CS 162
Intro to Computer Science II

Lecture 16

Inheritance

2/23/24

Oregon State University

Odds and Ends

- Assignment 3 due Sunday midnight

Today’s topics

- Inheritance

Big Three Activity

Function	Prototype	Job	When is it called	Default Behavior if not defined?

Constructor	ClassName();
	ClassName(w/ params)
		Build the object	Default is called when object is declared with no parameters and no “=“ sign. Nondefault is called if parameters are given	The compiler will provide a default one. It will initialize all variables with garbage values, will not set up pointers

Copy Constructor				
Assignment Operator Overload				
Destructor				

Big Three Activity

Function	Prototype	Job	When is it called	Default Behavior if not defined?

Constructor	ClassName();
	ClassName(w/ params)
		Build the object	Default is called when object is declared with no parameters and no “=“ sign. Nondefault is called if parameters are given	The compiler will provide a default one. It will initialize all variables with garbage values, will not set up pointers

Copy Constructor	ClassName(const ClassName &);	Copies the contents of the passed in object to the destination object	1. Pass by value 2. Return value 3. When initializing an object with this constructor	Shallow copy, will only copy the values stored in each variable

Big Three Activity

Function	Prototype	Job	When is it called	Default Behavior if not defined?
Assignment Operator Overload	ClassName & operator=(const ClassName &);	Copies the contents of the right operand to the left operand	When setting an object of the same class type to another object of the same class type	Shallow copy, will only copy the values stored in each variable
Destructor	~ClassName();	Destroys the object	Any time an object goes out of scope 1. When a function ends 2. When the program ends 3. A block containing a local variable ends 4. A delete operator is called	Will delete anything on the stack

Asm3 Hints:

- Which class needs Big 3?
- Where to implement the “add a flight” functionality?
- Where to implement the “remove a flight” functionality?

- Is it a good practice to access Flight internals from the Manager class?
    - i.e., get_airports()[0].get_flight()[0].get_flight_number()?
    - NO!!! THIS VIOLATES THE RULE OF ENCAPSULATION!!!!

- Game flow?
- What’s inside your main()? driver.cpp?
- Frequently check memory leaks!!!

Introduction to Inheritance

- Suppose that we implement two C++ classes with the following member variables:
  - Student:
    - ID
    - Email address
    - Phone number
    - Major
    - GPA
  - Instructor
    - ID
    - Email address
    - Phone number
    - Office
    - Office hours
    - Salary

Basic of Inheritance

- The process by which a new class is created from another class
- Derived (Child) class: Classes that inherit properties
- Base (Parent) class: more general class which derived class are created from

- Examples:

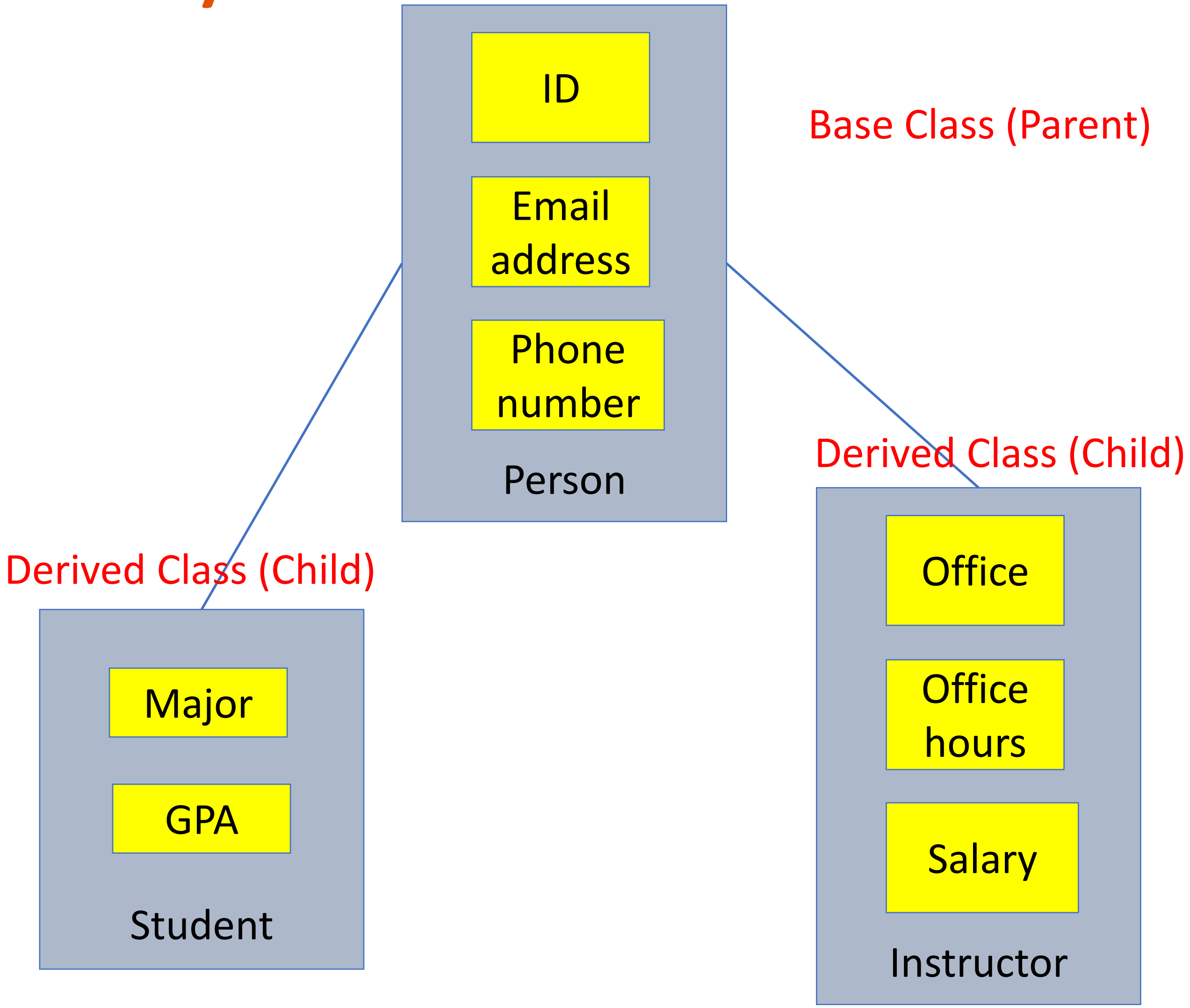
• Parent: Animal	Child: Dog, Cat…
• Parent: Fruit	Child: Apple, Orange…
• Parent: Shape	Child: Triangle, Rectangle, Circle …

Why is Inheritance useful?

- Avoid redefining the information from the base class in our derived class.
    - If a `Student` and an `Instructor` are both derived class, we don’t need to write the same code twice
    - Define a Parent class, `Person`, that would hold any redundant information
- Not only saves work
    - If we update or modify the base class, all derived classes will automatically inherit the changes!

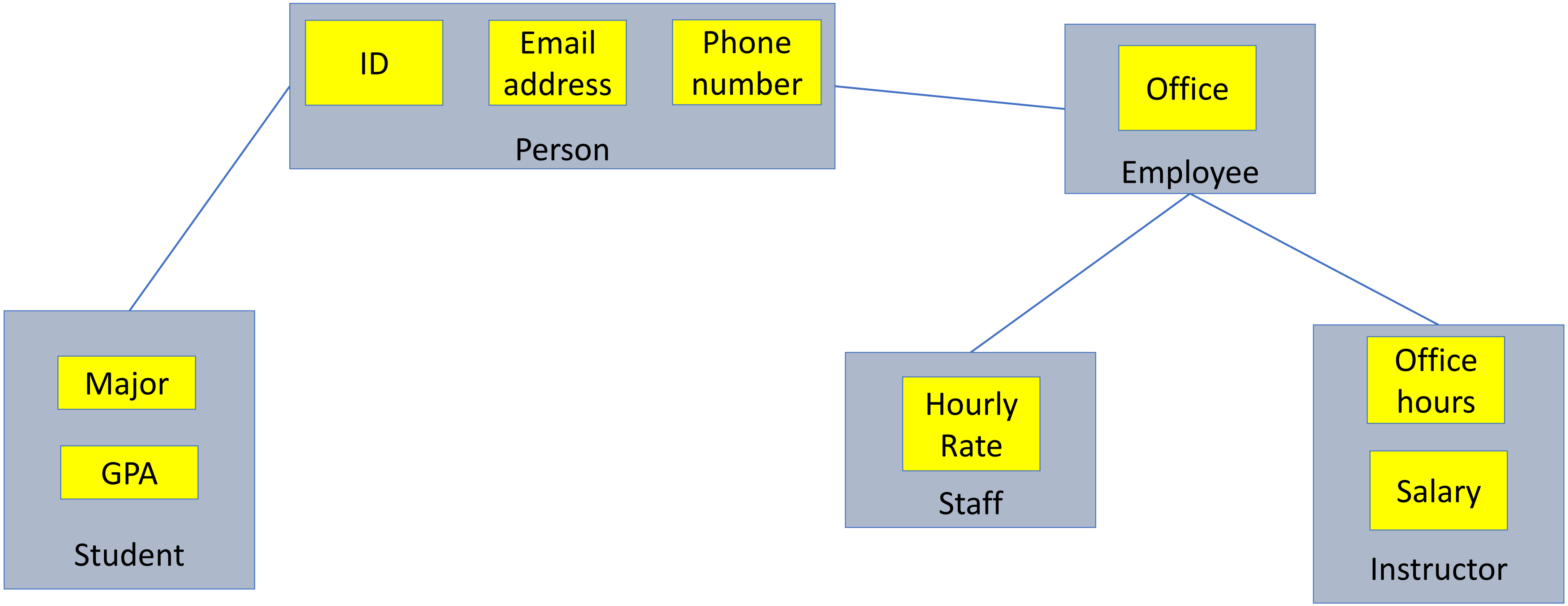
Let’s draw the hierarchy

- Student:
    - ID
    - Email address
    - Phone number
    - Major
    - GPA
- Instructor
    - ID
    - Email address
    - Phone number
    - Office
    - Office hours
    - Salary



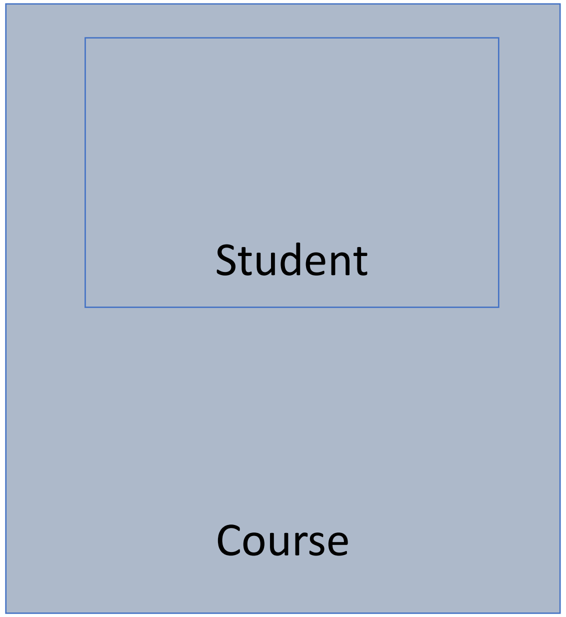
Inheritance (cont…)

- Inheritance is not limited to a single level
  - Let’s add an Employee class to the hierarchy

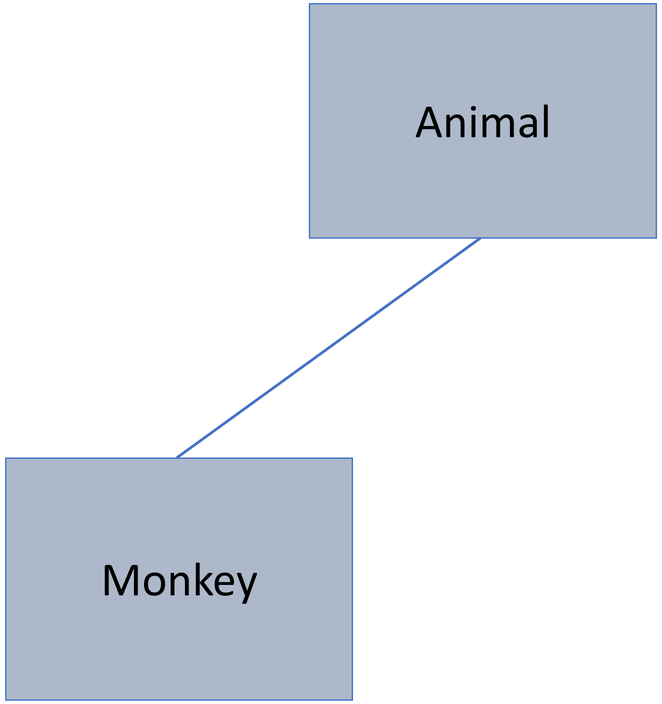


Inheritance vs. Composition

- Composition
  - Course “has a” Student



- Inheritance
  - Monkey “is a” Animal



Define Inheritance

- Parent class declared and defined as normal
- Child class:
  - class Derived:public Base {};
    - i.e. `class Monkey : public Animal {};`
  - List only member variables you want to add, not what is inherited
  - Only redeclare inherited member functions if you want to redefine them
    - When an inherited member function definition is changed in the derived class

- Derived classes can be used anywhere the base class would be used, but not the other way around
  - i.e. anywhere you use the Animal, you can use the Monkey, but not everywhere you use the Monkey can you use the Animal

These things are NOT Inherited:

- Base class constructor
  - Though it can be called from the derived class
  - `Child::Child():Parent(){}`
  - Base is called first to initialize all of the base member variables
  - If base constructor is not specified, the base default constructor will be used
- Copy Constructor
- Assignment Operator Overload
- Destructor

Interface (.h)

- Declare the Parent as normal
- The Child:

```
class Child : public Parent {
      private:
            //any members which are unique to the child
      public:
            Child(); //default constructor
            //other members including redefined functions from Parent
};
```

Implementation (.cpp)

- Parent class defined as normal
- Child:

```
Child::Child():Parent() {//child makes call to parent constructor first
             //initialize the member variables that are unique to child
}
//define all other member functions as normal
//redefining of parent functions follows the normal way of defining functions
```

Inheritance with the Big 3

- Recall: Big 3 are needed whenever there is dynamic memory or pointers, they are not inherited from the parent. To use in child successfully, they must be defined correctly in parent.

```
Child& Child::operator = (const Child& other) {
        Parent::operator = (other); //invoke parent class AOO
        //continue with things unique to child
}

Child::Child(const Child& copy):Parent(copy) {
        //continue with things unique to child
}

Child::~Child() {
        //define as normal, parent’s will be automatically called after the child’s completes
        //destructors go in the reverse of constructors calls
}
```

Inherited but Restricted

- Private member variables are inherited but cannot be accessed by name
  - Need to use accessor and mutator functions
- Private member functions are inherited but cannot be accessed by the derived class

- Recalled that we’ve seen two access specifiers:
  - `private, public`
- Now the 3rd one: **`protected`**
  - Allows for the derived class to be able to access things directly by name
  - Every other class would view them as private

Public vs. Private vs. Protected

- Anything public in the parent is public to the child
- Anything private in the parent is private to the child
  - This means the child cannot use private parent functions
  - This means the child cannot use private member variables of parent by name, have to use the inherited accessor and mutator functions
- Anything protected in the parent is public to the child but private to everyone else
  - This means the child can use protected member variables and functions of parent by name

Creation

- Base class Object:
  - i.e. Creating an Animal object: `Animal a1;`
  - `Animal` constructor is invoked, memory allocated for the base class

- Child class Object:
  - i.e. Creating an Monkey object: `Monkey m1;`
  - First, the `Animal` constructor is invoked, then the `Monkey` constructor invoked
  - memory allocated with enough space for the base class (`Animal`) and the derived class (`Monkey`)

Deletion

- Base Class Object:
  - i.e. Delete an Animal object: `a1`;
  - `Animal` destructor is invoked, memory deallocated for the base class

- Child Class Object:
  - i.e. Deleting an Monkey object: `m1`;
  - First, the `Monkey` destructor is invoked, then the `Animal` destructor invoked
  - **Note: Deletion has reverse order of creation**

More on Access Control

- Protected
  - `protected` in the parent is `public` to the child but `private` to everyone else

- Using protected access is a double-edged sword:
  - It can make it easier to implement classes by avoiding writing a public interface for some members.
  - *But*, it makes your derived classes vulnerable to changes to the protected members of the base class
    - Using a public interface can insulate you from the need to make changes in the derived classes.

More on Access Control

- Recall:
    - `class Monkey : public Animal { ... };`
    - This means that we used public inheritance.

- Use public inheritance to implement a true “is-a” relationship between objects

More on Access Control

- Public inheritance
  - private members of the base class are inaccessible in the derived class; protected members remain protected; and public members remain public

- Protected inheritance
  - private members of the base class are inaccessible in the derived class; protected members remain protected; **but public members become protected**

- Private inheritance
  - private members of the base class are inaccessible in the derived class; **protected and public members become private**