C Standards and Implementations

C standards are the set of rules which define the features, syntax, meaning of code, or one can say, the grammar of C language. These standards ensure compatibility and portability of C programs across different compilers and platforms. Standards are generally provided in the form of a document by the standardization committee.

These rules are then implemented by a software that converts your C language code to code executable by your computer. This software is called a **Compiler**. A compiler checks whether the given code is according to the rules or standard. If it is not, it does not convert your code and shows error. There is also not a single C compiler, but many of them are available for C such as **GCC (GNU Compiler Collection), MSVC (Microsoft Visual C++), Clang, Oracle C**, etc.

Why were C Standards needed?

Although provided by different vendors, all C compilers currently are implemented according to the C standard, so a C program that runs on GCC should also run on Clang. But it was not the case in earlier times.

Since C was developed by Dennis Ritchie in 1972, it was not standardized. Due to this, when it got popular, many different vendors created their own C compliers. However, due to lack of defined set of rules, each compiler implemented their own version of C which caused programs written for one compiler often failed to work on another leading to portability issues.

Standardization of C

To resolve portability issue, the first informal standard of C was released as a book named **"K & R C"** by Brian **K**ernighan and Dennis **R**itchie in 1978. Later on, American National Standard Institute (ANSI) took the role as standardization committee and released another standard **C89 (or ANSI C)** in 1989. Since, then it released various standards to enhance and modernize the language:

- **C99 (1999):** Added support for inline functions, variable-length arrays, and new data types.
- **C11 (2011):** Focused on multithreading, Unicode support, and safety features.
- **C17 (2017):** A minor revision to C11, improving clarity and fixing issues.
- **C23 (2023):** The latest standard, introducing features to improve usability and support for modern computing needs.

Different compilers were also created according to these standards. We can check which standard the given version of compiler implements by looking at its documentation. Also, newer versions keep the backward compatibility to the previous versions except for the deprecated features.

Conclusion

In conclusion, standards are the set of rules that define the language while the compilers are the software that validate these rules and convert your code to binary code.

Marked as Read

Report An Issue