



IMPLEMENT MISSING C++26 FEATURES IN CLANG

Author:
- Muhammad Bassiouni

Mentors:
- Vassil Vassilev
- Aaron Jomy



Where we left off

01

[P1814R0] paper

Started working on [P1814R0] CTAD for alias templates paper.

02

Filed a [PR](#)

Tried to work on one edge case for CTAD alias templates.

```
template<typename T>  
struct A { A(T){} };  
  
template<typename T>  
using Proxy = T;
```

```
template<typename T>  
using C = Proxy<A<T>>;  
  
C test{ 42 };
```

What's new

01

CTAD PR got blocked!

There was a debate about the equivalence of alias templates.

- Only GCC implements that.
- There's an open issue for that and it's not finalized:
[CWG2186](#)

02

It can't be fixed soon!

After talking to Aaron, Hubert, Vlad, Corentin and other Clang maintainers we figured that we need to submit new wording to be discussed with the standard committee.

03

Not so fast! It's not a priority :\

It won't be picked for discussion in the first place as it's not a priority.

Submitting a wording would take too much time frame to fit the fellowship schedule!

What's new

04

Pivoted to `constexpr` math!

The next best option was C++23/26 `constexpr` math papers.

Half of the effort is in Clang for built-ins and I worked on the LLVM libc side of things already!

05

Also not so fast!

There's a [PR](#) for qualifying `__builtin_exp` as `constexpr`.

We want this merged before I can start my work because maintainers need to agree on the approach.

Things are moving too slow for us

06

Yet another pivot!

I had to choose another paper to implement!

Decided to move forward with C++29 default postfix operator paper.

What's in motion

[P3668R4] Defaulting Postfix Increment and Decrement Operations

- Filed a PR to implement the paper wording.
- Ported to C++03. Hence we need to address many language features availability like `constexpr` and other qualifiers.
- Working with reviewers to finalize the implementation.

```
class foo{
    int member;
public:
    constexpr foo& operator++(){
        ++member;
        return *this;
    }
    constexpr foo operator++(int){
        auto copy{*this};
        this->operator++();
        return copy;
    }
};
```

```
class foo{
    int member;
public:
    constexpr foo& operator++(){
        ++member;
        return *this;
    }
    constexpr foo operator++(int) = default;
};
```

```
struct S0 {
    int v;
    S0& operator++() { ++v; return *this; }
    S0 operator++(int) = default;
};

struct S1 {
    int v;
    S1& operator++() { ++v; return *this; }
    S1 operator++(this S1&, int) = default;
};

struct S2 {
    int v;
    S2& operator++() { ++v; return *this; }
};
S2 operator++(S2&, int) = default;
```

THANK YOU



Feel free to ask questions or give feedback!

FIND ME AT

muhammad.m.bassiouni@gmail.com

github.com/bassiounix

linkedin.com/in/bassiounix