CHARACTERS AND STRINGS

CSSE 120—Rose Hulman Institute of Technology

Characters and Strings

Characters in Python

- Just a one-character ***string***

```
>>> myChar = 'C'
>>> print myChar
C
>>> print ord(myChar) # converts character to int
67
>>> print chr(67)          # converts int to character
C
```

Characters in C

- C's **char** type is really a kind of number!
- A **char** takes 1 byte (8 bits) of storage space
  - Today's world requires more than 1 byte of space to cover all the characters in all the world's languages. Hence there are provisions (that we will not pursue) for extended-length characters.
- Predict the output:

```
char myChar;
myChar = 'C';
printf("%c\n",  myChar);   /* %c is format spec. for char */
printf("%d\n", myChar);
printf("%c\n", 67);
myChar++;
printf("%c\n",  myChar);
```

Q1

Seven Ways to Say 'A'

```
int i = 'A';
printf("A");
printf("%c", 'A');
printf("%c", 'B'-1);
printf("%c", i); /* %d here would print 65 */
putchar('A');  /* can "push" single characters to console*/
putchar(toupper('a'));  /* Need to #include <ctype.h> */
putchar(i);
printf("Eh!");
```

Q2

ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20	Space	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	DEL

Summary: Math with Characters

```
'C' + 1 == 'D'
char b = 'b';
b--;
putchar(b);  /* outputs a */
```

- Checkout **26-CharactersAndStrings**
- Do TODO #1 and #2
  - It asks you to combine these ideas to write a **for** loop that prints the characters from 'a' to 'z' on a single line

Character Input

- To read a single character from the console use:
  - **`getchar()`**
  - Caveat: **`getchar()`** returns an ***int*** (not a *char*), either a character value or **EOF** (end of file). Store its returned value in an *int*, not a *char*. (Though *char* works on *some* systems.)
- Do TODO's #3 and 4 in **27-CharactersAndStrings**

```
int inChar;
int count = 0;
printf("Type some text, then press 'Enter': ");
fflush(stdout);
inChar = getchar();
while (inChar != '\n') {
        count++;
        inChar = getchar();
}
printf("You entered %d characters.\n", count);
```

Note: most operating systems only pass characters to your program after the user presses the **enter** key

EOF is control-z in Windows, control-d in Unix

Q3

Character Functions: *ctype.h*

- Conversion Functions:
  - `int tolower(int c);`
  - `int toupper(int c);`
- Test functions:
  - `isdigit(int c)`
  - `isalpha(int c)`
  - `islower(int c)`
  - `isupper(int c)`
  - `isspace(int c)`

See the C *Library Reference* link on the Course Resources for more functions.

- Do TODO #5 and #6 in **27-CharactersAndStrings**
  - Use `isspace`

Strings

- A string in C is just
  - An array of characters,
  - **with a '\0' at the end**
- Examples (two ways to do the same thing):

```
char r[4];
r[0] = 'b';
r[1] = 'o';
r[2] = 'b';
r[3] = '\0';
```

```
char r[4] = "bob";
```

- Do TODO #7 and #8 in **27-CharactersAndStrings**
  - Note what goes wrong if you forget the '\0' and/or don't allocate enough space for the string (including the '\0').

String functions in *string.h*

Function	Purpose
char* strncpy(char* dest, char* src, int n)	copy up to *n* characters of string *src* to string *dest*; return *dest*. Includes the '0' only if it fits. **Strings are *mutable* in C, unlike Python!** **Must reserve space for dest before calling *strncpy*.** Safer than *strcpy*, which will overflow *dest* if *src* is too long.
char* strncat(char* dest, char* src, int n)	concatenate up to n characters of string *src* to end of *dest*; return *dest*. **Must reserve space for *dest* before calling *strncat*.** Safer than *strcat*.
int strncmp(char* str1, char* str2, int n)	compare 1st n characters of string *str1* to string *str2*, return a negative number if *str1*<*str2*, zero if *str1*==*str2*, or positive otherwise (use lexicographical – alphabetic – ordering)
size_t strlen(char* str)	return length of *str* (*size_t* is a typedef for *int* on most systems) **Doesn't include the null character.** Will give wrong answer or may crash if str is mistakenly not null-terminated.

Note: we usually ignore the return values from strncpy and strncat, since their purpose is to mutate dest.

See Kochan or the C *Library Reference* link on Course Resources page for more info. The *string.h* library is perhaps C's weakest and most easily abused library.

Example: strncpy

- Example: Suppose you have a string s:

```
char s[source_size] = ...;
        ...
```

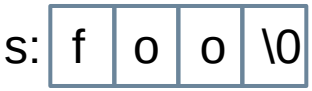
Then the following copies *s* into string *t*, but copying no more than *destination_size* characters, thus avoiding a potential buffer overrun (which is why *strcpy* is unwise).

```
char t[destination_size];
strncpy(t, s, destination_size);
t[destination_size - 1] = '\0';
```

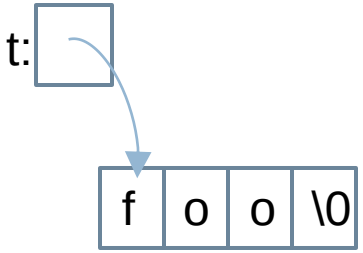
Do TODO #9 – 14 in **27-CharactersAndStrings**

String variables vs. constants

- String Variable
- char s[] = “foo”;



- String Constant
- char *t = “foo”;
- Strings declared in this way **cannot** be mutated!



- Do TODO #15 and 16 in **27-CharactersAndStrings**

Summary: Strings in C

- Strings are arrays of characters:
  - `char firstName[4] = "Lou";`

  or
  - `char lastName[10];`
    `strncpy(lastName, "Gehrig", 10);`
    `lastName[9] = '0';`
- "Null terminated", that is, a '\0' at the end
- Strings are (generally) mutable (since arrays are pointers)
- Don't forget to reserve enough space to hold the string

When C Gives You Lemons…

- Problem:
  - Python includes high level functions for strings
  - C (and some other languages) do not
  - What if you need to use C, but also need strings?
- Solution: Make your own string functions!
- Homework:
  - Finish the functions in **27-CharactersAndStrings**
    - TODO's 17 - 21
  - Read the upcoming project (see homework27).