K&R Chapter 1
A Tutorial Introduction

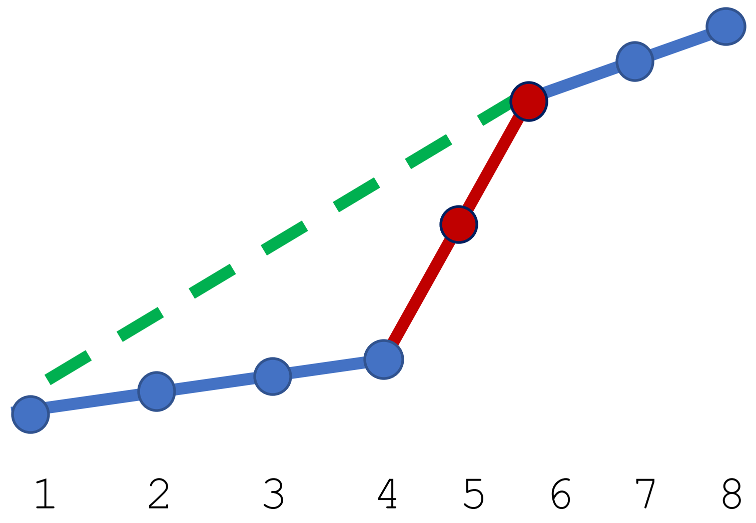
Dr. Charles R. Severance
www.cc4e.com
code.cc4e.com (sample code)
online.dr-chuck.com

Learning Path: online.dr-chuck.com

- Internet History, Technology, and Security – ihts.pr4e.com
- Python – www.py4e.com
- Django (Python, HTML, CSS, SQL, JavaScript) – www.dj4e.com
- Web Applications (PHP, HTML, CSS, SQL, JavaScript) – www.wa4e.com
- PostgreSQL (SQL) – www.pg4e.com
- C Programming – www.cc4e.com ← We are here ☺
- Computer Architecture
- Java Enterprise Application Development

Outline of the book

- Chapters 1-4 – Mostly syntax
  - Just another programming language
  - Arrays, strings and the weird fact that strings are character arrays
- Chapter 5 – Pointers and Arrays
- Chapter 6 – Structures
- Chapter 7 – 8 – Detailed C Features
  - The detail we skipped



Chapter 1

- Sections 1.1 – 1.5 – “Just another programming language”
- Section 1.6 – Arrays
  - Static allocation – cannot be resized until Chapter 5 ☺
- Section 1.7 – 1.8 – Functions and Parameters
  - Call by value is simple – call by reference is in Chapter 5 ☺
- Section 1.9 – Character Arrays
  - This is important because there ***<u>is no string object</u>*** in C
- Section 1.10 – Variable scoping between functions

Character Arrays

- We must carefully understand the “size” of the character array and not exceed it – in C nothing is “auto-extended”

```
x = ""
for i in range(1000) :
    x += '*'
print(x)

$ python3 cc_01_01.py
*********…***********
```

```
#include <stdio.h>
int main() {
    char x[10];
    int i;

    for(i=0; i<1000; i++ ) x[i] = '*';
    printf("%s\n",x);
}

$ a.out
Segmentation fault: 11
```

kr_01_01.c

String / Character Constants

- Languages like PHP, Python, and JavaScript treat single and double quotes nearly the same. Both create *string* constants.
- In C single quotes are a character and double quotes are a character array (neither are a string)
- In C, a character is a byte – a short (typically 8-bit) integer

```
#include <stdio.h>
int main() {
    char x[3] = "Hi";
    char y[3] = { 'H', 'i' };
    printf("x %s\n", x);
    printf("y %s\n", y);
    printf("%s\n", "Hi");
    printf("%c%c\n", 'H', 'i');
}
```

```
$ a.out
x Hi
y Hi
Hi
Hi
```

Character Sets

- The C **char** type is just a number – character representations depend on the character set.
- Modern characters including 😊 are represented in multi-byte sequences using Unicode and UTF-8 – but in 1978 we used ASCII and other character sets.

```
print('A', ord('A'))
print('😊',
ord('😊’))

$ python3 kr_01_03.py
A 65
😊 128522
```

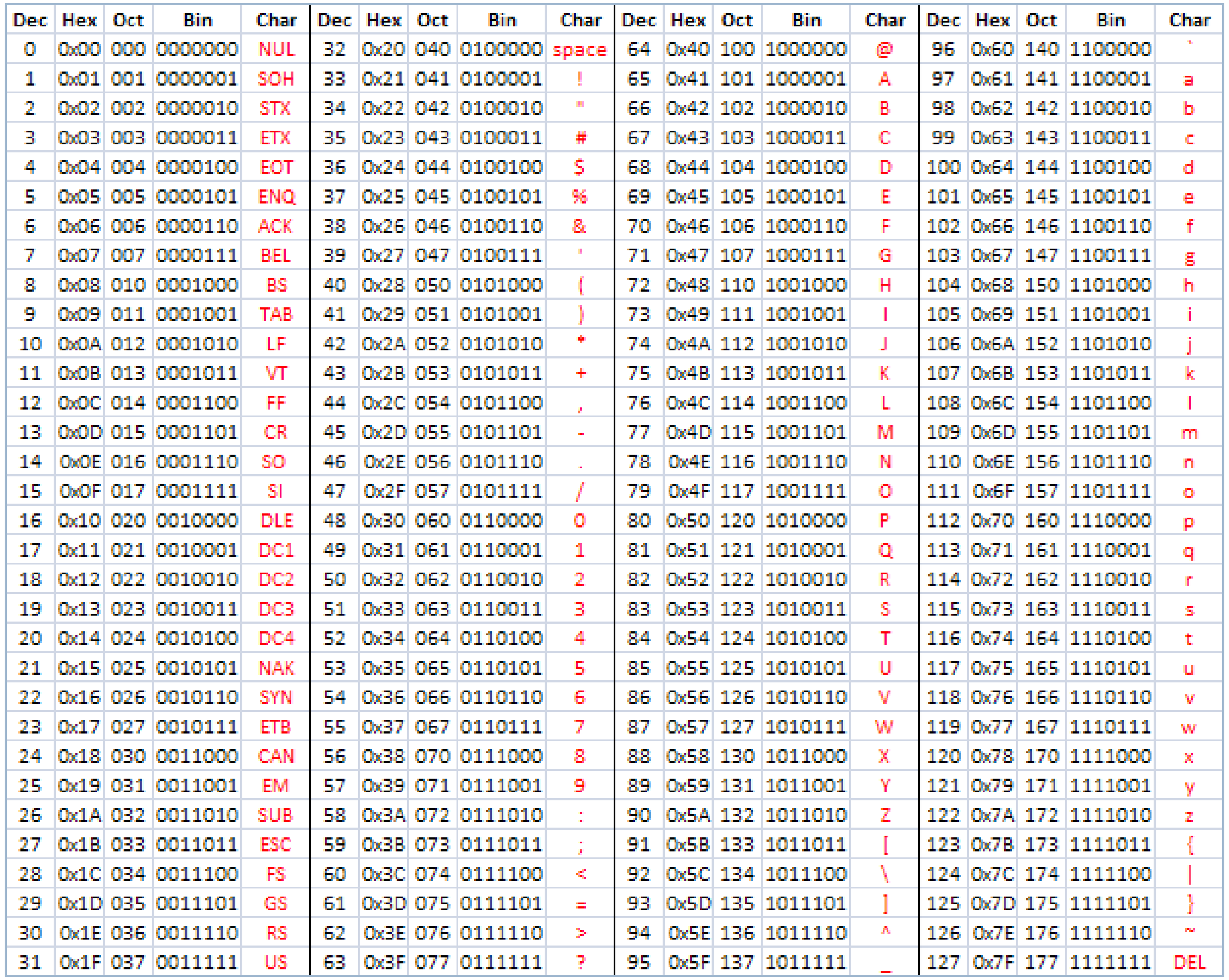
```
#include <stdio.h>
int main()
{
    printf("%c %d\n", 'A', 'A');
}

$ a.out
A 65
```

kr_01_03.c

ASCII

- American Standard Code for Information Interchange



Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

Wikipedia: ASCII

Image source: catonmat.net

Terminating a String

- The size of a “string” stored in a C array is not the length of the array
- C uses a special character ' \0 ' that *marks* the string end by convention
- Character arrays need to allocate an extra byte to store the line-end character

```c
#include <stdio.h>
int main() {
    char x[6];
    x[0] = 'H';
    x[1] = 'e';
    x[2] = 'l';
    x[3] = 'l';
    x[4] = 'o';
    x[5] = '\0';
    printf("%s\n",x);

    x[2] = 'L';
    printf("%s\n",x);

    x[3] = '\0';
    printf("%s\n",x);
}
```

```
$ a.out
Hello
HeLlo
HeL
```

kr_01_04.c

String Length

- In C string “length” must be computed in a loop that scans for a zero character
- There the **strlen()** function in **string.h** computes string length

```
x = 'Hello'
print(x, len(x))

$ python3 kr_01_05.py
Hello 5
```

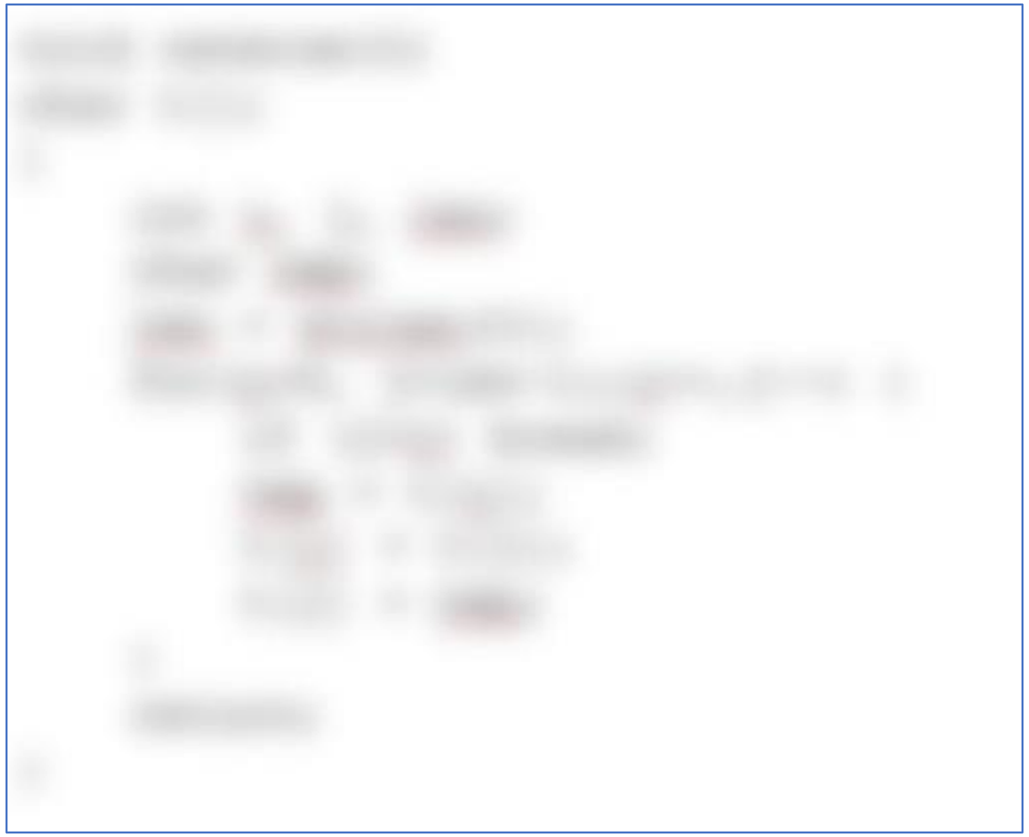
```
#include <stdio.h>
int main() {
    char x[] = "Hello";
    int py_len();
    printf("%s %d\n",x, py_len(x));
}
int py_len(self)
    char self[];
{
    int i;
    for(i=0; self[i]; i++);
    return i;
}

$ a.out
Hello 5
```

kr_01_05.c

Reverse a String in C

- A beloved / terrible coding interview question
- Exercise 1-17 in K&R
- Do *not* cheat, do not look for the answer (1000's are out there)
- Your struggle is very valuable for you
- The reversal must be done in-place
- Even length strings, odd length strings, empty strings, single character strings – think about them all



Summary

- Overview and approach for the book
- Representing ”strings” in C character arrays
- Actually doing your homework because it is good for you

Acknowledgements / Contributions