Strings in C:

Introduction to Strings in C

In C, a **string** is a sequence of characters stored in a contiguous block of memory and terminated by a null character (\0). The null character signifies the end of the string, enabling functions to determine the string's length and process its contents.

- Strings are essentially arrays of characters.
- The standard library <string.h> provides various built-in functions for manipulating strings.

Declaring and Initializing Strings

1. Static Declaration (Implicit Null Terminator)

char str[] = "Hello";

- Here, the size of str is 6 (5 characters + 1 null character).

2. Explicit Size Declaration

char str[6] = "Hello";

- The size is explicitly defined, and the null character is automatically added.

3. Character-by-Character Initialization

char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};

4. Using Pointers

char *str = "Hello";

- The string is stored in a read-only section of memory. Modifying it may cause undefined behavior.

Input and Output of Strings

1. Using scanf

- Reads a string until whitespace.

char str[100];

scanf("%s", str); // Input: "Hello World" -> Stores: "Hello"

2. Using gets *(deprecated)*:

- Reads an entire line including spaces.

char str[100];

gets(str); // Input: "Hello World" -> Stores: "Hello World"

3. Using fgets

- A safer alternative to gets.

fgets(str, sizeof(str), stdin); // Input: "Hello World"

4. Output Functions

- **printf**:

printf("%s", str);

- **puts**:

puts(str); // Automatically appends a newline

String Manipulation Functions in C

C provides a set of string manipulation functions in the <string.h> library. Below are some commonly used functions:

1. Finding Length: strlen()

- Returns the length of the string (excluding the null character).

#include <string.h>

size_t strlen(const char *str);

Example:

char str[] = "Hello";

printf("Length: %ld\n", strlen(str)); // Output: 5

---

2. Copying Strings: strcpy()

- Copies the contents of one string into another.

char *strcpy(char *dest, const char *src);

Example:

char src[] = "Hello";

char dest[20];

strcpy(dest, src);

printf("%s\n", dest); // Output: Hello

---

3. Concatenating Strings: strcat()

- Appends one string to the end of another.

```
char *strcat(char *dest, const char *src);
```

Example:

```
char str1[20] = "Hello, ";
char str2[] = "World!";
strcat(str1, str2);
printf("%s\n", str1);  // Output: Hello, World!
```

---

4. Comparing Strings: strcmp()

- Compares two strings lexicographically.

```
int strcmp(const char *str1, const char *str2);
```

Return Values:

- 0: Strings are equal.
- < 0: First string is less than the second.
- > 0: First string is greater than the second.

Example:

```
char str1[] = "Hello";
char str2[] = "World";
if (strcmp(str1, str2) == 0) {
   printf("Strings are equal.\n");
} else {
   printf("Strings are not equal.\n");
}
```

---

5. Reversing Strings: strrev() ***(Non-standard)***

- Reverses a string (available in Turbo C or custom implementations).

```
char str[] = "Hello";
printf("%s\n", strrev(str));  // Output: olleH
```

---

6. Finding Substring: strstr()

- Finds the first occurrence of a substring in a string.

```
char *strstr(const char *haystack, const char *needle);
```

Example:

```
char str[] = "Hello, World!";
char *result = strstr(str, "World");
if (result) {
   printf("Found: %s\n", result);  // Output: World!
}
```

---

7. Lowercase Conversion: strlwr() ***(Non-standard)***

- Converts all characters of a string to lowercase.

```
char str[] = "HELLO";
printf("%s\n", strlwr(str));  // Output: hello
```

---

8. Uppercase Conversion: strupr() ***(Non-standard)***

- Converts all characters of a string to uppercase.

```
char str[] = "hello";
printf("%s\n", strupr(str));  // Output: HELLO
```

Dynamic Strings

Strings can also be dynamically allocated using memory management functions like malloc and free.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
   char *str;
   str = (char *)malloc(50 * sizeof(char));  // Allocate memory for 50 characters
   if (str == NULL) {
      printf("Memory allocation failed\n");
      return 1;
   }
```

```
    strcpy(str, "Dynamic String");
    printf("%s\n", str);

    free(str);  // Free allocated memory
    return 0;
}
```

Multidimensional Strings (Array of Strings)

You can create arrays of strings to store multiple words or sentences.

Example:

```
#include <stdio.h>

int main() {
    char strings[3][20] = {"Hello", "World", "C Programming"};

    for (int i = 0; i < 3; i++) {
        printf("%s\n", strings[i]);
    }
    return 0;
}
```

Common String-Related Problems

Problem 1: Palindrome Check

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[100], rev[100];
    printf("Enter a string: ");
    scanf("%s", str);
```

```
    strcpy(rev, str);
    strrev(rev);  // Reverse the string

    if (strcmp(str, rev) == 0) {
        printf("Palindrome\n");
    } else {
        printf("Not a palindrome\n");
    }
    return 0;
}
```

Problem 2: Count Vowels in a String

```
#include <stdio.h>

int main() {
    char str[100];
    int vowels = 0;
    printf("Enter a string: ");
    scanf("%s", str);

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' ||
            str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U') {
            vowels++;
        }
    }

    printf("Number of vowels: %d\n", vowels);
    return 0;
}
```

Summary

1. Strings are arrays of characters terminated by \0.
2. String manipulation is facilitated by <string.h>.
3. Common operations include copying, concatenation, comparison, and substring search.
4. Strings can be static or dynamically allocated.

What is a String in C?

- A **string** is a sequence of characters terminated by a special character called the null character (\0).
- Strings in C are represented as arrays of characters and are stored in contiguous memory locations.

Declaration of Strings

1. **Static Declaration:**

```
char str[] = "Hello";
```

2. **Explicit Declaration:**

```
char str[6] = "Hello";
```

3. **Using a Pointer:**

```
char *str = "Hello";
```

Input and Output of Strings

1. **Input using scanf**:

```
char str[100];
scanf("%s", str);  // Reads until whitespace
```

2. **Input using gets (Deprecated)**:

```
char str[100];
gets(str);  // Reads an entire line
```

3. **Input using fgets (Safe)**:

```
fgets(str, sizeof(str), stdin);
```

4. **Output using printf**:

```
printf("%s", str);
```

5. **Output using puts**:

```
puts(str);  // Adds a newline automatically
```

String Functions in C

C provides a library <string.h> that contains many useful string manipulation functions. Below is the list of functions with syntax, examples, and outputs.

---

1. strlen() – String Length

- **Purpose**: Finds the length of a string (excluding the null character).
- **Syntax**:

```
size_t strlen(const char *str);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello, World!";
    printf("Length of string: %ld\n", strlen(str));
    return 0;
}
```

- **Output**:

```
Length of string: 13
```

---

2. strcpy() – String Copy

- **Purpose**: Copies one string into another.
- **Syntax**:

```
char *strcpy(char *dest, const char *src);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char src[] = "C Programming";
    char dest[50];
```

```
    strcpy(dest, src);
    printf("Copied string: %s\n", dest);
    return 0;
}
```

- **Output**:

```
Copied string: C Programming
```

---

3. strcat() – String Concatenation

- **Purpose**: Appends one string to the end of another.
- **Syntax**:

```
char *strcat(char *dest, const char *src);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[50] = "Hello, ";
    char str2[] = "World!";
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);
    return 0;
}
```

- **Output**:

```
Concatenated string: Hello, World!
```

---

4. strcmp() – String Comparison

- **Purpose**: Compares two strings lexicographically.
- **Syntax**:

```
int strcmp(const char *str1, const char *str2);
```

- **Return Values**:
    - 0: Strings are equal.

- < 0: First string is less than the second.
- > 0: First string is greater than the second.

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[] = "Hello";
    char str2[] = "World";

    if (strcmp(str1, str2) == 0) {
        printf("Strings are equal.\n");
    } else {
        printf("Strings are not equal.\n");
    }
    return 0;
}
```

- **Output**:

```
sql
Strings are not equal.
```

---

5. strrev() – String Reverse (Non-standard)

- **Purpose**: Reverses the string.
- **Syntax**:

```
char *strrev(char *str);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello";
```

```
    printf("Original string: %s\n", str);
    printf("Reversed string: %s\n", strrev(str));
    return 0;
}
```

- **Output**:

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

---

6. strlwr() – Convert to Lowercase (Non-standard)

- **Purpose**: Converts all characters in a string to lowercase.
- **Syntax**:

```
char *strlwr(char *str);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "HELLO";
    printf("Lowercase string: %s\n", strlwr(str));
    return 0;
}
```

- **Output**:

```
Lowercase string: hello
```

---

7. strupr() – Convert to Uppercase (Non-standard)

- **Purpose**: Converts all characters in a string to uppercase.
- **Syntax**:

```
char *strupr(char *str);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>
```

```
int main() {
    char str[] = "hello";
    printf("Uppercase string: %s\n", strupr(str));
    return 0;
}
```

- **Output**:

```
Uppercase string: HELLO
```

---

8. strstr() – Find Substring

- **Purpose**: Finds the first occurrence of a substring in a string.
- **Syntax**:

```
char *strstr(const char *haystack, const char *needle);
```

- **Example**:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello, World!";
    char substr[] = "World";
    char *result = strstr(str, substr);

    if (result) {
        printf("Substring found: %s\n", result);
    } else {
        printf("Substring not found.\n");
    }
    return 0;
}
```

- **Output**:

```
yaml
```

Substring found: World!

Working with Dynamic Strings

Strings can also be dynamically allocated using memory management functions.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
   char *str;
   str = (char *)malloc(50 * sizeof(char));  // Allocate memory
   if (str == NULL) {
      printf("Memory allocation failed\n");
      return 1;
   }

   strcpy(str, "Dynamic String");
   printf("%s\n", str);

   free(str);  // Free allocated memory
   return 0;
}
```

Output:

mathematica

Dynamic String

Summary Table of String Functions

Function	Description	Example
strlen	Finds string length.	strlen("Hello")

Function	Description	Example
strcpy	Copies one string to another.	strcpy(dest, src)
strcat	Concatenates two strings.	strcat(dest, src)
strcmp	Compares two strings.	strcmp(str1, str2)
strstr	Finds a substring.	strstr(str, substr)
strrev	Reverses a string (non-standard).	strrev(str)
strlwr	Converts to lowercase (non-standard)	strlwr(str)
strupr	Converts to uppercase (non-standard)	strupr(str)