- An **array** is a special variable having one name but stores a list of data items.
  - Each item in the array is known as an **element**
  - Some languages use a similar construct known as a **vector**
- An **index** is the location at which a value is in an array
- [] are known as **brackets** and {} are **braces**
- C strings are like: **char movies[20] = "Star Wars"**
  - Because its not 20 letters, the remaining characters are left as **null characters** aka "\0"
    - Any array characters ending with a null character is known as a **null-terminating string**
  - Upon print, it will just look like a normal string. ex. "Star Wars"
- C++ comes with a CString librarby to use
  - #include <cstring>

strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including the null character) to destStr. destStr must have enough space for sourceStr, including the null character.
		`strcpy(targetText, userText);`
		`  // Copies "UNICEF" + null char to targetText`
		`strcpy(targetText, orgName);`
		`  // Error: "United Nations" has > 10 chars`
		`targetText = orgName;`
		`  // Error: Strings can't be copied this way`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters from sourceStr. destStr must have enough space for the copied characters, including the null character. If sourceStr has less than numChars characters, strncpy will fill the remaining space with the null character.	`strncpy(orgName, userText, 6);` `  // orgName is "UNICEF Nations"`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr to *end* of destStr (starting at destStr's null character) and then appends a null character. destStr must have enough space for the addition of sourceStr plus the null character.	`strcat(orgName, userText);` `  // orgName is "United NationsUNICEF"`

strncat()	`strncat(destStr, sourceStr, numChars)`
	
	Copies up to numChars characters from sourceStr to *end* of destStr (starting at destStr's null character) and then appends a null character. destStr must have enough space for the addition of sourceStr plus the null character. Copying of sourceStr characters continues until either the end of sourceStr or numChars is reached.
		`strcpy(targetText, "abc");`
		`  // targetText is "abc"`
		`strncat(targetText, "123456789", 3);`
		`  // targetText is "abc123"`

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the cstring library.
		`if (strchr(orgName, 'U') != NULL) { // 'U' exists in orgName?`
		`  ...  // 'U' exists in "United Nations", branch taken`
		`}`
		`if (strchr(orgName, 'u') != NULL) { // 'u' exists in orgName?`
		`  ...  // 'u' doesn't exist (case matters), branch not taken`
		`}`

strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);  // Assigns 14 to x`
		`x = strlen(userText);  // Assigns 6 to x`
		`x = strlen(targetText); // Error: targetText may lack null char`

strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName, "United Nations") == 0) {`
		`  ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) != 0) {`
		`  ... // Not equal, branch taken`
		`}`

isalpha(c) -- Returns true if c is alphabetic: a-z or A-Z.	`isalpha('A');  // Returns true`
	`isalpha(myString[0]);  // Returns true because 'H' is alphabetic`
	`isalpha(myString[3]);  // Returns false because '9' is not alphabetic`

isdigit(c) -- Returns true if c is a numeric digit: 0-9.	`isdigit(myString[3]);  // Returns true because '9' is numeric`
	`isdigit(myString[4]);  // Returns false because ! is not numeric`

isalnum(c) -- Returns true if c is alphabetic or a numeric digit. Thus, returns true if either isalpha or isdigit would return true.	`isalnum('A');  // Returns true`
	`isalnum(myString[3]);  // Returns true because '9' is numeric`

isspace(c) -- Returns true if character c is a whitespace.	`isspace(myString[5]);  // Returns true because that character is a space ' '.`
	`isspace(myString[0]);  // Returns false because 'H' is not whitespace.`

islower(c) -- Returns true if character c is a lowercase letter a-z.	`islower(myString[0]);  // Returns false because 'H' is not lowercase.`
	`islower(myString[1]);  // Returns true because 'e' is lowercase.`
	`islower(myString[3]);  // Returns false because '9' is not a lowercase letter.`

isupper(c) -- Returns true if character c is an uppercase letter A-Z.	`isupper(myString[0]);  // Returns true because 'H' is uppercase.`
	`isupper(myString[1]);  // Returns false because 'e' is not uppercase.`
	`isupper(myString[3]);  // Returns false because '9' is not an uppercase letter.`

isblank(c) -- Returns true if character c is a blank character. Blank characters include spaces and tabs.	`isblank(myString[5]);  // Returns true because that character is a space ' '.`
	`isblank(myString[0]);  // Returns false because 'H' is not blank.`

isxdigit(c) -- Returns true if c is a hexadecimal digit: 0-9, a-f, A-F.	`isxdigit(myString[3]);  // Returns true because '9' is a hexadecimal digit.`
	`isxdigit(myString[1]);  // Returns true because 'e' is a hexadecimal digit.`
	`isxdigit(myString[6]);  // Returns false because 'G' is not a hexadecimal digit.`

ispunct(c) -- Returns true if c is a punctuation character. Punctuation characters include: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~	`ispunct(myString[4]);  // Returns true because '!' is a punctuation character.`
	`ispunct(myString[6]); // Returns false because 'G' is not a punctuation character.`

isprint(c) -- Returns true if c is a printable character. Printable characters include alphanumeric, punctuation, and space characters.	`isprint(myString[0]);  // Returns true because 'H' is a alphabetic.`
	`isprint(myString[4]);  // Returns true because '!' is punctuation.`
	`isprint(myString[5]);  // Returns true because that character is a space ' '.`
	`isprint('\0');  // Returns false because the null character is not printable`

iscntrl(c) -- Returns true if c is a control character. Control characters are all characters that are not printable.	`iscntrl(myString[0]);  // Returns false because 'H' is a not a control character`
	`iscntrl(myString[5]);  // Returns false because space is a not a control character`
	`iscntrl('\0');  // Returns true because the null character is a control character`

toupper(c) -- If c is a lowercase alphabetic character (a-z), returns the uppercase version (A-Z). If c is not a lowercase alphabetic character, just returns c.	`letter = toupper(myString[0]);  // Returns 'H' (no change)`
	`letter = toupper(myString[1]);  // Returns 'E' ('e' converted to 'E')`
	`letter = toupper(myString[3]);  // Returns '9' (no change)`
	`letter = toupper(myString[5]);  // Returns ' ' (no change)`

tolower(c) -- If c is an uppercase alphabetic character (A-Z), returns the lowercase version (a-z). If c is not an uppercase alphabetic character, just returns c.	`letter = tolower(myString[0]);  // Returns 'h' ('H' converted to 'h')`
	`letter = tolower(myString[1]);  // Returns 'e' (no change)`
	`letter = tolower(myString[3]);  // Returns '9' (no change)`
	`letter = tolower(myString[5]);  // Returns ' ' (no change)`

- 2D arrays aka **row-major order**
- Vectors need to be imported via `#include <vector>`

    - Format to create a vector follows: `vector<datatype> name;`
- **push_back()** appends a new element at the end of an existing vector while **.(at)** accesses at a specific index
- **<>** are known as **angle brackets** or **chevrons**
- **resize(N)** can be used to adjust the size of a vector
- **back()** returns the last element of a vector
- **pop_back()** removes the last element of a vector
-