TASK 1:

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

// Define the structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Function to add a node at the beginning of the list
void addNodeAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}

// Function to add a node at the end of the list
void addNodeAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}
```

```
// Function to print the linked list
void printLinkedList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL; // Initialize an empty linked list

    // Adding nodes to the beginning and end of the list
    addNodeAtBeginning(&head, 5);
    addNodeAtEnd(&head, 10);
    addNodeAtEnd(&head, 15);

    // Printing the linked list
    printf("Linked List: ");
    printLinkedList(head);

    // Freeing allocated memory
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

TASK 2:

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void addNodeAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}

void addNodeAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

// Function to insert a node after a specific value
void insertAfterValue(struct Node* head, int valueToInsertAfter, int value) {
    struct Node* current = head;
    while (current != NULL) {
        if (current->data == valueToInsertAfter) {
            struct Node* newNode = createNode(value);
            newNode->next = current->next;
            current->next = newNode;
            return;
        }
        current = current->next;
    }
    printf("Value %d not found in the list.\n", valueToInsertAfter);
}

// Function to delete a node with a given value
void deleteNodeWithValue(struct Node** head, int valueToDelete) {
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current != NULL) {
        if (current->data == valueToDelete) {
            if (prev == NULL) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }
    printf("Value %d not found in the list.\n", valueToDelete);
}

// Function to print the linked list
void printLinkedList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    addNodeAtBeginning(&head, 5);
    addNodeAtEnd(&head, 10);

    // Insert 25 after 10
    insertAfterValue(head, 10, 25);

    // Delete the node with the value 10
    deleteNodeWithValue(&head, 10);

    // Insert 20 at position 2
    insertAfterValue(head, 5, 20);

    // Delete the node at position 3
    deleteNodeWithValue(&head, 15); // Assuming 15 is at position 3

    printf("Linked List: ");
    printLinkedList(head);

    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

TASK 3:

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = value;
   newNode->next = NULL;
   return newNode;
}

void addNodeAtBeginning(struct Node** head, int value) {
   struct Node* newNode = createNode(value);
   newNode->next = *head;
   *head = newNode;
}

void addNodeAtEnd(struct Node** head, int value) {
   struct Node* newNode = createNode(value);
   if (*head == NULL) {
      *head = newNode;
      return;
   }
   struct Node* current = *head;
   while (current->next != NULL) {
      current = current->next;
   }
   current->next = newNode;
}

void insertAfterValue(struct Node* head, int valueToInsertAfter, int value) {
   struct Node* current = head;
   while (current != NULL) {
      if (current->data == valueToInsertAfter) {
            struct Node* newNode = createNode(value);
            newNode->next = current->next;
            current->next = newNode;
            return;
        }
        current = current->next;
    }
    printf("Value %d not found in the list.\n", valueToInsertAfter);
}

void deleteNodeWithValue(struct Node** head, int valueToDelete) {
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current != NULL) {
        if (current->data == valueToDelete) {
            if (prev == NULL) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }
    printf("Value %d not found in the list.\n", valueToDelete);
}

// Function to reverse the linked list in-place
void reverseLinkedList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    *head = prev;
}
```

```
// Function to print the linked list
void printLinkedList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;

    addNodeAtBeginning(&head, 5);
    addNodeAtEnd(&head, 10);
    insertAfterValue(head, 10, 25);
    deleteNodeWithValue(&head, 10);
    insertAfterValue(head, 5, 20);

    // Reverse the linked list
    reverseLinkedList(&head);

    printf("Linked List: ");
    printLinkedList(head);

    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

TASK 4:

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void addNodeAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}

void addNodeAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

// Function to check if the linked list contains a cycle and find the cycle start node
int hasCycle(struct Node* head, struct Node** cycleStart) {
    struct Node* slow = head;
    struct Node* fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) {
            // The linked list has a cycle
            *cycleStart = head;
            while (*cycleStart != slow) {
                *cycleStart = (*cycleStart)->next;
                slow = slow->next;
            }
            return 1;
        }
    }

    // No cycle found
    return 0;
}

int main() {
    struct Node* head = NULL;

    addNodeAtBeginning(&head, 5);
    addNodeAtEnd(&head, 10);
    addNodeAtEnd(&head, 15);

    // Create a cycle by connecting the last node to the second node (10)
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = head->next;

    struct Node* cycleStart = NULL;
    int hasCycleResult = hasCycle(head, &cycleStart);

    if (hasCycleResult) {
        printf("Has Cycle: Yes\n");
        printf("Cycle Start Node: %d\n", cycleStart->data);
    } else {
        printf("Has Cycle: No\n");
    }
    // Free the memory (Note: In the presence of a cycle, you would need to break the cycle
before freeing the memory)
    current->next = NULL;
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

TASK 5:

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void addNodeAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

// Function to merge two sorted linked lists
struct Node* mergeSortedLists(struct Node* listA, struct Node* listB) {
    struct Node dummy; // Dummy node to simplify the code
    struct Node* tail = &dummy;
    dummy.next = NULL;

    while (1) {
        if (listA == NULL) {
            tail->next = listB;
            break;
        }
        if (listB == NULL) {
            tail->next = listA;
            break;
        }

        if (listA->data <= listB->data) {
            tail->next = listA;
            listA = listA->next;
        } else {
            tail->next = listB;
            listB = listB->next;
        }

        tail = tail->next;
    }

    return dummy.next;
}

// Function to print the linked list
void printLinkedList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* listA = NULL;
    struct Node* listB = NULL;

    addNodeAtEnd(&listA, 5);
    addNodeAtEnd(&listA, 10);

    addNodeAtEnd(&listB, 7);
    addNodeAtEnd(&listB, 12);

    printf("List A: ");
    printLinkedList(listA);

    printf("List B: ");
    printLinkedList(listB);

    // Merge the two sorted lists
    struct Node* mergedList = mergeSortedLists(listA, listB);

    printf("Merged List: ");
    printLinkedList(mergedList);

    // Free the memory
    while (mergedList != NULL) {
        struct Node* temp = mergedList;
        mergedList = mergedList->next;
        free(temp);
    }

    return 0;
}
```