C++

Polymorphism

Systems Programming

C++ Polymorphism

- Polymorphism Examples
- Relationships Among Objects in an Inheritance Hierarchy
  - Invoking Base-Class Functions from Derived-Class Objects
  - Aiming Derived-Class Pointers at Base-Class Objects
  - Derived-Class Member-Function Calls via Base-Class Pointers
  - Virtual Functions
- Summary of the Allowed Assignments Between Base-Class and Derived-Class Objects and Pointers
- Type Fields and **switch** Statements
- Abstract Classes and Pure **virtual** Functions
- Polymorphism Case Study {No time for this!!}

24.1 Introduction

- Polymorphism with inheritance hierarchies
  - "Program in the general" vs. "program in the specific"
  - Process objects of classes that are part of the same hierarchy as if they are all objects of the base class.
  - Each function performs the correct tasks for that object's type
    - Different actions occur depending on the type of object.
  - New classes can be added with little or not modification to existing code.

Polymorphism Examples

Example `Animal` hierarchy

- `Animal` base class – every derived class has a function `move`.
- Different animal objects are maintained as a `vector` of `Animal` pointers.
- Program issues same message (`move`) to each animal generically.
- Proper function gets called
  - A `Fish` will `move` by swimming.
  - A `Frog` will `move` by jumping.
  - A `Bird` will `move` by flying.

24.2 Polymorphism Examples

Polymorphism occurs when a program invokes a `virtual` function through a base-class pointer or reference.

- C++ dynamically chooses the correct function for the class from which the object was instantiated.

Example: `SpaceObjects`

- Video game manipulates objects of types that inherit from `SpaceObject`, which contains member function `draw`.
- Function `draw` implemented appropriately for the different derived classes.
- A screen-manager program maintains a container of `SpaceObject` pointers.
- Call `draw` on each object using `SpaceObject` pointers
  - The proper `draw` function is called based on object's type.
- A new class derived from `SpaceObject` can be added without affecting the screen manager.

24.3 Relationships among Objects in an Inheritance Hierarchy

1. Aim base-class pointer at base-class object
   - Invoke base-class functionality
2. Aim derived-class pointer at derived-class object
   - Invoke derived-class functionality
3. Aim base-class pointer at derived-class object
   - Because derived-class object *is an* object of base class
   - Invoke base-class functionality

- **Invoked functionality depends on the type of the handle used to invoke the function, not on the type of the object to which the handle points.**
- **virtual** functions
  - Make it possible to invoke the object type's functionality, rather than invoke the handle type's functionality.

  *** This is crucial to implementing polymorphic behavior.**

Invoking Base-Class Functions from Derived-Class Objects

```
1  // Fig. 24.1: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
```

Invoking Base-Class Functions from Derived-Class Objects

```
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Function `earnings` will be redefined in derived classes to calculate the employee's earnings

Function `print` will be redefined in derived class to print the employee's information

Invoking Base-Class Functions from Derived-Class Objects

```
1  // Fig. 24.2: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14    setGrossSales( sales ); // validate and store gross sales
15    setCommissionRate( rate ); // validate and store commission rate
16 } // end CommissionEmployee constructor
17
18 // set first name
19 void CommissionEmployee::setFirstName( const string &first )
20 {
21    firstName = first; // should validate
22 } // end function setFirstName
23
24 // return first name
25 string CommissionEmployee::getFirstName() const
26 {
27    return firstName;
28 } // end function getFirstName
```

Invoking Base-Class Functions from Derived-Class Objects

```
29
30 // set last name
31 void CommissionEmployee::setLastName( const string &last )
32 {
33    lastName = last;  // should validate
34 } // end function setLastName
35
36 // return last name
37 string CommissionEmployee::getLastName() const
38 {
39    return lastName;
40 } // end function getLastName
41
42 // set social security number
43 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
44 {
45    socialSecurityNumber = ssn; // should validate
46 } // end function setSocialSecurityNumber
47
48 // return social security number
49 string CommissionEmployee::getSocialSecurityNumber() const
50 {
51    return socialSecurityNumber;
52 } // end function getSocialSecurityNumber
53
54 // set gross sales amount
55 void CommissionEmployee::setGrossSales( double sales )
56 {
57    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
58 } // end function setGrossSales
```

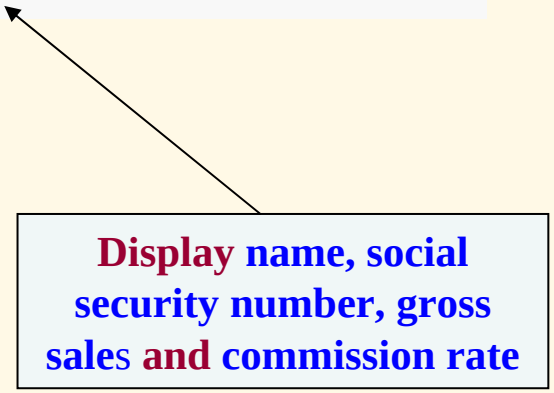
Invoking Base-Class Functions from Derived-Class Objects

```
59
60 // return gross sales amount
61 double CommissionEmployee::getGrossSales() const
62 {
63    return grossSales;
64 } // end function getGrossSales
65
66 // set commission rate
67 void CommissionEmployee::setCommissionRate( double rate )
68 {
69    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
70 } // end function setCommissionRate
71
72 // return commission rate
73 double CommissionEmployee::getCommissionRate() const
74 {
75    return commissionRate;
76 } // end function getCommissionRate
77
78 // calculate earnings
79 double CommissionEmployee::earnings() const
80 {
81    return getCommissionRate() * getGrossSales();
82 } // end function earnings
```

Calculate earnings based on commission rate and gross sales

Invoking Base-Class Functions from Derived-Class Objects

```
83
84 // print CommissionEmployee object
85 void CommissionEmployee::print() const
86 {
87    cout << "commission employee: "
88       << getFirstName() << ' ' << getLastName()
89       << "\nsocial security number: " << getSocialSecurityNumber()
90       << "\ngross sales: " << getGrossSales()
91       << "\ncommission rate: " << getCommissionRate();
92 } // end function print
```



Invoking Base-Class Functions from Derived-Class Objects

```
1  // Fig. 24.3: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

Redefine functions `earnings` and `print`

Invoking Base-Class Functions from Derived-Class Objects

```
1  // Fig. 24.4: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
19 // set base salary
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
25 // return base salary
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28    return baseSalary;
29 } // end function getBaseSalary
```

Invoking Base-Class Functions from Derived-Class Objects

```
30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    return getBaseSalary() + CommissionEmployee::earnings();
35 } // end function earnings
36
37 // print BasePlusCommissionEmployee object
38 void BasePlusCommissionEmployee::print() const
39 {
40    cout << "base-salaried ";
41
42    // invoke CommissionEmployee's print function
43    CommissionEmployee::print();
44
45    cout << "\nbase salary: " << getBaseSalary();
46 } // end function print
```

Redefined `earnings` function incorporates base salary

Redefined `print` function displays additional BasePlusCommissionEmployee details

Invoking Base-Class Functions from Derived-Class Objects

```
1  // Fig. 24.5: fig24_05.cpp
2  // Aiming base-class and derived-class pointers at base-class
3  // and derived-class objects, respectively.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10 using std::setprecision;
11
12 // include class definitions
13 #include "CommissionEmployee.h"
14 #include "BasePlusCommissionEmployee.h"
15
16 int main()
17 {
18    // create base-class object
19    CommissionEmployee commissionEmployee(
20       "Sue", "Jones", "222-22-2222", 10000, .06 );
21
22    // create base-class pointer
23    CommissionEmployee *commissionEmployeePtr = 0;
```

Utilizing pointers now!

Invoking Base-Class Functions from Derived-Class Objects

```
24
25    // create derived-class object
26    BasePlusCommissionEmployee basePlusCommissionEmployee(
27       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
28
29    // create derived-class pointer
30    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
31
32    // set floating-point output formatting
33    cout << fixed << setprecision( 2 );
34
35    // output objects commissionEmployee and basePlusCommissionEmployee
36    cout << "Print base-class and derived-class objects:\n\n";
37    commissionEmployee.print(); // invokes base-class pr
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // invokes derived-class prin
40
41    // aim base-class pointer at base-class object and print
42    commissionEmployeePtr = &commissionEmployee; // perfectly natural
43    cout << "\n\n\nCalling print with base-class pointer to "
44       << "\nbase-class object invokes base-class print function:\n\n";
45    commissionEmployeePtr->print(); // invokes base-class print
```

Utilizing pointers now!

Aiming base-class pointer at base-class object and invoking base-class functionality

Invoking Base-Class Functions from Derived-Class Objects

```
46
47    // aim derived-class pointer at derived-class object and print
48    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
49    cout << "\n\n\nCalling print with derived-class pointer to "
50       << "\nderived-class object invokes derived-class "
51       << "print function:\n\n";
52    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
53
54    // aim base-class pointer at derived-class object and print
55    commissionEmployeePtr = &basePlusCommissionEmployee;
56    cout << "\n\n\nCalling print with base-class pointer to "
57       << "derived-class object\ninvokes base-class print "
58       << "function on that derived-class object:\n\n";
59    commissionEmployeePtr->print(); // invokes base-class print
60    cout << endl;
61    return 0;
62 } // end main
```

Aiming derived-class pointer at derived-class object and invoking derived-class functionality

Aiming base-class pointer at derived-class object and invoking base-class functionality

Invoking Base-Class Functions from Derived-Class Objects

```
Print base-class and derived-class objects:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06
```

(Continued at top of next slide…)

Invoking Base-Class Functions from Derived-Class Objects

(…Continued from bottom of previous slide)

```
Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to derived-class object
invokes base-class print function on that derived-class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
```

24.3.2 Aiming Derived-Class Pointers at Base-Class Objects

- Aim a derived-class pointer at a base-class object.
  - C++ compiler generates error.
    - `CommissionEmployee` (base-class object) is not a `BasePlusCommissionEmployee` (derived-class object)
  - If this were to be allowed, programmer could then attempt to access derived-class members which do not exist.
    - Could modify memory being used for other data.

Aiming Derived-Class Pointers at Base-Class Objects

```
1  // Fig. 24.6: fig24_06.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aim derived-class pointer at base-class object
13    // Error: a CommissionEmployee is not a BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15    return 0;
16 } // end main
```

Cannot assign base-class object to derived-class pointer because *is-a* relationship does not apply

Borland C++ command-line compiler error messages:

```
Error E2034 Fig24_06\fig24_06.cpp 14: Cannot convert 'CommissionEmployee *'
   to 'BasePlusCommissionEmployee *' in function main()
```

GNU C++ compiler error messages:

```
Fig24_06.cpp:14: error: invalid conversion from `CommissionEmployee*' to
   `BasePlusCommissionEmployee*'
```

Microsoft Visual C++.NET compiler error messages:

```
C:\examples\ch24\Fig24_06\fig24_06.cpp(14) : error C2440:
   '=' : cannot convert from 'CommissionEmployee *__w64 ' to
   'BasePlusCommissionEmployee *'
         Cast from base to derived requires dynamic_cast or static_cast
```

24.3.3 Derived-Class Member-Function Calls via Base-Class Pointers

- Aiming base-class pointer at derived-class object.
  - Calling functions that exist in base class causes base-class functionality to be invoked.
  - Calling functions that do not exist in base class (may exist in derived class) will result in error.
    - Derived-class members cannot be accessed from base-class pointers.
    - However, this can be accomplished using downcasting (Section 13.8).

Aiming base-class pointer at derived-class object

```
1  // Fig. 24.7: fig24_07.cpp
2  // Attempting to invoke derived-class-only member functions
3  // through a base-class pointer.
4  #include "CommissionEmployee.h"
5  #include "BasePlusCommissionEmployee.h"
6
7  int main()
8  {
9     CommissionEmployee *commissionEmployeePtr = 0; // base class
10    BasePlusCommissionEmployee basePlusCommissionEmployee(
11       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // derived class
12
13    // aim base-class pointer at derived-class object
14    commissionEmployeePtr = &basePlusCommissionEmployee;
15
16    // invoke base-class member functions on derived-class
17    // object through base-class pointer
18    string firstName = commissionEmployeePtr->getFirstName();
19    string lastName = commissionEmployeePtr->getLastName();
20    string ssn = commissionEmployeePtr->getSocialSecurityNumber();
21    double grossSales = commissionEmployeePtr->getGrossSales();
22    double commissionRate = commissionEmployeePtr->getCommissionRate();
23
24    // attempt to invoke derived-class-only member functions
25    // on derived-class object through base-class pointer
26    double baseSalary = commissionEmployeePtr->getBaseSalary();
27    commissionEmployeePtr->setBaseSalary( 500 );
28    return 0;
29 } // end main
```

Cannot invoke derived-class-only members from base-class pointer

Borland C++ command-line compiler error messages:

```
Error E2316 Fig24_07\fig24_07.cpp 26: 'getBaseSalary' is not a member of
   'CommissionEmployee' in function main()
Error E2316 Fig24_07\fig24_07.cpp 27: 'setBaseSalary' is not a member of
   'CommissionEmployee' in function main()
```

Microsoft Visual C++.NET compiler error messages:

```
C:\examples\ch24\Fig24_07\fig24_07.cpp(26) : error C2039:
   'getBaseSalary' : is not a member of 'CommissionEmployee'
      C:\cpphtp5_examples\ch24\Fig24_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
C:\examples\ch24\Fig24_07\fig24_07.cpp(27) : error C2039:
   'setBaseSalary' : is not a member of 'CommissionEmployee'
      C:\examples\ch24\Fig24_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

GNU C++ compiler error messages:

```
Fig24_07.cpp:26: error: `getBaseSalary' undeclared (first use this function)
fig24_07.cpp:26: error: (Each undeclared identifier is reported only once for
   each function it appears in.)
Fig24_07.cpp:27: error: `setBaseSalary' undeclared (first use this function)
```

24.3.4 Virtual Functions

- Normally the handle determines which class's functionality to invoke.
- With virtual functions
  - The type of the object being pointed to, not the type of the handle, determines which version of a virtual function to invoke.
  - This allows a program to dynamically (at runtime rather than compile time) determine which function to use.
    - Referred to as dynamic binding or late binding.

24.3.4 Virtual Functions

- Declared by preceding the function's prototype with the keyword **virtual** in the base class.

Example

```
virtual void draw () const;
```

would appear in the base class **Shape**.

- If the program invokes a virtual function through a base-class pointer to a derived-class object (e.g., **shapePtr->draw()**), the program will choose the correct derived-class **draw** function dynamically based on the object type.
- Derived classes **override virtual functions** to enable **polymorphic behavior**.

24.3.4 Virtual Functions

- Once declared **virtual**, a function remains **virtual** all the way down the hierarchy.
- When a **virtual** function is called by referencing a specific object by name using the dot member-selection operator(e.g., **squareObject.draw()**), the function invocation is resolved at compile time.{This is **static binding** and this is **Not** polymorphic behavior!}
- Dynamic binding with **virtual** functions only occurs off pointer and reference handles.

Virtual Functions

```
1  // Fig. 24.8: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
```

Virtual Functions

```
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    virtual double earnings() const; // calculate earnings
31    virtual void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Declaring `earnings` and `print` as `virtual` allows them to be overridden, not redefined

Virtual Functions

```
1  // Fig. 24.9: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, do
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base sala
20
21    virtual double earnings() const; // calculate earnings
22    virtual void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

Functions `earnings` and `print` are already `virtual` – good practice to declare `virtual` even when overriding function

Virtual Functions

```
1  // Fig. 24.10: fig24_10.cpp
2  // Introducing polymorphism, virtual functions and dynamic binding.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // include class definitions
12 #include "CommissionEmployee.h"
13 #include "BasePlusCommissionEmployee.h"
14
15 int main()
16 {
17    // create base-class object
18    CommissionEmployee commissionEmployee(
19       "Sue", "Jones", "222-22-2222", 10000, .06 );
20
21    // create base-class pointer
22    CommissionEmployee *commissionEmployeePtr = 0;
23
24    // create derived-class object
25    BasePlusCommissionEmployee basePlusCommissionEmployee(
26       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
27
28    // create derived-class pointer
29    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
```

Virtual Functions

```
30
31    // set floating-point output formatting
32    cout << fixed << setprecision( 2 );
33
34    // output objects using static binding
35    cout << "Invoking print function on base-class and derived-class "
36       << "\nobjects with static binding\n\n";
37    commissionEmployee.print(); // static binding
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // static binding
40
41    // output objects using dynamic binding
42    cout << "\n\n\nInvoking print function on base-class and "
43       << "derived-class \nobjects with dynamic binding";
44
45    // aim base-class pointer at base-class object and print
46    commissionEmployeePtr = &commissionEmployee;
47    cout << "\n\nCalling virtual function print with base-clas
48       << "\nto base-class object invokes base-class "
49       << "print function:\n\n";
50    commissionEmployeePtr->print(); // invokes base-class print
```

Aiming base-class pointer at base-class object and invoking base-class functionality

Virtual Functions

```
51
52    // aim derived-class pointer at derived-class object and print
53    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
54    cout << "\n\nCalling virtual function print with derived-class "
55       << "pointer\nto derived-class object invokes derived-class "
56       << "print function:\n\n";
57    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
58
59    // aim base-class pointer at derived-class object and print
60    commissionEmployeePtr = &basePlusCommissionEmployee;
61    cout << "\n\nCalling virtual function print with base-class pointer "
62       << "\nto derived-class object invokes derived-class "
63       << "print function:\n\n";
64
65    // polymorphism; invokes BasePlusCommissionEmployee's print;
66    // base-class pointer to derived-class object
67    commissionEmployeePtr->print();
68    cout << endl;
69    return 0;
70 } // end main
```

Aiming derived-class pointer at derived-class object and invoking derived-class functionality

Aiming base-class pointer at derived-class object and invoking derived-class functionality via polymorphism and `virtual` functions

Virtual Functions

```
Invoking print function on base-class and derived-class
objects with static binding

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Invoking print function on base-class and derived-class
objects with dynamic binding

Calling virtual function print with base-class pointer
to base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

Calling virtual function print with derived-class pointer
to derived-class object invokes derived-class print function:
```


(Coninued at the top of next slide …)

Virtual Functions

(…Continued from the bottom of previous slide)

```
base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

Calling virtual function print with base-class pointer
to derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
```

Summarizing Allowed Assignments Between Base-Class and Derived-Class Objects and Pointers

- Four ways to aim base-class and derived-class pointers at base-class and derived-class objects
  - Aiming a base-class pointer at a base-class object
    - Is straightforward.
  - Aiming a derived-class pointer at a derived-class object
    - Is straightforward.
  - Aiming a base-class pointer at a derived-class object
    - Is safe, but can be used to invoke only member functions that base-class declares (unless downcasting is used).
    - Can achieve **polymorphism** with **virtual** functions
  - Aiming a derived-class pointer at a base-class object
    - Generates a compilation error.

24.4 Type Fields and switch Statements

- A switch statement can be used to determine the type of an object at runtime.
  - Include a type field as a data member in the base class.
  - This enables the programmer to invoke appropriate action for a particular object.
  - Causes problems
    - A type test may be forgotten.
    - May forget to add new types.

24.5 Abstract Classes and Pure virtual Functions

- Abstract classes
  - Classes from which the programmer never intends to instantiate any objects.
    - Incomplete—derived classes must define the "missing pieces".
    - Too generic to define real objects.
  - Normally used as base classes and called abstract base classes.
    - Provides an appropriate base class from which other classes can inherit.
- Classes used to instantiate objects are called concrete classes.
  - Must provide implementation for every member function they define.

Abstract Classes and Pure virtual Functions

- Pure virtual function:: A class is made abstract by declaring one or more of its `virtual` functions to be "pure" by placing "`= 0`" in its declaration.

Example

```
virtual void draw() const = 0;
```

  - "`= 0`" is known as a pure specifier.
  - Does not provide implementation.

Abstract Classes and Pure virtual Functions

- Every concrete derived class must override all base-class pure virtual functions with concrete implementations.
- If not overridden, the derived-class will also be abstract.
- Used when it does not make sense for base class to have an implementation of a function, but the programmer wants all concrete derived classes to implement the function.

Software Engineering Observation 24.8

- An **abstract class** defines a common public interface for the various classes in a class hierarchy.
- An **abstract class** contains one or more pure **virtual** functions that **concrete derived classes** must override.

Abstract Classes and Pure virtual Functions

- The abstract base class can be used to declare pointers and references that can refer to objects of any concrete class derived from the abstract class.
- Programs typically use such pointers and references to manipulate derived-class objects polymorphically.
- Polymorphism is particularly effective for implementing layered software systems.

Examples:

1. Reading or writing data from and to devices.
2. An iterator class that can traverse all the objects in a container.