5.3 — Numeral systems (decimal, binary, hexadecimal, and octal)

👤 **ALEX**   🕐 **JANUARY 20, 2024**

> ### Author's note
>
> This lesson is optional.
>
> Future lessons reference hexadecimal numbers, so you should at least have a passing familiarity with the concept before proceeding.

In everyday life, we count using **decimal** numbers, where each numerical digit can be 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Decimal is also called "base 10", because there are 10 possible digits (0 through 9). In this system, we count like this: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, … By default, numbers in C++ programs are assumed to be decimal.

```
int x { 12 }; // 12 is assumed to be a decimal number
```

In **binary**, there are only 2 digits: 0 and 1, so it is called "base 2". In binary, we count like this: 0, 1, 10, 11, 100, 101, 110, 111, …

Decimal and binary are two examples of **numeral systems**, which is a fancy name for a collection of symbols (e.g. digits) used to represent numbers. There are 4 main numeral systems available in C++. In order of popularity, these are: decimal (base 10), binary (base 2), hexadecimal (base 16), and octal (base 8).

Octal and hexadecimal literals

Octal is base 8 -- that is, the only digits available are: 0, 1, 2, 3, 4, 5, 6, and 7. In Octal, we count like this: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, … (note: no 8 and 9, so we skip from 7 to 10).

Decimal	0	1	2	3	4	5	6	7	8	9	10	11
Octal	0	1	2	3	4	5	6	7	10	11	12	13

To use an octal literal, prefix your literal with a 0 (zero):

```
#include <iostream>

int main()
{
    int x{ 012 }; // 0 before the number means this is octal
    std::cout << x << '\n';
    return 0;
}
```

This program prints:

```
10
```

Why 10 instead of 12? Because numbers are output in decimal by default, and 12 octal = 10 decimal.

Octal is hardly ever used, and we recommend you avoid it.

Hexadecimal is base 16. In hexadecimal, we count like this: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, ...

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11

You can also use lower case letters (though upper case is more common).

To use a hexadecimal literal, prefix your literal with `0x` :

· · ·

```
#include <iostream>

int main()
{
    int x{ 0xF }; // 0x before the number means this is hexadecimal
    std::cout << x << '\n';
    return 0;
}
```

This program prints:

```
15
```

You can also use a `0X` prefix, but `0x` is conventional because its easier to read.

Using hexadecimal to represent binary

Because there are 16 different values for a hexadecimal digit, we can say that a single hexadecimal digit encompasses 4 bits. Consequently, a pair of hexadecimal digits can be used to exactly represent a full byte.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110

Consider a 32-bit integer with binary value 0011 1010 0111 1111 1001 1000 0010 0110. Because of the length and repetition of digits, that's not easy to read. In hexadecimal, this same value would be: 3A7F 9826, which is much more concise. For this reason, hexadecimal values are often used to represent memory addresses or raw data in memory (whose type isn't known).

Binary literals

Prior to C++14, there is no support for binary literals. However, hexadecimal literals provide us with a useful workaround (that you may still see in existing code bases):
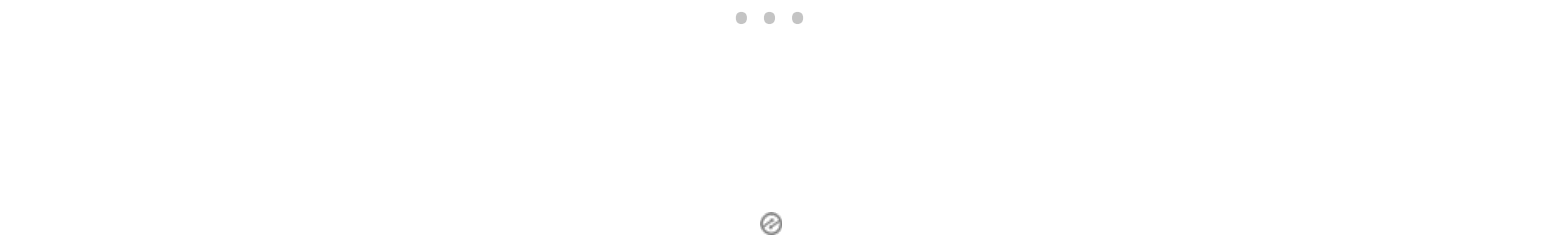
```cpp
#include <iostream>

int main()
{
    int bin{};    // assume 16-bit ints
    bin = 0x0001; // assign binary 0000 0000 0000 0001 to the variable
    bin = 0x0002; // assign binary 0000 0000 0000 0010 to the variable
    bin = 0x0004; // assign binary 0000 0000 0000 0100 to the variable
    bin = 0x0008; // assign binary 0000 0000 0000 1000 to the variable
    bin = 0x0010; // assign binary 0000 0000 0001 0000 to the variable
    bin = 0x0020; // assign binary 0000 0000 0010 0000 to the variable
    bin = 0x0040; // assign binary 0000 0000 0100 0000 to the variable
    bin = 0x0080; // assign binary 0000 0000 1000 0000 to the variable
    bin = 0x00FF; // assign binary 0000 0000 1111 1111 to the variable
    bin = 0x00B3; // assign binary 0000 0000 1011 0011 to the variable
    bin = 0xF770; // assign binary 1111 0111 0111 0000 to the variable

    return 0;
}
```

In C++14 onward, we can use binary literals by using the 0b prefix:

```cpp
#include <iostream>

int main()
{
    int bin{};          // assume 16-bit ints
    bin = 0b1;          // assign binary 0000 0000 0000 0001 to the variable
    bin = 0b11;         // assign binary 0000 0000 0000 0011 to the variable
    bin = 0b1010;       // assign binary 0000 0000 0000 1010 to the variable
    bin = 0b11110000;   // assign binary 0000 0000 1111 0000 to the variable

    return 0;
}
```

Digit separators

Because long literals can be hard to read, C++14 also adds the ability to use a quotation mark (') as a digit separator.

```cpp
#include <iostream>

int main()
{
    int bin { 0b1011'0010 };   // assign binary 1011 0010 to the variable
    long value { 2'132'673'462 }; // much easier to read than 2132673462

    return 0;
}
```

Also note that the separator can not occur before the first digit of the value:

```cpp
int bin { 0b'1011'0010 };  // error: ' used before first digit of value
```

Digit separators are purely visual and do not impact the literal value in any way.

Outputting values in decimal, octal, or hexadecimal

• • •

By default, C++ outputs values in decimal. However, you can change the output format via use of the `std::dec`, `std::oct`, and `std::hex` I/O manipulators:

```cpp
#include <iostream>

int main()
{
    int x { 12 };
    std::cout << x << '\n'; // decimal (by default)
    std::cout << std::hex << x << '\n'; // hexadecimal
    std::cout << x << '\n'; // now hexadecimal
    std::cout << std::oct << x << '\n'; // octal
    std::cout << std::dec << x << '\n'; // return to decimal
    std::cout << x << '\n'; // decimal

    return 0;
}
```

This prints:

```
12
c
c
14
12
12
```

Note that once applied, the I/O manipulator remains set for future output until it is changed again.

Outputting values in binary

Outputting values in binary is a little harder, as `std::cout` doesn't come with this capability built-in. Fortunately, the C++ standard library includes a type called `std::bitset` that will do this for us (in the <bitset> header).

To use `std::bitset`, we can define a `std::bitset` variable and tell `std::bitset` how many bits we want to store. The number of bits must be a compile-time constant. `std::bitset` can be initialized with an integral value (in any format, including decimal, octal, hex, or binary).

```cpp
#include <bitset> // for std::bitset
#include <iostream>

int main()
{
    // std::bitset<8> means we want to store 8 bits
    std::bitset<8> bin1{ 0b1100'0101 }; // binary literal for binary 1100 0101
    std::bitset<8> bin2{ 0xC5 }; // hexadecimal literal for binary 1100 0101

    std::cout << bin1 << '\n' << bin2 << '\n';
    std::cout << std::bitset<4>{ 0b1010 } << '\n'; // create a temporary std::bitset and print it

    return 0;
}
```

This prints:

```
11000101
11000101
1010
```

In the above code, this line:

```cpp
std::cout << std::bitset<4>{ 0b1010 } << '\n'; // create a temporary std::bitset and print it
```

creates a temporary (unnamed) `std::bitset` object with 4 bits, initializes it with binary literal `0b1010`, prints the value in binary, and then discards the temporary object.

Related content

We cover std::bitset in more detail in lesson O.1 -- Bit flags and bit manipulation via std::bitset (https://www.learncpp.com/cpp-tutorial/bit-flags-and-bit-manipulation-via-stdbitset/).

In C++20 and C++23, we have better options available via the new Format Library (C++20) and Print Library (C++23):

```cpp
#include <format> // C++20
#include <iostream>
#include <print> // C++23

int main()
{
    std::cout << std::format("{:b}\n", 0b1010);  // C++20
    std::cout << std::format("{:#b}\n", 0b1010); // C++20

    std::println("{:b} {:#b}", 0b1010, 0b1010);  // C++23

    return 0;
}
```

This prints:

```
1010
0b1010
1010 0b1010
```

Leave a comment...

Name*

Email*

Notify me about replies: 🔔 **POST COMMENT**

🐞 Find a mistake? Leave a comment above!❓

👤 Avatars from https://gravatar.com/ are connected to your provided email address.

120 COMMENTS   Newest ▾

We and our partners share information on your use of this website to help improve your experience.

Do not sell my info: ⬤

OKAY

✕