`

RV UNIVERSITY Go, change the world an initiative of RV EDUCATIONAL INSTITUTIONS	USN **School of Computer Science and Engineering** **B.Tech (Hons.)** **Midterm Answer Scheme Set 1** **Academic Year 2024-2025**		
Course: Programming in C		**Course Code: CS 1003**	**Semester: 1**
Time: 10:00 AM to 11:30 AM	**Duration: 90 minutes**	**Date : 5-12-2024**	**Max Marks: 25**

Notes/ Instructions:

a) Answer all questions

Sl. No.	PART A – (MCQs) Max Marks(5)	Marks	L1-L6	CO
1.	**What will be the output of the following code?** **#include <stdio.h>** **int main() {** **int a = 5, b = 2, c;** **c = a > b ? a + b : a - b;** **printf("%d\n", c);** **return 0;** **}** **A) 3** **B) 7** **C) 5** **D) 2** Ans : b) 7	1	L2	CO3
2.	**What is the result of using the increment operator i++ ?** **a) The variable i is incremented by 1 immediately, and the incremented value is used in the current expression.** **b) The variable i is incremented by 1 after its current value is used in the expression.** **c) The variable i is decremented by 1, and the decremented value is used in the expression.** **The value if i remains unchanged.** **Ans : b) The variable i is incremented by 1 after its current value is used in the expression.**	1	L2	CO3
3.	**What will be the output of the following code?** **#include <stdio.h>** **int main() {** **int i;** **for (i = 0; i < 10; i++) {** **if (i == 5) {** **break;**	1	L3	CO2

L1- Remember, L2- Understand, L3 – Apply, L4- Analyse, L5 – Evaluate, L6 – Create

	}
	printf("%d ", i);
	}
	return 0;
	}
	a) 0 1 2 3 4 5
	b) 0 1 2 3 4 5 6 7 8 9 10
	c) 0 1 2 3 4
	d) No output
	
	Answer – c) As soon as i==5, 'break' keyword is executed and the loop is exited.v			

4.	A _____is defined to terminate the recursion function when the boundary conditions are met. a) Base case b) Recursive case c.) Functions case d) String case **Answer – a)**	1	L2	CO3

5.	How many times will the following C code execute the printf statement?
	
	#include <stdio.h>
	int main() {
	int i = 5;
	do {
	printf("Hello, World!\n");
	} while (i < 5);
	return 0;
	}
	A) 0
	B) 1
	C) 5
	D) Infinite
	
	B) 1
	The do-while loop executes the body of the loop at least once, even if the condition is false. Here, the condition i < 5 is false from the beginning, but the printf statement is executed once before the condition is checked.
		1	L2	CO3

Sl. No.	PART B – Max Marks (20)	Marks	L1-L6	CO
1.	A teacher wants to calculate the average marks of students in a class. The number of students is not fixed and is	5	L2	CO3

	entered by the user at runtime. After entering the marks for all students, the program should display the average marks.
	
	Write a program using a while loop to solve this. Assume that the user enters -1 to indicate the end of marks entry.
	
	#include <stdio.h> (5 marks)
	
	int main() {
	int count = 0; // To keep track of the number of students
	float sum = 0; // To store the total sum of marks
	float marks;
	
	printf("Enter marks of students (Enter -1 to stop):\n");
	
	while (1) {
	scanf("%f", &marks);
	
	if (marks == -1) { // Check for termination condition
	break;
	}
	
	sum += marks; // Add the marks to the total sum
	count++; // Increment the count of students
	}
	
	if (count == 0) {
	printf("No marks were entered.\n");
	} else {
	float average = sum / count; // Calculate the average
	printf("The average marks of the students are: %.2f\n", average);
	}
	
	return 0;
	}			

2.	**2a.) Solve the following statement using correct precedence operators and provide the output** **11 + 2 > 6 && !0 || 11! = 7 && 11-2 <= 5** **Provide the answer stepwise.** **Ans : Step 1: Evaluate arithmetic operations (1 Mark)** We first calculate the results of arithmetic operations, as they have higher precedence than relational, logical, or equality operators: 1. 11 + 2 = 13 2. 11 - 2 = 9	**3+2**	**L3, L1**	**CO2**

	Updated expression: 13 > 6 && !0 || 11 != 7 && 9 <= 5 **Step 2: Evaluate the logical NOT (!) operator** Logical NOT (!) has higher precedence than relational or logical operators: • !0 = 1 (as 0 is false, and its NOT is true) Updated expression: 13 > 6 && 1 || 11 != 7 && 9 <= 5 **Step 3: Evaluate relational operators (>, <=, !=)** Relational operators are evaluated next. They compare values and return 1 for true and 0 for false: 1. 13 > 6 = 1 (true) 2. 11! = 7 = 1 (true) 3. 9 <= 5 = 0 (false) Updated expression: 1 && 1 || 1 && 0 **Step 4: Evaluate logical AND (&&) (1 Mark)** Logical AND (&&) is evaluated from **left to right**, as it has higher precedence than logical OR (||): 1. 1 && 1 = 1 (both operands are true) 2. 1 && 0 = 0 (one operand is false) Updated expression: 1 || 0 **Step 5: Evaluate logical OR (||) (1 Mark)** Finally, logical OR (||) is evaluated. It returns 1 if any operand is true: • 1 || 0 = 1 (true) **Final Answer:** The result of the expression is **1** (true).			
	2b) Demonstrate the use of pointers with syntax and example. • A pointer is a variable that stores the memory address of another variable. Instead of holding a data value directly, a pointer holds the location in memory where the data is stored.			

	• Example:
	  • Declaring pointers (1 mark)
	  • int *integerPointer; // Pointer to an integer
	  • char *charPointer; // Pointer to a character
	  • float *floatPointr; // Pointer to a float
	  • Initializing Pointers
	  • A pointer should be initialized with the address of a variable of the same type using the address-of operator (&).
	  • int a = 10;
	  • int *ptr = &a; // pointer now holds the address of var
	  (1 mark)
	  • Pointers allow functions to modify variables
	  • outside their own scope so inturn provides better efficiency
	  • Pointers save memory space. Execution time with pointers is faster because data are manipulated with the address, that is, direct access to memory location.
	  • Pointers provide a way to iterate through arrays and manipulate strings efficiently.
	  • Pointers are essential for allocating and managing memory at runtime using functions like malloc, calloc, and free
	  • It can help in reducing the code and improving the performance.			

3	**a. Explain in detail about call by reference with an example**
	
	Pass by Reference
	In Call by Reference, the address of the actual parameter is passed to the function. The function operates on the address and can modify the original variable. (1 Mark)
	Example:
	#include <stdio.h>
	void modifyValue(int *num) {
	  *num = 20; // Changes the value at the address
	  printf("Inside modifyValue, *num = %d\n", *num);
	}
	int main() {
	  int x = 10;
	  printf("Before function call, x = %d\n", x);
	  modifyValue(&x); // Pass the address of x
	  printf("After function call, x = %d\n", x); // x is modified
	  return 0;
	}
	Output:
		2+3	**L1**	**CO3**

Before function call, x = 10
Inside modifyValue, *num = 20
After function call, x = 20 (1 Mark)

b. Differentiate between local & global variable with an example.

Global Variables (1 Mark)

- **Definition**: Declared outside all functions, typically at the top of the program.
- **Scope**: Accessible throughout the entire program, including all functions.
- **Lifetime**: Exists for the duration of the program's execution.
- **Default Value**: Automatically initialized to 0 (or NULL for pointers) if not explicitly initialized.
- **Usage**: Useful for sharing data between multiple functions or maintaining state across the program.

Example:

```
#include <stdio.h>

int globalVar = 10; // Global variable

void display() {
   printf("Global Variable: %d\n", globalVar);
}

int main() {
   printf("Accessing Global Variable in main: %d\n",
globalVar);
   display(); // Function can access the global variable
   return 0;
}
```

Local Variables (1 Mark)

- **Definition**: Declared inside a function or block (e.g., inside {}).
- **Scope**: Accessible only within the function or block in which they are declared.
- **Lifetime**: Exists only during the execution of the function or block where they are defined.
- **Default Value**: Not initialized automatically; contains garbage values if not explicitly initialized.
- **Usage**: Used for temporary data required only within a specific function or block.

Example:

```
#include <stdio.h>

void display() {
   int localVar = 20; // Local variable
```

```
    printf("Local Variable: %d\n", localVar);
}

int main() {
    display();
    // printf("Trying to access Local Variable: %d\n",
localVar); // Error: localVar not accessible here
    return 0;
}
```

EXAMPLES (1 MARK)

Key Differences

Feature	Global Variable	Local Variable
Declaration	Outside all functions.	Inside a function or block.
Scope	Entire program.	Limited to the function/block.
Lifetime	Throughout program execution.	During function/block execution.
Default Value	Initialized to 0 or NULL.	Uninitialized (garbage value).
Memory Allocation	Allocated in the data segment.	Allocated in the stack.
Access	Accessible by any function.	Accessible only within its scope.

4	**Explain the concept of Recursion and Write a program in C using recursion to find GCD of two numbers** **Sol : Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem. Each recursive call reduces the problem size, and a base case is defined to stop the recursion when the problem becomes simple enough to solve directly. (2 Marks)** **#include <stdio.h> (3Marks)** **// Recursive function to find GCD** **int findGCD(int a, int b) {** **// Base case** **if (b == 0)** **return a;**	**5**	**L3**	**CO3**

```
    // Recursive case
    return findGCD(b, a % b);
}

int main() {
    int num1, num2;

    // Input two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Call the recursive GCD function
    int gcd = findGCD(num1, num2);

    // Print the result
    printf("The GCD of %d and %d is: %d\n", num1,
num2, gcd);

    return 0;
}
```

Course Outcomes

1. Understand and analyze the concepts of number system, memory, fundamental concepts in C programming language.
2. Demonstrate fundamental data types, operators and console I/O functions in C.
3. Implement programs using decision control statements, control flow statements, arrays, strings, functions, and pointers to solve mathematical problems.
4. Analyze and evaluate the programming concepts based on user defined data types and file handling in C language.
5. Design and implement appropriate concepts of C to solve real-world problems.

Marks Distribution									
L1	L2	L3	L4	L5	L6	CO1	CO2	CO3	CO4
9	12	6	-	-	-	0	6	19	-

L1- Remember, L2- Understand, L3 – Apply, L4- Analyse, L5 – Evaluate, L6 – Create

`

RV UNIVERSITY Go, change the world an initiative of RV EDUCATIONAL INSTITUTIONS	USN **School of Computer Science and Engineering** **B.Tech (Hons.)** **Midterm Answer Scheme Set 2** **Academic Year 2024-2025**		
Course: Programming in C		**Course Code: CS 1003**	**Semester: 1**
Time: 10:00AM to 11:30AM	**Duration: 90 minutes**	**Date : 05/12/2024**	**Max Marks: 25**

Notes/ Instructions:

a) Answer all questions

b) Smartphone and scientific calculator not allowed.

Sl. No.	PART A – (MCQs) Max Marks(5)	Marks	L1-L6	CO
1.	**What will be the output of the following code?** **#include <stdio.h>** **int main() {** **printf("%d", 5<<2);** **}** **return 0;** **}** **a) 10** **b) 20** **c) 0** **d) No output** **Answer: b) 20.**	1	L2	CO3
2.	**What is the difference between the prefix (--x) and postfix (x--) decrement operators in C?** **a) Both decrement the value of x by 1 but return the value differently.** **b) The prefix operator does not decrement the value, while the postfix does.** **c) The postfix operator decrements the value first, while the prefix does it afterward.** **d) Both decrement the value of x and return the same result.** **Answer:** a) Both decrement the value of x by 1 but return the value differently.	1	L2	CO3
3.	**What is the output of the following code?** **int i = 5;** **while (i > 0) {** **printf("%d", i--);**	1	L3	CO2

	} **A) 54321 B) 12345 C) 55555 D) 0** **Answer: a) 54321**			
4.	**Which of the following is true about functions in C?** **A) A function can have more than one return statement.** **B) A function must always return a value.** **C) Functions in C cannot take arguments.** **D) A function's return type is optional.** **Answer: A) A function can have more than one return statement.**	**1**	**L2**	**CO3**
5.	**What is the output of the following code?** **#include <stdio.h>** **int main() {** **int x = 1;** **do {** **printf("%d ", x);** **x++;** **} while (x <= 3);** **return 0;** **}** A) 1 2 B) 1 2 3 C) 1 2 3 4 D) No output Answer: B) 1 2 3	**1**	**L2**	**CO3**

Sl. No.	PART B – Max Marks (20)	Marks	L1-L6	CO

6.	**Write a C program to calculate the sum of the series: 1+x+x^2+x^3+...+x^n.**
	where x and n are given as inputs. Use a loop to calculate the sum of the series.
	Answer:
	#include <stdio.h>
	int main() {
	int x, n;
	int sum = 1; // Start with 1, which is x^0
	int term = 1; // First term (x^0)
	printf("Enter the value of x: ");
	scanf("%d", &x);
	printf("Enter the value of n: ");
	scanf("%d", &n);
	for (int i = 1; i <= n; i++) {
	term *= x; // Multiply the term by x for each power
	sum += term; // Add the term to the sum
		5	**L2**	**CO3**

	} printf("The sum of the series is: %d\n", sum); return 0; }			

7.	**a. Solve the following statement using correct precedence operators and provide the output where a=5, b=3, c=1, d=2, e=5.**
	!(++a + b-- * c > d || -b < --d && e/ ++a == c)
	provide the answer stepwise.
	Answer: !(++5 + 3-- * 1 > 2 || -3 < --2 && 5/ ++5 == 1)
	Step 1: Solving the first statement before logical OR:
	!(6+ 3-- * 1 > 2 || -3 < --2 && 5/ ++5 == 1) Unary has higher precedence over arithmetic
	!(6 + 3 * 1 > 2 || -3 < --2 && 5/ ++5 == 1)
	!(6 + 3 * 1 > 2 || -3 < 1 && 5/ ++5 == 1)
	!(6 + 3 * 1 > 2 || -3 < 1 && 5/ 6 == 1)
	!(6 + 3 > 2 || -3 < 1 && 5/ 6 == 1) Arithmetic operators have higher priority over relational operators.
	!(9 > 2 || -3 < 1 && 5/ 6 == 1)
	!(9 > 2 || -3 < 1 && 0.83 == 1) Relational has higher precedence over logical operators.
	!(1 || -3 < 1 && 0.83 == 1)
	!(1 || 1 && 0.83 == 1)
	!(1 || 1 && 0)
	!(1 || 0)
	!(1)= 0
	b. What is type casting? List its types with examples.
	Answer: Type Casting is the process of converting a variable from one data type to another. In C, type casting is used to change the data type of a variable temporarily, which is often necessary for performing calculations or working with variables of different types.
	There are two main types of type casting in C:
	i. Implicit Type Casting:
	Example:
	int a = 5;
	float b = a; // 'int' to 'float' conversion
	printf("%f", b); // Output: 5.000000
	ii. Explicit type casting:
	Example:
	float a = 5.9;
	int b = (int)a; // 'float' to 'int' conversion
	printf("%d", b); // Output: 5
	Here, a (a float) is cast to int, which truncates the decimal part and assigns 5 to b.
		3+2	**L3, L1**	**CO2**

8.	**a. Define function and explain its elements.**	**3+2**	**L1**	**CO3**

	Answer: A function in C is a block of code that performs a specific task. Functions allow us to break down complex problems into smaller, manageable parts, promoting code reusability and modularity. A function typically has:
	
	A name for identification.
	Parameters (optional) that serve as input values.
	A return type that specifies the data type of the output.
	A function body containing the code to be executed.
	
	Syntax of a Function
	return_type function_name(parameter1_type parameter1, parameter2_type parameter2, ...) {
	// function body
	// code to execute
	return value; // (optional, depending on return_type)
	}
	b. Differentiate between global and static variables in C.
	Answer:
	Global Variables:
	● A global variable is declared outside any function, typically at the top of the program, making it accessible throughout the entire program. Any function can read or modify it.
	● Global variables are useful when multiple functions need to share data.
	● They are stored in the data segment and persist throughout the program's execution.
	
	Static Variables:
	● A static variable is declared inside a function or a block but uses the static keyword. It retains its value between multiple function calls but is only accessible within the function or file where it is defined.
	● Static variables are often used when you need to preserve the state of a variable across function calls without making it accessible globally.
	● Static variables are also stored in the data segment but do not get reinitialized every time the function is called, unlike local variables.			

9.	**Write a C program for this scenario:** **A factory produces a certain number of products every day. The number of products follows a factorial pattern, i.e., on day 1, it produces 1 product, on day 2 it produces 2 products, on day 3 it produces 3 products, and so on. Calculate the total number of products produced by the factory up to day n using recursion.** **Answer:** #include <stdio.h>	**5**	**L3**	**CO3**

```
// Recursive function to calculate the total number of
products produced up to day n
long long totalProducts(int n) {
    if (n == 1) {
        return 1;  // Base case: On day 1, the total products = 1
    }
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;  // Calculate the factorial for the current day
    }
    return fact + totalProducts(n - 1);  // Add current factorial
and call recursively for previous days
}

int main() {
    int n;

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

    // Calculate and print the total number of products
produced up to day n
    long long result = totalProducts(n);
    printf("Total number of products produced up to day %d:
%lld\n", n, result);

    return 0;
}
```

Course Outcomes

1. Understand and analyze the concepts of number system, memory, fundamental concepts in C programming language.
2. Demonstrate fundamental data types, operators and console I/O functions in C.
3. Implement programs using decision control statements, control flow statements, arrays, strings, functions, and pointers to solve mathematical problems.
4. Analyze and evaluate the programming concepts based on user defined data types and file handling in C language.
5. Design and implement appropriate concepts of C to solve real-world problems.

Marks Distribution									
L1	L2	L3	L4	L5	L6	CO1	CO2	CO3	CO4
5	11	9	-	-	-	0	6	19	-

L1- Remember, L2- Understand, L3 – Apply, L4- Analyse, L5 – Evaluate, L6 – Create