Introduce uninitialized_storage

Document #:	PXXXXR0
Date:	2021-01-08
Project:	Programming Language C++
Audience:	Library Evolution Group
Reply-to:	Michael Schellenberger Costa
	<mschellenbergercosta@googlemail.com>

1 Introduction

This paper proposes adding a new library type `uninitialized_storage`, that fixes the deficiencies of `aligned_storage` and enables the use during constant evaluation.

While the use cases are sparse this is an essential tool for container developers, for example when developing a SSO optimized container.

2 Motivation and Scope

Currently there are few ways to create uninitialized storage on the stack. The classic approach `aligned_storage` has been widely critized and there is already a call for its deprecation [P1413R2].

Other alternatives are buffers of `std::byte` or `char`, which are aligned to the target type `T`.

However, both solution have fundamental problems with respect to `constexpr` support. Imagine we have a local buffer of uninitialized storage

```cpp
typename std::aligned_storage<sizeof(T), alignof(T)>::type data[N];
// or equivalently
alignas(T) std::byte data[N * sizeof(T)];
```

1. Getting pointers of type `T*` to the buffer.

In order to get access to the storage we need to `reinterpret_cast` to T "'cpp
T* begin() { return std::launder(reinterpret_cast<T*>(data)); }

T* end() { return std::launder(reinterpret_cast<T*>(data)) + N; } "'

None of these are remotely available during constant evaluation. There are only two possible solutions for this:

a. Add a function with a blessed name such as `std::construct_at`
b. Add a blessed type with the correct behavior

2. Creating elements of type `T`

A simplified emplace function for the uninitialized storage may look like this: `cpp        template<typename ...Args>     vo`

Now again this is not possible during constant evaluation. We would need `std::construct_at` for that, but it takes a `T*` for the location, which brings us back to #1 or how to get a pointer.

So this leaves us with two choices. We can either define a blessed function that works with the different existing solutions or a blessed type that does the right thing.

Here we argue, that introducing a blessed function is the worse solution. First of all there are way more things to consider. What are the different allowed input types? `aligned_storage`, `std::byte[N]` are obvious choices but what about `char` or `signed char`?

Also alignment is an easy to misuse paint poitn. We require a lot from a user to simply get an uninitialized buffer.

So the solution here is to propose a new type `std::uninitialized_storage<T, N>`, together with appropriate member function to make it a `constexpr` enabled contiguous range.

3 Impact on the Standard

This proposal is a pure library extension. That said, when introducing this new facility it would be an oportune moment to follow trough with [P1413R2] and deprecate `alligned_storage` for good.

4 Proposed Wording

4.1 Add a new section [specialized.storage] to 20.10.2 [memory.syn] of [N4762] after [specialized.destroy]:

```
// [specialized.storage], Class template uninitialized_storage
template<class T, size_t N>
  struct uninitialized_storage;
```

4.2 Add 25.11.10 [specialized.storage] to [N4762]:

```
__25.11.10 Class template uninitialized_storage__
// [specialized.storage], uninitialized_storage
template<class T, size_t N>
  struct uninitialized_storage;
```

5 Implementation Experience

— Given that this requires compiler support there is currently no implementation experience. If the proposal gathers sufficient support, we will try to implement it in clang and libc++.

6 References

[N4762] Richard Smith. 2018-07-07. Working Draft, Standard for Programming Language C++.
https://wg21.link/n4762

[P1413R2] 2019. Deprecate std::aligned_storage and std::aligned_union.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1413r2.pdf