

# Module 25: Programming in C++

## Inheritance: Part 5

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ernet.in*

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

# Module Objectives



- Explore restricted forms of inheritance (`private` and `protected`) in C++ and their semantic implications

# Module Outline

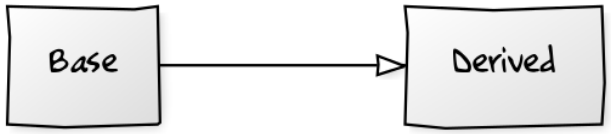


- `ISA` Relationship
- Inheritance in C++
  - Semantics
  - Data Members and Object Layout
  - Member Functions
    - Overriding
    - Overloading
  - `protected` Access
  - Constructor & Destructor
  - Object Lifetime
- Example – Phone Hierarchy
- Inheritance in C++ (`private`)
  - Implemented-As Semantics

# Inheritance in C++: Semantics



- Derived **ISA** Base




```
class Base;                    // Base Class = Base
class Derived: public Base; // Derived Class = Derived
```

- Use keyword `public` after class name to denote inheritance
- Name of the Base class follow the keyword

# Inheritance Exercise: What is the output?



```
class B {
public:
    B() { cout << "B "; }
    ~B() { cout << "~B "; } };

class C {
public:
    C() { cout << "C "; }
    ~C() { cout << "~C "; } };

class D : public B {
    C data_;
public:
    D() { cout << "D " << endl; }
    ~D() { cout << "~D "; }
};

int main() {
    D d;

    return 0;
}
```

# Inheritance Exercise: What is the output?



```
class B {
public:
    B() { cout << "B "; }
    ~B() { cout << "~B "; } };

class C {
public:
    C() { cout << "C "; }
    ~C() { cout << "~C "; } };

class D : public B {
    C data_;
public:
    D() { cout << "D " << endl; }
    ~D() { cout << "~D "; }
};

int main() {
    D d;

    return 0;
}
```

**Output:**

```
B C D
~D ~C ~B
```

# private Inheritance



- private Inheritance
  - Definition

    ```
    class Base;
    class Derived: private Base;
    ```

  - Use keyword private after class name
  - Name of the Base class follow the keyword
  - private inheritance does not mean generalization / specialization

**• Private inheritance means nothing during software design, only during software implementation**

**• Private inheritance means is-implemented-in-terms of. It's usually inferior to composition, but it makes sense when a derived class needs access to protected base class members or needs to redefine inherited virtual functions**

– Scott Meyers in Item 32, Effective C++ (3rd. Edition)

# private Inheritance



**public Inheritance**

```
class Person {...};

class Student:
    public Person {...};

// anyone can eat
void eat(const Person& p);

// only students study
void study(const Student& s);

Person p;  // p is a Person

Student s; // s is a Student

eat(p);    // fine, p is a Person

eat(s);    // fine, s is a Student,
           // and a Student is-a Person

study(s);  // fine

study(p);  // error! p isn't a Student
```

Compilers converts a derived class object (Student) into a base class object (Person) if the inheritance relationship is public

**private Inheritance**

```
class Person { ... };

class Student: // inheritance is now private
    private Person { ... };

// anyone can eat
void eat(const Person& p);

// only students study
void study(const Student& s);

Person p;  // p is a Person

Student s; // s is a Student

eat(p);    // fine, p is a Person

eat(s);    // error! a Student isn't a Person
```

Compilers will not convert a derived class object (Student) into a base class object (Person) if the inheritance relationship is private

# protected Inheritance



- protected Inheritance
  - Definition

    ```
    class Base;
    class Derived: protected Base;
    ```

  - Use keyword protected after class name
  - Name of the Base class follow the keyword
  - protected inheritance does not mean generalization / specialization

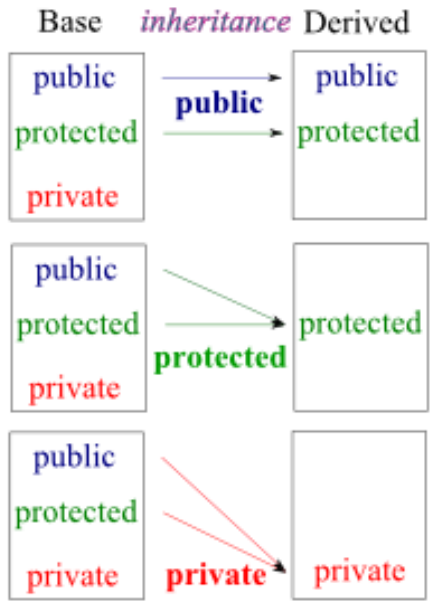
**• Private inheritance means something entirely different (from public inheritance), and protected inheritance is something whose meaning eludes me to this day**

– Scott Meyers in Item 32, Effective C++ (3rd. Edition)

# Visibility across Access and Inheritance

- Visibility Matrix

| Visibility \ Inheritance | public | protected | private |
|---|---|---|---|
| public | public | protected | private |
| protected | protected | protected | private |
| private | private | private | private |



# Inheritance Exercise: What is the output?



```
class B {
protected:
    B() { cout << "B "; }
    ~B() { cout << "~B "; }
};
class C : public B {
protected:
    C() { cout << "C "; }
    ~C() { cout << "~C "; }
};
class D : private C {
    C data_;
public:
    D() { cout << "D " << endl; }
    ~D() { cout << "~D "; }
};

int main() {
    D d;

    return 0;
}
```

# Inheritance Exercise: What is the output?



```
class B {
protected:
    B() { cout << "B "; }
    ~B() { cout << "~B "; }
};
class C : public B {
protected:
    C() { cout << "C "; }
    ~C() { cout << "~C "; }
};
class D : private C {
    C data_;
public:
    D() { cout << "D " << endl; }
    ~D() { cout << "~D "; }
};

int main() {
    D d;

    return 0;
}
```

**Output:**

```
B C B C D
~D ~C ~B ~C ~B
```

# Inheritance Exercise: Access Rights



**Inaccessible Members**

```
class A {
private: int x;
protected: int y;
public: int z;
};
class B : public A {
private: int u;
protected: int v;
public: int w; void f() { x; }
};
class C: protected A {
private: int u;
protected: int v;
public: int w; void f() { x; }
};
class D: private A {
private: int u;
protected: int v;
public: int w; void f() { x; }
};
class E : public B {
public: void f() { x; u; }
};
class F : public C {
public: void f() { x; u; }
};
class G : public D {
public: void f() { x; y; z; u; }
};
```

**Accessible Members**

```
void f(A& a,
       B& b, C& c, D& d,
       E& e, F& f, G& g) {
    a.z;

    b.z;
    b.w;

    c.w;

    d.w;

    e.z;
    e.w;

    f.w;

    g.w;
}
```

# Car HAS–A Engine: Composition OR `private` Inheritance?



**Simple Composition**

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

class Engine {
public:
    Engine(int numCylinders) { }
    // Starts this Engine
    void start() { }
};

class Car {
public:
    // Initializes this Car with 8 cylinders
    Car() : e_(8) { }

    // Start this Car by starting its Engine
    void start() { e_.start(); }
private:
    Engine e_; // Car has-a Engine
};

int main() {
    Car c;

    c.start();

    return 0;
}
```

`private` **Inheritance**

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

class Engine {
public:
    Engine(int numCylinders) { }
    // Starts this Engine
    void start() { }
};

class Car : private Engine { // Car has-a Engine
public:
    // Initializes this Car with 8 cylinders
    Car() : Engine(8) { }

    // Start this Car by starting its Engine
    using Engine::start;

};

int main() {
    Car c;

    c.start();

    return 0;
}
```

# private Inheritance



- Use composition when you can, private inheritance when you have to

**• Private inheritance means nothing during software design, only during software implementation**

**• Private inheritance means is-implemented-in-terms of. It's usually inferior to composition, but it makes sense when a derived class needs access to protected base class members or needs to redefine inherited virtual functions**

– Scott Meyers in Item 32, Effective C++ (3rd. Edition)

# Module Summary



- Introduced restricted forms of inheritance and `protected` specifier
- Discussed how `private` inheritance is used for *Implemented-As* Semantics

# Instructor and TAs



| Name | Mail | Mobile |
|---|---|---|
| Partha Pratim Das, *Instructor* | ppd@cse.iitkgp.ernet.in | 9830030880 |
| Tanwi Mallick, *TA* | tanwimallick@gmail.com | 9674277774 |
| Srijoni Majumdar, *TA* | majumdarsrijoni@gmail.com | 9674474267 |
| Himadri B G S Bhuyan, *TA* | himadribhuyan@gmail.com | 9438911655 |