Google Summer of Code Proposal: Boost.StaticViews

March 2018

Abstract

Yet to be written. For now, here's the last year's abstract:

This document proposes an addition to `Boost C++ Libraries` – a *compile-time* hash table. There are multiple good implementations of unordered associate containers (e.g. `std::unordered_map`, `Google`'s `sparsehash`). These implementations provide both lookup and insertion/deletion functionality. They are, however, not the perfect fit for the case when the contents of the container are fixed upon construction or even known at compile time. `std::vector` vs. `std::array` is a good analogy here. We propose a `static_map` – an associate container with focus on `constexpr` usage.

1 Personal Details

Name	Tom Westerhout
University	Radboud University of Nijmegen, The Netherlands
Degree Program	Masters in Physics
Email	`kot.tom97@gmail.com`
Availability	I am available from May till the end of August. I consider `GSoC` a full-time job/internship and thus plan to spend at least 40 hours per week working on the project. 14th (Monday) of May (the official start date) seems like a good starting point to me. My summer vacation officially starts only in the beginning of July, before that I have some courses to follow, exams to take and a research project to work on. Although it seems much, last year's experience shows that I still find enough time to work on the project. Also, I will have more free time in July and August and can easily compensate for the lost time, if any.

2 Background Information

I am currently in my first year of Masters in Physics program at Radboud University (The Netherlands). Although I am a Physics major, I am also passionate about programming. I follow many programming-related courses and am considering doing a Masters in Computer Science. So far, I have followed the following project-related courses:

- *Imperative Programming 1 & 2* (Introduction to `C++`).
- *Hacking in C* (Memory layout, calling conventions and debugging).

- *Object Orientation.*
- *Algorithms & Datastructures* (I've also worked as a TA (Teaching Assistant) for this course).
- *Languages and Automata.*
- *Functional Programming 1 & 2.*
- *Advanced Programming* (Generic programming, DSLs (Domain Specific Languages), etc.)
- *Compiler Construction.*

Also, in January 2018 I have participated in 3COWS (Composability, Comprehensibility, Correctness Winter School). It was an intensive programme for Ms. and PhD. students extending the CEFP (Central European Functional Programming) summer school.

For two and a half years I have been worked as a teaching assistant at Radboud University. I have taught physics, math, and computer science courses.

As for the projects, as part of my Bachelor internship, I have written some `C++14` code for distributed memory systems for doing some physical calculations. We have recently published a paper about the results of these computations. Although the algorithms involved are not particularly difficult, as far as I know, this is still the only open source solution for calculating plasmonic properties of non-translational invariant quantum systems.

Currently, I am working on applying swarm intelligence to obtaining excited states of many-body quantum systems. High level algorithms are written in `Haskell` and performance critical parts — in `C++17` with a `C` wrapper in between. We use shallow neural network (Restricted Boltzmann Machine) as a variational ansatz by tracing out all hidden nodes. This part is written in `C++`. Also, Monte-Carlo simulations which are used to estimate the fitness function are written in `C++` as well. We hope to present some first results at "Beyond Digital Computing" conference in Heidelberg near end of March 2018.

Finally, last summer I have successfully completed a Google Summer of Code working on the very same project I am proposing here. I believe StaticViews library to be very useful. It is however not possible to create a Boost-quality library in just one summer. Having a relatively high study load (due to my following both Physics and CS courses) I don't have enough time to work on the project from September till May. I did however manage to follow all the major discussions on `boost-dev` mailing list. Also, I feel like I may need some guidance to complete this project. These two reasons are my main motivation for doing GSoC this year.

As for my knowledge of the listed languages/technologies/tools, I would rate it as follows:

`C++ 98/03` ≈ 3;

`C++ 11/14/17` ≈ 4;

`C++ Standard Library` ≈ 4;

`Boost C++ Libraries` ≈ 4;

`Git` ≈ 3.

I use `Linux` as my primary desktop operating system and have grown used to doing as much as possible via command line. Thus, usually I write code in `Neovim` and use some build system to compile it. From the `C++`–related ones I am familiar with `Boost.Build` and `CMake`. For documentation, I have grown to like `Sphinx` (which StaticViews currently uses). I also have experience in using `Doxygen` and have played around with `Standardese`.

3 Project Proposal

Motivation There is often need for associative containers. `C++ Standard Library` provides `std::map` and `std::unordered_map`. `Google` has `dense_hash_map`. There are many more implementations. These all focus on both insertion/deletion and lookup. However, sometimes we are only interested in the lookup capabilities. For example, when the container is initialised from constant static data or even constexpr data. While a C-style associative array may serve well as a storage for such data, it is inefficient in terms of accessing the data by key. Common associate containers, on the other hand, implement efficient lookup, but use dynamic memory allocation for initialisation which may be undesired in some cases. The following example illustrates this further.

```
#include <iostream>
#include <initializer_list>
#include <map>
#include <unordered_map>
#include <experimental/string_view>
using std::experimental::string_view;

enum class weekday { sunday, monday, tuesday, wednesday
                   , thursday, friday, saturday };

#define STRING_VIEW(str) string_view{ str, sizeof(str)-1 }
constexpr std::initializer_list<std::pair< const string_view
                                         , weekday>> string_to_weekday
    { { STRING_VIEW("sunday"),    weekday::sunday    }
    , { STRING_VIEW("monday"),    weekday::monday    }
    , { STRING_VIEW("tuesday"),   weekday::tuesday   }
    , { STRING_VIEW("wednesday"), weekday::wednesday }
    , { STRING_VIEW("thursday"),  weekday::thursday  }
    , { STRING_VIEW("friday"),    weekday::friday    }
    , { STRING_VIEW("saturday"),  weekday::saturday  }
    };

int main(void)
{
    {
        // Calls malloc() at least 7 times
        static const std::map<string_view, weekday>
            to_weekday1 = string_to_weekday;
        std::cout << "'monday' maps to "
                  << static_cast<int>(to_weekday1.at("monday"))
                  << std::endl;
        std::cout << "'friday' maps to "
                  << static_cast<int>(to_weekday1.at("friday"))
                  << std::endl;
        // Calls free() at least 7 times
    }
    {
        // Calls malloc() at least 8 times
        static const std::unordered_map<string_view, weekday>
            to_weekday2 = string_to_weekday;
        std::cout << "'monday' maps to "
                  << static_cast<int>(to_weekday2.at("monday"))
                  << std::endl;
        std::cout << "'friday' maps to "
                  << static_cast<int>(to_weekday2.at("friday"))
                  << std::endl;
        // Calls free() at least 8 times
    }
    return 0;
}
```

Even though the `string_to_weekday` is `constexpr`, to make use of fast lookups ($\mathcal{O}(1)$ for `unordered_map` and $\mathcal{O}(\log(N))$ for `map` compared to $\mathcal{O}(N)$ of linear searching) we have to copy data at runtime, which involves dynamic memory allocation. And although one may implement stack allocator to prevent dynamic memory allocation for small datasets, initialisation and lookups will still happen at runtime, because they are not marked `constexpr`.

With `C++14` relaxed constraints on `constexpr` functions it is possible to implement an associative container with `constexpr` (whenever possible) key lookups (as a proof, see the toy implementation of such a container in Appendix **??**). I think that a high quality implementation of such a container would be a worthwhile addition to `Boost`.

Proposal As part of the Google Summer of Code 2017 project I propose to do the following:

- To seek consensus from the Boost Developer's mailing list on a suitable design of a `static_map` class with the following design features:

  1. The number of key–value pairs is fixed upon construction.
  2. All features can be used in constant expressions, i.e. all member functions are marked `constexpr`.
  3. The container can be statically initialised in the mind the compiler or global static storage.
  4. Values, though neither keys nor the number of items, are modifiable.
  5. The container performs constexpr key lookups to the "maximum possible extent".

  This list of requirements is harder to implement than it may seem at first sight. Consider the following example:

```
constexpr std::pair<int, char const*>
    map_data[] = { { 5, "apple" }
                 , { 8, "pear" },
                 , { 0, "banana" }
                 };

// Generates no runtime code. Easy, because the standard requires this to be executed
// at compile-time.
constexpr auto cmap = make_static_map(map_data);
constexpr char const* what_is_5 = cmap[5];

// Generates no runtime code. Easy, because 0 is a non-type template parameter.
char const* what_is_0 = std::get<0>(cmap);

// Challenging: should only generate code loading immediately from a memory location.
// It must NOT generate any additional runtime overhead like hashing or searching.
char const* what_is_8 = cmap[8];

// Challenging: again, should only generate code loading from memory.
auto cmap2 = make_static_map(map_data);
auto& at_0 = map_data[0];
at_0 = "orange";
```

  If all inputs for a `constexpr` operation are constant expressions, but the result is not used as a constant expression, then the compiler in *not required* to execute the operation at compile-time. This is what makes lines 19, 23 and 24 so challenging.

- To implement a `static_map` unordered associative container class which satisfies the above outlined requirements on top of the requirements in the standard. This includes the implementation of utility classes/functions for `constexpr` string comparison and hashing. The implementation should work on at least two major compilers in, hopefully, `C++14` mode.[1]

- To implement a comprehensive unit test suite for the `static_map` class with focus on ensuring no runtime overhead for the challenging cases mentioned above (i.e. when the result of a `constexpr` function is not used as a constant expression).

- To configure a per-commit continuous integration for the unit test suite on at least one of the major public open source CI services (e.g. Travis, Appveyor).

- To write documentation to `Boost` quality levels for the `static_map` container class. This includes time and space complexity guarantees and benchmarks, and exception guarantees for each function in the API and each use of each function.

[1] In theory, one may even make the implementation `C++11` compatible by making heavy use of template recursion. This, however, requires many functions to be implemented twice, and thus will not be pursued in this project.

Schedule I propose the following schedule to achieve the goals outlined above:

29th May — 2nd June 5th June — 9th June	Consult Boost Developer's mailing list and do some background reading to come up with a suitable design for `static_map` class and all required utility classes. In the meantime configure the system for development, i.e. get compilers, CI service, doc tools, etc., working.
12th June — 16th June 19th June — 23th June 26th June — 30th June	Implement `static_map` class for the simple case of keys being of integral type. Start writing tests and documentation. It would be nice to have a working implementation (for this simple case) before the first phase evaluation deadline.
3rd July — 7th July 10th July — 14th July 17th July — 21th July 24th July — 28th July	Complete the implementation of `static_map` class and utility classes. Continue working on tests and documentation. It would be nice to have a fully working (on at least two compilers) implementation before the second phase evaluation deadline.
31st July — 4th August 7th August — 11th August	Finish implementing tests. Finish the documentation. Maybe do some work to support other compilers.
14th August — 18th August	Extra
21st August — 25th August	Final touches, submitting results.