C Programming: Strings

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

January 16, 2026

Topics Covered

What are Strings?

- Sequence of characters
- Stored as array of characters
- Terminated by null character `\0`
- Null terminator marks end of string
- Size = number of characters + 1 (for `\0`)

String Representation:

- String literal: `"Hello"`
- Stored as: `'H', 'e', 'l', 'l', 'o', '\0'`
- Array notation: `char str[6]`
- Last element must be `\0`

Important:

- Always reserve space for null terminator
- String functions rely on `\0`
- Missing `\0` causes undefined behavior

String Declaration and Initialization

Declaration:

```c
char str[size];
char str[] = "Hello";
char *str = "Hello";
```

Initialization Methods:

```c
char str1[6] = "Hello";            // Auto adds \0
char str2[] = "Hello";             // Size = 6
char str3[6] = {'H','e','l','l','o','\0'};
char str4[10] = "Hi";              // Rest = \0
```

Important:

- Size must accommodate \0
- `"Hello"` needs 6 bytes $(5 + 1)$
- Partial init fills rest with \0

Program 1: String Declaration

```c
#include <stdio.h>
int main() {
  char str1[6] = "Hello";
  char str2[] = "World";
  char str3[10] = "Hi";
  int i;
  printf("str1: %s\n", str1);
  printf("str2: %s\n", str2);
  printf("str3: %s\n\n", str3);
  printf("str1 chars: ");
  for (i = 0; i < 6; i++) {
    if (str1[i] == '\0') {
      printf("\\0 ");
    } else {
      printf("%c ", str1[i]);
    }
  }
  printf("\n");
  return 0;
}
```

Output:

```
str1: Hello
str2: World
str3: Hi

str1 chars: H e l l o \0
```

Explanation:

- %s prints until \0
- str1 has explicit size 6
- str2 size inferred as 6
- str3 has extra null chars

Program 2: Character-by-Character Init

```c
#include <stdio.h>
int main() {
  char str[6] = {'H','e','l','l','o','\0'};
  int i;
  printf("String: %s\n\n", str);
  printf("Character array:\n");
  for (i = 0; i < 6; i++) {
    printf("str[%d] = ", i);
    if (str[i] == '\0') {
      printf("'\\0' (null)\n");
    } else {
      printf("'%c' (ASCII %d)\n",
             str[i], str[i]);
    }
  }
  return 0;
}
```

Output:

```
String: Hello

Character array:
str[0] = 'H' (ASCII 72)
str[1] = 'e' (ASCII 101)
str[2] = 'l' (ASCII 108)
str[3] = 'l' (ASCII 108)
str[4] = 'o' (ASCII 111)
str[5] = '\0' (null)
```

Note:

- Shows ASCII values
- Explicit null terminator

Program 3: String Input with scanf

```c
#include <stdio.h>
int main() {
  char name[20];
  char input[] = "John";
  int i;
  printf("Enter name: ");
  for (i = 0; input[i] != '\0'; i++) {
    name[i] = input[i];
  }
  name[i] = '\0';
  printf("%s\n", input);
  printf("\nName entered: %s\n", name);
  printf("Length: %d\n", i);
  return 0;
}
```

Output:

```
Enter name: John

Name entered: John
Length: 4
```

Note:

- `scanf("%s", name)` reads until space
- No & for string name
- Stops at whitespace
- Automatically adds \0

Program 4: String Input with gets (unsafe)

```c
#include <stdio.h>
int main() {
  char line[50];
  char input[] = "Hello World";
  int i;
  printf("Enter line: ");
  for (i = 0; input[i] != '\0'; i++) {
    line[i] = input[i];
  }
  line[i] = '\0';
  printf("%s\n", input);
  printf("\nLine: %s\n", line);
  printf("\nNote: gets() is unsafe!\n");
  printf("Use fgets() instead.\n");
  return 0;
}
```

Output:

```
Enter line: Hello World

Line: Hello World

Note: gets() is unsafe!
Use fgets() instead.
```

Warning:

- `gets()` is deprecated
- No bounds checking
- Can cause buffer overflow
- Use `fgets()` instead

Program 5: String Input with fgets

```c
#include <stdio.h>
#include <string.h>
int main() {
  char line[50] = "Hello World\n";
  int len;
  printf("Enter line: %s", line);
  len = strlen(line);
  if (line[len-1] == '\n') {
    line[len-1] = '\0';
    len--;
  }
  printf("\nLine: %s\n", line);
  printf("Length: %d\n", len);
  printf("\nfgets() is safe!\n");
  return 0;
}
```

Output:

```
Enter line: Hello World

Line: Hello World
Length: 11

fgets() is safe!
```

Note:

- `fgets(str, size, stdin)`
- Reads whole line with spaces
- Includes newline `\n`
- Remove `\n` manually
- Safe - bounds checked

Program 6: strlen - String Length

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "Hello";
  char str2[] = "Programming";
  char str3[] = "";
  printf("str1: \"%s\"\n", str1);
  printf("Length: %lu\n\n",
         strlen(str1));
  printf("str2: \"%s\"\n", str2);
  printf("Length: %lu\n\n",
         strlen(str2));
  printf("str3: \"%s\"\n", str3);
  printf("Length: %lu\n",
         strlen(str3));
  return 0;
}
```

Output:

```
str1: "Hello"
Length: 5

str2: "Programming"
Length: 11

str3: ""
Length: 0
```

Note:

- Counts chars until \0
- Does not include \0
- Empty string has length 0
- Returns size_t type

Program 7: strcpy - String Copy

```c
#include <stdio.h>
#include <string.h>
int main() {
  char src[] = "Hello";
  char dest[20];
  printf("Before copy:\n");
  printf("Source: %s\n", src);
  strcpy(dest, src);
  printf("\nAfter copy:\n");
  printf("Source: %s\n", src);
  printf("Destination: %s\n", dest);
  printf("\nNote: dest must be\n");
  printf("large enough!\n");
  return 0;
}
```

Output:

```
Before copy:
Source: Hello

After copy:
Source: Hello
Destination: Hello

Note: dest must be
large enough!
```

Note:

- Copies src to dest
- Includes \0
- dest must have space
- No bounds checking

Program 8: strcat - String Concatenation

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[20] = "Hello";
  char str2[] = " World";
  printf("str1: \"%s\"\n", str1);
  printf("str2: \"%s\"\n\n", str2);
  strcat(str1, str2);
  printf("After strcat(str1, str2):\n");
  printf("str1: \"%s\"\n", str1);
  printf("str2: \"%s\"\n", str2);
  return 0;
}
```

Output:

```
str1: "Hello"
str2: " World"

After strcat(str1, str2):
str1: "Hello World"
str2: " World"
```

Note:

- Appends str2 to str1
- str1 must have space
- str2 unchanged
- Result in str1

Program 9: strcmp - String Comparison

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "Apple";
  char str2[] = "Apple";
  char str3[] = "Banana";
  int result;
  result = strcmp(str1, str2);
  printf("strcmp(\"%s\", \"%s\") = %d\n",
         str1, str2, result);
  result = strcmp(str1, str3);
  printf("strcmp(\"%s\", \"%s\") = %d\n",
         str1, str3, result);
  result = strcmp(str3, str1);
  printf("strcmp(\"%s\", \"%s\") = %d\n",
         str3, str1, result);
  return 0;
}
```

Output:

```
strcmp("Apple", "Apple") = 0
strcmp("Apple", "Banana") = -1
strcmp("Banana", "Apple") = 1
```

Return Values:

- 0: strings are equal
- <0: str1 <str2 (lexicographic)
- >0: str1 >str2
- Compares ASCII values

Program 10: strchr - Find Character

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello World";
  char *ptr;
  printf("String: %s\n\n", str);
  ptr = strchr(str, 'o');
  if (ptr != NULL) {
    printf("First 'o' found at: %ld\n",
           ptr - str);
    printf("Substring: %s\n", ptr);
  }
  ptr = strchr(str, 'x');
  if (ptr == NULL) {
    printf("\n'x' not found\n");
  }
  return 0;
}
```

Output:

```
String: Hello World

First 'o' found at: 4
Substring: o World

'x' not found
```

Note:

- Returns pointer to first match
- NULL if not found
- Can calculate index
- Pointer arithmetic

Program 11: strstr - Find Substring

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello World";
  char *ptr;
  printf("String: %s\n\n", str);
  ptr = strstr(str, "World");
  if (ptr != NULL) {
    printf("'World' found at: %ld\n",
           ptr - str);
    printf("Substring: %s\n", ptr);
  }
  ptr = strstr(str, "xyz");
  if (ptr == NULL) {
    printf("\n'xyz' not found\n");
  }
  return 0;
}
```

Output:

```
String: Hello World

'World' found at: 6
Substring: World

'xyz' not found
```

Note:

- Finds substring in string
- Returns pointer to match
- NULL if not found
- Case-sensitive

Program 12: Manual String Length

```c
#include <stdio.h>
int main() {
  char str[] = "Programming";
  int length = 0;
  int i;
  for (i = 0; str[i] != '\0'; i++) {
    length++;
  }
  printf("String: %s\n", str);
  printf("Length: %d\n", length);
  printf("\nCounting manually:\n");
  for (i = 0; str[i] != '\0'; i++) {
    printf("str[%d] = '%c'\n", i, str[i]);
  }
  return 0;
}
```

Output:

```
String: Programming
Length: 11

Counting manually:
str[0] = 'P'
str[1] = 'r'
str[2] = 'o'
str[3] = 'g'
str[4] = 'r'
str[5] = 'a'
str[6] = 'm'
str[7] = 'm'
str[8] = 'i'
str[9] = 'n'
str[10] = 'g'
```

Program 13: Manual String Copy

```c
#include <stdio.h>
int main() {
  char src[] = "Hello";
  char dest[20];
  int i;
  printf("Source: %s\n", src);
  for (i = 0; src[i] != '\0'; i++) {
    dest[i] = src[i];
  }
  dest[i] = '\0';
  printf("Destination: %s\n\n", dest);
  printf("Manual copy complete!\n");
  return 0;
}
```

Output:

```
Source: Hello
Destination: Hello

Manual copy complete!
```

Logic:

- Copy char by char
- Loop until \0
- Don't forget to add \0
- Same as strcpy

Program 14: Reverse a String

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello";
  int len = strlen(str);
  int i;
  char temp;
  printf("Original: %s\n", str);
  for (i = 0; i < len/2; i++) {
    temp = str[i];
    str[i] = str[len-1-i];
    str[len-1-i] = temp;
  }
  printf("Reversed: %s\n", str);
  return 0;
}
```

Output:

```
Original: Hello
Reversed: olleH
```

Logic:

- Swap first and last
- Move toward center
- Loop len/2 times
- In-place reversal

Program 15: Check Palindrome

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "madam";
  char str2[] = "hello";
  int len, i, palindrome;
  len = strlen(str1);
  palindrome = 1;
  for (i = 0; i < len/2; i++) {
    if (str1[i] != str1[len-1-i]) {
      palindrome = 0;
      break;
    }
  }
  printf("%s: %s\n", str1,
         palindrome ? "Palindrome" :
                      "Not palindrome");
  len = strlen(str2);
  palindrome = 1;
  for (i = 0; i < len/2; i++) {
    if (str2[i] != str2[len-1-i]) {
      palindrome = 0;
      break;
    }
  }
  printf("%s: %s\n", str2,
         palindrome ? "Palindrome" :
                      "Not palindrome");
  return 0;
}
```

Output:

```
madam: Palindrome
hello: Not palindrome
```

Logic:

- Compare first and last
- Move toward center
- If any mismatch, not palindrome
- Check half the string

Program 16: Count Vowels and Consonants

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello World";
  int vowels = 0, consonants = 0;
  int i;
  char ch;
  printf("String: %s\n\n", str);
  for (i = 0; str[i] != '\0'; i++) {
    ch = str[i];
    if (ch >= 'A' && ch <= 'Z') {
      ch = ch + 32;
    }
    if (ch == 'a' || ch == 'e' ||
        ch == 'i' || ch == 'o' ||
        ch == 'u') {
      vowels++;
    } else if (ch >= 'a' && ch <= 'z') {
      consonants++;
    }
  }
  printf("Vowels: %d\n", vowels);
  printf("Consonants: %d\n", consonants);
  return 0;
}
```

Output:

```
String: Hello World

Vowels: 3
Consonants: 7
```

Logic:

- Convert to lowercase
- Check if vowel
- Else check if letter
- Ignore spaces/punctuation

Program 17: Convert to Uppercase

```c
#include <stdio.h>
int main() {
  char str[] = "Hello World";
  int i;
  printf("Original: %s\n", str);
  for (i = 0; str[i] != '\0'; i++) {
    if (str[i] >= 'a' && str[i] <= 'z') {
      str[i] = str[i] - 32;
    }
  }
  printf("Uppercase: %s\n", str);
  return 0;
}
```

Output:

```
Original: Hello World
Uppercase: HELLO WORLD
```

Logic:

- Check if lowercase letter
- Subtract 32 from ASCII
- 'a' = 97, 'A' = 65
- Difference = 32

Program 18: Convert to Lowercase

```c
#include <stdio.h>
int main() {
  char str[] = "HELLO WORLD";
  int i;
  printf("Original: %s\n", str);
  for (i = 0; str[i] != '\0'; i++) {
    if (str[i] >= 'A' && str[i] <= 'Z') {
      str[i] = str[i] + 32;
    }
  }
  printf("Lowercase: %s\n", str);
  return 0;
}
```

Output:

```
Original: HELLO WORLD
Lowercase: hello world
```

Logic:

- Check if uppercase letter
- Add 32 to ASCII
- 'A' + 32 = 'a'
- Simple conversion

Program 19: Count Words in String

```c
#include <stdio.h>
int main() {
  char str[] = "Hello World from C";
  int i, words = 0;
  int inWord = 0;
  printf("String: %s\n\n", str);
  for (i = 0; str[i] != '\0'; i++) {
    if (str[i] == ' ' || str[i] == '\t' ||
        str[i] == '\n') {
      inWord = 0;
    } else if (inWord == 0) {
      inWord = 1;
      words++;
    }
  }
  printf("Word count: %d\n", words);
  return 0;
}
```

Output:

```
String: Hello World from C

Word count: 4
```

Logic:

- Track if inside word
- Space/tab ends word
- Non-space starts new word
- Increment on word start

Program 20: Remove Spaces

```c
#include <stdio.h>
int main() {
  char str[] = "Hello   World";
  int i, j = 0;
  printf("Original: \"%s\"\n", str);
  for (i = 0; str[i] != '\0'; i++) {
    if (str[i] != ' ') {
      str[j] = str[i];
      j++;
    }
  }
  str[j] = '\0';
  printf("No spaces: \"%s\"\n", str);
  return 0;
}
```

Output:

```
Original: "Hello   World"
No spaces: "HelloWorld"
```

Logic:

- Two indices: i and j
- i scans entire string
- j tracks write position
- Skip spaces, copy others

Program 21: Common String Mistakes

```c
#include <stdio.h>
int main() {
  char str1[6] = "Hello";
  char str2[6];
  printf("Mistake 1:\n");
  printf("str2 = str1 is WRONG!\n");
  printf("Use strcpy instead\n\n");
  printf("Mistake 2:\n");
  printf("if(str1==str2) is WRONG!\n");
  printf("Use strcmp instead\n\n");
  printf("Mistake 3:\n");
  printf("Forgetting \\0 causes\n");
  printf("undefined behavior\n\n");
  printf("Mistake 4:\n");
  printf("Buffer overflow if\n");
  printf("dest too small\n");
  return 0;
}
```

Output:

```
Mistake 1:
str2 = str1 is WRONG!
Use strcpy instead

Mistake 2:
if(str1==str2) is WRONG!
Use strcmp instead

Mistake 3:
Forgetting \0 causes
undefined behavior

Mistake 4:
Buffer overflow if
dest too small
```

Strings - Summary

Key Points:

- String $=$ character array $+$ \0
- Always reserve space for null terminator
- %s for string I/O
- Cannot assign strings with =
- Cannot compare with ==
- Use string.h library functions
- `strlen()`, `strcpy()`, `strcat()`, `strcmp()`
- `fgets()` safer than `gets()`
- Manual operations use loops

String Library Functions

Function	**Purpose**
`strlen(s)`	Length of string
`strcpy(dest, src)`	Copy string
`strcat(dest, src)`	Concatenate strings
`strcmp(s1, s2)`	Compare strings
`strchr(s, c)`	Find character
`strstr(s1, s2)`	Find substring
`strupr(s)`	Convert to uppercase
`strlwr(s)`	Convert to lowercase

Best Practices

1. **Always null-terminate** strings
2. **Check buffer sizes** before copying
3. **Use fgets()** instead of gets()
4. **Use strncpy()** instead of strcpy() for safety
5. **Validate input** length
6. **Initialize arrays** before use
7. **Use string.h functions** when available
8. **Avoid buffer overflow** - check bounds
9. **Remember** \0 when calculating size
10. **Handle empty strings** (length 0)

Common Mistakes

1. **Forgetting** \0: Causes undefined behavior
2. **Buffer overflow**: Destination too small
3. **Using = for assignment**: Use strcpy()
4. **Using == for comparison**: Use strcmp()
5. **Off-by-one errors**: Size vs length
6. **Using gets()**: Unsafe, use fgets()
7. **Not checking NULL**: After strchr, strstr
8. **Modifying string literals**: Undefined behavior
9. **scanf with spaces**: Use fgets() for lines
10. **Not removing newline**: From fgets() input

Practice Exercises

Try these programs:

1. Find frequency of each character
2. Remove duplicate characters
3. Check if two strings are anagrams
4. Find longest word in sentence
5. Replace all occurrences of character
6. Trim leading and trailing spaces
7. Check if string contains only digits
8. Convert string to integer (atoi)
9. Split string by delimiter
10. Find all permutations of string