COMSC-200 Lecture Topic 12
Polymorphism

Reference
Deitel Ch.13

Working With Mixed Object Types
eliminate need for `switch` statements
getting base-class pointers...
  ...to call derived-class functions

Terminology
abstract base class (usually generic, like Animal)
multiple derived classes (usually specific, like Cat or Dog)
class heirarchy (parent-children relationships)
virtual functions and destructors
  pure virtual functions
"dynamic binding": associating a call with
  a particular function at runtime
"dynamic casting": converting pointers and
  reference variables at runtime

Without Polymorphism

```
class Animal
{
  ...
  void talk(){cout<<"???";}
};

class Cat : public Animal
{
  ...
  void talk(){cout<<"meow";}
};

  ...
  Animal a;
  Cat c;
  ...
  Animal* p = &c; // valid: inheritance
  ...
  a.talk(); // ???
  c.talk(); // meow
  p->talk(); // ???
```

Syntax For Polymorphism

```
class Animal
{
  ...
  virtual void talk(){cout<<"???";}
};

class Cat : public Animal
{
  ...
  void talk(){cout<<"meow";}
};

  ...
  Animal a;
  Cat c;
  ...
  Animal* p = &c; // valid: inheritance
```

The `virtual` Keyword
need not be repeated in derived class declarations
needed in declaration only -- *not definition*

```
class Animal
{
  ...
  virtual void talk();
};
...
void Animal::talk()
{
  cout<<"???";
}
```

virtual carries over to all child classes

Pure Virtual Functions
include `= 0;` in function declaration
  and provide no function definition

```
class Animal
{
  ...
  virtual void talk() = 0;
};
```

Abstract Classes
when class has one or more pure virtual functions
cannot instantiate:

```
  Animal a; // compiler error!
  Animal* p = new Animal; // compiler error!
```

Virtual Destructors
a good programming practice
  even if base class needs no destructor
enables this to call the right destructor

```
  Animal* p = new Cat;
  ...
  delete p;
```

syntax:

```
  virtual ~Animal(){}
```

Scope Resolution For Functions
same as before (see Inheritance)
in Circle class:
  `output();` calls Circle class' version
  `LocatableShape::output();` calls base class' version
can go anywhere up the heirarchy
exception, private functions: can rewrite but cannot invoke

How This Applies To Lab 12
lab 2 used a `void*` array and `switch`
lab 12 uses an array of base class pointers and *no* `switch`
  `vector<Shape*> shapes;`

```
...
a.talk(); // ???
c.talk(); // meow
p->talk(); // meow: polymorphism
```

the "virtual" keyword in a function's *declaration* makes it virtual

The "Polymorphism Tax"

`sizeof(Animal)` before, after the `virtual` keyword

- Yes, you can define a friend function inline.
- Yes, polymorphism applies to objects that are *not* dynamically created.