`import std;` and stream macros

Introduction of global constants as substitutes for stream macros in standard library modules

Document #: P3208R0
Date: 2024-04-16
Reply-to: Sunghyun Min
<hubble.stein@gmail.com>
Audience: WG21

Abstract

In C++23, `import std;` does not expose macros including C streams(`stdin, stdout, stderr`). Users have to include the corresponding header files. To improve the usage of standard library modules, this paper proposes global constants to address this issue: `std::in, std::out, and std::err`.

1. Rationale

In C++23, `import std;` was introduced to facilitate the use of standard libraries. Users can access all library features by one line and it helps to reduce compilation time. However, modules don't expose macros intentionally. Therefore, users have to figure out the issue and eventually include the matching header files again even though they are contained in std module [StackOverflow]. This method can be a solution. However, P2465R3 suggests that non-macro approaches should be utilized other than feature test macros. Other efforts have been done to try to resolve macro issues: [P2654R0], [P2883R0], and [P2884R0]. This paper proposes global constants, `std::in, std::out, and std::err` as substitutes for stream macros to improve the usability of `std` module.

2. Motivation and Proposal

C++23	Proposed

`#include <cstdio> // for stderr`
`// import <cstdio>; // alternative`
`import std;`
 	
`int main()`
`{`
`  std::println(stderr, "...");`
`}`
	`import std;`
	
	`int main()`
	`{`
	`  std::println(std::err, "...");`
	`}`

Of course, there is `std::println(std::cerr, "...")` but `std::print` uses `stdout` by default, so it is reasonable to use `stderr` for error output, which is slightly faster than the counterpart (see Appendix).

Instead of providing additional APIs to hide or wrap stream macros, e.g., `std::eprint`, a fundamental approach is necessary for other use cases. This paper proposes `std::in, std::out, and std::err` as substitutes for `stdin, stdout, and stderr`, respectively. They are global constants with the type of `std::FILE* const`.

Proposed (succinct)	Alternative (maybe familiar but repetitive)
`std::in`	`std::stdin`
`std::out`	`std::stdout`
`std::err`	`std::stderr`

Instead of global constants, inline functions returning pointers can be considered if they are preferred for implementation.

3. Stream Macros in GCC and MSVC

The stream macros in major compilers are defined as follows.

3.1 GCC 13.2.1

C streams are global variables in GCC and redefined as macros. “If any of these `std::FILE*` lvalue is modified, subsequent operations on the corresponding stream result in unspecified or undefined behavior.” (cppreference)
Even though the proposed streams are constant pointers, pointees are still mutable.

```
// filename: stdio.h

/* Standard streams.  */
extern FILE *stdin;             /* Standard input stream.  */
extern FILE *stdout;            /* Standard output stream.  */
extern FILE *stderr;            /* Standard error output stream.  */
/* C89/C99 say they're macros.  Make them happy.  */
#define stdin stdin
#define stdout stdout
#define stderr stderr
```

3.2 MSVC 2022 Version 17.9.2

Standard streams in MSVC are global constant pointers. (Microsoft)

```
// filename: corecrt_wstdio.h

_ACRTIMP_ALT FILE* __cdecl __acrt_iob_func(unsigned _Ix);

#define stdin  (__acrt_iob_func(0))
#define stdout (__acrt_iob_func(1))
#define stderr (__acrt_iob_func(2))
```

4. Possible Interface

4.1 Option A: `std` module only

This option makes stream constants available only in the modules world.

```
// std module file
// ...
export module std;
namespace std
{
  export extern std::FILE* const in;
  export extern std::FILE* const out;
  export extern std::FILE* const err;
}
```

4.2 Option B: a new header file and `std` module

In this option, users can include the header file separately to use stream constants if necessary without importing standard library modules.

```
// filename: cstream
// ...

namespace std
{
  extern std::FILE* const in;
  extern std::FILE* const out;
  extern std::FILE* const err;
}
```

```
// std module file
module

// ...
#include <cstream>

export module std;

// ...
namespace std
{
  export using std::in;
  export using std::out;
  export using std::err;
}
```

5. Conclusion

The support of macros including C streams is a missing block in `import std;` for good reason and there are workarounds. This proposal probably makes C++ more complicated and is a small deviation from C. However, by introducing substitute constants for stream macros, not only can the convenience of `import std;` be improved but also these macros can be replaced. Hopefully, `import std;` is supposed to be sufficient to consume standard libraries in the future version of the language after other macro issues will have been addressed.

6. Acknowledgments

Thanks to Arthur O'Dwyer for feedback.

7. References

[P2465R3]
Standard Library Modules std and std.compat
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2465r3.pdf

[P2654R0]
Macros And Standard Library Modules
import should suffice
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2654r0.pdf

[P2883R0]
offsetof Should Be A Keyword In C++26
Supporting standard C++23 macros in module std
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2883r0.pdf

[P2884R0]
assert Should Be A Keyword In C++26
Supporting standard C++23 macros in module std
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2884r0.pdf

[StackOverflow]
How to use standard library macros with std module in C++23
https://stackoverflow.com/questions/75041883/how-to-use-standard-library-macros-with-std-module-in-c23

Appendix

The following code was tested using MSVC and GCC. Error messages were redirected and discarded.

```
#ifdef _MSVC_LANG
#pragma comment(lib, "Shlwapi.lib")
#endif

#include <benchmark/benchmark.h>
#include <cstdio>

#ifdef _MSVC_LANG
import std;
#else
#include <iostream>
#include <print>
#endif

constexpr auto msg{ "is not an integer. Please, enter again." };
constexpr const double num{ 1.23 };

static void test_fprintf(benchmark::State& s)
{
    for (auto _ : s)
        std::fprintf(stderr, "%.2f %s\n", num, msg); // print trailing zeros by default
}
BENCHMARK(test_fprintf);

static void print_stderr(benchmark::State& s)
{
    for (auto _ : s)
        std::println(stderr, "{} {}", num, msg);
}
BENCHMARK(print_stderr);

static void print_std_cerr(benchmark::State& s)
{
    std::ios_base::sync_with_stdio(true);
    for (auto _ : s)
        std::println(std::cerr, "{} {}", num, msg);
}
BENCHMARK(print_std_cerr);

static void print_std_cerr_no_sync(benchmark::State& s)
{
    std::ios_base::sync_with_stdio(false);
    for (auto _ : s)
        std::println(std::cerr, "{} {}", num, msg);
}
BENCHMARK(print_std_cerr_no_sync);

BENCHMARK_MAIN();
```

A.1 MSVC

Compiler version and options

```
Microsoft Visual Studio Community 2022 (64-bit) - Preview
Version 17.10.0 Preview 3.0

built inside the IDE using vcpkg with the following options:
```

```
/permissive- /ifcOutput "x64\Release\" /GS /GL /W3 /Gy /Zc:wchar_t /Zi /Gm- /O2 /sdl
/Fd"x64\Release\vc143.pdb" /Zc:inline /fp:precise /D "NDEBUG" /D "_CONSOLE" /D
"_UNICODE" /D "UNICODE" /errorReport:prompt /WX- /Zc:forScope /Gd /Oi /MD /std:c++latest
/FC /Fa"x64\Release\" /EHsc /nologo /Fo"x64\Release\" /Fp"x64\Release\
benchmark_stderr_std_cerr.pch" /diagnostics:column
```

Run script

```
benchmark_stderr_std_cerr.exe --benchmark_out=b.txt --benchmark_out_format=console > nul
2>&1
```

Outcome (CPU: i5-12500)

```
Run on (12 X 2995 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x6)
  L1 Instruction 32 KiB (x6)
  L2 Unified 1280 KiB (x6)
  L3 Unified 18432 KiB (x1)
-------------------------------------------------------------------
Benchmark                         Time             CPU   Iterations
-------------------------------------------------------------------
test_fprintf                   579 ns          343 ns      1866667
print_stderr                  1092 ns          752 ns      1246609
print_std_cerr                1208 ns          698 ns       896000
print_std_cerr_no_sync        1212 ns          774 ns       746667
```

A.2 GCC on Linux

Compiler version and options

```
Linux fedora 6.8.5-201.fc39.x86_64

gcc version 14.0.1 20240405 (experimental) (GCC)

g++ main.cpp -std=c++23 -O3 -DNDEBUG -lfmt -lbenchmark -lpthread
```

Run script

```
./a.out --benchmark_out=b.txt --benchmark_out_format=console >/dev/null 2>&1
```

Outcome (CPU: i5-12500)

```
 Run on (12 X 4600 MHz CPU s)
 CPU Caches:
   L1 Data 48 KiB (x6)
   L1 Instruction 32 KiB (x6)
   L2 Unified 1280 KiB (x6)
   L3 Unified 18432 KiB (x1)
 Load Average: 0.55, 0.82, 0.78
 ***WARNING*** CPU scaling is enabled, the benchmark real time measurements may be noisy
and will incur extra overhead.
 -------------------------------------------------------------------
 Benchmark                         Time             CPU   Iterations
 -------------------------------------------------------------------
 test_fprintf                   179 ns          179 ns      3937211
 print_stderr                   222 ns          222 ns      3158193
 print_std_cerr                 251 ns          251 ns      2793033
 print_std_cerr_no_sync         242 ns          242 ns      2892901
```