1. NRVO and RVO

#include <iostream>

using namespace std;

class foo {

public:

foo() { cout <<"foo::foo()\n"; }

foo( const foo& ){ cout << "foo::foo( const foo& )\n"; }

~foo(){ cout << "foo::~foo()\n"; }

};

foo bar(){ foo local\_foo; return local\_foo; }

int main()

{

foo f = bar();

}

Without return value optimization the function would have to construct

a temporary object (which will be used as the return value) and then the

caller of that function would have to make a copy constructor call in

order to copy the contents of that temporary object to the destination

object.

With return value optimization the return value is constructed

directly onto the destination object. No temporary object is created nor

the copy constructor is called.

This has obvious advantages if making a copy constructor call is a

heavy operation (eg. if the object is an enormous deep-copying array),

plus it saves memory (during the copy constructor call there has to be

two objects taking that much memory).

(Note, however, that you can't \*trust\* that RVO is always used. There

are certain situations where the compiler cannot do it and the temporary

object is created and the copy constructor called.)

NRVO расшифровывается как named return value optimization, оптимизация именованного возращаемого значения. NRVO применяется в случаях чуть отличных от предыдущего.

std::string foo()  
{  
 std::string a('a',1000000);  
 return a;  
}

Тут Стандарт позволяет компилятору не создавать локальный объект, хотя не требует такого поведения (12.8/15).

При NRVO не просто не происходит вызова конструктора копирования, здесь даже не будет побитового присваивания.

RVO doesn’t apply to function arguments!

*in a return statement in a function with a class return type, when the expression is the name of a non-volatile automatic object (other than a function or catch-clause parameter) with the same cv-unqualified type as the function return type, the copy/move operation can be omitted by constructing the automatic object directly into the function’s return value*

The price of moving is lower than copying but higher than RVO, so never apply std::move to local objects if they would otherwise be eligible for the RVO.

### **-fno-elide-constructors**

Depending on the quality of your compiler, you may note some “missing copies” relating to our calls of copy() and copy2(). We (humans) can see that those functions do nothing: they just copy a value unmodified from input to output. If a compiler is smart enough to notice that, it is allowed to eliminate the calls to the copy constructor. In other words, a compiler is allowed to assume that a copy constructor copies and does nothing but copy. Some compilers are smart enough to eliminate many spurious copies. However, compilers are not guaranteed to be that smart, so if you want portable performance, consider move operations (§18.3.4).