CppAlliance Fellowship at Compiler-Research
Implement missing C++26 features in Clang

1 Contact Details

- **Name:** Muhammad Bassiouni
- **GitHub username:** bassiounix
- **Email:** muhammad.m.bassiouni@gmail.com

2 Project Info

- **Project Name:** Implement missing C++26 features in Clang
- **Selected Papers:**
  - if declaration in C2y: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3356.htm
  - CTAD alias templates: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1814r0.html

3 Abstract

The C and C++ Working Groups proposed several papers for different standards of the languages to introduce new features, 2 of which are the *if declaration in C2y* and *CTAD alias templates in C++20*. The *if declaration* proposes a C++-style if statement with a declaration support; while the *CTAD alias templates* propose a way to simplify the use of alias templates to make CTAD useful for all aspects of C++. The goal of this proposal is to complete the implementation of these features in Clang

4 Benefits to Community

4.1 If declaration in C2y

This feature makes C code easier to read and write. It lets programmers declare a variable inside an if statement, so the variable is only used where it is needed. This reduces mistakes and keeps code cleaner. Adding it to Clang helps developers use modern C features earlier and keeps Clang up to date with the C2y standard. There's nearly nobody working on C2y features in Clang, so this will be a step forward for C support in Clang.

4.2 CTAD for alias templates in C++20

This feature lets the compiler automatically figure out template types even when using alias templates. Today, this often fails or needs extra code, which is confusing. Supporting it in Clang makes code shorter, simpler, and more consistent with how normal templates already work. It also improves compatibility with the C++20 standard and other compilers.

5 Implementation Plan

5.1 if declarations for C2y

Since this feature is taken from C++ directly, there exists a working implementation for it in Clang. However, it's very specific to C++ behavior and rules such as allowing a second declaration inside the if condition, which

is not allowed in C2y. Most of the work here would be to refactor the existing code for *if declarations* to work for both C and C++ since there exist shared behavior between the 2 standards. This feature is expected to work mostly as a syntax sugar for the existing code, so most of the work would be to transform the new declaration into existing AST nodes and reuse the existing code for the rest of the compilation process. This could change depending on the final design of the implementation, but this is the expected plan for it.

- Phase 1: investigate the existing implementation for *if declarations* in Clang.
- Phase 2: design a refactored version that can work for both C and C++ without mixing the 2 wordings and only share the common syntax between them.
- Phase 3: start implementing and allowing declaration in the if condition using the semi-colon as a separator between the declaration and the condition expression.
- Phase 4: implement direct declaration and initialization inside the if condition which translates to the previous phase without an extra condition expression.

Part of all implementation phases would be to write tests for the new feature and make sure it works in all contexts and with all possible cases.

5.2 CTAD for alias templates in C++20

There exists a basic support for this feature in Clang, however, it's very limited and only works for some cases. The work here would be to extend the existing implementation to support all cases of alias templates and make it work in all contexts. Some examples of missed cases are:

- Setting the right value for the __cpp_deduction_guides feature macro in both C++17 and C++20.
- Support for alias templates with non-type template parameters. Issue#167513
- Clang crashes on CTAD alias template code. Issue#139442

and many more cases that are expected to be found in the implementation code. The work here would be to investigate the existing implementation, find the missed cases, and implement them with tests.
This work and investigation is a stretch goal for this proposal.

6 Timeline

This is a rough timeline for the implementation of both features, it could change depending on the design and implementation details that are expected to be found during the implementation process. The timeline is designed to allow enough time for investigation and design before starting the implementation, and also to allow enough time for testing and fixing any issues that might be found during the implementation process. Also taking in mind that code review may take time depending on the availability of code reviewers.

- **Community Bonding Period (Before Coding Starts):**
  - Get familiar with the Clang codebase and the existing implementation for the 2 features.
  - Communicate with the mentors and other contributors to discuss the design and implementation plan for both features.
  - Investigate the existing implementation for both features and find the shared code between them that can be refactored to support both C and C++ without mixing the 2 wordings.
  - Investigate the existing implementation for CTAD alias templates and find as many missed cases as possible that need to be implemented.
- **May 25 - June 30:**
  - Identify the common code between the existing implementation for *if declarations* in C and C++.
  - Refactor the existing implementation for *if declarations* to work for both C and C++ without mixing the 2 wordings and only share the common syntax between them.

- **July 1 - July 30:**
  - Implement the third phase of *if declarations* which is to allow declaration in the if condition using the semi-colon as a separator between the declaration and the condition expression.
  - Write tests for the new feature and make sure it works in all contexts and with all possible cases.
- **August 1 - August 30:**
  - Implement the fourth phase of *if declarations* which is to allow direct declaration and initialization inside the if condition which translates to the previous phase without an extra condition expression.
  - Write tests for the new feature and make sure it works in all contexts and with possible cases.
  - **Deliverable:** Full implementation for *if declarations* in `C2y` and all tests for it.
- **September 1 - September 30** *(Stretch Goal)*:
  - Investigate the existing implementation for CTAD alias templates and find as many missed cases as possible that need to be implemented.
  - Implement the missed cases for CTAD alias templates that are found during the investigation phase.
  - Write tests for the new cases and make sure they work in all contexts and with all possible cases.
  - **Deliverable:** Extended support for CTAD alias templates in `Clang` to support as many cases and contexts as possible.
- **October 1 - October 25:**
  - Fix any issues that might be found during the implementation process for both features.
  - Finalize the implementation and make sure it works in all contexts and with all possible cases.
  - Go through related issues and make sure they are fixed and closed.
  - Fix divergences from other compilers implementation if there exist any.
  - **Deliverable:** Final implementation for both features and all tests for them.

7 Expected Final Results

- Fully support and implement the *if declaration* feature in `Clang` for `C2y` without affecting existing functionality and performance.
- Extend the existing support for CTAD alias templates in `Clang` to support as many cases and contexts as possible *as a stretch goal*.

8 About Me

I'm a software engineer and **maintainer of LLVM C Library**. I'm also a GSoC mentor this year, working on integrating LLVM `libc` math routines into `compiler-rt` floating-point builtins. In the past, I've worked in backend web development with Node.js and Java, but my motivation comes from my contributions to open-source projects, particularly in the compilers and low level space.

For the past 16 months, I've been contributing to LLVM, where I'm proposing and developing RFCs, improving existing workflows, and collaborating across projects like LLVM `libc` and `Clang`. I'm also officially maintaining and developing parts of LLVM `libc` like constexpr math, `C11` bounds-checking interfaces and unicode conversion functions.

Some highlights of my work include:

- Working with Google engineers to unblock LLVM `libc++` streams by implementing wctype header and unicode conversion functions in LLVM `libc`.
- Proposing an RFC for Annex K of `C11`, bounds-checking interfaces, which I am now implementing in collaboration with Arm engineers.

- Contributing to `C++26` `constexpr` math function implementations through refactoring 493 math functions to header-only with constant evaluation support and code sharing with other LLVM projects like `Clang`, LLVM, and `libc++`.
- Review patches and PRs from other maintainers and contributors, and help new contributors to navigate and contribute to LLVM project, building a community around the project.
- Adding BFloat16 support macros in `Clang`.

Along with various bug fixes.

8.1 Prior Contributions to LLVM:

- **Partnered with Google engineers** to build `libc++` streams by implementing `wctype` in `libc`. RFC
- **Partnered with Arm engineers** to add C11 bounds-checking interfaces in LLVM `libc`. RFC
- Implemented C++26 `constexpr` math functions by refactoring **493** math functions to header-only with constant evaluation support in LLVM `libc`. RFC
- Improved build accuracy and consistency by fixing LLVM `libc` header generation. PR
- Extended `Clang` capabilities by adding `__bf16` type support macros with number literal suffix. PR
- Optimized LLVM `IR` performance by fixing missed `i128` split optimization. PR

8.2 Prior Contributions to general Open Source projects:

- PR#3: Port GNOME Swap Finger Gestures extension to new versions of GNOME.
- PR#22: Add detection support for GeForce RTX 3050 Ti Mobile GPU to GhostBSD XConfig script.

9 Availability

I plan to dedicate 40 hours per week to this project, and I am flexible with my schedule to accommodate the needs of the project and the mentors. I will be available for regular communication and collaboration with the mentors and other contributors throughout the duration of the project. The only conflicting commitments I have now is that I'm a GSoC mentor this year in LLVM, but it shouldn't impact my availability for the project.

10 Planned AI Usage

I plan to use AI tools as low as possible in only understanding and solving specific and niche problems during the program. I already depend on reading the code, stepping through the code using the debugger to understand the existing flow, looking at other implementations of similar features in other compilers, asking people about unknown topics, and going through resources for the related areas either in docs, articles or videos, thus minimizing my reliance on AI tools.