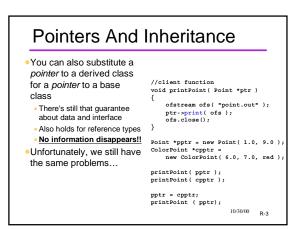
## CSE 143

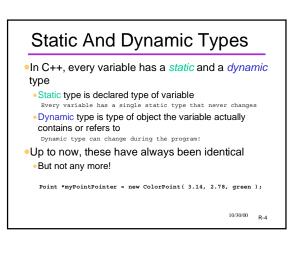
## Dynamic Dispatch and Virtual Functions

[Chapter 8 pp.354-370]

10/30/00 R-1

Substituting Derived Classes Recall that an instance of a derived class can always //client function (not a method) be substituted for an void printPoint ( Point pt ) instance of a base class pt.print( cout );
//the question: which print? Derived class guaranteed to have (at least) the same data 3 and interface as base class But you may not get the Point p( 1.0, 9.0 ); ColorPoint cp( 6.0, 7.0, red ); behaviour you want! printPoint( p ); p = cp; //information lost printPoint( p ); printPoint( cp ); 10/30/00 R-2





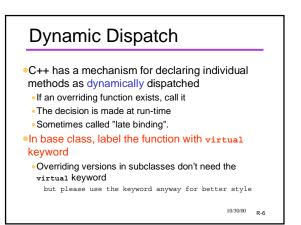
## "Dispatch"

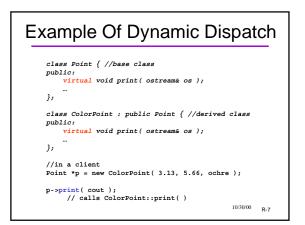
- "Dispatching" is the act of deciding which piece of code to execute when a method is called
- •Static dispatch means that the decision is made statically, *i.e.* at compile time
- Decision made based on static (declared) type of receiver

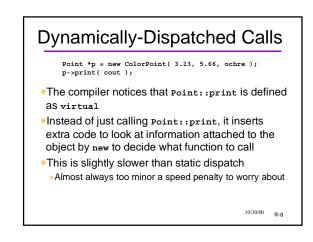
Point \*myPointPointer = new ColorPoint( 3.14, 2.78, green );

myPointPointer->print( cout );
 // myPointPointer is a Point\*, so call Point::print

10/30/00 R-5







## When Does This Happen?

- Dynamic dispatch ONLY happens when BOTH of these two conditions are met:
- 1. The object is accessed through a pointer (or reference) 2. The method is virtual
- •In ALL other cases, you get static dispatch
- •Some common cases
- Objects passed by pointer to a function
- An array of pointers to objects
- A pointer to a class as a member variable of another class (rather than the object itself)

10/30/00 R-9

# •An array of pointers to objects derived from the same base class:

- mammal \* zoo[20]; // An array of 20 pointers.
- All the objects pointed to are mammals, but some might be dogs, people, aardvarks, hedgehogs, etc.
- Each class might have its own methods for behavior like "scream" "fight" "laugh", etc.
- If I write zoo[i]->laugh() I want to get the appropriate behavior for that type of animal Won't happen unless laugh is virtual in mammal class

10/30/00 R-10

## Contrast

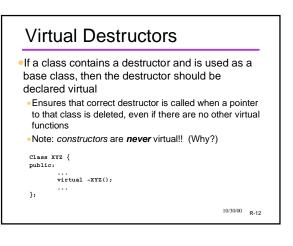
#### •mammal mlist[20];

- all array elements are of the same type
- Everything in the list is treated as a mammal, period regardless of whether methods are virtual or not

#### •mammal \* vmlist[20];

 Each critter behaves like "mammal" for the non-virtual functions, and like its own particular kind of mammal for the virtual methods.

<sup>10/30/00</sup> R-11



### Abstract vs Concrete Classes

- Some classes are so abstract that instances of them shouldn't even exist
- What does it mean to have an instance of widget? of pushbutton? Of Animal?
- It may not make sense to attempt to fully implement all functions in such a class
- What should pushbutton::clicked() do?
- An abstract class is one that should not or can not be instantiated - it only defines an interface
- declaration of public methods, partial implementation •A concrete class can have instances
  - 10/30/00 R-13

#### Abstract Class in C++ No special "abstract" keyword in C++ Are recognized by being classes with unimplementable methods "pure virtual functions" (next slide) Such a class is only intended to be used as a base class

10/30/00 R-14

## **Pure Virtual Functions**

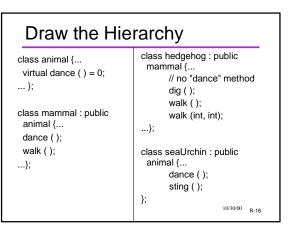
- •A "pure virtual" function is not implemented in the base class
- must implement in derived classes

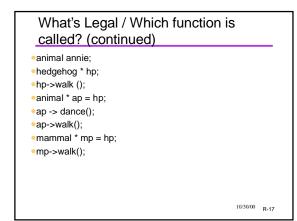
 Syntax: append "= 0" to base method declaration class pushbutton : public widget {
 public:

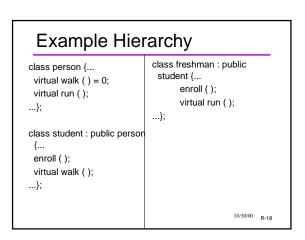
public: virtual void clicked() = 0; };

- Compiler guarantees that class with pure virtual functions cannot be instantiated
- •If you call a pure virtual function, you'll use the version from some derived class pushbutton \*b = new quitbutton; b->clicked();

10/30/00 R-15







#### What's Legal / Which function is called? (continued) • person paula; • student \*stu = new freshman(); • stu->enroll(); • student sara = \*stu; • sara.run(); • person \*pp = stu; • pp->run(); • pp->walk(); • freshman \*fred = pp; • fred->enroll();

10/30/00 R-19

#### Draw hierarchy & call graph START at plug::dispatch() class lir : public plug { public: virtual void boof() class plug { public: { biff(); } virtual void boof() } { bang(); } virtual void bang() class vop : public plug { { nalg(); } public: void dispatch() virtual void bang() { trog.boof(); } { whing(); } protected: protected: plug \*trog; int log; }; } <sup>10/30/00</sup> R-20