Formatted Input

This lesson explains how the format of data can be specified at the time of input.

WE'LL COVER THE FOLLOWING

What is formatted input?

It is possible to specify the format of the data expected to be read as input using `readf`. It specifies both the data that should be read and the characters that should be ignored.

D's input format specifiers are similar to the ones present in the C language. As previously used, the format specifier `" %s"` reads the data according to the type of the variable. For example, since the type of the variable used in `readf` is `double`, the characters read at the input would be treated as a floating point number:

```
double number;
readf(" %s", &number);
```

The format string can contain three types of information:

- **The space character:** indicates zero or more whitespace characters at the input and specifies that all of those characters should be read and ignored.
- **Format specifier:** similar to the output format specifiers, input format specifiers start with the % character and determine the format of the data that is to be read.
- **Any other character:** the characters that are expected at the input as is;

they should be read and ignored.

The format string makes it possible to select specific information from the input and ignore the rest.

Let's look at an example that uses all of the three types of information in the format string. Let's assume that the student number and the grade are expected to appear at the input in the following format:

```
number:123 grade:90
```

Let's further assume that the tags `number:` and `grade:` must be ignored. The following format `string` would select the values of `number` and `grade` and would ignore the other characters:

```
int number;
int grade;
readf("number:%s grade:%s", &number, &grade);
```

The format characters `"number:%s grade:%s"` must appear at the input exactly as specified; `readf()` reads and ignores them.
The single space character that appears in the format `string` above would cause all of the whitespace characters that appear exactly at that position to be read and ignored.
As the `%` character has a special meaning in format strings, when that character itself needs to be read and ignored, it must be written twice in the format `string` as `%%`.

Reading a single line of data from the input has been recommended as `strip(readln())` in the strings lesson. Instead of that method, a `\n` character at the end of the format string can achieve a similar goal:

> **Note:** Provide all the inputs before running the code.

```
import std.stdio;

void main() {
    write("First name: ");
    string firstName;
    readf(" %s\n", &firstName);    // ← \n at the end
    write("Last name : ");
    string lastName;

    readf(" %s\n", &lastName);     // ← \n at the end

    write("Age : ");
    int age;
    readf(" %s", &age);

    writefln("%s %s (%s)", firstName, lastName, age);
}
```

Reading formatted data using \n character

The `\n` characters at the end of the format strings when reading `firstName` and `lastName` would cause the new-line characters to be read from the input and to be ignored. However, potential whitespace characters at the ends of the strings may still need to be removed by `strip()`.

Format specifier characters

The way the data should be read is specified with the following format specifier characters:

- `d:` read an integer in the decimal system.
- `o:` read an integer in the octal system.
- `x:` read an integer in the hexadecimal system.
- `f:` read a floating point number.
- `s:` read according to the type of the variable. This is the most commonly used specifier.
- `c:` read a single character. This specifier allows reading whitespace characters as well (they are not ignored anymore).
  As an example, if the input contains "23 23 23," the values would be read differently according to different format specifiers:

```
int number_d;
int number_o;
int number_x;
```

```
readf(" %d %o %x", &number_d, &number_o, &number_x);

writeln("Read with %d: ", number_d);
writeln("Read with %o: ", number_o);
writeln("Read with %x: ", number_x);
```

```
import std.stdio;

void main() {
    int number_d;
    int number_o;
    int number_x;

    readf(" %d %o %x", &number_d, &number_o, &number_x);

    writeln("Read with %d: ", number_d);
    writeln("Read with %o: ", number_o);
    writeln("Read with %x: ", number_x);
}
```

However, when the input contains three sets of “23” characters, the values of the variables are different:

```
Read with %d: 23
Read with %o: 19
Read with %x: 35
```

> **Note:** Very briefly, “23” is equal to 2x8+3=19 in the octal system and to 2x16+3=35 in the hexadecimal system.

In the next lesson, you will find a coding challenge based on the concepts of formatted input.