std::div, std::ldiv, std::lldiv

Defined in header <cstdlib>

std::div_t div(int x, int y);	(1)	(constexpr since C++23)
std::ldiv_t div(long x, long y);	(2)	(constexpr since C++23)
std::lldiv_t div(long long x, long long y);	(3)	(since C++11) (constexpr since C++23)
std::ldiv_t ldiv(long x, long y);	(4)	(constexpr since C++23)
std::lldiv_t lldiv(long long x, long long y);	(5)	(since C++11) (constexpr since C++23)
Defined in header <cinttypes>		
std::imaxdiv_t div(std::intmax_t x, std::intmax_t y);	(6)	(since C++11)
std::imaxdiv_t imaxdiv(std::intmax_t x, std::intmax_t y);	(7)	(since C++11)

Computes both the quotient and the remainder of the division of the numerator x by the denominator y.

Overload of std::div for std::intmax_t is provided in <cinttypes> if and only if std::intmax_t is an extended integer type.	(since C++11)
The quotient is the algebraic quotient with any fractional part discarded (truncated towards zero). The remainder is such that `quot * y + rem == x`.	(until C++11)
The quotient is the result of the expression `x/y`. The remainder is the result of the expression `x%y`.	(since C++11)

Parameters

x, y - integer values

Return value

If both the remainder and the quotient can be represented as objects of the corresponding type (`int`, `long`, `long long`, std::imaxdiv_t, respectively), returns both as an object of type std::div_t, std::ldiv_t, std::lldiv_t, std::imaxdiv_t defined as follows:

std::div_t

```
struct div_t { int quot; int rem; };
```

or

```
struct div_t { int rem; int quot; };
```

std::ldiv_t

```
struct ldiv_t { long quot; long rem; };
```

or

```
struct ldiv_t { long rem; long quot; };
```

std::lldiv_t

```
struct lldiv_t { long long quot; long long rem; };
```

or

```
struct lldiv_t { long long rem; long long quot; };
```

std::imaxdiv_t

```
struct imaxdiv_t { std::intmax_t quot; std::intmax_t rem; };
```

or

```
struct imaxdiv_t { std::intmax_t rem; std::intmax_t quot; };
```

If either the remainder or the quotient cannot be represented, the behavior is undefined.

Notes

Until C++11, the rounding direction of the quotient and the sign of the remainder in the built-in division and remainder operators was implementation-defined if either of the operands was negative, but it was well-defined in std::div.

On many platforms, a single CPU instruction obtains both the quotient and the remainder, and this function may leverage that, although compilers are generally able to merge nearby `/` and `%` where suitable.

Example

Run this code

```
#include <string>
#include <cmath>
#include <cstdlib>
#include <iostream>

std::string itoa(int n, int base /*[2..16]*/)
{
    std::string buf;
    std::div_t dv{}; dv.quot = n;
    do {
        dv = std::div(dv.quot, base);
        buf += "0123456789abcdef"[std::abs(dv.rem)];  // string literals are arrays
    } while(dv.quot);
    if(n<0) buf += '-';
    return {buf.rbegin(), buf.rend()};
}

int main()
{
    std::cout << itoa(12345, 10) << '\n'
              << itoa(-12345, 10) << '\n'
              << itoa(42, 2) << '\n'
              << itoa(65535, 16) << '\n';
}
```

Output:

```
12345
-12345
101010
ffff
```

See also

fmod **fmodf** (C++11) **fmodl** (C++11)	remainder of the floating point division operation (function)
remainder (C++11) **remainderf** (C++11) **remainderl** (C++11)	signed remainder of the division operation (function)
remquo (C++11) **remquof** (C++11) **remquol** (C++11)	signed remainder as well as the three last bits of the division operation (function)

C documentation for **div**