

EECS 498 System Design of a Search Engine

Winter 2026

Lecture 4: HTML, Utf8, HTTP and redirects

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

Agenda

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

Agenda

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

details

1. Need to finalize the groups right away.
2. Questions about homework, group photos?

Agenda

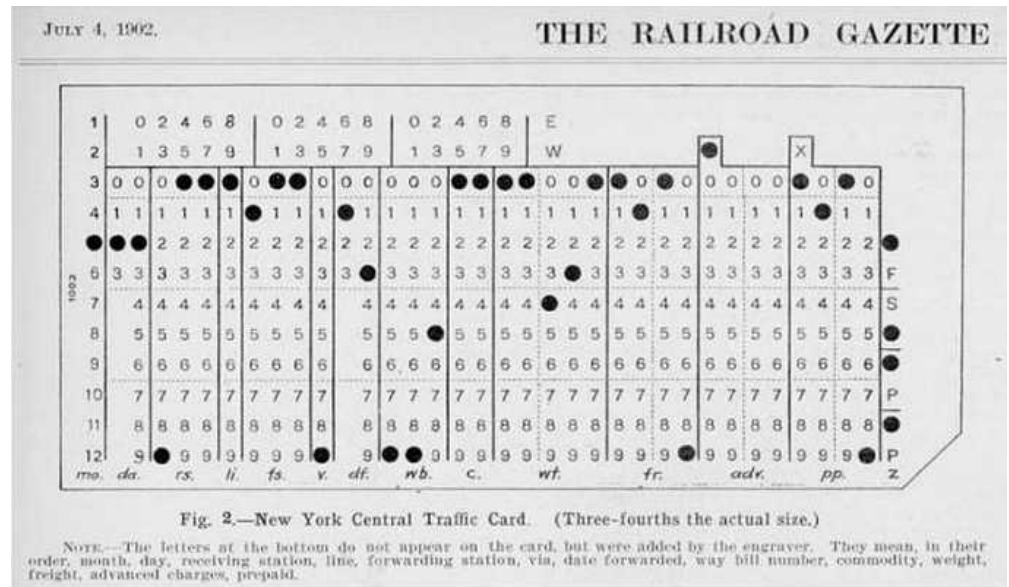
1. Course details.
2. Unicode and Utf8.
3. HTML.
4. HTTP.
5. Redirects.

in the beginning

Punch cards.

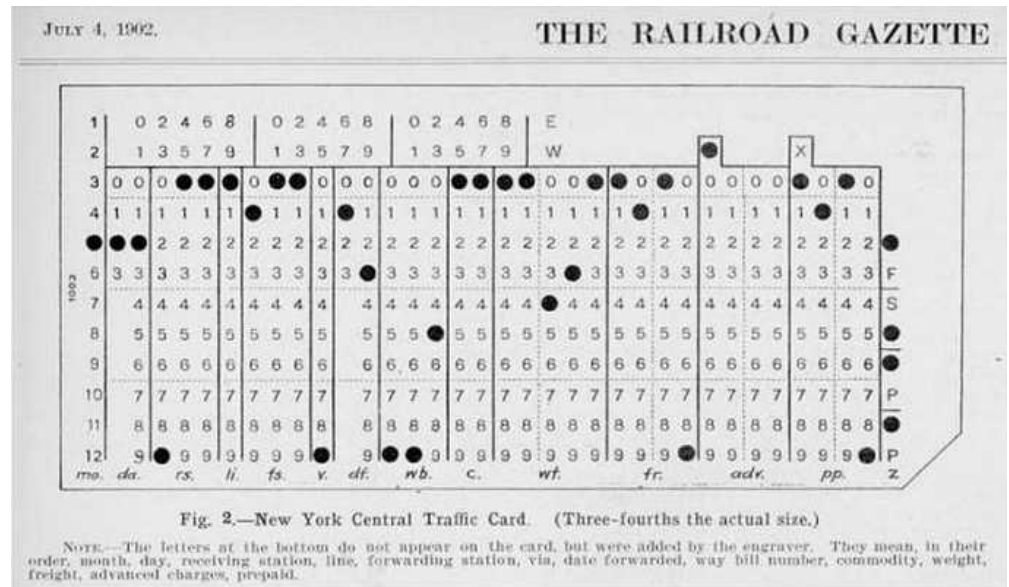
Invented by Herman Hollerith in the 1890s for tabulating census data.

Introduced a columnar format and a 6-bit character code.



in the beginning

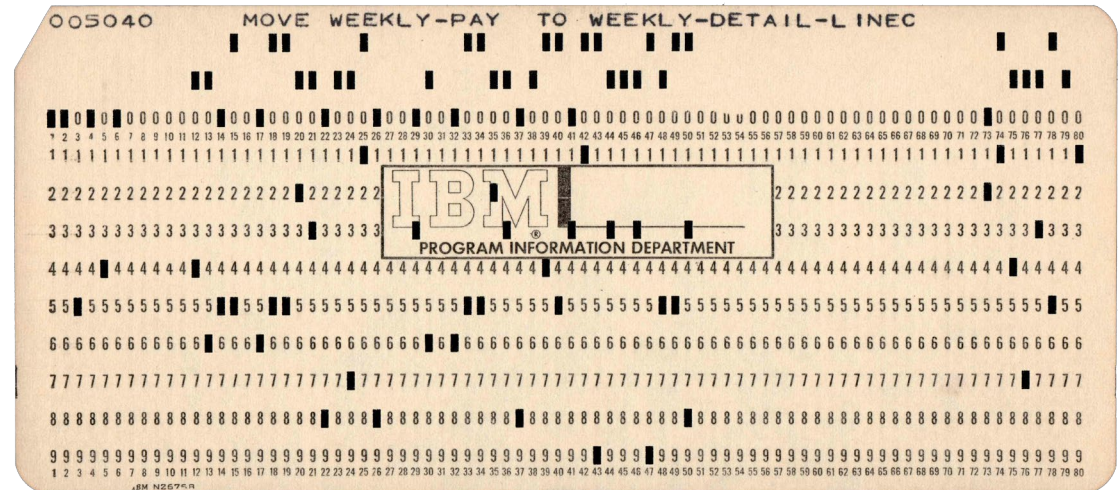
IBM was formed in 1911
in a merger of 3
companies, including
Hollerith's Tabulating
Machine Company.



in the beginning

The IBM card became ubiquitous in the 1960s with its 80-column format.

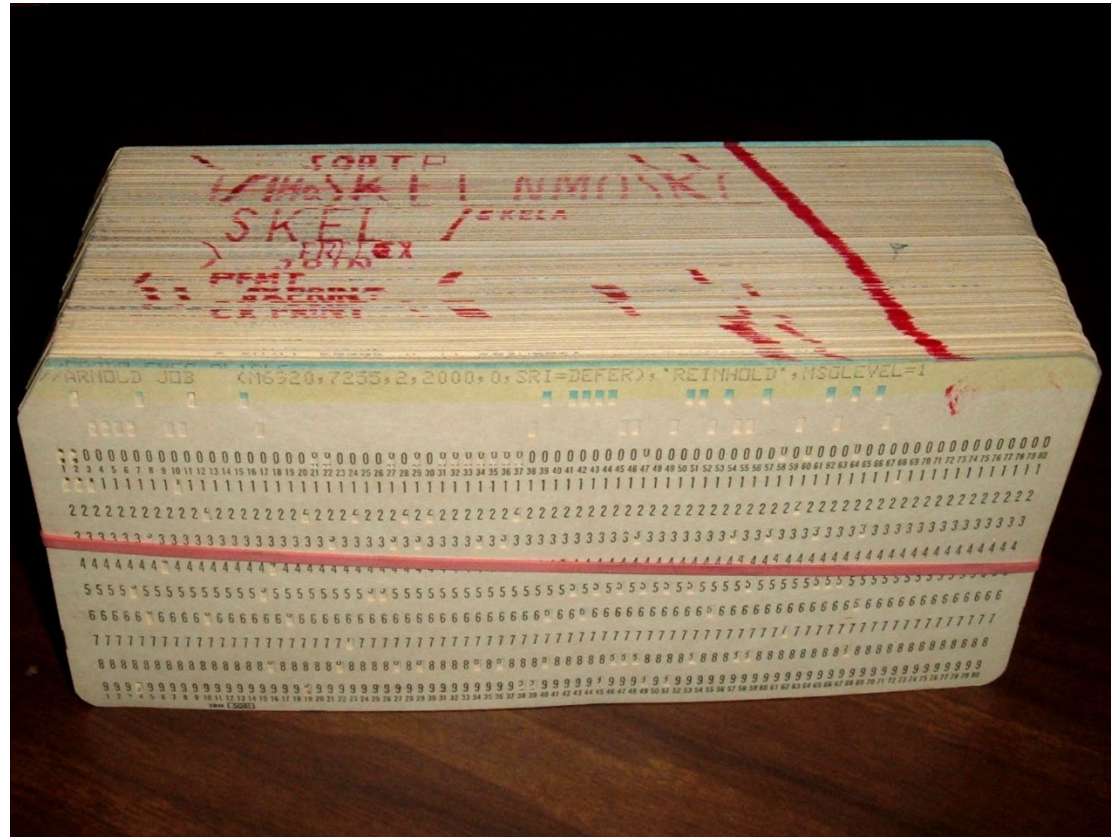
The character encoding was based on the earlier 6-bit encoding.



in the beginning

People walked around
with decks of these cards
holding their programs.

Imagine if you spilled one.



in the beginning

You used a keypunch to make them.

This is an IBM 029, released in 1964 as part of the System/360 rollout.

Businesses usually typed every card twice to verify they were correctly transcribed.



in the beginning

To sort data, you used a card sorter.

This is an IBM 082, introduced in 1949 but still in use into the 1960s.

Like all card sorters, it sorted one column at a time.



how do you represent text

In 1963, ASCII (American Standard Code for Information Interchange) was finalized as a 7-bit standard.

\$00	NUL	\$20	SP	\$40	@	\$60	`
\$01	SOH	\$21	!	\$41	A	\$61	a
\$02	STX	\$22	"	\$42	B	\$62	b
\$03	ETX	\$23	#	\$43	C	\$63	c
\$04	EOT	\$24	\$	\$44	D	\$64	d
\$05	ENQ	\$25	%	\$45	E	\$65	e
\$06	ACK	\$26	&	\$46	F	\$66	f
\$07	BEL	\$27	'	\$47	G	\$67	g
\$08	BS	\$28	(	\$48	H	\$68	h
\$09	HT	\$29	)	\$49	I	\$69	i
\$0A	LF	\$2A	*	\$4A	J	\$6A	j
\$0B	VT	\$2B	+	\$4B	K	\$6B	k
\$0C	FF	\$2C	,	\$4C	L	\$6C	l
\$0D	CR	\$2D	-	\$4D	M	\$6D	m
\$0E	SO	\$2E	.	\$4E	N	\$6E	n
\$0F	SI	\$2F	/	\$4F	O	\$6F	o
\$10	DLE	\$30	0	\$50	P	\$70	p
\$11	DC1	\$31	1	\$51	Q	\$71	q
\$12	DC2	\$32	2	\$52	R	\$72	r
\$13	DC3	\$33	3	\$53	S	\$73	s
\$14	DC4	\$34	4	\$54	T	\$74	t
\$15	NAK	\$35	5	\$55	U	\$75	u
\$16	SYN	\$36	6	\$56	V	\$76	v
\$17	ETB	\$37	7	\$57	W	\$77	w
\$18	CAN	\$38	8	\$58	X	\$78	x
\$19	EM	\$39	9	\$59	Y	\$79	y
\$1A	SUB	\$3A	:	\$5A	Z	\$7A	z
\$1B	ESC	\$3B	;	\$5B	[	\$7B	{
\$1C	FS	\$3C	<	\$5C	\	\$7C	
\$1D	GS	\$3D	=	\$5D	]	\$7D	}
\$1E	RS	\$3E	>	\$5E	^	\$7E	~
\$1F	US	\$3F	?	\$5F	_	\$7F	DEL

how do you represent text

As part System/360, IBM developed and introduced their own 8-bit EBCDIC (Extended Binary Coded Decimal Interchange Code) in 1964 derived from their punch card machines.

Notice the breaks in the character codes between I and J and R and S.

ASCII won.



The diagram illustrates the EBCDIC character set layout. It features a grid where the columns are labeled by the 1st hex digit (0-F) and the rows by the 2nd hex digit (0-F). The characters are organized into several distinct groups based on their hex values:

- 0-9 (0x00-0x09):** Control and punctuation characters (NUL, DLE, DS, SP, &, -, etc.).
- A-F (0x0A-0x0F):** Digits 0-9.
- 1-9 (0x10-0x19):** Lowercase letters a-i.
- A-J (0x20-0x29):** Uppercase letters A-I.
- 2-9 (0x30-0x39):** Lowercase letters j-s.
- A-I (0x40-0x49):** Uppercase letters J-S.
- 3-9 (0x50-0x59):** Lowercase letters t-z.
- A-I (0x60-0x69):** Uppercase letters T-Z.
- 4-9 (0x70-0x79):** Lowercase letters [\] ^ _ ` { | } ~ .
- A-I (0x80-0x89):** Uppercase letters [\] ^ _ ` { | } ~ .
- 5-9 (0x90-0x99):** Lowercase letters _ ` { | } ~ .
- A-I (0xA0-0xA9):** Uppercase letters _ ` { | } ~ .
- 6-9 (0xB0-0xB9):** Lowercase letters ~ .
- A-I (0xC0-0xC9):** Uppercase letters ~ .
- 7-9 (0xD0-0xD9):** Lowercase letters .
- A-I (0xE0-0xEF):** Uppercase letters .
- 8-9 (0xF0-0xF9):** Lowercase letters .
- A-I (0x100-0x10F):** Uppercase letters .

	1st hex digit															
2nd hex digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-									0
1	SOH	DC1	SOS			/			a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z		I	R	Z	9
A	SMM	CC	SM		C CENT	!	:									
B	VT	CU1	CU2	CU3		\$	.	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	:	>	=								
F	SI	IUS	BEL	SUB		-	?	*								

how do you represent text

But 7 bits is not enough to represent all the characters in languages other than English.

\$00	NUL	\$20	SP	\$40	@	\$60	'
\$01	SOH	\$21	!	\$41	A	\$61	a
\$02	STX	\$22	"	\$42	B	\$62	b
\$03	ETX	\$23	#	\$43	C	\$63	c
\$04	EOT	\$24	\$	\$44	D	\$64	d
\$05	ENQ	\$25	%	\$45	E	\$65	e
\$06	ACK	\$26	&	\$46	F	\$66	f
\$07	BEL	\$27	*	\$47	G	\$67	g
\$08	BS	\$28	(	\$48	H	\$68	h
\$09	HT	\$29	)	\$49	I	\$69	i
\$0A	LF	\$2A	*	\$4A	J	\$6A	j
\$0B	VT	\$2B	+	\$4B	K	\$6B	k
\$0C	FF	\$2C	,	\$4C	L	\$6C	l
\$0D	CR	\$2D	-	\$4D	M	\$6D	m
\$0E	SO	\$2E	.	\$4E	N	\$6E	n
\$0F	SI	\$2F	/	\$4F	O	\$6F	o
\$10	DLE	\$30	0	\$50	P	\$70	p
\$11	DC1	\$31	1	\$51	Q	\$71	q
\$12	DC2	\$32	2	\$52	R	\$72	r
\$13	DC3	\$33	3	\$53	S	\$73	s
\$14	DC4	\$34	4	\$54	T	\$74	t
\$15	NAK	\$35	5	\$55	U	\$75	u
\$16	SYN	\$36	6	\$56	V	\$76	v
\$17	ETB	\$37	7	\$57	W	\$77	w
\$18	CAN	\$38	8	\$58	X	\$78	x
\$19	EM	\$39	9	\$59	Y	\$79	y
\$1A	SUB	\$3A	:	\$5A	Z	\$7A	z
\$1B	ESC	\$3B	;	\$5B	[	\$7B	{
\$1C	FS	\$3C	<	\$5C	\	\$7C	
\$1D	GS	\$3D	=	\$5D	]	\$7D	}
\$1E	RS	\$3E	>	\$5E	^	\$7E	~
\$1F	US	\$3F	?	\$5F	_	\$7F	DEL

how do you represent text

The solution was to introduce the notion of code pages, 8-bit codes where the high half, 0x80 to 0xff depended on what country you were in.

\$00	NUL	\$20	SP	\$40	@	\$60	'
\$01	SOH	\$21	!	\$41	A	\$61	a
\$02	STX	\$22	"	\$42	B	\$62	b
\$03	ETX	\$23	#	\$43	C	\$63	c
\$04	EOT	\$24	\$	\$44	D	\$64	d
\$05	ENQ	\$25	%	\$45	E	\$65	e
\$06	ACK	\$26	&	\$46	F	\$66	f
\$07	BEL	\$27	*	\$47	G	\$67	g
\$08	BS	\$28	(	\$48	H	\$68	h
\$09	HT	\$29	)	\$49	I	\$69	i
\$0A	LF	\$2A	*	\$4A	J	\$6A	j
\$0B	VT	\$2B	+	\$4B	K	\$6B	k
\$0C	FF	\$2C	,	\$4C	L	\$6C	l
\$0D	CR	\$2D	-	\$4D	M	\$6D	m
\$0E	SO	\$2E	.	\$4E	N	\$6E	n
\$0F	SI	\$2F	/	\$4F	O	\$6F	o
\$10	DLE	\$30	0	\$50	P	\$70	p
\$11	DC1	\$31	1	\$51	Q	\$71	q
\$12	DC2	\$32	2	\$52	R	\$72	r
\$13	DC3	\$33	3	\$53	S	\$73	s
\$14	DC4	\$34	4	\$54	T	\$74	t
\$15	NAK	\$35	5	\$55	U	\$75	u
\$16	SYN	\$36	6	\$56	V	\$76	v
\$17	ETB	\$37	7	\$57	W	\$77	w
\$18	CAN	\$38	8	\$58	X	\$78	x
\$19	EM	\$39	9	\$59	Y	\$79	y
\$1A	SUB	\$3A	:	\$5A	Z	\$7A	z
\$1B	ESC	\$3B	;	\$5B	[	\$7B	{
\$1C	FS	\$3C	<	\$5C	\	\$7C	
\$1D	GS	\$3D	=	\$5D	]	\$7D	}
\$1E	RS	\$3E	>	\$5E	^	\$7E	~
\$1F	US	\$3F	?	\$5F	_	\$7F	DEL

W Code page - Wikipedia x +

← → ↻ https://en.wikipedia.org/wiki/Code_page#The_code_page_numbering_system 🔍 ☆ ABP H ⓘ ⚙️ 👤 ⋮

- [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		U+201A , 130	U+0192 f 131	U+201E ,, 132	U+2026 ... 133	U+2020 † 134	U+2021 ‡ 135	U+02C6 ^ 136	U+2030 ‰ 137	U+0160 Š 138	U+2039 < 139	U+0152 Œ 140		U+017D Ž 142	
9_		U+2018 , 144	U+2019 , 145	U+201C “ 146	U+201D ” 147	U+2022 • 148	U+2013 — 149	U+2014 — 150	U+02DC ~ 151	U+2122 ™ 152	U+0161 š 153	U+203A > 154	U+0153 œ 155		U+017E ž 156	U+0178 ÿ 157
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/>

code pages

But this meant that it was difficult to share files between users in different countries.

They didn't even have those foreign characters!

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
8_	U+20AC €	128	U+201A ‚	U+0192 ƒ	U+201E „	U+2026 …	U+2020 †	U+2021 ‡	U+02C6 ^	U+2030 ‰	U+0160 Š	U+2039 ‹	U+0152 Œ	141	U+017D Ž	142
9_	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A_	U+00A0 NBSP	U+00A1 ¡	U+00A2 ¢	U+00A3 £	U+00A4 ¤	U+00A5 ¥	U+00A6 ¦	U+00A7 §	U+00A8 ¨	U+00A9 ©	U+00AA ª	U+00AB «	U+00AC ¬	U+00AD ®	U+00AE ¯	U+00AF `
B_	U+00B0 °	U+00B1 ±	U+00B2 ²	U+00B3 ³	U+00B4 ´	U+00B5 µ	U+00B6 ¶	U+00B7 ·	U+00B8 ¸	U+00B9 ¹	U+00BA º	U+00BB »	U+00BC ¼	U+00BD ½	U+00BE ¾	U+00BF ¿
C_	U+00C0 À	U+00C1 Á	U+00C2 Â	U+00C3 Ã	U+00C4 Ä	U+00C5 Å	U+00C6 Æ	U+00C7 Ç	U+00C8 È	U+00C9 É	U+00CA Ê	U+00CB Ë	U+00CC Ì	U+00CD Í	U+00CE Î	U+00CF Ï
D_	U+00D0 Ð	U+00D1 Ñ	U+00D2 Ò	U+00D3 Ó	U+00D4 Ô	U+00D5 Õ	U+00D6 Ö	U+00D7 ×	U+00D8 Ø	U+00D9 Ù	U+00DA Ú	U+00DB Û	U+00DC Ü	U+00DD Ý	U+00DE Þ	U+00DF ß
E_	U+00E0 à	U+00E1 á	U+00E2 â	U+00E3 ã	U+00E4 ä	U+00E5 å	U+00E6 æ	U+00E7 ç	U+00E8 è	U+00E9 é	U+00EA ê	U+00EB ë	U+00EC ì	U+00ED í	U+00EE î	U+00EF ï
F_	U+00F0 ð	U+00F1 ñ	U+00F2 ò	U+00F3 ó	U+00F4 ô	U+00F5 õ	U+00F6 ö	U+00F7 ÷	U+00F8 ø	U+00F9 ù	U+00FA ú	U+00FB û	U+00FC ü	U+00FD ý	U+00FE þ	U+00FF ÿ

Unicode

In the late 1980s, Unicode (Universal Coded Character Set) was invented.

Used 16-bit characters = 65,536 code points, thought at the time to be more than anyone could ever need.

Characters were still always the same size.

Formalized as UTF-16 encoding.



UTF-16

Microsoft adopted UTF-16 for their core Windows API as their “wide characters.”

macOS, IBM I, Java, JavaScript, .NET, Python, and even SMS text messaging also adopted this encoding.



UTF-16

Required a Byte Order Mark (BOM) at the start of a file to indicate if it was “big-endian” (high byte first) or “little-endian” (low byte first).

ff fe Little-endian UTF-16

fe ff Big-endian UTF-16

If you saw the BOM you expected, it meant you were reading it correctly.



UTF-16

Did not get wider use because it was incompatible with ASCII, and seen wasteful, requiring two bytes for every character at a time when most PCs only had 640K.



UTF-16

A bigger problem was what to do if you needed more than 64K code points.

Unicode now defines over a 1M code points, including almost 4,000 emojis.



UTF-16

The solution was to introduce “surrogates”.

Values greater than 0xffff were broken into two 16-bit characters holding 10 bits each.

Notorious for causing bugs in programs that didn't anticipate that characters could be variable length.



UTF-8

Invented in 1992 by Ken Thompson and Rob Pike, sketched out on a dinner placemat over a meal in a NJ diner.

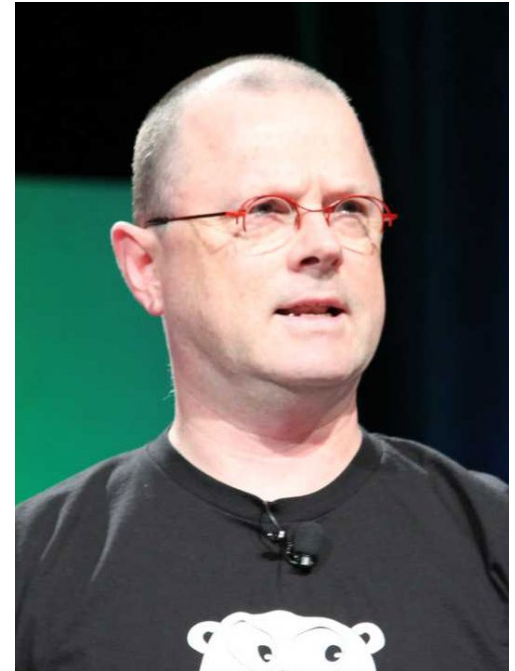


Image sources: https://en.wikipedia.org/wiki/Ken_Thompson
https://en.wikipedia.org/wiki/Rob_Pike

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 byte gives you an additional 5 bits. Some sequences are invalid.
5. Capable of encoding 31 bits.
6. BOMs are allowed but not needed and discouraged.

UTF-8 sequences

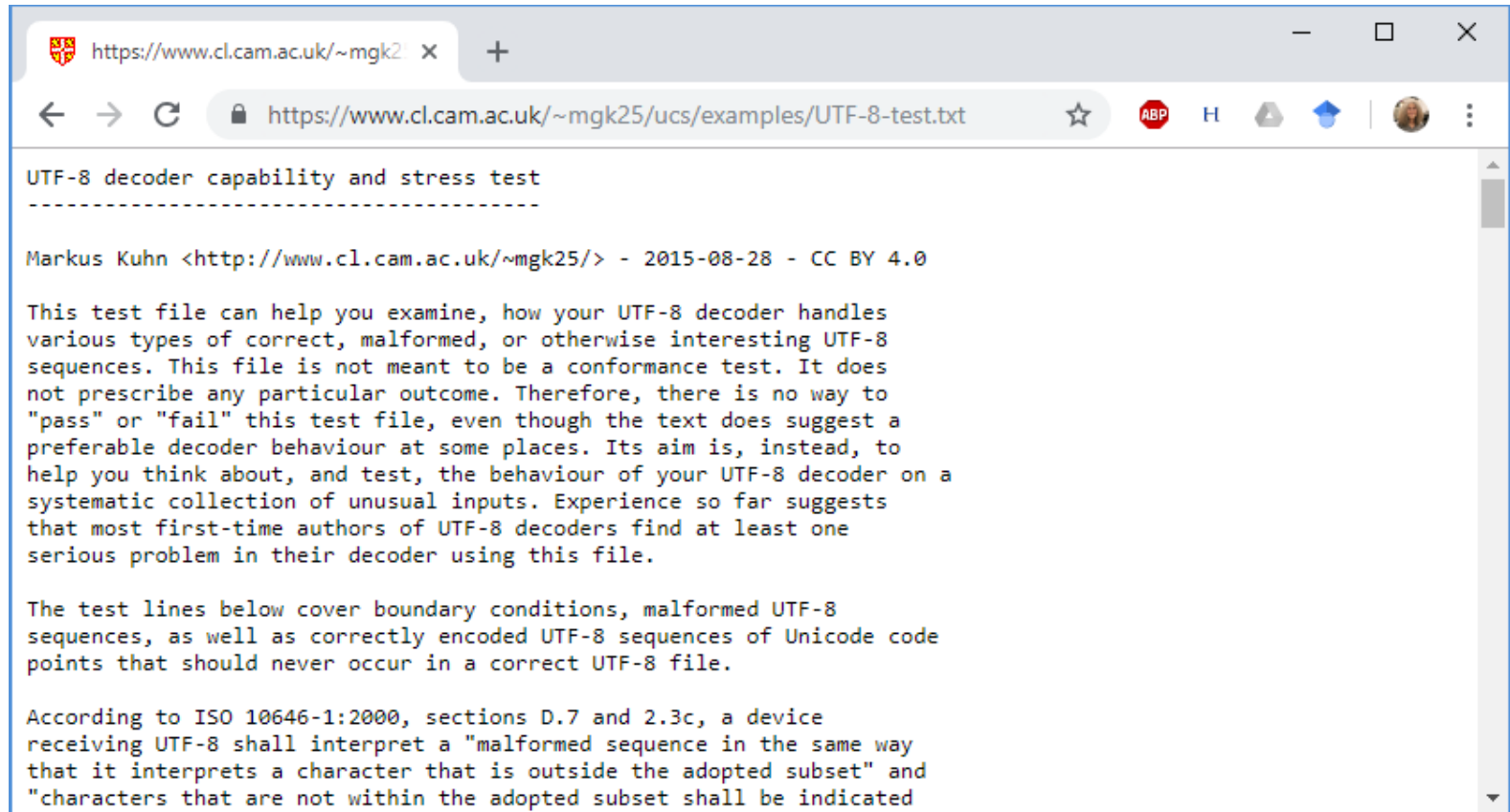
Valid UTF-8 sequences

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 test case

A screenshot of a web browser window. The address bar shows the URL <https://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt>. The page content is as follows:

UTF-8 decoder capability and stress test

Markus Kuhn <<http://www.cl.cam.ac.uk/~mgk25/>> - 2015-08-28 - CC BY 4.0

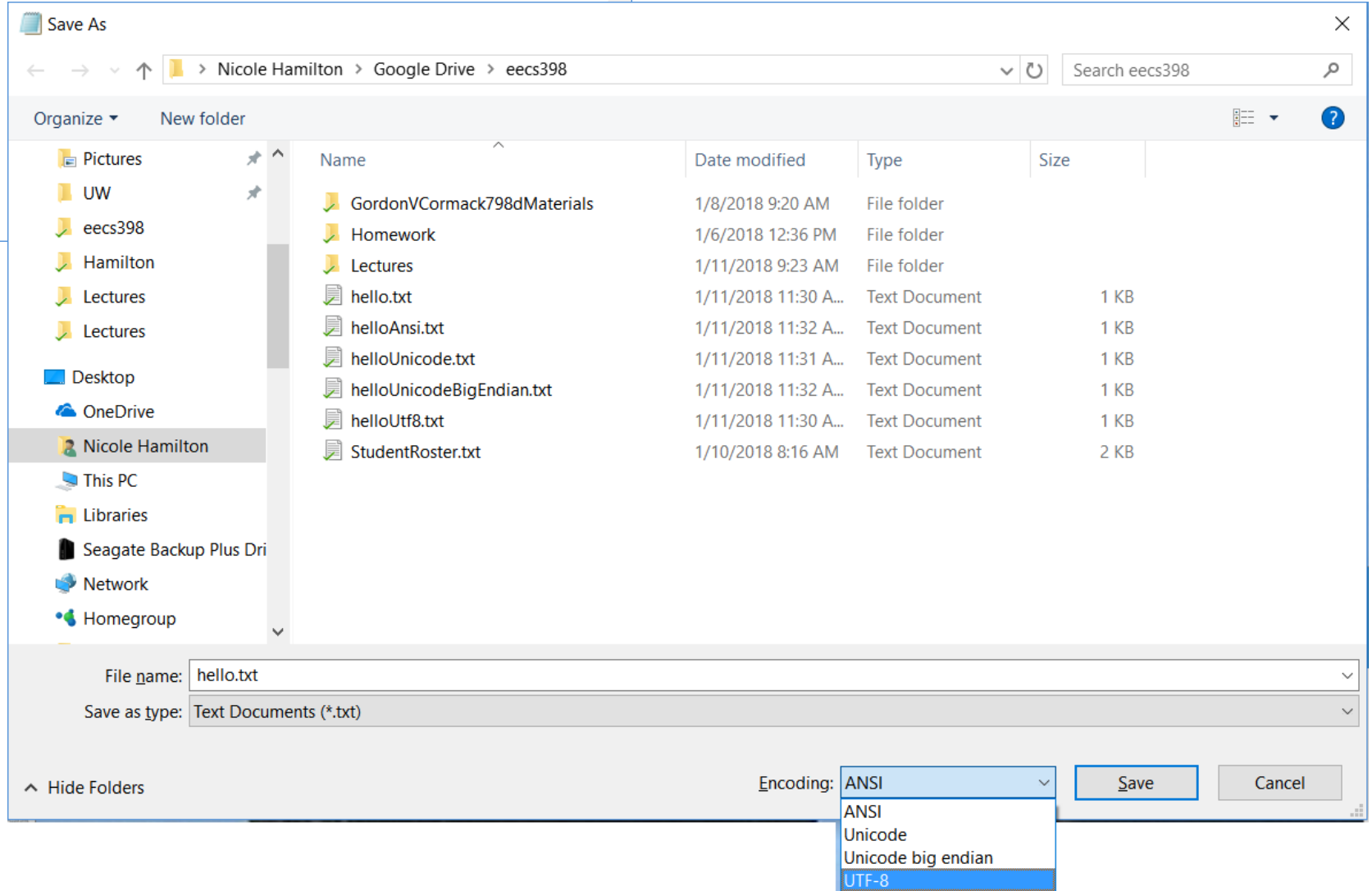
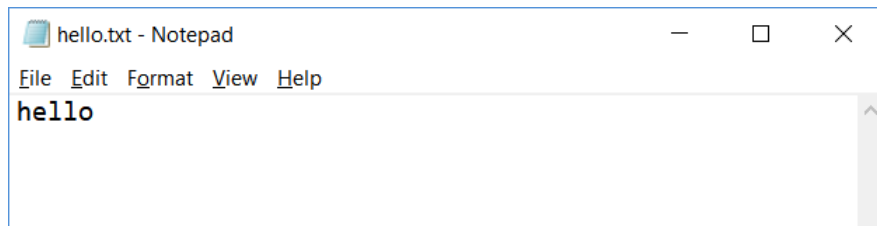
This test file can help you examine, how your UTF-8 decoder handles various types of correct, malformed, or otherwise interesting UTF-8 sequences. This file is not meant to be a conformance test. It does not prescribe any particular outcome. Therefore, there is no way to "pass" or "fail" this test file, even though the text does suggest a preferable decoder behaviour at some places. Its aim is, instead, to help you think about, and test, the behaviour of your UTF-8 decoder on a systematic collection of unusual inputs. Experience so far suggests that most first-time authors of UTF-8 decoders find at least one serious problem in their decoder using this file.

The test lines below cover boundary conditions, malformed UTF-8 sequences, as well as correctly encoded UTF-8 sequences of Unicode code points that should never occur in a correct UTF-8 file.

According to ISO 10646-1:2000, sections D.7 and 2.3c, a device receiving UTF-8 shall interpret a "malformed sequence in the same way that it interprets a character that is outside the adopted subset" and "characters that are not within the adopted subset shall be indicated

UTF-8 decoder capability and stress test

<http://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt>



```

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%

```

Share of web pages with different encodings

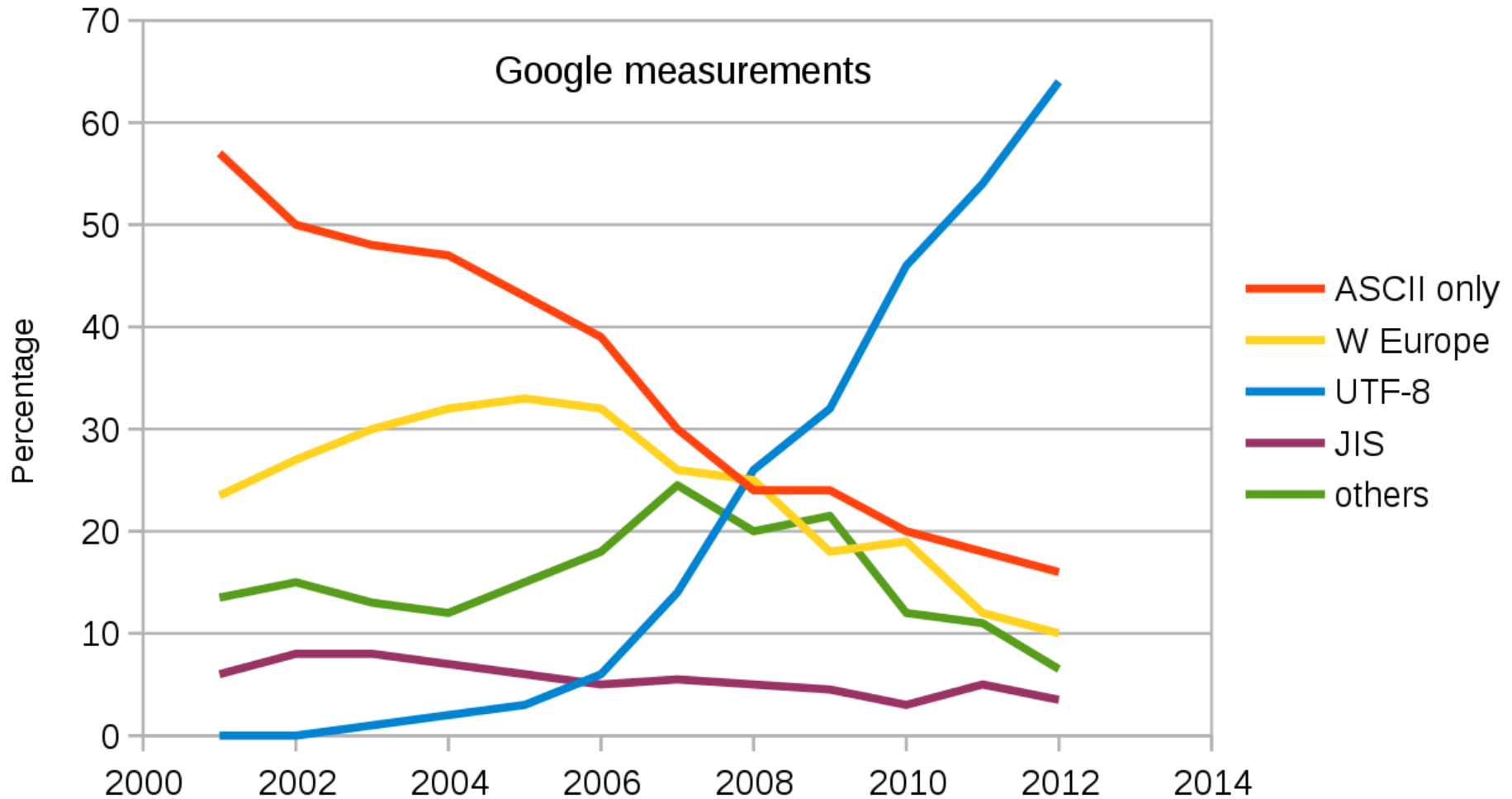
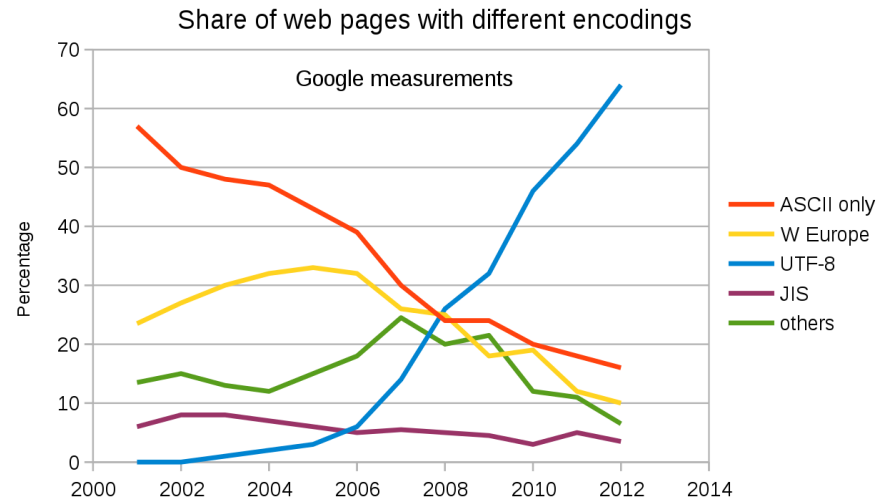


Image source: <https://en.wikipedia.org/wiki/UTF-8>

today

UTF-8 is overwhelmingly dominant on the web, used by nearly 99% of websites as of early 2026, making it the standard for the internet, default for HTML5, and crucial for displaying diverse languages, emojis, and symbols globally.



your homework

```
#include <stddef>
#include <stdint>
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 );
```

Agenda

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

the basic parts of a search engine

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

initial focus

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.

initial focus

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

Depend on understanding
how a browser talks to a
webpage.

initial focus

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.

initial focus

Need to serialize and
deserialize data.

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

initial focus

Need to serialize and deserialize data.

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

Examples:

.zip: Lossless, binary, stateless.

.jpg: Lossy, binary, stateless.

.html: Lossless, text, possibly stateful, e.g., login, shopping carts.

initial focus

Need to serialize and deserialize data.

1. Lossy or lossless.
2. Human-readable or binary.
3. Stateless or stateful.
4. Question of where to store state.

Examples:

.zip: Lossless, binary, stateless.

.jpg: Lossy, binary, stateless.

.html: Lossless, text, possibly stateful, e.g., login, shopping carts.

the webpage concept

Invented by Tim Berners-Lee as a way to 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.

a typical HTML page

```
<!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>
```

a typical HTML page

```
<!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 !DOCTYPE and html tags describe the standards the page ostensibly conforms to. (Generally useless.)

a typical HTML page

```
<!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 meta tag describes the content, character set, and other attributes. ([Google list.](#))

a typical HTML page

```
<!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 page title is highly relevant.

a typical HTML page

```
<!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>
```

Scripts are literally gibberish unless you execute them. Google and other engines probably do run scripts on some sites.

a typical HTML page

```
<!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.

a typical HTML page

```
<!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.)

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.

a typical HTML page

```
<!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>
```

Description may be useful since search engines often display it in search results.

a typical HTML page

```
<!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>
```

Not displayed in a browser window. Easily
spammed. Probably not useful.

a typical HTML page

```
<!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 body text of the page.

<body> and <div>

```
<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.

<div> ... </div> sections can be used to create separate panels on the page, applying different CSS styles.

<a>

```
<div id="content">
```

```
  <a href="clickURL1">  
      
  </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.

, <h1>, and <p>

Images are added with tags.

```
<div id="content">
```

```
  
```

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

```
  <p>
  :
</p>
```

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

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

```
  <p>
  :
</p>
```

```
  <p>
  :
</p>
```

Paragraphs are marked with <p> tags. The closing </p> are often missing.

```
</div>
```

&-escapes

```
<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.

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?*

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%
```