Document number: D????R0
Project: Programming Language C++
Audience: Evolution Working Group, Library Evolution Working Group

Antony Polukhin <antoshkka@gmail.com>, <antoshkka@yandex-team.ru>
Alexander Zaitsev <zamazan4ik@tut.by>

Date: 2017-04-05

Changing attack vector of the constexpr_vector

"The easiest way to solve a problem is to deny it exists."

— *Isaac Asimov*

I. Introduction and Motivation

P0597R0 proposed to add a new `std::constexpr_vector` type that is usable in constexpr context. That's going to be a very useful class and many people already wish to have it in Standard Library.

However, users will definitely wish for more:

- Reflections SG would like to have a constexpr_string
- Boost is already trying to invent constexpr associative containers
- constexpr_deque seems to be very useful as it has a handy `pop_front` function
- Users have their own containers and one day they would like to make them `constexpr`
- Numerics SG may wish to have constexpr unbounded integers some day
- constexpr_path?..
- ...

This proposal **is not** an attempt to prevent further work on P0597R0. This proposal is **an attempt to change problem attack vector of the P0597R0** to make it more generic and solve more problems without duplicating each container in the Standard Library.

II. The idea

Instead of providing multiple constexpr containers we can provide a single `std::constexpr_allocator` and mark existing containers with constexpr.

III. Proof of concept

It took roughly 10 man-hours to implement naive `std::constexpr_allocator` as a library only solution and tune `std::vector` to be usable in constexpr context. Now it is possible to write code like the following:

```
constexpr bool vector_testing_constexpr(unsigned size) {
    std::vector<unsigned, constexpr_allocator<unsigned>> compile_time;
    for (unsigned i = 0; i <= size; ++ i)
        compile_time.push_back(i);

    compile_time.emplace_back(0);
    compile_time.pop_back();

    return compile_time.back() == size;
}
```

```
int main() {
    constexpr auto r = vector_testing_constexpr(5);
    static_assert(r, "");
}
```

The proof of concept implementation could be found at https://github.com/ZaMaZaN4iK/constexpr_allocator (all the major changes are in `modif_*` headers).

IV. Challenges and solutions

Following problems were discovered while implementing the proof of concept prototype:

- Standard Library containers miss `constexpr`
  - **Solution:**It is simple to add `constexpr` all around the container declaration
- Standard Library containers have non trivial destructors
  - **Solution 1:** This can be worked around by querying the allocator, checking that it is a constexpr allocator and changing the destructors to trivial ones. That's the solution that was used in the prototype
  - **Solution 2:** This can be fixed in a more general way by allowing non trivial destructors in constant expressions
- Many utility functions and algorithms miss `constexpr` specifiers
  - **Solution:** This is solved by P0202R1
- `try` and `catch` are not allowed in constant expressions
  - **Solution:** Exceptions could not be thrown in constant expression so it seems OK to allow `try` and `catch` in constant expressions that just do nothing
- No way to allocate memory in constant expressions
  - **Solution 1:** That's were the P0597R0 is stepping in. Instead of having a magic `constexpr_vector` that allocates memory, please change it to magic `constexpr_allocator` that allocates memory in constant expressions.
  - **Solution 2:** Prototype used an array of default initialized objects instead of allocating memory and constructing objects in place. Users may use the similar approach for providing their own constexpr allocators.

V. Pros and Cons

Pros of constexpr_allocator approach:

- Extensible - it is simple to add new containers
- Brief - no duplication of existing Standard Library containers
- Follows existing practice - uses allocators for memory allocation, not fuses the allocator behavior into the container behavior
- Future proof - if one day someone invents a constant expression usable `new`, we won't need to deprecate a bunch of constexpr containers. We'll just deprecate a single constexpr allocator
- User friendly - could be used with user defined containers without tedious wrapping and reinventing constexpr allocators from the `std::constexpr_vector`.

Cons of constexpr_allocator approach:

- Touches Allocators - people try to avoid that
- Requires More Work - instead of having a single big `std::constexpr_vector` proposal the `constexpr_allocator` approach requires multiple smaller proposals to cycle trough almost all the subgroups.

VI. References

[P0597R0] "std::constexpr_vector<T>" proposal. Available online at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0597r0.html

[P0202R1] "Add Constexpr Modifiers to Functions in <algorithm> and <utility> Headers" proposal. Available online at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0202r1.html