Chapter 9: Strings

(To avoid confusion, C-style strings will be referred to as "C-string", regular C++ strings from the string class will be referred to as "String").

C-Strings

- Made up of character arrays that stores strings of characters. Different from a regular character array since the end of the string is marked with the null character '\0'.
- Example: char s[] = "Hello";

'H'	'e'	'l'	'l'	'o'	'\0'
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]

- String literals, such as "Hello", are C-Strings.

Initialization:

- Can be initialized by declaring a character array with a set size:
  char s[6] = "Hello";
- Omit the size and C++ will automatically make the best fit:
  char s[] = "Hello";
- The size declared does not have to be a perfect fit with the string literal:
  char s[20] = "Hello";

Assignment:

- Can use the equal sign to assign strings only when declaring and initializing at the same time.
- Example of invalid code:
  char s[6];
  s = "Hello"; // Illegal!
- However, you can manipulate a character at a time:
  char s[6] = "hello";
  s[0] = 'c';
  cout << s << endl; // prints **cello**

Be careful not to accidentally wipe out the null character '\0', since other built-in string functions depend on it to mark the end of the C-string.

- Instead of the using the = operator, you can use the **strcpy** function to assign a value to a C-string. (To use, add #include <cstring> to the list of preprocessor directives).
- Example:
  char s[10] = "hello";
  strcpy(s, "goodbye");
  cout << s << endl; // prints **goodbye**

- However, **strcpy** does not check if the assignment remains within bounds in the array. This can have dangerous consequences as memory locations not allocated for use can be replaced with the string content.

  Example:
  ```
  char s[10] = "hello";
  strcpy(s, "supercalifragilisticexpialidocious");
  cout << s << endl;      // prints supercalifragilisticexpialidocious
  ```
- A remedy to this is the **strlcpy** (L, not i) function, where an extra 3rd parameter SIZE is added for the maximum size, where it will copy the first SIZE-1 characters to the target and reserve the last slot for the null character.

  Example:
  ```
  char s[10] = "hello";
  strlcpy(s, "supercalifragilisticexpialidocious", 10);
  cout << s << endl;      // prints supercali
  ```

  Cons to strlcpy:
  1. Not part of the C++ standard library
  2. No solution to 0 characters (when SIZE is 0)

<cstring> functions:

- Copy: *strcpy (TARGET_CSTR, CSTR)*
- Concatenation*: strcat (TARGET_CSTR, CSTR)*
- Length: *strlen (CSTR)*
- Comparator: *strcmp (CSTR1, CSTR2)*

Strings

- We should be familiar with how Strings work. Instead of treating strings as character arrays, C++ Strings treat them as a basic datatype without having to keep track of array manipulation. Strings are part of the <string> class.
- Examples:
  ```
  string s = "hello";     // declaration + initialization
  string x;               // declaration
  x = "goodbye";          // assignment
  ```
- Can also manipulate the characters in a string:

  Example:
  ```
  string s = "hello";
  s[0] = "c";
  cout << s << endl;      // prints cello
  ```

C-strings vs. Strings

C-String	String
strcpy	= operator
strcat	+ operator
strlen	.length()
strcmp	==, <, >, <=, >=, != operators

<cctype> functions:
Takes in a character parameter. For boolean functions, it returns a 0 if false, and a non-negative number if true.

- toupper
- tolower
- isupper
- islower
- isdigit
- isalpha

I/O

Code	Console
string name; cout << "What is your name? "; cin >> name; cout << "Hello, " << name << "!" << endl;	What is your name? John Smith Hello, John!

When using **cin** to read strings, the input is read until it hits a whitespace (such as blanks, tabs, line breaks).

To read an entire line, use the **getline** function.

- **C-string**: *cin.getline (TARGET_ARRAY, MAX_SIZE)*
  Example:
  ```
  char a[80];
  cout << "Enter an input: ";
  cin.getline(a, 80);
  ```
  If the line is more than the MAX_SIZE, it will store the first MAX_SIZE – 1 characters to the array and reserve the last slot for the null character.
- **String**: *getline (cin, VARIABLE_NAME)*
  Example:
  ```
  string line;
  cout << "Enter an input: ";
  getline (cin, line);
  ```

The **get** function stores a C-string/character, regardless of whitespace.

- **C-string**: cin.get (TARGET_ARRAY, MAX_SIZE);
  Example:
  ```
  char a[80];
  cout << "Enter an input: ";
  cin.get(a, 80);
  ```
- **Character**: cin.get(TARGET_CHAR);
  Example:
  ```
  char next;
  cout << "Enter an input: ";
  cin.get(next);
  ```