The Senmantics of Constructors

The Senmantics of Constructors

Default Constructor

Member Class Object has Default Constructor

- 一个class中没有constructor，但是其中的member object有一个default constructor；编译器会为该class合成一个default constructor；但是会推迟到该constructor被调用时发生

```
1   class Foo{
2       public:
3               Foo();
4               Foo(int);
5       //...
6   };
7   class Bar{
8       public:
9               Foo foo;
10              char*str;
11      //...
12  };
13
14  void foo_bar() {
15      Bar bar;        // 编译器合成一个default constructor
16
17      if(str){}
18  }
```

- 例如类A包含两个数据成员对象，分别为：`string str`和`char *Cstr`，那么编译器生成的默认构造函数将只提供对string类型成员的初始化，而不会提供对char*类型的初始化。
- 假如类X的设计者为X定义了默认的构造函数来完成对str的初始化，形如：`A::A() {Cstr="hello"};`因为默认构造函数已经定义，编译器将不能再生成一个默认构造函数。但是编译器将会扩充程序员定义的默认构造函数——在最前面插入对初始化str的代码。若有多个定义有默认构造函数的成员对象，那么这些成员对象的默认构造函数的调用将依据声明顺序排列。

Base class has Default Constructor

- 如果一个没有定义任何构造函数的类**派生自带有默认构造函数的基类**，那么编译器为它定义的默认构造函数，将按照声明顺序为之依次调用其基类的默认构造函数。若该类没有定义默认构造函数而定义了多个其他构造函数，那么**编译器扩充它的所有构造函数——加入必要的基类默认构造函数**。另外，编译器会将**基类的默认构造函数代码加在对象成员的默认构造函数代码之前**。

Base class has Default Constructor

1. class声明或继承一个virtual function
2. class派生自一个继承串链

- 一个virtual function table(**vtbl**)会被编译器产生，放置class的虚函数地址
- 在每个class object中，存在一个指向vtbl的指针**vfptr**

Class has a Virtual Base Class

- 在这种情况下，编译器要将虚基类在类中的位置准备妥当，提供支持虚基类的机制。也就是说要在所有构造函数中加入实现前述功能的的代码。没有构造函数将合成以完成上述工作。
- 在每个class object中加入**虚基表指针(vbptr)**，指向虚继承的class

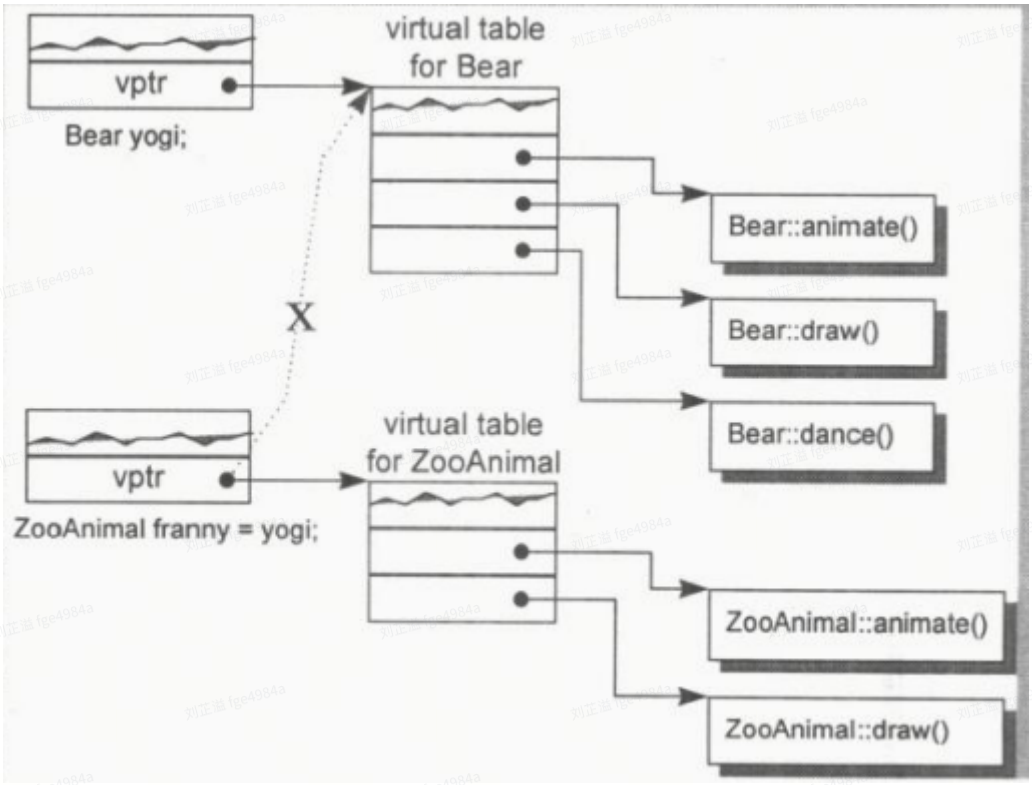
Copy Constructor

Default memberwise initialization

- 把某个object拷贝一份到另一个object，对member class object使用递归memberwise initialization

Don't Bitwise Copy Semantics

- class中含有copy constructor的member class object
- class继承自一个有copy constructor的base class
- class声明了一个或多个virtual functions(**使用位逐次拷贝，会使得vptr乱掉**)
- class派生自一个继承串链，有一个或多个virtual base classes



命名返回值优化(NRV)

- 对于一个如 `foo()` 这样的函数，它的每一个返回分支都返回相同的对象，编译器有可能对其做Named return Value优化（下文都简称NRV优化），**方法是以引用的方式传入一个参数 `result` 取代返回对象。(返回值将作为一个额外的参数提供给函数，来传回函数内部的值)**

```
X foo() //原型
{
    X xx;
    if(...)
        returnxx;
    else
        returnxx;
}
// NRV优化
void  foo(X &result)
{
    result.X::X();
    if(...)
    {
        //直接处理result
        return;
    }
    else
    {
        //直接处理result
        return;
```

```
22        }
23    }
```

- 对比优化前与优化后的代码可以看出，对于一句类似于 `X a = foo()` 这样的代码，NRV优化后的代码相较于原代码节省了一个临时对象的空间（省略了xx）,同时减少了两次函数调用（减少xx对象的默认构造函数和析构函数，以及一次拷贝构造函数的调用，增加了一次对X的默认构造函数的调用）。
- 另外，有一点要注意的是，NRV优化，有可能带来程序员并不想要的结果，最明显的一个就是——当你的类依赖于构造函数或拷贝构造函数，甚至析构函数的调用次数的时候，想想那会发生什么。由此可见、Lippman的cfront对NRV优化抱有更谨慎的态度，而MS显然是更大胆。

Member Initialization List

有四种情况必须用到初始化列表：

前两者因为要求定义时初始化，所以必须明确的在初始化队列中给它们提供初值。后两者因为不提供默认构造函数，所有必须显示的调用它们的带参构造函数来定义即初始化它们。

- 有const成员
- 有引用类型成员
- 成员对象没有默认构造函数
- 基类对象没有默认构造函数

总的来说：编辑器会对initialzation list一一处理并可能重新排列，以反映出members的声明顺序，它会安插一些代码到 constructor 体内，并置于任何 explicit user code之前。