Reg. No. .....

Indian Institute of Information Technology, Surat
Department of Computer Science & Engineering

End Semester Examination 1st Sem, 2025-26

Fundamentals of Computer and C Programming (CS101)

Timing: 10:00 AM - 1:00 PM **Date: 1 Dec 2025** **Max Marks: 100**

Instructions: Attempt all parts of a question at one place. If you do not, later parts may not be graded. **This exam has 3 printed pages**. Please ensure that you have received all three.

1. State whether the following statements are **true** or **false**. If the statement is false then **write the correct statement.** **(8x2=16)**

Grading scheme: Zero for wrong answer. 1 for a correct answer without or with wrong explanation. 2 for the correct answer with the correct explanation.

a. Writing comments in a C program increases its runtime.
False. Comments are ignored by the compiler when it produces the executable.

b. The format specification `"%4s"` prints the first `4` characters of the given string .
False. The format specification `"%4s"` pads the given string so that at least `4` characters are printed.

c. In von Neumann architecture, there are separate memories for instructions and data.
False. In von Neumann architecture there is single memory for instructions and data.

d. For an unsigned integer `x`, the expression `x&111` is identical to `x%8`.
False. For an unsigned integer `x`, the expression `x&7` (or `x&0b111`) is identical to `x%8`.

e. Global variables are accessible by all functions of the program.
True.

f. Memory allocated by `malloc` inside a function gets freed when the function returns.
False. Memory allocated by `malloc` is allocated on heap which needs to be explicitly freed using `free()` function.

g. `typedef int* IP; IP ptr` defines a variable `ptr` that is a pointer to integers.
True.

h. If `fopen()` fails to open a file, it returns the `EOF` (End-Of-File) macro.
False. If `fopen()` fails to open a file, it returns the `NULL` macro/pointer.

2. Choose the single correct answer for each of the following questions. **(8x2=16)**

Grading scheme: Boolean. Either zero or 2.

a. What is copied into a local variable inside the function when an array is passed to a function as an argument?
i. Entire array ii. The value of the first element of the array
iii. The pointer to the first element of the array iv. The name of the array

b. Nesting is allowed for which of the following statements?
i. if ii. if-else iii. switch **iv. All of these**

c. Which of the following is true for the C language?
i. Interpreted **ii. Compiled** iii. Low level language iv. Case insensitive

d. Consider an array declared as `int arr[10][10]`. Which of the following **does not** point to the base address of this array?
i. `arr` ii. `arr[0]` **iii. `arr[0][0]`** iv. `&arr[0][0]`

e. Which constant value can be assigned to a pointer variable?
i. `0` ii. `NULL` **iii. Both i & ii** iv. None of i or ii

f. What is the return data type of the `malloc()` function?
i. `void*` ii. `int*` iii. `pointer` iv. `struct`

g. Which of the following is an exit-controlled loop?
i. `for` ii. `while` **iii. `do-while`** iv. `goto`

h. When you execute the compiled C program, where are the variables stored?
i. ROM **ii. RAM** iii. Hard Drive iv. Terminal

3. Write the output for the C code snippets below. Disregard compilation warnings, if any. There is **no** runtime error or undefined behavior. Assume standard context and 64-bit Linux environment identical to lab computers. **(4x2=8)**

Grading scheme: Boolean. Either zero or 2.

a.

```
long x = 42;
long *y = &x;
int inc_x = (long)(x+1) - (long)x;
int inc_y = (long)(y+1) - (long)y;
printf("inc_x=%d ", inc_x);
printf("inc_y=%d ", inc_y);
```

```
inc_x=1 inc_y=8
```

b.

```
int x = 10;
int count = 0;
count += (x==10 && x>10 && !x);
count += (x==10 && x<10 || !x);
count += (x==10 && x>10 || !x);
count += (x==10 || x>10 || !x);
printf("x=%d count=%d", x, count);
```

```
x=10 count=1
```

c.

```
int arr1[5][20];
int* arr2 = (int*)malloc(sizeof(arr1));
printf("sz1=%d ", sizeof(arr1));
printf("sz2=%d ", sizeof(arr2));
```

```
sz1=400 sz2=8
```

d.

```
struct measurement {
    int val;
    char unit;
};
struct measurement h = {1, 'm'};
struct measurement w = h;
h.val = 2;
printf("h=%d%c ", h.val, h.unit);
printf("w=%d%c", w.val, w.unit);
```

```
h=2m w=1m
```

4. For **any three** of the C code snippets below, classify it as compilation error, runtime error, or undefined behavior. Then, **fix and rewrite** the corrected code snippet assuming standard context. **(3x5=15)**

Grading scheme: 2 points for correct classification. 3 points for completely correct fix.

a. `char wish[10] = "good luck!"; //String wish should store “good luck!”`
Undefined Behaviour.
`char wish[] = "good luck!"; OR char wish[11] = "good luck!";`

a. `/* Strings prefix, code, and course should store "CS", "101", and "CS101" respectively. String course should be derived from prefix and code. */`

```
char *prefix = "CS"; char code[] = "101";
char course[101] = prefix + code;
```

Compilation Error.

```
char *prefix = "CS"; char code[] = "101"; char course[101];
strcpy(course, prefix); strcat(course, code);
```

b. `int num1, num2;  int* ptr1 = num1, ptr2 = num2; /* ptr1 and ptr2 should store pointers to num1 and num2 respectively. */`

Compilation Error / Undefined behaviour.

```
int num1, num2;  int* ptr1 = &num1, *ptr2 = &num2; //Note 3 fixes
```

c. `int even_primes[1]; even_primes = {2}; /* even_primes is an array of length one and should store a single element: number 2. */`

Compilation Error.

```
int even_primes[] = {2}; OR int even_primes[1]; even_primes[0]=2;
```

5. Differentiate between **any three** of the following: **(3x5=15)**

Grading scheme: In parts a, b, and c, 3 points for correct theory and 2 points for completely correct code/computation. No partial points for incorrect code/computation. In part d, 2 points for correct description of changing variable inside function and 3 points for other theory. Marks should be integral.

a. Function declaration and function definition. Give a code example where a function declaration is strictly necessary for the successful compilation.

Function declaration informs the compiler of the function name, return type, and types of parameters. It has no function body. Function definition contains function name, return type, types of parameters along with function body. Function declaration is mandatory when function is called before its definition. Example where function declaration is necessary:

```c
#include <stdio.h>
int is_even(int n); // Declaration strictly necessary
int main() {
    if(is_even(42)) printf("Even");
    return 0;
}
// Definition occurs after main
int is_even(int n) {
    return n % 2 == 0;
}
```

b. Explain the 'structure' declaration and their initialization in C. Demonstrate through a coding example how the elements of a structure are accessed.

A struct is a user-defined data type that groups related variables (called fields) of potentially different data types under a single name. Initialization is done either using curly braces `{}` assigning values in the order of member declaration or specifying names of the underlying fields.

```c
#include <stdio.h>
struct Point {
    int x;
    int y;
```

```
};
int main() {
    struct Point p1 = {10, 20}; //Initialization in order of fields
    struct Point p2 = {.y=20, .x=10};// Initialization by field names
    //Accessing elements using dot operator
    printf("Point p1: (%d,%d)\n", p1.x, p1.y);
    // Accessing elements using arrow operator
    printf("Point p2: (%d,%d)", (&p2)->x, (&p2)->y);
    return 0;
}
```

c. 1's Complement and 2's Complement conventions to store negative integers. State the representation of **-22** as a **32**-bit signed integer using 2's complement notation.

1's Complement of a negative integer is obtained by inverting all bits. It suffers from having two representations for zero. 2's Complement of a negative integer is obtained by first taking its 1's Complement and then adding 1. It has a unique representation for zero and simplifies arithmetic circuits.

The 8-bit representation of integer 22 is `0001 0110`, or simply `16` in hex. Hence, the 32-bit representation of -22 is `ff ff ff ea`. This translates to `1111 1111 1111 1111 1111 1111 1110 1010` in binary.

d. Pass-by-Value and Pass-by-Reference for function arguments. State in one line how changes made inside the function affect the actual arguments in the caller function.

In Pass-by-Value, the function receives a copy of the actual argument's value. The memory location of the local variable inside the function is distinct from that of the actual argument. In Pass-by-Reference, the function receives an address (reference) of the original variable, meaning both the formal parameter and the actual argument refer to the same memory location. Any changes made to the parameter inside the function only affect the copy and are destroyed when the function finishes execution. The original variable remains unchanged. Any changes made to the parameter inside the function directly modify the original variable in the calling function.

6. Write complete C programs for **any two** of the following. You should properly indent and add comments in your code. **(2x15=30)**

Grading scheme: For completely incorrect logic, cap marks at 5 points. 2 points for indentation and 3 points for comments. Deduct 1 point each for missing semicolon, braces, includes, and wrong loop bounds. Deduct 2 points each for an infinite loop and accessing out of bounds memory.

a. Read a line of text (string) from the user, followed by a single character to be searched for. You may assume the string will not exceed `1000` characters. Implement a function with the prototype: `void replace_char(char *str, char c);` It should use pointers to iterate through `str` and replace every occurrence of `c`, if any, with an '`*`'. The main function should then display the updated string.

```
#include <stdio.h>
#include <string.h>
#define BUFF_SIZE 1002
void replace_char (char *str, char c) {
    //loop over string until it ends, i.e. encounter null delimiter
```

```
    while(*str != '\0') {
        if(*str == c) *str = '*';
        str++;
    }
}
int main() {
    char s[BUFF_SIZE], c;
    fgets(s, BUFF_SIZE, stdin);
    s[strlen(s)-1] = '\0'; //removing new line
    scanf("%c", &c);
    replace_char(s, c);
    printf("%s", s);
}
```

b. Take as input a **5x5** integer matrix as a **2**D array from the user and determine if it is symmetric. Print **Symmetric** or **Not Symmetric** accordingly. Then, find the sum of all of its eigenvalues (with multiplicity) and print it.

```
#include <stdio.h>
int main() {
    const int n=5; //Fixing n=5 for readability
    int M[n][n], sym = 1, trace=0;
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            scanf("%d", &M[i][j]);
             //If i>j, we have already scanned M[j][i]
            if(i>j && M[i][j]!=M[j][i]) sym=0;
        }
        //Recall that trace of a square matrix is the sum of its
eigenvalues including multiplicity
       trace += M[k][k];
    }
    if(sym) printf("Symmetric ");
    else printf("Not Symmetric ");
    printf("Trace=%d\n", trace);
}
```

c. The Collatz sequence (also known as the **3n+1** problem) for any positive integer **n** is as follows: If **n** is **1**, the sequence ends. If **n** is even, the next term is **n/2**. If **n** is odd, the next term is **3n+1**. Implement a recursive function **void collatz(int n)** that takes input an integer **n** and prints the corresponding Collatz sequence. For example, the Collatz sequence for **n=10** is: **10 5 16 8 4 2 1**. You may assume that the input **n** is a positive **32**-bit **int** chosen such that the sequence is finite and its maximum term is less than 2^{31}. Write and call this recursive function **collatz()** from the **main** function with parameter **n=2025**.

```
#include <stdio.h>
void collatz(int n) {
    printf("%d ", n);
    if(n == 1) return; //Base case
    else if(n%2) collatz(3*n+1); //Odd recursive branch
    else collatz(n/2); //Even recursive branch
}
int main() {
    collatz(2025);
    return 0;
}
```