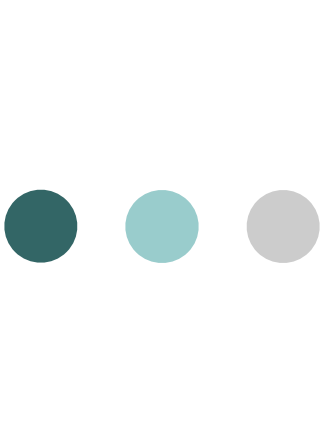

15.4

Redefining Base Class Functions

Redefining Base Class Functions

- Redefining function: function in a derived class that has the same name and parameter list as a function in the base class

- Typically used to replace a function in base class with different actions in derived class

Redefining Base Class Functions

- Not the same as overloading – with overloading, parameter lists must be different

- Objects of base class use base class version of function; objects of derived class use derived class version of function

Redefining Base Class Functions: Base Class

```
#include <iostream>
using namespace std;
// override a function

class Mammal
{   string mammalSound;
public:
    Mammal()
    {   mammalSound = "Meeeoowww";
    }
    void display()
    {   cout<< "\nI am a Mammal and I sound: "
            <<getMammalSound();
    }
    string getMammalSound() const { return mammalSound; }
};

```

Redefining Base Class Functions: Derived Class

```
19  class Lion : public Mammal
20  {
21      string lionSound;
22  public:
23      Lion() : Mammal()
24      {   lionSound = "Raawwrrrrr";
25      }
26      void display()
27      {   cout<< "\nI am a Lion and I sound: "
28              <<getLionSound();
29      }
30      string getLionSound() const { return lionSound; }
31  };
```

Redefining Base Class Functions

```
33  // using the derived class
34  int main()
35  {    Lion theKing;
36       theKing.display();
37       //theKing.Mammal::display();
38  }
39
40
41  /* Output of the program: */
42       I am a Lion and I sound: Raawwrrrrr
```

Problem with Redefining

- Consider this situation:
  - Class BaseClass defines
    - functions blue() and yellow()
    - blue() calls yellow()
  - Class DerivedClass inherits from BaseClass and redefines function yellow()
  - An object dr_obj of class DerivedClass is created and function blue() is called
  - When blue() is called then which yellow() function is used, the one defined in BaseClass or the redefined one in DerivedClass?

Problem with Redefining

BaseClass

```
void blue();
void yellow();
```

DerivedClass

```
void yellow();
```

```
DerivedClass dr_obj;
dr_obj.blue();
```

Object dr_obj invokes function blue() in BaseClass

Function blue() invokes function yellow() in BaseClass and not function yellow() in DerivedClass

Function calls are bound at compile time.
This phenomenon is called static binding.

Problem with Redefining

```
class Mammal
{   string mammalSound;
public:
    Mammal()
    {   mammalSound = "Meeeoowww";
    }
    void display()
    {   cout<< "\nI am a Mammal and I sound: "
            <<getSound();
    }
    string getSound() const { return mammalSound; }
};
```

```
class Lion : public Mammal
{
    string lionSound;
public:
    Lion() : Mammal()
    {   lionSound = "Raawwrrrrr";
    }
    string getSound() const { return lionSound; }
};

// using the derived class
int main()
{   Lion theKing;
    theKing.display();
}
```

15.6

Polymorphism and Virtual Member Functions

Polymorphism and Virtual Member Functions

- Virtual member function
  - function in base class that expects to be redefined in the derived class
- Function defined with key word virtual:
  - virtual void yellow() {…}
- Supports dynamic binding
  - functions bound at run time to function that they call
- Without virtual member functions, C++ uses static (compile time) binding

Virtual Functions

- A virtual function is dynamically bound to calls at runtime.
- At runtime, C++ determines the type of object making the call, and binds the object to the appropriate version of the function.

Virtual Functions (cont)

- To make a function virtual, place the virtual key word before the return type in the base class's declaration:

  virtual string getSound() const

- The compiler will not bind the function to calls. Instead, the program will bind them at runtime.

- The function also becomes virtual in all derived classes automatically!
  - We do not need to redefine it to be a virtual function in the derived version of the function

Virtual Functions: dynamic binding

```
class Mammal
{   string mammalSound;
public:
    Mammal()
    {   mammalSound = "Meeeoowww";
    }
    void display()
    {   cout<< "\nI am a Mammal and "
            <<"I sound: "<<getSound();
    }
    virtual string getSound() const
    { return mammalSound;
    }
};
```

```
class Lion : public Mammal
{
    string lionSound;
public:
    Lion() : Mammal()
    {   lionSound = "Raawwrrrrr";
    }
    string getSound() const { return lionSound; }
};

// using the derived class
int main()
{   Lion theKing;
    theKing.display();
}
```

Polymorphism: Base Class Object

- In object oriented programming, a derived class object can be assigned to the object of the base class

```
DerivedClass xDerived;
BaseClass xBase = xDerived; // legal
```

Polymorphism: Base Class Object

```
class Mammal
{   string mammalSound;
public:
    Mammal()
    {   mammalSound = "Meeeoowww";
    }
    void display()
    {   cout<< "\nI am a Mammal and "
            <<"I sound: "<<getSound();
    }
    virtual string getSound() const
    { return mammalSound;
    }
};
```

```
class Lion : public Mammal
{   string lionSound;
public:
    Lion() : Mammal()
    {   lionSound = "Raawwrrrrr";
    }
    string getSound() const
    { return lionSound;
    }
    void display()
    {   cout<< "\nI am a Lion and "
            <<"I sound: "<<getSound();
    }
};
```

Polymorphism: Base Class Object

```
51  // using the parent object
52  int main()
53  {    Lion theKing;
54       theKing.display();
55
56       // assiging the child to the parent object
57       Mammal * varMammal = &theKing;
58       varMammal->display();
59  }
60
61  /* Output of the program:
62       I am a Lion and I sound: Raawwrrrrr
63       I am a Mammal and I sound: Raawwrrrrr
64  */
```

Polymorphism: Base Class Object

- We can define a pointer to a base class object
  - and assign it the address of a derived class object
- Base class object knows about members of the base class
  - So, we can't use a base class object to call a function that is defined in the derived class
- Redefined functions in derived class will be ignored unless base class declares the function virtual