Characters: Types and Literals

In this lesson, we see character types and literals. Furthermore, we will explore control characters and the use of single quotes and backslash in D.

WE'LL COVER THE FOLLOWING

The character types of D

There are three D types to represent characters. These characters correspond to the three Unicode encodings mentioned previously: UTF-32, UTF-16, and UTF-8. Copying from the fundamental types lesson:

Type	Definition	Initial Value
char	UTF-8 code unit	0xFF
wchar	UTF-16 code unit	0xFFFF
dchar	UTF-32 code unit and Unicode code point	0x0000FFFF

Compared to some other programming languages, characters in D may consist of a different number of bytes. For example, because 'Ğ' must be represented by at least 2 bytes in Unicode, it doesn't fit in a variable of type `char`. On the other hand, because `dchar` consists of 4 bytes, it can hold any Unicode character.

Character literals

Literals are constant character values that are written in the program as a part of the source code. In D, character literals are specified within single quotes:

```
char letter_a = 'a';
wchar letter_e_acute = 'é';
```

Double quotes are not valid for characters because double quotes are used when specifying strings. 'a' is a character literal and "a" is a string literal that consists of a single character.
Variables of type `char` can only hold letters that are in the ASCII table. There are many ways of inserting characters in code:

- Most naturally, typing them on the keyboard.
- Copying from another program or another text. For example, you can copy and paste from a website or from a program that is specifically for displaying Unicode characters (e.g., character map on Linux).
- Using short names of the characters. The syntax for this feature is '\&character_name;'. For example, the name of the Euro character is *euro* and it can be specified in the program like the following:

  ```
  wchar currencySymbol = '\&euro;';
  ```

  See the list of named characters for all characters that can be specified this way.
- Specifying characters by their integer Unicode values:

  ```
  char a = 97;
  wchar Ğ = 286;
  ```
- Specifying the codes of the characters of the ASCII table either by *\value_in_octal* or *\xvalue_in_hexadecimal* syntax:

  ```
  char questionMarkOctal = '\77';
  char questionMarkHexadecimal = '\x3f';
  ```
- Specifying the Unicode values of the characters by using the *\ufour_digit_value* syntax for `wchar`, and the *\Ueight_digit_value* syntax for `dchar` (note u versus U). The Unicode values must be specified in

hexadecimal:

```
wchar Ğ_w = '\u011e';
dchar Ğ_d = '\U0000011e';
```

These methods can be used to specify the characters within strings as well. For example, the following two lines have the same string literals:

```
writeln("Résumé preparation: 10.25€"); writeln("\x52\&eacute;sum\u00e9 pre
paration: 10.25\&euro;");
```

Control characters

Some characters only affect the formatting of the text. They don't have a visual representation themselves. For example, the new-line character, which specifies that the output should continue on a new line, does not have a visual representation. Such characters are called **control characters**. Some common control characters can be specified with the *control_character* syntax.

Syntax	Name	Definition
\n	new line	Moves the printing to a new line
\r	carriage return	Moves the printing to the beginning of the current line
\t	tab	Moves the printing to the next tab stop

Control characters

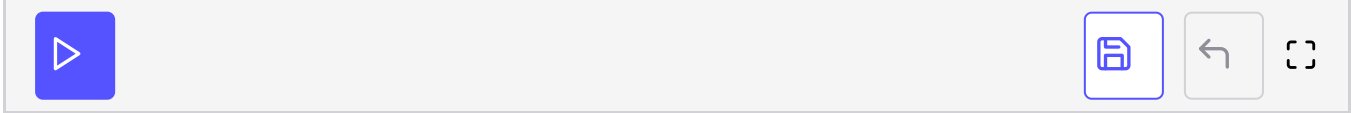
For example, the `write()` function, which does not start a new line automatically, would do so for every `\n` character. Every occurrence of the `\n` control character in the following code represents the start of a new line:

```
import std.stdio;

void main() {

    write("first line\nsecond line\nthird line\n");

}
```



Single quote and backslash

Single quote and backslash

The single quote character itself cannot be written within single quotes because the compiler would take the second one as the closing character of the first one:

```
// this will cause an error
'''
```

The first two would be the opening and closing quotes, and the third one would be left alone, causing a compilation error.
Similarly, since the backslash character has a special meaning in the control character and literal syntaxes, the compiler would take it as the start of such a syntax: '\ '. The compiler then would be looking for a closing single quote character, not finding one, and emitting a compilation error. To display these characters properly, the single quote and the backslash characters are escaped by a preceding backslash character:

Syntax	Name	Definition
\'	single quote	Allows specifying the single quote character:'\''
\\	backslash	Allows specifying the backslash character: '\\' or "\\"

In the next lesson, we will see the `std.uni` module.