Rename “_default_init” Functions, Rev0

For C++20 we introduce the term “default initialization” as part of the name for some functions, such as make_unique_default_init(), make_shared_default_init(), allocate_shared_default_init().

According to the NB comment DE002 this name in the API should be changed because it is confusing for ordinary programmers. It sounds like being safe (initialize data with default values) but in fact is not (because default initialization does not always initialize).

As with C++20 we establish this API name part as a common base for all future ways to have default initialization, we think it is important to propose a better name right now before we ship C++20.

In the corresponding LEWG discussion the vote was to come up with a better name and restrict it to trivially constructible types. The nice effect is that if code like the following compiles:

```cpp
auto p1 = std::make_unique_uninitialized<double[]>(1024); // indetermined value
struct X { double data[1024]; }; // indetermined value
auto p2 = std::make_shared_uninitialized<X>();
```

we have successfully documented the intent of not initializing the data of the allocated memory. This code is self-explanatory.

On the other hand

```cpp
auto p3 = std::make_shared_uninitialized<vector<int>>(); // compile-time error
```

will not compile. This means that generic code might need an `if constexpr`, which was discussed in LEWG and considered to be OK because it will only rarely occur in code written by experts.

There was a poll about better names and because we now know that we definitely don’t initialize we can express that through the name. LEWG voted the following favorites:

20 make_shared_uninitialized
11 make_shared_trivially_constructed
17 make_shared_no_init
4 make_shared_default_init

For the members of the German NB, “_uninitialized” was the clear favorite, so that we propose that name.
Proposed Wording

(All against N4835)

In 20.10.2 Header <memory> synopsis [memory.syn]

Rename as follows:

```cpp
namespace std {
    ...
    template<class T>
    unique_ptr<T> make_unique_default_inituninitialized(); // T is not array
    template<class T>
    unique_ptr<T> make_unique_default_inituninitialized(size_t n); // T is U[N]
    template<class T, class... Args>
    unspecified make_unique_default_inituninitialized(Args&&...) = delete; // T is U[N]
    ...
    template<class T>
    shared_ptr<T> make_shared_default_inituninitialized(); // T is not U[]
    template<class T, class A>
    shared_ptr<T> allocate_shared_default_inituninitialized(const A& a); // T is not U[]
    template<class T>
    shared_ptr<T> make_shared_default_inituninitialized(size_t N); // T is U[]
    template<class T, class A>
    shared_ptr<T> allocate_shared_default_inituninitialized(const A& a, size_t N); // T is U[]
};
```

20.11.4 Creation [unique.ptr.create]

Change:

```cpp
template<class T> unique_ptr<T> make_unique_default_inituninitialized();
Remarks: This function shall not participate in overload resolution unless
is_trivially_constructible_v<T> is true.

Constraints: T is not an array.

Returns: unique_ptr<T>(new T).
```

```cpp
template<class T> unique_ptr<T> make_unique_default_inituninitialized(size_t n);
Remarks: This function shall not participate in overload resolution unless
is_trivially_constructible_v<T> is true.

Constraints: T is an array of unknown bound.

Returns: unique_ptr<T>(new remove_extent_t<T>[n]).
```

```cpp
template<class T, class... Args> unspecified
make_unique_default_inituninitialized(Args&&...)= delete;
Remarks: This function shall not participate in overload resolution unless
is_trivially_constructible_v<T> is true.

Constraints: T is an array of known bound.
```

20.11.5.6 Creation [util.smartptr.shared.create]:

The common requirements that apply to all make_shared, allocate_shared, make_shared_default_inituninitialized, and allocate_shared_default_inituninitialized overloads, unless specified otherwise, are described below.

```cpp
... template<class T, ...>
shared_ptr<T> make_shared_default_inituninitialized(args);
template<class T, class A, ...>
shared_ptr<T> allocate_shared_default_inituninitialized(const A& a, args);
Remarks: The make_shared_uninitialized and allocate_shared_uninitialized templates shall not participate in overload resolution unless is_trivially_constructible_v<T> is true.
```

Effects: ... The allocate_shared and allocate_shared_default_inituninitialized templates...
template<class T>
shared_ptr<T> make_shared_default_inituninitialized();

template<class T, class A>
shared_ptr<T> allocate_shared_default_inituninitialized(const A& a);

Remarks: These functions shall not participate in overload resolution unless

is_trivially_constructible_v<T> is true.

Constraints: T is not an array of unknown bound.

Returns: A shared_ptr to an object of type T.

Example:
struct X { double data[1024]; };
shared_ptr<X> p = make_shared_default_inituninitialized<X>();

// shared_ptr to a default-initialized X, where each element in X::data has an indeterminate value
shared_ptr<double[1024]> q = make_shared_default_inituninitialized<double[1024]>().

// shared_ptr to a default-initialized double[1024], where each element has an indeterminate value

—end example

template<class T>
shared_ptr<T> make_shared_default_inituninitialized(size_t N);

template<class T, class A>
shared_ptr<T> allocate_shared_default_inituninitialized(const A& a, size_t N);

Remarks: These functions shall not participate in overload resolution unless

is_trivially_constructible_v<T> is true.

Constraints: T is an array of unknown bound.

Returns: A shared_ptr to an object of type U[N], where U is remove_extent_t<T>.

Example:
shared_ptr<double[]> p = make_shared_default_inituninitialized<double[]>(1024);

// shared_ptr to a default-initialized double[1024], where each element has an indeterminate value

—end example

Feature Test Macro

This is fixing new features and does not need a feature test macros in itself.

Acknowledgements

Thanks to a lot of people who discussed the issue, proposed information and possible wording.