Review: Execution Order of Derived Class Constructor

1. Allocating space for the entire object to store base class plus derived class’s members.
2. Calling the base class’s constructor to initialize the base-class members.
3. Initializing the derived class members using the constructor initialization list.
4. Executing the body of the derived class constructor.

Review: Constructors Operation

```cpp
//constructor
AudioBook::AudioBook(string title_in, int year_in, double length_in): Book(title_in, year_in)
{ length = length_in; } //assign length in to length - step 3
//constructor
AudioBook::AudioBook(string title_in, int year_in, double length_in): length(length_in)
{ } //Book::Book() - step 2 (default)
//constructor
AudioBook::AudioBook(string title_in, int year_in, double length_in): length(length_in)
{ Book::Book(title_in, year_in); } //assign length in to length - step 3
//constructor
AudioBook::AudioBook(string title_in, int year_in, double length_in): Book::Book(title_in, year_in)
{ } //Book::Book() - step 2
//constructor
AudioBook::AudioBook(string title_in, int year_in): length(length_in)
{ Book::Book(string, int); } //assign length in to length - step 4
AudioBook a("JavaScript", 2008.2.5); //constructor
AudioBook a("JavaScript", 2008.2.5); //constructor
```

Review: Static Binding

- The type of **implicit parameter** determines which of the overriding functions to call.
- The type of **explicit parameter** determines which of the overloading functions to call.

```cpp
//class AudioBook:
public Book
AudioBook a(...);
//Book::f(int);
//AudioBook::f(string);
AudioBook b(...);
//AudioBook::f(int);
//AudioBook::f(string);
```
Static vs. Dynamic Binding

• C++ supports both static and dynamic binding for overriding functions calls.
• Dynamic binding only applies to virtual functions that are called by object pointers:

<table>
<thead>
<tr>
<th>Function calls</th>
<th>binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectPointer -&gt; virtualFunction</td>
<td>dynamic</td>
</tr>
<tr>
<td>Object -&gt; virtualFunction</td>
<td>static</td>
</tr>
<tr>
<td>ObjectPointer -&gt; nonVirtualFunction</td>
<td>static</td>
</tr>
</tbody>
</table>

Examples

class Book{
public:
  virtual double shipping (){
    void update(int);
  ...
}
class AudioBook : public Book{
public:
  double shipping();
  void update(double);
  ...
}

AudioBook b(...);
Book a=b; //ok
a.shipping();
//call ?, dynamic/static
Book* ap=&b;
ap->shipping();
//call ?, dynamic/static
ap->update(1995);
//call ?, dynamic/static

Virtual Function Overhead

• The space overhead of the virtual table (vtable):
  – At compile time, the constructor of an object containing virtual functions must initialize the virtual table, which is the table of pointers to its member functions.
• The speed overhead resulting from the dynamic binding process:
  – At run time, virtual functions are called using pointer indirection, which results in a few extra instructions per method invocation as compared to a non-virtual method invocation.
• Virtual functions whose resolution is only known at run-time cannot be inlined.

Dynamic Binding Provides Abstraction and Facilitates Code Reuse

• We want to implement the same rule for Book and AudioBook volume discount.
• We can implement 2 identical member functions for each of the 2 classes.
• Alternatively, we can implement only 1 function that takes Book* as the explicit parameter:

  double VolumeDiscount(int qty, Book* book) {
    double discount=0.0;
    if(qty > book->minOrder())
      discount+=0.1;
    if(qty*book->shipping() > book->shippingDis())
      discount+=0.1;
    return discount;
  }
  Declare minOrder(), shipping() and shippingDis() as virtual functions.
C++ vs. Java in Dynamic Binding

• C++ manipulates object itself for computation.
• Dynamic binding only applies to virtual functions on object pointers:
  AudioBook b(...);
  Book* ap=b;
ap->shipping();
  //call AudioBook::shipping()
• Java manipulates object reference for computation.
• All overriding functions have dynamic binding:
  Book a=new Book(...);
a.shipping();
  //call Book::shipping()
  Book b=new AudioBook("C++", 2010, 2.5);
b.shipping();
  //call AudioBook::shipping()

Function Objects

• Unlike functional languages, where functions are first-class and can be passed as parameters to a function and can be returned from a function, C++ functions can not be copied to an function argument or be returned from a function.

JavaScript examples:

function operate(operator,x,y){
  return operator(x,y);
}
function add(x,y){return x+y;}
var f=add;
document.writeln(operate(f,2,3)); //?
function addx(x){
  return functionaddy(y){
    return x+y
  }
}
document.writeln(addx(2)(3)); //?

Function Pointers

• How, you can use function pointers to make function first-class.
  • Declare fp as a pointer to a function that takes 2 int arguments and returns an int value.
    int (*fp)(int,int);
    //global variable
    int add(int x, int y){return x+y;}
    int operate(int (*fp)(int,int), int x, int y){return (*fp)(x,y);}
    int main()
    {
      fp=add;
      cout << operate(fp, 2,3) << endl; //?
      cout << operate(add, 2,3) << endl; //?
    }