Strings

- Strings are not a built-in data type.
- C provides almost no special means of defining or working with strings.
- A string is an array of characters terminated with a “null character” (\'0\')

String literals

```c
char *name = "csc209h";
printf("This is a string literal\n");
```

- String literals are stored as character arrays, but you can't change them.

```c
name[1] = 'c'; /* Error */
```

- The compiler reserves space for the number of characters in the string plus one to store the null character.

String Variables

- arrays are used to store strings
- strings are terminated by the null character (\'0\') (That's how we know a string's length.)
- Initializing strings:
  - `char course[8] = "csc209h";`
  - course is an array of characters
  - `char *s = "csc209h";`
  - s is a pointer to a string literal

String functions

- The library provides a bunch of string functions which you should use (most of the time).
  - `man string`
  - `int strlen(char *str)`
    - returns the length of the string. Remember that
      the storage needed for a string is one plus its length
### Copying a string

`char *strncpy(char *dest, char *src, int size)`
- copy up to size bytes of the string pointed to by src in to dest. Returns a pointer to dest.
- Do not use `strcpy` (buffer overflow problem)

```c
char str1[3];
char str2[5] = "abcd";
/*common error*/
strncpy(str1, str2, strlen(str2));
```

### Concatenating strings

`char *strncat(char *s1, const char *s2, size_t n)`
- appends the contents of string `s2` to the end of `s1`, and returns `s1`.
- only appends up to `n` bytes to `s1`
- Watch out! It is easy to forget how much space is left.
  ```c
  char str1[6] = "abc";
  strncat(str1, "def", 6); /*wrong*/
  ```

### Comparing strings

`int strcmp(const char *s1, const char *s2)`
- compares `s1` and `s2`, returning a value less than, equal to, or greater than 0 depending on whether `s1` is less than, equal to, or greater than `s2`.

```c
if( strcmp(str1, str2) <= 0 )
  /* is str1 <= str2 */
```