Character Testing

- A **C-string**: sequence of characters stored in adjacent memory locations and terminated by the `NULL` character
  - An array of chars can be used to define storage for a string
  - MUST leave `NULL` at the end of the char array
  - Values can be entered using `cin` or `>>`
    - For inputs containing spaces use `cin.getline()`
  - More specifically, C-strings are stored in a *char* array ending in null (`\0`)

> **C String Initialization**
>
> ```
> char s1[9]= {'J', 'o', 'h', 'n', ' ', 'L', 'e', 'e','\0'};
> char s2[9]= "John Lee";
> char s3[] = {'J', 'o', 'h', 'n', ' ', 'L', 'e', 'e', '\0'};
> char s4[] = "John Lee";
> const char *  s5 = "John Lee";
> ```
>
> ■ Not a C string
>
> ```
> char arr [8] = {'J', 'o', 'h', 'n', ' ', 'L', 'e', 'e'};    // WHY ?
> ```

```
char s1[9];
s1 = "John Lee";          //invalid
```

Using an array, you can initialize a C string at creation but not afterwards.

  - The moment a `\0` is found, the string automatically stops
  -

- **String literal (string constant)**: sequence of characters enclosed in double quotes
- `cin.get(strName, numChar+1)` is a method to get user input, by setting a limit or until `\n` is found

    - strName is the string object that the result is put in
- *Review: strlen(str) returns the length of str, strcat(str1, str2) combines str2 to the end of str1*
- **strstr(str1,str2)** finds the first instance of str2 in str1... (seems pretty useful)
    - Returns a pointer to match or `NULL`
- **strchr(str1, char)** is also a good tool, finds the first occurrence of a char
    - DOES NOT RETURN -1 LIKE JAVA

`cctype` Class

FUNCTION	MEANING
`isalpha`	`true` if arg. is a letter, `false` otherwise
`isalnum`	`true` if arg. is a letter or digit, `false` otherwise
`isdigit`	`true` if arg. is a digit 0-9, `false` otherwise
`islower`	`true` if arg. is lowercase letter, `false` otherwise
`isprint`	`true` if arg. is a printable character, `false` otherwise
`ispunct`	`true` if arg. is a punctuation character, `false` otherwise
`isupper`	`true` if arg. is an uppercase letter, `false` otherwise
`isspace`	`true` if arg. is a whitespace character, `false` otherwise

`cstring` Class

- Functions that takes greater than 1 C-string as arguments can use
    - C-string name
    - Pointers to C-string
    - literal string
-

`cstdlib` Class

- *Seems like a good tool to convert strings into different types*

FUNCTION	PARAMETER	ACTION
`atoi`	C-string	converts C-string to an `int` value, returns the value
`atol`	C-string	converts C-string to a `long` value, returns the value
`atof`	C-string	converts C-string to a `double` value, returns the value
`itoa`	`int`, C-string, `int`	converts 1st `int` parameter to a C-string, stores it in 2nd parameter. 3rd parameter is base of converted value

- `itoa` does not check for bounds, make sure that enough space is allocated

`string` Class

- *special data type working wiht strings!*
- Can use relational operators directly to compare string objects

Definition	Meaning
`string name;`	defines an empty string object
`string myname("Chris");`	defines a string and initializes it
`string yourname(myname);`	defines a string and initializes it
`string aname(myname, 3);`	defines a string and initializes it with first 3 characters of `myname`
`string verb(myname,3,2);`	defines a string and initializes it with 2 characters from `myname` starting at position 3
`string noname('A', 5);`	defines string and initializes it to 5 `'A'`s

OPERATOR	MEANING
`>>`	extracts characters from stream up to whitespace, insert into string
`<<`	inserts string into stream
`=`	assigns string on right to string object on left
`+=`	appends string on right to end of contents on left
`+`	concatenates two strings
`[]`	references character in string using array notation
`>, >=, <, <=, ==, !=`	relational operators for string comparison. Return `true` or `false`

- Strings have many overloaded operators (like a lot...)
  - **assignment:** `assign, copy, data`
  - **modification:** `append, clear, erase, insert, replace, swap`
  - **space management:** `capacity, empty, length, resize, size`
  - **substrings:** `find, substr`
  - **comparison:** `compare`