C-STRING FUNCTIONS

Library or API Functions

#include <cstring>

Delroy A. Brinkerhoff

C-STRINGS ARE PRIMITIVE

- C-Strings are based on arrays and pointers: they are a primitive data type
  - Header file only needed when using the C-string functions
  - `#include <cstring>`
  - `#include <string.h>`

ASCII ENCODING

- Individual characters are encoded as integer values
  - '0' – '9' as 48 – 57
  - 'A' – 'Z' as 65 – 90
  - 'a' – 'z' as 97 – 122
  - punctuation characters are mixed in
  - control characters are 0 – 31
  - 128 – 255 are the extended ASCII

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

https://www.asciitable.com/

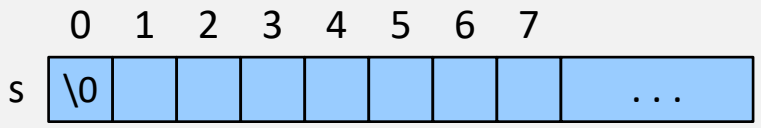
Source: www.LookupTables.com

nullptr

- First introduced in chapter 4, nullptr indicates when a pointer variable isn't pointing to anything (C++ also allows NULL and the numeral 0)
  - `char* p;`
  - `p = nullptr;`
  - `if (p == nullptr) . . .`
  - `if (p != nullptr) . . .`
- Function arguments and return values can be nullptr

EMPTY C-STRINGS

- An empty C-string is not the same as a null C-string
- Null C-strings do not have allocated memory
- Empty C-strings have memory but do not have data
- Must have the null-termination character
  - `char s[100] = "";`
  - `char s[100];  s[0] = '\0';`



PSEUDO DATA TYPES

- Pseudo data types are
  - aliases created as symbolic constants
  - converted to real types at compile time
  - used to improve code portability
  - typically named end with a "_t" at the end
- `size_t`
  - an integer suitable for hold a data size
- `errno_t`
  - an integer that encodes an error number

C-STRING FUNCTION ARGUMENTS

- The C-string arguments for the C-string functions(`<cstring>`) are shown as char*
- `char* strcpy(char* destination, const char* source);`
- Pointers must point to allocated memory
  - character array: `char s1[100];`
  - dynamic: `char* s2 = new char[100];`
  - `strcpy(s2, s1);`

CHARACTER POINTER RETURN VALUES

- Many C-string functions return a character pointer
  - `char* strcpy(char* destination, const char* source);`
  - The pointer is often one of the arguments; `return destination;`
  - This a convenience that allows embedding the call in a larger context:
  - `cout << strcpy(s2, s1) endl;`

MICROSOFT VISUAL STUDIO

- Microsoft replaces many of the standard C-string functions with secure versions whose names end with "_s"
  - One additional argument
  - Integer return type
  - Suppress with: `#define _CRT_SECURE_NO_WARNINGS`
  -
```
errno_t strcpy_s(char *strDestination,
    size_t numberOfElements,
    const char *strSource);
```