CL1002 ***Programming Fundamentals Lab***	Lab 09 Function, String Library, & 2D Character Array

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2024

Author: Mr. Ghulam Qadir

AIMS AND OBJECTIVES

Aim:

To develop an understanding of functions, string manipulation using string library's functions, and handling 2D character arrays in C programming.

Objectives:

- Familiarize with function declaration, definition, and calling in C.
- Learn how to pass values and arrays to functions.
- Utilize standard library string functions Like strlen, strcpy, strcat, and strcmp, etc.
- Explore 2D character arrays to manage multi-dimensional string data.

Introduction

In C programming, **functions** are blocks of code designed to perform a specific task. Functions help in organizing code, making it reusable, modular, and more readable. In C, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. Every C program has at least one function, main (), which serves as the entry point. However, custom functions can be defined to encapsulate repetitive or complex tasks, improving program efficiency and maintainability. Functions are declared, defined, and called **procedure** or subroutine in other programming languages of code that can perform specific tasks and return results. Functions enhance program readability and help manage complex programs.

In C programming, the **string library** provides a set of functions specifically designed for handling strings, which are sequences of characters terminated by a null character ('\0'). Unlike other languages, C does not have a dedicated string data type; instead, strings are represented as arrays of char. To work with strings, the **`<string.h>`** library includes various functions that enable manipulation, comparison, and analysis of these character arrays.

A **2D character** array is used to store a collection of strings, where each row in the array represents a separate string or sequence of characters. This structure is useful when managing multiple strings simultaneously, such as an array of words, sentences, or user input lines. Using standard library string functions simplifies common string manipulations like concatenation, comparison, and copying. Additionally, 2D character arrays (arrays of strings) allow efficient storage and manipulation of textual data, ideal for handling multiple strings within a single structure.

Section 1: Functions

Functions in C are declared with a return type, name, and parameters list, defined with a block of code performing a specific task, and then called where needed in the program.

- It provides modularity to your program's structure.
- It makes your code reusable. You just have to call the function by its name to use it, wherever required.
- In the case of large programs with thousands of code lines, debugging and editing become easier if you use functions.
- It makes the program more readable and easier to understand.

It allows modular coding by breaking down complex processes into smaller, reusable parts. Below are the user-defined function essentials:

Declaration:

```
01 return_type function_name(parameter_list);
```

Definition:

```
01 return_type function_name(parameter_list) {
02     // Function body
03 }
```

Calling a Function:

To execute a function, you simply use `function_name(arguments);`

Syntax of Function Definition

```
01 returnType functionName(type1 argument1, type2 argument2)
02 {
03    //body of the function
04 }
```

When a function is called, the control of the program is transferred to the function definition and the compiler starts executing the codes inside the body of a function.

Passing Arguments to a Function

Argument refers to the variable passed to the function. The parameters a and b accept to pass the arguments in the function definition. These arguments are called formal parameters of the function.

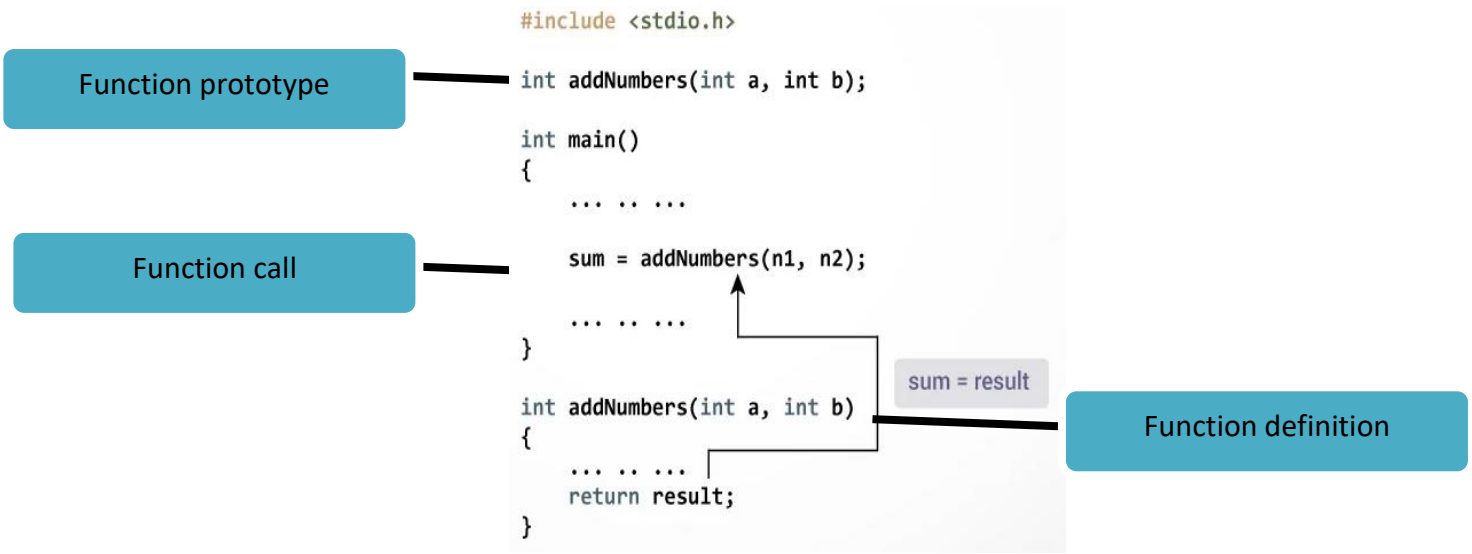
```
01 type1 argument1, type2 argument2
```

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement. We have the data type **int instead of void**. This means that the function returns an int value.

```
01 return (expression);
```

How User-Defined Function Works?

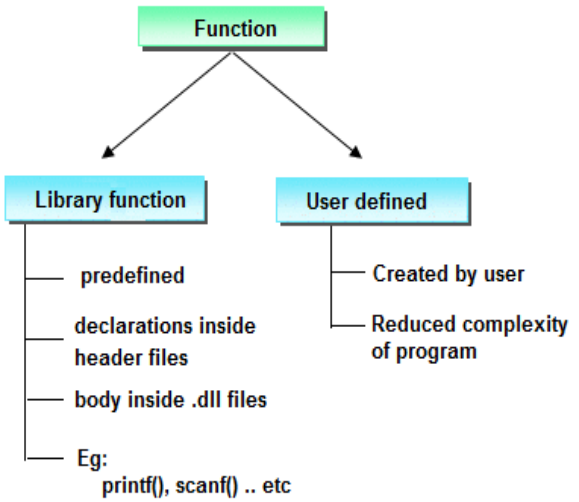


Types of Functions in C Programming

Depending on whether a function is defined by the user or already included in C compilers, there are **two types** of functions in C programming

1. Standard library functions
2. User defined functions

Functions Name	Description
printf()	Print data
scanf()	Read data
getchar()	Read a single a character
sqrt()	Calculate the square
pow()	Calculate the power
fopen()	Open the specified file



Example 1: Function to Calculate Sum of Two Numbers.

```
#include <stdio.h>

int sum(int a, int b);

int main() {

    int result = sum(5, 10);

    printf("Sum is: %d\n", result);

    return 0;

}

int sum(int a, int b) {

    return a + b;

}
```

Output: *Sum is: 15*

Example 2: Function to Find Maximum of Two Numbers .

```
#include <stdio.h>

int max(int a, int b) {

      if (a > b) {

            return a;

      }

      else {

            return b;

      }

}

int main() {

    int result = max(8, 25);

    printf("Maximum is: %d\n", result);

    return 0;

}
```

Output:

Maximum is: 25

Example 3: Passing an Array to Find its Sum.

```
#include <stdio.h>

int sumArray(int arr[], int size);

int main() {

    int arr[] = {1, 2, 3, 4, 5};

    printf("The Sum of Array is: %d\n", sumArray(arr, 5));

    return 0;

}

int sumArray(int arr[], int size) {

    int sum = 0;

    for (int i = 0; i < size; i++) {

        sum += arr[i];

    }

    return sum;

}
```

Output:

The Sum of Array is: 15

Problems:

1. Write a C function that takes two numbers as input and returns their product.
2. Write a function that checks if a given number is even or odd.

Section 2: String Library

String is a series of characters treated as a single unit. A string may include letters, digits and various special characters such as +, -, *, / and $. String literals, or string constants in C are written in double quotation marks. C provides several built-in functions for string handling, such as strlen, strcpy, strcat, strcmp, etc.

Key Functions of String Library Function:

String Length: strlen(str) – Returns the number of characters in a string (excluding the null terminator) – **strlen vs sizeof:** strlen returns you the length of the string stored in an array, however sizeof returns the total allocated size assigned to the array.

```
01 char str[] = "Hello";
02 int len = strlen(str);  // len will be 5
03 int size = sizeof(str); // size will be 6 bytes
```

String Copy: strcpy(dest, src) – Copies the string src into the array dest.

```
01 char src[] = "Hello";
02 char dest[6];
03 strcpy(dest, src);  // dest will contain "Hello"
```

String Concatenation: strcat(dest, src) – Appends the string src to the end of dest.

```
01 char dest[12] = "Hello, ";
02 strcat(dest, "World!");  // dest will be "Hello, World!"
```

strncat(dest, src, n) – Appends up to n characters from src to the end of dest.

```
01 char dest[20] = "Good ";
01 char src[] = "Morning!";
02 strncat(dest, src, 4);  // dest will be "Good Morni"
03 strncat(dest, src, 6);  // dest will be "Good MorniMorni"
```

String Comparison: strcmp(str1, str2) – Compares two strings and returns 0 if they are equal, a positive number if str1 is greater, or a negative number if str2 is greater. It compares the two strings and returns an integer value.

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

```
04 if (strcmp("apple", "banana") < 0) {
01     printf("apple is less than banana");
02 }
```

strncmp(str1, str2, n) – Compares up to n characters of str1 and str2.

```
01 char str1[] = "Hello";
02 char str2[] = "Help";
03 // result will be 0 since first 3 chars "Hel" are the
   same
04 int result = strncmp(str1, str2, 3);
05 // result will be negative since 'l' < 'p' in "Hello" vs
   "Help"
06 result = strncmp(str1, str2, 4);
```

Find Character: strchr(str, ch) – Searches for the first occurrence of character in str.

```
01 char str[] = "Hello, World!";
02 char ch = 'W';
03 if (strchr(str, ch)) {
04     printf("Character %c found in the string.\n", ch);
05 } else {
06         printf("Character %c not found.\n", ch);
07 }
08 // Output: "Character W found in the string."
```

Find Substring: strstr(str, sub) – Finds the first occurrence of the substring needle in haystack.

```
09 char str[] = "This is a simple string";
10 char sub[] = "simple";
01 if (strstr(str, sub)) {
02       printf("Substring '%s' found.\n", sub);
03 } else {
04       printf("Substring '%s' not found.\n", sub);
05   }
06 // Output: "Substring 'simple' found."
```

Example 1: Using strlen to Find String Length.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello, World!";
    printf("Length: %lu\n", strlen(str));
    return 0;
}
```

Output: *Length: 13*

Example 2: Copying Strings with strcpy.

```
#include <stdio.h>
#include <string.h>
int main() {
    char source[] = "Hello";
    char destination[20];
    strcpy(destination, source);
    printf("Copied String: %s\n", destination);
    return 0;
}
```

Output:

Copied String: Hello

Example 3: Compare both the strings till n characters or in other words it compares first n characters of both the strings.

```
#include <stdio.h>

#include <string.h>

int main () {

   char str1[15];

   char str2[15];

   int ret;

   strcpy(str1, "abcdef");

   strcpy(str2, "ABCDEF");

   ret = strncmp(str1, str2, 4);

   if(ret < 0) {

      printf("str1 is less than str2");

   } else if(ret > 0) {

      printf("str2 is less than str1");              //OUTPUT

   } else {                                  //str2 is less  than str1

      printf("str1 is equal to str2");

   }

   return 0;

}
```

Output: *str2 is less than str1*

Problems:

1. Write a program that takes a destination string and a source string as input. Then, take an integer n as input and append only the first n characters of the source string to the destination. Print the new concatenated string.

Section 3: 2D Character Array

A 2D character array in C can be thought of as an array of strings, where each "row" is a separate string. This data structure is useful for handling multiple strings simultaneously, making it ideal for working with lists of names, words, or sentences. 2D character arrays, often used for handling arrays of strings, can be declared as:

Declaration:

```
01 char array_name[rows][columns];
```

where,

- rows represent the number of strings (or lines).
- columns represent the maximum number of characters each string can hold

Initialization:

```
01 char words[3][10] = {"Hello", "World", "C"};
```

Using User Input with scanf:

```
01 char input[2][10];
02 printf("Enter two words:\n");
03 scanf("%s", input[0]);
04 scanf("%s", input[1]);
```

Accessing Elements:

Elements in a 2D character array are accessed using two indices:

```
01 arrayName[row][column];
```

To access a full row (string), use the row index alone:

```
01 printf("%s", arrayName[rowIndex]);
```

Taking Input / Output for a 2D Character Array:

Method 1: Using scanf & printf

The scanf function can be used to take input for each string in a 2D array. Each call to scanf reads a single word. We can print each row (string) individually using a loop.

```
01 char words[3][20];
02 printf("Enter 3 words:\n");
03 for (int i = 0; i < 3; i++) {
04   scanf("%s", words[i]);  // Reads a single word
05 }
06 for (int i = 0; i < 3; i++) {
07    printf("%s\n", words [i]);
08 }
```

Method 2: Using gets (Not Recommended)

The gets function reads an entire line of input, but it's not safe because it doesn't prevent buffer overflows. The puts can be used to output each row (string), automatically appending a newline.

```
01 char sentences[3][50];
02 printf("Enter 3 sentences:\n");
03 for (int i = 0; i < 3; i++) {
04     gets(sentences[i]);  // Not recommended
05 }
06 for (int i = 0; i < 3; i++) {
07   puts(sentences [i]);  // Automatically adds a newline
08 }
```

Method 3: Using fgets (Recommended)

The fgets function is safer than gets and allows you to specify the maximum number of characters to read. The fputs function writes a string to a specified output stream.

```
01 char sentences[3][50];
02 printf("Enter 3 sentences:\n");
03 for (int i = 0; i < 3; i++) {
04 // Reads up to 49 characters + '\0'
05     fgets(sentences[i], 50, stdin);
06 }
07 for (int i = 0; i < 3; i++) {
08     fputs(sentences[i], stdout);
09 fputs("\n", stdout); }// Print newline after each string
```

Example 1: Initialize a 2D character array with names, then print each name.

```
#include <stdio.h>

int main() {

    char names[3][10] = {"Alice", "Bob", "Charlie"};

    for (int i = 0; i < 3; i++) {

        printf("%s ", names[i]);

    }

    return 0;

}
```

Output: *Alice Bob Charlie*

Example 2: Take 3 names as user-defined input and print them.

```
#include <stdio.h>

int main() {

    char names[3][20];

    printf("Enter 3 names:\n");

    for (int i = 0; i < 3; i++) {

        scanf("%s", names[i]);

    }

    printf("\nNames entered:\n");

    for (int i = 0; i < 3; i++) {

        printf("%s\n", names[i]);

    }

    return 0;

}
```

Example 3: To display the characters individually, use nested loops to access each element.

```
#include <stdio.h>

int main() {

    char words[3][10] = {"Hello", "World", "C"};

    // Output each character individually

    for (int i = 0; i < 3; i++) {

        for (int j = 0; words[i][j] != '\0'; j++) {

            printf("%c ", words[i][j]);

        }

        printf("\n");

    }

    return 0;

}
```

Output:

H e l l o

W o r l d

C

Problems:

1. Write a program that initializes a 2D character array with a list of words. Then, take a word as input from the user and check if it exists in the array. Display "Found" if it's there, otherwise display "Not Found".

Note for 2D Character Arrays:

In a 2D character array, each row is effectively a 1D array (string) of characters that ends with a null character \0. When working with character arrays, string functions like scanf, printf, and fgets can treat each row as a single entity (string) because they process the characters until they encounter the null terminator. So mostly there is no need for nested loops.

More problems :'D

1. In a library management system, users are fined for returning books late. The fine is $2 per day after the due date. Write a program that calculates the total fine using a user-defined function calculateFine().

   The function takes the due date and the return date (both as day, month, year).
   Use predefined date handling functions from time.h to calculate the number of days between the due and return dates, and then compute the fine.
   Scenario: A user borrows a book with a due date of October 1, 2024, and returns it on October 15, 2024. Calculate the fine for the user.

2. Create a program that accepts a 2D array of strings (e.g., 5 words with a max length of 20 characters each). Determines if each word (row) is a palindrome. Outputs "Palindrome" or "Not Palindrome" for each word. A palindrome is a word that reads the same forward and backward. For example: "madam", "racecar", "level", "radar".

3. Develop an encryption and decryption system for secure communication. Write a C program with:

- A user-defined function encryptMessage() that encrypts a message by shifting each character by a specified number of positions in the ASCII table (Caesar cipher).
- A function decryptMessage() that decrypts the message.

   Scenario: A user enters the message "HELLO" and shifts by 3. Encrypt and then decrypt the message to verify the output.

4. Implement a function that checks if a given integer is a prime number. Use this function in the main program to check if numbers entered by the user are prime.

5. Write a program to find the intersection of two arrays (common elements between them) using pointers. You must avoid array indexing and use pointer arithmetic.

6. Create a function that reverses a given string and returns the reversed string. Use this function in the main program to display the reversed string entered by the user.

7. Write a C program to calculate the area of a circle using the predefined M_PI constant from math.h. The program should take the radius as input and use the formula area = π * r^2.

8. Create a program that finds both the maximum and minimum values in an array of integers using pointers. You must iterate through the array using pointers and avoid array indexing.