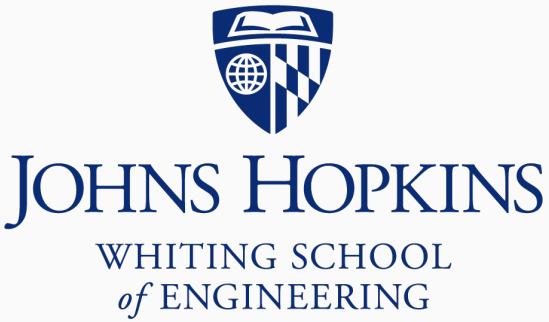
Characters & strings

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Characters & strings

`char` variable holds a single character

- `char digit = '4';`
- `char bang = '!';`
- These *must* be single quotes; double quotes are for strings only

Behind the scenes, `char` is much like `int`

- This is valid: `char digit = '4' - 1;`
- `digit` now contains the character `'3'`

`printf` format string for `char` is %c

ASCII

ASCII governs the mapping between typical characters and integers

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Character example

```c
#include <stdio.h>

// Convert decimal character into corresponding int
int main() {
    char char_0 = '0';
    int int_0 = char_0 - '0';
    printf("Character printed as character: %c\n", char_0);
    printf("Character printed as integer: %d\n", char_0);
    printf("Integer printed as integer: %d\n", int_0);
}
```

```
$ gcc convert_digit_0.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Character printed as character: 0
Character printed as integer: 48
Integer printed as integer: 0
```

Character example

```c
#include <stdio.h>

// Convert decimal character into corresponding int
int main() {
    char char_7 = '7';
    int int_7 = char_7 - '0';
    printf("Character printed as character: %c\n", char_7);
    printf("Character printed as integer: %d\n", char_7);
    printf("Integer printed as integer: %d\n", int_7);
}
```

```
$ gcc convert_digit_7.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Character printed as character: 7
Character printed as integer: 55
Integer printed as integer: 7
```

Strings

Variety of real-word data is represented as strings:

- Natural language: `"Hello world"`
- DNA: `"GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGC"`
- Stock movements: `"BAC -0.3 GOOGL +0.8 CIT -1.2"`
- Chess moves: `"1.e4 e5 2.Nf3 f6 3.Nxe5 fxe5"`

Strings

In C, a string is an array of characters with final character equal to the "null character" `'\0'`

To declare string:

```
char day[] = "monday";

char *day_ptr = "monday";

const char cday[] = "monday";
const char *cday_ptr = "monday";
// can't modify characters in cday or cday_ptr
```

A string in double quotes (e.g. "monday") is a *string literal*

Strings

These declarations show that strings have a “pointer nature” and an “array nature”:

```
char day[] = "monday";

char *day_ptr = "monday";
```

These are largely interchangeable; some differences covered later

Strings

A C string is an array of characters with the final character equal to the "null character" `'\0'`

Also called the the *null terminator*

```
// this definition:
char day1[] = "monday";

// is the same as this:
char day2[] = {'m', 'o', 'n', 'd', 'a', 'y', '\0'};
//                                terminator -> ^^^^
```

A character in single quotes (e.g. `'m'`) is a *character literal*

Strings

Access the characters in a string using square brackets, same as arrays:

```c
#include <stdio.h>

// Convert decimal character into corresponding int
int main() {
    char str[] = "hello";
    str[0] = 'H'; // modify first character
    // print whole string, then 3rd char, then 5th
    printf("%s %c %c\n", str, str[2], str[4]);
    return 0;
}
```

```
$ gcc string_indexing_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Hello l o
```

Strings

To print string with `printf` use %s format specifier

```c
#include <stdio.h>

// Convert decimal character into corresponding int
int main() {
    const char str[] = "World";
    printf("Hello, %s!\n", str);
    return 0;
}
```

```
$ gcc string_indexing_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Hello, World!
```

Strings

What's the mistake?

```c
#include <stdio.h>

int main() {
    const char str[] = "hello";
    char str_copy[5];
    for(int i = 0; i < 5; i++) {
        str_copy[i] = str[i];
    }
    printf("%s\n", str);
    printf("%s\n", str_copy);
    return 0;
}
```

Strings

```
gcc %PREV% -std=c99 -pedantic -Wall -Wextra
./a.out
```

```
hello
hello???
```

Failed to null-terminate the copy, so `printf` prints beyond the end

- That's memory we don't own and didn't initialize
- It's "undefined"; could be anything!
- In this case, it's "garbage" that prints out as question marks

Strings

```c
#include <stdio.h>

int main() {
    const char str[] = "hello";
    char str_copy[6]; // was [5]
    for(int i = 0; i < 6; i++) { // was i < 5
        str_copy[i] = str[i];
    }
    printf("%s\n", str);
    printf("%s\n", str_copy);
    return 0;
}
```

```
$ gcc string_copy_2.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
hello
hello
```

Strings

Long string can be declared across multiple lines:

```
// beginning of human mitochrondrial sequence
const char *dna = "GATCACAGGTCTATCACCCTATTAA"
                  "CCACTCACGGGAGCTCTCCATGCAT"
                  "TTGGTATTTTCGTCTGGGGGGTGTG"
                  "CACGCGATAGCATTGCGAGACGCTG"
                  "GAGCCGGAGCACCCTATGTCGCAGT"
                  "ATCTGTCTTTGATTCCTGCCTCATT"
                  "CTATTATTTATCGCACCTACGTTCA"
                  "ATATTACAGGCGAACATACCTACTA"
                  "AAGTGTGTTAATTAATTAATGCTTG"
                  "TAGGACATAATAATAACAATTGAAT";
```

Strings

`#include <string.h>` for helpful string functions

```
size_t strlen(const char *s);
```

Returns length of string `s` *not including* terminator

```
#include <stdio.h>   // for printf
#include <string.h>  // for strlen

int main() {
    const char str[] = "hello";
    printf("length(%s) = %lu\n", str, strlen(str));
    return 0;
}
```

```
$ gcc strlen_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
length(hello) = 5
```

Strings

More <string.h> functions

- strlen(s) returns length of string s
- strcmp(s1, s2) compares two strings alphabetically
  - negative: s1 alphabetically befores2‘
  - zero: s1 and s2 equal
  - positive: s2 alphabetically before s1
- strncpy(dst, src) copies characters from src into dst
- strncat(dst, src) copies src onto the end of dst
- strstr(lng, shrt) search for occurrence of shrt within lng

http://www.cplusplus.com/reference/cstring/

Strings

Some C string functions are considered unsafe and *should not* be used:

- `strcat` (instead we use `strncat`)
- `strcpy` (instead we use `strncpy`)

Common vulnerabilities guide for C programmers