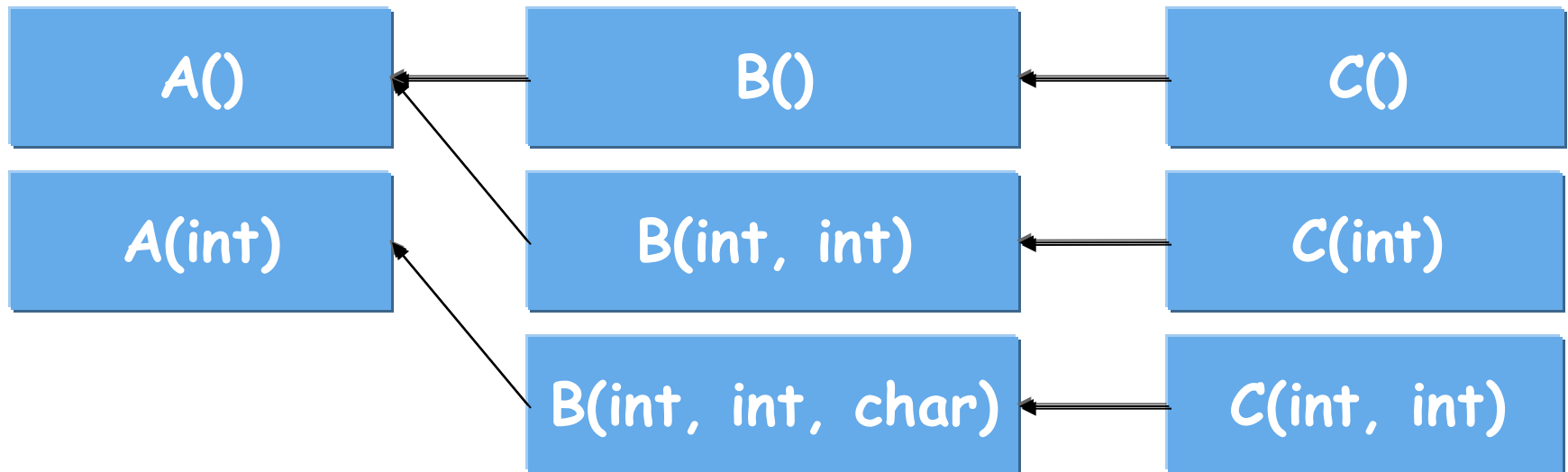


An example

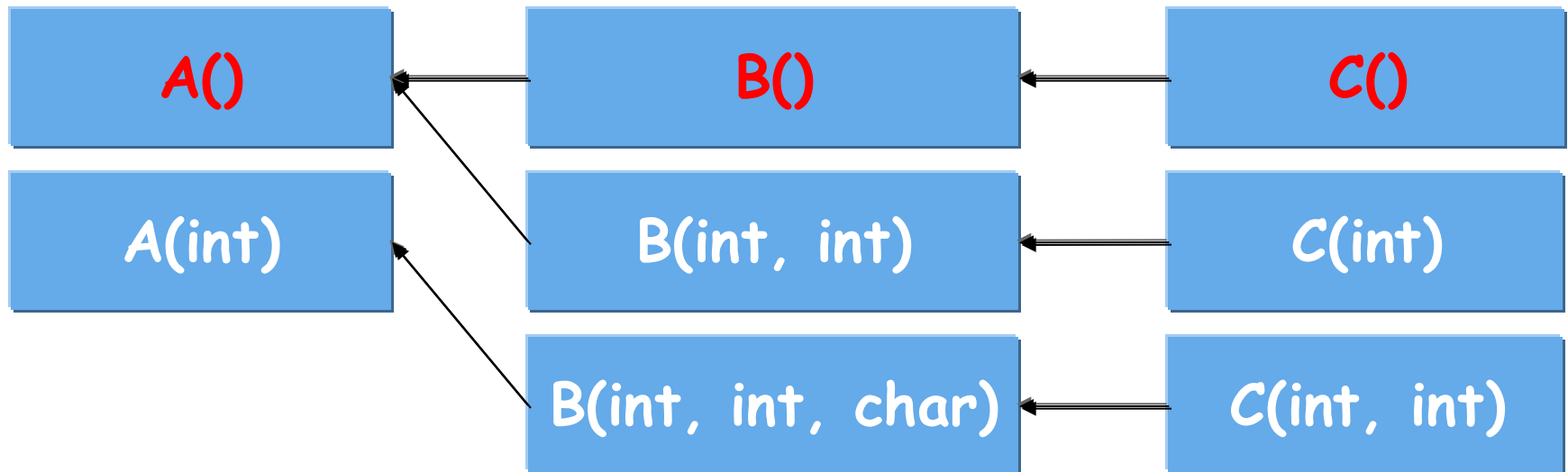
```
class A
{
public:
    A();
    A(int);
};
class B : public A
{
public:
    B(int);
};
```

```
class A
{
public:
    A();
    A(int);
};
class B : public A
{
public:
    B(int t) : A(t)
    { //...
    }
};
```


An example



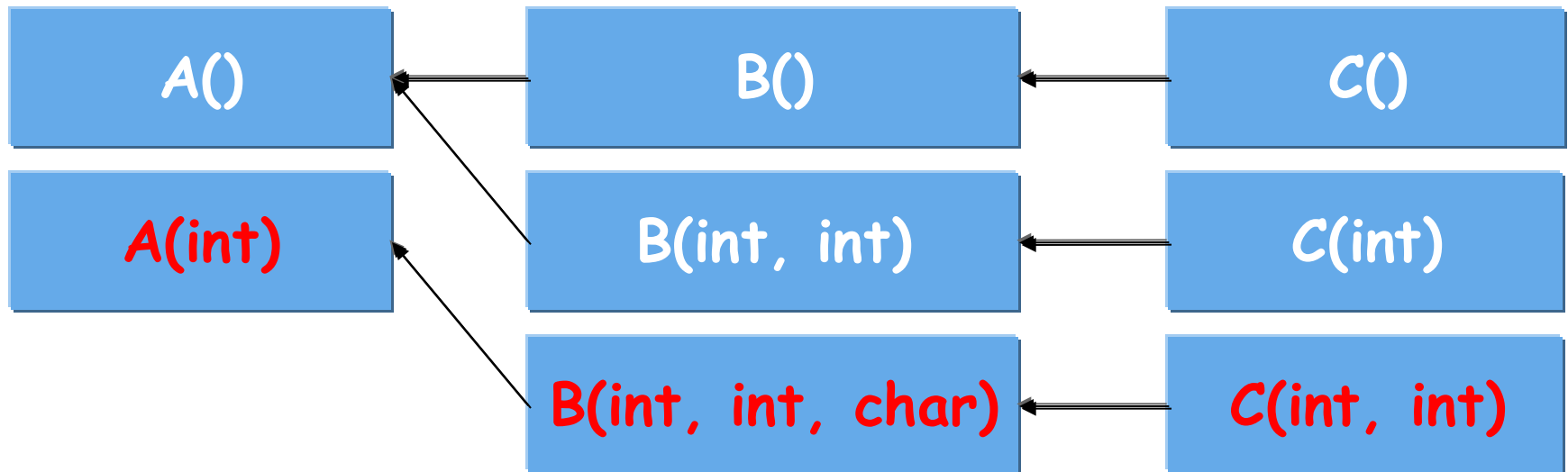
An example



C myObj;

1. The **constructor** of A is invoked: A()
2. The **constructor** of B is invoked: B()
3. The **constructor** of C is invoked: C()

An example

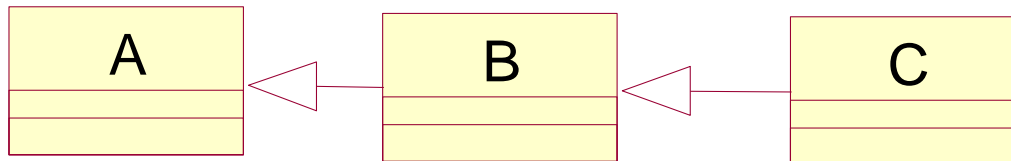


C myObj(5, 7);

1. The **constructor** of A is invoked: A(int)
2. The **constructor** of B is invoked: B(int, int, char)
3. The **constructor** of C is invoked: C(int, int)

Destructor in inheritance

- ❖ When an object of the derived class finishes its lifespan:
 - The **destructor** of the **derived class** is invoked first.
 - Then, the **destructor** of the **base class** is called later.
- ❖ Example:



- when an object of C is destroyed, the following functions will be invoked: $\sim C() \rightarrow \sim B() \rightarrow \sim A()$

"Re-define" member functions

- ❖ Sometimes, we need to "re-define" the member functions of the **base class** in the **derived class**.
 - It can be done by re-defining the functions inside the **derived class**

Notes: this **re-definition** will **hide** other **overloading member functions** of this function from the **base class**.

An example

```
class A {  
public:  
    void test();  
    void test(int);  
    void test(int, int);  
    ...  
};
```

```
class B : public A  
{  
public:  
    void test(int);  
};
```

```
int main()  
{  
    B b;  
    int x, y;  
    ...  
    b.test(x); // OK  
    b.test(x, y); //error  
    ...  
}
```

This function
will hide other
overloading functions
inherited from A

using keyword

```
class A {  
public:  
    void test();  
    void test(int);  
    void test(int, int);  
    ...  
};
```

```
class B : public A  
{  
public:  
    using A::test;  
    void test(int);  
};
```

```
int main()  
{  
    B b;  
    int x, y;  
    ...  
    b.test(x); // OK  
    b.test(x, y); //OK  
    ...  
}
```

B uses overloading
functions inherited
from A

Assignment operator

- ❖ It is not inherited from the base class
- ❖ To implement the assignment operator for the derived class:
 - Calling the assignment operator of the base class to assign data members of the base class part in the two objects first.
 - Then, implement the assignment for data member of the derived class part.

An example

```
B& B::operator=(const B& src)
{
    if (this == &src)
        return *this;
    A::operator=(src);
    delete [] ptr;
    iSize = src.iSize;
    ptr = new int [iSize];
    for (int i=0; i<iSize; ++i)
        ptr[i] = src.ptr[i];
    return *this;
}
```

