DPST1092 24T3 — Characters and Unicode

https://www.cse.unsw.edu.au/~dp1092/24T3/

Character Data

Character data has several possible representations (encodings)

The two most common:

- ASCII (ISO 646)
  - 7-bit values, using lower 7-bits of a byte (top bit always zero)
  - can encode roman alphabet, digits, punctuation, control chars
- UTF-8 (Unicode)
  - 8-bit values, with ability to extend to multi-byte values
  - can encode all human languages plus other symbols

  e.g.:

√ ∑ ∀ ∃ 

ASCII Character Encoding

Uses values in the range `0x00` to `0x7F` (0..127)

Characters partitioned into sequential groups

- control characters (0..31) ... e.g. `'\0'`, `'\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 allows for things like `(ch - '0')` Eg.

See `man 7 ascii`

Unicode

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

More compact character encodings have been developed (e.g. UTF-8)

UTF-8 Character Encoding

UTF-8 uses a variable-length encoding as follows

#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 ...

ASCII Character Encoding

UTF-8 examples

ch	unicode	bits	simple binary	UTF-8 binary
$	U+0024	7	`010 0100`	`00100100`
¢	U+00A2	11	`000 1010 0010`	`11000010 10100010`
€	U+20AC	16	`0010 0000 1010 1100`	`11100010 10000010 10101100`

Unicode strings can be manipulated in C (e.g."안녕하세요")

Like other C strings, they are terminated by a 0 byte (i.e. `'\0'`)

Warning: Functions like strlen may not work as expected.

Printing UTF-8 in a C program

```
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_unicode.c

Converting Unicode Codepoints to UTF-8

```
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

Converting Unicode Codepoints to UTF-8

```
    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

Exercise 1: UTF-8 Unicode Encoding

For each of the following symbols, with their Unicode value

- show the bit-string that would be used to represent them

Symbols:

- & U+00026
- µ U+000B5

Given that ♥ has the code U+02665

- Convert it into the bitstring that would represent it
- Write a C program to print
  ♥ beats

Summary of UTF-8 Properties

- 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.