

C++

Inheritance: access control

1

Example

Let us add a method print() to the class Teacher:

```
Teacher::print()
{
    cout << name << " "
         << employee_id << " "
         << lab
         << endl;
}
```

This code will not compile

2

Example: remarks

- The previous example does not compile
- **Teacher::print()** can not access the member **name**
 - Reason: **name** is a **private** member of the base class, therefore is accessible to the base class only
 - Despite the public inheritance, private data can only be accessed by their own class and not by other classes (including derived classes)

3

Example: solution

- One solution to the previous problem is to make the member data **name** in the class **Person** **protected** instead of private

```
class Person {
protected:
    string name;
    // ...
};
```

```
class Student {
protected:
    int student_id;
    int year;
    // ...
};
```

```
class Teacher {
protected:
    int employee_id;
    std::string lab;
    // ...
};
```

4

Access control in C++

- C++ offer three levels of access control:
 - Public: members (data and methods) can be used by the class and everybody else (other classes, functions, etc)
 - Protected: members can be accessed by the class (and its friends) and its derived classes
 - Private: members can be accessed only by the class (and its friends)
- Remark: without inheritance private and protected are the same

5

Protected vs private

- When to use protected and when to use private ?
- Protected is weakening encapsulation so:
 - Use protected only when you have no choices and private rest of the time
- Example:
 - in class Person, we decide to change the type of name from string to a class CName then:
 - If name is protected then derived classes need to have their code reviewed since implementation details of the base class changed
 - If name is private, since nobody else could access it, no code outside of the class will break

6

Protected vs private

- For the previous example, it is better to keep **name** private and used the accessor **get_name()** and **set_name()** that are provided by the base class

7