to_chars

In this part we learn about the to_char function and it's two types of declarations; the integral and floating point.

WE'LL COVER THE FOLLOWING

`to_chars` is a set of overloaded functions for integral and floating point types.

Integral types declaration

```
std::to_chars_result to_chars(char* first, char* last, TYPE value, int base = 10);
```

Where `TYPE` expands to all available signed and unsigned integer types and `char`.

Since `base` might range from 2 to 36, the output digits that are greater than 9 are represented as lowercase letters: `a...z`.

Floating point types declaration

Basic function

```
std::to_chars_result to_chars(char* first, char* last, FLOAT_TYPE value);
```

`FLOAT_TYPE` expands to `float`, `double` or `long double`.

The conversion works the same as with `printf` and in default (“C”) locale. It uses **%f** or **%e** format specifier favoring the representation that is the shortest.

Function with std::chars_format

The next function adds `std::chars_format` `fmt` that let’s you specify the output format:

```
std::to_chars_result to_chars(char* first, char* last, FLOAT_TYPE value, std::chars_format fm
```

Full version function

There’s a “full” version that also allows to specify precision:

```
std::to_chars_result to_chars(char* first,
                              char* last,
                              FLOAT_TYPE value,
                              std::chars_format fmt,
                              int precision);
```

When the conversion is successful, the range [first, last) is filled with the converted string.

Return Value of all functions

The returned value for all functions (for integer and floating point support) is `to_chars_result`, it’s defined as follows:

```
struct to_chars_result
{
  char* ptr;
  std::errc ec;
};
```

The type holds information about the conversion process:

- On **Success** - `ec` equals value-initialized `std::errc` and `ptr` is the one-past-the-end pointer of the characters written. Note that the string is not NULL-terminated.
- On **Error** - `ptr` equals `first` and `ec` equals `std::errc::invalid_argument`. `value` is unmodified.
- On **Out of range** - `ec` equals `std::errc::value_too_large` the range `[first, last)` in unspecified state.

to_char: An Example

> At the time of writing there was no support for floating-point overloads, so the example uses only integers.

```
#include <iostream>
#include <charconv> // from_chars, to_chars
#include <string>
int main()
{
    std::string str { "xxxxxxxx" };
    const int value = 1986;
    const auto res = std::to_chars(str.data(),str.data() + str.size(),value);
    if (res.ec == std::errc())
    {
      std::cout << str << ", filled: "<< res.ptr - str.data() << " characters\n";
    }
    else
    {
        std::cout << "value too large!\n";
    }
}
```

Below you can find a sample output for a set of numbers:

`value`	output
`1986`	`1986xxxx, filled: 4 characters`
`-1986`	`-1986xxx, filled: 5 characters`
`19861986`	`19861986, filled: 8 characters`
`-19861986`	`value too large!` (the buffer is only 8 characters)

The next lesson will introduce you to the concept of benchmark which helps measure the performances of the studied conversion methods.