

## As you arrive:

1. Start up your computer and plug it in
2. **Log into Angel** and go to CSSE 120
3. Do the **Attendance Widget** – the PIN is on the board
4. Go to the course **Schedule Page**
5. Open the **Slides** for today if you wish
6. Check out today's project:

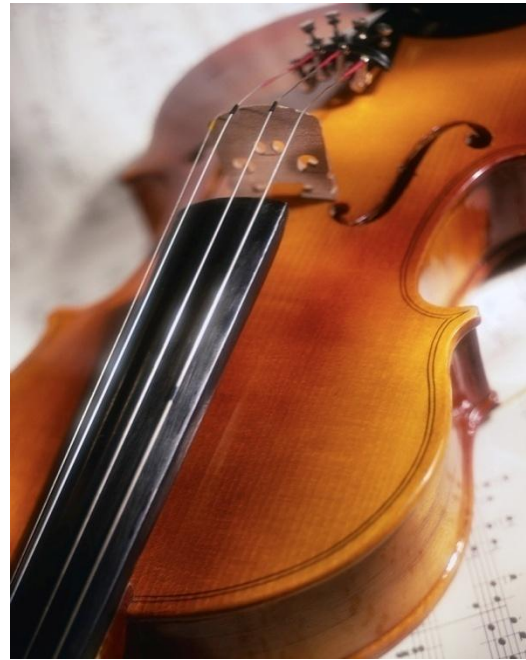
**Session27\_CharactersAndStrings**

*Plus in-class time working on these concepts AND practicing previous concepts, continued as homework.*

# Characters and Strings

- Characters: What they are. How to use them.
- Strings: What they are. How to use them.

# Characters and Strings



# Outline

## □ Characters

- What they are
- Math with characters. ASCII
- Character output with `putchar()`
- Character input with `getchar()`
- Character functions from the `ctype.h` library, e.g.:
  - `toupper()`
  - `isspace(c)`

## □ Strings

- What they are
  - Arrays of characters
  - Terminated by a `'\0'`
- How to allocate space and initialize them
- String functions from the `string.h` library
- Gotcha's regarding strings
- Looping through strings, mutating 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
- Predict the output:

```
char myChar;  
myChar = 'C';
```

```
printf("%c\n", myChar); // %c is format spec. for char
```

```
printf("%i\n", myChar);
```

```
printf("%c\n", 67);
```

```
myChar++;
```

```
printf("%c\n", myChar);
```

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.

# Seven Ways to Say 'A'

```
printf("A");  
printf("%c", 'A');  
printf("%c", 'B'-1);  
char a1 = 'A';  
int a2 = 'A';  
printf("%c %c", a1, a2);  
putchar('A'); // can "push" single characters to console  
putchar(toupper('a')); // Need to #include <ctype.h>  
putchar(a1); putchar(a2);  
printf("Eh!");
```

# ASCII Table

ASCII is gradually giving way to Unicode, which allows for larger alphabets.

Note that the letters are “in order”, although the lower case follow the upper case with some special characters in between.

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' is true, so:

```
char b = 'b';  
b--;  
putchar(b); /* outputs 'a' */
```

- Combine these ideas to write a **for** loop that prints the characters from 'a' to 'z' on a single line
  - ▣ Try this in Eclipse; you may work with a neighbor
    - Today's project [Session27\\_CharactersAndStrings](#) has a function called `printAtoZ` ready for you to do this.
      - TODO's #2 and #3 in the project



# Character Input

- To read a single character from the console use:

`getchar()`

- Study the following code – we typed it for you into function `countInputUntilEndOfLine` in today's project `Session27_CharactersAndStrings`. Then run it (TODO #4).

```
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 %i characters.\n", count);
```

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.)

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

**Q3**

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

# Character Functions: *ctype.h*

## □ Conversion Functions:

- `int tolower(int c);`

- `int toupper(int c);`

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

## □ Test functions:

- `isdigit(int c)`

- `isalpha(int c)`

- `islower(int c)`

- `isupper(int c)`

- `isspace(int c)`

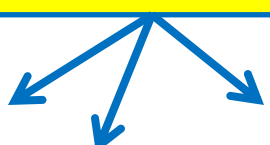
Returns true if `c` is a “white-space” character – space, form feed, newline, carriage return, tab, or vertical tab.

- Write a function called `countSpaces` to count the number of *white-space* characters entered in a line of input
  - We placed this function into today’s project: TODO’s #5 and 6.
  - Use `isspace`

# Strings

- A **string** in C is just:
  - An array of characters
  - With a `'\0'` at the end
- Examples:

Three ways to do the same thing. We will shortly see a 4<sup>th</sup> way, using `strcpy`.



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

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

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

- The `printAString` function in today's project shows what can go wrong if you forget the `'\0'` and/or don't allocate enough space for the string (including the `'\0'`).
  - Examine `printAString`, then run it (per TODO #7).
  - Do you see how to correct the two errors in `printAString`?

# String variables vs. constants

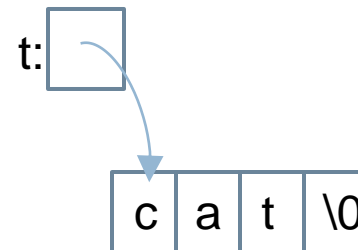
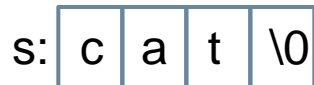
- String Variable

```
char s[] = "cat";
```

- String Constant

```
char *t = "cat";
```

- Strings declared in this way **cannot** be mutated!



# String Functions: *string.h*

We usually ignore return values from `strcpy` and `strcat`, since they mutate *dest*.

Function	Purpose
<code>char* strcpy(char *dest, char *src)</code>	Copy string <i>src</i> to string <i>dest</i> , including <code>'\0'</code> ; return <i>dest</i> . <b>Note: strings are mutable in C, unlike Python! Must reserve space for <i>dest</i> before calling <code>strcpy</code>.</b>
<code>char* strcat(char *dest, char *src)</code>	Concatenate string <i>src</i> to end of string <i>dest</i> ; return <i>dest</i> . <b>Must reserve space for <i>dest</i> before calling <code>strcat</code>.</b>
<code>int strcmp(char *str1, char *str2)</code>	Compare string <i>str1</i> to string <i>str2</i> ; return a negative number if <i>str1</i> < <i>str2</i> , zero if <i>str1</i> == <i>str2</i> , or positive otherwise. Use lexicographical – alphabetic – ordering.
<code>size_t strlen(char *str)</code>	Return length of <i>str</i> ( <code>size_t</code> is a typedef for <code>int</code> on most systems). <b>Doesn't include the null character.</b>

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 (insecure) library. For example, there is no check that enough space is allocated or that the strings are null-terminated.

# String Concatenation Using *strcat()*

- Consider:

```
char s1[] = "Go, Red! Go, White! ";  
char s2[] = "Go Rose, Fight!";
```

```
/* You write code here */
```

```
printf("%s\n", s3);
```

- What goes in the middle? We want:

- the output to be

```
Go, Red! Go, White! Go Rose, Fight!
```

- and no additional string literals

# Summary: Strings in C



Key  
Points!

- Strings are *arrays* of *characters*:

```
char firstName[] = "Lou";
```

or

```
char lastName[10];
```

```
strcpy(lastName, "Gehrig");
```

- “Null terminated”, that is, a `'\0'` at the end
- Strings are (generally) mutable (since arrays are like pointers)
- Strings can be abused easily. Be careful to:
  - ▣ Terminate the string with a `'\0'`.
  - ▣ Reserve enough space to hold the string, including its `'\0'`.
  - ▣ Use the *string.h* functions we listed with caution – if the arguments are not what you expect, things can go quite bad.

# 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!
  - ▣ Rest of today:
    - Finish the functions in today's project:  
**Session27\_CharactersAndStrings**
      - That is, do TODO's 8 and following.
      - Let's start it together.
    - Ask questions as needed!
    - Don't merely make the code "work".  
Make sure you understand the C notation and how to use it.