C++

程式語言（二）

Introduction to Programming (II)

Inheritance

Joseph Chuang-Chieh Lin

Dept. CSE, NTOU

Platform/IDE

- Dev-C++

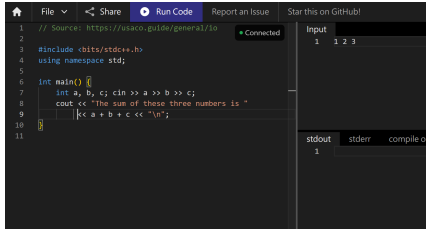
https://www.pngegg.com/en/search?q=Dev-C

- Codeblocks

https://icons8.com/icons/set/code-blocks

- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)



Textbooks (We focusing on C++11)

- ***Learn C++ Programming by Refactoring* (由重構學習 C++ 程式設計). Pang-Feng Liu (劉邦鋒). NTU Press. 2023.**
- ***C++ Primer. 5th Edition.* Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. 2019.**
- *Effective C++*. Scott Meyers. O'Reilly. 2016.
- *Thinking in C++. Vol. 1: Introducing to Standard C++*. 2nd Edition. Bruce Eckel. Prentice Hall PTR. 2000.

Useful Resources

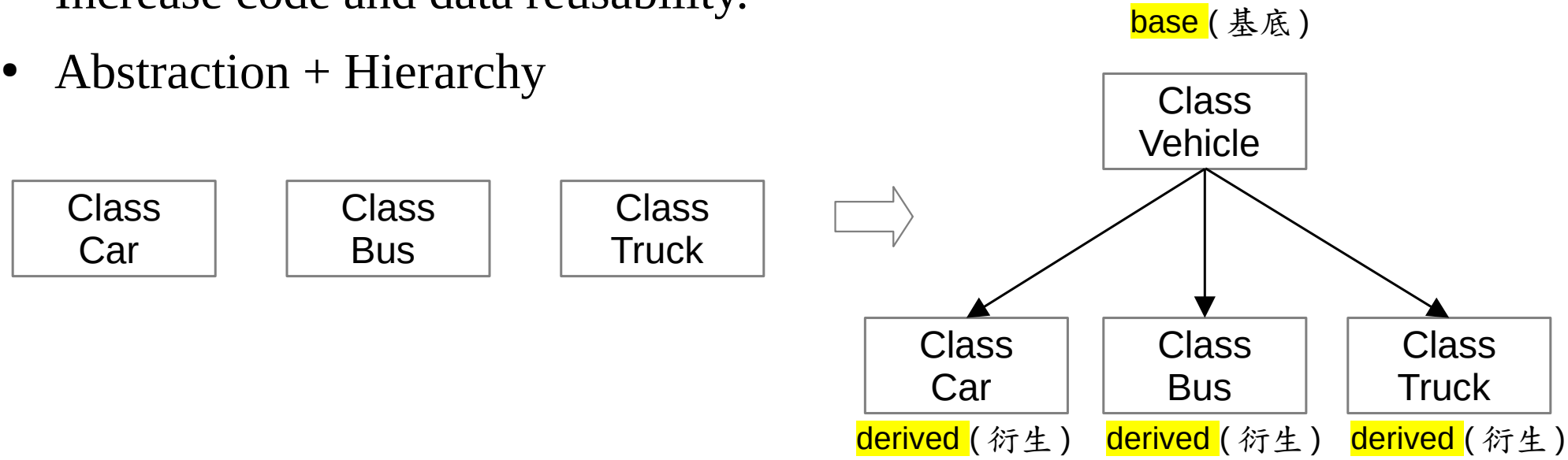
- Tutorialspoint
  - https://www.tutorialspoint.com/cplusplus/index.htm
  - Online C++ Compiler
- Programiz
  - https://www.programiz.com/cpp-programming
- LEARN C++
  - https://www.learncpp.com/
- MIT OpenCourseWare - Introduction to C++
  - https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/pages/lecture-notes/
- Learning C++ Programming
  - https://www.programiz.com/cpp-programming
- GeeksforGeeks
  - https://www.geeksforgeeks.org/c-plus-plus/

Inheritance

Inheritance

https://www.geeksforgeeks.org/inheritance-in-c/?ref=lbp

- Get rid of duplication of the same codes.
- Decrease the chance of error.
- Increase code and data reusability.
- Abstraction + Hierarchy



An Easy Illustrating Example

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};
```

access mode

```
class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A
// 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

Modes of Inheritance

Just like going through a mask...

Example: https://onlinegdb.com/Z7tf4BU0x

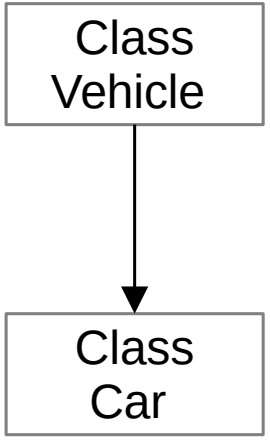
- `Public`
  - public member of the base class => **public** in the derived class.
  - protected members of the base class => **protected** in derived class.
  - **private** members of the base class => not accessible.
- `Protected`
  - public member of the base class => **protected** in the derived class.
  - protected members of the base class => **protected** in derived class.
  - **private** members of the base class => not accessible.
- `Private`
  - public member of the base class => **private** in the derived class.
  - protected members of the base class => **private** in derived class.
  - **private** members of the base class => **NOT** accessible.

Single Inheritance

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

class Vehicle {
public:
    Vehicle() {
      cout << "This is a Vehicle. " << endl;
    }
};

class Car : public Vehicle {
// nothing to do here so far...
};
```




```
int main()
{
    // invoke the constructors
    Car obj;
    return 0;
}
```

Output:

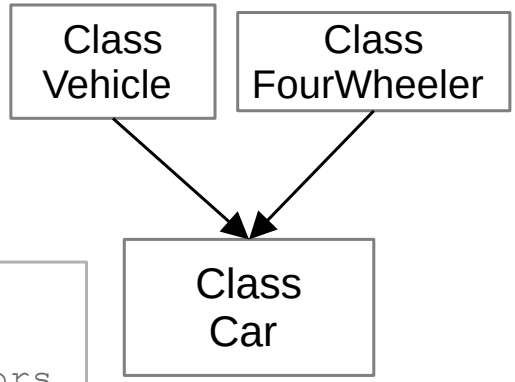
```
This is a Vehicle.
```

Multiple Inheritance

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

class Vehicle {
public:
    Vehicle() {
      cout << "This is a Vehicle."
           << endl;
    }
};
class FourWheeler {
public:
    FourWheeler() {
         cout << "This is a 4 wheeler
                  Vehicle. " << endl;
    }
};
```

```
class Car : public Vehicle, public FourWheeler {
    // nothing to do here so far...
};
```




```
int main()
{
    // invoke the constructors
    Car obj;
    return 0;
}
```

Output:

```
This is a Vehicle.
This is a 4 wheeler Vehicle.
```

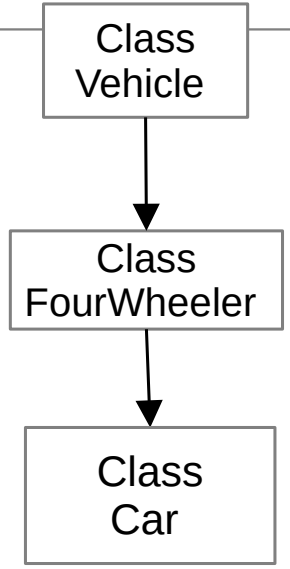
Multilevel Inheritance

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

class Vehicle {
public:
    Vehicle() {
      cout << "This is a Vehicle."
           << endl;
    }
};
class FourWheeler: public Vehicle {
public:
    FourWheeler() {
         cout << "A 4 wheeler Vehicle."
              << endl;
    }
};
```

```
class Car: public FourWheeler {
public:
     Car() {
       cout << "A Car has 4 Wheels." << endl;
     }
};
```

```
int main()
{
    // invoke the constructors
    Car obj;
    return 0;
}
```



Output:

```
This is a Vehicle.
A 4 wheeler Vehicle.
A Car has 4 Wheels.
```

More Details in Examples

- https://www.programiz.com/cpp-programming/public-protected-private-inheritance

Exercise

```
class Shape {
public:
    string type;
protected:
    double parameter;
};
```

```
class Circle : protected Shape {
private:
    double area = 0.0;
public:
    void compute_area()     {
/* please implement this member function */
    }
    void setRadius() {
/* please implement this member function */
    }
    double getArea() {
/* please implement this member function */
    }
};
```

```
int main()
{
    Circle obj;
    obj.setRadius();
    obj.compute_area();
    cout << "Area: " << obj.getArea();
    return 0;
}
```

Sample Input & Output:

```
3.2
Area: 32.1699
```

Exercise

```
class A {
public:
     int x = 0;
     int get_pvt() { return z; }
protected:
     int y = 1;
private:
     int z = 2;
};

class B : public A {
    // x is public
    // y is protected
    // z is not accessible from B
};
```

```
class C : protected A {
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A {
// 'private' is default for classes
    // x is private
    // y is private
    // z is not accessible from D
};
```

Please modify the code here by "adding appropriate member functions" in the the classes B, C, and D.

```
int main (){
     B obj1;
     C obj2;
     D obj3;
     // cout << obj1.x << obj2.y << obj3.y;
     // try to print these three values
     // by adding appropriate member
     // functions
}
```

Constructors and Destructors w.r.t. Inheritance

About the order of constructors

Based on the material at https://www.geeksforgeeks.org/order-constructor-destructor-call-c/

- To create an object of a class, a corresponding constructor of that class must be invoked **automatically** to **initialize** the members of the class.

- The data members and member functions of **base class** comes automatically in derived class based on the *access specifier* but the definition of these members exists in base class only.

- The constructor of base class is called **first** to initialize all the inherited members.

Example

https://www.geeksforgeeks.org/order-constructor-destructor-call-c/

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

// base class
class Parent {
public:

    // base class constructor
    Parent() {
        cout << "Inside base class" << endl;
    }
};
```

```
// derived class
class Child : public Parent {
public:

    //sub class constructor
    Child() {
        cout << "Inside sub class" << endl;
    }
};
```

```
// main function
int main() {

    // creating object of sub class
    Child obj;

    return 0;
}
```

```
Inside base class
Inside sub class
```

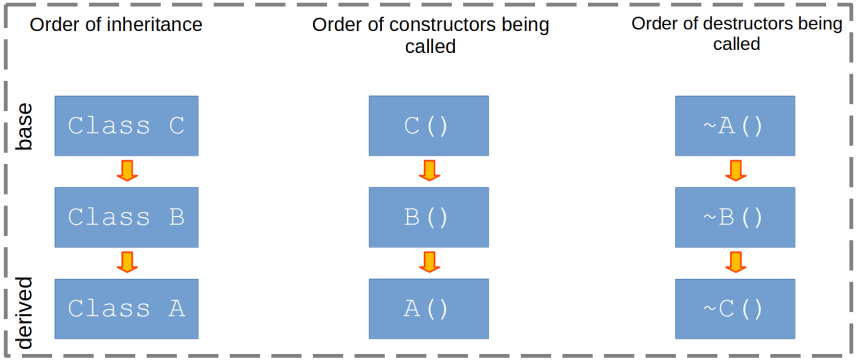
Another Example (Multiple Inheritance)

Refer to https://www.geeksforgeeks.org/order-constructor-destructor-call-c/

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

// first base class
class Parent1 {
 public:

    Parent1() {
        cout << "Constructor: Inside 1st base class" << endl;
    }
    ~Parent1() { cout << "Destructor for Parent1" << endl; };
};
```




```
class Parent2 {
public:
    // second base class's Constructor
    Parent2()  {
        cout << "Constructor: Inside 2nd base class" << endl;
    }
    ~Parent2() { cout << "Destructor for Parent2" << endl; };

};
```

```
class Child : public Parent1, public Parent2 {
 public:

    // child class's Constructor
    Child() {
        cout << "Constructor: Inside child class"
             << endl;
    }
    ~Child() {
            cout << "Destructor for Child"
                 << endl;
    };
};
```

https://onlinegdb.com/OlcKDGw51

Another Example (Multiple Inheritance)

+ virtual destructors

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

// first base class
class Parent1 {
 public:

    Parent1() {
        cout << "Constructor: Inside 1st base class" << endl;
    }
    virtual ~Parent1() { cout << "Destructor for Parent1" <<
endl; };
};
```

```
class Parent2 {
public:
    // second base class's Constructor
    Parent2()  {
        cout << "Constructor: Inside 2nd base class" << endl;
    }
    virtual ~Parent2() { cout << "Destructor for Parent2" <<
endl; };

};
```

```
class Child : public Parent1, public Parent2 {
 public:

    // child class's Constructor
    Child() {
        cout << "Constructor: Inside child class"
             << endl;
    }
    ~Child() {
            cout << "Destructor for Child"
                 << endl;
    };
};
```

```
int main() {
    Parent1 *obj = new Child; // obj1;
    delete obj;
    return 0;
}
```

https://onlinegdb.com/1UxmKEomO

Example: Book Sales

https://onlinegdb.com/RsQlo9BZw https://onlinegdb.com/2A_6kkIoo

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

// Below we define the base class
class Quote {
public:
    Quote() = default;
    Quote(const std::string &book, double sales_price):
        bookNo(book), price(sales_price) { }
    std::string isbn() const { return bookNo; }
    //returns the total sales price for the specified number of items
    virtual double net_price(std::size_t n) const
        { return n * price; }
    virtual ~Quote() = default; // dynamic binding for the destructor
private:
    std::string bookNo;
protected:
    double price = 0.0; // this is protected because we want it to be used by
                        // the derived classes
};

//Below we define a class derived from the base class Quote
class Bulk_quote : public Quote {
public:
    Bulk_quote() = default;
    Bulk_quote(const std::string &book, double p, std::size_t qty, double disc):
        Quote(book, p), min_qty(qty), discount(disc) { }
    double net_price(std::size_t) const override;
private:
    std::size_t min_qty = 0; // minimum purchase for the discount to apply
    double discount = 0.0; // the discount to apply
};
```

An object of `Bulk_quote`

inherited from `Quote`

defined by `Bulk_quote`

bookNo price
min_qty discount

```
double Bulk_quote::net_price(size_t cnt)  {
    if (cnt >= min_qty)
        return cnt * (1-discount) * price;
    else
        return cnt * price;
}
```

Why and when do we need the keyword override? [link]

The Diamond Problem

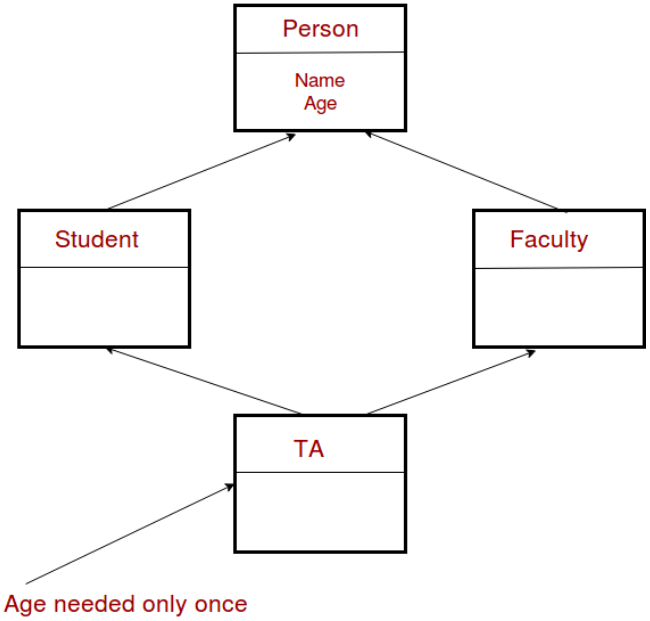
https://www.geeksforgeeks.org/multiple-inheritance-in-c/

```
class Person {
   // Data members of person
public:
     Person(int x)  {
          cout << "Person::Person(int ) called" << endl;

     }
};
```

```
class Faculty : public Person {
   // data members of Faculty
public:
    Faculty(int x): Person(x)   {
       cout<<"Faculty::Faculty(int ) called"<< endl;
    }
};
```

```
class Student : public Person {
   // data members of Student
public:
    Student(int x): Person(x) {
        cout<<"Student::Student(int ) called"<< endl;
    }
};
```




```
class TA : public Faculty, public Student  {
public:
    TA(int x): Student(x), Faculty(x)   {
        cout<<"TA::TA(int ) called"<< endl;
    }
};
```

The Diamond Problem

https://www.geeksforgeeks.org/multiple-inheritance-in-c/

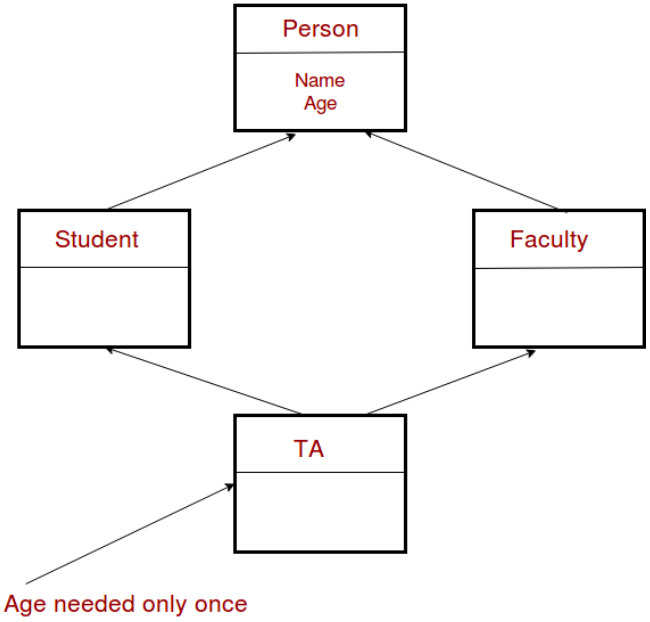
```
Person::Person(int ) called
Faculty::Faculty(int ) called
Person::Person(int ) called
Student::Student(int ) called
TA::TA(int ) called
```

```
class Person {
   // Data members of person
public:
     Person(int x)  {
          cout << "Person::Person(int ) called" << endl;

     }
};
```

```
class Faculty : public Person {
   // data members of Faculty
public:
    Faculty(int x): Person(x)   {
       cout<<"Faculty::Faculty(int ) called"<< endl;
    }
};
```

```
class Student : public Person {
   // data members of Student
public:
    Student(int x): Person(x) {
        cout<<"Student::Student(int ) called"<< endl;
    }
};
```




```
class TA : public Faculty, public Student  {
public:
    TA(int x): Student(x), Faculty(x)   {
        cout<<"TA::TA(int ) called"<< endl;
    }
};
```

What’s the issue?

- The constructor of `Person` is called twice.
- The destructor of `Person` is called twice, too!

The Diamond Problem (solution)

https://www.geeksforgeeks.org/multiple-inheritance-in-c/

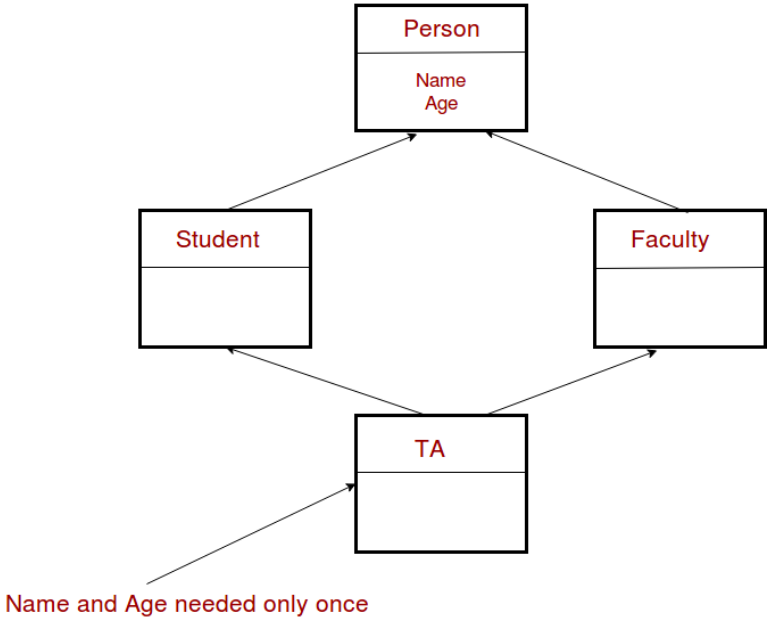
https://onlinegdb.com/-sRGnq3k9

```
class Person {
   // Data members of person
public:
     Person() = default;
     Person(int x)  {
          cout << "Person::Person(int ) called" << endl;

     }
};
```

```
class Faculty : virtual public Person {
   // data members of Faculty
public:
     Faculty(int x): Person(x)   {
       cout<<"Faculty::Faculty(int ) called"<< endl;
     }
};
```

```
class Student : virtual public Person {
   // data members of Student
public:
     Student(int x): Person(x) {
        cout<<"Student::Student(int ) called"<< endl;
     }
};
```




```
class TA : public Faculty, public Student  {
public:
    TA(int x): Student(x), Faculty(x)   {
        cout<<"TA::TA(int ) called"<< endl;
    }
};
```

The ‘virtual’ keyword

- `Faculty` and `Student` will be made as virtual base classes by using the keyword `virtual`.

The Diamond Problem (solution+virtual destructors)

- https://onlinegdb.com/u7F3GU0AD

  Student::Student(int) called => the constructor with parameter is called.

  Person::Person() called => the constructor without parameter is called.

- What if the base class explicitly define a parameterized constructor?

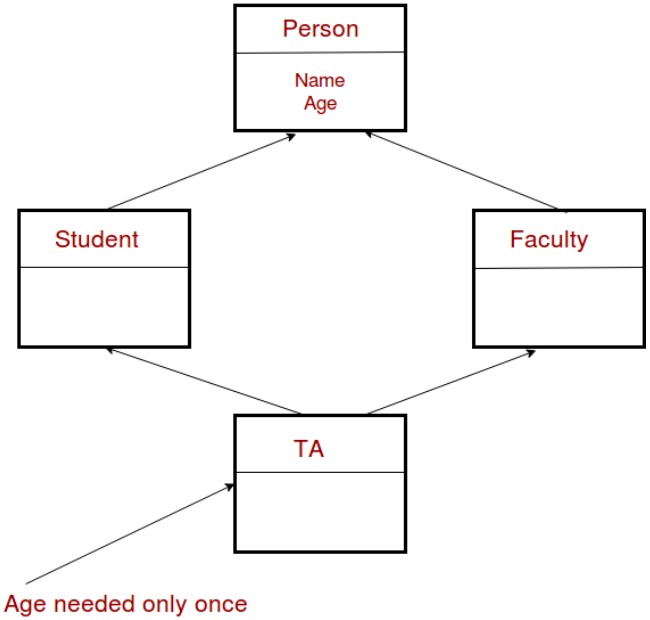
The Diamond Problem (solution)

https://www.geeksforgeeks.org/multiple-inheritance-in-c/

```
class Person {
   // Data members of person
public:
     Person(){ cout << "Person::Person() called" << endl; }
     Person(int x)  {
          cout << "Person::Person(int ) called" << endl;
     }
};
```

```
class Faculty : virtual public Person {
   // data members of Faculty
public:
     Faculty(int x): Person(x)   {
       cout<<"Faculty::Faculty(int ) called"<< endl;
     }
};
```

```
class Student : virtual public Person {
   // data members of Student
public:
     Student(int x): Person(x) {
        cout<<"Student::Student(int ) called"<< endl;
     }
};
```




```
class TA : public Faculty, public Student  {
public:
    TA(int x): Student(x), Faculty(x), Person(x){
        cout<<"TA::TA(int ) called"<< endl;
    }
};
```

Class Exercise (1)

Predict the output of following program (fix the error if necessary).

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

class A {
  int x;
public:
  void setX(int i) {x = i;}
  void print() { cout << x; }
};

class B: public A {
public:
  B()  { setX(10); }
};

class C:  public A {
public:
  C()  { setX(20); }
};

class D: public B, public C { };

int main() {
    D d;
    d.print();
    return 0;
}
```

Class Exercise (2)

Predict the output of following program (fix the error if necessary).

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

class A {
  int x;
public:
  A() = default;
  A(int i) { x = i; }
  void print() { cout << x; }
};

class B: virtual public A {
public:
  B():A(10) {  }
};

class C: virtual public A {
public:
  C():A(20) {  }
};

class D: public B, public C { };

int main() {
    D d;
    d.print();
    return 0;
}
```

Discussions & Questions