GSoC 2024

Project Proposal

Language Interoperability

CxxWrap.jl

Moida Praneeth Jain

Mentor: Bart Janssens

Google Summer of Code

Table of Contents

1. Introduction 3
  1.1. Project Synopsis 3
  1.2. Why I chose this project 3
  1.3. Relevant Work 3
  1.4. Technical Skills 3
  1.5. Why choose me 3
  1.6. Contact Information 3
2. Benefits to Community 3
3. Deliverables 3
  3.1. Primary Goals 3
    3.1.1. Add STL Container Types 3
    3.1.2. Add STL Algorithms 3
    3.1.3. Documentation 4
    3.1.4. Testing 4
  3.2. Stretch Goals 4
    3.2.1. Add Iterator Support 4
    3.2.2. Add more STL Container Types 4
4. Project Details 4
  4.1. Codebase 4
  4.2. STL Containers 4
    4.2.1. libcxxwrap component 4
    4.2.2. CxxWrap component 5
  4.3. STL Algorithms 5
5. Project Schedule 5
  5.1. Community Bonding Period 5
  5.2. First Coding Period 6
  5.3. Midterm Evaluation 6
  5.4. Second Coding Period 6
  5.5. Final Submission 6
  5.6. Availability 6

1. Introduction

1.1. Project Synopsis

1.2. Why I chose this project

1.3. Relevant Work

1.4. Technical Skills

1.5. Why choose me

1.6. Contact Information

2. Benefits to Community

3. Deliverables

Through this project, I aim to expose a larger portion of the C++ standard library to Julia.

3.1. Primary Goals

3.1.1. Add STL Container Types

The following containers, along with their commonly used methods, will be added

- `std::set`
- `std::multiset`
- `std::stack`
- `std::priority_queue`
- `std::unordered_set`
- `std::unordered_multiset`
- `std::bitset`
- `std::list`
- `std::forward_list`

3.1.2. Add STL Algorithms

The following algorithms will be added

- `std::ranges::lower_bound`
- `std::ranges::upper_bound`
- `std::ranges::binary_search`
- `std::ranges::sort`
- `std::ranges::stable_sort`
- `std::ranges::max`
- `std::ranges::max_element`
- `std:;ranges::min`
- `std::ranges::min_element`
- `std::ranges::minmax`
- `std::ranges::minmax_element`
- `std::ranges::clamp`

3.1.3. Documentation

Currently, `StdVector` and `StdString` are documented. I will document the functionality of the existing containers (`StdValArray`, `StdDeque` and `StdQueue`) and all the new containers that I will be adding.

The algorithms being added will also be documented, along with usage examples for them.

I will also be documenting the implementation steps for exposing more of the standard library to help future contributors.

3.1.4. Testing

I will be implementing unit tests for all the containers and algorithms being added.

For integration testing on the libcxxwrap.jl component, the automated tests currently work for pull requests. I will update the testing solution such that it works outside of pull requests as well.

3.2. Stretch Goals

If time permits, I would like to make general improvements to the core of CxxWrap, and add more STL containers.

3.2.1. Add Iterator Support

Many STL algorithms depend upon the use of iterators. For this, an iterator type for containers has to be exposed from the C++ side, so that it can be used to call these algorithms from the Julia side.

3.2.2. Add more STL Container Types

These containers have been introduced in C++ 23

- `std::flat_set`
- `std::flat_multiset`

4. Project Details

4.1. Codebase

Currently, the standard library interface is implemented in a single file `StdLib.jl`. Since I will be adding many containers and algorithms, my first step will be modularizing the codebase. I will be splitting it into folders for containers and algorithms respectively, with appropriate files for each of them.

4.2. STL Containers

https://en.cppreference.com/w/cpp/container

I will be going over my plan for implementing STL containers using the example of `std::queue`

To implement the containers listed, I will be taking a two-step approach

4.2.1. libcxxwrap component

The functionalities to be exposed need to be wrapped in a struct on the C++ side.

For the case of `std::queue`, I have exposed the `front`, `push`, `pop` and `size` functionalities.

```cpp
template<typename T>
struct WrapQueueImpl
{
  template<typename TypeWrapperT>
  static void wrap(TypeWrapperT&& wrapped)
  {
    using WrappedT = std::queue<T>;

    wrapped.module().set_override_module(StlWrappers::instance().module());
    wrapped.method("cppsize", &WrappedT::size);
    wrapped.method("push_back!", [] (WrappedT& v, const T& val) { v.push(val); });
    wrapped.method("front", [] (WrappedT& v) -> const T { return v.front(); });
    wrapped.method("pop_front!", [] (WrappedT& v) { v.pop(); });
    wrapped.module().unset_override_module();
  }
};
```

4.2.2. CxxWrap component

The exposed functions need to mapped to the appropriate methods on the Julia interface.

```julia
Base.size(v::StdQueue) = (Int(cppsize(v)),)
Base.push!(v::StdQueue, x) = push_back!(v, x)
Base.first(v::StdQueue) = front(v)
Base.pop!(v::StdQueue) = pop_front!(v)
```

4.3. STL Algorithms

https://en.cppreference.com/w/cpp/algorithm/ranges

I will implement STL algorithm interfaces as constrained algorithms (introduced in C++ 20) using `std::ranges` on the C++ side. I have chosen to do so because these abstract away iterators, and allow for passing the containers directly. This leads to a much cleaner implementation on the Julia side.

Since it is hard to cover all the STL algorithms, I have chosen the ones that are most frequently used.

5. Project Schedule

The proposed schedule has been made keeping in mind the GSoC timeline.

At the end of both the coding periods, I have allocated a buffer week. This will be used in case the project would be behind schedule due to unforeseen circumstances. In case the project is on time, the buffer weeks would be used to implement the stretch goals.

5.1. Community Bonding Period

May 1 - May 26: During this period, I aim to
- Decide and set up a weekly status update method with the mentor
- Further familiarize myself with the codebase
- Add more tests for the existing code
- Update the testing method in `libcxxwrap` to work outside of PRs as well

5.2. First Coding Period

In the first phase, I will be working on the STL containers, their interfaces, documentation and testing. I plan on writing the tests and documentation along with the actual implementations, rather than pushing them towards the end.

May 27 - June 2: `stack` and `priority_queue`

June 3 - June 9: `set` and `multiset`

June 10 - June 16: `unordered_set` and `unordered_multiset`

June 17 - June 23: `list` and `forward_list`

June 24 - June 30: `bitset`

July 1 - July 7: Buffer week

5.3. Midterm Evaluation

July 8 - July 12: Write the mid report and further polish the documentation and tests. Since at this point I will know how the code structure would be, it will be a good time to refactor the code.

5.4. Second Coding Period

In the second phase, I will be working on STL algorithms, along with their documentation and testing.

July 15 - July 21: `max`, `max_element`, `min`, `min_element`

July 22 - July 28: `minmax`, `minmax_element`, `clamp`

July 29 - August 4: `sort`, `stable_sort`

August 5 - August 11: `lower_bound`, `upper_bound`, `binary_search`

August 12 - August 18: Buffer week

5.5. Final Submission

August 19 - August 26: Write the final report and document the process for future contributors

5.6. Availability

I have my summer vacation from 10^{th} May to 1^{st} August, and don't have other commitments over this time period. I will be giving $\approx$ 30 hours per week to this project in this time period, and plan to get the majority of the project done here.

For the remainder of the time period, (2^{nd} August to 26^{th} August), my college will resume and I will be able to give $\approx$ 15 hours per week.

If something does come up that clashes with the timeline, it will be informed in a timely manner and I will ensure that the working hours won't be affected.