Strings

String Literals

- C and Java both have string literals
- For both, string literals are surrounded by double quotes
- Examples:
  "hello, world"
  "Please enter your name: "
  "x = %d, y = %d"
  "<html>\n<body>\nUnder Construction!\n</body>\n</html>\n"

String as a data type?

- Java has a string data type - the String class. A String is an object that may be initialized with a string literal or by some other means.

```
String msg1 = "hello";
System.out.println("the length of msg1 is " + msg1.length);
if (msg1.equals(msg2))
    System.out.println("they're equal!");
```

- C does not have a string data type.
- In C, a string is considered to be a null-terminated sequence of characters, typically stored in a char array*. The null character ('\0') is used to signify the end of the string. Example:

```
char msg[] = "hello";
```

* we'll discuss another scenario shortly

Populating a char Array

- Several ways to do the same thing:
  1. char msg[] = "hello";
  2. char msg[6]; // make sure to allocate space for the '\0'!
     msg[0] = 'h';
     msg[0] = 'e';
     msg[0] = 'l';
     msg[0] = 'l;
     msg[0] = 'o';
     msg[0] = '\0';
  3. char msg[] = {'h', 'e', 'l', 'l', 'o', '\0'};
  4. char msg[6] = "hello";

- The array may be bigger than a string inside it
  char msg[9] = "hello";

Tips

h	e	l	l	o	\0	?	?	?

- Make sure your array is big enough,
  max string size + 1 (for the null character)
- If you're constructing a string with individual characters, make sure to put '\0' at the end
- An array can hold strings of varying lengths. The array above could be overwritten with a smaller string (e.g. "hi"). Note that now just the first 3 chars are relevant and that old data that isn't overwritten remains.

h	i	\0	l	o	\0	?	?	?

- The array could be overwritten with a bigger string (e.g. "bigger").

b	i	g	g	e	r	\0	?	?

Literals w/out Arrays

- A string literal not assigned to an array is read-only and stored in the Data/Static area of memory (along with the program code, global variables, and static variables)
- Arrays are stored on the stack (unless global/static)

```
void foo(char *);

int main(){
  char line[] = "test 123";
  char *beta = "Oct 22,2010";
  foo(line); // changes line array to "best 123"
  // foo(version) would fail;version is read-only
}

void foo(char *p){
    *p = 'b';
}
```

Pointer Traversal

Because strings are null-terminated, we can find the

A pointer is commonly used to traverse a string.

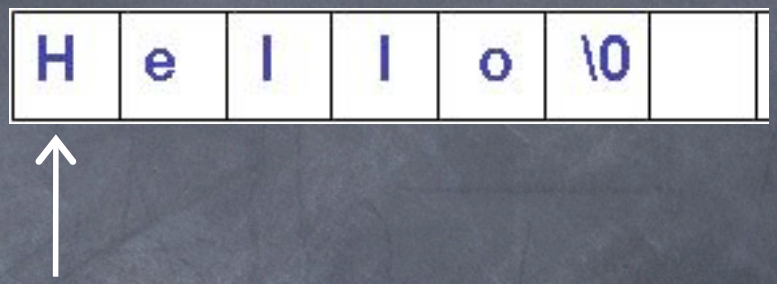
```
char word[] = "hello";
```

```
// With [] notation:
int i = 0;
for (i = 0; word[i] != '\0'; i++)
    printf("%c ", word[i]);
```

```
// With pointer notation:
char *p;
for (p = word; *p != '\0'; p++)
    printf("%c ", *p);
```

Why are C Strings useful?

- The NULL termination means we can find the length of the string using a for loop
- As long as we NULL terminate every string we use and allocate big enough char arrays, we can deal with variable sized text.
- Opens up a new world of functional possibilities



The string.h library

- We are not the first people to find out that NULL termination is a good idea.
- The **string.h** library is an set of C functions devoted to NULL terminated char arrays
- All the functions in string.h accept pointers to C strings as arguments.
- They operate on them or give us the information we need to parse / control them

The big string.h functions

- Some of the most important string.h functions:
- int strlen(char* string)
  - Returns the string length (to '\0')
  - NOT THE ARRAY LENGTH
- char* strcpy(char* dest, char* src):
  - Loops through src, copies to dest. Also returns dest

The big string.h functions ctd

- int strcmp(char* a, char* b)
- Compares the first character in a and b
- If equal, moves on to the next character.
- For first characters that are not equal,
  - returns a positive number if ascii value of a[0] is > ascii value of b[0]
  - returns a negative number if ascii value of a[0] is < ascii value of b[0]

Other string.h Functions

- char* strcat(char* dest, char* src)
  - Concatenates src onto dest
  - Dest must be large enough to hold src
- char* strncmp(char* a, char* b, size_t n)
  - Compare a and b for n characters
- char* strrchr(char* str, char c)
  - Locate last occurrence of c in str, return pointer

Bottom Line

- When in Rome, do as the Romans do
- When in C, use C strings to hold text
  - Extremely convenient
  - List of functions that can manipulate them easily
  - It is the standard
- Don't forget: Array size must be (max string length + 1) (why?)