Subject: Vector<T>::Set(int i, T&& x) proposal
Posted by Novo on Tue, 19 Dec 2017 22:30:19 GMT
View Forum Message <> Reply to Message

I propose to add method below to the Vector class.

```
template <class T>
T& Vector<T>::Set(int i, T&& x) {
 ASSERT(i >= 0);
 const int count = GetCount();
 if (i == count)
  return Add(pick(x));
 else if (i > count) {
  At(i - 1);
  return Add(pick(x));
 }

 T* addr = vector + i;
 addr->~T();
 ::new(addr) T(pick(x));
 return *addr;
}
```

Or it can be implemented like this:

```
template <class T>
T& Vector<T>::Set(int i, T&& x) {
 ASSERT(i >= 0);
 At(i);
 T* addr = vector + i;
 addr->~T();
 ::new(addr) T(pick(x));
 return *addr;
}
```

Second implementation has less memory allocation stuff. It is hard to tell which one is better.
I believe there is no need to check that x is already contained in Vector because it is impossible to get an rvalue of a Vectors's element.

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by mirek on Sun, 28 Jan 2018 21:47:29 GMT
View Forum Message <> Reply to Message

I am considering this, but single element Set was always just "convenience" in addition to 'count' variant and you can achieve the same with

```
v.At(i) = pick(src);
```

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by Novo on Tue, 30 Jan 2018 02:30:25 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 28 January 2018 16:47I am considering this, but single element Set was always just v.At(i) = pick(src);

Well, yes. This will work as well. But this is significantly less intuitive. Your way of solving this problem definitely didn't come to my mind when I was looking for a solution. Adding of an rvalue-based version of Set definitely won't break API because you already have a reference-based version. This is just another performance optimization.

My solution is based on move-constructor and yours is based on move-operator.

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by mirek on Wed, 31 Jan 2018 18:34:54 GMT
View Forum Message <> Reply to Message

Novo wrote on Tue, 30 January 2018 03:30mirek wrote on Sun, 28 January 2018 16:47I am considering this, but single element Set was always just v.At(i) = pick(src);

Well, yes. This will work as well. But this is significantly less intuitive. Your way of solving this problem definitely didn't come to my mind when I was looking for a solution. Adding of an rvalue-based version of Set definitely won't break API because you already have a reference-based version. This is just another performance optimization.

My solution is based on move-constructor and yours is based on move-operator.

OK. After further thinking, for some time now I think it is worthwhile to add

```
const T& Vector::Get(int i, const T& default) { return i >= 0 && i < GetCount() ? Get(i) : default; }
```

Set makes a nice complement to this, so it makes sense to add Set(int, T&&) too.

Do you see any problem with trivial implementation

```
void Vector::Set(int i, T&& x) { At(i) = pick(x); }
```

?

BTW, this is simple. But if I am about to add it here, I have to do that for every container where it

makes sense and fill documentation too, add autotests. Thats just to explain my hesitation...

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by Novo on Wed, 31 Jan 2018 19:01:22 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 31 January 2018 13:34
Do you see any problem with trivial implementation

void Vector::Set(int i, T&& x) { At(i) = pick(x); }

IMHO, it should look like below.
T& Vector::Set(int i, T&& x) { return At(i) = pick(x); }

My current coding pattern (without the T&& version) looks like that:
T& v = vector.At(ind) = T(args);

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by mirek on Thu, 01 Feb 2018 17:53:34 GMT
View Forum Message <> Reply to Message

done...

Subject: Re: Vector<T>::Set(int i, T&& x) proposal
Posted by Novo on Thu, 01 Feb 2018 18:03:22 GMT
View Forum Message <> Reply to Message

Thanks!