Inheritance

• Inheritance is a mechanism for
  – defining new class types to be a
    – specialization
    – generalization
      of existing class types
  – specifying hierarchical relationships between
    class types
  – Terminology: “the child (derived or subclass) class type
    inherits (or is derived from) the parent (base or
    superclass) class type.”

Derived vs. Base Class

• Derived class adds specification to base class by
  – Changing (override) base class operations that do not apply to the derived class.
• Derived class adds generalization to base class by
  – Adding new operations/data to extend the derived class characteristics.

Inherited & Non-inherited Features

• Derived class inherits
  – all data members
  – member functions defined in the base class.
• However, derived class has no access to the inherited
  private members in the base class.
• Derived class does not inherit:
  – its base class’s constructors and destructor
  – its base class’s assignment operator
  – its base class’s friends
Access Right of Derived Classes

<table>
<thead>
<tr>
<th>Access</th>
<th>Derived Class</th>
<th>Base Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>public</td>
<td>public</td>
</tr>
<tr>
<td>private</td>
<td>private</td>
<td>public</td>
</tr>
<tr>
<td>protected</td>
<td>protected</td>
<td>protected</td>
</tr>
</tbody>
</table>

Type of Inheritance

Derived Class Constructors

```cpp
//default constructor
Book::Book():year(0){}
//copy constructor
Book::Book(const Book& other): title(other.title), year(other.year){}
//constructor
Book::Book(string title_in, int year_in):
  title(title_in), year(year_in){}
//destructor
Book::~Book(){}
```

```cpp
//default constructor
AudioBook::AudioBook():Book()
//length(0.0){}
//copy constructor
AudioBook::AudioBook(const AudioBook& other):Book(other), length(other.length){}
//constructor
AudioBook::AudioBook(string title_in, int year_in, double length_in):Book
  (title_in, year_in), length(length_in){}
//destructor
AudioBook::~AudioBook(){}
```

Execution Order of Derived Class Constructor & Destructor

- Allocating space for the entire object to store base class plus derived class's members.
- Calling the base class's constructor to initialize the base-class members.
- Initializing the derived class members using the constructor initialization list.
- Executing the body of the derived class constructor.
- When an object is destroyed (goes out of scope or is deleted) the derived class's destructor is called on the object first;
- Then the base class's destructor is called on the object;
- Finally the allocated space for the full object is reclaimed;

Derived Class Assignment Operator

```cpp
Books::Book& operator=(const Book& rhs){
  //check for self-assignment
  if(this != rhs)
    {
      title=rhs.title;
      year=rhs.year;
    }
  return *this;
}
```

```cpp
AudioBooks::AudioBook& operator=(const AudioBook& rhs){
  //check for self-assignment
  if(this != rhs)
    {
      Book::operator=(rhs);
      length=rhs.length;
    }
  return *this;
}
```
Derived Class Added Functions

- Derived Class can add new member functions:
  ```cpp
double AudioBook::length_v();
```
- Derived Class can override functions in the base class:
  ```cpp
double Book::shipping();
```
- Derived Class can overload functions in the Base class:
  ```cpp
void Book::update(string);  // ok, information lost
void Book::update(int);
void AudioBook::update(double);
```

Statically Binding

- Both Overloading and overriding functions have the same name.
- Depending on the class type of the object that the function is invoked on (as explicit or implicit parameter), compiler can decide which function to call – static binding.

Exercise

```cpp
int main()
{
    Book a("C", 1990);
    AudioBook b;
    AudioBook c("JavaScript", 2008, 1.5);
    AudioBook d(c);
    a=c; // ok, information lost
    a.update(1990);
    b=a; //error
    c.update("C++");
    a.shipping();
    b.sipping();
    d.update(3.5);
}
```

Polymorphism

- In programming languages, polymorphism refers to objects of different type are handled using the same interface (function).
- Function overloading:
  - Compile time type resolution (static binding).
- Generic functions (parametric polymorphism):
  - Compile time type resolution (static binding).
- Inheritance (subtype polymorphism):
  - Compile time type resolution (static binding)
  - Run-time type resolution (dynamic binding).
Dynamic Binding

- Dynamic binding: the choice of which of the overriding functions to call is deferred until runtime.
- C++ implementation:
  - Base class object pointer (*) can store derived class class pointer.
    ```cpp
    AudioBook a(...);
    Book* bp=a;
    ```
- The function for dynamic binding in the base class is declared as virtual;
  ```cpp
  virtual double Book::shipping() {return 5.0;}
  ```
- Derived class also defines the function.
  ```cpp
  AudioBook::shipping() {return 3.0;}
  ```
- Calling virtual function:
  ```cpp
  double cost = bp->shipping();
  //AudioBook::shipping() is called
  ```

Why Bother? 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 function parameter:
  ```cpp
  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;
  }
  ```
  ```cpp
  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;
  }
  ```
- minOrder(), shipping() and shippingDis() are dynamically bound depending on the book type.