EECS 440 System Design of a Search Engine

Winter 2021

Lecture 4: HTML, Utf8, HTTP and redirects

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

details

1. Need to finalize the groups right away.
2. Hope to get the HW3 AG up tonight (or I may have to move the due date.)
3. Should we do tee shirts?

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

The basic parts to a Search Engine

1. HTML parser.
2. Crawler.
3. Index.
4. Constraint solver.
5. Query language.
6. Ranker.
7. Front end.

Task: Extract the content from a HTML file as:

1. A series of tokens in the contents and
2. A set of links with anchor text to other documents.

The basic parts to a Search Engine

1. HTML parser.
2. Crawler.
3. Index.
4. Constraint solver.
5. Query language.
6. Ranker.
7. Front end.

Depends on understanding how a browser talks to a webpage.

The basic parts to a Search Engine

1. HTML parser.
2. Crawler.
3. Index.
4. Constraint solver.
5. Query language.
6. Ranker.
7. Front end.

Topics:

1. HTML as a markup language for the contents of a webpage.
2. Basic HTTP handshake and redirects.

Recurring problem

Need to serialize and deserialize data.

1. Lossy or lossless.
2. Human-readable or binary.
3. Stateless or stateful.

Recurring problem

Need to serialize and deserialize data.

1. Lossy or lossless.
2. Human-readable or binary.
3. Stateless or stateful and where any state is stored.

Examples: Archive (tar, zip) and compressed (.jpg) file formats and the exchange of information between a browser and a webserver.

The webpage problem

Serialize the text on the page along with formatting information, include images and hyperlinks to other pages.

Human-readable because it was written by hand.

Tim Berners-Lee

Image source: https://en.wikipedia.org/wiki/File:Tim_Berners-Lee_April_2009.jpg

The World Wide Web is born

1980 Tim Berners-Lee, a contractor at CERN, creates a prototype system to share documents.

1989 He invents HTML and wrote a browser and a server.

1991 First publicly available HTML specification. Defines only 18 elements. (There are now 139.)

Tim Berners-Lee

Image source: https://en.wikipedia.org/wiki/File:Tim_Berners-Lee_April_2009.jpg

XML

The web is linked through text files with hyperlinks.

The basic idea is similar to XML (Extensible Markup Language), intended as a way to allow human-readable text serialization of complex objects.

```
<myThing myProperties="...">
    anything I want
</myThing>
```

XML

The web is linked through text files with hyperlinks.

The basic idea is similar to XML (Extensible Markup Language), intended as a way to allow human-readable text serialization of complex objects.

```
<myThing myProperties="...">
   anything I want
</myThing>
```

It's recursive. This can be another XML structure.

XML

If the object doesn't enclose anything, it can be written in a short form.

```
<myLeafThing myProperties="..."/>
```

HTML

Hypertext Markup Language.

Loosely follows XML conventions.

Grown somewhat organically as the web grew in the 90s and browsers began offering their own features.

There are standards but not everyone cares.

Closing tags and both opening and closing quotes are notoriously absent a lot of the time.

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

   <title>Page title shown in the browser tab</title>
   <script>Lots of gibberish</script>

   <link href="MyStyles.css" rel="stylesheet" type="text/css" />
   <link href="https://mydomain/favicon.ico" rel="shortcut icon" />

   <meta name="DC.Rights" content="Copyright 2018 my name"/>
   <meta name="description" content="“Short abstract."/>
   <meta name="keywords" content="“arbitrary list of words"/>
</head>

<body>
:
</body>
</html>
```

The interesting content will be in the title, possibly the description and keywords, and the body.

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

   <title>Page title shown in the browser tab</title>
   <script>Lots of gibberish</script>

   <link href="MyStyles.css" rel="stylesheet" type="text/css" />
   <link href="https://mydomain/favicon.ico" rel="shortcut icon" />

   <meta name="DC.Rights" content="Copyright 2018 my name"/>
   <meta name="description" content="“Short abstract."/>
   <meta name="keywords" content="“arbitrary list of words"/>
</head>

<body>
:
</body>
</html>
```

The stylesheet contains CSS formatting information which we don’t really care about. The shortcut icon is to the little icon that appears on the browser tab.

CSS

```
html,body {
   height: 100%;
   background-color: #f2f2f7;
   color: black;
   font-family: Verdana, Arial, Helvetica, sans-serif;
   font-size: 0.9em;
}

p {
   line-height: 17px;
   margin-top:14px;
   margin-bottom:17px;
}

h1,h2,h3,h4,h5,h6 {
   color: #3d6c87;
}

a {
   color: #5f8ea9;
   text-decoration: none;
}

a:hover {
   color: #33627d;
}
```

The CSS describes fonts, colors, etc., to be used when drawing various HTML elements but it doesn't change the words on the page, only how they appear.

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

   <title>Page title shown in the browser tab</title>
   <script>Lots of gibberish</script>

   <link href="MyStyles.css" rel="stylesheet" type="text/css" />
   <link href="https://mydomain/favicon.ico" rel="shortcut icon" />

   <meta name="DC.Rights" content="Copyright 2018 my name"/>
   <meta name="description" content="“Short abstract."/>
   <meta name="keywords" content="“arbitrary list of words"/>
</head>

<body>
:
</body>
</html>
```

JavaScript may be used to generate the content and on some pages, scripts generate ALL the content. Not clear how many engines index content generated from scripts.

HTML

```
<body>
   <div id="masthead" style="padding-top: 20px; padding-left:74px">
   :
   </div>

   <div id="content">
   :
   </div>

   <div id="sidebar">
   :
   </div>
</body>
```

The <body> contains the actual content of the page.

 … sections can be used to create separate panels on the page, applying different CSS styles.

HTML

```
<div id="content">

   <a href="clickURL1">
      <img src="myimage.jpg" .../>
   </a>

   <p>
   Plain text<br/>
   Link to <a href="https://www.nytimes.com">The New York Times</a>
   </p>

</div>
```

Hyperlinks are marked with <a> tags. The text between the opening <a> and closing tags are the anchor text and are very relevant for ranking.

HTML

```
<div id="content">

   <img src="mypath.png" alt="..." width="..."/>

   <h1 style="...">This is a big heading</h1>

   <p>
   :
   </p>

   <h3 style="...">This is a smaller heading</h3>

   <p>
   :
   </p>

   <p>
   :
   </p>

</div>
```

Images are added with <img> tags.

Heading use <h1>, <h2>, etc., tags.

Paragraphs are marked with tags. The closing are often missing.

HTML

```
<div id="content">

   <p>
   if ( b<a ) ...
   </p>

</div>
```

```
<div id="content">

   <p>
   if ( b&lt;a ) ...
   </p>

</div>
```

Special characters are written using &-escapes.

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

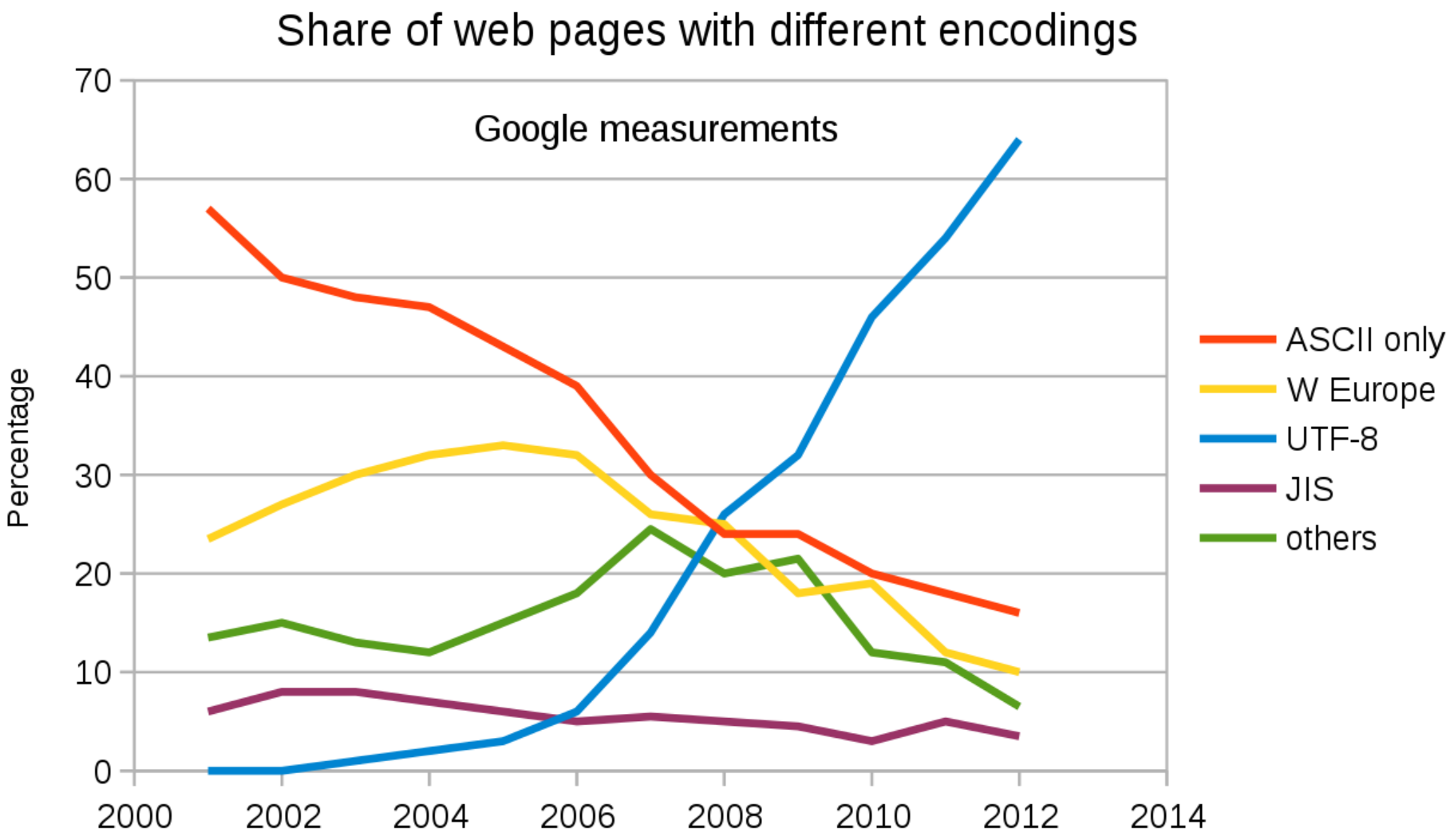


Image source: https://en.wikipedia.org/wiki/UTF-8

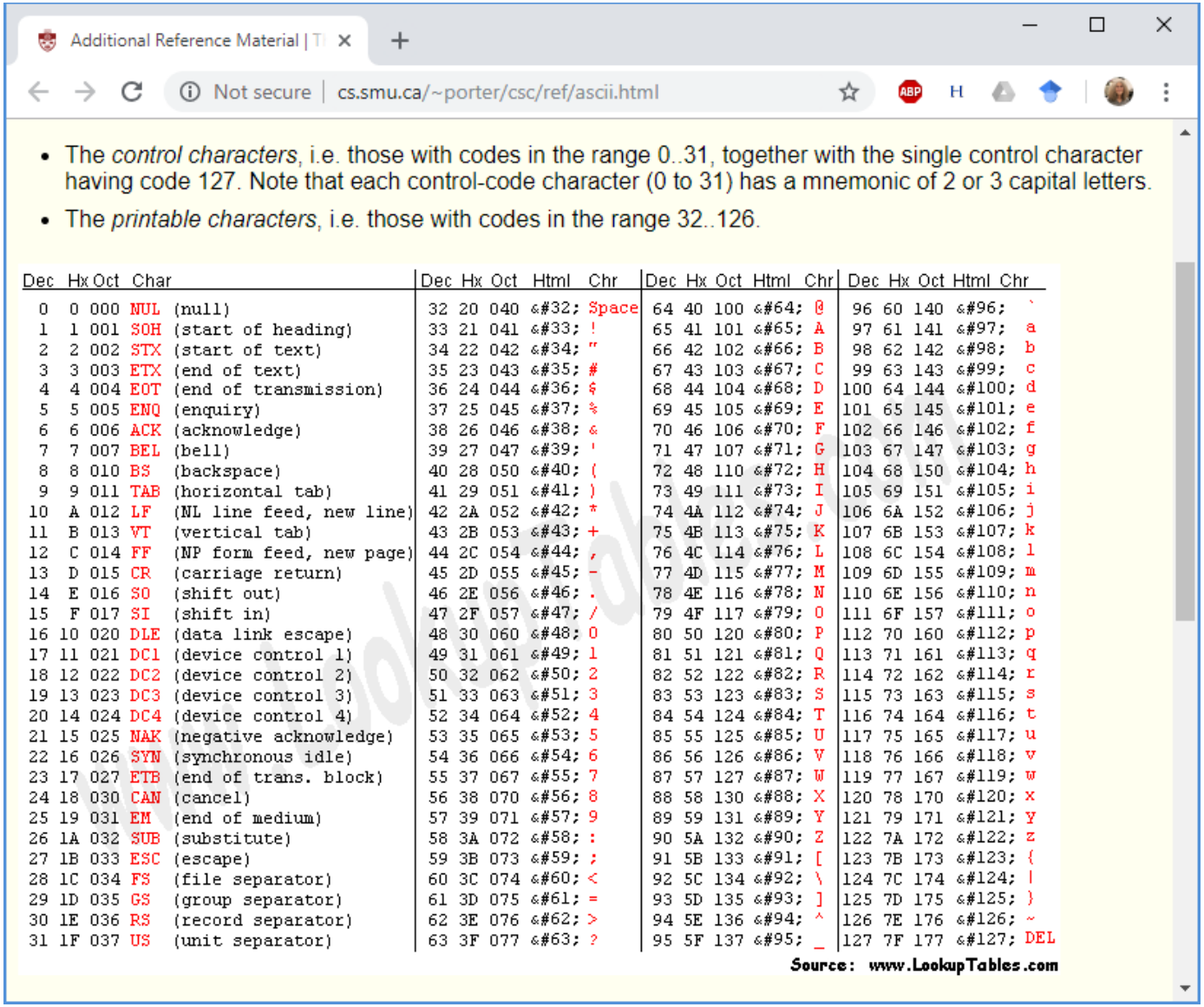
Codesets

Characters will be encoded as:

ASCII	7 bits per character.
ANSI	8 bits per character, *nationalized* with codepages for character values > 0x7f.
Unicode	16 bits per character .
UTF-8	1 to 6 bytes used to encode characters of 7 to 31 bits.

There will sometimes be a BOM (byte order mark) at the beginning.

Seven bit ASCII

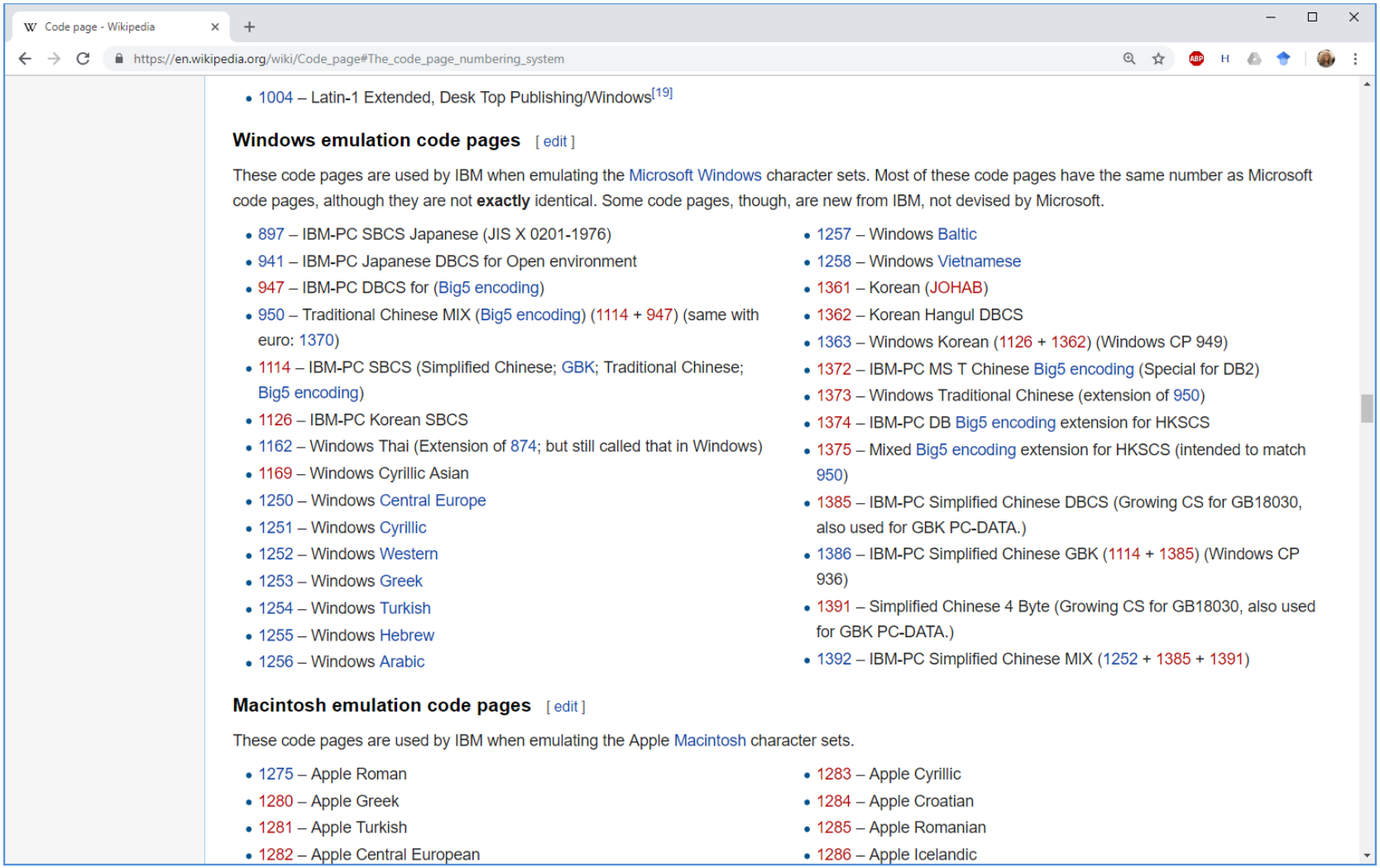


- The *control characters*, i.e. those with codes in the range 0..31, together with the single control character having code 127. Note that each control-code character (0 to 31) has a mnemonic of 2 or 3 capital letters.
- The *printable characters*, i.e. those with codes in the range 32..126.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Source: http://cs.smu.ca/~porter/csc/ref/ascii.html



- 1004 – Latin-1 Extended, Desk Top Publishing/Windows[19]

Windows emulation code pages [edit]

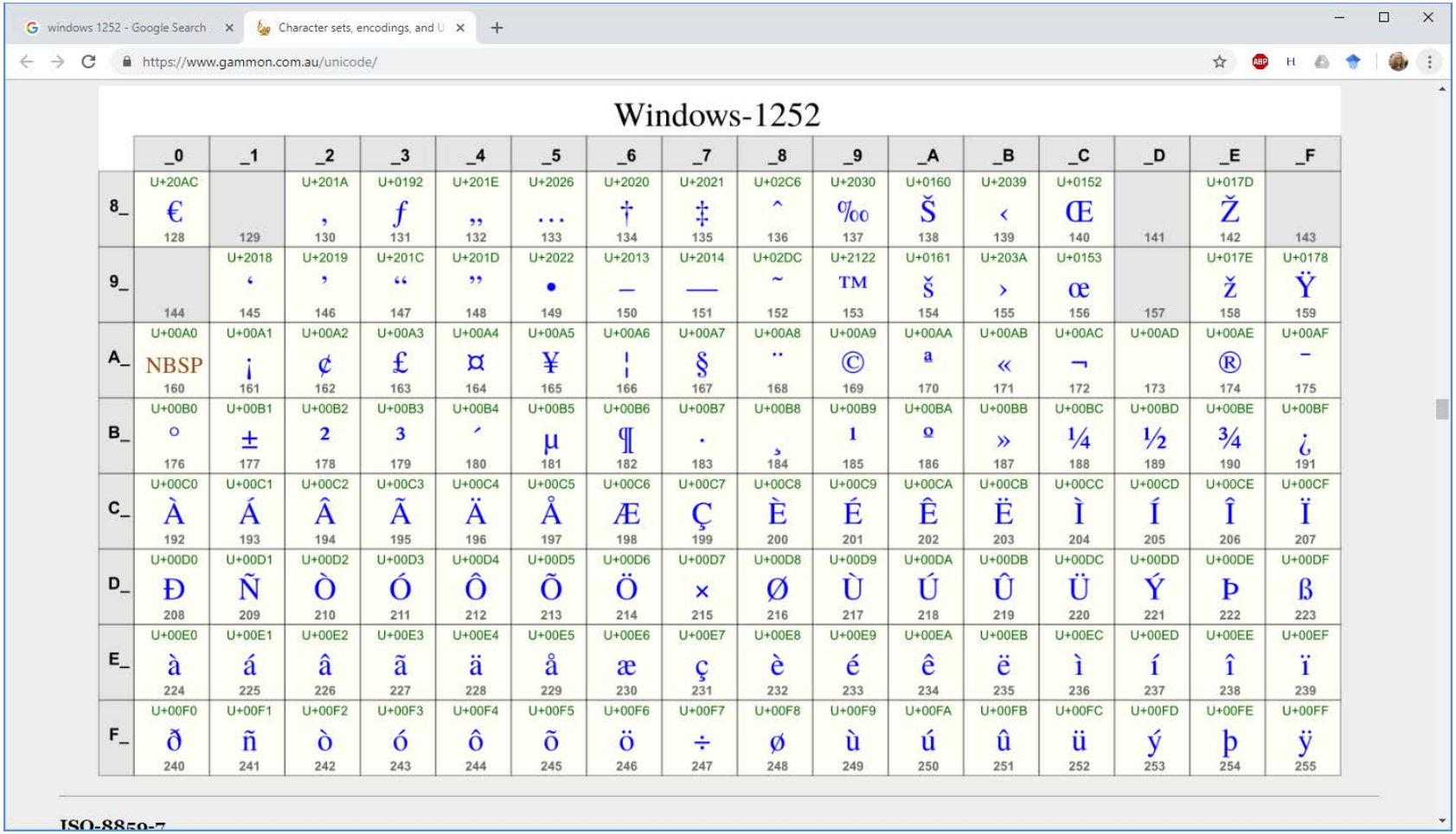
These code pages are used by IBM when emulating the Microsoft Windows character sets. Most of these code pages have the same number as Microsoft code pages, although they are not **exactly** identical. Some code pages, though, are new from IBM, not devised by Microsoft.

- 897 – IBM-PC SBCS Japanese (JIS X 0201-1976)
- 941 – IBM-PC Japanese DBCS for Open environment
- 947 – IBM-PC DBCS for (Big5 encoding)
- 950 – Traditional Chinese MIX (Big5 encoding) (1114 + 947) (same with euro: 1370)
- 1114 – IBM-PC SBCS (Simplified Chinese; GBK; Traditional Chinese; Big5 encoding)
- 1126 – IBM-PC Korean SBCS
- 1162 – Windows Thai (Extension of 874; but still called that in Windows)
- 1169 – Windows Cyrillic Asian
- 1250 – Windows Central Europe
- 1251 – Windows Cyrillic
- 1252 – Windows Western
- 1253 – Windows Greek
- 1254 – Windows Turkish
- 1255 – Windows Hebrew
- 1256 – Windows Arabic
- 1257 – Windows Baltic
- 1258 – Windows Vietnamese
- 1361 – Korean (JOHAB)
- 1362 – Korean Hangul DBCS
- 1363 – Windows Korean (1126 + 1362) (Windows CP 949)
- 1372 – IBM-PC MS T Chinese Big5 encoding (Special for DB2)
- 1373 – Windows Traditional Chinese (extension of 950)
- 1374 – IBM-PC DB Big5 encoding extension for HKSCS
- 1375 – Mixed Big5 encoding extension for HKSCS (intended to match 950)
- 1385 – IBM-PC Simplified Chinese DBCS (Growing CS for GB18030, also used for GBK PC-DATA.)
- 1386 – IBM-PC Simplified Chinese GBK (1114 + 1385) (Windows CP 936)
- 1391 – Simplified Chinese 4 Byte (Growing CS for GB18030, also used for GBK PC-DATA.)
- 1392 – IBM-PC Simplified Chinese MIX (1252 + 1385 + 1391)

Macintosh emulation code pages [edit]

These code pages are used by IBM when emulating the Apple Macintosh character sets.

- 1275 – Apple Roman
- 1280 – Apple Greek
- 1281 – Apple Turkish
- 1282 – Apple Central European
- 1283 – Apple Cyrillic
- 1284 – Apple Croatian
- 1285 – Apple Romanian
- 1286 – Apple Icelandic



Windows-1252

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
8_	U+20AC € 128	129	U+201A ‚ 130	U+0192 ƒ 131	U+201E „ 132	U+2026 … 133	U+2020 † 134	U+2021 ‡ 135	U+02C6 ˆ 136	U+2030 ‰ 137	U+0160 Š 138	U+2039 ‹ 139	U+0152 Œ 140	141	U+017D Ž 142	143
9_	144	U+2018 ‘ 145	U+2019 ’ 146	U+201C “ 147	U+201D ” 148	U+2022 • 149	U+2013 – 150	U+2014 — 151	U+02DC ˜ 152	U+2122 ™ 153	U+0161 š 154	U+203A › 155	U+0153 œ 156	157	U+017E ž 158	U+0178 Ÿ 159
A_	U+00A0 NBSP 160	U+00A1 ¡ 161	U+00A2 ¢ 162	U+00A3 £ 163	U+00A4 ¤ 164	U+00A5 ¥ 165	U+00A6 ¦ 166	U+00A7 § 167	U+00A8 ¨ 168	U+00A9 © 169	U+00AA ª 170	U+00AB « 171	U+00AC ¬ 172	U+00AD 173	U+00AE ® 174	U+00AF ¯ 175
B_	U+00B0 ° 176	U+00B1 ± 177	U+00B2 ² 178	U+00B3 ³ 179	U+00B4 ´ 180	U+00B5 µ 181	U+00B6 ¶ 182	U+00B7 · 183	U+00B8 ¸ 184	U+00B9 ¹ 185	U+00BA º 186	U+00BB » 187	U+00BC ¼ 188	U+00BD ½ 189	U+00BE ¾ 190	U+00BF ¿ 191
C_	U+00C0 À 192	U+00C1 Á 193	U+00C2 Â 194	U+00C3 Ã 195	U+00C4 Ä 196	U+00C5 Å 197	U+00C6 Æ 198	U+00C7 Ç 199	U+00C8 È 200	U+00C9 É 201	U+00CA Ê 202	U+00CB Ë 203	U+00CC Ì 204	U+00CD Í 205	U+00CE Î 206	U+00CF Ï 207
D_	U+00D0 Ð 208	U+00D1 Ñ 209	U+00D2 Ò 210	U+00D3 Ó 211	U+00D4 Ô 212	U+00D5 Õ 213	U+00D6 Ö 214	U+00D7 × 215	U+00D8 Ø 216	U+00D9 Ù 217	U+00DA Ú 218	U+00DB Û 219	U+00DC Ü 220	U+00DD Ý 221	U+00DE Þ 222	U+00DF ß 223
E_	U+00E0 à 224	U+00E1 á 225	U+00E2 â 226	U+00E3 ã 227	U+00E4 ä 228	U+00E5 å 229	U+00E6 æ 230	U+00E7 ç 231	U+00E8 è 232	U+00E9 é 233	U+00EA ê 234	U+00EB ë 235	U+00EC ì 236	U+00ED í 237	U+00EE î 238	U+00EF ï 239
F_	U+00F0 ð 240	U+00F1 ñ 241	U+00F2 ò 242	U+00F3 ó 243	U+00F4 ô 244	U+00F5 õ 245	U+00F6 ö 246	U+00F7 ÷ 247	U+00F8 ø 248	U+00F9 ù 249	U+00FA ú 250	U+00FB û 251	U+00FC ü 252	U+00FD ý 253	U+00FE þ 254	U+00FF ÿ 255

ISO-8859-7

Source: https://www.gammon.com.au/unicode/

Unicode

Universal Coded Character Set.

16-bit characters = 65,536 code points.

With multiple symbol sets, currently 137,220 characters defined.

Covers 139 modern and historic scripts + multiple symbol sets.

ISO/IEC 10646 standard maintained by the Unicode Consortium.

History of Unicode

Version 1.0.0 October 1991.

Used internally in Windows NT.

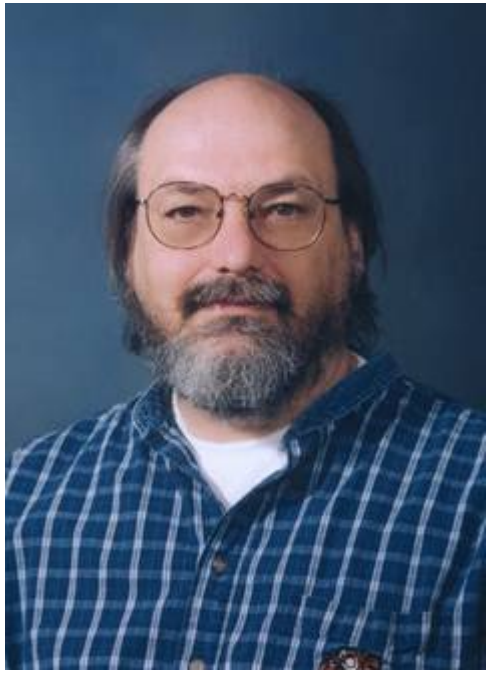
Did not see wider adoption.

Incompatible with ASCII.

Seen as wasteful if most of the text was ASCII.

UTF-8

Invented by Ken Thompson and Rob Pike, allegedly on a napkin over a meal.



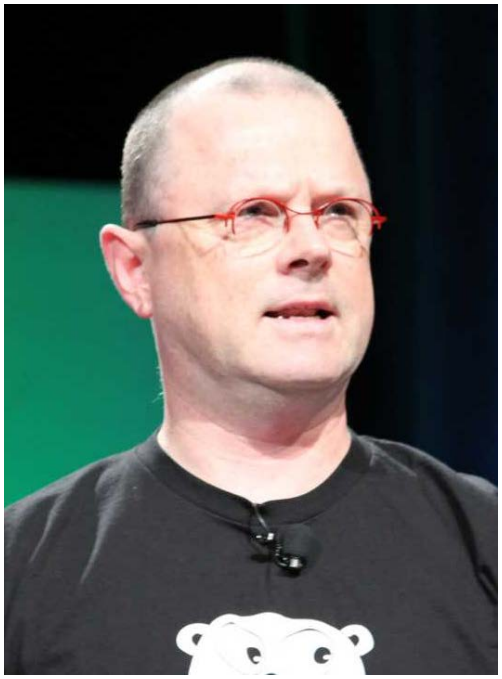


Image sources: https://en.wikipedia.org/wiki/Ken_Thompson
https://en.wikipedia.org/wiki/Rob_Pike

UTF-8

1. Backward compatible with 7-bit ASCII.
2. Characters > 0x7F require more bytes.
3. The larger the character value, the more bytes.
4. Each additional bytes gives you an additional 6 bits but you lose one from the first byte.
5. Last byte marked by a high-order zero.
6. Some sequences are invalid.
7. Capable of encoding 31 bits (Utf-32) but only Unicode in common use.

UTF-8 sequences

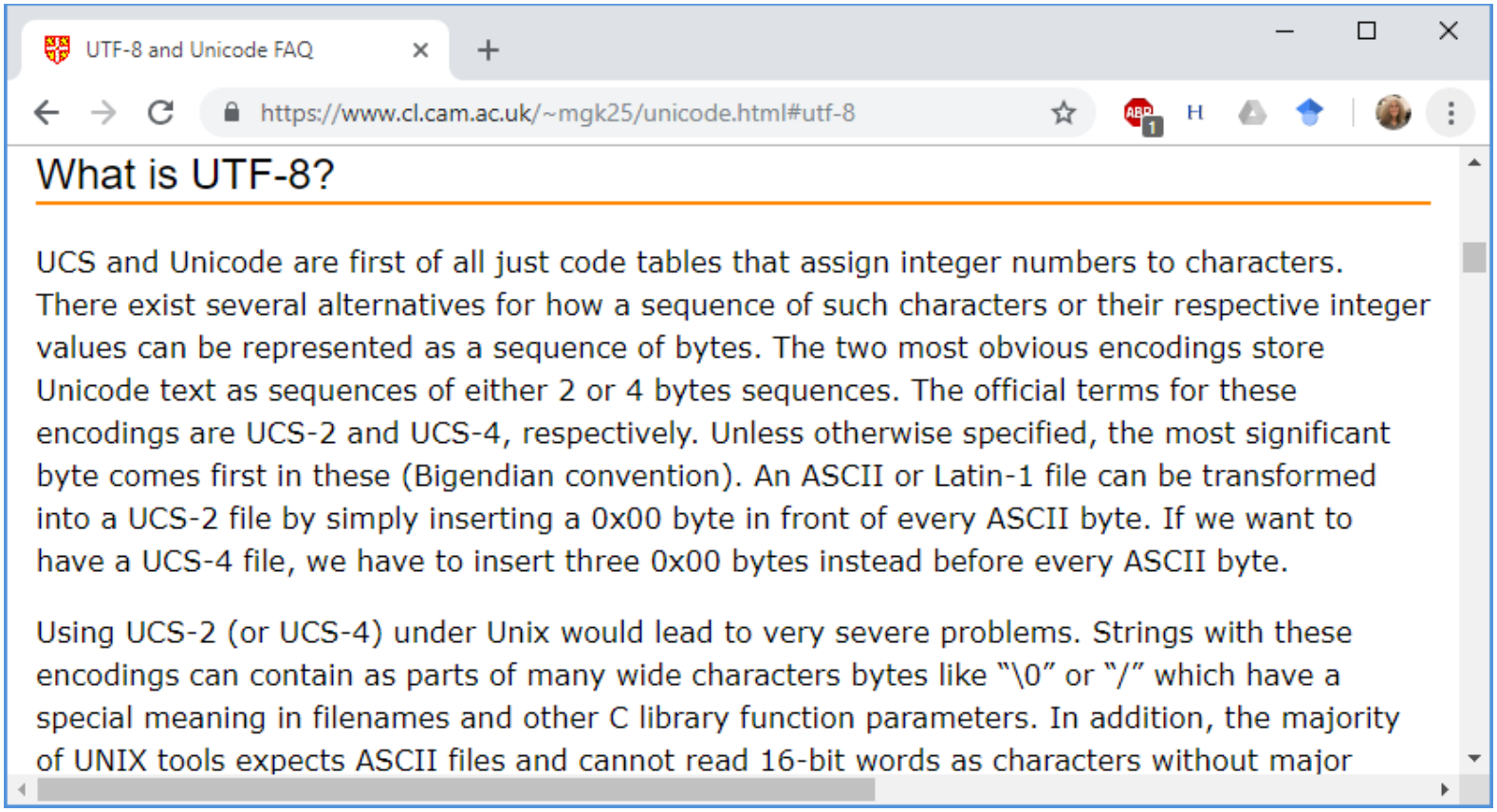
Valid UTF-8 sequences

```
U-00000000 - U-0000007F:   0xxxxxxx 7 bits
U-00000080 - U-000007FF:   110xxxxx 10xxxxxx 11 bits
U-00000800 - U-0000FFFF:   1110xxxx 10xxxxxx 10xxxxxx 16 bits
U-00010000 - U-001FFFFF:   11110xxx 10xxxxxx 10xxxxxx 10xxxxxx 21 bits
U-00200000 - U-03FFFFFF:   111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 26 bits
U-04000000 - U-7FFFFFFF:   1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 31 bits
```

Invalid, Overlong UTF-8 Sequences which should never be accepted because of the *security risk* and are replaced with `0xfffd`:

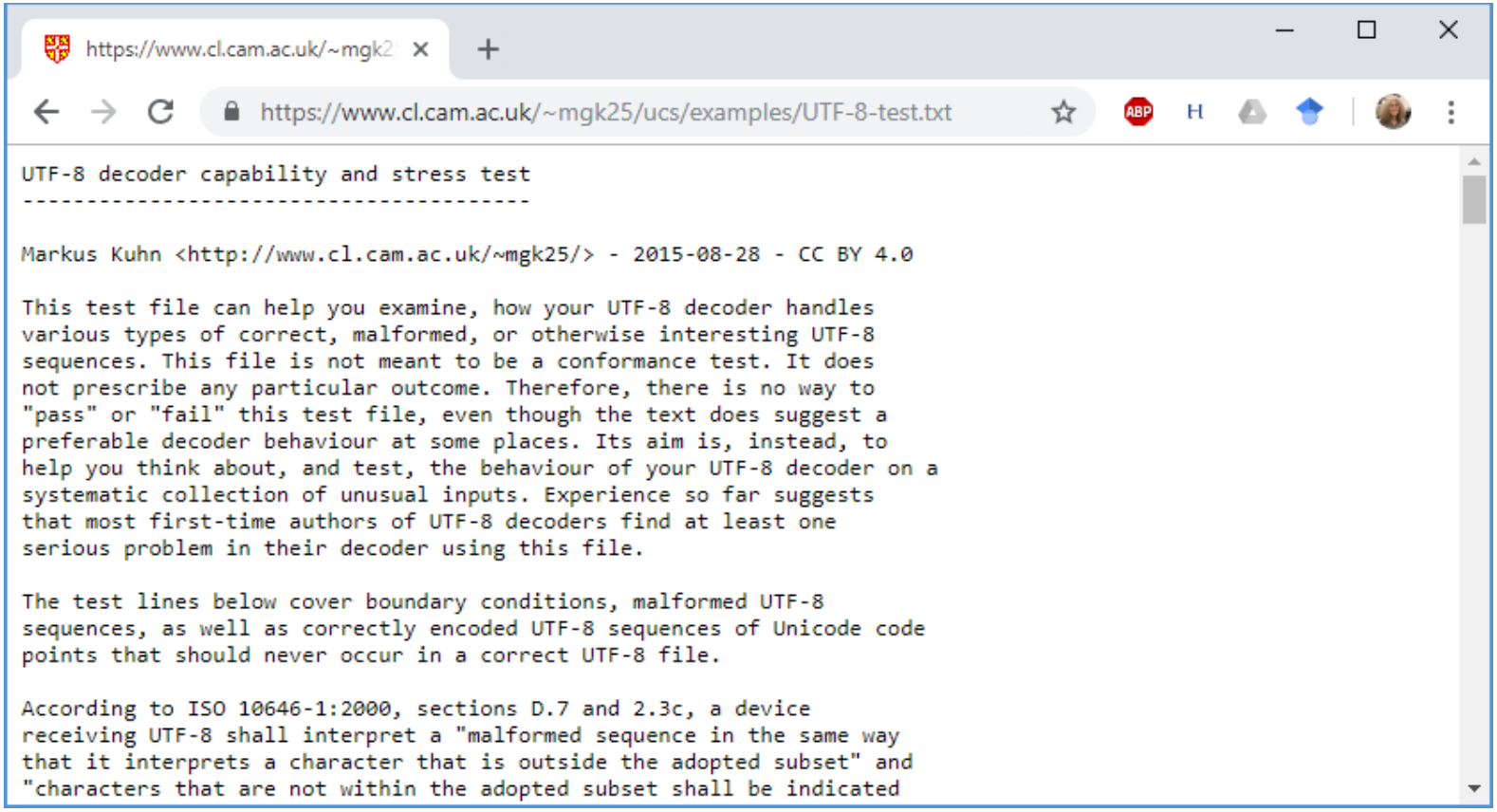
```
11 bits to encode 7        1100000x 10xxxxxx
16 bits to encode 11       11100000 100xxxxx 10xxxxxx
21 bits to encode 16       11110000 1000xxxx 10xxxxxx 10xxxxxx
26 bits to encode 21       11111000 10000xxx 10xxxxxx 10xxxxxx 10xxxxxx
31 bits to encode 26       11111100 100000xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Utf8 FAQ



UTF-8 and Unicode FAQ
http://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8

Utf8 test case



UTF-8 decoder capability and stress test
http://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt

Not a homework

```
#include <cstddef>
#include <cstdint>
typedef uint32_t Utf32;
typedef uint16_t Unicode;
typedef uint8_t  Utf8;

// SizeOfUtf8 tells the number of bytes it will take to encode the
// specified Unicode value.

int SizeOfUtf8( Unicode c );

// Get the UTF-8 character as a Unicode value.
// If it's an invalid UTF-8 encoding for a U-16
// character, return the special malformed
// character code.

Unicode GetUtf8( Utf8 *p );

// NextUtf8 will scan forward to the next byte
// which could be the start of a UTF-8 character.
// If it's on a null character, it scans over it.

Utf8 *NextUtf8( Utf8 *p );
```

```
// Scan backward for the first PREVIOUS byte which could
// be the start of a UTF-8 character.

Utf8 *PreviousUtf8( Utf8 *p );

// Write a Unicode character in UTF-8.

Utf8 *WriteUtf8( Utf8 *p, Unicode c );

// UTF-8 String compares.
// Same return values as strcmp( ).

int StringCompare( Utf8 *a, Utf8 *b );

// Unicode string compare up to 'N' UTF-8 characters (not bytes)
// from two UTF-8 strings.

int StringCompare( Utf8 *a, Utf8 *b, size_t N );
```

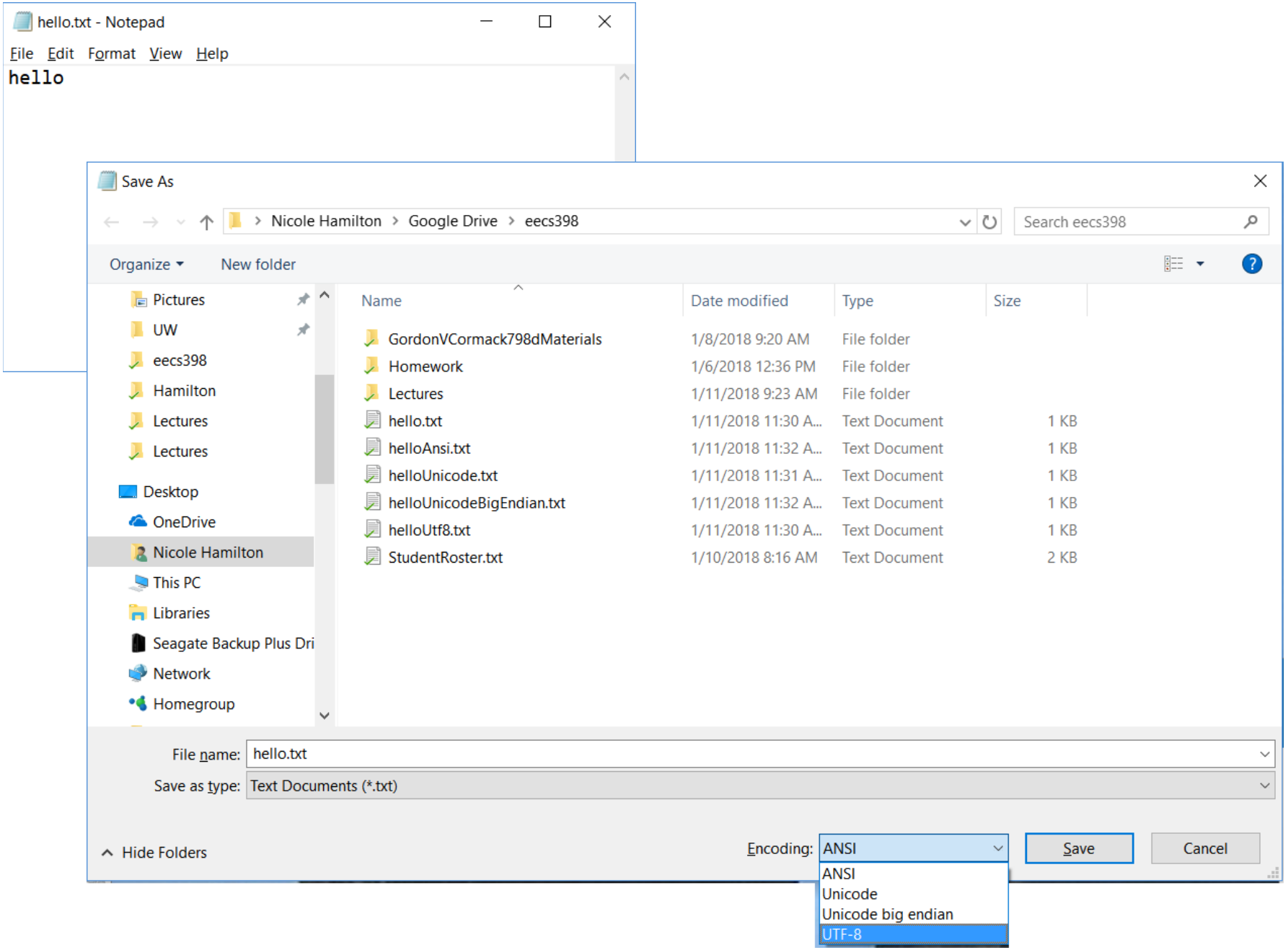
Byte Order Marks

A BOM may tell you how a document is encoded in the first 2 or 3 bytes bytes.

`ff fe`	Unicode
`ef bb bf`	Utf-8
`fe ff`	Big-endian Unicode

If there's no BOM, it's generally ASCII (or ANSI).

Conforming applications like Windows Notepad do this.


hello.txt - Notepad
File Edit Format View Help
hello
Save As
Nicole Hamilton › Google Drive › eecs398
Search eecs398
Organize
New folder
Pictures
UW
eecs398
Hamilton
Lectures
Lectures
Desktop
OneDrive
Nicole Hamilton
This PC
Libraries
Seagate Backup Plus Dri
Network
Homegroup
Name
Date modified
Type
Size
GordonVCormack798dMaterials 1/8/2018 9:20 AM File folder
Homework 1/6/2018 12:36 PM File folder
Lectures 1/11/2018 9:23 AM File folder
hello.txt 1/11/2018 11:30 A... Text Document 1 KB
helloAnsi.txt 1/11/2018 11:32 A... Text Document 1 KB
helloUnicode.txt 1/11/2018 11:31 A... Text Document 1 KB
helloUnicodeBigEndian.txt 1/11/2018 11:32 A... Text Document 1 KB
helloUtf8.txt 1/11/2018 11:30 A... Text Document 1 KB
StudentRoster.txt 1/10/2018 8:16 AM Text Document 2 KB
File name: hello.txt
Save as type: Text Documents (*.txt)
Hide Folders
Encoding: ANSI
Save
Cancel
ANSI
Unicode
Unicode big endian
UTF-8

```
256 C% ls hello*.txt
hello.txt                 helloUnicode.txt          helloUtf8.txt
helloAnsi.txt             helloUnicodeBigEndian.txt
257 C% xd hello.txt
00000000:   6865 6c6c  6f0d 0a                            *hello..*
258 C% xd helloAnsi.txt
00000000:   6865 6c6c  6f0d 0a                            *hello..*
259 C% cmp hello.txt helloAnsi.txt
260 C% xd helloUtf8.txt
00000000:   efbb bf68  656c 6c6f   0d0a                   *...hello ..*
261 C% xd helloUnicode.txt
00000000:   fffe 6800  6500 6c00   6c00 6f00  0d00 0a00    *..h.e.l. l.o.....*
262 C% xd helloUnicodeBigEndian.txt
00000000:   feff 0068  0065 006c   006c 006f  000d 000a    *...h.e.l .l.o....*
263 C%
```

If there was no BOM, how would you decide what it was?

If you guessed wrong, how would you know?

How would you detect binary files?

HTML Parser

Extract the content from a HTML file as a series of tokens in the title and the body of the document and a set of links with anchor text to other documents.

Up to you to decide:

1. *What information should you collect?*
2. *How will you deal with malformed content?*
3. *How should the information you collect be represented as an object?*

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

URLs

Basic format	*protocol:path*
Common protocols	http: https: ftp: mailto: file:

The path may be relative to the directory of the referring page or it may start at a root of the website.

http: and https: are the protocols for reading web pages.

We will do this by opening sockets to these URLs in C++ but for now, you can use tools like curl to experiment.

HTTP and HTTPS

These are protocols for exchanging content between a server and a browser.

It’s a human-readable format of requests and responses.

The difference between HTTP and HTTPS is whether it’s conducted over a secure socket layer (SSL) with encryption.

HTTP and HTTPS

It’s a simple handshake for exchanging data.

1. Browser sends GET or other method + a path and a protocol.
2. Server returns an OK + the requested content.
3. Any number of additional headers in any order.
4. Every line terminated with \r\n.
5. End of the header marked by a blank line, followed by the content.

Browser requests a page with a GET.

```
GET / HTTP/1.1
Host: www.nytimes.com
User-Agent: LinuxGetSsl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close
```

Server responds with an HTTP 200 OK message, a bunch of optional HTTP headers, a blank line and then the content.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 1150280
Server: nginx
Content-Type: text/html; charset=utf-8
x-nyt-data-last-modified: Mon, 01 Feb 2021 15:59:19 GMT
Last-Modified: Mon, 01 Feb 2021 15:59:19 GMT
:

<!DOCTYPE html>
<html lang="en-US"
xmlns:og="http://opengraphprotocol.org/schema/">
:
```

As seen with the server you'll build, the browser requests a page.

```
tcsh-29% sudo ./LinuxTinyServer 5000 '/mnt/c/Users/hamil/Google Drive/UMich
Faculty Page/'
Listening on 0.0.0.0:5000

Connection accepted from 127.0.0.1:54588

Connection accepted from 127.0.0.1:54589

GET /index.htm HTTP/1.1
Host: localhost:5000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3
Sec-Fetch-Site: none
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

This happens to be a request from a Chrome browser. Notice the user agent.

```
tcsh-29% sudo ./LinuxTinyServer 5000 '/mnt/c/Users/hamil/Google Drive/UMich
Faculty Page/'
Listening on 0.0.0.0:5000

Connection accepted from 127.0.0.1:54588

Connection accepted from 127.0.0.1:54589

GET /index.htm HTTP/1.1
Host: localhost:5000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3
Sec-Fetch-Site: none
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

In your own requests, identify yourself for complaints. Here's an example:

```
Connection accepted from 127.0.0.1:54659

GET /index.htm HTTP/1.1
Host: localhost
User-Agent: LinuxGetUrl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close
```

The server maps the requested /index.htm to the actual file and responds.

```
HTTP/1.1 200 OK
Content-Length: 8964
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

   <title>Nicole Hamilton</title>

   <link href="Styles/Hamilton.css" rel="stylesheet" type="text/css" />
   :
   :
```

Errors are reported as 400 and other codes. (You'll build your own LinuxGetUrl.)

```
tcsh-2% ./LinuxGetUrl http://localhost:5000/zork.htm
Service = http, Host = localhost, Port = 5000, Path = zork.htm
Host address length = 16 bytes
Family = 2, port = 5000, address = 127.0.0.1
GET /zork.htm HTTP/1.1
Host: localhost
User-Agent: LinuxGetUrl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close


HTTP/1.1 404 Not Found
Content-Length: 0
Connection: close

tcsh-3%
```

Agenda

1. Course details.
2. HTML.
3. Unicode and Utf8.
4. HTTP.
5. Redirects.

Often, you'll encounter redirects, e.g., from HTTP to HTTPS.

```
tcsh-8% ./LinuxGetUrl http://en.wikipedia.org
:
GET / HTTP/1.1
:


HTTP/1.1 301 TLS Redirect
Date: Thu, 12 Sep 2019 17:17:56 GMT
Server: Varnish
X-Varnish: 316194658
X-Cache: cp1087 int
X-Cache-Status: int-front
Server-Timing: cache;desc="int-front"
Set-Cookie: WMF-Last-Access=12-Sep-2019;Path=/;HttpOnly;secure;Expires=Mon, 14
Oct 2019 12:00:00 GMT
Set-Cookie: WMF-Last-Access-Global=12-Sep-
2019;Path=/;Domain=.wikipedia.org;HttpOnly;secure;Expires=Mon, 14 Oct 2019
12:00:00 GMThttps://en.wikipedia.org/
X-Client-IP: 68.51.181.4
Location: https://en.wikipedia.org/
Content-Length: 0
Connection: close

tcsh-9%
```

One redirect may lead to another. (You'll build your own LinuxGetSsl as well.)

```
tcsh-10% ./LinuxGetSsl https://en.wikipedia.org/
:
HTTP/1.1 301 Moved Permanently
Date: Thu, 12 Sep 2019 17:23:22 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: close
Server: mw1268.eqiad.wmnet
X-Powered-By: HHVM/3.18.6-dev
P3P: CP="See https://en.wikipedia.org/wiki/Special:CentralAutoLogin/P3P for
more info."
Cache-control: s-maxage=1200, must-revalidate, max-age=0
X-Content-Type-Options: nosniff
Location: https://en.wikipedia.org/wiki/Main_Page
Last-Modified: Thu, 12 Sep 2019 17:14:50 GMT
:

tcsh-11%
```