Copy Elision

----Necokeine

Test Class

```
class TestClass {
 public:
  TestClass();                                          // Default Constractor
  TestClass(const TestClass& other);                    // Copy Constractor
  TestClass(TestClass&& other);                         // Move Constractor in C11
  TestClass& operator=(const TestClass& other);         // Copy Assignment
  TestClass& operator=(TestClass&& other);              // Move Assignment in C11
};
```

Example

```
Foo a                  // default constructor
Foo b(a)               // copy constructor
Foo c = b              // copy constructor
a = c                  // copy assignment

void func(Foo a);
func(a);               // copy constructor

Foo Build();
Foo a = Build();       // copy constructor
Foo b
b = Build();           // copy assignment
```

RVO

- RVO在return中直接构造返回值

```
TestClass func(int argu) {
    return TestClass(argu);                    // 在 return 中直接构建实体。
}
```

NRVO

- NRVO：所有return中都返回同一个预先定义的变量。

```
TestClass func(int argu) {
    TestClass result;                // 定义最终的返回变量。
    result.initialize(argu);
    ….
    return return;                   // 直接返回之前定义的变量。
}
```

反例写法（RVO and NRVO）

```
TestClass func(int argu) {
    TestClass resultA, resultB;                    // 有多个初始化形成的变量。
    resultA.initialize(argu / 2);
    resultB.initialize(argu - 1);
    ….
    if (Hash(argu) % 2) {
        return resultA;                            // 多句return，返回不同变量。
    } else {
        return resultB;
    }
}
```

写在return value 上的std::move

```
TestClass func(int argu) {

    TestClass result;

    …

    return std::move(result);                    // 这里的move是无意义的。
}
```

写在function定义上的&

```
TestClass& func(int argu) {                    // & 没意义
    TestClass result;
    …
    return result;
}
```

Copy Elision

```
class Widget {
 public:
  …

 private:
  string name_;
};
```

```
explicit Widget(const string& name) :
name_(name) {}    // 第一种写法。

explicit Widget(string name) :
name_(std::move(name)) {}    // 第二种
```

哪一种比较优秀？

分类讨论

- 使用右值调用函数：
- Widget widget(StrCat(bar, baz));　　// 使用值传递可以触发copy elision
- 使用左值调用函数：
- string local_str;
  Widget widget(local_str);　　// 都需要触发copy
- 如果是setter当中，那么应该用const reference

最后

- Readability 永远都优先复制或者其他细节上的优化。
- passing by const reference is simpler and safer, so it's still a good default choice.

Thanks

Hint

- You can't control whether NRVO happens or not. Compiler versions and options can change this from underneath you. Do not depend on it for correctness.
- You can't control whether returning a local variable triggers an implicit move or not. The type you use might be updated in the future to support move operations. Moreover, future language standards will apply implicit moves on even more situations so you can't assume that just because it is not happening now it won't happen in the future.