Adv C Module

1. Subjective

2.1 Basic Refreshers

1. What is the difference between post-increment and pre-increment?
2. What will be the output of the following code snippet?

```
#include <stdio.h>
void solve() {
    char ch[10] = "abcdefghij";
    int ans = 0;
    for (int i = 0; i < 10; i++) {
        ans += (ch[i] - 'a');
    }
    printf("%d", ans);
}
int main() {
    solve();
    return 0;
}
```

3. What will be the output of the following code snippet?

```
#include <stdio.h>
void main() {
    char c[] = "GATE2011";
    char *p = c;
    printf("%s", p + p[3] - p[1]);
}
```

4. int x = 5; printf("%d%d", x++, ++x); — What will be the output?
5. Explain modifiers and qualifiers in C.
6. Explain bitwise operators with examples.

2.2 1D Pointers and Functions

1. What are pointers? Explain with an example.
2. What is pass-by-value, and give an example of pass-by-reference.

2.3 Storage classes and memory segments

1. What is a static variable with an example?
2. What is the difference between a local and a static variable?
3. What are storage classes in C? Explain in detail.
4. What is a static function?
5. What are global variables?

2.4 2D Pointers and DMA

1. Explain dynamic memory allocation and the difference between malloc and calloc.
2. Explain memory management in C.
3. What is a dangling pointer?
4. What is a wild pointer?
5. What are the .bss and .data segments in memory?
6. Why is the .data segment divided into initialized and uninitialized sections?
7. When is memory allocated for initialized and uninitialized variables?

2.5 Preprocessing

1. What happens during the compilation stages in C?
2. Explain the stages of compilation in C.

2.6 UDT

1. What is the difference between structures and unions?
2. How to define a structure?
3. What is the difference between a macro and a function in C?

2.7 Miscellaneous

1. What is the purpose of the const keyword in C?
2. What is the purpose of the volatile keyword in C?

2.8 Projects

1. Explain an MP3 tag reader. How are MP3 files compressed? Mention any audio file formats other than .mp3.
2. Where is steganography used, and where is cryptography used?

2.9 FILE I/O

1. Explain fseek, ftell, fopen, and fclose.

2. Programming

1. Write a program to print the GCD of two numbers.
2. Write a program to generate random variables in C.
3. Write a program for pass by value and pass by reference.
4. Write a structure with different data types, find its size, explain how to avoid padding, and show its size without padding.
5. Write a program to print random numbers.
6. Write a program to find the factorial of a number using recursion.
7. Write a program to check whether a number is prime.
8. Write a program to print a spiral matrix pattern.
9. Write a program to find a word in a sentence and replace it with another given word.
10. Write a program to print all prime numbers up to a given range and optimize it for better performance.
11. Write a recursion program to reverse a string.
12. Write a program to swap the 3rd and 5th bits of a number.
13. Write a program to check if the bits of a number form a palindrome.

MC Module

1. Basic electronics

1. Explain the Class C amplifier.
2. Threshold voltage increases with which parameters?
3. What is the application of a Schmitt trigger?
4. What are the advantages of CMOS over TTL?
5. Give the Boolean expressions for NAND, OR, and NOT gates.
6. Write the equation of cutoff frequency in a pure capacitive circuit.
7. Explain the working and applications of an Op-amp.
8. What is phase difference? Explain with an example.
9. What is the difference between avalanche breakdown and zener breakdown?
10. What are SCR and TRIAC in a thyristor? Explain their working and applications.

2. Basics

1. What is the difference between a microcontroller and a microprocessor?
2. What is device driver programming?
3. Why did you choose Microcontroller (MC) over Microprocessor (MP)?
4. Which application would you choose for Microcontroller, and why?
5. What is a watchdog timer? Explain its use in embedded systems.
6. Can we use a Microprocessor in place of a Microcontroller? Why or why not?

3. Interrupts

1. What are interrupts? Explain their types.
2. What is the difference between timers and counters?
3. What is the main purpose of timers?
4. Explain synchronous timers and asynchronous timers.
5. How do you create a delay using a timer?
6. What is timer overflow?
7. What is the difference between polling and interrupt?

4. Projects

1. Explain the car black box project.
2. Explain the ‘Pick-to-Light’ system.
3. How is CAN protocol used in a Pick-to-Light system?
4. Explain the CAN-based automotive dashboard.

5. ADC

1. What is ADC and how do you configure ADC?

6. Protocols

1. What is the difference between I^2C and SPI protocols?
2. Explain the difference between SPI and UART.
3. Explain the I^2C protocol with a frame format.
4. Explain the SPI protocol with a frame format.
5. Explain the UART protocol with a frame format.
6. Explain the CAN protocol with a frame format.
7. What is the maximum number of nodes that can communicate using I^2C?
8. What is baud rate?

9. How does the UART protocol work?
10. If the baud rate is 9600 and we need to send 4 bytes of data using the UART protocol, what will be the data transmission speed?
11. How do you communicate with an EEPROM using the I²C protocol?
12. What is the maximum data transfer distance for UART, and why is it limited?
13. What is the highest baud rate that can be used in UART? What happens if we exceed this baud rate?

7. Programming

1. Write a program to toggle an LED in Port 2, Pin 0 with a delay of 500 ms.
2. Write a program to blink 3 LEDs in sequence with a delay of 200 ms between each.
3. Write a program to count the number of 1s in an 8-bit number given by the user.
4. Write a program to configure a GPIO pin for controlling an LED.
5. Human Detection via Radar: Signal Processing and Event Reporting
   Problem Statement:
   Write a C program that:
   - Reads a sequence of simulated radar sensor values (from a file or generated data).
   - Applies a simple threshold or moving average algorithm to detect "human presence" events.
   - Log detection events with timestamps.
   - (Bonus) Formats detection events as JSON or as a binary packet for transmission.

   General Requirements:
   - The program must read or generate simulated radar sensor values.
   - Implement a simple threshold or moving average algorithm to detect human presence.
   - Log detection events with timestamps.
   - (Bonus) Format detection events as JSON or as a binary packet.

   Inputs :
   Your program can either:

- Read from a file: A text file containing simulated radar sensor values (e.g., floating-point numbers representing signal strength), one value per line.
- Generate data internally: Use random values or a predefined sequence to simulate radar readings.

Sample Input File (radar_data.txt):

```
0.1
0.2
0.15
0.3
0.4
0.25
0.35
0.5
0.45
0.2
```

Test Cases

These test cases will help you verify your program's correctness under various conditions.

1. Normal Case
    - Description: Read from a file and detect events above the threshold.
    - Input: radar_data.txt with values: 0.1, 0.2, 0.15, 0.3, 0.4, 0.25, 0.35, 0.5, 0.45, 0.2

Expected Output:

```
[timestamp] Human presence detected (for 0.4)
{"timestamp": "[timestamp]", "event": "Human presence
detected"}
[timestamp] Human presence detected (for 0.35)
{"timestamp": "[timestamp]", "event": "Human presence
detected"}
[timestamp] Human presence detected (for 0.5)
{"timestamp": "[timestamp]", "event": "Human presence
detected"}
[timestamp] Human presence detected (for 0.45)
{"timestamp": "[timestamp]", "event": "Human presence
detected"}
```

        - Verification: Events logged only for values > 0.3 (i.e., 0.4, 0.35, 0.5, and 0.45).
    2. No Detection Case
        - Description: All values are below the threshold.
        - Input: radar_data.txt with values: 0.1, 0.2, 0.15, 0.25
        - Expected Output: No output (no detections).
        - Verification: No log entries generated.
    3. Moving Average Algorithm
        - Description: Use a moving average instead of a simple threshold (alternative detection method).
        - Input: radar_data.txt with values: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
        - Expected Output: Detect when the moving average exceeds the threshold (e.g., 0.3).
        - Verification: Implement moving average logic (e.g., window size of 3) and check detections.
    4. Error Case - File Not Found
        - Description: Simulate a failure to read the input file.
        - Input: Non-existent file (e.g., radar_data.txt missing).
        - Expected Output: Error message, e.g., Error opening file: No such file or directory.
        - Verification: Program exits gracefully with an error message.
    5. Bonus Case - Binary Packet
        - Description: Format events as binary packets (optional bonus).
        - Input: Same as Normal Case.
        - Expected Output: Binary representation of events (e.g., timestamp and event code in bytes).
        - Verification: Correct binary formatting (requires additional implementation).

6. Vehicle Parking Automation: Occupancy Status and Reporting
   Problem Statement:
   Create a C program that simulates a parking system with multiple spots. The program should:
        - Simulate occupancy sensor readings for each spot, either randomly toggled or read from a file.

- Maintain and update the current status (occupied or vacant) of each spot, thereby tracking occupancy.
- Detect entry and exit events: an entry occurs when a spot changes from vacant to occupied, and an exit occurs when it changes from occupied to vacant.
- On each status change, generate a report payload that includes the spot ID, event type (entry or exit), and timestamp, and print it.
- (Bonus) Maintain daily counts of parking events (entries) per spot.

General Requirements:

- Simulate occupancy sensor readings for multiple parking spots.
- Maintain and update the status of each parking spot.
- Generate a report payload on each status change and print it.
- (Bonus) Maintain daily counts of parking events per spot.

Inputs

Your program can accept inputs in one of two ways:

- Randomly Generated Data: For each parking spot, randomly toggle the occupancy status (0 for vacant, 1 for occupied) at each time step to simulate sensor readings.
- File Input: Read from a text file containing simulated occupancy data for multiple spots over time. Each line
- represents a time step, with space-separated values (0 or 1) indicating the status of each spot.

Sample Input File

Below is an example of a file named parking_data.txt:

```
# Time step 1: All spots vacant
0 0 0
# Time step 2: Spot 1 and Spot 3 occupied
1 0 1
# Time step 3: Spot 1 and Spot 2 occupied
1 1 0
# Time step 4: Only Spot 2 occupied
```

```
0 1 0
```

Outputs

Your program should produce the following outputs:

Report Payloads:

- For each status change (entry or exit), print a report payload with the spot ID, event type (Entry or Exit), and a timestamp.

```
Spot 1: Entry at 2023-10-01 12:00:05
Spot 3: Entry at 2023-10-01 12:00:05
Spot 2: Entry at 2023-10-01 12:00:10
Spot 1: Exit at 2023-10-01 12:00:15
Spot 3: Exit at 2023-10-01 12:00:15
```

Bonus: Daily Counts (Optional):

- Track and print the number of entries per spot for the day.

```
Spot 1: 1 entry today
Spot 2: 1 entry today
Spot 3: 1 entry today
```

Test Cases

Below are several test cases to verify your program's functionality under different scenarios.

Test Case 1: Normal Operation

Description: Simulate a typical sequence of parking events from a file.

Input File (parking_data.txt):

```
0 0 0
1 0 1
1 1 0
0 1 0
```

Expected Output:

```
Spot 1: Entry at [timestamp]
Spot 3: Entry at [timestamp]
Spot 2: Entry at [timestamp]
Spot 1: Exit at [timestamp]
Spot 3: Exit at [timestamp]
Spot 1: 1 entry today
Spot 2: 1 entry today
Spot 3: 1 entry today
```

    - Verification: Check that entry/exit events are detected correctly and daily counts (if implemented) are accurate.

Test Case 2: No Status Changes

Description: All spots remain in the same state across time steps.

Input File (parking_data.txt):

```
0 0 0
0 0 0
0 0 0
```

Expected Output: (No output since there are no status changes)

- Verification: Ensure the program does not generate any report payloads when statuses remain unchanged.

Test Case 3: Multiple Events on One Spot

Description: A single spot experiences multiple entries and exits.

Input File (parking_data.txt):

```
0 0 0
1 0 0
0 0 0
1 0 0
```

Expected Output:

```
Spot 1: Entry at [timestamp]
Spot 1: Exit at [timestamp]
Spot 1: Entry at [timestamp]
Spot 1: 2 entries today
Spot 2: 0 entries today
Spot 3: 0 entries today
```

    - Verification: Confirm that multiple events for the same spot are handled correctly, and daily counts reflect all entries.

Test Case 4: File Not Found Error

Description: Simulate a failure to locate the input file.

Input: Attempt to read from a non-existent file (e.g., parking_data.txt is missing).

Expected Output:

```
Error opening file: No such file or directory
```

        - Verification: Ensure the program exits gracefully with an appropriate error message.

Test Case 5: Bonus - Daily Counts Reset (Optional)

Description: Simulate a new day where daily counts should reset (requires additional implementation).

Input File (parking_data.txt):

```
0 0 0
1 0 1
```

Expected Output (after simulating a new day):

```
Spot 1: Entry at [timestamp]
Spot 3: Entry at [timestamp]
Spot 1: 1 entry today
Spot 2: 0 entries today
Spot 3: 1 entry today
[New day starts]
Spot 1: 0 entries today
Spot 2: 0 entries today
Spot 3: 0 entries today
```

- Verification: Validate that daily counts reset correctly at the start of a new day.

7. Smart Parking Simulation Using FSM for Radar and Camera Modules

   Problem Statement

   - You are tasked with developing a simulation for a simplified embedded parking slot management system that uses state machines to detect vehicle presence and calculate parking duration. This project is designed to test your ability to implement modular Finite State Machines (FSMs) in a simulated sensor-driven embedded environment using C programming.

   The simulation must:

   - Read input from a predefined log file car_info.txt.
   - Detects the presence of cars using a simulated radar FSM.
   - Recognize the vehicle using a camera FSM.
   - Calculate the total parking time based on entry/exit timestamps.
   - Log the result to output.txt.

General Requirements

- Implement two separate FSMs:
    - Radar FSM in radar.c – handles car detection, state transitions, and final output writing.
    - Camera FSM in camera.c – handles plate recognition and returns it to the Radar FSM.
- Use the provided car_info.txt input file, which contains car entry/exit logs with plate numbers.
- The FSMs must simulate sensor interaction without actual hardware or delays.
- Write the output to a new file output.txt showing the vehicle number and parking duration.

Inputs

The program reads from a predefined log file named car_info.txt, which contains parking event logs. Each line represents a single parking event with the following format:

- [Entry Time] [Plate Number] [Exit Time]
    - Entry Time: Format HH:MM (24-hour clock, e.g., 08:00).
    - Plate Number: Alphanumeric string (e.g., KA01AB1234).
    - Exit Time: Format HH:MM (24-hour clock, e.g., 10:15).

Sample Input File (car_info.txt)

```
08:00 KA01AB1234 10:15
09:30 MH02CD5678 11:00
12:00 TN03EF9012 12:45
```

Outputs

The program writes to a file named output.txt with the following format for each parking event:

- [Plate Number] [Duration in HH:MM]
    - Plate Number: Matches the input file's plate number.
    - Duration: Calculated as Exit Time - Entry Time, formatted as HH:MM.

Sample Output File (output.txt)

```
KA01AB1234 02:15
MH02CD5678 01:30
TN03EF9012 00:45
```

CPP Module

1. What is the difference between C and C++?
2. What are the main features of C++?
3. What is an inline function?
4. Explain object, class, inheritance, and the four pillars of OOP.
5. What is polymorphism in OOP?

Ds Module

1. Basics

1. Explain time complexity and space complexity with examples.
2. What are the different types of data structures?

2. Linked lists

1. What is a linked list, and write a C program to allocate memory for a linked list node.
2. Explain a singly linked list and write a program to implement it.
3. In a linked list, how do you delete the last node?
4. Explain a doubly linked list.
5. Write a program to insert a node in a linked list.
6. Write the structure declaration for a linked list.
7. Write a program to find the second largest number in a linked list.
8. Write a program to count the number of vowels and consonants in a string.
9. Write a program to generate the Fibonacci series using recursion.

3. Stack

1. Explain the stack implementation (using an array or linked list).
2. Explain the stack data structure.

4. Searching and Sorting Techniques

1. What are common sorting algorithms? Explain their time and space complexities.

5. Queue

1. Explain stack and queue data structures.
2. What is a circular queue?

6. Hashing

1. What is a hash table? How does it work?

7. Project

1. Explain the inverted search project.
2. Explain the APC project.

8. Trees

1. Write a structure for a binary tree and create a center node with a left node.
2. What is the difference between a binary tree and a binary search tree?

LI Module

1. Basics

1. What is the OS kernel?
2. What is the difference between a hard link and a soft link in Linux?
3. What are static linking and dynamic linking?

2. System call

1. What are system calls?
2. Explain the wait system call.

3. Process

1. What is a process in an operating system?
2. What is a Process Control Block (PCB)?
3. What is process priority in scheduling?
4. Explain process scheduling algorithms.
5. What is context switching?
6. How does the fork system call work in operating systems?
7. What is a zombie process?

4. IPC

1. What is Inter-Process Communication (IPC)? Explain different IPC mechanisms.
2. What are pipelines in operating systems?
3. What is shared memory in IPC?

5. Signal

1. What are signals, and what is a signal handler?
2. What is a signal in operating systems?

6. Threads

1. What is a deadlock? Can you give a real-time example?
2. Explain semaphore and mutex. How do they differ?
3. What is the difference between a process and a thread?
4. Explain multithreaded programming to print even and odd numbers.
5. What are thread synchronization mechanisms?
6. What is a race condition?
7. How are locks and unlocks used in a mutex?

General Questions
Aptitude

1. How many times will the hour and minute hands overlap, and at what times?
2. In a room with 23 people, what is the probability that all of them share the exact same birthday?
3. Given the series 2, 6, 12, 20, what is the next number?
4. Given the series 1, 4, 9, 25, what is the next number?
5. If a man sells an object for ₹450 with a 10% loss, what is the cost price of the object?
6. How can you measure exactly 4 liters of water using a 5-liter mug and a 3-liter mug with an unlimited water supply?
7. What is the difference between ReactJS and AngularJS?
8. Write the syntax for opening a web server on port 3000.
9. Design a parking lot system using structures with the following details:
    - Slot number
    - Vehicle type
    - Plate number

    Use a linked list for dynamic memory allocation.
10. Design a C program to simulate an asset tracking device that collects sensor data and prepares it for transmission to a server. The program

should generate temperature and GPS location data, use a appropriateAPI to package the data into a compact payload, and display the payload. Additionally, the program should demonstrate the ability to process server responses to adjust its operation, such as updating the reporting interval.
General Requirements

- Generate random temperature data and GPS coordinates (latitude, longitude).
- Use the appropriate API (via static library and header file) to package the temperature and GPS data into a compact binary or hex-encoded payload suitable for transmission.
- Print the payload as a hex-encoded string to simulate transmission to a server.
- (Bonus) Implement a function that uses the appropriate API to parse a hex-encoded downlink message and extract a new reporting interval (in seconds), then print the updated interval.

Inputs

The program does not require user input since it generates random data and simulates server communication. However, the following are
considered "inputs" in the context of the simulation:

1. Random Sensor Data (Generated Internally):
    - Temperature: A floating-point value between -20.0°C and 50.0°C.
    - Latitude: A floating-point value between -90.0 and 90.0 degrees.
    - Longitude: A floating-point value between -180.0 and 180.0 degrees.
2. Downlink Message (Bonus, Simulated):
    - A hex-encoded string (e.g., "01020304") received from the server, which the API decodes to extract a new reporting interval in seconds.

Outputs

The program should produce the following outputs:

1. Hex-Encoded Payload:
    - A string representing the binary payload (generated by the API) in hexadecimal format, printed to the console.

        - Example: Payload: 1A2B3C4D5E6F
    2. Updated Reporting Interval (Bonus):
        - An integer value (in seconds) extracted from the downlink message, printed to the console.
        - Example: Updated reporting interval: 60 seconds

Test Cases

1. Normal Run
    - Input: Random temp (-20.0–50.0°C), lat (-90.0–90.0), lon (-180.0–180.0); hex "01020304" (60s).
    - Output: Hex payload; interval = 60s.
    - Check: Valid hex; interval matches.
2. Data Limits
    - Input: Min (-20.0°C, -90.0, -180.0); Max (50.0°C, 90.0, 180.0); hex "000000FF" (255s).
    - Output: Payloads for both; interval = 255s.
    - Check: Within ranges; correct payload & interval.
3. API Fail
    - Input: Random data; bad hex "INVALID".
    - Output: Error msg; non-zero exit.
    - Check: Graceful handling.
4. Empty Payload
    - Input: Random data; hex "0102" (30s).
    - Output: Empty hex; interval = 30s.
    - Check: Empty string; downlink works.
5. Interval Variety (Bonus)
    - Input: Random data; hex "00000001" (1s), "00000384" (900s).
    - Output: Payload; intervals = 1s, 900s.
    - Check: Correct intervals printed.