C and Linux Programming
Eastern Washington University
Computer Science
March 30th – June 12th, 2020

Lecture 3

C String Processing and Files

Joe Dumoulin

Mar 30, 2020

Eastern Washington University

C String Processing and Functions

The Story So Far

Basic C syntax

- Variable Types
- Statements
- Functions

Next we will use these concepts to see how to work with characters and functions in C.

EASTERN WASHINGTON UNIVERSITY

Basic Character Representations - 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

Source: www.LookupTables.com

EASTERN WASHINGTON UNIVERSITY

What is ASCII?

ASCII stands for "American Standard Code for Information Interchange"

ASCII encodes a symbol into a number and a number to a symbol. The ASCII table is a codebook for changing numbers into characters.

The first 32 ASCII characters are called **Control Characters**. They don't get printed. Instead they cause actions to occur in the terminal.

the characters from 32-127 are characters used to print words, numbers, punctuation and white space for writing english.

ASCII is insufficient for most languages. Modern languages and libraries rely on **Unicode**.

EASTERN WASHINGTON UNIVERSITY

A Brief Interlude About Unicode

Modern applications must support more than english. For this we use **Unicode**.

Unicode uses a 32-bit **code point** to represent each possible character in any language (including made-up languages like elvish and klingon).

東京

Москва - столица России

بِسمِ اللّٰهِ الرَّحمٰنِ الرَّحِيمِ

We will not work with unicode, but we should recognize that for modern programming it is very important.

EASTERN WASHINGTON UNIVERSITY

Categories of ASCII codes

Categories of ASCII Codes that we Care About

Numbers	ASCII Code
48-57	Digit symbols ('0' - '9')
65-90	Upper-case letters ('A' - 'Z')
97-122	Lower-case letters ('a' - 'z')
33-47 58-64 133-140 123-126	Punctuation
10-13, 32	Whitespace

We can write code to recognize these characters.

EASTERN WASHINGTON UNIVERSITY

Finding Character Types in a String

The following function takes a character as a parameter and tests if the character is an upper-case letter. It returns 1 if the character is an upper-case letter and zero if it is anything else.

```
int isupperalphabetic(char c)
{
  if (c >= 65 && c <= 90) {
    return 1;
  }
  // else
  return 0;
}
```

EASTERN WASHINGTON UNIVERSITY

Finding Character Types in a String

We can make the function easier to understand by adding some constants for 1 and 0. This makes it possible for us to talk about the true or false nature of the function. We can also better understand the function by using the ASCII character symbols instead of numbers for comparison.

```
#define TRUE 1
#define FALSE 0

int isupperalphabetic(char c)
{
  if (c >= 'A' && c <= 'Z') {
    return TRUE;
  }
  // else
  return FALSE;
}
```

Predicate Functions

Functions and statements that evaluate to True or False are called **Predicates**. Predicates can be used to make decisions. Here is another predicate function that identifies a lower-case letter if it is found:

```
int isloweralphabetic(char c)
{
  if (c >= 'a' && c <= 'z') {
    return TRUE;
  }
  // else
  return FALSE;
}
```

Composing Functions

Functions can be combined to make more complex functions. Here's an example of a predicate function created from combining the two previous predicates.

```
int isalphabetic(char c)
{
  if (isupperalphabetic(c) || isloweralphabetic(c)) {
    return TRUE;
  }
  // else
  return FALSE;
}
```

What does this function do?

EASTERN WASHINGTON UNIVERSITY

Counting Characters in a String

Another type of function we can make is one that counts the number of characters in a string. We can use the fact that c strings are terminated with '\0'. We can find the length of a string by counting all the characters in the string until we find the terminating character.

```
int stringlength(char* s)
{
  int x = 0;
  while(s[x] != '\0') {
    x++;
  }
  return x;
}
```

EASTERN WASHINGTON UNIVERSITY

Writing a Program With Functions

We can use these functions to write a program that prints each letter in the string.

```
// include the functions defined above ...
int main(void)
{
  char testStr[] = "When shall we three meet again \
In thunder, lightning, or in rain?";
  printf("string = %s\n", testStr);
  int i;
  int len = stringlength(testStr);
  printf("%d\n", len);

  for (i = 0; i < len; i++){
    char c = testStr[i];
    if (isalphabetic(c)) {
      printf("LETTER: %c\n", c);
    }
  }
  return EXIT_SUCCESS;
}
```

Question: What is the output of this program?

EASTERN WASHINGTON UNIVERSITY

Another function: Lower-case a Character

Each upper case and lower case letter in the ASCII table is separated by a constant. We can calculate that constant by subtracting two of the codes for a letter like this:

'a' -'A'

The code for this is:

```
// if a character is upper case, then change it to lower case
char lower(char c)
{
  if (isupperalphabetic(c)) {
    return c + ('a'-'A');
  }
  // else
  return c;
}
```

Counting Letters in a String

We can use the functions we have developed so far to do many things. Let's do something a little more complcated. we will write a program that calculates the number of each letter that appears in a string.

The main function for this task is longer than we have seen so far.

We will need to:

- Define an array of 26 numbers (one for each letter) and initialize it
- Define a string to process
- For each character c in the string,
  - If c is an upper case letter, change it to lower case
  - If c is a letter, add one to that letter's entry in the array
  - Ignore all other characters
- print the contents of the array

EASTERN WASHINGTON UNIVERSITY

1. Define and initialize an array

```
int main(void)
{
  int char_counts[26];  // one aray slot for each letter

  int i;   // an iterator
  // initailize counts to zero.
  for (i = 0; i < 26; i++) {
    char_counts[i] = 0;
  }
....
```

2. Define the string to process

Define the string, print it, and then print its length.

```
  char testStr[] = "Accepting the absurdity of everything \
around us is one step, a necessary experience: it should not \
become a dead end. It arouses a revolt";
  printf("string = %s\n", testStr);

  int len = stringlength(testStr);
  printf("%d\n", len);
```

3. For each character in the string

```
...
  // count characters in the string
  for (i = 0; i < len; i++){
    char c = testStr[i];
    // check for uppercase.  if found then lowercase
    if (isupperalphabetic(c)) {
      c =  lower(c);
    }

    // if c is a character, count it.
    if (isloweralphabetic(c)) {
      int j = c - 'a';
      // char_counts[j] = char_counts[j] + 1;
      char_counts[j]++; // The same as above
    }
  }
...
```

4. Print the array

For each element in the array, we print the character for that array element and the count.

```
...
  // print each character followed by its count
  for (i = 0; i < 26; i++) {
    printf("character: %c appears %d times.\n", i + 'a', char_co
  }
  return EXIT_SUCCESS;
}
```

EASTERN WASHINGTON UNIVERSITY

Assignment 2

We have used the C Programming Language concepts to build some programs that can process ASCII strings. We can use these functions to do other things as well. For example:

- Write a function to recognize if a character is a digit.
- Write a function to recognize if a character is whitespace.
- Write a function that can recognize a number (a consecutive set of digits).
- Write a function to recognize words (consecutive letters between spaces and punctuation).

EASTERN WASHINGTON UNIVERSITY