

Lab 4: Inheritance and STL

CSCI 104



Why Inheritance?

- Code reuse!!
- Makes code more readable and less repetitive
- Similar concept as for loop

```
cout << arr[0] << endl;  
cout << arr[1] << endl;  
cout << arr[2] << endl;
```



```
cout << i + " : " + arr[i],
```

What is Inheritance?

- Major in OOP
- Class B inherits from Class A
 - Class B = child class
 - Class A = parent/base class
 - Class B can access all the data members and functions of Class A
 - Class B can also create new data members and functions AND overwrite functions from Class A

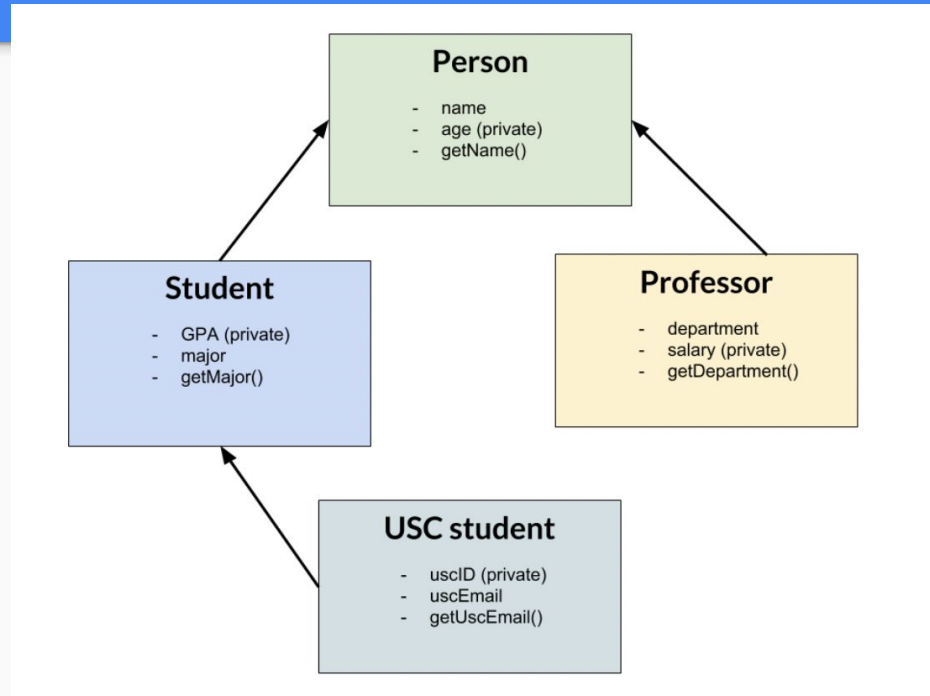
```
class B: public A
{
    // ...
}
```

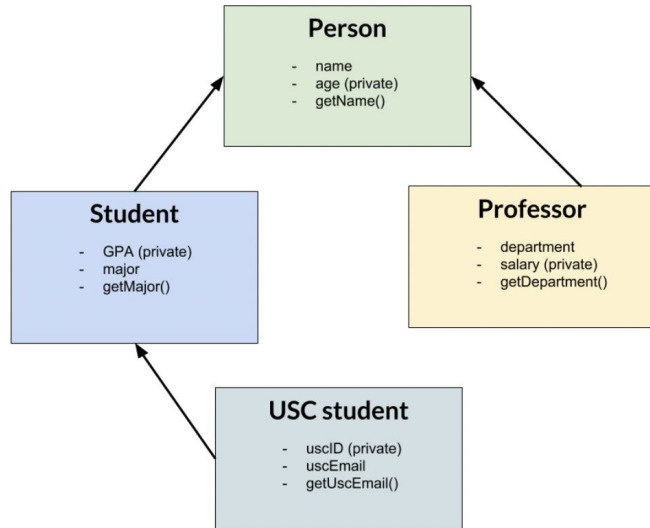
Example

- Print out majors for students and department for professors
- Print out names for everyone

```
class Student {  
    public:  
        Student(std::string name, std::string major);  
        std::string getName();  
        std::string getMajor();  
    private:  
        std::string mName;  
        std::string mMajor;  
};  
  
class Professor {  
    public:  
        Professor(std::string name, std::string department);  
        std::string getName();  
        std::string getDepartment();  
    private:  
        std::string mName;  
        std::string mDepartment;  
};
```

Example but with Inheritance





```
class Person {
    public:
        Person(std::string name);
        std::string getName();
    private:
        std::string mName;
        int mAge;
};

class Professor : public Person {
    public:
        Professor(std::string name, std::string department);
        std::string getDepartment();
    private:
        int mSalary;
        std::string mDepartment;
};

class Student : public Person {
    public:
        Student(std::string name, std::string major);
        std::string getMajor();
    private:
        std::string mMajor;
};

class UscStudent : public Student {
    public:
        UscStudent(std::string name, std::string major);
        std::string getUscEmail();
    private:
        int mUscID;
        std::string mUscEmail;
};
```

Private, Protected, Public

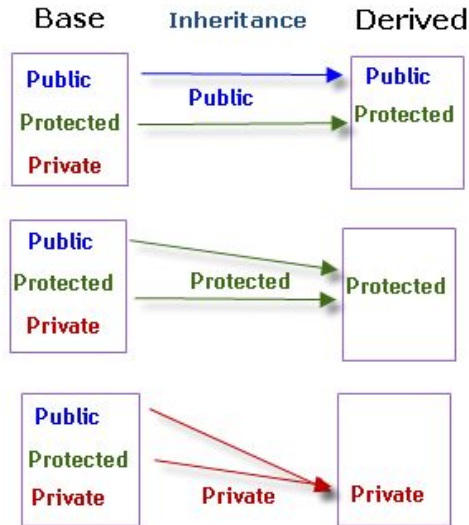


Fig: Visibility after inheritance

- *public*: anyone who works with your class can call the function/use the variable
 - *protected*: only derived classes can see these functions/variable
 - *private*: no one outside the class can access this directly
-
- **Ask yourself 2 questions:**
 - What does the base class let me see?
 - (everything public and protected)
 - What do I let other classes see?
 - (choose your access accordingly)

Polymorphism

- Determining whether to use the function in child or base class

```
class Person {  
    public:  
        printTitle(); // prints "Person"  
};  
  
class Professor {  
    public:  
        printTitle(); // prints "Professor"  
};  
  
class Student {  
    public:  
        printTitle(); // prints "Student"  
};  
  
class UscStudent {  
    public:  
        printTitle(); // prints "USC Student"  
};
```


Static Binding

- Looks at the Type of pointer

```
UscStudent* u = new UscStudent();  
u->printTitle(); // will print "USC Student"
```

What will this print out?

```
Person* p = new UscStudent();  
p->printTitle();
```

Person

```
class Person {  
    public:  
        printTitle(); // prints "Person"  
};  
  
class Professor {  
    public:  
        printTitle(); // prints "Professor"  
};  
  
class Student {  
    public:  
        printTitle(); // prints "Student"  
};  
  
class UscStudent {  
    public:  
        printTitle(); // prints "USC Student"  
};
```

Dynamic Binding

- Virtual keyword
- Looks at the type of object that is being pointed at
- Note: all base classes should have virtual destructor

```
Person* p = new UscStudent();  
p->printTitle(); // USC Student
```

```
class Person {  
    public:  
    virtual void printTitle(); // prints "Person"  
};  
  
class Professor {  
    public:  
    void printTitle(); // prints "Professor"  
};  
  
class Student {  
    public:  
    virtual void printTitle(); // prints "Student"  
};  
  
class UscStudent {  
    public:  
    void printTitle(); // prints "USC Student"  
};
```

Abstract Classes

- Class that has at least one pure virtual function
- Virtual function
 - Member function declared in the base class
 - re-defined/overridden in base class
- Pure virtual function
 - Virtual function
 - Only declare it in the base class
 - Implement in the child classes
 - Indication by “=0”

Abstract Class Example

```
class Shape {  
    public:  
        virtual double getArea() = 0; // = 0 indicates that this class doesn't implement this  
        virtual double getPerimeter() = 0;  
}
```

- Child classes could include: Circle, Rectangle, Triangle, etc
- To instantiate this class, we need to implement these functions in the children classes

Summary

- **Syntax:**

```
class B : public A
{
    // ...
}
```

- **Dynamic binding:** go back and check what you are @ function call
 - “virtual”
- **Static binding:** whatever you’re labelled as is what it runs

- “IS-A” relationship (B is-a A)
- **Scope of inheritance**
 - public - everything stays the same
 - protected - all public elements become protected
 - private - everything is private
- **abstract** (virtual function = 0)
 - you can’t instantiate it!

Part 1: Constructors

- Run make in part1 of the folder
- ERROR: “no matching function for call to ‘Person::Person()’ ”
- Compiler confused
 - Inheriting from Person class, need to call constructor
 - Since we didn’t call constructor, default constructor implicitly called
 - But there’s no default constructor for Person
 - Need to explicitly call Person constructor

```
Student::Student(std::string name, std::string major) : Person(name) {  
    // rest of student constructor  
}
```

Make these changes (to Student, Professor, and UscStudent), and now your code should compile.

Inheritance Visibility

- Write public function `printTranscript()` in `UscStudent` class which prints out name of school, student's name, GPA, and their major
- PROBLEMS when compiling!

Inheritance Visibility

- Need to change access level of GPA data member
- Would compile if we made it public but we don't want it to be public because then even third parties can access
- Still want to access it from UscStudent Class
- What should we do???

Choosing an Access Modifier

- UscStudent is type of student: needs same data members as Student
- But we don't want outsiders to have access to setGPA() function
- What inheritance should we use???

```
class UscStudent : protected Student {
```

Part 2: STL (standard template library)

Maps

Maps

- Key-value pairs of items
- Operations: search, remove, insert
 - More specifics on these operations in the Bytes page lab writeup
- All $O(\log(n))$

Iterators

- If we want to loop through all elements
- This for loop will take $O(n)$
- Very similar to normal for loops
- Use `.begin()` and `.end()`

```
std::map<std::string, std::string>::iterator it;
for(it = myMap.begin(); it != myMap.end(); ++it)
{
    std::cout << it->first << std::endl;
    std::cout << it->second << std::endl;
}
```

Iterator notes:

for an array:

```
for(int i = 0; i < n; ++i){  
    std::cout << arr[i] << std::endl;  
}
```

for a map

```
std::map<std::string, std::string>::iterator it;  
for(it = myMap.begin(); it != myMap.end(); ++it)  
{  
    std::cout << it->first << std::endl;  
    std::cout << it->second << std::endl;  
}
```

- same thing: initialization; termination condition; increment
- make sure your iterators are from the **same** map:

```
std::map<std::string, std::string>::iterator it;  
for(it = example.getMap().begin(); it != example.getMap().end();  
++it){  
    //more code here  
}
```

do NOT !! do this! won't compare correctly! 2 calls to getMap() = 2 instances

STL

Sets

- Similar to maps
- Only have keys (no values)
- Keys are unique
- Use iterator to walk through all elements

```
// insert into the set
set<string> radioStations;
radioStations.insert("KCRW");
radioStations.insert("KXSC");
string stationName = "KPWR";
radioStations.insert(stationName);

// iterating through the set
for(set<string>::iterator it=radioStations.begin(); it != radioStations.end(); ++it)
{
    // note that we don't have the concept of it->first or it->second, because there are no values, only keys
    cout << "Station: " << *it << endl;
}

stationName = "KPWR";

// find an element
if(radioStations.find(stationName) != radioStations.end()) {
    cout << stationName + " is a radio station!" << endl;
}
else {
    cout << "Couldn't find this station!" << endl;
}

radioStations.erase("KCRW"); // remove KCRW from the set of names
// if we try to find "KCRW" now, find() will return radioStations.end()
```

Cheat Sheet

insert	<code>.insert()</code> <code>map["key"] = __</code>
find	<code>.find()</code> <code>map["key"]</code>
remove	<code>.erase()</code>
iterate	<code>.begin()</code> to <code>.end()</code>

- to see what functions are available to you, google it!
- "c++ stl map" or "c++ stl set"
- <https://www.cplusplus.com/reference/map/map/>
- <https://www.cplusplus.com/reference/set/set/>

The Lab

- Follow the bytes page on the lab
- Part1 helps conceptually and the lab page walks you through everything
- Then look at part 2 files
 - Three major classes: Schedule, Assignment, Course
 - Functions in Assignment and Course are complete, but you need to make small change to Assignment to pass all tests
 - Need to implement functions in Schedule
- “Make” will run tests for you
- Need to pass all tests to get checked off OR be working throughout the whole lab section