C++ User Group - Talk #660

bringing physics units to C++

2014-06-02 15:39 - Kretz, Matthias

Status:	New	**Start date:**	2014-06-02
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Talk Author(s):		**Presenter:**	

Description

In the std::chrono talk (#566) we've seen the use of std::ratio for encoding the unit prefix / scale associated with an normal numerical value (i.e. seconds vs. milliseconds vs. days vs. ...). In the following discussion we talked about the power of encoding numeric base-type, physical unit, and scaling in the type as the compiler can thus enforce correct usage of physical units.

A talk could explore existing solutions (if there are any published ones) and/or present how such a solution can be implemented and how it affects application development.

Related issues:

Blocked by Talk #794: Templates and Meta-Programming	**Presented**	**2014-07-22**	**2014-12-17**

History

#1 - 2014-07-15 10:19 - Glesaaen, Jonas

There is actually an example of this in the boost documentation for their library on template metaprogramming:

http://www.boost.org/doc/libs/1_55_0/libs/mpl/doc/tutorial/dimensional-analysis.html

Although it does not explicitly go into how one can also combine it with other unit systems (natural units vs SI units vs CGS), the example might be combined with std::ratio to achieve such an effect.

#2 - 2014-07-21 14:38 - Kretz, Matthias

Thanks. The boost documentation (and implementation idea) is a nice reference. Though, IIUC, the boost::mpl doesn't do anything other than providing the vector_c class template and mpl::transform to ease implementation. Here's a short and maybe more flexible alternative:

```
#include <type_traits>

template <typename T> struct plus       { constexpr T operator()(T a, T b) { return a + b; } };
template <typename T> struct minus      { constexpr T operator()(T a, T b) { return a - b; } };
template <typename T> struct multiplies { constexpr T operator()(T a, T b) { return a * b; } };
template <typename T> struct divides    { constexpr T operator()(T a, T b) { return a / b; } };

template <int v0, int... values>
class dimension {
  template <typename Op, int w0, int... ws>
  static constexpr dimension<Op()(v0, w0), Op()(values, ws)...>
      transform_internal(dimension<w0, ws...> &&);

  template <int position>
  static constexpr int get_internal(std::true_type) {
    return v0;
  }
  template <int position>
  static constexpr int get_internal(std::false_type) {
    static_assert(position < sizeof...(values) + 1,
                  "dimension::get: position too large, must be smaller than "
                  "the number of template arguments to dimension.");
    static_assert(
        position >= 0,
        "dimension::get: position too small, must be zero or positive.");
    return dimension<values...>::template get<position - 1>();
  }

 public:
  template <int position>
  static constexpr int get() {
```

```
    return get_internal<position>(
        std::integral_constant<bool, position == 0>());
  }

  template <template <typename T> class Op, typename OtherDimension>
  using transform =
      decltype(transform_internal<Op<int>>(std::declval<OtherDimension>()));
};
```

One could then define

```
using LengthDim = dimension<1, 0, 0, 0, 0>;
```

In the next step you could create a type that defines the relative factor of each dimension to SI units. Then couple that with a std::ratio and create a quantity class that is even more powerful than the example in boost. I.e. you could write:

```
quantity<double, PlanckUnits, LengthDim, std::kilo> l0(1e32);
quantity<double, SiUnits, LengthDim> l1 = l0;
std::cout << l1.value() << " meters\n";
```

and have that print

> 1.6162 meters

#3 - 2014-07-22 08:09 - Kretz, Matthias

Jonas, would you want to take this presentation? (As always, I'd be happy to help preparing it.)

#4 - 2014-07-22 10:04 - Glesaaen, Jonas

Kretz, Matthias wrote:

> Jonas, would you want to take this presentation? (As always, I'd be happy to help preparing it.)

I have to say I am a bit intimidated looking at the code example you posted, meaning that I probably shouldn't. I haven't actually done any template metaprogramming before, and I just posted the link as I stumbled upon it when I wanted to learn a bit of metaprogramming. If I had more time, then maybe we could work through it together, but I am definitely still a padawan in C++ and TMP and have much to learn.

Yes, I know the boost example doesn't do anything nearly as complicated as what we talked about, and doesn't even do the same things (using std::ratio to convert between orders of magnitude or different choices in units), but I found it to be interesting and maybe an introduction to a way of thinking, or as a reference as you put it.

#5 - 2014-07-22 10:20 - Kretz, Matthias

There's no better way to learn it, IMHO. But as you say it requires time. If you think you can make up some time to get into this (with one or two weeks of half-a-day playing with these classes you should get quite far already), I'll be happy to get you started.

Maybe, though, it would make sense to have a presentation on templates and some template meta-programming first?

#6 - 2014-07-22 10:26 - Glesaaen, Jonas

If you haven't already had a talk about template meta-programming in the user group already, I think that would be a good idea. At least I know that I myself would be interested in that.

As for time, I should have some time to play with these things myself (and some start help would be very much appreciated), but as the next user group is scheduled for next Wednesday, I am not sure I would have the time to first learn or get into TMP as well as put together a good presentation on the topic.

#7 - 2014-07-22 10:30 - Kretz, Matthias

Ah, I didn't mean to ask you for this month's presentation. Only for this topic...

I think we'll drop the August meeting because of summer holidays, so the first chance for presenting this would be end of September - where we might do #543. So there's plenty of time.

#8 - 2014-07-22 10:36 - Glesaaen, Jonas

Ah I see, sorry I inferred that. In my mind I sort of thought that we needed a topic for this month and that is why you asked.

In that case, I would love to have a talk on TMP if I am able to wrap my head around it. It is a topic which has intrigued me ever since I found out

about it.

If you have time to get me started on TMP (or simply point me towards some recommended literature) that would be very much appreciated as I am not sure where to start reading.

#9 - 2014-07-22 10:36 - Kretz, Matthias

- Blocked by Talk #794: Templates and Meta-Programming added

#10 - 2014-07-22 10:45 - Kretz, Matthias

The Alexandrescu book — Modern C++ design got the ball rolling. I think it's a very nice read, even if it's rather old by now. Since then most of his suggestions have found their way into either boost, the C++ language itself, or the C++ standard library.

I learned TMP by trying it out. It's my preferred style of learning stuff, so can't point to many good resources (other than the search engine + stack overflow + IRC + user group combination :)).