std::longjmp

Defined in header <csetjmp>

```
              void longjmp( std::jmp_buf env, int status );     (until C++17)
[[noreturn]] void longjmp( std::jmp_buf env, int status );     (since C++17)
```

Loads the execution context env saved by a previous call to setjmp. This function does not return. Control is transferred to the call site of the macro setjmp that set up env. That setjmp then returns the value, passed as the status.

If the function that called setjmp has exited, the behavior is undefined (in other words, only long jumps up the call stack are allowed)

No destructors for automatic objects are called. If replacing of std::longjmp with `throw` and setjmp with `catch` would execute a non-trivial destructor for any automatic object, the behavior of such std::longjmp is undefined.

Parameters

env	–	variable referring to the execution state of the program saved by `setjmp`
status	–	the value to return from setjmp. If it is equal to `0`, `1` is used instead

Return value

(none)

Notes

longjmp is the mechanism used in C to handle unexpected error conditions where the function cannot return meaningfully. C++ generally uses exception handling for this purpose.

Example

Run this code

```
#include <iostream>
#include <csetjmp>

std::jmp_buf my_jump_buffer;

[[noreturn]] void foo(int count)
{
    std::cout << "foo(" << count << ") called\n";
    std::longjmp(my_jump_buffer, count+1);  // setjmp() will return count+1
}

int main()
{
    volatile int count = 0; // modified locals in setjmp scope must be volatile
    if (setjmp(my_jump_buffer) != 5) { // equality against constant expression in an if
        count = count + 1; // ++count, count += 1, etc on 'volatile'-qualified
                           // left operand are deprecated since C++20 (P1152)
        foo(count); // This will cause setjmp() to exit
    }
}
```

Output:

```
foo(1) called
foo(2) called
foo(3) called
foo(4) called
```

See also

setjmp	saves the context (function macro)

C documentation for **longjmp**