Inheritance: Introduction
Example: University Admin Info

• Let’s implement a system for maintaining university administration information.

  – Teacher and Student are two completely separate classes. Their implementation uses separate code.

  – However, they share many methods and members that are implemented in the same way: handling name, address, and department.

  – Why do we implement the same function twice?

  – This is not good software reuse!
enum Department { accounting, business, engineering, mathematics, unknown };
class Student
{
    private:
        string name;
        string address;
        Department dept;
        Course *enrolled;
        int num_courses;

    public:
        Student(string n, string a, Department d) :
            name(n), address(a), dept(d), enrolled(NULL), num_courses(0) { };

        void set_name(const char* name);
        void set_address(const char* adr);
        void set_department(Department dept);
        string get_name() const;
        string get_address() const;
        Department get_department() const;
        bool enroll_course(const string&);
        bool drop_course(const Course&);  
};
Example: U. Admin Info – teacher.h

enum Rank { instructor, assistant_prof, associate_prof, professor, dean };
class Teacher
{
    private:
        string name;
        string address;
        Department dept;
        Rank rank;
    public:
        Teacher(string n, string a, Department d, Rank r) :
            name(n), address(a), dept(d), rank(r) { };
        void set_name(const char* name);
        void set_address(const char* adr);
        void set_department(Department dept);
        void set_rank(Rank rank);
        string get_name() const;
        string get_address() const;
        Department get_department() const;
        Rank get_rank() const;
};
Things to Consider

• We want a way to say that Student and Teacher both have the same members: name, address, dept, but yet require them to keep a separate copy of these members.

• We want to share the code for set_name, etc., between Student and Teacher as well.

• We want this code to act like member functions (to permit consistency of state of the objects), so they cannot be written as global functions.
Solution 1: Re-use by Copying

- Copy the code from one class to the other class, and change the class names.
  - This is very error prone.
  - It is also a maintenance nightmare.
  - What if we find a bug in the code in one class?
  - What if we want to improve the code? Perhaps by introducing a new class `Address`.

- “REUSE by COPYING” is a bad idea!
Inheritance

- Inheritance enables code reuse.
- Inheritance is the ability to define a new class based on an existing class with a hierarchy.
- The derived class *inherits* the data members and member methods) of the base class.
- New members and methods can be added to the derived class.
- Since the new class only has to implement the behavior that is different from the base class, we can reuse the code for the base class.
- “Inheritance” is the traditional term, but C++ calls it “derivation”.
Solution 2: By Inheritance – person.h

class Person
{

private:
    string name;
    string address;
    Department dept;

public:
    Person(string n, string a, Department d):
        name(n), address(a), dept(d) {}
    void set_name(const char* name);
    void set_address(const char* adr);
    void set_department(Department dept);
    string get_name() const;
    string get_address() const;
    Department get_department() const;
};
Solution 2: By Inheritance – student.h

class Student : public Person
{

private:
    Course* enrolled;
    int num_courses;

public:
    Student(string n, string a, Department d) :
        Person(n, a, d), enrolled(NULL), num_courses(0) { }

    bool enroll_course(const string&);
    bool drop_course(const Course&);
};
class Teacher : public Person
{
  private:
    Rank rank;

  public:
    Teacher(string n, string a, Department d, Rank r) :
      Person(n, a, d), rank(r) {}

    void set_rank(Rank rank);
    Rank get_rank() const;
};
Inheritance

• Person is the base class of Student.

• Student is a derived class of Person.

• The effect is that Student inherits all data members and methods from Person.

• The data members of Student are the data members of Person (name, address, dept), plus the extra data members declared in the definition of Student (enrolled, num_courses).
Example: Inherited Members

```cpp
void some_func(Person& person, Student& student)
{
    cout << person.get_name() << endl;
    cout << student.get_name() << endl;

    student.set_department(engineering);
    Department dept = person.get_department();
    student.enroll_course("COMP151");
    person.enroll_course("COMP001");       // Error!
}
```
“Is-a” Relationship

• Inheritance implements the is-a relationship.
  – Recall: membership (composition) implements the has-a relationship.

• Since Student inherits from Person,
  – every object of type Student can be used like an object of type Person
  – all methods of Person can be called on a Student object

• In other words, a Student object definitely is a Person object under all circumstances.

• In general: a derived class object can be treated like a base class object under all circumstances.
Example: Derived Objects as Base Class Object

```cpp
bool print_mailing_label(const Person& person)
{
    string name = person.get_name();
    string adr = person.get_address();

    // code to print the label
}
```

- Since a Student is a Person, we can print a mailing label for a student like this:

```cpp
Student student(“Tom”, “Sai Kung”, mathematics);
print_mailing_label(label(student));
```
Direct and Indirect Inheritance

• Let’s add a new class PG_Student:

```cpp
class PG_Student : public Student
{
    private:
        Topic research_topic;
    public:
        PG_Student(string n, string a, Department d)
            : Student(n, a, d), research_topic(NONE) {}
        void set_topic(const Topic& x) { research_topic = x; }
};
```

• PG_Student is directly derived from Student.
• It is indirectly derived from Person.
• So a PG_Student object is a Person object.
• Person is called an indirect base class for PG_Student.