Format Specifiers: Format_Character

This lesson explains the format_character, which is a part of formatted output.

WE'LL COVER THE FOLLOWING

- format_character

format_character

The following format characters are used in the `wrtiefln()` function:

- `b`: an integer argument is displayed in the binary system.
- `o`: an integer argument is displayed in the octal system.
- `x` and `X`: an integer argument is displayed in the hexadecimal system; with lowercase letters when using `x` and with uppercase letters when using `X`.
- `d`: an integer argument is displayed in the decimal system; a negative sign is also displayed if it is a signed type and the value is less than zero.

```
import std.stdio;

void main()
{
    int value = 12;

    writefln("Binary : %b", value);
    writefln("Octal : %o", value);
    writefln("Hexadecimal: %x", value);
    writefln("Decimal : %d", value);
}
```

format_character

- `e`: a floating point argument is displayed according to the following rules. A floating point argument must have:
  - a single digit before the decimal mark
  - a decimal mark if precision is nonzero
  - the required digits after the decimal mark, the number of which is determined by precision (default precision is 6)
  - the *e* character (meaning “10 to the power of”)
  - the `-` or `+` character, depending on whether the exponent is less than or greater than zero
  - the exponent, consisting of at least two digits
- `E`: same as `e`, with the exception of displaying character `E` instead of `e`.
- `f` and `F`: a floating point argument is displayed in the decimal system; there is at least one digit before the decimal mark and the default precision is 6 digits after the decimal mark.
- `g`: same as `f` if the exponent is between -5 and precision; otherwise same as `e`.
  Here, *precision* does not specify the number of digits after the decimal mark, but the significant digits of the entire value. If there are no significant digits after the decimal mark, then the decimal mark is not displayed. The rightmost zeros after the decimal mark are not displayed.
- `G`: Same as `g`, with the exception of displaying the character `E`.
- `a`: a floating point argument is displayed in the hexadecimal floating point notation:
  - the characters 0x
  - a single hexadecimal digit
  - a decimal mark if precision is nonzero

    - the required digits after the decimal mark, the number of which is determined by precision; if no precision is specified, then as many digits as necessary
    - the *p* character (meaning “2 to the power of”)
    - the `-` or `+` character, depending on whether the exponent is less than or greater than zero
    - the exponent, consisting of at least one digit (the exponent of the value 0 is 0) .
- `A`: same as `a`, with the exception of outputting the characters 0X and P.

```
import std.stdio;

void main(){
    double value = 123.456789;

    writefln("with e: %e", value);
    writefln("with f: %f", value);
    writefln("with g: %g", value);
    writefln("with a: %a", value);
}
```

Format characters

- `s`: the value is displayed in the same way as in regular output, according to the type of the argument. It consists of:
    - `bool` values as `true` or `false`
    - integer values same as `%d`
    - floating point values same as `%g`
    - `string` in UTF-8 encoding; precision determines the maximum number of bytes to use (remember that in UTF-8 encoding, the number of bytes is not the same as the number of characters; for example, the `string` “ağ” has 2 characters, consisting a total of 3 bytes)
    - `struct` and `class` objects as the return value of the `toString()`

member functions of their types; precision determines the maximum

number of bytes to use

- arrays as their element values, side by side

```
import std.stdio;

void main(){
    bool b = true;
    int i = 365;
    double d = 9.87;
    string s = "formatted";

    int[] a = [ 2, 4, 6, 8 ];

    writefln("bool : %s", b);
    writefln("int : %s", i);
    writefln("double: %s", d);
    writefln("string: %s", s);
    writefln("array : %s", a);
}
```

format_character

In the next lesson, we will explore four more parts of the format specifiers, including width, separator, precision and flags.