# Final Exam: ASD2 - model answer

**Exercise 1 (5 pts) :**

```
#include <stdio.h>

// 1. Define an enumeration named GradeCategory
enum GradeCategory {  (0.5)
    FAIL,
    PASS,            (1)
    GOOD,
    EXCELLENT
};

// 2. Define a structure named StudentModule
struct StudentModule {    (0.5)
    char moduleName[50];        (0.5)
    float score;
};

// 3. Write a function getCategory
enum GradeCategory getCategory(struct StudentModule sm) {  (0.5)

     if (sm.score < 10)         (0.5)
        return FAIL;

     if (sm.score < 12)        (0.5)
        return PASS;

     if (sm.score < 15)         (0.5)
        return GOOD;

     return EXCELLENT;       (0.5)

}

int main() {
    // Test cases for Exercise 1 : this is optional
```

```
    struct StudentModule student1 = {"Math", 9.5};
    struct StudentModule student2 = {"Physics", 16.0};


    printf("Module: %s, Score: %.1f, Category: ",
student1.moduleName, student1.score);
    switch (getCategory(student1)) {
        case FAIL: printf("FAIL\n"); break;
        case PASS: printf("PASS\n"); break;
        case GOOD: printf("GOOD\n"); break;
        case EXCELLENT: printf("EXCELLENT\n"); break;
    }

    printf("Module: %s, Score: %.1f, Category: ",
student2.moduleName, student2.score);
    switch (getCategory(student2)) {
        case FAIL: printf("FAIL\n"); break;
        case PASS: printf("PASS\n"); break;
        case GOOD: printf("GOOD\n"); break;
        case EXCELLENT: printf("EXCELLENT\n"); break;
    }

    return 0;
}
```

**Exercise 2 (5 pts) :**

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *inputFile, *evenFile, *oddFile;      (0.25)
    int number;   (0.25)
    int evenCount = 0;    (0.25)
    int oddCount = 0;   (0.25)

    // Open numbers_list.txt for reading
    inputFile = fopen("numbers_list.txt", "r");     (0.25)
    if (inputFile == NULL) {
        printf("Error opening numbers_list.txt");    (0.25)
```

```
    return 1;
}

// Open even_numbers.txt for writing
evenFile = fopen("even_numbers.txt", "w");    (0.25)
if (evenFile == NULL) {
    printf("Error opening even_numbers.txt");     (0.25)
    fclose(inputFile);
    return 1;
}

// Open odd_numbers.txt for writing
oddFile = fopen("odd_numbers.txt", "w");     (0.25)
if (oddFile == NULL) {
    printf("Error opening odd_numbers.txt");    (0.25)
    fclose(inputFile);
    fclose(evenFile);
    return 1;
}

// Read integers from numbers_list.txt
while (fscanf(inputFile, "%d", &number) == 1) {    (0.25)
    if (number % 2 == 0) {    (0.25)
        fprintf(evenFile, "%d\n", number);   (0.25)
        evenCount++;   (0.25)
    } else {
        fprintf(oddFile, "%d\n", number);    (0.25)
        oddCount++;   (0.25)
    }
}

// Close all files
fclose(inputFile);
fclose(evenFile);            (0.5)
fclose(oddFile);

// Print total counts
printf("Total even numbers: %d\n", evenCount);   (0.25)
```

```
    printf("Total odd numbers: %d\n", oddCount);   (0.25)

    return 0;
}
```

**Exercise 3 (5 pts) :**

```
#include <stdio.h>

// 1. Write a function void incrementBy(int *value, int amount)
void incrementBy(int *value, int amount) {    (1)
    *value += amount;   (1)
}

int main() {
    // 2. In the main function:
    // Declare an integer variable counter and initialize it to 50.
    int counter = 50;   (0.5)
    // Declare an integer variable step and initialize it to 10.
    int step = 10;    (0.5)

    // Print the value of counter before the function call
    printf("Counter before increment: %d\n", counter);   (0.5)

    // Call incrementBy to increase counter by step.
    incrementBy(&counter, step);   (1)

    // Print the value of counter after the function call to verify
// the change.
    printf("Counter after increment: %d\n", counter); (0.5)

    return 0;
}
```

**Exercise 4 (5 pts) :**

```
#include <stdio.h>
#include <stdlib.h>

// Given structure for a singly linked list node:
typedef struct Node {
    int data;                    (1)
    struct Node *next;
} Node;

// 1. Write a function Node* insertAtEnd(Node* head, int newData)
Node* insertAtEnd(Node* head, int newData) {
    Node* newNode = (Node*)malloc(sizeof(Node));   (0.25)
    if (newNode == NULL) {
```

```
        printf("Failed to allocate memory for new node");   (0.25)
        return head;
    }

    newNode->data = newData;   (0.25)
    newNode->next = NULL;

    if (head == NULL)
        return newNode;    (0.25)

    Node* current = head;      (0.25)
    while (current->next != NULL)
        current = current->next;    (0.25)

    current->next = newNode;   (0.25)

     return head;   (0.25)

}

int main() {
    // 2. In the main function:
    // Create an empty linked list.
    Node* list = NULL;     (0.5)

    // Use insertAtEnd to add the values 10, 20, and 30 to the list.
    list = insertAtEnd(list, 10);
    list = insertAtEnd(list, 20);        (0.5)
    list = insertAtEnd(list, 30);

    // Write a small loop to print all elements of the list to
//verify the insertion.
    printf("List elements: ");
    Node* current = list;                    (0.5)
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
```

```
    printf("\n");

    // Free the allocated memory
    current = list;
    while (current != NULL) {              (0.5)
        Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

GOOD LUCK!