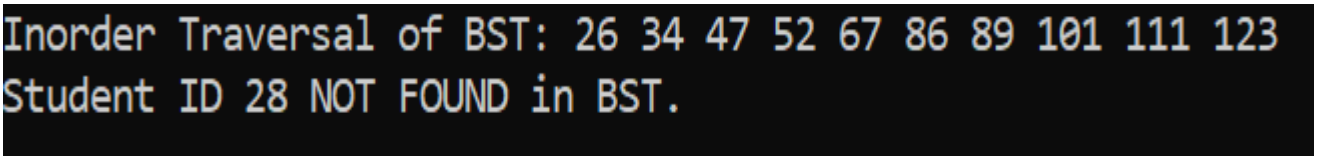
QL1: BST Implementation (Based on Mr. Arif's Solution)

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

struct Node* insert(struct Node* root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}
```

```
void inorder(struct Node* root) {
   if (root == NULL) return;
   inorder(root->left);
   printf("%d ", root->data);
   inorder(root->right);
}

int search(struct Node* root, int key) {
   if (root == NULL) return 0;
   if (root->data == key) return 1;
   if (key < root->data)
      return search(root->left, key);
   else
      return search(root->right, key);
}

int main() {
   int values[] = {26, 47, 89, 34, 101, 67, 86, 123, 52, 111};
   int n = sizeof(values) / sizeof(values[0]);
   struct Node* root = NULL;

   for (int i = 0; i < n; i++) {
      root = insert(root, values[i]);
   }
```

```
    printf("Inorder Traversal of BST: ");
    inorder(root);
    printf("\n");

    int studentID = 28 ;
    if (search(root, studentID))
        printf("Student ID %d FOUND in BST.\n", studentID);
    else
        printf("Student ID %d NOT FOUND in BST.\n", studentID);

    return 0;
}
```



QL2: Linked List Implementation (Based on Mr. Tamim's Solution)

Part A: Delete 'XX' from the linked list

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

typedef struct Node {
    float data;
    struct Node* next;
```

```
} Node;

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

void insertEnd(Node** head, float value) {
    Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}
```

```
void deleteNode(Node** head, float key) {
    Node *temp = *head, *prev = NULL;
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Value %.2f not found in the list.\n", key);
        return;
    }
    prev->next = temp->next;
    free(temp);
}

void printList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%.1f -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
```

```
}

void freeList(Node* head) {
    Node* current = head;
    Node* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

int main() {
    Node* head = NULL;
    insertEnd(&head, 10.5);
    insertEnd(&head, 6.26);
    insertEnd(&head, 15.7);
    insertEnd(&head, 34.3);

    printf("Original linked list: ");
    printList(head);

    deleteNode(&head, 6.26);

    printf("List after deleting XX (6.26): ");
    printList(head);
```

```
    freeList(head);
    return 0;
}
```

```
Original linked list: 10.5 -> 6.3 -> 15.7 -> 34.3 -> NULL
List after deleting XX (6.26): 10.5 -> 15.7 -> 34.3 -> NULL

Process returned 0 (0x0)   execution time : 0.153 s
Press any key to continue.
```

Part B: Display elements in reverse order

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

typedef struct Node {
    float data;
    struct Node* next;
} Node;

Node* createNode(float value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = value;
```

```
    newNode->next = NULL;
    return newNode;
}

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

void printList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%.1f -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```

```
void printReverseRecursive(Node* head) {
    if (head == NULL) {
        return;
    }
    printReverseRecursive(head->next);
    printf("%.1f ", head->data);
}

void printReverseStack(Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    float stack[100];
    int top = -1;
    Node* current = head;
    while (current != NULL && top < 99) {
        stack[++top] = current->data;
        current = current->next;
    }
    printf("Reversed list (using stack): ");
    while (top >= 0) {
        printf("%.1f ", stack[top--]);
    }
    printf("\n");
}
```

```
void freeList(Node* head) {
    Node* current = head;
    Node* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

int main() {
    Node* head = NULL;
    insertEnd(&head, 10.5);
    insertEnd(&head, 6.26);
    insertEnd(&head, 15.7);
    insertEnd(&head, 34.3);

    printf("Original linked list: ");
    printList(head);

    printf("Reversed list (using recursion): ");
    printReverseRecursive(head);
    printf("\n");

    printReverseStack(head);
```

```
    freeList(head);

    return 0;
}
```

```
Original linked list: 10.5 -> 6.3 -> 15.7 -> 34.3 -> NULL
Reversed list (using recursion): 34.3 15.7 6.3 10.5
Reversed list (using stack): 34.3 15.7 6.3 10.5

Process returned 0 (0x0)   execution time : 0.148 s
Press any key to continue.
```