MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1112 (Problem Solving Using C)

Block 2: Arrays and Strings

Unit 6: Character Strings Handling

Structure	
6.0	Introduction
6.1	Objectives
6.2	Strings Declaration and Initialization of Strings
6.3	Built-In String Function and Applications
6.4	Other String Functions
6.5	Standard I/O Functions of String
6.6	Summary
6.7	Questions
6.8	References

6.0 Introduction

Character strings are a fundamental aspect of programming in C, providing a versatile way to handle text data. In C, a string is defined as an array of characters terminated by a null character ('\0'). This null terminator is crucial as it signals the end of the string, allowing functions to determine its length and process its contents correctly. Handling character strings in C involves various operations such as creating, copying, concatenating, comparing, and searching strings. The C Standard Library, specifically the **string.h** header, provides a comprehensive set of functions to perform these operations efficiently.

Effective string handling is essential for a wide range of applications, from simple text processing to complex data manipulation. Mastery of string functions enables programmers to build robust and efficient programs. For instance, functions like **strcpy, strcat,** and **strcmp** are frequently used for basic string operations, while safer alternatives like **fgets** ensure secure input handling by preventing buffer overflows. Understanding these functions not only aids in developing reliable code but also enhances the programmer's ability to troubleshoot and optimize string-related tasks in their applications.

6.1 Objectives

After completing this unit, you will be able to understand,

- Learn the fundamental concepts of character strings in C, including how strings are represented as arrays of characters terminated by a null character ('\0').

- Gain proficiency in declaring and initializing strings in various ways, including using character arrays and pointers to string literals.
- Master the use of the C Standard Library functions for string manipulation, such as strlen, strcpy, strncpy, strcat, strncat, strcmp, and strncmp.
- Understand the importance of safe string handling practices, particularly avoiding deprecated functions like gets and using safer alternatives such as fgets for input.
- Learn to utilize functions like strchr and strstr for finding characters and substrings within strings.
- Become familiar with functions like puts for writing strings to standard output and understanding their behavior.
- Develop the ability to incorporate string handling functions into larger programs, performing tasks such as copying, concatenation, comparison, and searching in real-world scenarios.

6.2 Strings Declaration and Initialization of Strings

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null. The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

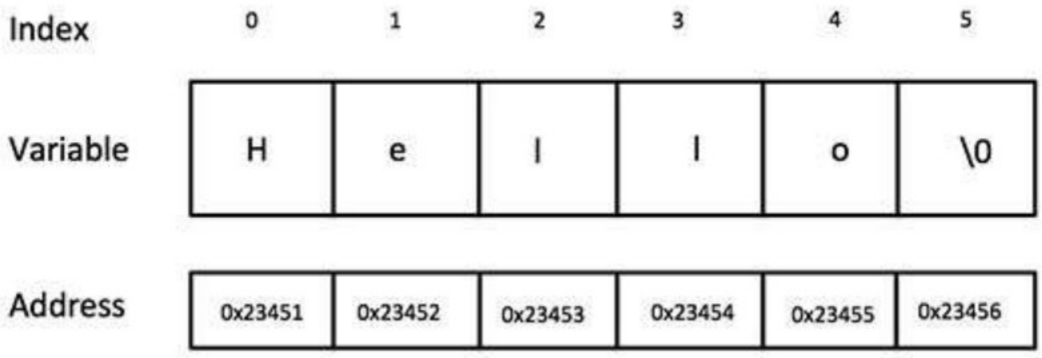
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

The rule of array initialization, they can write the statement as:

char greeting[] = "Hello";

Strings in C are group of characters, digits, and symbols enclosed in quotation marks or simply we can say the string is declared as a “character array”. The end of the string is marked with a special character, the ‘\0’.

The memory of the string in C/C++:



The null character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array.

string:

#include

int main ()

{ char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

```
printf("Greeting message: %s\n", greeting );
return 0;
}
```

Result:

Greeting message: Hello

Key Characteristics

1. **Null Termination:**
   - All strings in C are null-terminated, which means the last character in a string is always '\0'. This null character indicates the end of the string, enabling string handling functions to determine where the string ends.
2. **Fixed Size in Arrays:**
   - When strings are stored in arrays, the size of the array must be large enough to hold all characters of the string plus the null terminator. If the size of the array is not specified, it will be automatically determined based on the initializer.

Initialization of Strings:

A string in C is simply a sequence of characters. To declare a string, specify the data type as char and place the number of characters in the array in square brackets after the string name.

The syntax is:

char string-name[size];

Example:

char name[20];

char address[25];

char city[15];

The string can be initialized as:

char name[8] = {‘P’, ‘R’, ‘O’, ‘G’, ‘R’, ‘A’, ‘M’, ‘\0’};

Each character of string occupies 1 byte of memory (on 16 bit computing). The size of character is machine dependent, and varies from 16 bit computers to 64 bit computers. The characters of strings are stored in the contiguous (adjacent) memory locations.

1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
P	R	O	G	R	A	M	\0
1001	1002	1003	1004	1005	1006	1007	1008

The C compiler inserts the NULL (\0) character automatically at the end of the string. So initialization of the NULL character is not essential.

String constants have double quote marks around them, and can be assigned to char pointers. Alternatively, you can assign a string constant to a char array - either with no size specified, or you can specify a size, but don’t forget to leave a space for the null character! Suppose you create the following two code fragments and run them:

```
/* Fragment 1 */
```

```
{
char *s;
s=hello";
printf("%s\n",s);
}
/* Fragment 2 */
{
char s[100];
strcpy(s, " hello");
printf("%s\n",s);
}
```

These two fragments produce the same output, but their internal behaviour is quite different. In fragment 2, you cannot say s = "hello";. To understand the differences, you have to understand how the string constant table works in C.

Example:

Write a program to read a name from the keyboard and display message Hello onto the monitor Program.

```
/*Program that reads the name and display the hello along with your name*/
#include main()
{
char name[10];
printf("\nEnter Your Name : ");
scanf("%s", name);
printf("Hello %s\n", name);
}
```

OUTPUT:

```
Enter Your Name:  Raghav
  Hello Raghav
```

6.3 Built-In String Function and Applications

The header file contains some string manipulation functions. The following is a list of the common string managing functions in C.

Strlen Function

The strlen function returns the length of a string. It takes the string name as argument. The syntax is as follows:

n = strlen (str);

where str is name of the string and n is the length of the string, returned by strlen function.

Strcpy Function

In C, you cannot simply assign one character array to another. You have to copy element by element. The string library contains a function called strcpy for this purpose. The strcpy function is used to copy one string to another.

The syntax is:

strcpy(str1, str2);

where str1, str2 are two strings. The content of string str2 is copied on to string str1.

Strcmp Function

The strcmp function in the string library function which compares two strings, character by character and stops comparison when there is a difference in the ASCII value or the end of any one string and returns ASCII difference of the characters that is integer. If the return value zero means the two strings are equal, a negative value means that first is less than second, and a positive value means first is greater than second.

The syntax is:

n = strcmp(str1, str2);

where str1 and str2 are two strings to be compared and n is returned value of differed characters.

Strcat Function

The strcat function is used to join one string to another. It takes two strings as arguments; the characters of the second string will be appended to the first string.

The syntax is:

strcat(str1, str2);

where str1 and str2 are two string arguments, string str2 is appended to string str1.

Strlwr Function

The strlwr function converts upper case characters of string to lower case characters.

The syntax is:

strlwr(str1);

where str1 is string to be converted into lower case characters.

Strrev Function

The strrev funtion reverses the given string.

The syntax is:

strrev(str);

where string str will be reversed.

Strspn Function

The strspn function returns the position of the string, where first string mismatches with second string.

The syntax is:

n = strspn (first, second);

Where first and second are two strings to be compared, n is the number of character from which first string does not match with second string.

6.4 Other String Functions

Strncpy Function

The strncpy function same as strcpy. It copies characters of one string to another string up to the specified length.

The syntax is:

```
strncpy(str1, str2, 10);
```

where str1 and str2 are two strings. The 10 characters of string str2 are copied onto string str1.

Stricmp Function

The stricmp function is same as strcmp, except it compares two strings ignoring the case (lower and upper case).

The syntax is:

```
n = stricmp(str1, str2);
```

strncmp function The strncmp function is same as strcmp, except it compares two strings up to a specified length.

The syntax is:

```
n = strncmp(str1, str2, 10);
```

where 10 characters of str1 and str2 are compared and n is returned value of differed characters.

Strchr Function

The strchr funtion takes two arguments (the string and the character whose address is to be specified) and returns the address of first occurrence of the character in the given string.

The syntax is:

```
cp = strchr (str, c);
```

Where str is string and c is character and cp is character pointer.

String Handling Program

Here's a simple program demonstrating basic string operations:

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[20] = "Hello, World!";
   char str2[20];
   char str3[20] = "Hello";
   char str4[50] = "Hello, ";
   // Length of str1
```

```
    printf("Length of str1: %lu\n", strlen(str1));
    // Copy str1 to str2
    strcpy(str2, str1);
    printf("str2 after strcpy: %s\n", str2);
    // Concatenate str3 to str4
    strcat(str4, "C Programming!");
    printf("str4 after strcat: %s\n", str4);
    // Compare str1 and str2
    if (strcmp(str1, str2) == 0) {
        printf("str1 and str2 are equal\n");
    } else {
        printf("str1 and str2 are not equal\n");
    }
    // Find the first occurrence of 'W' in str1
    char *pos = strchr(str1, 'W');
    if (pos != NULL) {
        printf("Found 'W' at position: %ld\n", pos - str1);
    } else {
        printf("'W' not found\n");
    }
    return 0;
}
```

6.5 Standard I/O Functions of String

gets and **puts** are functions in C used for string input and output, respectively. While **gets** reads a line from standard input into a string until a newline or EOF, it has been deprecated due to its inability to prevent buffer overflows, leading to potential security risks. Instead, **fgets** is recommended for safe input handling as it allows specifying the buffer size. **puts**, on the other hand, remains a reliable function for output, writing a string to standard output followed by a newline character, ensuring safe and simple string printing in C programs.

gets and puts in String Handling

In C, **gets** and **puts** are standard I/O functions used for reading and writing strings. They are part of the **stdio.h** library. Here's a detailed look at these functions:

gets Function

The gets function is used to read a line of text from the standard input (typically the keyboard) and store it in a character array. It reads input until a newline character or end-of-file (EOF) is encountered.

```
char *gets(char *str);
```

- **Deprecation:** gets is deprecated and removed in C11 due to its potential for causing buffer overflows. It does not perform bounds checking, which can lead to security vulnerabilities.
- **Replacement:** It is recommended to use fgets instead, which allows you to specify the maximum number of characters to read, thereby avoiding buffer overflow.

```
#include <stdio.h>
int main() {
   char buffer[100];
      printf("Enter a string: ");
   gets(buffer);  // Warning: Unsafe and deprecated function!
   printf("You entered: %s\n", buffer);
      return 0;
}
```

fgets Replacement

Syntax:

```
char *fgets(char *str, int n, FILE *stream);
```

Example:

```
#include <stdio.h>
int main() {
   char buffer[100];
      printf("Enter a string: ");
   fgets(buffer, 100, stdin);
   // Remove newline character if present
   buffer[strcspn(buffer, "\n")] = '\0';
   printf("You entered: %s\n", buffer);
    return 0;
}
```

puts Function

The puts function is used to write a string to the standard output (typically the screen), followed by a newline character.

Syntax:

```
int puts(const char *str);
```

- **puts** automatically appends a newline character to the output.
- It returns a non-negative number on success and **EOF** on error.

Example:

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

```
int main() {
   char buffer[100] = "Hello, World!";
      puts(buffer);  // Output: Hello, World!
      return 0;
}
```

Combining gets and puts

Although gets is unsafe, here's a combined example using **fgets** for safe input and **puts** for output:

```
#include <stdio.h>
int main() {
   char buffer[100];
   printf("Enter a string: ");
   fgets(buffer, sizeof(buffer), stdin);
   // Remove newline character if present
   buffer[strcspn(buffer, "\n")] = '\0';
    puts("You entered:");
   puts(buffer);
      return 0;
}
```

6.6 Summary

Character string handling is a crucial skill in C programming, involving the use of arrays of characters terminated by a null character ('\0') to represent and manipulate text data. The C Standard Library provides a rich set of functions within the string.h header to perform common string operations such as calculating length (strlen), copying (strcpy, strncpy), concatenating (strcat, strncat), comparing (strcmp, strncmp), and searching (strchr, strstr).

A key aspect of string handling is ensuring the safety and security of input operations. Functions like gets are deprecated due to their vulnerability to buffer overflows, making fgets a preferred alternative for reading strings safely. Output functions like puts are used to write strings to standard output, automatically appending a newline character. Understanding and effectively using these functions allows programmers to develop robust, efficient, and secure programs, enabling them to handle text processing and string manipulation tasks competently while avoiding common pitfalls.

6.7 Questions

1. **What is a character string in C?**

Ans.1 A character string in C is a sequence of characters terminated by a null character (**'\0'**). It is represented as an array of characters, where the null terminator signifies the end of the string.

2. **What is the difference between strcpy and strncpy?**

Ans.2 **strcpy** copies a source string to a destination string, assuming the destination is large enough to hold the source string plus the null terminator. **strncpy** copies up to a specified number of

characters from the source to the destination, and if the source is shorter than the specified length, the destination is not null-terminated.

3. **Why is the gets function considered unsafe, and what should be used instead?**

Ans.3 **gets** is considered unsafe because it does not perform bounds checking, which can lead to buffer overflows and potential security vulnerabilities. Instead, **fgets** should be used, as it allows specifying the maximum number of characters to read, thus preventing buffer overflow.

4. **How do you concatenate two strings in C?**

Ans.4 To concatenate two strings, you can use the **strcat** function:

```
char str1[50] = "Hello, ";
char str2[] = "World!";
strcat(str1, str2); // str1 now contains "Hello, World!"
```

For a safer alternative that limits the number of characters concatenated, use **strncat**.

5. **What does the strlen function do?**

Ans.5 The **strlen** function calculates the length of a string, excluding the null terminator.

For example:

```
char str[] = "Hello, World!";
size_t length = strlen(str); // length is 13
```

6. **How can you safely read a string from the user in C?**

Ans.6 You can safely read a string from the user using the **fgets** function:

```
char buffer[100];
fgets(buffer, sizeof(buffer), stdin);
buffer[strcspn(buffer, "\n")] = '\0'; // Remove newline character if present
```

7. **What is the purpose of the strchr function?**

Ans.7 The **strchr** function is used to find the first occurrence of a character in a string. It returns a pointer to the character within the string or **NULL** if the character is not found:

```
char str[] = "Hello, World!";
char *pos = strchr(str, 'W'); // pos points to the 'W' in "World!"
```

6.8 References:

- Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
- King, K. N. (2008). C Programming: A Modern Approach (2nd ed.). W. W. Norton & Company.
- Prata, S. (2014). C Primer Plus (6th ed.). Addison-Wesley Professional.
- Deitel, P., & Deitel, H. (2016). C How to Program (8th ed.). Pearson.

***