COMSC-165 Lecture Topic 7
C Strings and C++ Strings

Reference
Deitel, chapter 6.11-6.12
Tutorial

C++ String
```
#include <string>
using std::string;
```
basically a vector of `char`s
...with == and other relational operators added
...and + and += for concatenation
...and substr function
strings are *objects*

C Strings
not a data type in C
use arrays to get string functionality
no library to include, UNLESS you use functions
```
#include <cstring>
  strlen(const char*);
  strcpy(char*, const char*);
```
  do not use `=`
```
  strcmp(const char*, const char*);
```
  do not use `==`
```
  strupr(char*);
```
Microsoft only
```
  strlwr(char*);
```
Microsoft only
leave a `char` for the **null terminator**
  to store a string of 11 letters,...
  ...you need a size 12 `char` array

not all `char` arrays are C strings
  C strings have null terminators

Initializing C Strings
```
  char hello[] =
    {'h', 'e', 'l', 'l', 'o', '\0'};
  char world[] = {"World"};
  char world[] = "World";
```
use for "strings"

C String I/O
Mixing **`cin >>`** and **`cin.getline`** for C strings
  use **`cin.ignore(1000, 10);`** *after* **`>>`**

Two Ways To Write C Strings
```
  char c[] = "Hello";
  char* c = "Hello";
```
// dangerous -- see below
  ...not exactly the same!

Character Arrays
initialization methods wo/pointers
```
  char c[] = "Hello"; // ok
  c = "Hello, World"; // fails to compile
  c[0] = 'X'; // ok
```
initialization methods w/pointers
```
  char* c = "Hello"; // dangerous
  c = "Hello, World!"; // ok
  c[0] = 'X'; // BAD IDEA
  const char* c = "Hello"; // best
```
initializating string arrays
```
  char d[7][32] = {"Sunday",...
```
    cannot reset wo/**`strcpy()`**
```
  char* d[7] = {"Sunday",...
```
    can reset w/**`d[0] = "SUN";`**
  no **`delete`** without **`new`**
  no initialization with **`new`** *(wo/"constructors")*

Passing C Strings To Functions
with a terminator...
```
  void fun(char[]); // mutable C string
```
  ...or `void fun(char*); // mutable, reassignable`
  ...or `void fun(const char*); // immutable`
```
  fun(anArray);
```
1. with the size of the array (or #of elements) specified...
```
  void fun(int, int[]);
```
  ...or `void fun(int, int*);`
```
  fun(100, anArray);
```
2. with the size unspecified, using a sentinel
3. with the size unspecified, but written as number

2D Array as Array of Strings
```
  char nums[2][6] = {"one","two"};
```
  the "2" is optional
```
  strcpy(nums[0], "hello");
  strcpy(nums[1], "world");
```

`cout` displays *contents of string* -- not memory address
Use **`getline(cin, ...)`** for C++ strings
in <string>, with using std::getline

e.g., `char c[80];` // a "buffer"
`cin.getline(c, 80);` // store up to 79 chars
any extra chars, plus ENTER, are consumed

Advanced C String Functions

`#include <cstring>`
`strlen` *returns length of string*
`strcmp`, `stricmp` (*Microsoft only*), `strncmp` *compare contents*
...because `==` compares memory addresses of strings
`strcpy`, `strncpy` *copy one string to another*
`strcat` *add one string to end of another*
`strchr`, `strrchr`, `strstr`