```
6) a)

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


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


struct Node* create_node(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

void insert_end(struct Node** head, int data) {
    struct Node* new_node = create_node(data);
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new_node;
}

void print_list(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d\t ", temp->data);
        temp = temp->next;
    }
```

```
    printf("\n");
}

void sort_list(struct Node* head) {
    if (head == NULL) return;
    struct Node* i = head;
    struct Node* j = NULL;
    int temp;
    while (i != NULL) {
        j = i->next;
        while (j != NULL) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
}

void reverse_list(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;
}


void concatenate_lists(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
```

```
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    insert_end(&list1, 4);
    insert_end(&list1, 2);
    insert_end(&list1, 3);
    insert_end(&list1, 1);
    // insert_end(&list1, 5);

    printf("Original List 1: ");
    print_list(list1);


    reverse_list(&list1);
    printf("Reversed List 1: ");
    print_list(list1);

    sort_list(list1);
    printf("Sorted List 1: ");
    print_list(list1);


    insert_end(&list2, 6);
    insert_end(&list2, 7);
    insert_end(&list2, 8);
    // insert_end(&list2, 9);

    printf("Original List 2: ");
    print_list(list2);
```

```
    concatenate_lists(&list1, list2);
    printf("Concatenated List: ");
    print_list(list1);

    return 0;
}
```

6) b)

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


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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
```

```
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    struct Node* temp = *front;
    int dequeuedData = temp->data;
    *front = (*front)->next;
    if (*front == NULL) {
        *rear = NULL;
    }
    free(temp);
    return dequeuedData;
}


void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
```

```
    while (temp != NULL) {
        printf("%d\t ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* stackTop = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;

    int choice, data;

    while (1) {
        printf("\nChoose operation:\n");
        printf("1. Stack Push\n");
        printf("2. Stack Pop\n");
        printf("3. Queue Enqueue\n");
        printf("4. Queue Dequeue\n");
        printf("5. Display Stack\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&stackTop, data);
                printf("data is pushed to stack \n");
                break;
            case 2:
                data = pop(&stackTop);
                if (data != -1) {
                    printf("Popped from stack: %d\n", data);
                }
                break;
```

```
            case 3:

                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                printf("data is enqueued  \n");
                break;

            case 4:
                data = dequeue(&queueFront, &queueRear);
                if (data != -1) {
                    printf("Dequeued from queue: %d\n", data);
                }
                break;
            case 5:
                printf("Stack: ");
                printList(stackTop);
                break;
            case 6:
                printf("Queue: ");
                printList(queueFront);
                break;
            case 7:
                printf("Exiting the Program.....\n");
                exit(0);
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
}
```

6a)

```
Output

Original List 1: 4   2   3   1
Reversed List 1: 1   3   2   4
Sorted List 1: 1     2   3   4
Original List 2: 6   7   8
Concatenated List: 1     2   3   4   6   7   8
```

6b)

```
Output

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 23
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 24
data is pushed to stack
```

Output

```
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 3
Enter data to enqueue: 26
data is enqueued

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 3
Enter data to enqueue: 26
data is enqueued
```

Output

```
data is enqueued

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 2
Popped from stack: 25

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 2
Popped from stack: 24
```

Output

```
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 26
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 5
Stack: 26    25  23

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 26    26  28
```