The Big Three

C++ Object Oriented Programming
Pei-yih Ting
93/04 NTOU CS

Contents
- Destructor
- Copy constructor
- Assignment operator
- The managed pointer

Introduction
- When the class has the functionality of resource management, it is very likely that the destructor (dtor), the copy constructor (copy ctor), and the assignment operator occur together.
- Resource management: ex.

```cpp
class Account {
    public:
        Account(const char *name, const char *phone, const char *address);
        ~Account();
    private:
        char *m_name;
        char *m_phone;
        char *m_address;
    };

Account::Account(const char *name, const char *phone, const char *address) {
    m_name = new char[strlen(name)+1]; strcpy(m_name, name);
    m_phone = new char[strlen(phone)+1]; strcpy(m_phone, phone);
    m_address = new char[strlen(address)+1]; strcpy(m_address, address);
}

Account::~Account() {
    delete[] m_name; delete[] m_phone; delete[] m_address;
}
```

Called the BIG 3
dtor
called the BIG 3

Copy Constructor (copy ctor)
- What is a copy constructor? X(X&)
  Account(Account &src);
- When is the copy constructor invoked? when the object is copied
  Case 1: Account customer1("Sean Pan", "123-4567890", "1234 Sunset Blvd.");
  Account customer2(customer1);
  Account customer3 = customer1;
  Case 2: void fun1(Account customer) {
  …
  }
  Case 3: Account fun2() {
  Account x;
  …
  return x;
  }
Copy Constructor

- If you do not define the copy constructor, the compiler will synthesize one for your class. This copy constructor copies all the bits in the object to initiate the new object. For many cases this implementation does the right thing, but for a class which allocates memory or handles other resources itself, this usually leads to errors.

```
customer 1
  m_name
  m_phone
  m_address

shallow copy
```

```
customer 2
  m_name
  m_phone
  m_address
```

Is this really we want?

```
"Sean Pan"
"123-4567890"
"1234 Sunset Blvd."
```

Problems: Dangling Reference

- Consider the following function call

  ```
  void fun(Account customerLocal) {
      ...
  } // the dtor would deallocate the memory belongs to customerLocal
  // however, these memory blocks are the same as those of customer
  
  void main() {
      Account customer("Sean Pan", "123-4567890", "1234 sunset Blvd.");
      ...
      fun(customer);
      ...
      customer.display(); // show all the customer information
  }
  ```

- At the above line, the statement `fun(customer)` would cause dangling reference and the statement `customer.display()` would access memory blocks previously belonged to this customer object and display some strange contents.

Problems: Unexpected Release

- Sometimes, the resource might be unexpectedly released, ex.

  ```
  void readFile(ifstream is) {
      ...
  }
  
  void main() {
      ifstream infile("input.dat");
      readFile(infile);
      ...
  }
  ```

- This is a complicated problem. The program will have runtime error. Why does the error occurs? You won't be able to correct this by supplying a copy constructor for ifstream because it is a library class. The only thing you can easily do is not invoking the copy ctor by passing the parameter with reference.

Example Copy Constructor

```c
Account(Account &src) {
    m_name = new char[strlen(src.m_name)+1];
    strcpy(m_name, src.m_name);
    m_phone = new char[strlen(src.m_phone)+1];
    strcpy(m_phone, src.m_phone);
    m_address = new char[strlen(src.m_address)+1];
    strcpy(m_address, src.m_address);
}
```
Member Object and Base Class

- Copy constructor is a constructor, member objects and base class must be initialized through initialization list
- For example:
  ```cpp
class Derived: public Base
{
    public:
      ...
      Derived(Derived &src);
      ...
    private:
      Component m_obj;
    }
    Derived::Derived(Derived &src): Base(src), m_obj(src.m_obj) {
      ...
    }
```  

Assignment Operator

- Where is the assignment operator invoked?
  ```cpp
  Account customer1("abc", "1234", "ABC street");
  Account customer2, customer3;  // assume default ctor defined
  customer2 = customer1;
  customer2.operator=(customer1);
  customer3 = customer2 = customer1;
  ```
- Note: `Account customer2 = customer1;` does not invoke the assignment operator
- What is its prototypes?
  ```cpp
  Account &operator=(Account &rhs);
  ```
- Designed for continuously assignment
  ```cpp
  customer3.operator=(customer2.operator=(customer1));
  ```

Assignment Operator

- You can declare the assignment operator in the private section to prevent public usage of the assignment semantics.
- If there is a reference variable or a const variable defined in the class, there is no way to define the assignment operator.
- Usually, the assignment operator repeats the codes both in the copy ctor and the dtor. It is common to prepare common functions to be called in assignment operator, copy ctor and the dtor.
- Again, three make a team. Do not forget any one of them.
Managed Pointer

1. Standard template class auto_ptr<T>:
   include <memory>
   auto_ptr<Fred> acts like a Fred* except that it owns the referent
   (the Fred object)

2. You can declare a managed pointer with NULL value initially
   auto_ptr<Fred> ptr;

3. You can invoke the assignment operator later
   ptr = auto_ptr(new Fred());

4. You can also construct a pointer with
   auto_ptr<Fred> ptr(new Fred()); or auto_ptr<Fred> ptr=new Fred();

5. This object can be used anywhere like a Fred* pointer.
   ptr->services();
   *ptr.services();
   Fred *ptrRaw = ptr.get();

6. When this object goes out of scope, the
   destructor will delete the owned Fred object.

7. What about an explicit delete?
   delete ptr; // syntax error

8. If you copy the managed pointer from another managed pointer
   without ownership to the real object, the new managed pointer
   does not have ownership to the real object. If you construct a
   new managed pointer with a raw pointer twice, both objects have
   ownership. Fortunately, delete in its destructor will only succeed once.
   But using a pointer without ownership to the real object is likely
   to be a dangling reference like a raw pointer.