# Should there be a standard C++ pattern for this? transform_to

devblogs.microsoft.com/oldnewthing/20200228-00

February 28, 2020

Raymond Chen

I've got one type of collection and I want to apply a function to each member of the collection, thereby producing a new collection.

Surely there's a standard pattern for this?

In JavaScript, it's called `map`:

```
function getOldValues()
{
    return ["a", "b", "c", "d"];
}

var newValues = getOldValues().map(v => v.charCodeAt(0));
// result: [97, 98, 99, 100]
```

In C#, it's `Select`.

```
string[] GetOldValues() => new[] { "a", "b", "c", "d" };

var newValues = GetOldValues().Select(v => (int)v[0]).ToArray();
// result: int[] { 97, 98, 99, 100 };
```

In C++, it's, um, this clumsy `std::transform`.

```
std::vector<std::string> GetOldValues()
{
   return { "a", "b", "c", "d" };
}

auto oldValues = GetOldValues();
std::vector<int> newValues;
newValues.reserve(oldValues.size());
std::transform(oldValues.begin(), oldValues.end(),
    std::back_inserter(newValues),
    [](auto&& v) { return v[0]; });
```

It's clumsy because you need to give a name to the thing being transformed, because you need to call both `begin` and `end` on it. But giving it a name extends its lifetime, so you end up carrying this `oldValues` vector around for no reason.[1]

It's clumsy because you have to construct an empty `newValues` and then fill it in.

Would be nice if there were some helper function like

```
template<typename T, typename U, typename TLambda>
T transform_to(U&& u, TLambda&& lambda)
{
  T result;
  if constexpr (has_size_v<U> && has_reserve_v<T>)
  {
    result.reserve(u.size());
  }
  std::transform(u.begin(), u.end(), std::back_inserter(result),
                 std::forward<TLambda>(lambda));
  return result;
}

auto newValues = std::transform_to<std::vector<int>>(
    GetOldValues(), [](auto&& v) { return v[0]; });
```

Maybe one exists and I'm missing it? Help me out here.

[1] You can avoid extending the lifetime beyond the transform by pushing it into a lambda:

```
auto newValues = [&]()
{
    auto oldValues = GetOldValues();
    std::vector<int> newValues;
    newValues.reserve(oldValues.size());
    std::transform(oldValues.begin(), oldValues.end(),
        std::back_inserter(newValues),
        [](auto&& v) { return v[0]; });
    return newValues;
}();
```

but that's basically just taking the `transform_to` function and inlining it as a lambda.

Raymond Chen

**Follow**