COMP1511 Mock Exam - Student Work Assessment

Question 1

Original Question:

You are given a linked list of integers. Write a function that finds and returns the second largest value in the linked list. If there are fewer than 2 unique values, return -1.

```c
struct node {
    int data;
    struct node *next;
};

int second_largest(struct node *head) {
    // Your code here
}
```

Example:

- List: 5 → 2 → 8 → 2 → 1 → 8
- Output: 5 (largest is 8, second largest is 5)

Student Work with Assessment Comments:

```c
int second_largest(struct node *head) {
    // CORRECT: Good edge case handling
    if (head == NULL || head->next == NULL) {
        return -1;
    }

    // INCORRECT: Limiting initialization - should use INT_MIN
    int first = head->data;

    // CRITICAL ERROR: Uninitialized variable contains garbage values
    int second;

    struct node *current = head;
    while (current != NULL) {
        // LOGIC ERROR: Only updates second when finding new maximum
        // Missing case for values between current first and second
        if (current->data > first) {
            second = first;
            first = current->data;
        }
        // MISSING: else if condition for updating second

        current = current->next;
    }

    // ERROR: Returning potentially uninitialized variable
    return second;
}
```

Major Issues:

- Uninitialized `second` variable
- Flawed logic that misses valid second largest values
- No handling for duplicate values

Sample Answer:

```
int second_largest(struct node *head) {
    if (head == NULL || head->next == NULL) {
        return -1;
    }

    int first = INT_MIN;
    int second = INT_MIN;

    struct node *current = head;
    while (current != NULL) {
        if (current->data > first) {
            second = first;
            first = current->data;
        } else if (current->data > second && current->data != first) {
            second = current->data;
        }
        current = current->next;
    }

    return (second == INT_MIN) ? -1 : second;
}
```

Question 2

Original Question:

You are given an array of structs representing student marks. Write a function that counts how many students have a mark above the class average.

```
struct student {
    int mark;
```

```c
    char name[50];
};

int above_average_count(int size, struct student students[]) {
    // Your code here
}
```

Example:

- Students with marks: [85, 72, 90, 65, 78]
- Average: 78, Students above average: 3

Student Work with Assessment Comments:

```c
// SYNTAX ERROR: Extra space in function name
int above_average _count(int size, struct student students[]) {
    // CRITICAL ERROR: Uninitialized variable contains garbage
    int total;

    // SYNTAX ERROR: Comma instead of semicolon
    for (int i = 0; i < size,; i++) {
        // COMPILATION ERROR: Capital I instead of lowercase i
        total += students[I].mark;
    }

    // CORRECT: Using float for average
    float average = total / size;

    int count = 0;

    // SYNTAX ERROR: Same comma/semicolon mistake
    for (int i = 0; i < size,; i++) {
        // CORRECT: Logic is right here
        if (students[i].mark > average) {
            count++;
        }
    }
    return count;
}
```

Major Issues:

- Multiple syntax errors preventing compilation
- Uninitialized `total` variable
- Variable name error (I vs i)

Sample Answer:

```
int above_average_count(int size, struct student students[]) {
    int total = 0;

    for (int i = 0; i < size; i++) {
        total += students[i].mark;
    }

    float average = (float)total / size;

    int count = 0;
    for (int i = 0; i < size; i++) {
        if (students[i].mark > average) {
            count++;
        }
    }

    return count;
}
```

Question 3

Original Question:

Write a function that removes all nodes from a linked list that contain values divisible by a given number. The function should return the new head of the modified list. Remember to properly free the memory of removed nodes.

```
struct node {
    int data;
    struct node *next;
};

struct node *remove_divisible(struct node *head, int divisor) {
    // Your code here
}
```

Example:

- Original list: 12 → 5 → 18 → 7 → 9 → 6
- Divisor: 3
- Result: 5 → 7 (removed 12, 18, 9, 6)

Student Work with Assessment Comments:

```
struct node *remove_divisible(struct node *head, int divisor) {
    struct node *current = head;
    struct node *previous = NULL;
    while (current != NULL) {
        // CORRECT: Good division by zero check
        if (divisor == 0) {
            // TYPE ERROR: Should return struct node*, not boolean
            return false;
        }

        if (current->data % divisor == 0) {
            // CRITICAL ERROR: Segmentation fault if previous is NULL (removing head)
            previous->next = current->next;

            // CORRECT: Good memory management
            free(current);
            current = current->next;
        } else {
            // ERROR: previous pointer never updated
            current = current->next;
        }

        // CRITICAL ERROR: Return inside loop - only checks first node
        return head;
    }
}
```

Major Issues:

- Premature return inside loop
- No handling for head node removal
- Previous pointer never updated
- Wrong return type for error case

Sample Answer:

```
struct node *remove_divisible(struct node *head, int divisor) {
    if (divisor == 0) {
        return head;
    }

    // Remove from head
    while (head != NULL && head->data % divisor == 0) {
        struct node *temp = head;
        head = head->next;
        free(temp);
    }

    // Remove from rest of list
    struct node *current = head;
    while (current != NULL && current->next != NULL) {
        if (current->next->data % divisor == 0) {
            struct node *temp = current->next;
            current->next = current->next->next;
            free(temp);
        } else {
            current = current->next;
        }
    }

    return head;
}
```

**Question 4 **

Original Question:

You're given an array of integers and need to find the longest sequence of consecutive increasing numbers. Return the length of this sequence.

```
int longest_increasing_sequence(int size, int array[]) {
    // Your code here
}
```

Example:

- Array: [1, 3, 2, 3, 4, 5, 1, 2]
- Longest increasing sequence: 2, 3, 4, 5 (length = 4)

Student Work with Assessment Comments:

```
int longest_increasing_sequence(int size, int array[]) {
    // CORRECT: Good initialization
    int current_length = 1;

    // MISSING: Need max_length variable to track longest sequence

    for (int i = 1; i < size; i++) {
        // CORRECT: Good logic for detecting increasing sequence
        if (array[i] > array[i-1]) {
            current_length++;
        } else {
            // ERROR: Reset without saving maximum length
            current_length = 1;
        }
    }

    // ERROR: Returns last sequence length, not longest
    return current_length;
}
```

Major Issues:

- Missing variable to track maximum length
- Returns length of final sequence instead of longest
- No edge case handling for empty arrays

Sample Answer:

```c
int longest_increasing_sequence(int size, int array[]) {
    if (size <= 0) {
        return 0;
    }

    int current_length = 1;
    int max_length = 1;

    for (int i = 1; i < size; i++) {
        if (array[i] > array[i-1]) {
            current_length++;
        } else {
            if (current_length > max_length) {
                max_length = current_length;
            }
            current_length = 1;
        }
    }

    if (current_length > max_length) {
        max_length = current_length;
    }

    return max_length;
}
```