Lab 5: Inheritance and STL
Why use Inheritance?

- It's another way to reuse our code!
- Reusing code makes it more readable and less repetitive
- Similar concept to for loops

```cpp
cout << arr[0] << endl;
cout << arr[1] << endl;
cout << arr[2] << endl;
```
What is Inheritance?

- Core concept in OOP!
- Class B inherits from Class A
  - Class B = child class
  - Class A = parent/base class
  - Class B can access the data members and functions of Class A
  - Class B can also:
    - Create its own data members and functions
    - Overwrite functions from Class A
Example

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

```cpp
#include <string>

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

- Common data members and functions can go into the parent class Person
- No need to copy code for each class!
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();
        int mSalary;
        std::string mDepartment;
};

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

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

```cpp
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???
Private, Protected, Public

- 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???

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

---

Fig: Visibility after inheritance
Polymorphism

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

```cpp
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

```cpp
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"
};
```

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

What will this print out?

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

Person
Dynamic Binding

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

```cpp
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"
};

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

```cpp
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
STL - Maps

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

```cpp
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;
}
```
STL - Sets

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

```cpp
// 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
    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()
```
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