Subject: Re: Should the pick semantics be changed?
Posted by mirek on Fri, 07 Mar 2014 08:00:50 GMT
View Forum Message <> Reply to Message

OK, I was playing a bit with &&:

```
#include <Core/Core.h>

using namespace Upp;

struct Foo {
 int a;

 Foo(Foo&& x)            { LOG("MOVE"); a = x.a; }
 void operator=(Foo&& x) { LOG("MOVE="); a = x.a; }

 Foo()       {}
 Foo(const Foo& x, int) { LOG("COPY"); a = x.a; }
};

template <class T>
T&& pick(T& a) { return static_cast<T&&>(a); }

template <class T>
T clone(const T& a) { T c(a, 1); return c; }

struct Bar {
 Foo a, b;

 Bar(Bar&&) = default;
 Bar& operator=(Bar&&) = default;
 Bar() = default;
 Bar(const Bar& b, int) : a(b.a, 1), b(b.b, 1) {}
};

Bar test_bar()
{
 LOG("In FN");
 Bar x, y;
 x.a.a = 123;
 y.a.a = 421;
 return Random(2) ? pick(x) : pick(y); // This is somewhat unexpected and ugly...
}

Foo test() {
 Foo h;
 h.a = 22;
```

```
 return h;
}

CONSOLE_APP_MAIN
{
 {
  Foo y;
  Foo z = pick(y);
  LOG("----------");
  Foo x = clone(z);
  LOG("----------");
  x = pick(y);
  LOG("----------");
  y = clone(x);
  LOG("----------");
  x = test();
 }
 LOG("======================");
 {
  Bar y;
  Bar z = pick(y);
  LOG("----------");
  Bar x = clone(z);
  LOG("----------");
  x = pick(y);
  LOG("----------");
  y = clone(x);
  LOG("----------");
  x = test_bar();
 }
}
```

I think this route is actually superior to what we have now, with all operations being stated directly where it matters.

Interestingly, 'clone' would work even with C++03 fine and is clearly superior to Foo x(y, 1); notation, perhaps even to operator<<=...

What sort of surprised me is the need to use 'pick' in test_bar. IMO, this makes that original purpose of && sort of moot...

Mirek