物件導向程式設計

Other Materials

Joseph Chuang-Chieh Lin
Dept. CSIE, Tamkang University

Platform

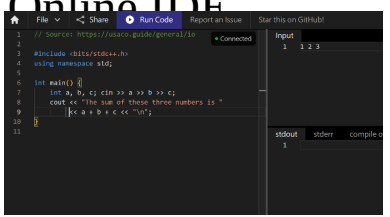
- Dev-C++

  Click here to download.

  Note: Please use this version otherwise you can't compile your programs/projects in Win10.

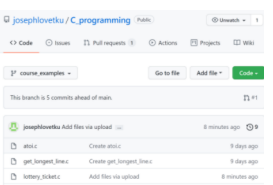
- OnlineGDB (https://www.onlinegdb.com/)

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



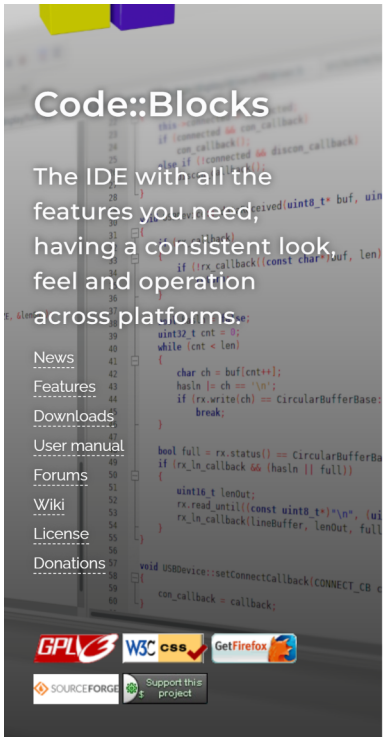
- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

My GitHub page:
click the link here to visit.



Platform/IDE

- https://www.codeblocks.org/



Code::Blocks

Code::Blocks

The free C/C++ and Fortran IDE.

Code::Blocks is a free C/C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable.

Built around a plugin framework, Code::Blocks can be extended with plugins. Any kind of functionality can be added by installing/coding a plugin. For instance, event compiling and debugging functionality is provided by plugins!

If you 're new here, you can read the **user manual** or visit the **Wiki** for documentation. And don't forget to visit and join our **forums** to find help or general discussion about Code::Blocks.

We hope you enjoy using Code::Blocks!

The Code::Blocks Team

Latest news

Migration successful

We are very happy to announce that the process of migrating to the new infrastructure has completed successfully!

Read more

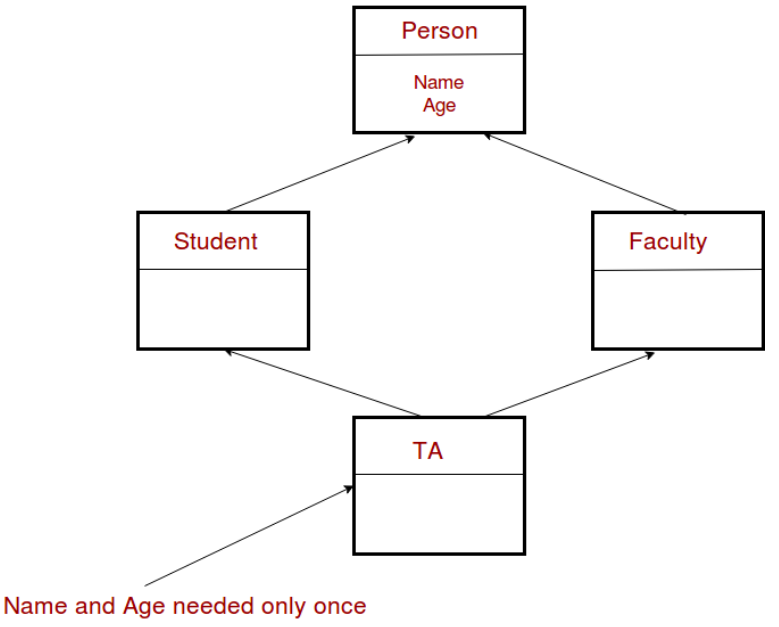
The Diamond Problem

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

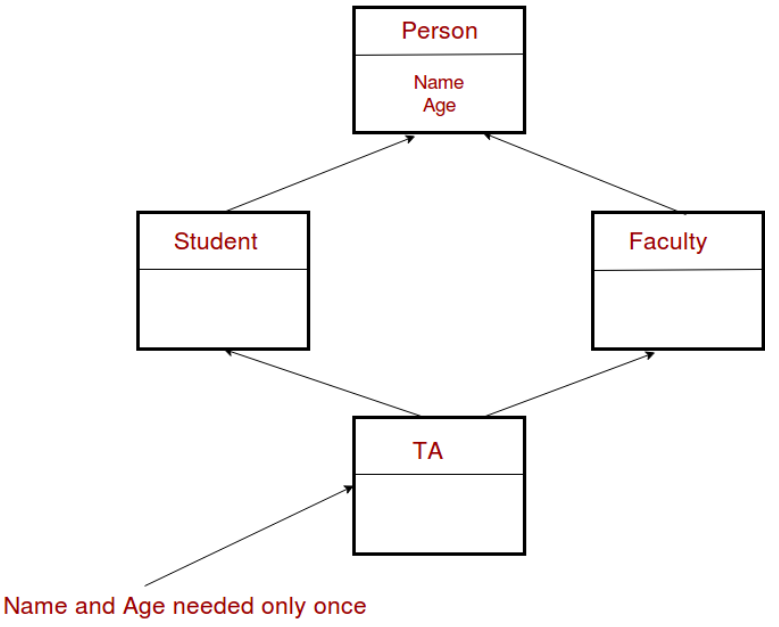
https://www.geeksforgeeks.org/multiple-inheritance-in-c/

```
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)

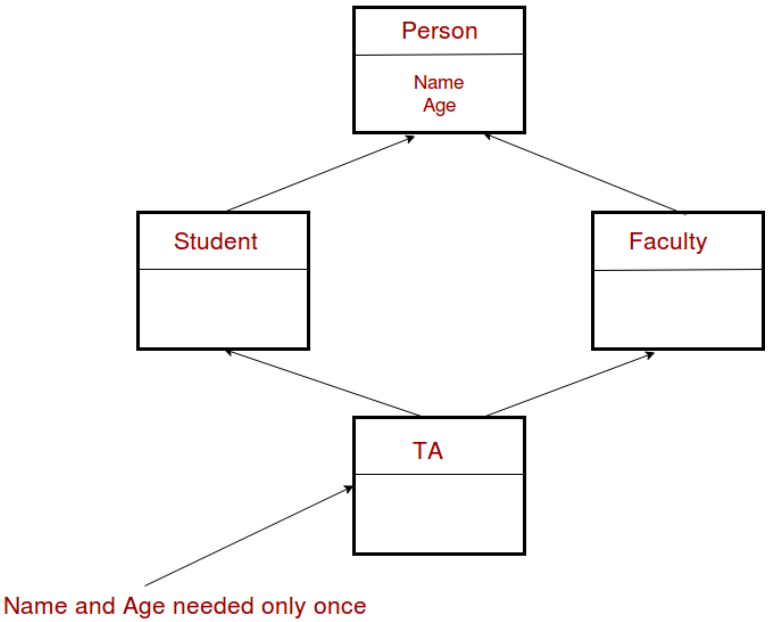
https://www.geeksforgeeks.org/multiple-inheritance-in-c/

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?

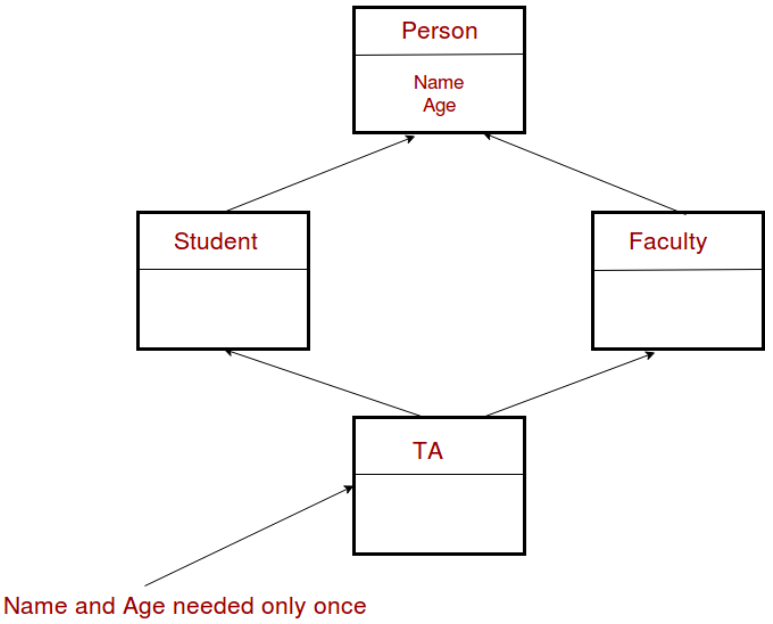
The Diamond Problem (solution)

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;
    }
};
```

Some more exercises...

- https://www.cs.purdue.edu/homes/bxd/CandC++/cpp.ch8.html
- https://courses.cs.washington.edu/courses/cse333/12su/lectures/lec12.pdf
  https://courses.cs.washington.edu/courses/cse333/12su/lectures/lec14.pdf
- Template slides + Exercises CSE 333 2021.
  https://courses.cs.washington.edu/courses/cse333/21sp/sections/06/cse333_sec06_21sp_slides.pdf
- From C++ Practical Programming:

  https://www.linuxtopia.org/online_books/programming_books/c++_practical_programming/c++_practical_programming_138.html
- From London University:

  https://github.com/CPPLondonUni/templates_exercises

Some more exercises

- Stack with nodes that contains multiple data types and structures.
  - https://onlinegdb.com/hPOk1YpxY (try to make Vector be a templated one)
- Template Specialization

```
template <class T>
int comp(const T& a, const T& b);

template <class T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}
```

```
int main() {
    cout << comp<int>(10, 20) << endl;
    cout << comp<string>("TKU", "NCTU")
         << endl;
    return 0;
}
```

```
-1
-1
```

Template Specialization

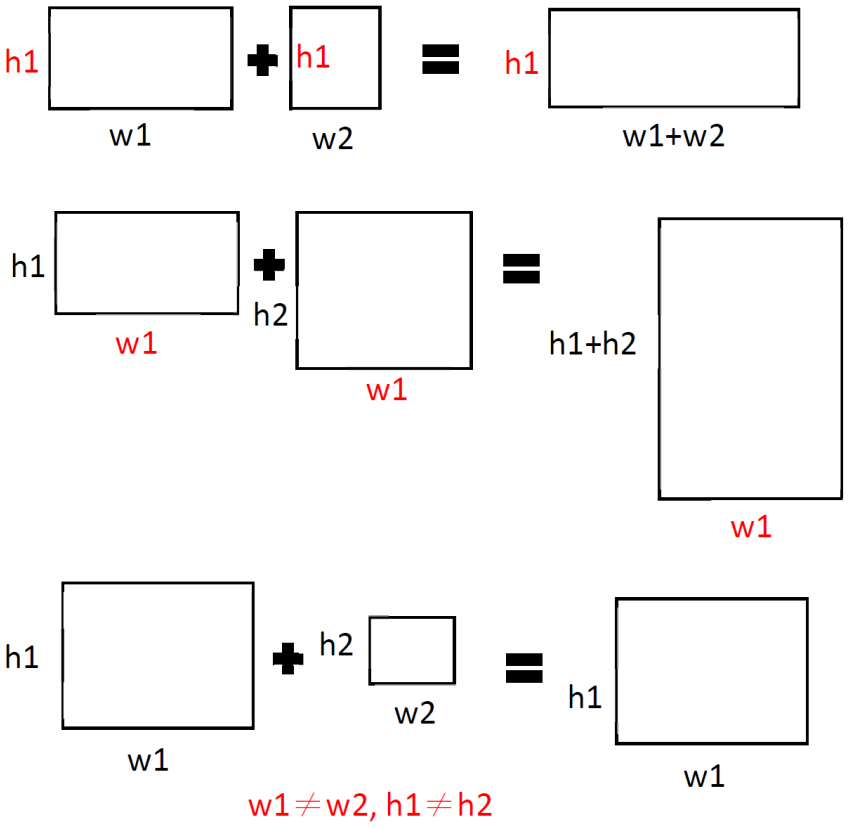
```
template <class T>
class A {
    public:
        T addition (const T& a, const T& b);
};

template <class T>
T A<T>::addition(const T& a, const T& b) {
    return a+b;
}
```

```
int main() {
    A<int> obj1;
    A<string> obj2;
    cout << obj1.addition(10, 20) << endl;
    cout << obj2.addition("TKU", "CSIE") << endl;
    cout << endl;
    return 0;
}
```

```
30
7
```

Rectangle + Operator Overloading

Exercise: https://onlinegdb.com/v2sOAhsNe



Inheritance of a templated class

- https://onlinegdb.com/GOB_A5Gt_

Template Class + Operator Overloading

- https://onlinegdb.com/rPoSGoibq6

Supplementary

- Why not use ‘virtual’ always?
  - Efficiency concern.
    - Non-virtual function is faster.
  - Controllable:
    - If a function `f()` calls `g()` in some class `A` and `g()` is not virtual, then we are guaranteed that we call `A::g()` and not `g()` in some other classes.
- Virtual can be sticky…
  - If `A::f()` is declared virtual, then it (vtable) would be created for class A and all its subclasses.

Clock (Exercise: make it a template)

- https://www.onlinegdb.com/edit/kCTGleTuB

Discussions

- Stack (+vector in template)
- Template specialization
- A friend function cannot be inherited (example).

Turtle vs. Rabbit

Problem

有三個類別 Animal, Turtle 以及 Rabbit，其中Turtle 與 Rabbit繼承於 Animal類別。
Turtle 與 Rabbit分別擁有起始位置(int position) 與移動函數 (void move(int))。
其中烏龜每一步之步長為1，兔子每一步步長為 10，
而 compare_position(Animal&, Animal&)函式能夠比較 Turtle物件（即烏龜）與Rabbit物件（即兔子）移動後的位置，
若烏龜的位置大於或等於兔子的位置則烏龜獲勝。**請根據註解說明完成或修改所需部分。**

```
class Animal {
public:
    Animal() = default;
    Animal(int pos): position(pos) {}
    void move(int); //請修改為純虛擬函式使 Animal 為抽象類別
private:
    /* 請宣告 compare_position() 函式為友誼函式 */
};
class Rabbit : public Animal {
public:
    Rabbit(int pos): position(pos) {}
    void move(int steps) { position += 10*steps; }
};
class Turtle : public Animal {//請修改必要部分
public:
    Turtle(int pos): position(pos) {};
    void move(int steps) { position += steps; }
};
/* 請實作 bool compare_position()函式 */
```

```
int main() //請勿修改
{
    int t_pos, r_pos, t_steps, r_steps;
    cin >> t_pos >> r_pos >> t_steps >> r_steps;
    Turtle t(t_pos);
    Rabbit r(r_pos);
    Animal &a1 = t, &a2 = r;
    a1.move(t_steps);
    a2.move(r_steps);
    if (compare_position(a1,a2))
        cout << "The turtle wins!" << endl;
    else
        cout << "The rabbit wins!" << endl;
    return 0;
}
```

範例輸入	範例輸出
10 0 5 2	The rabbit wins!
100 10 50 10	The turtle wins!