

Huge number of character representations (encodings) exist
you need know only two:

- ASCII (ISO 646)
 - single byte values, only low 7-bit used, top bit always 0
 - can encode roman alphabet a-zA-Z, digits 0-9 , punctuation, control chars
 - complete alphabet for English, Bahasa
 - no diacritics, e.g: ç , so missing a little of alphabet for other latin languages, e.g.: German, French, Spanish, Italian, Swedish, Tagalog, Swahili
 - characters for most of world's languages completely missing
- UTF-8 (Unicode)
 - contains all ASCII (single-byte) values
 - also has 2-4 byte values, top bit always 1 for bytes of multi-byte values
 - contains symbols for essentially all human languages plus other symbols, e.g.:

✓ ∑ ∇ ∃



- Uses values in the range 0x00 to 0x7F (0..127)
- Characters partitioned into sequential groups
 - control characters (0..31) ... e.g. '\n'
 - punctuation chars (32..47,91..96,123..126)
 - digits (48..57) ... '0'..'9'
 - upper case alphabetic (65..90) ... 'A'..'Z'
 - lower case alphabetic (97..122) ... 'a'..'z'
- Sequential nature of groups allow ordination e.g.
'3' - '0' == 3 'J' - 'A' == 10
- See `man 7 ascii`

1

2

Unicode

UTF-8 Encoding

- Widely-used standard for expressing “writing systems”
 - not all writing systems use a small set of discrete symbols
- Basically, a 32-bit representation of a wide range of symbols
 - around 140K symbols, covering 140 different languages
- Using 32-bits for every symbol would be too expensive
 - e.g. standard roman alphabet + punctuation needs only 7-bits
 - Several Unicode encodings have been developed
 - UTF-8 most widely used encoding, dominates web-use
 - designed by Ken Thompson on napkin in New Jersey diner

#bytes	#bits	Byte 1	Byte 2	Byte 3	Byte 4
1	7	0xxxxxxx	-	-	-
2	11	110xxxxx	10xxxxxx	-	-
3	16	1110xxxx	10xxxxxx	10xxxxxx	-
4	21	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

- The 127 1-byte codes are compatible with ASCII
- The 2048 2-byte codes include most Latin-script alphabets
- The 65536 3-byte codes include most Asian languages
- The 2097152 4-byte codes include symbols and emojis and ...

ch	code-point	unicode binary	UTF-8 encoding
\$	U+0024	0100100	00100100
¢	U+00A2	00010100010	11000010 10100010
€	U+20AC	0010000010101100	11100010 10000010 10101100

3

4

```
printf("The unicode code point U+1F600 encodes in UTF-8\n");
printf("as 4 bytes: 0xF0 0x9F 0x98 0x80\n");
printf("We can output the 4 bytes like this: \xF0\x9F\x98\x80\n");
printf("Or like this: ");
putchar(0xF0);
putchar(0x9F);
putchar(0x98);
putchar(0x80);
putchar('\n');
```

source code for hello.c

5

```
uint8_t encoding[5] = {0};
if (code_point < 0x80) {
    encoding[0] = code_point;
} else if (code_point < 0x800) {
    encoding[0] = 0xC0 | (code_point >> 6);
    encoding[1] = 0x80 | (code_point & 0x3f);
} else if (code_point < 0x10000) {
    encoding[0] = 0xE0 | (code_point >> 12);
    encoding[1] = 0x80 | ((code_point >> 6) & 0x3f);
    encoding[2] = 0x80 | (code_point & 0x3f);
} else if (code_point < 0x200000) {
    encoding[0] = 0xF0 | (code_point >> 18);
    encoding[1] = 0x80 | ((code_point >> 12) & 0x3f);
    encoding[2] = 0x80 | ((code_point >> 6) & 0x3f);
    encoding[3] = 0x80 | (code_point & 0x3f);
}
```

source code for utf8_encode.c

6

```
printf("U+%x UTF-8: ", code_point);
for (uint8_t *s = encoding; *s != 0; s++) {
    printf("0x%02x ", *s);
}
printf(" %s\n", encoding);
}
int main(void) {
    print_utf8_encoding(0x42);
    print_utf8_encoding(0x00A2);
    print_utf8_encoding(0x10be);
    print_utf8_encoding(0x1F600);
}
```

source code for utf8_encode.c

7

- Compact, but not minimal encoding; encoding allows you to resync immediately if bytes lost from a stream.
- ASCII is a subset of UTF-8 - complete backwards compatibility!
- All other UTF-8 bytes > 127 (0x7f)
 - no byte of multi-byte UTF-8 encoding is valid ASCII.
- No byte of multi-byte UTF-8 encoding is 0
 - can still use store UTF-8 in null-terminated strings.
- 0x2F (ASCII /) and 0x00 can not appear in multi-byte characters
 - hence can use UTF-8 for Linux/Unix filenames
- C programs can treat UTF-8 similarly to ASCII.
- Beware: number of bytes in UTF-8 string != number of characters.

8