

Advanced C++ Issues

Overloading, Inheritance, Virtual Functions, and Friends

Overloading

- ◆ Overloading is a good example of polymorphism
 - Taking on many forms

Function Overloading

- ◆ main function is overloaded
 - `int main ()`
 - `int main (int, char **)`

Function Overloading Example

```
class Node
    Node* next
    int value

Node* max (Node* item1, Node* item2)
    if ( item1->value > item2->value )
        return item1
    else
        return item2

int max (int x, int y)
    if (x > y)
        return x
    else
        return y
```

Function Overloading - File I/O

◆fstream file

- file.open (const char* filename)
- file.open (const char* filename,
openmode mode);

◆ openmode:

ios::in
ios::out
ios::ate
ios::app
ios::trunc
ios::binary

Elementary Data Structures

5

Operator Overloading

◆ You have already used operator overloading

■ Insertion operator

- ◆ cout << blah;
- ◆ outFile << blah;

■ Extraction operator

- ◆ cin >> blah;
- ◆ inFile >> blah;

◆ You can overload any C++ operator except ., .*, ::, and ?:

Elementary Data Structures

6

Operator Overloading

```
class Node:
```

```
private:
```

```
    Node* next
```

```
    int value
```

```
cout << *node OR outFile << *node
```

```
friend ostream&
```

```
operator<<(ostream& os, const Node& node) {
```

```
    os << node.value;
```

```
    return os;
```

```
}
```

Elementary Data Structures

7

Operator Overloading

```
class Node:
```

```
private:
```

```
    Node* next
```

```
    int value
```

```
if ( node1 == node2 ) //Compares addresses
```

```
if (*node1 == *node2 ) //Compares what you define
```

```
int Node::operator==(const Node& rightNode) const
```

```
if (value == rightNode.value) //or this.value
```

```
    return true
```

```
else
```

```
    return false
```

Elementary Data Structures

8

Inheritance

- ◆ Inheritance - the ability of a class to derive properties from previously defined classes

Why Use Inheritance

```
class Employee
private:
    string name
    string department
    Date hiringDate
public:
    void DisplayStats ()
    void setName (string)
    void setDept (string)
```

Bad solution because you duplicate code

```
class Manager
private:
    string name,
    string department
    Date hiringDate
    Employee list
    string level
public:
    void DisplayStats ()
    void setLevel ()
    void setName (string)
    void setDept (string)
```

Why Use Inheritance

```
class Employee
private:
    string name
    string department
    Date hiringDate
public:
    void DisplayStats ()
    void setName (string)
    void setDept (string)
```

```
class Manager
private:
    Employee* empInfo
    Employee list
    string level
public:
    void DisplayStats ()
    void setLevel
    void setName (string)
    void setDept (string)
```

Still not the best solution. A manager is an employee but this method does not infer that information

Inheritance

- ◆ Want to explicitly state that a manager is an employee. Not that a manager has employee characteristics
- ◆ This is accomplished with inheritance

```
class Manager : public Employee
private:
    Employee list
    string level
```

Inheritance

```
class Manager : public Employee
private:
    Employee list
    string level
```

- ◆ Manager is derived from employee
- ◆ Employee is a base class for manager
- ◆ Manager class has all member variables/functions for employee class as well as its own

Inheritance

```
class Manager : public Employee
private:
    Employee list
    string level
```

Manager \longrightarrow Employee

```
class DerivedClass : kind BaseClass
    kind = public, private, or protected
```

Inheritance does not imply access privileges

Inheritance

- ◆ Derived class inherits properties from its base class

```
Employee employee
Manager manager
Employee* ptr
ptr = &employee
ptr = &manager
```

Inheritance - Constructors

- ◆ Base constructor is called before derived constructor
- ◆ Information can be passed to the base constructor through the derived constructor

Inheritance - Constructors

```
class Employee
private:
    string name
    string department
    Date hiringDate
public:
    void DisplayStats ()
    void setName (string)
    void setDept (string)
    Employee(string setName,
              string setDept,
              Date* hDate);

class Manager : public Employee
private:
    Employee list
    string level
public:
    void DisplayStats ()
    void setLevel
    Manager(string setLevel
            string setName.
            string setDept,
            Date* hDate);
```

Elementary Data Structures

17

Inheritance - Constructors

Declaration:

```
Manager manager ( "entry level", "Bob Davidson",
                  "tire", hireDate );
```

Constructors:

```
Manager::Manager ( string setLevel, string, setName,
                  string setDept, Date& hireDate ) :
    Employee ( setName, setDept,
              hireDate )
```

```
Employee::Employee ( string setName, string setDept,
                    Date* hireDate )
```

Elementary Data Structures

18

Functions with the same name

- ◆ Both classes have a *displayStats* function

```
Employee employee
Manager manager
employee.displayStats()
manager.displayStats()
```

- ◆ Compiler determines which to call based on the variable type - static/early binding

Elementary Data Structures

19

Functions with the same name

```
void
Manager::displayStats ( ) {
    cout << level << endl;
    Employee::displayStats(); ←
}
```

Necessary to distinguish between the two functions

Elementary Data Structures

20

Functions with the same name

- ◆ Alternatively, to displayStats, you can overload the << operator

Add to both classes:

```
friend ostream& operator<<(ostream&, const Manager&)  
friend ostream& operator<<(ostream&, const Employee&)
```

```
friend ostream&  
operator<<(ostream& os, const Manager& manager)  
    os << manager.level << endl  
    os << (employee) manager  
    return os
```

Early/static Binding

- ◆ Early/static binding can lead to problems

```
Employee employee  
Manager manager  
Employee* empPtr = &emp;  
empPtr->displayStats() // Calls Employee displayStats  
empPtr = &manager  
empPtr->displayStats() // Still calls the Employee  
// displayStats even though  
// empPtr points to a variable  
// of type Manager
```

Early/static Binding

- ◆ The compiler determines the version of the method to invoke from the type of the pointer instead of what type of object the pointer is pointing to

Virtual Functions

- ◆ You can signal to the compiler that a derived class may redefine a function by making the function virtual
 - Add to Employee class:
virtual void displayStats()
- ◆ Invocation of the function is now determined at execution time - late/dynamic binding

Virtual Functions

- ◆ Make a function virtual when you want a derived class function to override a base class function
 - In the derived class `displayStats` you can call the base class `displayStats`

Friends

- ◆ Functions or classes
- ◆ Declaring a non-member function as a friend to a class allows that function to access all of the private and protected members of the class
- ◆ Input and output functions are often friend functions - they are not members of the class itself and are defined outside the class

Friends

- ◆ Example
 - Overloaded insertion and extractions operators are friend functions
 - Perform input and output operations outside of the class provides more flexibility