(Affiliated to University of Mumbai)	***End Semester Examination (R-24) SH 2025 Answer Key with marking scheme***
Branch: ALL	Course: C-PROGRAMMING
Year/ Semester: I/ F.Y.	Course code: FEC104
Time: 02 hours	Marks: 60

		Marks
Q. 1	Attempt any Three. (All questions carry equal marks)	
A.	**Differentiate between structure and union. (05 marks)** **Answer (Explanation):** • **Structure (struct): Groups variables of different types under one name. Each member gets its own memory; total memory = sum of size of members (+ padding). Example use: student record with name, id, marks.** • **Union: Groups variables but all members share the same memory location; size = size of largest member. Only one member holds a meaningful value at a time. Useful to save memory when fields are mutually exclusive (e.g., variant fields).** • **Access: Both use dot operator for member access. Declaration syntax similar.** **Marking Scheme:** • **Definition of structure — 2 marks** • **Definition of union — 2 marks** • **Example/use-case or one key difference (memory behavior) — 1 mark**	
B.	**What is the difference between malloc and calloc? (05 marks)** **Answer (Explanation):** • malloc(size) — allocates size bytes and returns void*. Memory is **uninitialized** (garbage). • calloc(nmemb, size) — allocates memory for an array of nmemb elements of size bytes each, **initialized to zero**. • realloc() changes size of previously allocated memory. free() releases memory. Example: int *p = malloc(10*sizeof(int)); vs int *q = calloc(10, sizeof(int));. **Marking Scheme:** • Explanation of malloc — 2 marks • Explanation of calloc — 2 marks • Short example / mention of initialization difference — 1 mark	

C.	**Write a program using if-else-if ladder to check whether a number is positive, negative, or zero. (05 marks)**
	Answer (Program & Explanation):
	#include <stdio.h>
	int main() {
	int n;
	scanf("%d", &n);
	if (n > 0)
	printf("Positive\n");
	else if (n < 0)
	printf("Negative\n");
	else
	printf("Zero\n");
	return 0;
	}	

	Explanation: The if-else-if ladder checks conditions in order; the first true branch runs. **Marking Scheme:** • Correct condition logic (if, else if, else) — 3 marks • Correct I/O and syntax — 2 marks	
D.	**Explain the various file open modes in C. (05 marks)** **Answer (Explanation):** Common fopen() modes: • "r" — open for reading; file must exist. • "w" — open for writing; creates file or truncates existing file. • "a" — open for appending; writes appended to end; creates file if not present. • "r+" — read and write; file must exist. • "w+" — read and write; creates/truncates file. • "a+" — read and append; reading allowed, writes append. • Appending "b" for binary when required (e.g., "rb", "wb"). **Marking Scheme:** • Any 4 correct modes with brief explanation — 4 marks • One example or caution (e.g., w truncates file) — 1 mark	
Q.2	Attempt any Three. (All questions carry equal marks)	

A.	**A. Differentiate between call by value and call by reference. Write a program to swap two numbers using call by reference concept. (10 marks)**
	Answer (Explanation & Program):
	• **Call by value:** Function receives copies of arguments. Changes in function do not affect original variables.
	• **Call by reference:** Function receives addresses (pointers) to arguments. Function can modify original variables through dereferencing.
	Swap program using call by reference (pointers):
	#include <stdio.h>
	
	void swap(int *a, int *b) {
	int temp = *a;
	*a = *b;
	*b = temp;
	}
	
	int main() {
	int x = 5, y = 10;
	swap(&x, &y);
	printf("x = %d, y = %d\n", x, y); // x = 10, y = 5
	return 0;
	}
	Explanation: Addresses passed using &; inside swap, *a and *b access original variables.
	Marking Scheme:
	• Explanation of call by value — 2 marks
	• Explanation of call by reference — 2 marks
	• Correct swap program using pointers — 4 marks
	• Correct demonstration/sample output — 2 marks	

B.	**Write a program in C to print following pattern. (10 marks)**	

a)
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
b)
* * * * *
* * * *
* * *
* *
*

Answer (Explanation & Sample Code):

- Use nested loops. Outer loop controls rows; inner loop prints column items for each row.

Pattern (a) sample code:

```
for(int i = 1; i <= 5; i++) {
   for(int j = 1; j <= i; j++)
      printf("%d ", j);
   printf("\n");
}
```

Pattern (b) sample code:

```
for(int i = 5; i >= 1; i--) {
   for(int j = 1; j <= i; j++)
      printf("*  ");
   printf("\n");
}
```

Marking Scheme:

- Pattern (a) logic & correct nested loops — 5 marks
- Pattern (b) logic & correct nested loops — 5 marks

(NB: spacing/formatting considered in awarding full marks)

C. **(i) Write a program to read a text file and count number of lines. (ii) Explain fopen(), fclose(), and fgets(). (10 marks)**

Answer (Program & Explanation):

(i) Program to count lines using fgets:

```
#include <stdio.h>

int main() {
   FILE *fp = fopen("input.txt", "r");
   if (fp == NULL) {
      printf("Cannot open file\n");
      return 1;
   }
   char buf[256];
   int lines = 0;
   while (fgets(buf, sizeof(buf), fp) != NULL)
      lines++;
   fclose(fp);
   printf("Lines = %d\n", lines);
```

	return 0; } **(ii) Explanation of functions:** • fopen(filename, mode) — opens the file in specified mode; returns FILE *; returns NULL on failure. • fclose(fp) — closes opened file, flushes buffers, returns 0 on success. • fgets(str, n, fp) — reads at most n-1 characters into str or until newline/EOF; returns str on success, NULL on EOF/error. Safe for line-based reading with buffer size control. **Marking Scheme:** • Correct program handling fopen failure & counting lines — 6 marks • Clear explanation of fopen, fclose, fgets — 4 marks	
D.	**What is control loop? Explain all the control loops with example. (10 marks)** **Answer (Explanation & Examples):** • **Control loops** repeat a block of statements until a condition fails. Main loops in C: for, while, do-while. o **for loop:** used when iteration count is known or controlled. Syntax: for(init; condition; update) { /* body */ } ▪ Example: for(int i=0; i<5; i++) printf("%d", i); o **while loop:** entry-controlled; checks condition before each iteration. ▪ Example: int i=0; while(i<5) { printf("%d", i); i++; } o **do-while loop:** exit-controlled; body executes at least once; condition checked after body. ▪ Example: int i=0; do { printf("%d", i); i++; } while(i<5); **Use-case note:** choose for for counted loops, while when pre-condition needed, do-while when body must run once. **Marking Scheme:** • Explanation & example for for — 3 marks • Explanation & example for while — 3 marks • Explanation & example for do-while — 3 marks • One short remark on differences/use-cases — 1 mark	

E.	**. What is the need of structure. Write a program to calculate area using structure with members (length, breadth). (10 marks)**
	Answer (Explanation & Program):
	• **Need:** Structures group related variables (possibly different types) under a single name, representing records or composite types (e.g., student, rectangle). Improves code organization, readability, and easier parameter passing.
	Sample program:
	#include <stdio.h>
	
	struct Rectangle {
	double length;
	double breadth;
	};
	
	int main() {
	struct Rectangle r;
	r.length = 5.0;
	r.breadth = 3.0;	

	double area = r.length * r.breadth;
	printf("Area = %.2f\n", area);
	return 0;
	}
	Explanation: Use dot operator for member access. Structures can be passed by value or by pointer for efficiency.
	Marking Scheme:
	• Explanation of need for structure — 3 marks
	• Correct program with structure, assignment, area calculation — 5 marks
	• Correct output/sample demonstration — 2 marks	

Q.3	Attempt any Three. (All questions carry equal marks)	
A.	**Explain the data types in C with suitable examples. (05 marks)** **Answer (Explanation & Examples):** • **Basic types:** int (integer), char (character), float, double, void. • **Derived types:** arrays, pointers, functions returning types, references via pointers. • **User-defined:** struct, union, enum, typedef. • **Examples:** int a = 10; char c = 'A'; float f = 3.14f; double d = 3.14159; **Marking Scheme:** • Listing and brief explanation of basic and derived types with examples — 5 marks	
B.	**Short note on Dynamic memory allocation. (05 marks)** **Answer (Explanation):** • Dynamic memory allocation uses malloc, calloc, realloc, and free to allocate and manage memory at runtime. Necessary when size is unknown at compile time. Always check returned pointer for NULL. After usage, free() must be called to release memory and avoid leaks. realloc() adjusts size of allocated block (may move memory). **Marking Scheme:** • Explanation of functions and good practices (NULL check, free) — 5 marks	

C.	**. Differentiate between break and continue with examples. (05 marks)**
	Answer (Explanation & Examples):
	• **break:** Immediately exit the nearest enclosing loop or switch.
	Example:
	for(i=0;i<5;i++){
	if(i==3) break; // loop ends when i==3
	}
	continue: Skip remainder of current loop iteration and proceed to next iteration.
	Example:
	• for(i=0;i<5;i++){
	• if(i==3) continue; // skips actions for i==3
	• printf("%d", i);
	• }
	•
	Marking Scheme:
	• Definition of break and continue — 3 marks
	• Correct examples demonstrating difference — 2 marks	

D.	**Explain the difference between Call by value and call by reference? (05 marks)** **Answer (Explanation):** • **Call by value:** Copies of arguments are passed; modifications inside function do not change originals. • **Call by reference:** Addresses (pointers) are passed; function can modify original variables using dereference. • Brief example: Swap implemented using call by value will not swap caller values; swap using pointers (call by reference) will. **Marking Scheme:** • Explanation of call by value — 2 marks • Explanation of call by reference — 2 marks • Short illustrative remark/example — 1 mark	