CMR INSTITUTE OF TECHNOLOGY

USN

Internal Assessment Test I – Feb 2025

Sub:	Programming and Problem Solving in C							**Sub Code:**	MMC101
Date:	5-02-2025	**Duration:**	90 min's	**Max Marks:**	50	**Sem:**	I	**Branch:**	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	Give the structure of C program. Also explain the compilation process in detail. **OR**	[10]	CO1	L2
2	What is the use of switch statement? Write a C program to simulate a simple calculator using switch statement.	[10]	CO1	L2
	PART II			
3	What are the different categories of pre-processor directives used in C. Give examples. **OR**	[10]	CO1	L2
4	Explain the different types of looping statements in C with examples. Also give the use of break and continue statements in loops.	[10]	CO1	L2
	PART III			
5	Write an algorithm and develop a C program that reads N integer numbers and arrange them in ascending order using selection Sort. **OR**	[10]	CO2	L3
6	What is an array? Write a C program to input N integers and find largest and second largest element in array.	[10]	CO2	L2,L3
	PART IV			
7	Discuss any 5 string library functions with syntax . Write a C program to copy one string to another without using built in function. **OR**	[10]	CO2	L2,L3
8	Develop a C program to print the following pattern. H HE H E L H E L L H E L L O H E L L O H E L L H E L H H	[10]	CO2	L3

9	**PARTV** Define function. With an example code, explain the following: i. Function Declaration ii. Function Definition iii. Function Call iv. Argument Passing **OR**	[10]	CO3	L1,L3
10	What are the different categories of functions? Write a C-program using function to generate the Fibonacci series.	[10]	CO3	L2,L3

1. Give the structure of C program. Also explain the compilation process in detail.

```
Documentation Section
Pre-processor directives
Definition Section and Global declarations void
main()
{
        Declaration part

        Executable part

}
Sub Program Section
{
        Body of the Sub

}
```

Comments:

- Comments are used to convey a message and used to increase the readability of a program.
- They are not processed by the compiler. There are two types of comments:
    1. Single line comment
    2. Multi line comment

Single line comment
A single line comment starts with two forward slashes (//) and is automatically terminated with the end of the line.
E.g. //First C program
Multi-line comment
A multi-line comment starts with /* and terminates with */.
E.g. /* This program is
used to find Area of the
Circle */

Section 1.Preprocessor Directive section

- The preprocessor directive section is optional.
- The pre-processor statement begins with # symbol and is also called the pre-processor directive.
- These statements instruct the compiler to include C preprocessors such as header files and symbolic constants before compiling the C program.

- They are executed before the compiler compiles the source code. Some of the pre-processor statements are listed below:

(i)Header files

#include<stdio.h>- to be included to use standard I/O functions : prinf(),scanf()

(ii)Symbolic constants

#define PI 3.1412

#define TRUE 1

Section 2: Global declaration Section

This section is used to declare a global variable. These variables are declared before the main() function as well as user defined functions.

Global variables are variables that are used in more than one function.

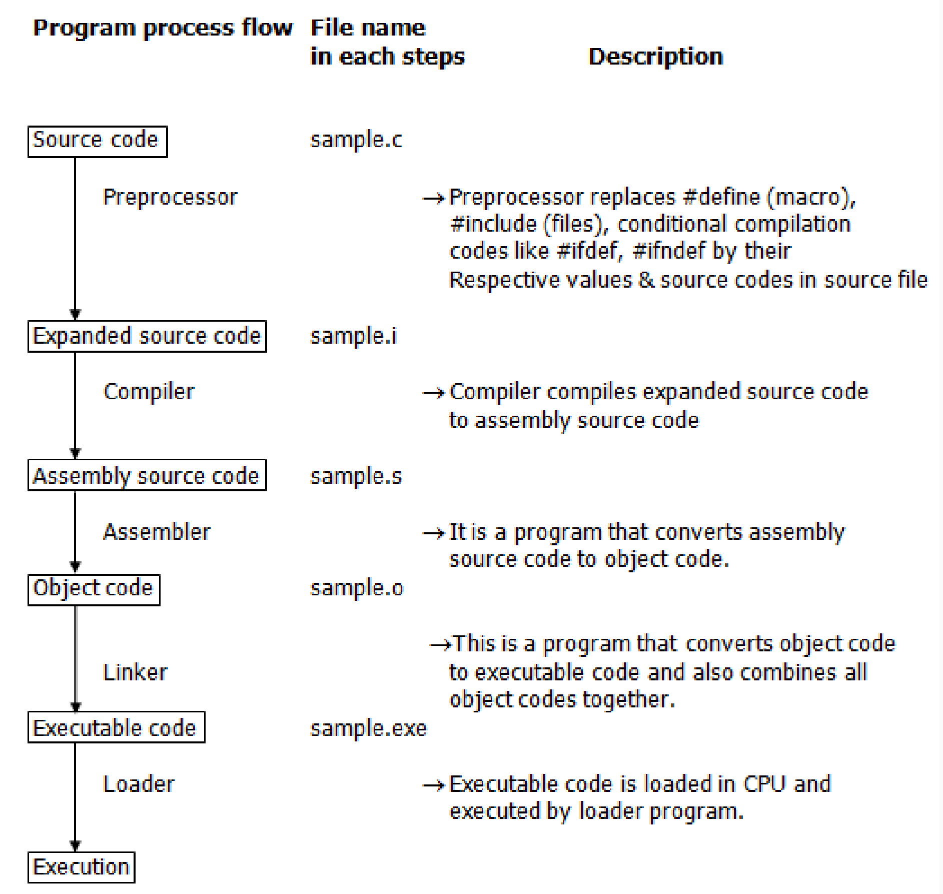
Section 3: Function Section

This section is compulsory. This section can have one or more functions. Every program written in C language must contain main () function. The execution of every C program always begins with the function main ().

Every function consists of 2 parts

1. Header of the function
2. Body of the function

Compilation and Linking process



Program process flow	File name in each steps	Description
Source code	sample.c	
Preprocessor		→ Preprocessor replaces #define (macro), #include (files), conditional compilation codes like #ifdef, #ifndef by their Respective values & source codes in source file
Expanded source code	sample.i	
Compiler		→ Compiler compiles expanded source code to assembly source code
Assembly source code	sample.s	
Assembler		→ It is a program that converts assembly source code to object code.
Object code	sample.o	
Linker		→ This is a program that converts object code to executable code and also combines all object codes together.
Executable code	sample.exe	
Loader		→ Executable code is loaded in CPU and executed by loader program.
Execution		

There are 4 regions of memory which are created by a compiled C program. They are,

1. **First region** – This is the memory region which holds the executable code of the program.
2. **2nd region** – In this memory region, global variables are stored.
3. **3rd region** – stack
4. **4th region** – heap

Compiling

- The process of converting the source code into object code is called compiling. Compiler converts source file to object code and save as separate file with an extension .obj.
- If there is an error in the source code, it does not compile source code and indicates error.

Linking

- A C program contains predefined function. The necessary libraries are linked to the object code file by the linker and produce .exe file. The executable files are executed by the machine.

Executing

- System loader loads the .exe file in the main memory for the execution of the program.

Loader

- Loader is the program of the operating system which loads the executable from the disk into the primary memory (RAM) for execution. It allocates the memory space to the executable module in main memory and then transfers control to the beginning instruction of the program.

2. What is the use of switch statement? Write a C program to simulate a simple calculator using switch statement.

The switch statement in C is a **multi-way decision-making statement** that allows a variable to be tested against multiple values. It is useful when we have multiple conditional branches based on the value of a variable.

Uses of switch Statement:

- Menu-driven programs
- Simple decision-making based on a single variable
- Reduces multiple if-else if conditions for better readability

Syntax of switch Statement:

```
switch (expression) {
    case value1:
        // Code block for value1
        break;
    case value2:
        // Code block for value2
        break;
    default:
        // Default block if no case matches
}
```

```
#include <stdio.h>

int main() {
    char operator;
    double num1, num2, result;

    // Taking input from the user
    printf("Enter first number: ");
    scanf("%lf", &num1);

    printf("Enter an operator (+, -, *, /): ");
    scanf(" %c", &operator);  // Space before %c to handle newline

    printf("Enter second number: ");
    scanf("%lf", &num2);

    // Switch case for arithmetic operations
    switch (operator) {
        case '+':
            result = num1 + num2;
            printf("Result: %.2lf\n", result);
            break;
        case '-':
            result = num1 - num2;
            printf("Result: %.2lf\n", result);
            break;
        case '*':
            result = num1 * num2;
            printf("Result: %.2lf\n", result);
            break;
        case '/':
            if (num2 != 0)
                result = num1 / num2;
            else {
                printf("Error! Division by zero.\n");
                return 1;
            }
            printf("Result: %.2lf\n", result);
            break;
        default:
            printf("Invalid operator!\n");
    }

    return 0;
}
```

3. **What are the different categories of pre-processor directives used in C. Give examples**.
   Preprocessor directives in C are instructions that are processed before compilation. They begin with # and help in code modularity, efficiency, and readability**.**

Categories of Preprocessor Directives

i. Macro Substitution Directives (#define)

- Used to define constants and macros (code replacement).
- Example:
-    #define PI 3.14159
-    #define SQUARE(x) (x * x) // Macro function
- Use Case: Reduces redundancy and improves readability.

ii. File Inclusion Directives (#include)

- Used to include header files.
  Example:
-    #include <stdio.h>   // Standard library file

iii. Conditional Compilation Directives (#ifdef, #ifndef, #if, #else, #endif)
   Used to compile specific parts of the code based on conditions.
      Example:
   #define DEBUG // Uncomment to enable debugging

```
#ifdef DEBUG
printf("Debug mode is ON\n");
#endif
```

Example Program Using Preprocessor Directives

```
#include <stdio.h>
#define PI 3.14159
#define AREA(r) (PI * r * r)

int main() {
   double radius = 5.0;
   printf("Area of circle: %.2lf\n", AREA(radius));

   #ifdef DEBUG
      printf("Debugging mode enabled.\n");
   #endif

   return 0;
}
```

4. **Explain the different types of looping statements in C with examples. Also give the use of break and continue statements in loops.**

In C programming, looping statements are used to execute a block of code multiple times until a specified condition is met. There are three types of loops:

1. for Loop

The for loop is used when the number of iterations is known beforehand. It consists of three parts: initialization, condition, and increment/decrement.

Syntax:

```
for(initialization; condition; increment/decrement) {
   // Code to be executed
}
```

Example:

```
#include <stdio.h>
int main() {
   for(int i = 1; i <= 5; i++) {
      printf("%d ", i);
   }
   return 0;
}
```

Output:

```
1 2 3 4 5
```

2. while Loop

The while loop is used when the number of iterations is not known in advance. It executes as long as the condition is true.

Syntax:

```
while(condition) {
   // Code to be executed
}
```

Example:

```
#include <stdio.h>
int main() {
   int i = 1;
   while(i <= 5) {
      printf("%d ", i);
      i++;
   }
   return 0;
}
```

Output:

```
1 2 3 4 5
```

3. do-while Loop

The do-while loop executes the block of code at least once, regardless of the condition.

Syntax:

```
do {
   // Code to be executed
} while(condition);
```

Example:

```
#include <stdio.h>
int main() {
   int i = 1;
   do {
      printf("%d ", i);
      i++;
   } while(i <= 5);
   return 0;
}
```

Output:

```
1 2 3 4 5
```

Use of break and continue Statements in Loops

1. break Statement

The break statement is used to exit a loop prematurely when a certain condition is met.

Example:

```
#include <stdio.h>
int main() {
   for(int i = 1; i <= 5; i++) {
      if(i == 3)
         break;  // Exits the loop when i is 3
      printf("%d ", i);
   }
   return 0;
}
```

Output:

```
1 2
```

2. continue Statement

The continue statement is used to skip the current iteration of the loop and move to the next iteration.

Example:

```
#include <stdio.h>
int main() {
   for(int i = 1; i <= 5; i++) {
      if(i == 3)
         continue;  // Skips iteration when i is 3
      printf("%d ", i);
   }
   return 0;
}
```

Output:

```
1 2 4 5
```

5. **Write an algorithm and develop a C program that reads N integer numbers and arrange them in ascending order using selection Sort.**

   Selection Sort works by repeatedly selecting the smallest element from the unsorted part of the array and swapping it with the first element of the unsorted part.

   Algorithm

1. Start
2. Input the number of elements **N**
3. Input **N** integer numbers into an array

```
4. Repeat for i = 0 to N-2
       Set minIndex = i
       For j = i+1 to N-1
           If arr[j] < arr[minIndex], update minIndex = j
       Swap arr[i] and arr[minIndex]
5. Print the sorted array
6. End
```

```c
#include <stdio.h>

// Function to perform Selection Sort
void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;

    for(i = 0; i < n-1; i++) {
        minIndex = i;

        // Find the minimum element in the unsorted part
        for(j = i+1; j < n; j++) {
            if(arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        // Swap the found minimum element with the first element of the unsorted part
        temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

// Function to display the array
void displayArray(int arr[], int n) {
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;

    // Input number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Input elements
    printf("Enter %d integers: ", n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Sorting the array
    selectionSort(arr, n);

    // Display sorted array
    printf("Sorted array in ascending order: ");
    displayArray(arr, n);

    return 0;
  }
```

6. **What is an array? Write a C program to input N integers and find largest and second largest element in array.**

   An array in C is a collection of elements of the same data type stored at contiguous memory locations. It allows efficient access and manipulation of multiple values using a single variable name and index.

```
#include <stdio.h>
int main() {
int array[10] = {101, 11, 3, 4, 50, 69, 7, 8, 9, 0};
int loop, largest, second;
if(array[0] > array[1]) {
largest = array[0];
second = array[1];
} else {
largest = array[1];
second = array[0];
}
for(loop = 2; loop < 10; loop++) {
if( largest < array[loop] ) {
second = largest;
largest = array[loop];
} else if( second < array[loop] ) {
second = array[loop];
}
}
printf("Largest - %d \nSecond - %d \n", largest, second);
return 0;
}
```

7. **Discuss any 5 string library functions with syntax . Write a C program to copy one string to another without using built in function. Five String Library Functions in C**

The C standard library provides several functions to manipulate strings, which are declared in the <string.h> header file.

***1.* strlen() - Find Length of a String**

Syntax: size_t strlen(const char *str);

Description: Returns the length of the string (excluding the null character \0).

Example:

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

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

Output: Length of string: 5

2. **strcpy() - Copy One String to Another**

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

Description: Copies the contents of src into dest (including the null character).

Example:

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

int main() {
    char source[] = "CMRIT";
    char destination[20];
    strcpy(destination, source);
    printf("Copied String: %s\n", destination);
    return 0;
      }
```

Output: Copied String: CMRIT

3. **strcmp() - Compare Two Strings**

Syntax: int strcmp(const char *str1, const char *str2);

Description: Compares two strings and returns:

0 if both are equal

A negative value if str1 < str2

A positive value if str1 > str2

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: Strings are not equal.

4. strcat() - Concatenate Two Strings

- **Syntax:** char *strcat(char *dest, const char *src);
- **Description:** Appends src to dest, modifying dest.
- **Example:**

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

Output: Concatenated String: Hello, World!

5. strrev() - Reverse a String (Non-Standard)

- **Syntax:** char *strrev(char *str);
- **Description:** Reverses the given string in place (not a standard function in <string.h>, but available in some compilers).
- **Example:**

```
•    #include <stdio.h>
•    #include <string.h>
•
•    int main() {
•        char str[] = "CMRIT";
•        printf("Reversed String: %s\n", strrev(str));
•        return 0;
•    }
```

Output: Reversed String: TIRMC

C Program to Copy One String to Another Without Using strcpy()

```
#include <stdio.h>

void copyString(char dest[], char src[]) {
    int i = 0;
    while (src[i] != '\0') {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0'; // Add null terminator
}

int main() {
    char source[100], destination[100];

    // Input string
    printf("Enter a string: ");
    scanf("%[^\n]s", source);  // Read string with spaces

    // Copy string manually
    copyString(destination, source);

    // Display copied string
    printf("Copied String: %s\n", destination);

    return 0;
}
```

8. Develop a C program to print the following pattern.

```
H
HE
H E L
H E L L
H E L L O
H E L L O
H E L L
H E L
H E
 H
```

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

int main() {
    char str[] = "HELLO";
    int len = strlen(str);

    // Upper half of the pattern
    for(int i = 0; i < len; i++) {
        for(int j = 0; j <= i; j++) {
            printf("%c ", str[j]);
        }
        printf("\n");
    }
    // Lower half of the pattern
    for(int i = len - 1; i > 0; i--) {
        for(int j = 0; j < i; j++) {
            printf("%c ", str[j]);
        }
        printf("\n");
    }

    return 0;
}
```

9. Define function. With an example code, explain the following:

i. Function Declaration **ii. Function Definition**
iii. Function Call **iv. Argument Passing**

A function in C is a block of code that performs a specific task. It helps in code reusability, modular programming, and better readability**.**

A function in C consists of:

1. **Function Declaration** (Prototype)
2. **Function Definition**
3. **Function Call**
4. **Argument Passing** (Call by Value / Call by Reference)

Example Program

Below is a C program that demonstrates function declaration, function definition, function call, and argument passing**.**

```
#include <stdio.h>

// Function Declaration (Prototype)
int addNumbers(int a, int b);

int main() {
    int num1 = 10, num2 = 20, result;

    // Function Call
    result = addNumbers(num1, num2);

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

    return 0;
}

// Function Definition
```

```
int addNumbers(int a, int b) {
    return a + b;  // Returns sum of two numbers
}
```

Explanation of Concepts

1. Function Declaration (Prototype)

- Syntax:
- return_type function_name(parameter_list);
- It tells the compiler about the function's name, return type, and parameters before it is defined.
- Example in the program:
- int addNumbers(int a, int b);

2. Function Definition

- **Syntax:**
- return_type function_name(parameter_list) {
- // Function body
- }
- It contains the actual implementation of the function.
- Example in the program:
- int addNumbers(int a, int b) {
- return a + b;
- }
- This function takes two integers as input and returns their sum.

3. Function Call

- A function is **called** in the main() function or another function.
- Example in the program:
- result = addNumbers(num1, num2);
- The function addNumbers(num1, num2) is called, and the returned value is stored in result.

4. Argument Passing

C functions can receive values using parameters. There are two types of argument passing:

1. **Call by Value:** (Used in the example above)
   - A copy of the argument is passed to the function.
   - Any changes inside the function do not affect the original values.
2. **Call by Reference:** (Uses pointers)
   - Instead of passing a copy, the function receives the actual memory address.

10. What are the different categories of functions? Write a C-program using function to generate the Fibonacci series.

There are two types of functions in C

- Library Functions(Built-in)
  These functions are provided by the system and stored in library; therefore it is also called 'Library Functions'.
  e.g. scanf(), printf(), strcpy, strlwr, strcmp, strlen, strcat etc.

To use these functions, we need to include the appropriate C header files.

- User Defined Functions

  These functions are defined by the user at the time of writing the program. It reduces the complexity of a big program and optimizes the code.

  Types of User-defined functions:

  1.C function with arguments (parameters) and with return value.
  2.C function with arguments (parameters) and without return value.
  3.C function without arguments (parameters) and without return value.
  4.C function without arguments (parameters) and with return value.

```
#include <stdio.h>

// Recursive Fibonacci function
int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;

    printf("Enter the number of terms: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");

    return 0;
}
```