Mock Exam 5: Advanced Data Structures and Memory Management

Time Limit: 1 hour
Total Marks: 100

Section A: Multiple Choice Questions (30 marks)

Choose the best answer for each question. 2 marks each.

Question 1: What is a linked list node composed of? a) Only data b) Only a pointer to the next node c) Data and a pointer to the next node d) An array of data

Question 2: What does `malloc(10 * sizeof(int))` do? a) Creates 10 integer variables b) Allocates memory for 10 integers on the heap c) Creates an array of 10 integers on the stack d) Returns the address of an integer

Question 3: Which operation is most efficient at the head of a linked list? a) Insertion b) Deletion c) Both insertion and deletion d) Neither insertion nor deletion

Question 4: What happens if you try to access memory after calling `free()`? a) The program continues normally b) Undefined behavior (potential crash) c) The memory is automatically reallocated d) A compilation error occurs

Question 5: In a linked list, how do you know you've reached the end? a) The next pointer is 0 b) The next pointer is NULL c) The data is 0 d) Both a and b are correct

Question 6: What is the main advantage of dynamic memory allocation? a) Faster execution b) Less memory usage c) Flexible memory size at runtime d) Automatic memory cleanup

Question 7: Which is the correct way to traverse a linked list? a) `for (int i = 0; i < size; i++)` b) `while (current != NULL)` c) `do { } while (current);` d) Arrays are better for traversal

Question 8: What does `realloc()` do? a) Frees memory b) Allocates new memory c) Resizes previously allocated memory d) Copies memory

Question 9: When deleting a node from a linked list, what must you remember to do? a) Update the pointers of surrounding nodes b) Free the memory of the deleted node c) Both a and b d) Nothing special is required

Question 10: What will this code do?

```
int *ptr = malloc(5 * sizeof(int));
free(ptr);
*ptr = 10;
```

a) Set the first integer to 10 b) Cause undefined behavior c) Reallocate the memory d) Compile successfully and run normally

Question 11: In multi-file projects, what are header files used for? a) Function implementations b) Function prototypes and declarations c) Main function only d) Variable definitions

Question 12: What does the `#ifndef` preprocessor directive do? a) Includes a file b) Defines a constant c) Prevents multiple inclusions of the same header d) Compiles conditionally

Question 13: How do you compile a multi-file C project? a) `gcc main.c` b) `gcc main.c utils.c` c) `gcc *.h` d) Compile each file separately, then link

Question 14: What's the time complexity of inserting at the head of a linked list? a) O(1) b) O(n) c) O(log n) d) $O(n^2)$

Question 15: Which is true about stack vs heap memory? a) Stack is manually managed, heap is automatic b) Stack is automatic, heap is manually managed c) Both are manually managed d) Both are automatically managed

---

Section B: Code Analysis and Debugging (30 marks)

Question 16: Debug the Linked List (10 marks) Find and fix the errors in this linked list insertion function:

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

struct node* insert_head(struct node *head, int value) {
    struct node *new_node = malloc(sizeof(struct node));
    new_node->data = value;
    new_node->next = head;
    return new_node;
}
```

What's wrong with this code? Write the corrected version:

Question 17: Memory Management Analysis (10 marks) Analyze this code and identify the memory management issues:

```c
int* create_array(int size) {
    int *arr = malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        arr[i] = i;
    }
    return arr;
}

int main(void) {
```

```
    int *numbers = create_array(10);
    printf("First number: %d\n", numbers[0]);
    return 0;
}
```

List the issues and provide the corrected code:

Question 18: Function Sorting (10 marks) Arrange these lines to create a working function that deletes a node with a specific value from a linked list:

```
A. if (head == NULL) return NULL;
B. struct node* delete_value(struct node *head, int value) {
C. }
D. if (head->data == value) {
E. struct node *temp = head;
F. head = head->next;
G. free(temp);
H. return head;
I. }
J. return head;
```

Write the correct order: _____

Section C: Programming Problems (40 marks)

Question 19: Student Record System (20 marks) Create a program that manages student records using a linked list:

Requirements:

1. **Define a struct** for student with: ID (int), name (char[50]), GPA (double)
2. **Implement these functions:**
   - `struct student* create_student(int id, char name[], double gpa)` - creates new student node
   - `struct student* add_student(struct student *head, int id, char name[], double gpa)` - adds to beginning
   - `void print_students(struct student *head)` - prints all students
   - `struct student* find_student(struct student *head, int id)` - finds student by ID
   - `void free_all_students(struct student *head)` - frees all memory
3. **In main:**
   - Add at least 3 students to the list
   - Print all students
   - Search for a specific student by ID

- Free all memory before exiting

Example output:

```
=== Student Records ===
ID: 103, Name: Charlie, GPA: 3.2
ID: 102, Name: Bob, GPA: 3.8
ID: 101, Name: Alice, GPA: 3.5

Searching for student 102...
Found: Bob, GPA: 3.8
```

Question 20: Dynamic Array with Resizing (20 marks) Write a program that demonstrates a smart array that can grow as needed:

Requirements:

1. **Create a struct** called `DynamicArray` with:
   - `int *data` (pointer to the array)
   - `int size` (current number of elements)
   - `int capacity` (maximum elements before resize needed)
2. **Implement these functions:**
   - `DynamicArray* create_array(int initial_capacity)` - creates new dynamic array
   - `int add_element(DynamicArray *arr, int value)` - adds element, resizes if needed
   - `void print_array(DynamicArray *arr)` - prints all elements and capacity info
   - `void free_array(DynamicArray *arr)` - frees all memory
3. **In main:**
   - Create array with initial capacity of 2
   - Add 6 numbers to trigger automatic resizing
   - Print the array after each addition to show growth
   - Free all memory

Key features:

- When array is full, double the capacity using `realloc()`
- Always check if memory allocation succeeds
- Print capacity changes when they occur

Example output:

```
Created array with capacity 2
Added 10 - Array: [10] (size: 1, capacity: 2)
Added 20 - Array: [10, 20] (size: 2, capacity: 2)
Added 30 - Resized to capacity 4 - Array: [10, 20, 30] (size: 3, capacity:
4)
```

```
Added 40 - Array: [10, 20, 30, 40] (size: 4, capacity: 4)
Added 50 - Resized to capacity 8 - Array: [10, 20, 30, 40, 50] (size: 5,
capacity: 8)
```

Answer Template

Section B Solutions:

Question 16 - Corrected Code:

```
// Write the corrected linked list insertion function
```

Question 17 - Issues and Corrections: Issues found:

1. _____
2. _____

Corrected code:

```
// Write corrected code here
```

Question 18 - Correct Order:

```
Line order: _____
```

Section C Solutions:

Question 19 - Student Record System:

```
// Write your complete program here
```

Question 20 - Dynamic Array with Resizing:

```
// Write your complete program here
```

Marking Rubric

- **MCQ (30 marks):** 2 marks per correct answer
- **Code Analysis/Debug (30 marks):** 10 marks per question
  - Issue identification: 5 marks
  - Correct solution: 5 marks
- **Programming Problems (40 marks):** 20 marks each
  - Correct data structure design: 8 marks
  - Function implementations: 8 marks
  - Memory management: 4 marks

Critical Success Factors:

- **Memory Safety:** Always check `malloc()` return values
- **Proper Cleanup:** Free all allocated memory
- **Pointer Management:** Update pointers correctly when modifying linked structures
- **Edge Cases:** Handle empty lists, single elements, etc.
- **Function Design:** Clear interfaces and proper return values

Final Reminders:

- Test your logic with small examples
- Draw diagrams for linked list operations
- Every `malloc()` needs a `free()`
- Check for NULL pointers before dereferencing
- Use meaningful variable names even under time pressure