# Module 5: Working with Text - C Strings (Days 18-20)

## 1. Philosophy Focus: Knowledge Graph

This module builds directly on your understanding of arrays and pointers. There is no string type in C. A "string" is simply a programming **convention** built on top of arrays of characters. Understanding this connection is key to mastering text manipulation in C and appreciating the safety of string types in other languages like Rust.

### **DAY 18: The C String Convention**

**Goal:** Understand that a C string is a char array ending with a special \0 character.

1. How a String is Stored

When you write a string literal like "hello", the compiler stores it in memory as an array of characters, and it automatically adds a special character at the end: the null terminator, written as \0.

Memory layout for "hello":

[ 'h' | 'e' | 'l' | 'l' | 'o' | '\0' ]

This \0 is crucial. It's how C functions know where the string ends. Without it, functions like printf would just keep reading memory forever (or until they crashed).

2. Declaring Strings

There are two primary ways to declare a string.

**Method 1: A char array.** This creates a **mutable** string in your program's local memory (the stack).

char my\_string[20] = "Hello"; // Creates a 20-char array, copies "Hello\0" into it.  
 // You have extra space to modify it later.  
my\_string[0] = 'J'; // This is OK. The string is now "Jello".

**Method 2: A char pointer.** This creates a pointer to a **read-only** string literal, which is stored in a special, protected part of the program's memory.

char \*my\_ptr\_string = "Hello"; // 'my\_ptr\_string' points to the 'H'.  
  
// DANGER: Attempting to modify this is UNDEFINED BEHAVIOR.  
// my\_ptr\_string[0] = 'J'; // DO NOT DO THIS. It will likely crash your program.

**Rule of Thumb:** If you plan to modify the string, use a character array. If it's a constant, a char\* is fine.

3. Printing Strings with %s

The %s format specifier in printf is designed for C strings. It tells printf to start at the given address and print characters until it finds a \0.

#include <stdio.h>  
  
int main(void) {  
 char greeting\_array[] = "Welcome"; // Compiler auto-calculates size (8 chars)  
 char \*greeting\_ptr = "Hello";  
  
 printf("Array version: %s\n", greeting\_array);  
 printf("Pointer version: %s\n", greeting\_ptr);  
   
 return 0;  
}

**4. Day 18 Practice**

1. Declare a mutable string using a char array that is large enough to hold your first name. Initialize it with your name.
2. Print the string.
3. Change the first character of the string to a different letter.
4. Print the modified string.

### **DAY 19: Common String Functions (<string.h>)**

**Goal:** Learn to use the standard library functions for common string operations.

Manually looping through character arrays to find their length or copy them is tedious and error-prone. C provides a standard library, <string.h>, to help.

**You must #include <string.h> to use these functions.**

**1. strlen(const char \*s): Get String Length**

* Takes a pointer to a string.
* Counts the number of characters **before** the null terminator.
* Returns the length as an integer (size\_t).
* strlen("hello") returns 5.

**2. strcpy(char \*dest, const char \*src): String Copy**

* Copies the string from src (source) **into** the dest (destination) buffer, including the \0.
* **DANGER:** This is the most infamous function in C. It does **not** check if dest is large enough! If src is larger than dest, it will write past the end of the buffer, causing a **buffer overflow**.
* A safer alternative is strncpy, which takes a size argument.

**3. strcmp(const char \*s1, const char \*s2): String Compare**

* Compares two strings lexicographically.
* Returns:
  + 0 if s1 is identical to s2.
  + < 0 if s1 comes before s2 alphabetically.
  + > 0 if s1 comes after s2 alphabetically.

**Inductive Example:**

#include <stdio.h>  
#include <string.h>  
  
int main(void) {  
 char name[50]; // A buffer to hold a name  
 char \*secret\_password = "password123";  
  
 printf("Enter your name: ");  
 scanf("%s", name); // scanf with %s is also dangerous, it can overflow!  
  
 printf("Hello, %s!\n", name);  
 printf("Your name has %zu characters.\n", strlen(name));  
  
 char password\_guess[50];  
 printf("Enter the password: ");  
 scanf("%s", password\_guess);  
   
 if (strcmp(password\_guess, secret\_password) == 0) {  
 printf("Access granted.\n");  
 } else {  
 printf("Access denied.\n");  
 }  
  
 return 0;  
}

**4. Day 19 Practice**

1. **Full Name:**
   * Declare two char arrays, first\_name and last\_name. Get input for both from the user.
   * Declare a third char array full\_name that is large enough to hold both, plus a space and a null terminator.
   * Use strcpy to copy the first\_name into full\_name.
   * Use strcat (string concatenation, look it up!) to add a space and then the last\_name.
   * Print the full\_name.

### **DAY 20: Spaced Repetition - Review Project**

**Goal:** Combine loops and string functions to process a string character-by-character.

1. Done-for-you Training Plan: String Reverser

Write a program that takes a string from the user and prints it in reverse.

**Project Requirements:**

1. Declare a char array (e.g., of size 100) to hold the user's input.
2. Prompt the user and read their input. A safer way to read a line of text (including spaces) is fgets(buffer, size, stdin);. Look up how to use it! scanf("%s", ...) will stop at the first space.
3. Write a function void reverse\_string(char \*str).
4. **In the function:**
   * First, find the length of the string using strlen.
   * Use a for loop (or a while loop) with two indices, start and end. start begins at 0, end begins at length - 1.
   * The loop should continue as long as start < end.
   * Inside the loop, swap the character at str[start] with the character at str[end]. You'll need a temporary char variable.
   * Increment start and decrement end in each iteration.
5. **In main:**
   * Call your reverse\_string function with the user's input.
   * Print the (now reversed) string.

Example:

Input: Hello World

Output: dlroW olleH

This project forces you to think of a string as what it truly is: a mutable array of characters that you can manipulate.