Document	P1734R0
Date	2019-06-14
Author	CJ Johnson <johnsoncj@google.com>
Audience	Evolution Working Group (EWG), Core Working Group (CWG)

Defaultable default constructors and destructors for all unions

Proposal

This paper proposes simplifying the rules around defaulted default constructors and destructors of unions without breaking backwards compatibility. In short, any union, regardless of the types of its members, should be given a well-formed default constructor and/or destructor when implicitly or explicitly defaulted.

Destructors

Currently, the only way to default the destructor of a union is to ensure all nonstatic members of the union are trivially destructible. If any nonstatic member is nontrivially destructible, the language *requires* the programmer to implement the destructor.

However, even if one or more nonstatic members of a union have nontrivial destructors, there is absolutely no requirement that any of those destructors are called. Looking back at N2544 [1], the rationale was that since the compiler would not be able to automatically pick the correct member to destroy, the language would simply leave it up to the programmer to write. This rule set sounds nice on the surface, but it fails to enforce correctness and at the same time it limits what programmers can express with unions. Take this example union definition:

```
template <typename T>
union U {
  T t = T();
  ~U() {}
};
```

This union is perfectly valid for all types for which `T t = T();` is valid, including nontrivially destructible `T`s. And yet, despite that, `~U()` never calls `~T()`. Is that a problem? Well it depends on the types with which it is used. It might be programmer error, or it might not be. There's no way to tell by simply looking at the example template.

The point of this illustration is that the initial goal of "force programmers to write the ones that the compiler cannot default" from N2544 [1] did not pan out. The language does not help programmers write better code simply by forcing some unions to have user defined destructors.

That being the case, I see no reason to continue to enforce this restriction on the language. Instead of forcing programmers to pay for a nontrivial destructor that they may not be using (Don't pay for what you don't use!), the language should be able to provide a default in the absence of a user provided implementation. This would not change the behavior of any existing unions but would instead allow the rules of newly-created unions going forward to be simpler.

- If a user provides a destructor implementation, be it inline or out-of-line
  - Use the user provided destructor, as always
- If the destructor is implicitly defaulted or inline explicitly defaulted
  - Default the destructor as trivial
- If the destructor is out-of-line explicitly defaulted
  - Default the destructor as nontrivial, but do not call the destructor of any of the members

Default constructors

Similarly, user provided default constructors on unions are not required to call the default constructor on any member, irrespective of triviality. For example:

```
template <typename T>
union U {
  T t;
  U() {}               // Doesn't call `T()`
  ~U() { t.~T(); }     // Calls `~T()`
};
```

If `~T()` is nontrivial, this union has a chance of invoking undefined behavior. And yet, this union definition is well formed in the language as it is today. Just like with destructors, forcing users to pay for a nontrivial default constructor (when one or more union member is nontrivially default consructible) does not actually enforce correctness, thus there's no reason for the language to include such a rule.

- If a user provides a default constructor implementation, be it inline or out-of-line
  - Use the user provided default constructor, as always
- If the default constructor is implicitly defaulted or inline explicitly defaulted
  - Default the default constructor as trivial, leaving no active nonstatic member
- If the default constructor is out-of-line explicitly defaulted
  - Default the default constructor as nontrivial, but do not call the default constructor of any of the members, leaving no active nonstatic member

The rules as stated here sound the same for both the default constructor and the destructor. However, that's not the whole picture. Trivial default construction only takes place in the event of default initialization. More rules are needed to account for other forms of initialization.

For value initialization of trivially default constructible unions, to maintain backwards compatibility, first every byte (padding or otherwise) in the union instance should be set to `0`. Then, the first nonstatic member should be value initialized. In doing so, the first nonstatic member becomes the active union member.

For aggregate initialization, the process should mirror value initialization. First every byte is set to `0`. Then, the first nonstatic member should be initialized with whatever initializer was provided, making it the active member.

Similarly, designated initializers should behave as one would expect. First set every byte to `0`. Then, initialize the designated member with the provided initializer, making it the active member.

Open questions

- Should other special member functions be defaultable in the face of nonstatic members with nontrivial implementations?

  Default constructors and destructors are unique among the special member function set. Trivial construction and trivial destruction are actually just that: trivial. The term is a bit overloaded when it comes to copy and move. For something to be trivially copyable, one might assume it must be stateless. That isn't the case, however. Trivial copy implies, in many cases, actual work to be done.

  It may be perfectly reasonable to perform trivial copy/move on unions with nontrivially copyable/movable nonstatic members. It would be surprising, but being that basically all union behavior is surprising, this would come as no exception and may even be desirable behavior.

  That said, the answer to this question is not obvious. Until there is clear consensus for making copy/move always defaultable, this proposal is keeping the status quo.

References

[1] N2544: Unrestricted Unions