System and Device Programming

Encodings

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Files

- Files store information is store for long period of times and independently from the power supply
- From the logical point of view a file is
  - A set of correlated information
    - All information (i.e., numbers, characters, images, etc.) are stored in a (electronic) device using a **coding system**
  - A contiguous address space on a long-term memory

How is this information encoded?

What is the actual organization of this space?

Encoding

- A **character set** comprises the set of characters one might use for a particular purpose
  - For example, the Western European languages
- A **coded character set** is a set of characters for which a unique number has been assigned to each character
  - A **code point** represents the position of a character in the coded character set
- The **character encoding** is the way the coded character set is mapped to bytes for manipulation in a computer

Encoding

- The most popular character encoding standards are currently being used all over the world are
  - ASCII
    - To represent text symbols, such as letters, digits, etc.
  - Unicode
    - The universal character encoding used to process, store and facilitate the interchange of text data in any language

ASCII

128 total characters
32 not printable
96 printable

- ASCII is a de-facto standard
  - ASCII = American Standard Code for Information Interchange
    - Originally based on the English alphabet
    - 128 characters are coded in 7-bit (binary numbers)
  - Extended ASCII (or high ASCII)
    - Extension of ASCII to 8-bit and 255 characters
    - Several versions exist
      - ISO 8859-1 (ISO Latin-1), ISO 8859-2 (Eastern European languages), ISO 8859-5 for Cyrillic languages, etc.

The alphabet of the Klingom language is not supported by Extended ASCII

ASCII

ASCII Table

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

(from commons.wikimedia.org)

128 total
32 non printable
96 printable chars

ASCII

256 total

Extended ASCII Table

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

From ASCII to Unicode

- Unfortunately, software for international use must be able to represent more characters
  - A variety of multi-byte encoding schemes are internationally used to represent
    - Non-Latin alphabets
    - Non-alphabetic languages such as Chinese, Japanese, Korean, etc.
- Many applications cannot support more than one encoding
  - It is usually impossible to combine different encoding in the same application
  - It is difficult to support multilingual documents

Unicode

- An industry standard
  - Consistent encoding, representation, and handling of text expressed in most of the world's writing systems
    - First version [1991]
      - Started out with 65,536 codes, encoded on 16 bits
    - Most recent version [June 2016]
      - Unicode 9.0, ISO/IEC 10646:2014 plus Amendments 1 & 2
      - Encodes 110,187 symbols out of the available 1.1 million code points
      - Covers 100 scripts and multiple symbol sets including emoji

1,114,112 characters from 0x000000 to 0x10FFFF

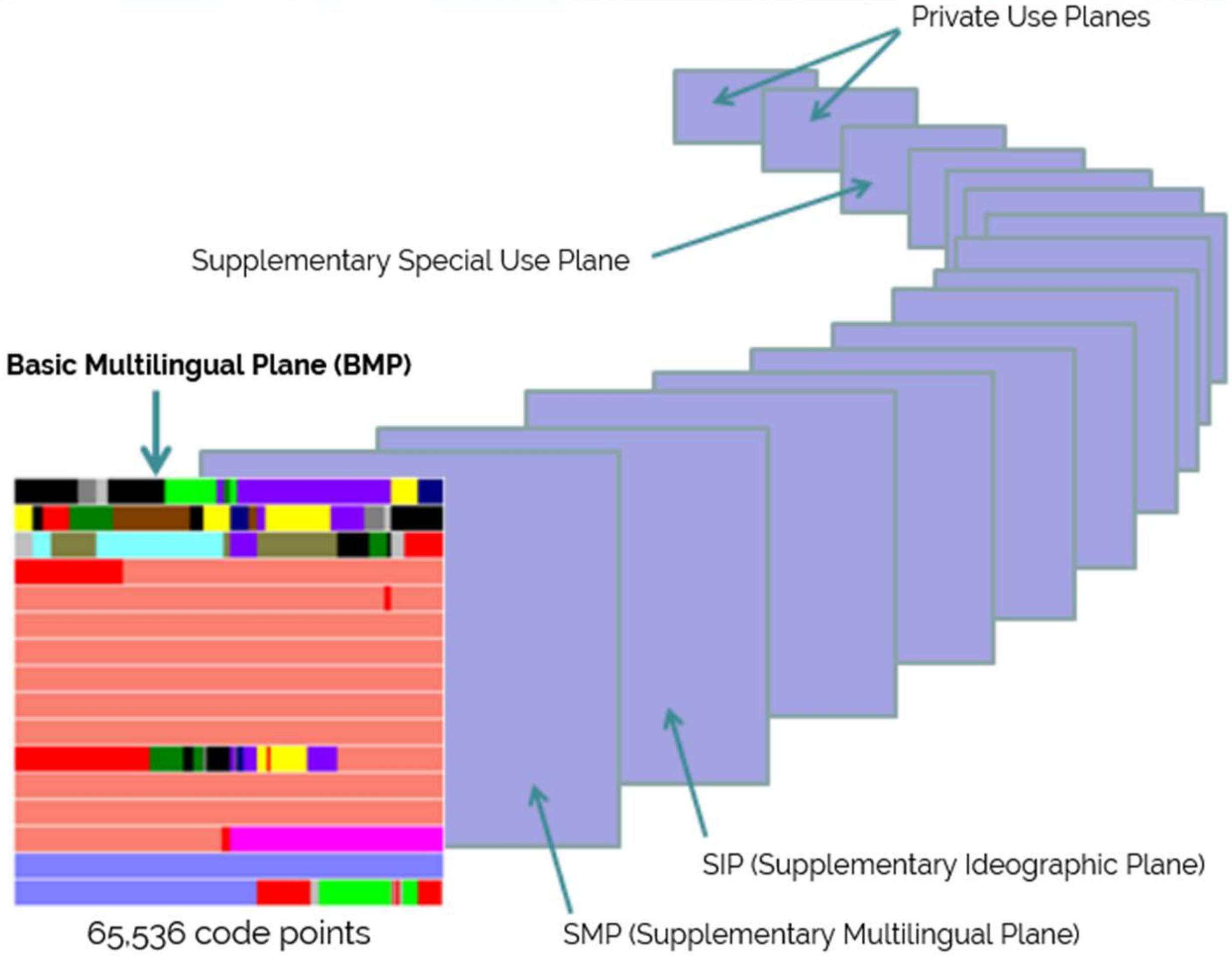
Unicode

- Unicode symbols are usually stored on 4 bytes
  - As storing 4 bytes per character is wasteful, Unicode can be defined using different encoding forms
  - In 1994 ISO-C standardized two encoding forms
    - Multi-byte characters
      - UTF-8 and UTF-16
      - Different width are used for different character, varying from one to several bytes
    - Wide characters
      - UTF-32
      - The same width is used for every character
      - **Easy** indexing (fixed-width) but space **inefficient**

The encoding UCS, UTF-1, and UTF-7 are obsolete

Unicode

In UTF-16 the first data unit set 65,536 code point positions.
These constitute the Basic Multilingual Plane (BMP), i.e., characters from 0x0000 to 0xFFFF.
The BMP includes most of the more commonly used characters.



The Unicode character set also contains space for around a million additional code point positions. Characters in this latter range are referred to as supplementary characters

Unicode Encodings

- UTF-8

The most popular; used in over 90% of websites on the World Wide Web as well as on most modern operating systems

- An 8-bit, **variable-width** encoding
- Uses from 1 to 4 units of 8-bits
  - 1 byte to represent characters in the ASCII set
  - 2 bytes for characters in several more alphabetic blocks
  - 3 bytes for the rest of the BMP
  - 4 bytes for supplementary characters
- For backward compatibility, the first 128 Unicode characters coincide with ASCII characters

C provides standard functions to convert formats

Unicode Encodings: An Example

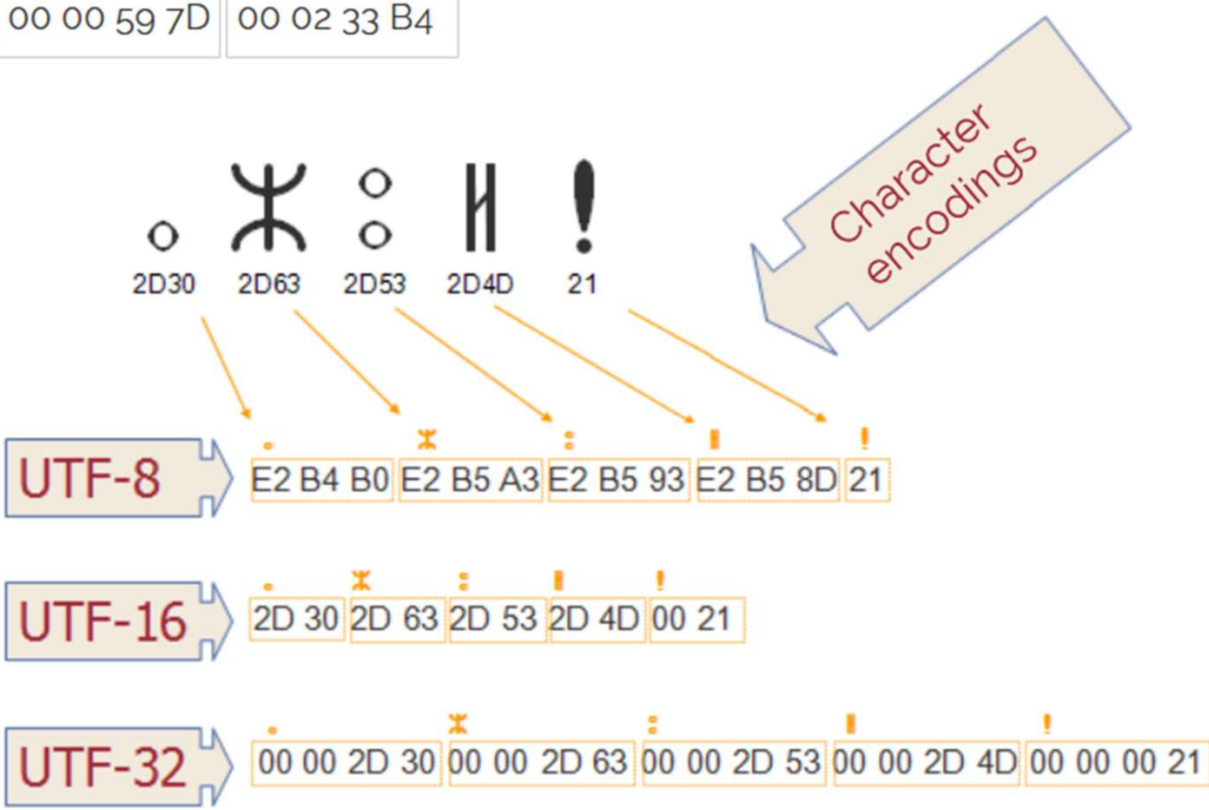
UTF-8

0xC0, 0xC1, 0xF5, 0xFF cannot appear in valid UTF-8 data

00000000 – 0000007F	0xxx xxxx			
00000080 – 000007FF	110x xxxx	10xx xxxx		
00000800 – 0000FFFF	1110 xxxx	10xx xxxx	10xx xxxx	
00010000 – 001FFFFF	1110 xxxx	10xx xxxx	10xx xxxx	10xx xxxx

Unicode Encodings: An Example

	A	א	好	𣎴
Code point	U+0041	U+05D0	U+597D	U+233B4
UTF-8	41	D7 90	E5 A5 BD	F0 A3 8E B4
UTF-16	00 41	05 D0	59 7D	D8 4C DF B4
UTF-32	00 00 00 41	00 00 05 D0	00 00 59 7D	00 02 33 B4



Unicode problems

- A few problems still remain
  - The order of the bytes depend on the **endianness** of the machine that created the text stream

The other bytes are placed in order in the next three bytes in memory

32 bits

- Big Endian
  - The most significant byte (the "big end") of the data is placed at the byte with the lowest address
  - 0x12345678 → 12 34 56 78

Increasing Adress

32 bits

- Little Endian
  - The least significant byte (the "little end") of the data is placed at the byte with the lowest address
  - 0x12345678 → 78 56 34 12

Increasing Adress

Unicode problems

- Which encoding is used to store the current file?
  - One counter-measure is the definition of a BOM (Byte Order Mark)
  - BOM = a special code-point (U+FEFF, zero width space) at the beginning of a text stream that indicates how the rest of the stream is encoded
  - It indicates both the UTF encoding and the endianess and is neutral to a text rendering engine
  - Unfortunately it is optional and many programmers claim their right to omit it, so accidents are still pretty common

The C library

https://en.cppreference.com/w/c/header

Header	Content
uchar.h	C11 UTF-16 and UTF-32 character utilities
wchar.h	Extended multiple-byte and wide character utilities
wctype.h	Functions to determine the type contained in wide characters

Example

```
#include <uchar.h>
#include <stdio.h>

#define N 3

char16_t *wcs[N] = {u"a", u"ß«, u"水", u"🍌"};
          // or "z\u00df\u6c34\U0001f34c"
for (int i=0; i<N; i++) {
  for (size_t j = 0; j < wcs[i][j] != 0; ++j) {
    printf("%#x ", wcs[i][j]);
  }
  printf("\n");
}
```

UTF-16

Displays:
07xa 0xdf 0x6c34 0xd83c 0xd4fc

Example

```
#include <uchar.h>
#include <stdio.h>

#define N 3

char32_t *ewcs[N] = {u"a", u"ß«, u"水", u"🍌"};
         // or "z\u00df\u6c34\U0001f34c"
for (int i=0; i<N; i++) {
  for (size_t j = 0; ewcs[i][j] != 0; ++j) {
    printf("%#x ", ewcs[i][j]);
  }
  printf("\n");
}
```

UTF-32

Displays:
07xa 0xdf 0x6c34 0x1f34c

Example

```
#include <stdio.h>
#include <locale.h>
#include <string.h>
#include <stdlib.h>
#include <wchar.h>

void print_mb(const char* ptr);

int main(void) {
  setlocale(LC_ALL, "en_US.utf8");
  // UTF-8 narrow multibyte encoding
  print_mb (u8"z\u00df\u6c34\U0001F34C");
           // or u8"zß水🍌"
}
```

Example

```
void print_mb(const char* ptr) {
  mbtowc(NULL, 0, 0); // reset the conversion state
  const char* end = ptr + strlen(ptr);
  int ret;
  for (wchar_t wc;
   (ret = mbtowc(&wc, ptr, end-ptr)) > 0;
    ptr+=ret) {
      wprintf (L"%lc", wc);
  }
  wprintf (L"\n");
}
```

Displays:
zß水🍌

Text and binary files

- A file is basically a sequence of bytes written one after the other on a physical device
  - Each byte includes 8 (or more) bits, with possible values 0 or 1
  - As a consequence all files are binary
- However, most people classify files in two categories
  - Text files (or ASCII)
  - Binary files

C sources, C++, Java, Python, etc.

Executables, Word, Excel, etc.

Remark:
The UNIX/Linux kernel does not distinguish between binary and textual files

Text files

- Files consisting of data encoded in ASCII (or Unicode)
  - Sequence of 0 and 1, which (in groups of 8 or more bits) codify ASCII (or Unicode) symbols
- ASCII files
  - Are stored as a sequence of binary values, i.e., a sequence of 1's and 0's
  - Are basically binary files, because they store binary numbers
  - **Are binary files that store ASCII (Unicode) codes**

Text files

- Text file are usually line-oriented
  - A newline is a set of bytes which convince the computer to go at the beginning of the next row
    - In UNIX/Linux and Mac OSX a newline is represented by a single character
      - Line Feed (go to next line, LF, 10_{10})
    - In Windows a newline is represented by two characters (as former mechanical typewriters)
      - Line Feed (go to next line, LF, 10_{10})
      - Carriage Return (push the carriage at the beginning of the line, CR , 13_{10})

Binary Files

- A sequence of 0 and 1, not "byte-oriented"
- The smallest unit that can be read/write is the bit
  - Non easy the management of the single bit
    - It's difficult to edit a binary file as individual bits should be edited
  - They usually include every possible sequence of 8 bits, which do not necessarily correspond to printable characters, new-line, etc.

Binary Files

- Why do people use binary files anyway?
  - Compactness
    - Example
      - Number 100000_{10}
      - Text/ASCII format
        - 6 characters, i.e., 6 bytes
      - Binary format
        - 100000_{10} is an integer value and it can be stored using 4 bytes

Example

String
Text or binary file

```
"ciao" → 'c'  'i'  'a'  'o'
ASCII        63 69 61 6F
UTF-16       0063 0069 0061 006F
UTF-32       00000063 00000069 00000061 0000006F
```

Base = 16
$6_{16} = 0110_2$, etc.

```
"231" → '2'  '3'  '1'
ASCII        32 33 31
UTF-16       0032 0033 0031
UTF-32       00000032 00000033 00000031
```

Integer number
Text file

"231" → "231_{10}"
11100111_2

Integer number
Binary file

Serialization

- In the context of data storage, **serialization** is the process of translating **data structure** or **objects** into a format that can be stored as a single entity
  - The process of serializing an object is also called **marschalling** an object
- The opposite operation, extracting a data structure from a series of bytes, is **deserialization**
  - Deserialization is also called **unmarschalling**

Serialization

- Using serialization
  - A structure can be stored in a file (or transmitted across a network connection link) as a unique entity
    - Manipulating single fields is **not** required !
  - When it is reconstructed (or received) later the same serialization format must be used to create a semantically identical clone of the original object

```
struct mys {
  int id;
  char name[L];
  ...
} s;
```

Serialization

- Serialization breaks the opacity of an abstract data type (ADT) by potentially exposing private implementation details
  - Trivial implementations which serialize all data members may violate encapsulation
  - For complex objects, such as those that uses references, this process is not straightforward
- Several languages directly support object serialization (or object archival)

```
struct mys {
  int id;
  char *name;
  ...
} s;
```

Example

Binary manipulation of a structure as a unique object

```
struct mys {
  int id;
  long int rn;
  char n[L], s[L];
  int mark;
} s;
```

```
fprintf(fp, "%d %ld %s %s",
  s.id, s.rn, s.n, s.s, s.mark);

write (fd, &s, sizeof(struct mys));
```

Write in text format (on file pointer fp)

Write in binary format (on file descriptor fd)

Example

Binary manipulation of a structure as a unique object

```
struct mys {
  int id;
  long int rn;
  char n[L], s[L];
  int mark;
} s;
```

```
1 100000 Romano Antonio 25
```

Text:
Single fields
Characters on 8 bits (ASCII)

```
fprintf(fp, "%d %ld %s %s",
  s.id, s.rn, s.n, s.s, s.mark);
```

Write in text format (on file pointer fp)

Write in binary format (on file descriptor fd)

Example

Binary manipulation of a structure as a unique object

```
struct mys {
  int id;
  long int rn;
  char n[L], s[L];
  int mark;
} s;
```

```
1 100000 Romano Antonio 25

† Romano ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
Antonio ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
```

Binary:
Entire structure
Ctr on 8 bits (ASCII)

```
fprintf(fp, "%d %ld %s %s",
  s.id, s.rn, s.n, s.s, s.mark);

write (fd, &s, sizeof(struct mys));
```

Write in text format (on file pointer fp)

Write in binary format (on file descriptor fd)

Example

Binary manipulation of a structure as a unique object

```
struct mys {
  int id;
  long int rn;
  char n[L], s[L];
  int mark;
```

Binary:
Entire structure
Ctr on 16 bits (UTF-16)
(note file size)

```
1 100000 Romano Antonio 25

† Romano ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
Antonio ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ

??? †?R?o?m?a?n?o???ÌÌÌÌÌÌÌÌÌÌÌÌ
ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
ÌÌA?n?t?o?n?i?o???ÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
```

```
fprintf(fp, "%d %ld %s %s",
  s.id, s.rn, s.n, s.s, s.mark);

write (fd, &s, sizeof(struct mys));
```

Write in text format (on file pointer fp)

Write in binary format (on file descriptor fd)