Subject: moveable with assert question
Posted by mtdew3q on Sun, 15 Sep 2019 15:30:18 GMT
View Forum Message <> Reply to Message

Hi all-

I thought this code below would kick out an error when compiled that it is a moveable violation.

Can you please give me a tip on how to check for moveable with the code below?

thnx. roboloki

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

using namespace Upp;

class Foo: Moveable<Foo> {
    int a;
    Foo *foo;
    int *ptr;
public:
    void Set(Foo * f) {
        foo = f;
    }

void Bad1() {
        foo = this;
    // member variable foo exists outside method
    // -> makes Foo non-moveable
    }
    ~Foo() {

    }
};

template <typename T> class Metallica
{
   T * t;
   public:

   Metallica (){};

   void Set(T * t1) {
        t = t1;


   }
```

```
    ~Metallica() {
    AssertMoveable<T>();
    }
};

CONSOLE_APP_MAIN
{

 Metallica<Foo> lu;

}
```

Subject: Re: moveable with assert question
Posted by mtdew3q on Sun, 15 Sep 2019 17:38:39 GMT
View Forum Message <> Reply to Message

Hi all-

I am trying to learn the core and requirements for moveable.
Do classes still have to be moveable for pick?

I understand that pick might be a synonym for std::move.

Here is another code snippet. I am not sure why it won't work.
Please see the small class in my other post. The vector code fails
when adding a foo object. // v1.Add(f);

Please show me a trick. thnx. roboloki

```
 Foo f, f2;

 f.Set(&f2);

    //Foo && r2 = std::move(f) ;

    // int b = r2.Get();

    //  Cout() << b;

    Vector<Foo> v1;

    v1.Add(f);

    Vector<int> n;
    n.Add(3);
```

Subject: Re: moveable with assert question
Posted by mtdew3q on Sun, 15 Sep 2019 18:51:19 GMT
View Forum Message <> Reply to Message

Hi Mirek and friends:

I don't really need those constructors like assignment and copy unless I have a pointer as a class member. I think I probably could implement a move assignment function and copy assignment function for move etc. if I needed to. I took the pointers out and then it compiled fine.

I will go put a pointer as a member and make a move copy constructor and check it out.

thanks,
roboloki

Subject: Re: moveable with assert question
Posted by mtdew3q on Sun, 15 Sep 2019 19:26:30 GMT
View Forum Message <> Reply to Message

Hi all-

It is still not working. I will list the last attempt here. Not sure what I am doing wrong!

```
class Foo:Moveable<Foo>  {

   char * buffer;
public:

   Foo(){
   buffer = NULL;
   }

   Foo(const char * c) {
      if( c!=NULL)
      {
         buffer= new char [strlen(c) + 1];
         strcpy(buffer, c);
      } else
          buffer = NULL;
   }

  Foo(Foo&& foosrc) {
     if(foosrc.buffer !=NULL) {
```

```
        buffer=foosrc.buffer;
       foosrc.buffer = NULL;
     }
}


CONSOLE_APP_MAIN
{
   Foo f("foo");
   Foo && r2 = std::move(f) ;

   Vector<Foo> v1;

    v1.Add(r2);

}
```

If I am making a glaring error that you can see please help me with the trick!

thanks roboloki

Subject: Re: moveable with assert question
Posted by mtdew3q on Mon, 16 Sep 2019 02:08:08 GMT
View Forum Message <> Reply to Message

Hi Mirek and friends:

I solved the problem. It kept telling me something about deleted. Then I switched views from the error view in U++ to the view that gives more detail.

I searched on the Internet and I found this link https://
stackoverflow.com/questions/43954808/why-is-the-move-constru
ctor-defined-and-the-assignment-operator-implicitly-delet

I checked my gcc version and then decided to tack on default to one of my methods that didn't have it.

Here is the updated code that works.

```
class Foo: Moveable<Foo>  {

   char * buffer;

public:
 Foo()=default;
   Foo(const char * c);
   Foo( const Foo&) =default;
```

```
    Foo(Foo&&) = default;
    Foo& operator=(Foo&&) = default;
    String Get(){
       return buffer;
    }
    ~Foo() {
     AssertMoveable<Foo>();
     delete buffer;
    }
};

Foo::Foo(const char * c) {
       if( c!=NULL)
       {
          buffer= new char [strlen(c) + 1];
          strcpy(buffer, c);
       } else
            buffer = NULL;
    }



CONSOLE_APP_MAIN
{
    Foo f("foo");
 Foo && r2 = std::move(f);
    Foo &  j = r2 ;

    Vector<Foo> a, b;
    b.Add(r2);

    Foo & myval = b.At(0);
    Cout() << myval.Get();


}
```

thnx : roboloki

Subject: Re: moveable with assert question
Posted by mtdew3q on Mon, 16 Sep 2019 03:29:24 GMT

View Forum Message <> Reply to Message

Hi all-

I think the references get invalidated in this case. I will work on my understanding of that next. I

am thinking either somehow I can use a smart pointer U++ style or C++ kind | pick a different container than vector.

I am headed back to a vicious work cycle thought (Mon. thru Fri.)

Hope everyone has a cool week :)

roboloki

Subject: Re: moveable with assert question
Posted by mirek on Mon, 16 Sep 2019 08:10:12 GMT
View Forum Message <> Reply to Message

I think the one thing that you misunderstood is that it is up to programmer to decide what is Moveable and what is not. Unfortunately, this too complex for any compiler check, so programmers is required to mark Moveable types (by deriving from Moveable<T>).

Mirek

Subject: Re: moveable with assert question
Posted by mtdew3q on Mon, 16 Sep 2019 11:37:21 GMT
View Forum Message <> Reply to Message

Hi mirek.

Oh...

No worries. You have good tips on deciding in the documentation.

Thnx again,
Jim
