Characters: Introduction

In this lesson, we see a brief history of characters and different unicode encodings of characters.

WE'LL COVER THE FOLLOWING

Characters

Characters are building blocks of *strings*. Any symbol of a writing system is called a **character**: letters of alphabets, numerals, punctuation marks, the space character, etc. Confusingly, building blocks of characters themselves are called characters as well.

Arrays of characters make up strings. We have seen arrays earlier in this chapter; strings will be covered later in this chapter.

Like any other data, characters are also represented as integer values that are made up of bits. For example, the integer value of the lowercase ‘a’ is 97 and the integer value of the numeral ‘1’ is 49. These values are merely conventions assigned when the ASCII standard was designed.

Character representation

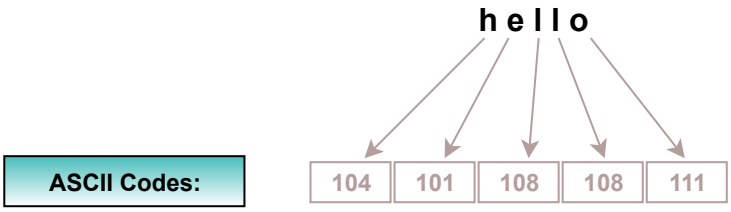
In many programming languages, characters are represented by the **char type**, which can hold only 256 distinct values. If you are familiar with the `char` type from other languages, you may already know that it is not large enough to support the symbols of many writing systems. Before getting into the three distinct character types of D, let's first take a look at the history of characters in computer systems.

History

ASCII table

The ASCII table was designed at a time when computer hardware was very limited compared to modern systems. Having been based on 7 bits, the ASCII table can have 128 distinct code values. That many distinct values are sufficient to represent the lowercase and uppercase versions of the 26 letters of the basic Latin alphabet, numerals, commonly used punctuation marks and some terminal control characters.

As an example, the ASCII codes of the characters of the string "hello" are the following:



ASCII codes for "hello" word

Every code above represents a single letter of "hello." For example, there are two 108 values corresponding to the two 'l' letters. The codes of the ASCII table were later increased to 8 bits to become the Extended ASCII table. The Extended ASCII table has 256 distinct codes.

IBM code pages

IBM Corporation has defined a set of tables, each of which assigns the codes of the Extended ASCII table from 128 to 255 to one or more writing systems. These code tables allow supporting the letters of many more alphabets. For example, the special letters of the Turkish alphabet are a part of IBM's code

page 857.

Despite being much more useful than ASCII, code pages have some problems and limitations: In order to display text correctly, it must be known what code page a given text was originally written in. This is because the same code corresponds to a different character in most other tables. For example, the code that represents 'Ğ' in table 857 corresponds to 'a' in table 437. In addition to the difficulty of supporting multiple alphabets in a single document, alphabets that have more than 128 non-ASCII characters cannot be supported by an IBM table at all.

ISO/IEC 8859 code pages

The ISO/IEC 8859 code pages are a result of international standardization efforts. They are similar to IBM's code pages in how they assign codes to characters. As an example, the special letters of the Turkish alphabet appear in code page 8859-9. These tables have the same problems and limitations as IBM's tables. For example, the Dutch digraph 'ij' does not appear in any of these tables.

Unicode

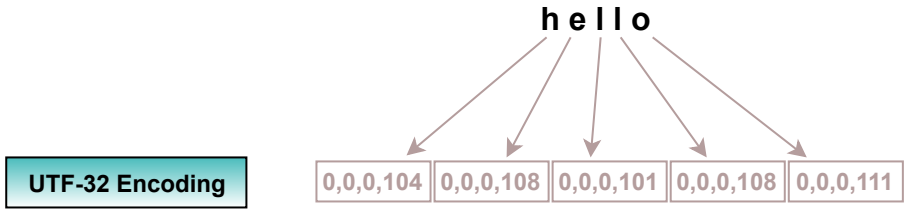
Unicode solves all the problems and limitations of previous solutions. Unicode includes more than a hundred thousand characters and symbols of the writing systems of many human languages, current and old. New characters are constantly under review for addition to the table. Each of these characters has a unique code. Documents that are encoded in Unicode can include all characters of separate writing systems without any confusion or limitations.

Unicode encodings

Unicode assigns a unique code for each character. Since there are more Unicode characters than an 8-bit value can hold, some characters must be represented by at least two 8-bit values. For example, the Unicode character code of 'Ğ' (286) is greater than the maximum value of a `ubyte`.
The way characters are represented in electronic mediums is called their **encoding**. We have seen above how the string "hello" is encoded in ASCII. We will now see three Unicode encodings that correspond to D's character types.

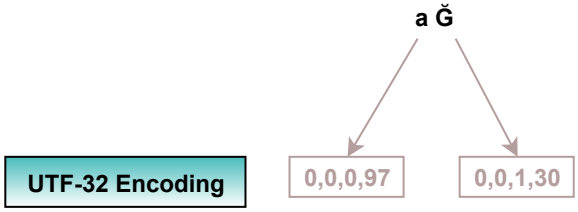
UTF-32

This encoding uses 32 bits (4 bytes) for every Unicode character. The UTF-32 encoding of “hello” is similar to its ASCII encoding, but every character is represented with 4 bytes:



UTF-32 Encoding for "hello" Word

As another example, the UTF-32 encoding of “aĞ” is the following:



UTF-32 encoding for "hello" Word

> **Note:** The order of the bytes of UTF-32 may be different on different computer systems.

‘a’ and ‘Ğ’ are represented by 1 and 2 significant bytes, respectively, and the values of the other 5 bytes are all zeros. These zeros can be thought of as filler bytes to make every Unicode character occupy 4 bytes each.

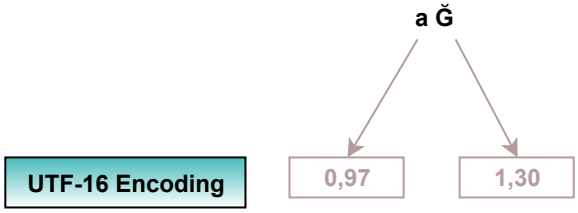
For documents based on the basic Latin alphabet, this encoding always uses 4 times as many bytes as the ASCII encoding. When most of the characters of a given document have ASCII equivalents, the 3 filler bytes for each of those characters make this encoding more wasteful compared to other encodings. On the other hand, there are benefits of representing every character by an equal number of bytes. For example, the next Unicode character is always exactly four bytes away.

UTF-16

This encoding uses 16 bits (2 bytes) to represent most of the Unicode

characters. Since 16 bits can have about 65 thousand unique values, the other

(less commonly used) 35 thousand Unicode characters must be represented using additional bytes.
As an example, “aĞ” is encoded by 4 bytes in UTF-16:



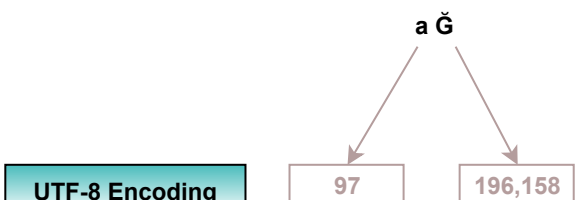
UTF-32 encoding for "hello" word

> **Note:** The order of the bytes of UTF-16 may be different on different computer systems.

Compared to UTF-32, this encoding takes less space for most documents, but because some characters must be represented by more than 2 bytes, UTF-16 is more complicated to process.

UTF-8

This encoding uses 1 to 4 bytes for every character. If a character has an equivalent in the ASCII table, it is represented by 1 byte with the same numeric code as in the ASCII table. The rest of the Unicode characters are represented by 2, 3 or 4 bytes. Most of the special characters of the European writing systems are among the group of characters that are represented by 2 bytes.
For most documents in Western countries, UTF-8 is the encoding that takes the least amount of space. Another benefit of UTF-8 is that the documents that were produced using ASCII can be opened directly (without conversion) as UTF-8 documents. UTF-8 also does not waste any space with filler bytes, as every character is represented by significant bytes. As an example, the UTF-8 encoding of “aĞ” is:



UTF-32 encoding for "hello" word

In the next lesson, we will see character types and literals in D.