Pulling a single item from a C++ parameter pack by its index, remarks

devblogs.microsoft.com/oldnewthing/20240930-00

September 30, 2024

In my earlier discussion of pulling a single item from a C++ parameter pack by its index, I noted that with the Pack Indexing proposal, you would be able to write this to pull an item from a parameter pack while preserving its reference category.

```
template<int index, typename...Args>
void example(Args&&... args)
{
    auto&& arg = (Args...[index]&&)args...[index];
    using Arg = Args...[index]&&;
}
```

Why did I need to apply the `(Args...[index]&&)` cast? Why can't I just write this:

```
    auto&& arg = args...[index];
```

or possibly

```
    decltype(auto) arg = args...[index];
```

Well, when you write the name of a variable, the result is an lvalue reference, *even if the variable is an rvalue reference*. Watch:

```
struct S
{
    S(S&); // construct from lvalue
    S(S&&); // construct from rvalue
};

void whathappens(S&& s)
{
    S t = s; // which will it use?
}
```

Try it out in your favorite compiler. This code constructs from an *lvalue* reference. Even though `s` is an rvalue reference, when you say its name, you get an lvalue reference, so that's the construtor that gets selected.

You sort of knew this already. For example, you can't take the address of an rvalue reference, but you can write this:

```
void whathappens(int&& v)
{
    int* p = &v; // legal!
}
```

And you've been writing `std::move` to say "It's okay to move from this object."

```
void whathappens(S&& s)
{
    S t = std::move(s); // force rvalue
}
```

I mean, that's why you've been writing `std::move` and `std::forward` all these years. If writing `s` already produced an rvalue reference, then there would be no need to `std::move` or `std::forward` it.

"Okay, I get it. Writing the name of a variable that represents an rvalue reference produces an lvalue. So what?"

Since writing the name of the variable produces an lvalue reference, `decltype(auto)` sees that the right hand side is an lvalue reference, so it deduces an lvalue reference.

Now, you could say "Well, sure, but let's make a special case for indexed elements from a parameter pack, so that saying their name produces an rvalue reference if the corresponding parameter is an rvalue." But that creates another weird special case in C++, and C++ is hard enough to language-lawyer without adding *even more* weird special cases.[1]

Bonus chatter: Instead of

```
    auto&& arg = (Args...[index]&&)args...[index];
```

I could also have written

```
    auto&& arg = std::forward<Args...[index]>(args...[index]);
```

which is wordier but probably clearer. I was sort of assuming people understood this common shortcut.

[1] I generally believe in the principle that it is better to have a set of simple rules that are easy to understand and explain, even if it means that some scenarios are awkward or suboptimal, as opposed to a set of rules that cover all scenarios but which are so complex that nobody

can understand them, much less explain them.