Subject: MSC problems: pick_ != const
Posted by Mindtraveller on Sun, 19 Apr 2009 21:07:04 GMT
View Forum Message <> Reply to Message

For example, this code causes error:
```
struct AOp : Moveable<AOp>
{
 One<AOpHardware> hardware;
};

class AOps : public Vector<AOp>
{
public:
 void Xmlize(XmlIO xml) {XmlizeContainer(xml, "aop", *this);}
};
```
This is caused by the fact that pick != const in MSC compiler. And somewhere within U++ Core we have copying of Vector member with const argument (not pick_!):
```
//Core/Topt.h @ 135
template <class T>
inline void DeepCopyConstruct(void *p, const T& x) {
 ::new(p) T(x);
}
```

This is strange. Class One supports pick behaviour, so accroding to U++ principles this code should compile without errors.
Am I right?

Subject: Re: MSC problems: pick_ != const
Posted by mirek on Sun, 19 Apr 2009 22:25:22 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 19 April 2009 17:07For example, this code causes error:
```
struct AOp : Moveable<AOp>
{
 One<AOpHardware> hardware;
};

class AOps : public Vector<AOp>
{
public:
 void Xmlize(XmlIO xml) {XmlizeContainer(xml, "aop", *this);}
};
```
This is caused by the fact that pick != const in MSC compiler. And somewhere within U++ Core we have copying of Vector member with const argument (not pick_!):
```
//Core/Topt.h @ 135
template <class T>
inline void DeepCopyConstruct(void *p, const T& x) {
```

::new(p) T(x);
}

This is strange. Class One supports pick behaviour, so accroding to U++ principles this code should compile without errors.
Am I right?

Yes and no.

pick definitely is not equal to semantics of const. (const guarantees no change).

That, after all, is why there is #define pick_...

Anyway, we are sort of at odds with C++ standard here as we would like to have pick_ constructors used when returning containers from functions. According to C++ standard, this is only possible if they are const (the critical rule is "nonconst references cannot be bound to temporaries").

Surprisingly, MSC++ has relaxed rules (you can call it a bug) w.r.t. to this, probably because some old MFC code was breaking the rule too. As we know when we are compiling with MSC, I decided make #define pick_ empty in that case; it better matches wanted pick_ semantics and is able to catch a bug here are there, like if you are trying to use DeepCopyConstruct on pick type.

Mirek

Subject: Re: MSC problems: pick_ != const
Posted by Mindtraveller on Mon, 20 Apr 2009 06:11:27 GMT
View Forum Message <> Reply to Message

Then the question is why U++ uses DeepCopyConstruct for Vector elements? The code seems "clean" from U++ point of view. So why isn`t it compiled successfully?

ADD: If the problem is with Xmlize, could you please tell why and how to solve it for pick types?

Subject: Re: MSC problems: pick_ != const
Posted by mirek on Mon, 20 Apr 2009 07:22:45 GMT
View Forum Message <> Reply to Message

You need to add deep copy contructor to AOp.

It can either be 'implicit' one or 'optional'.

With optional, you need to add uniform access using DeepCopyOption.

See

http://www.ultimatepp.org/srcdoc$Core$pick_$en-us.html

Note that would it be your way, data would be destroyed during 'Store' operation. I guess that is not one would expect...

Mirek
