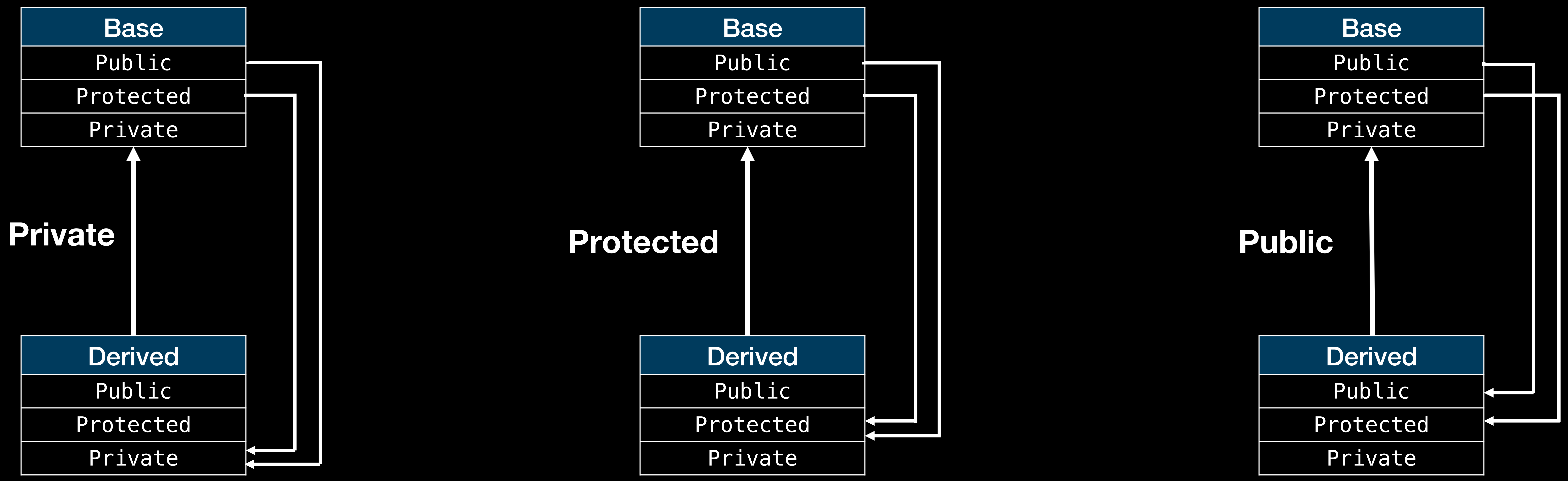
سلسلة تعلم البرمجة بلغة C++ الحديثة

Learn Modern C++ Programming Course

إعداد المهندس أحمد الديب

#37: Derived Classes P2

Access to Base Classes



Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Virtual Functions

```
class Employee {
 public:
  Employee(std::string name, std::string title) : _name{name}, _title{title} {}
  virtual void print() const;


 private:
  std::string _name;
  std::string _title;
};


class Manager : public Employee {
 public:
  Manager(std::string name, std::string title, int level)
      : Employee(name, title), _level{level} {}
  void add_managed(Employee* managed) { list.push_back(managed); }
  void print() const override;


 private:
  std::vector<Employee*> list;  // people managed
  int _level;
};
```

Virtual functions overcome the problems with the type-field solution by allowing the programmer to declare functions in a base class that can be redefined in each derived class. The compiler and linker will guarantee the correct correspondence between objects and the functions applied to them.

```
void print(Employee* emp) { emp->print(); }


int main() {
  //
  Employee ramy{"Ramy", "SW Engineer"};
  Employee hady{"Hady", "SW Engineer"};
  Manager fady{"Fady", "SW Manager", 2};


  fady.add_managed(&ramy);
  fady.add_managed(&hady);


  print(&ramy);
  print(&fady);
}
```

A function from a derived class with the same name and the same set of argument types as a virtual function in a base is said to override the base class version of the virtual function. Furthermore, it is possible to override a virtual function from a base with a more derived return type.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Virtual Functions

- A virtual function must be defined for the class in which it is first declared (unless it is declared to be a pure virtual function).
- A virtual function can be used even if no class is derived from its class, and a derived class that does not need its own version of a virtual function need not provide one.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Polymorphism

- Getting the right behavior from Employee’s functions independently of exactly what kind of Employee is actually used is called polymorphism. A type with virtual functions is called a polymorphic type or (more precisely) a run-time polymorphic type.
- To get runtime polymorphic behavior in C++, the member functions called must be virtual and objects must be manipulated through pointers or references.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Virtual Function Table

- Clearly, to implement polymorphism, the compiler must store some kind of type information in each object of class Employee and use it to call the right version of the virtual function print(). In a typical implementation, the space taken is just enough to hold a pointer.
- The usual implementation technique is for the compiler to convert the name of a virtual function into an index into a table of pointers to functions. That table is usually called the virtual function table or simply the vtbl. Each class with virtual functions has its own vtbl identifying its virtual functions.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Virtual Function Table

```
class Base {
 public:
  virtual void fun1();
  virtual void fun2();
};


void Base::fun1() { std::cout << "Base fun1\n"; }
void Base::fun2() { std::cout << "Base fun2\n"; }


class Derived : public Base {
 public:
  void fun2() override;
};


void Derived::fun2() { std::cout << "Derived fun2\n"; }


void call(Base& b) {
  b.fun1();
  b.fun2();
}

int main() {
  //
  Derived d;
  call(d);
}
```

vtbl of Base	
Name	Address
fun1	Base::fun1
fun2	Base::fun2

vtbl of Derived	
Name	Address
fun1	Base::fun1
fun2	Derived::fun2

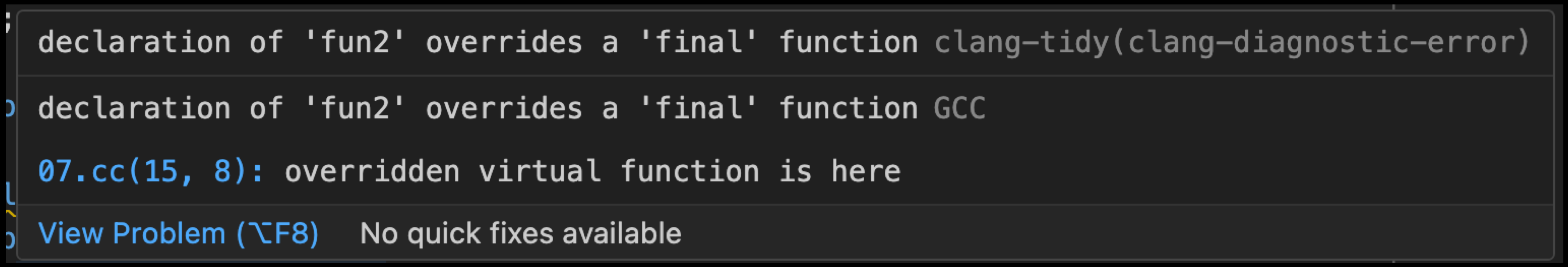
Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

final

```
class Derived : public Base {
 public:
  void fun2() override final;
};

void Derived::fun2() { std::cout << "Derived fun2\n"; }

class AnotherDerived : public Derived {
 public:
  void fun2() override;
};
```

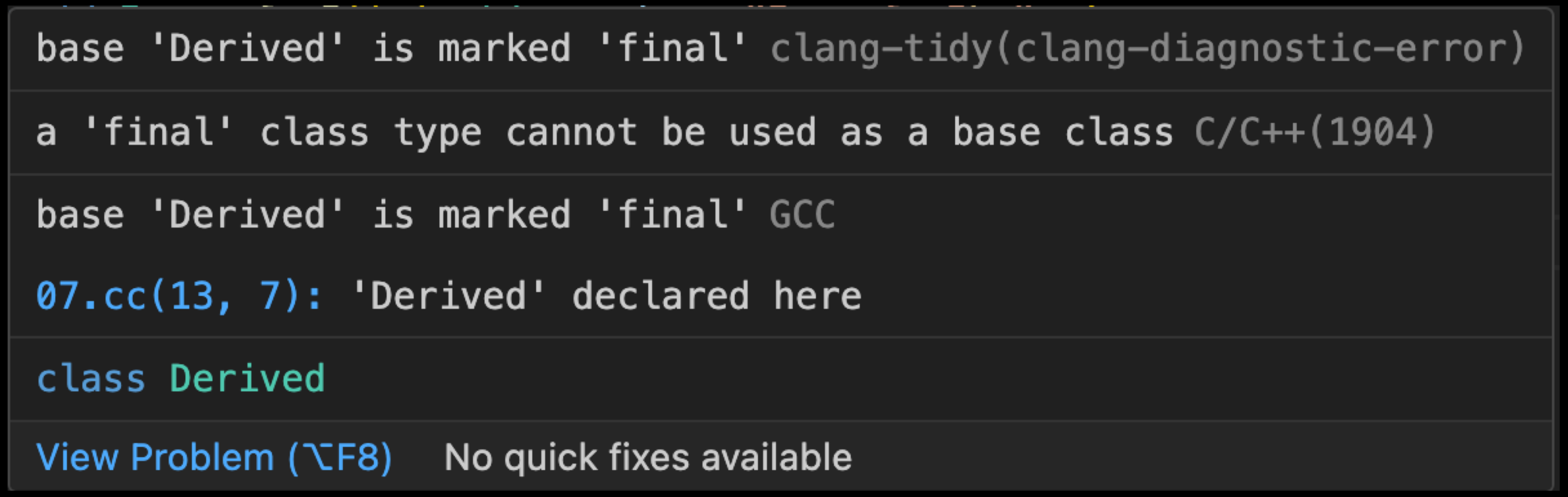


```
declaration of 'fun2' overrides a 'final' function clang-tidy(clang-diagnostic-error)
declaration of 'fun2' overrides a 'final' function GCC
07.cc(15, 8): overridden virtual function is here
```

```
class Derived final : public Base {
 public:
  void fun2() override;
};

void Derived::fun2() { std::cout << "Derived fun2\n"; }

class AnotherDerived : public Derived {
 public:
  void fun2() override;
};
```



```
base 'Derived' is marked 'final' clang-tidy(clang-diagnostic-error)
a 'final' class type cannot be used as a base class C/C++(1904)
base 'Derived' is marked 'final' GCC
07.cc(13, 7): 'Derived' declared here
class Derived
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Inheriting Constructors

```
class Employee {
 public:
  Employee(std::string name, std::string title) : _name{name}, _title{title} {}
  std::string name() const { return _name; }
  std::string title() const { return _title; }


 private:
  std::string _name;
  std::string _title;
};


class EmployeeUpdated : public Employee {
 public:
  using Employee::Employee;  // inherit constructors
  void print() const {
    std::cout << "Name: " << name() << ", Title: " << title() << '\n';
  }
  int x; // we forgot to provide initialization of x
};


int main() {
  //
  EmployeeUpdated ramy{"Ramy", "SW Engineer"};
  ramy.print();
}
```

Constructors are not inherited, if a class adds data members to a base or requires a stricter class invariant, it would be a disaster to inherit constructors.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Abstract Classes

```cpp
class Shape {  // abstract class
 public:
  virtual void rotate(int) = 0;   // pure virtual function
  virtual void draw() const = 0;  // pure virtual function
  // ...
  virtual ~Shape();  // virtual
};


class Circle : public Shape {
 public:
  Circle(Point p, int r) : _center{p}, _radius(r) {}
  void rotate(int) override {}
  void draw() const override;


 private:
  Point _center;
  int _radius;
};
```

Some classes, such as a class Shape, represent abstract concepts for which objects cannot exist. A Shape makes sense only as the base of some class derived from it. A class with one or more pure virtual functions is an abstract class, and no objects of that abstract class can be created.

It is usually important for an abstract class to have a virtual destructor. Because the interface provided by an abstract class cannot be used to create objects using a constructor, abstract classes don't usually have constructors.

A pure virtual function that is not defined in a derived class remains a pure virtual function, so the derived class is also an abstract class. This allows us to build implementations in stages

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Thank you