SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AIML

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT III – ARRAYS AND STRINGS

TOPIC – C STRINGS – STRING OPERATIONS

STRINGS - INTRODUCTION

Strings in C are robust data structures for storing information. They can be implemented with either a one-dimensional array of characters or pointers. There are many ways to initialize a string in C. These strings can be used to read lines of text and can also be passed to functions. Strings are an array of characters that terminate with a null character '\0'. The difference between a character array and a string is that, unlike the character array, the string ends with a null character. There are various built-in string functions in the C programming language.

WHAT IS A STRING?

A string is a collection of characters (i.e., letters, numerals, symbols, and punctuation marks) in a linear sequence. In C, a string is a sequence of characters concluded with a NULL character '\0'.

For example: char str[] = “C program.\0";

Like many other programming languages, strings in C are enclosed within ***double quotes(" ")***, whereas characters are enclosed within ***single quotes(' ')***. When the compiler finds a sequence of characters enclosed within the double quotation marks, it adds a null character (\0) at the end by default.

Thus, this is how the string is stored:

C		**p**	**r**	**o**	**g**	**r**	**a**	**m**	**.**	**\0**

DECLARATION OF STRING IN C

A String in C is an array with character as a data type. C does not directly support string as a data type, as seen in other programming languages like C++. Hence, character arrays must be used to display a string in C. The general syntax of declaring a string in C is as follows:

char variable[array_size];

Thus, the classic declaration can be made as follows:

```
char str[5];
char str2[50];
```

It is vital to note that we should always account for an extra space used by the null character(\0).

INITIALIZING A STRING

There are four methods of initializing a string in C:

1. Assigning a string literal with size

We can directly assign a string literal to a character array keeping in mind to keep the array size at least one more than the length of the string literal that will be assigned to it.

Note While setting the initial size, we should always account for one extra space used by the null character. If we want to store a string of size **n**, we should set the initial size to be **n+1**.

For example: **char str[6] = “Hello";**

The string length here is 5, but we have kept the size to be 6 to account for the Null character. The compiler adds the Null character(\0) at the end automatically. If the array cannot accommodate the entire string, it only takes characters based on its space.

For example:

char str[3] = "Scaler."; printf("%s",str);

Output: **Sca**

INITIALIZING A STRING

2. Assigning a string literal without size

It is also possible to directly assign a string literal to a character array without any size. The size gets determined automatically by the compiler at compile time.

char str[] = “HELLO.";

The critical thing to remember is that the name of the string, here "*str*" acts as a pointer because it is an array.

3. Assigning character by character with size

We can also assign a string character by character. However, it is essential to set the end character as '\0'. For example:

char str[6] = {‘H', ‘E', ‘L', ‘L', ‘O', '\0'};

4. Assigning character by character without size

Like assigning directly without size, we also assign character by character with the Null Character at the end. The compiler will determine the size of the string automatically.

char str[] = {‘H', ‘E', ‘L', ‘L', ‘O', '\0'};

ASSIGNING VALUE TO STRINGS

There are two methods to assign value to the strings.

1. Assign the value to a character array while initializing.

char str[6] = “Hello”;

2. We can use the strcpy() function to copy the value we want to assign to the character array. The syntax for **strcpy()** is as follows:

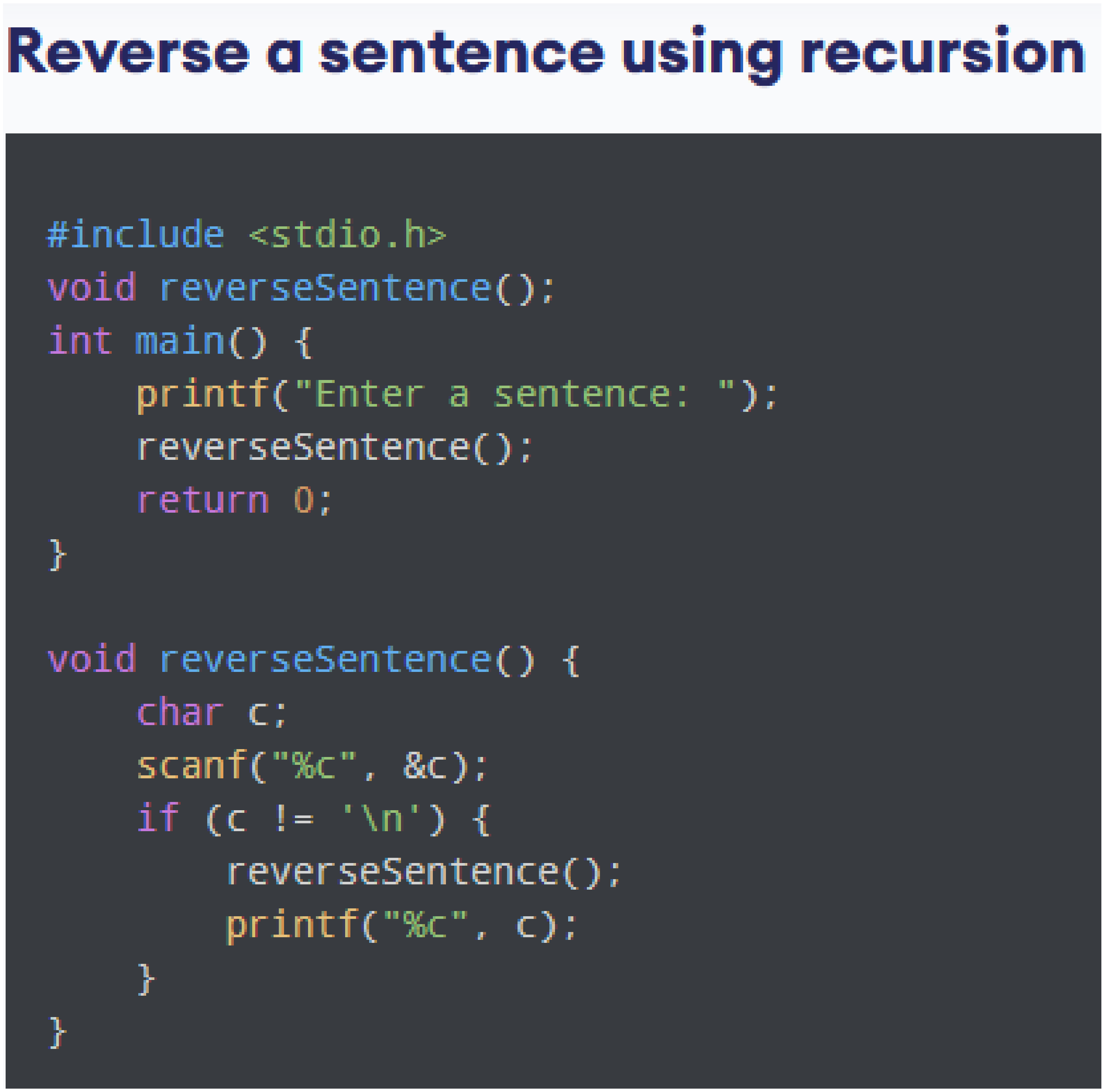
strcpy(char* destination, const char* source);

It copies the string pointed by the source (including the null character) to the destination.

For example:

char str[20]; strcpy(str,"Strings.");

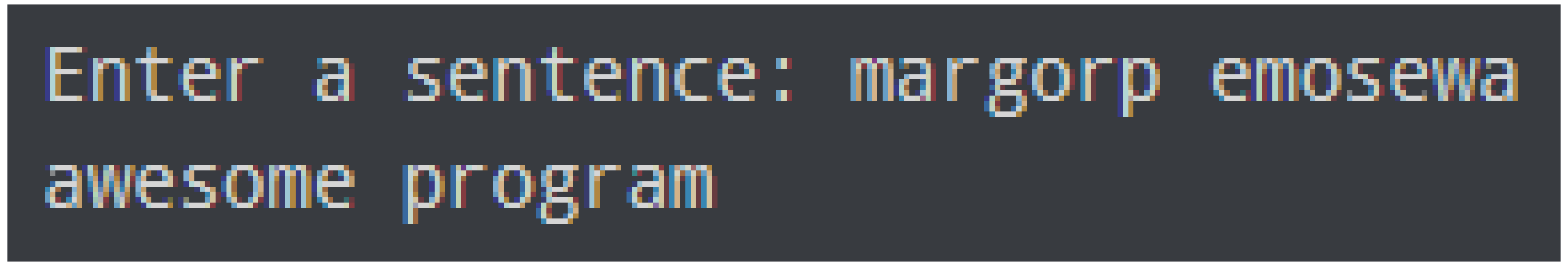
EXAMPLE



Reverse a sentence using recursion

```c
#include <stdio.h>
void reverseSentence();
int main() {
    printf("Enter a sentence: ");
    reverseSentence();
    return 0;
}

void reverseSentence() {
    char c;
    scanf("%c", &c);
    if (c != '\n') {
        reverseSentence();
        printf("%c", c);
    }
}
```



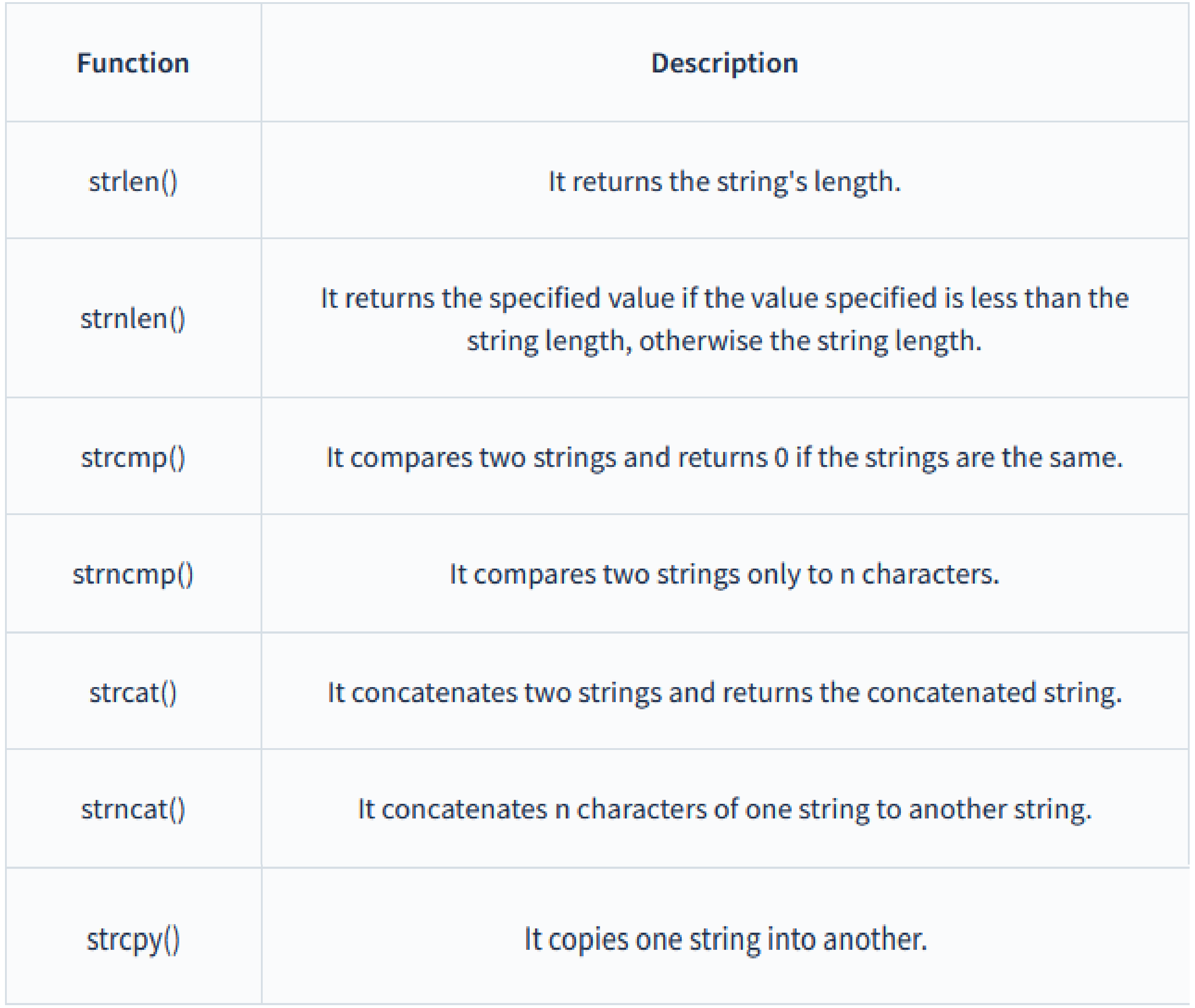
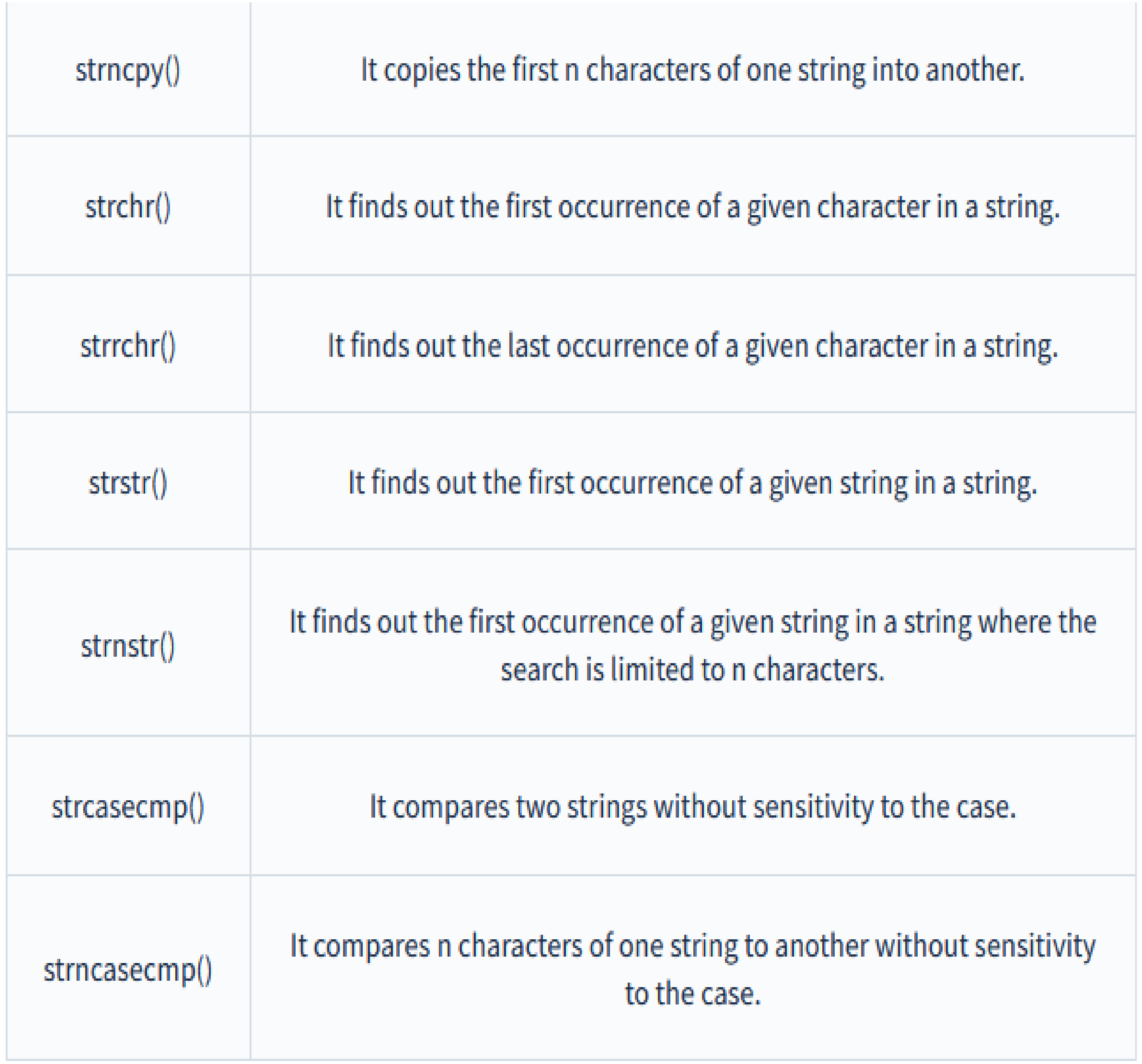
```
Enter a sentence: margorp emosewa
awesome program
```

STRING OPERATIONS

We're often required to modify the strings and perform several operations on them according to our needs. If we want to get the length of the string, we could run a loop and calculate its length, but it is not the best way in case of complex problems. Hence, string functions are used to make our code efficient and straightforward as they are pre-written so we can use them directly.

The string handling functions are defined in the header file string.h. This header file must be included in the C program to use the string handling functions.

STRING FUNCTIONS

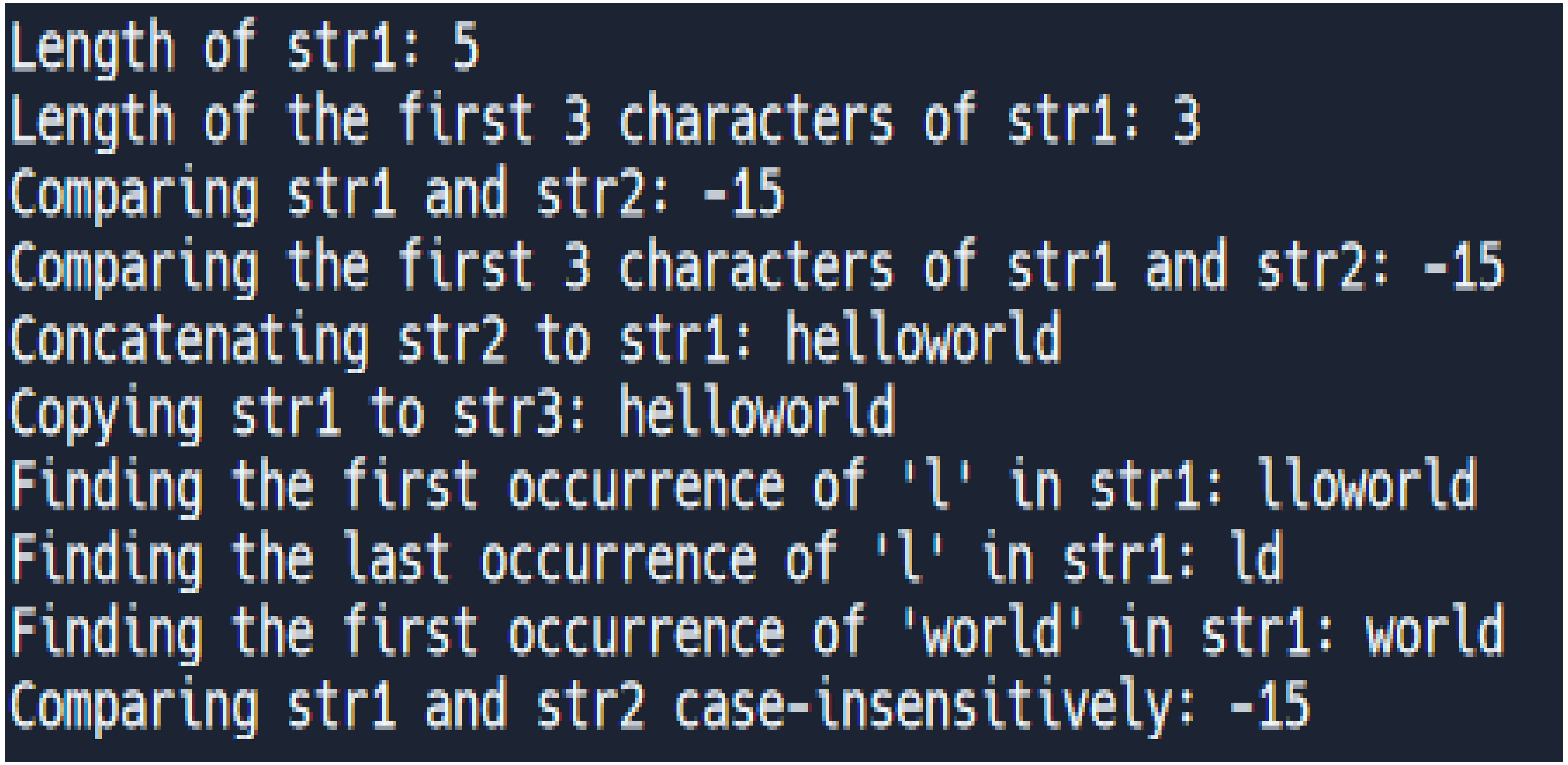



Function	Description
strlen()	It returns the string's length.
strnlen()	It returns the specified value if the value specified is less than the string length, otherwise the string length.
strcmp()	It compares two strings and returns 0 if the strings are the same.
strncmp()	It compares two strings only to n characters.
strcat()	It concatenates two strings and returns the concatenated string.
strncat()	It concatenates n characters of one string to another string.
strcpy()	It copies one string into another.
strncpy()	It copies the first n characters of one string into another.
strchr()	It finds out the first occurrence of a given character in a string.
strrchr()	It finds out the last occurrence of a given character in a string.
strstr()	It finds out the first occurrence of a given string in a string.
strnstr()	It finds out the first occurrence of a given string in a string where the search is limited to n characters.
strcasecmp()	It compares two strings without sensitivity to the case.
strncasecmp()	It compares n characters of one string to another without sensitivity to the case.

EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50] = "hello";
    char str2[] = "world";
    char str3[100];
    // strlen()
    printf("Length of str1: %ld\n", strlen(str1));
    // strnlen()
    printf("Length of the first 3 characters of str1: %ld\n",
strnlen(str1, 3));
    // strcmp()
    printf("Comparing str1 and str2: %d\n", strcmp(str1, str2));
    // strncmp()
    printf("Comparing the first 3 characters of str1 and str2:
%d\n", strncmp(str1, str2, 3));
    // strcat()
    strcat(str1, str2);
    printf("Concatenating str2 to str1: %s\n", str1);
// strcpy()
    strcpy(str3, str1);
    printf("Copying str1 to str3: %s\n", str3);
    // strchr()
    printf("Finding the first occurrence of 'l' in str1: %s\n",
strchr(str1, 'l'));
    // strrchr()
    printf("Finding the last occurrence of 'l' in str1: %s\n",
strrchr(str1, 'l'));
    // strstr()
    printf("Finding the first occurrence of 'world' in str1: %s\n",
strstr(str1, "world"));
    // strcasecmp()
    printf("Comparing str1 and str2 case-insensitively: %d\n",
strcasecmp(str1, str2));
    //strrev()
    printf("reverse of str1 is : %s\n", strrev(str1));
   return 0;
}
```

OUTPUT



```
Length of str1: 5
Length of the first 3 characters of str1: 3
Comparing str1 and str2: -15
Comparing the first 3 characters of str1 and str2: -15
Concatenating str2 to str1: helloworld
Copying str1 to str3: helloworld
Finding the first occurrence of 'l' in str1: lloworld
Finding the last occurrence of 'l' in str1: ld
Finding the first occurrence of 'world' in str1: world
Comparing str1 and str2 case-insensitively: -15
```