Subject: moveable question
Posted by mtdew3q on Wed, 28 Nov 2012 03:56:31 GMT
View Forum Message <> Reply to Message

Hi-
In the following sample I thought this would break the moveable assertion. I used a class instead of a struct. I hooked up ultimate++ tonight and I wanted to see how moveable worked.

Please see my one in-line comment.

thanks for any assistance. - jim

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

using namespace Upp;
using namespace std;

class SimpleVector: Moveable<SimpleVector>{
    UPP::Vector<int*> v;
    int * val;
    int a;
    SimpleVector * sv;

public:
   SimpleVector();

SimpleVector& operator=( const SimpleVector& rhs );

    ~SimpleVector();

    void TestMove();
};

SimpleVector::SimpleVector(){
  int  n = 2;
 int * y = &n;


  v.Add(y);
 cout<<*v[0];
}

SimpleVector::~SimpleVector(){

        AssertMoveable<SimpleVector>();
}

void SimpleVector::TestMove(){

 val = &a;
 sv = this;
 // thought these 2 would break moveable assertion per the example in docs ??

}
SimpleVector& SimpleVector::operator=( const SimpleVector& rhs ){

sv= rhs.sv;
     return *this;
}
CONSOLE_APP_MAIN
{


 SimpleVector s;
 s.TestMove();




}
```

---

Subject: Re: moveable question
Posted by dolik.rce on Wed, 28 Nov 2012 07:04:09 GMT
View Forum Message <> Reply to Message

Hi,

First let me comment on you constructor, see the comments:SimpleVector::SimpleVector(){

```
 int n = 2;
 int * y = &n;  // this is broken, you're taking addresss of temporary object,
              // the pointer will point to random data in memory when you leave the function
 v.Add(y);
 Cout()<<*v[0]; // you can use U++ Cout() instead of std::cout ;)
}
```

Now, the assertion, AssertMoveable merely checks if the object is moveable. It is users responsibility to assure that anything that inherits from Moveable<T> (or is marked with NTL_MOVEABLE() macro) is actually moveable. Moveable is just a hint from a programmer to the compiler, it doesn't really make it moveable, or prevent you from doing moveability-breaking operations.

To illustrate how the pointers in TestMove() break things, you have to think little further, lets see what happens when you try to copy the SimpleVector:CONSOLE_APP_MAIN

```
{
 SimpleVector s;
 s.TestMove();

 // To actually break it, try to perform a copy operation:
 SimpleVector v;
 v=s;

 // Now it is broken, because the pointers were just copied
 // The v.val points s.a, which is in most cases not what you want
 // Also v.sv == s.sv, same problem
 // Imagine what happens when s is destructed earlier than v
 //     -> you have pointers to non-existent objects, just asking for a crash ;)
}
```

Does that shed some light on the subject?

Best regards,
Honza

## Subject: Re: moveable question
Posted by mtdew3q on Wed, 28 Nov 2012 13:34:23 GMT
View Forum Message <> Reply to Message

Hi Honza,

I will fix it when I get home tonight. I have to rush to . Thanks very much for the cool tips. I know I can make it work now.

jim

## Subject: Re: moveable question
Posted by mtdew3q on Thu, 29 Nov 2012 05:12:16 GMT
View Forum Message <> Reply to Message

Hi-

I think this SimpleVector class is moveable because:
1) No references or pointers are stored to the object itself or to subobjects in the methods of the type, into variables that exist after the method finishes execution.
2) The types that can be moved in memory using memcpy are called moveable.

I think that is good enough. It was very hard for me to try to break the assertion but it is good practice to try out memcpy.
I never did end up breaking that assertion. I took out the Vector from the members.

thanks - jim

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

using namespace Upp;
using namespace std;

class SimpleVector: Moveable<SimpleVector>{
    int len;
    char * iptr;
    int a;
    int * ptr;

public:
   SimpleVector(const char *iptr2);

SimpleVector& operator=( const SimpleVector& rhs );
SimpleVector(const SimpleVector & s1);
    ~SimpleVector();
    char* GetString();
} sv1("");


SimpleVector::SimpleVector(const char *iptr2){
 std::cout<<"Entering constructor";
 std::cout<<endl;

 len = strlen(iptr2) + 1;
 iptr = new char[len + 1];
 strncpy(iptr, iptr2, len);
 iptr[len-1]='/0';
 a = 10;
 ptr = new int(2);



 std::cout<<"leaving constructor";
}
SimpleVector& SimpleVector::operator=( const SimpleVector& rhs ){

   Cout()<<"entering assignment";
   std::cout<<endl;
   if (this == &rhs)
       return *this;

   std::cout<<"debug test"<<endl;

   delete [] iptr;
   iptr=0;
   len = rhs.len;

    delete ptr;
    ptr=0;

   if(rhs.iptr){
       iptr = new char[len + 1];
       strncpy(iptr,rhs.iptr,len);
       a = rhs.a;
       Cout()<<*iptr;
       std::cout<<endl;
 }else
       iptr=0;

 if(rhs.ptr){
      ptr = new int(2);
      memcpy(ptr,rhs.ptr,sizeof(int));
 }else
     ptr=0;

 return *this;
}

SimpleVector::~SimpleVector(){
      Cout()<< "the new soundgarden album is out!";
      std::cout<<endl;
      delete[] iptr;
      iptr = 0;

      delete ptr;
      ptr = 0;
      std::cout<<"debug test"<<endl;
      AssertMoveable<SimpleVector>();
}


CONSOLE_APP_MAIN
{
    std::cout<<endl;
    std::cout<<"program begin!";
    cout<<endl;
 SimpleVector s("foo1");
 SimpleVector s2("foo2");
// SimpleVector s2 = s; copy constructor
 s2=s;
}
```