C++ Class Inheritance

한양대학교 컴퓨터소프트웨어학부
2017년 2학기

Class Inheritance

What is class inheritance?

- Build a class on top of existing classes.
  - Minimize re-implementing similar functionalities.
  - Establish relations between classes/types.
  - Customized functionalities.
  - Abstract class or interface.

Class Inheritance Example

```cpp
// Car class.

class Car {
 public:
  Car() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;
  int GetCapacity() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
  int capacity_;
};
```

```cpp
// Truck class.

class Truck {
 public:
  Truck() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;
  double GetMaxLoad() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
  double max_load_;
};
```

Class Inheritance

When do we use inheritance?

- "Is-a" relationship: use (public) inheritance when A is a B.
    - A car is a vehicle.
      A truck is a vehicle.
      A cart is a vehicle.
      …
    - A student is a person.
      A professor is a person.
      …
    - A person is an animal.
      A dog is an animal.
      ...

Class Inheritance

A classes inherits other classes' all members.

- If a class A inherits another class B,
  - The members of B is accessible in A's member functions.
  - A can have additional member variables and functions.

- Parent, child, ancestor, descendant, class hierarchy.

Class Inheritance Example

```cpp
// Car class.

class Car {
 public:
  Car() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;
  int GetCapacity() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
  int capacity_;
};
```

```cpp
// Truck class.

class Truck {
 public:
  Truck() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;
  double GetMaxLoad() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
  double max_load_;
};
```

Class Inheritance Example

```
// Vehicle class.

class Vehicle {
 public:
  Vehicle() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
};
```

```
// Car class.

class Car : public Vehicle {
 public:
  Car() : Vehicle() {}

  int GetCapacity() const;

 private:
  int capacity_;
};
```

```
// Truck class.

class Truck : public Vehicle {
 public:
  Truck() : Vehicle() {}

  double GetMaxLoad() const;

 private:
  double max_load_;
};
```

Class Inheritance Example

```
// Vehicle class.

class Vehicle {
 public:
  Vehicle() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
};
```

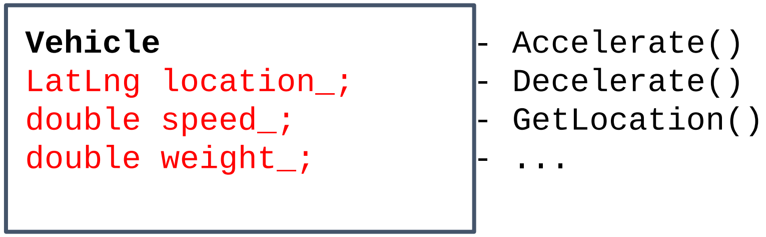
```
// Car class.

class Car : public Vehicle {
 public:
  Car() : Vehicle() {}

  int GetCapacity() const;

 private:
  int capacity_;
};
```

```
// Main routine.

int main() {
  Car car;
  cout << car.GetCapacity() << endl;
  cout << car.GetSpeed() << endl;
  cout << car.GetWeight() << endl;
  return 0;
}
```

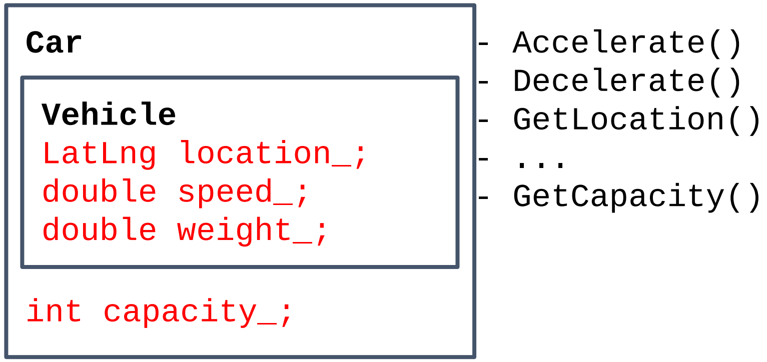
Class Inheritance Example

```
// Vehicle class.

class Vehicle {
 public:
  Vehicle() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
};
```

```
// Car class.

class Car : public Vehicle {
 public:
  Car() : Vehicle() {}

  int GetCapacity() const;

 private:
  int capacity_;
};
```



```
Vehicle
LatLng location_;        - Accelerate()
double speed_;           - Decelerate()
double weight_;          - GetLocation()
                         - ...
```



```
Car                          - Accelerate()
  Vehicle                    - Decelerate()
  LatLng location_;          - GetLocation()
  double speed_;             - ...
  double weight_;            - GetCapacity()

int capacity_;
```

Overriding Member Function

- You can override a member function to provide a custom functionality of the derived class.
- Define a member function with the same name as the inherited function.
  - All ancestor's member functions with the same name will be occluded.
  - To access the ancestor's member functions, use `Ancestor::MemberFunction()`.

Class Inheritance Example

```
// Vehicle class.

class Vehicle {
 public:
  Vehicle() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;

 private:
  LatLng location_;
  double speed_;
  double weight_;
};
```

```
// Car class.

class Car : public Vehicle {
 public:
  Car() : Vehicle() {}

  int GetCapacity() const;

  // Override the parent's GetWeight().
  double GetWeight() const {
    return Vehicle::GetWeight() +
      passenger_weight_;
  }

 private:
  int capacity_;
  double passenger_weight_;
};
```

```
// Main routine.

int main() {
  Car car;
  cout << car.GetCapacity() << endl;
  cout << car.GetSpeed() << endl;
  cout << car.GetWeight() << endl;
  return 0;
}
```

Class Inheritance Example

```
// Vehicle class.

class Vehicle {
 public:
  Vehicle() {}
  void Accelerate();
  void Decelerate();

  LatLng GetLocation() const;
  double GetSpeed() const;
  double GetWeight() const;

 protected:
  LatLng location_;
  double speed_;
  double weight_;
};
```

```
// Car class.

class Car : public Vehicle {
 public:
  Car() : Vehicle() {}

  int GetCapacity() const;

  // Override the parent's GetWeight().
  double GetWeight() const {
    return weight_ + passenger_weight_;
  }

 private:
  int capacity_;
  double passenger_weight_;
};
```

```
// Main routine.

int main() {
  Car car;
  cout << car.GetCapacity() << endl;
  cout << car.GetSpeed() << endl;
  cout << car.GetWeight() << endl;
  return 0;
}
```

Constructors & Destructors

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

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

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

```
int main() {
  {
    Child child;
    cout << endl;
  }
  cout << endl;
  {
    Test test;
    cout << endl;
  }
  cout << endl;
  return 0;
}
```

```
 Parent Child
 ~Child ~Parent
 Parent Child Test
 ~Test ~Child ~Parent
```

Class Inheritance

- Constructor and destructor chain:
  - Optionally call parent's constructor in the constructor.
  - Its own destructors and all ancestors' will be automatically called.

- Class member access control revisited.
  - `public`: everyone can access.
  - `private`: only its member functions can access.
  - `protected`: its member functions and
    the member functions of descendant classes can access.

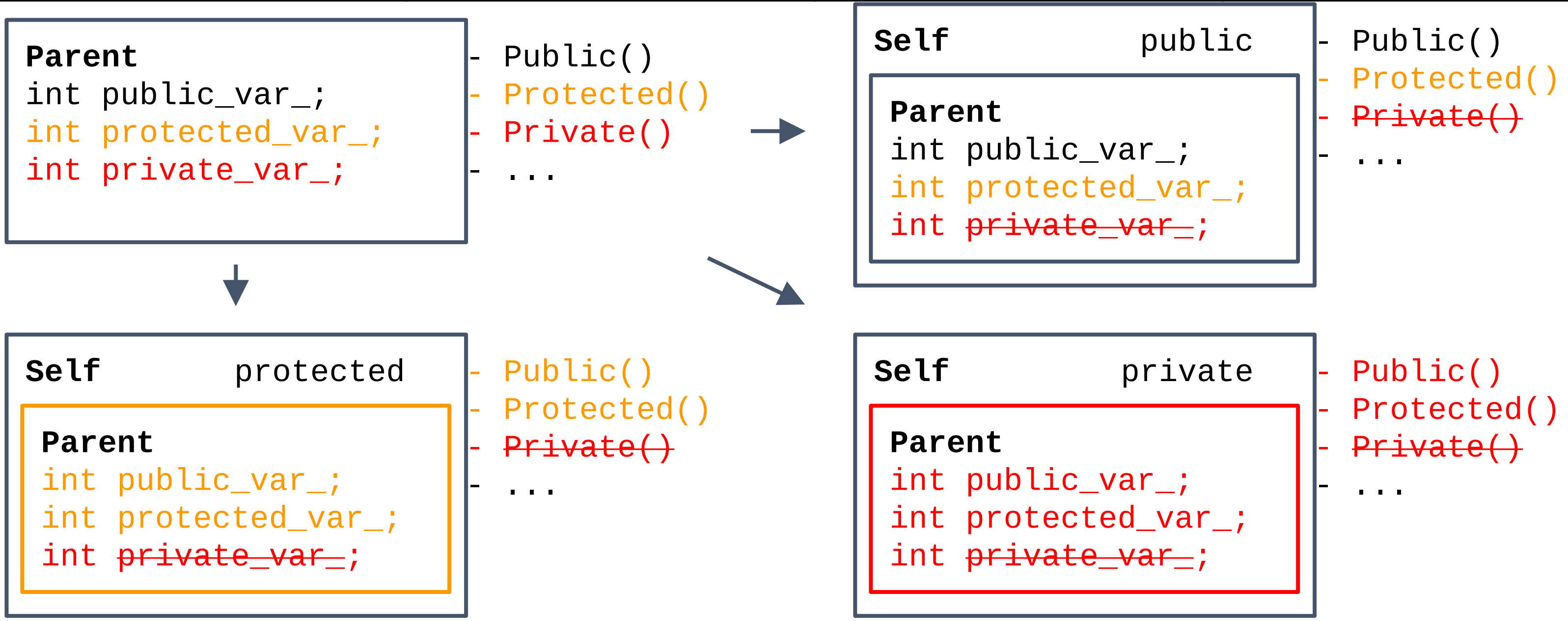
Class Inheritance Types

- Types of inheritance: `public`, `protected`, and `private`.
    - Depending on the inheritance types, the parent’s member has different access control IN the child class.
    - Most commonly used is public inheritance
      (and probably it's the only useful inheritance).

Type of inheritance	Parent’s public member	protected member	private member
public	public	protected	x
protected	protected	protected	x
private	private	private	x

Class Inheritance Types

Type of inheritance	Parent's public member	protected member	private member
public	public	protected	x
protected	protected	protected	x
private	private	private	x



Parent
int public_var_;
int protected_var_;
int private_var_;

- Public()
- Protected()
- Private()
- ...

Self public

Parent
int public_var_;
int protected_var_;
int ~~private_var_~~;

- Public()
- Protected()
- ~~Private()~~
- ...

Self protected

Parent
int public_var_;
int protected_var_;
int ~~private_var_~~;

- Public()
- Protected()
- ~~Private()~~
- ...

Self private

Parent
int public_var_;
int protected_var_;
int ~~private_var_~~;

- Public()
- Protected()
- ~~Private()~~
- ...

Public Inheritance Example

```
class A {
 public:
  void APublic() {}

 protected:
  void AProtected() {}

 private:
  void APrivate() {}
};

// Public inheritance.
class AA : public A {
 public:
  void AAPublic() {
    APublic();     // OK.
    AProtected();  // OK.
    APrivate();    // Error.
  }

 protected:
  void AAProtected() {
  }

 private:
  void AAPrivate() {
  }
};
```

```
class Client : public AA {
  void Function() {
    APublic();      // OK.
    AProtected();   // OK.
    APrivate();     // Error.

    AAPublic();     // OK.
    AAProtected();  // OK.
    AAPrivate();    // Error.
  }
};
```

```
// Main routine.

int main() {
  AA aa;
  aa.APublic();   // OK.
  aa.AAPublic();  // OK.
  ...
}
```

Protected Inheritance Example

```
class A {
 public:
  void APublic() {}

 protected:
  void AProtected() {}

 private:
  void APrivate() {}
};

// Protected inheritance.
class BA : protected A {
 public:
  void BAPublic() {
    APublic();     // OK.
    AProtected();  // OK.
    APrivate();    // Error.
  }

 protected:
  void BAProtected() {
  }

 private:
  void BAPrivate() {
  }
};
```

```
class Client : public BA {
  void Function() {
    APublic();      // OK.
    AProtected();   // OK.
    APrivate();     // Error.

    BAPublic();     // OK.
    BAProtected();  // OK.
    BAPrivate();    // Error.
  }
};
```

```
// Main routine.

int main() {
  BA ba;
  ba.APublic();   // Error.
  ba.BAPublic();  // OK.
  ...
}
```

Private Inheritance Example

```
class A {
 public:
  void APublic() {}

 protected:
  void AProtected() {}

 private:
  void APrivate() {}
};

// Private inheritance.
class CA : private A {
 public:
  void CAPublic() {
    APublic();     // OK.
    AProtected();  // OK.
    APrivate();    // Error.
  }

 protected:
  void CAProtected() {
  }

 private:
  void CAPrivate() {
  }
};
```

```
class Client : public CA {
  void Function() {
    APublic();      // Error.
    AProtected();   // Error.
    APrivate();     // Error.

    CAPublic();     // OK.
    CAProtected();  // OK.
    CAPrivate();    // Error.
  }
};
```

```
// Main routine.

int main() {
  CA ca;
  ca.APublic();   // Error.
  ca.CAPublic();  // OK.
  ...
}
```

Other Inheritance Examples

```
// Person class.

class Person {
 public:
  Person(const string& name);

  const string& name() const;
  const string& address() const;
  void ChangeAddress(const string& addr);
};

// Student class.

class Student : public Person {
 public:
  Student(const string& name);

  void RegisterClass(int class_id);
  int GetNumClasses() const;
  int ComputeTuition() const;
};
```

```
// Employee class

class Employee : public Person {
 public:
  Employee(const string& name, int salary);

  int salary() const;
  int ComputeIncomeTax() const;
  void SetSalary(int salary);
};

// Faculty class

class Faculty : public Employee {
 public:
  Faculty(const string& name, int salary);

  void TeachClass(int class_id);
};
```

person.h

```
#ifndef _PERSON_H_
#define _PERSON_H_

#include <string>

class Person {
 public:
  Person(const std::string& name)
      : name_(name) {}

  const std::string& name() const {
    return name_;
  }
  const std::string& address() const {
    return address_;
  }

  void ChangeAddress(const std::string& addr) {
    address_ = addr;
  }

 private:
  std::string name_, address_;
};

#endif
```

student.h

```
#ifndef _STUDENT_H_
#define _STUDENT_H_

#include <set>
#include "person.h"

class Student : public Person {
 public:
  Student(const std::string& name)
      : Person(name) {}

  void RegisterClass(int class_id) {
    registered_classes_.insert(class_id);
  }

  int GetNumClasses() const {
    return registered_classes_.size();
  }

  int ComputeTuition() const {
    return registered_classes_.size() * 100
        + 500;
  }

 private:
  std::set<int> registered_classes_;
};

#endif
```

employee.h

```
#ifndef _EMPLOYEE_H_
#define _EMPLOYEE_H_

#include "person.h"

class Employee : public Person {
 public:
  Employee(const std::string& name, int salary)
      : Person(name), salary_(salary) {}

  int salary() const { return salary_; }

  void SetSalary(int new_salary) {
    salary_ = new_salary;
  }

  int ComputeIncomeTax() const {
    return salary_ *
        (salary_ > 1000 ? 0.3 : 0.2);
  }

 private:
  int salary_;
};

#endif
```

faculty.h

```
#ifndef _FACULTY_H_
#define _FACULTY_H_

#include <set>
#include "employee.h"

class Faculty : public Employee {
 public:
  Faculty(const std::string& name, int salary)
      : Employee(name, salary) {}

  int GetNumClasses() const {
    return teaching_classes_.size();
  }

  void TeachClass(int class_id) {
    teaching_classes_.insert(class_id);
  }

  int salary() const {
    int num_classes = teaching_classes_.size();
    return Employee::salary() +
        (num_classes <= 2 ? 0 :
            (num_classes - 2) * 100 : 0);
  }

 private:
  std::set<int> teaching_classes_;
};

#endif
```

main.cc

```
#include "employee.h"
#include "faculty.h"
#include "student.h"
using namespace std;

// Let's implement the operator<< to ostream.

int main() {
  Student john("John"), david("David");
  Employee susan("Susan", 200);
  Faculty daniel("Daniel", 100);

  john.ChangeAddress("New York");
  david.RegisterClass(101);
  daniel.TeachClass(101);
  daniel.TeachClass(102);

  cout << john << endl;
  cout << david << endl;
  cout << susan << endl;
  cout << daniel << endl;

  return 0;
}
```

-

```
ostream& operator<<(ostream& os,
                    const Person& p) {
  os << p.name();
  if (!p.address().empty()) {
    os << " (" << p.address() << ")";
  }
  return os;
}

ostream& operator<<(ostream& os,
                    const Student& s) {
  return os << *(Person*) &s
      << " tuition: " << s.ComputeTuition()
      << " (" << s.GetNumClasses()
      << " classes)";
}

ostream& operator<<(ostream& os,
                    const Employee& e) {
  return os
      << static_cast<const Person&>(e)
      << " salary: " << e.salary();
}

ostream& operator<<(ostream& os,
                    const Faculty& f) {
  return os
      << static_cast<const Employee&>(f)
      << " salary: " << f.salary()
      << " (" << f.GetNumClasses()
      << " classes)";
}
```

C++-style Type Casting

- C-style type cast :
  - `(T)var` or `T(var)`.
- C++-style type cast :
  - `static_cast<T>(var)`
    - conversion from a compatible type.
    - no runtime checking.
  - `dynamic_cast<T*>(ptr)`
    - conversion from derived(child) to base(parent).
    - run-time checking.
  - `const_cast<T*>(ptr)`
    - removes 'const' from const T* ptr.
  - `reinterpret_cast<T*>(ptr)`
    - just like C-style cast.
- Refer to http://www.cplusplus.com/doc/tutorial/typecasting/

C++ Type Casting Examples

```
// Implicit conversion.

int a = 2000;
double b = a;

class A {};
class B { public: B(A a) {} };
A a;
B b = a;

// C-style casting.

int a = 2000;
float b = (float) b, c = float(a);

A = a;
B* pb = (B*) &a;

// C++-style casting

double pi = 3.14159265;
int i = static_cast<int>(pi);
```

```
// More C++-style casting examples.

class Base { virtual void dummy() {} };
class Derived: public Base { int a; };
class Other { int b; };

...
  Base* pbd = new Derived;
  Base* pb = new Base;
  Derived* pd;

  pd = dynamic_cast<Derived*>(pb);    // ERROR
  assert(pd == NULL);
  pd = dynamic_cast<Derived*>(pbd);   // OK
  assert(pd != NULL);

  Other* c = reinterpret_cast<Other*>(pb);

  const Base* cp = pb;
  Base* p = const_cast<Base*>(cp);
...
```

Multiple Inheritance

- C++ allows multiple inheritance -
  inheriting from two or more base classes.
    - The derived class has all the members of base classes.
    - What happens if we make a `GraduateStudent` class which inherits `Student` and `Employee`?
        - `Student` and `Employee` are both from `Person`.
    - What happens if base classes has same-named members?
- Avoid using it as much as possible.

Multiple Inheritance Examples

```
class Person {
 public:
  const string& name();
};

class Student : public Person {
 public:
  int GetNumClasses() const;
};

class Employee : public Person {
 public:
  int salary() const;
};

// Multiple inheritance example.

class GraduateStudent
    : public Student, public Employee {
 public:
  GraduateStudent(const string& name,
                  int salary)
      : Student(name),
        Employee(name + "*", salary) {}
};
```

```
int main() {
  GraduateStudent mark("Mark", 50);

  cout << static_cast<Employee&>(mark) << endl;
  cout << static_cast<Student&>(mark) << endl;
  cout << mark.ComputeTuition() << endl;
  return 0;
}
```

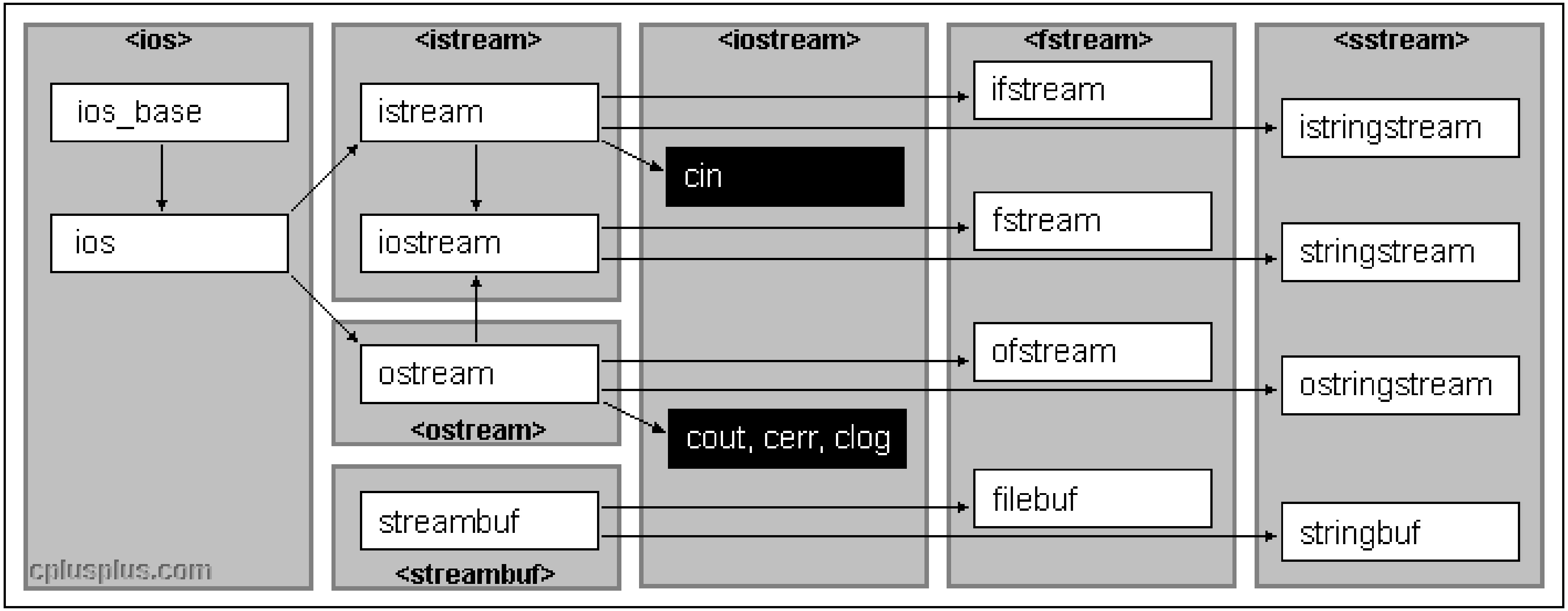
```
Mark* salary: 50
Mark tuition: 500 (0 classes)
```

Multiple Inheritance Examples

```
class Person {
 public:
  const string& name();
};

class Student : public Person {
 public:
  int GetNumClasses() const;
  void DoSomething();
};

class Employee : public Person {
 public:
  int salary() const;
  void DoSomething();
};

// Multiple inheritance example.

class GraduateStudent
    : public Student, public Employee {
// Eror - ambiguous function DoSomething
 public:
  GraduateStudent(const string& name,
                  int salary)
      : Student(name),
        Employee(name + "*", salary) {}
};
```

```
int main() {
  GraduateStudent mark("Mark", 50);

  cout << static_cast<Employee&>(mark) << endl;
  cout << static_cast<Student&>(mark) << endl;
  cout << mark.ComputeTuition() << endl;
  return 0;
}
```

Inheritance Example : std::ios



www.cplusplus.com

- `ios_base` : base class for streams (class)
- `ios` : base class for streams (type-dependent components) (class)

ios_base

State flag functions:

flags	Get/set format flags.
setf	Set specific format flags.
unsetf	Clear specific format flags.
precision	Get/Set floating-point decimal precision.
width	Get/set field width.

ios : ios_base

State flag functions:

good	Check whether state of stream is good.
eof	Check whether eofbit is set.
fail	Check whether either failbit or badbit is set.
bad	Check whether badbit is set.
rdstate	Get error state flags.
setstate	Set error state flag.
clear	Set error state flags.

istream : ios

Formatted input:

operator>>	Extract formatted input.

Unformatted input:

gcount	Get character count.
get	Get characters.
getline	Get line.
ignore	Extract and discard characters.
peek	Peek next character.
read	Read block of data.
readsome	Read data available in buffer.
putback	Put character back.
unget	Unget character.

Positioning:

tellg	Get position in input sequence.
seekg	Set position in input sequence.

Synchronization:

sync	Synchronize input buffer.

ostream : ios

Formatted output:

operator<<	Insert formatted output.

Unformatted output:

put	Put character.
write	Write block of data.

Positioning:

tellp	Get position in output sequence.
seekp	Set position in output sequence.

Synchronization:

flush	Flush output stream buffer

iostream : istream, ostream

Member functions inherited from parents:

ifstream : istream

open	Open file.
is_open	Check if a file is open.
close	Close file.
rdbuf	Get stream buffer.
operator=	Move assignment.
swap	Swap internals.

ofstream : ostream

open	Open file.
is_open	Check if a file is open.
close	Close file.
rdbuf	Get stream buffer.
operator=	Move assignment.
swap	Swap internals.

istringstream : istream

str	Get/set content.
rdbuf	Get stream buffer.
operator=	Move assignment.
swap	Swap internals.

ostringstream : istream

str	Get/set content.
operator=	Move assignment.
swap	Swap internals.

Q&A