Subject: Problem with Vector::Add (pick/clone semantics)
Posted by shutalker on Fri, 09 Aug 2019 10:14:12 GMT
View Forum Message <> Reply to Message

Hi all!

I've encountered with the following problem. When I try to define such object as

```
const static VectorMap<String, Vector<String>> MY_MAP = {
    {"s1", pick(Vector<String>{"s11", "s12", "s13", "s14"})},
    {"s2", pick(Vector<String>{"s21", "s22", "s23", "s24"})},
    {"s3", pick(Vector<String>{"s31", "s32", "s33", "s34"})},
    {"s4", pick(Vector<String>{"s41", "s42", "s43", "s44"})},
    {"s5", pick(Vector<String>{"s51", "s52", "s53", "s54"})}
};
```

I get several errors like this

```
/home/alexis/upp/uppsrc/Core/Vcont.hpp (158): error: call to implicitly-deleted copy constructor of
'Upp::Vector<Upp::String>'
...
/upp/uppsrc/Core/Core.h (357): In file included from /home/alexis/upp/uppsrc/Core/Core.h:357:
 (): T *q = new(Rdd()) T(x);
/home/alexis/upp/uppsrc/Core/Vcont.h (132): note: in instantiation of member function
'Upp::Vector<Upp::Vector<Upp::String> >::GrowAdd' requested here
 (): T&       Add(const T& x)                { return items < alloc ? *(new(Rdd()) T(clone(x))) :
GrowAdd(x); }
/home/alexis/upp/uppsrc/Core/Map.h (51): note: in instantiation of member function
'Upp::Vector<Upp::Vector<Upp::String> >::Add' requested here
 (): T&       Add(const K& k, const T& x)           { key.Add(k); return value.Add(x); }
/home/alexis/upp/uppsrc/Core/Map.h (179): note: in instantiation of member function
'Upp::AMap<Upp::String, Upp::Vector<Upp::String>, Upp::Vector<Upp::Vector<Upp::String> >
>::Add' requeste
d here
 (): AMap(std::initializer_list<std::pair<K, T>> init) { for(const auto& i : init) Add(i.first, i.second); }
/home/alexis/upp/uppsrc/Core/Map.h (236): note: in instantiation of member function
'Upp::AMap<Upp::String, Upp::Vector<Upp::String>, Upp::Vector<Upp::Vector<Upp::String> >
>::AMap' reques
ted here
 (): VectorMap(std::initializer_list<std::pair<K, T>> init) : B::AMap(init) {}
```

I guess the reason is

```
T *q = new(Rdd()) T(x); // <-- should be clone(x)
```

So I made a little patch that fixed the problem. Please, check it and give feedback if I did something wrong :)

UPD
I use upp from git repository https://github.com/ultimatepp/mirror

Used compiler: FreeBSD clang version 6.0.0 (tags/RELEASE_600/final 326565) (based on LLVM 6.0.0)

File Attachments

1) vcont.patch, downloaded 275 times

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by Novo on Fri, 09 Aug 2019 18:12:20 GMT
View Forum Message <> Reply to Message

I personally would say that this is not a bug. This is a feature. :)
clone was intentionally removed from Add to prevent implicit cloning.
Basically, std::initializer_list will create a temporary const object and after that it will force you to create another copy of it. This is an unnecessary allocation.
U++ is warning you about that and offering you other tools like

```
 VectorMap<String, Vector<String>> MY_MAP;
 MY_MAP.Add("s1", Vector<String>{"s11", "s12", "s13", "s14"});
```

In this case objects will be moved.

Ideally, it would be great to have a set of overloaded operators VectorMap& operator()(const K& k, const T& v)

More details on this problem can be found here.
A comment to this article has an interesting code snippet:

```
template<std::size_t N>
Vec(T(&&a)[N])
  : _vect(std::make_move_iterator(std::begin(a)), std::make_move_iterator(std::end(a)))
{}
```

Extra braces needed though, but somebody may find this more idiomatic:

```
Vec<int> v {{1, 2}};
```

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by Novo on Sat, 10 Aug 2019 04:33:51 GMT
View Forum Message <> Reply to Message

Actually, it is possible to move data from std::initializer_list with a little hack:

```
template <typename T>
```

```
struct Foo {
 Foo(std::initializer_list<T> init) {
  for(const T& i : init)
   v.Add(pick(const_cast<T&>(i)));
 }
// Foo(std::initializer_list<T> init) {
//  for(const T& i : init)
//   v.Add(i);
// }

 Vector<T> v;
};

struct Boo : Moveable<Boo> {
 Boo() {}
 Boo(const Boo&) = default;
 Boo(Boo&&) = delete;
};

CONSOLE_APP_MAIN
{
 Foo<Vector<int>> f = {{1}};
// Foo<Boo> f = {Boo()};
}
```

The problem is that this will require all types to have a move constructor.
A move constructor can be detected via std::is_move_constructible, but I couldn't figure out how to apply SFINAE to a constructor.

IMHO, all this code complexity is unnecessary in this case.

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by mirek on Tue, 13 Aug 2019 07:08:23 GMT
View Forum Message <> Reply to Message

I made it work, even without pick:

```
 const static VectorMap<String, Vector<String>> MY_MAP = {
     {"s1", Vector<String>{"s11", "s12", "s13", "s14"}},
     {"s2", Vector<String>{"s21", "s22", "s23", "s24"}},
     {"s3", Vector<String>{"s31", "s32", "s33", "s34"}},
     {"s4", Vector<String>{"s41", "s42", "s43", "s44"}},
     {"s5", Vector<String>{"s51", "s52", "s53", "s54"}}
 };
```

(making this work is perhaps slight departure from "use clone/pick always", OTOH I feel uneasy altering initialization data (by pick) anyway).

Mirek

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by Novo on Tue, 13 Aug 2019 14:02:21 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 13 August 2019 03:08I made it work, even without pick:

This won't compile:

```
 const VectorMap<Vector<String>, String> MY_MAP = {
    {Vector<String>{"s11", "s12", "s13", "s14"}, "s1"},
 };
```

pick wasn't needed because Vector<String>{...} is an rvalue by itself.

It would be great to have all overloads of
```
 VectorMap& VectorMap&::operator()(const K& k, const T& v)
```

similar to AMap::Add(k, v).

IMHO, the problem is not a constructor of VectorMap, but an implementation of std::initializer_list. I believe I saw an alternative implementation somewhere.

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by mirek on Tue, 13 Aug 2019 15:09:54 GMT
View Forum Message <> Reply to Message

Novo wrote on Tue, 13 August 2019 16:02mirek wrote on Tue, 13 August 2019 03:08I made it work, even without pick:

This won't compile:

```
 const VectorMap<Vector<String>, String> MY_MAP = {
    {Vector<String>{"s11", "s12", "s13", "s14"}, "s1"},
 };
```

pick wasn't needed because Vector<String>{...} is an rvalue by itself.

Works now. Thanks.

Quote:
It would be great to have all overloads of
 VectorMap& VectorMap&::operator()(const K& k, const T& v)

similar to AMap::Add(k, v).

Done. Long live std::forward...

Mirek

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by Novo on Thu, 15 Aug 2019 01:47:50 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 13 August 2019 11:09Done. Long live std::forward...

Mirek
Thank you!

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by shutalker on Mon, 26 Aug 2019 08:42:28 GMT
View Forum Message <> Reply to Message

Mirek, Novo, thank you! Your explanations and this
 article are very helpful, though I should find out more about initialization by myself :)

Subject: Re: Problem with Vector::Add (pick/clone semantics)
Posted by mr_ped on Mon, 26 Aug 2019 17:27:05 GMT
View Forum Message <> Reply to Message

About C++ initialization .. (animated gif .. sort of joke... but not really):
https://i.imgur.com/3wlxtI0.gifv