Batch: E2	**Roll No.:** 16010123325

Experiment / assignment / tutorial No. 3

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of basic Linked List – creation, insertion, deletion, traversal, searching an element

Objective: To understand the advantage of linked list over other structures like arrays in implementing the general linear list

Expected Outcome of Experiment:

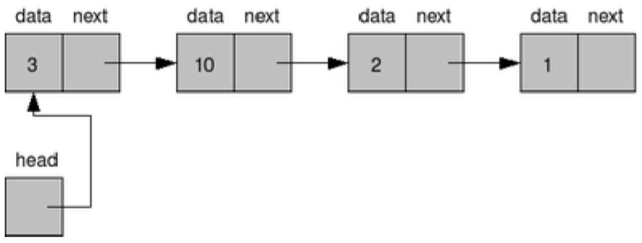
CO	Outcome
1	Comprehend the different data structures used in problem solving

Books/ Journals/ Websites referred:

https://www.geeksforgeeks.org/linked-list-data-structure/

Introduction:

A linear list is a list where each element has a unique successor. There are four common operations associated with a linear list: insertion, deletion, retrieval, and traversal. Linear list can be divided into two categories: general list and restricted list. In general list the data can be inserted or deleted without any restriction whereas in restricted list there is restrictions for these operations. Linked list and arrays are commonly used to implement general linear list. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.



A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This Structure that contains elements and pointers to the next structure is called a Node.

Related Theory: -

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers

Linked List ADT:

A **Linked List** is a linear data structure where each element, called a node, contains data and a reference (or pointer) to the next node in the sequence. Unlike arrays, nodes are not stored in contiguous memory locations but are connected through these references.

Operations-

1. createNode(int data): Allocates memory for a new node, sets its data, and initializes its next pointer to NULL.

2. createList(): Constructs a linked list by repeatedly adding new nodes based on user input.

3. printList(Node* head): Prints the entire linked list from the given node.

4. insertAtBegin(Node* head, int el): Inserts a new node with value el at the beginning of the list.

5. insertAtEnd(Node* head, int el): Adds a new node with value el at the end of the list.

6. insertAtKth(Node* head, int el, int k): Inserts a new node with value el after the specified position in the list.

7. deleteBegin(Node* head): Removes the first node of the list and updates the head.

8. deleteEnd(Node* head): Deletes the last node of the list by updating the next pointer of the second-to-last node.

9. deleteValue(Node* head, int el): Deletes the first node containing the specified data el.

10. search(Node* head, int el): Searches for a node with data x and prints if found or not.

11. lengthofLL(Node* head): Counts and prints the number of nodes in the list.

Algorithm for creation, insertion, deletion, traversal and searching an element in linked list:

1. Node Definition:

- Define a ‘Node’ structure with ‘data’ and a pointer to the next node.

2. Create Node:

- Allocate memory for a new node, set its data, and initialize the `next` pointer to ‘NULL’.

3. Create List:

- Initialize an empty list.
- Continuously prompt the user to enter data and create new nodes, linking them in the list.
- Continue until the user decides to stop.

4. Display Function:

- Traverse and print each node's data until the end of the list (NULL).

5. Insert at Beginning:

- Create a new node.
- Set its `next` pointer to the current head and update the head to the new node.

6. Insert at End:

- Create a new node.
- Traverse to the end of the list and link the new node to the last node.

7. Insert at Position:

- Traverse to the specified position.
- Insert the new node at that position, adjusting pointers accordingly.

8. Delete from Beginning:

- Remove the head node and update the head to the next node.

9. Delete from End:

- Traverse to the second-to-last node.
- Remove the last node and set the `next` pointer of the second-to-last node to `NULL`.

10. Delete by Value:

- Traverse the list to find the node with the specified value.
- Remove the node and adjust pointers.

11. Search:

- Traverse the list to find if a node with the specified value exists.

12. Count Elements:

- Traverse the list and count the number of nodes.

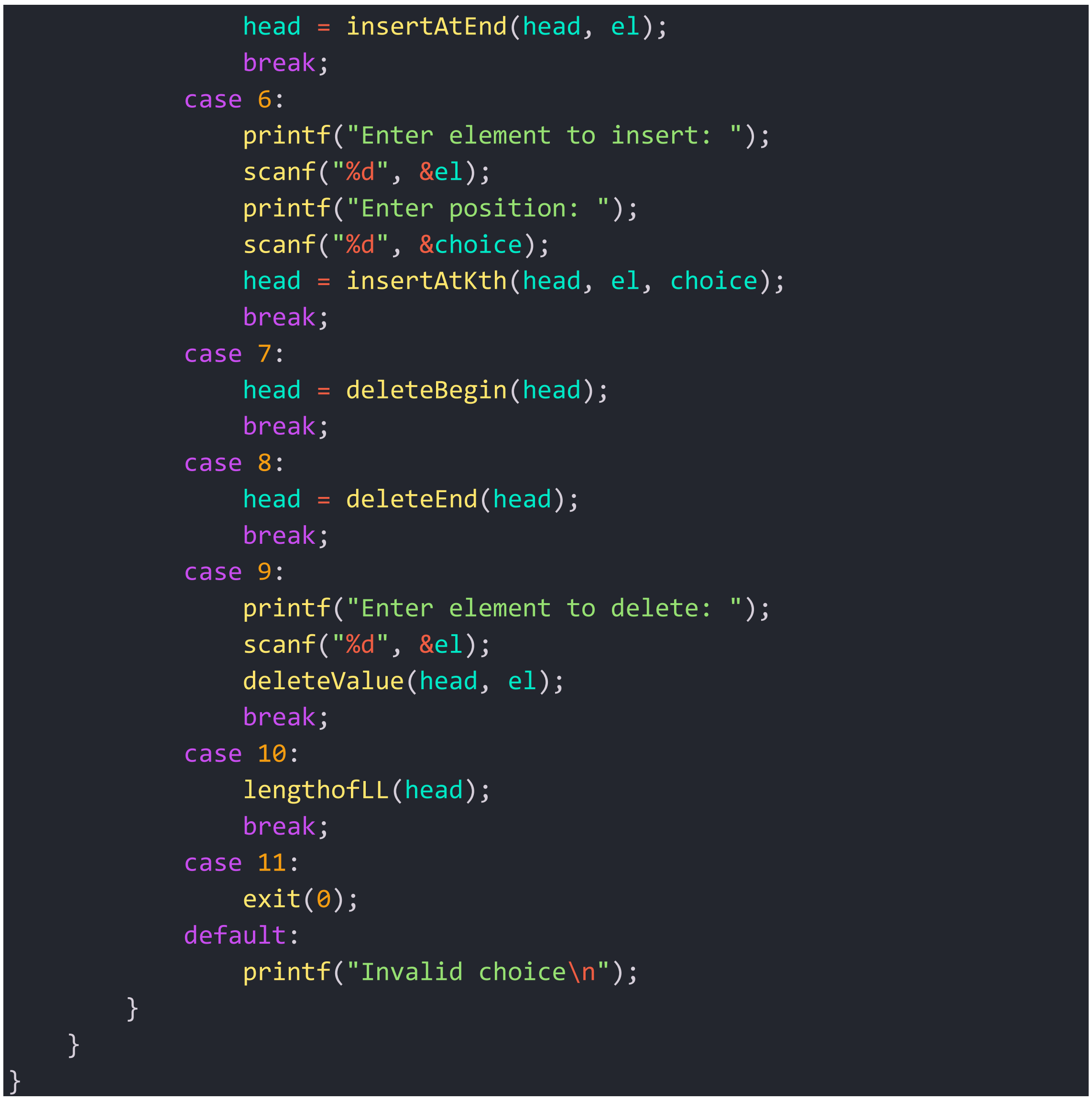
13. Main Menu:

- Provide a menu to the user to perform various operations on the linked list.
- Use a loop to handle user input and call the appropriate functions

Program source code:

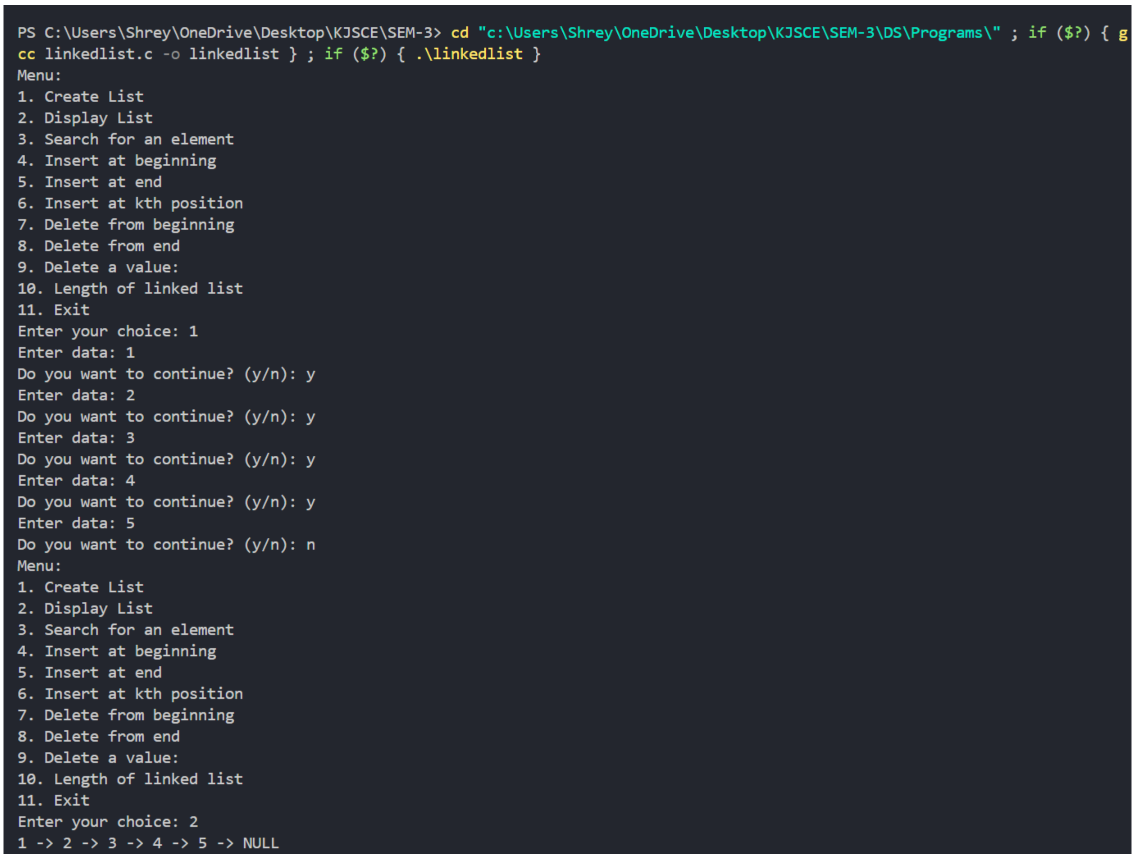
```c
#include <stdio.h>
#include <stdlib.h>

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

// Function to create a new node with given data
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory error\n");
        exit(0);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create a linked list
Node* createList() {
    Node* head = NULL;
    Node* temp = NULL;
    Node* newNode = NULL;
    int data;
    char ch = 'y';
    while (ch == 'y' || ch == 'Y') {
        printf("Enter data: ");
        scanf("%d", &data);
        newNode = createNode(data);
        if (head == NULL) head = newNode;
        else temp->next = newNode;
        temp = newNode;

        while (getchar() != '\n');
        printf("Do you want to continue? (y/n): ");
        scanf("%c", &ch);
    }
    return head;
}

// Function to display the linked list
void printList(Node* head) {
    Node* temp = head;
    while(temp) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to insert a new element at the beginning of the linked list
Node* insertAtBegin(Node* head, int el) {
    Node* newNode = createNode(el);
    if (head == NULL) head = newNode;
    else {
        newNode->next = head;
        head = newNode;
    }
    return head;
}

// Function to insert a new element at the end of the linked list
Node* insertAtEnd(Node* head, int el) {
    Node* newNode = createNode(el);
    if (head == NULL) head = newNode;
    else {
        Node* temp = head;
        while(temp->next != NULL) temp = temp->next;
        temp->next = newNode;
    }
    return head;
}

// Function to insert a new element at next position in the linked list
Node* insertAtKth(Node* head, int el, int k) {
    Node* newNode = createNode(el);
    Node* temp = head;
    int c = 1;
    while (c != k) {
        temp = temp->next;
        c++;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    return head;

}

// Function to delete the first element of the linked list
Node* deleteBegin(Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return head;
    }
    else {
        Node* temp = head;
        head = head->next;
        free(temp);
    }
    return head;
}

// Function to delete the last element of the linked list
Node* deleteEnd(Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return head;
    }
    else if (head->next == NULL) {
        free(head);
        head = NULL;
    }
    else {
        Node* temp = head;
```

```c
        while(temp->next->next != NULL) temp = temp->next;
        free(temp->next);
        temp->next = NULL;
    }
    return head;
}

// Function to delete the element of the linked list
Node* deleteValue(Node* head, int el) {
    if (head == NULL) return NULL;
    if (head->data == el) {
        Node* temp = head;
        head = head->next;
        free(temp);
        return head;
    }
    Node* temp = head;
    Node* prev = NULL;
    while (temp != NULL) {
        if (temp->data == el) {
            prev->next = temp->next;
            free(temp);
            break;
        }
        prev = temp;
        temp = temp->next;
    }
    return head;
}

// Function to search for an element in the linked list
void search(Node* head, int el) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->data == el) {
            printf("Element found\n");
            return;
        }
        temp = temp->next;
    }
    printf("Element not found\n");
}

// Function to find the length of the linked list
void lengthofLL(Node* head) {
    int cnt = 0;
    Node* temp = head;
    while(temp) {
        cnt++;
        temp = temp->next;
    }
    printf("Length of the linked list is: %d\n", cnt);
}

int main() {
    Node* head = NULL;
    int choice, el;
    while (1) {
        printf("Menu:\n");
        printf("1. Create List\n");
        printf("2. Display List\n");
        printf("3. Search for an element\n");
        printf("4. Insert at beginning\n");
        printf("5. Insert at end\n");
        printf("6. Insert at kth position\n");
        printf("7. Delete from beginning\n");
        printf("8. Delete from end\n");
        printf("9. Delete a value: \n");
        printf("10. Length of linked list\n");
        printf("11. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                head = createList();
                break;
            case 2:
                printList(head);
                break;
            case 3:
                printf("Enter element to search: ");
                scanf("%d", &el);
                search(head, el);
                break;
            case 4:
                printf("Enter element to insert at beginning: ");
                scanf("%d", &el);
                head = insertAtBegin(head, el);
                break;
            case 5:
                printf("Enter element to insert at end: ");
                scanf("%d", &el);
```

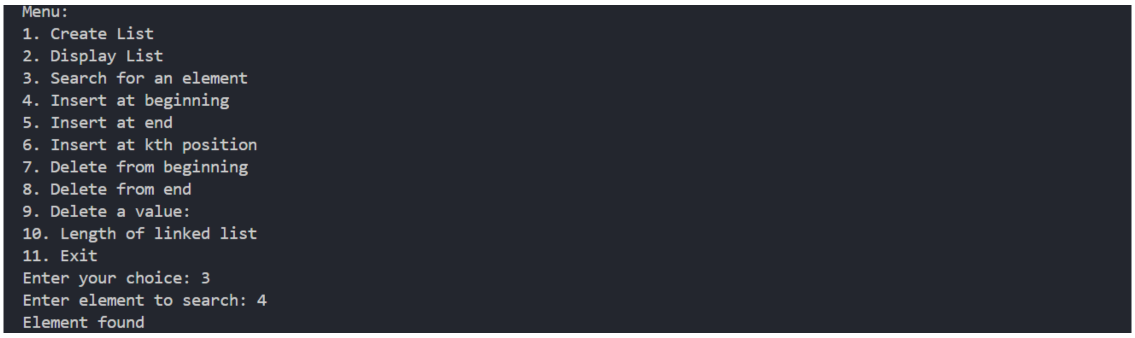


```
                head = insertAtEnd(head, el);
                break;
            case 6:
                printf("Enter element to insert: ");
                scanf("%d", &el);
                printf("Enter position: ");
                scanf("%d", &choice);
                head = insertAtKth(head, el, choice);
                break;
            case 7:
                head = deleteBegin(head);
                break;
            case 8:
                head = deleteEnd(head);
                break;
            case 9:
                printf("Enter element to delete: ");
                scanf("%d", &el);
                deleteValue(head, el);
                break;
            case 10:
                lengthofLL(head);
                break;
            case 11:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}
```

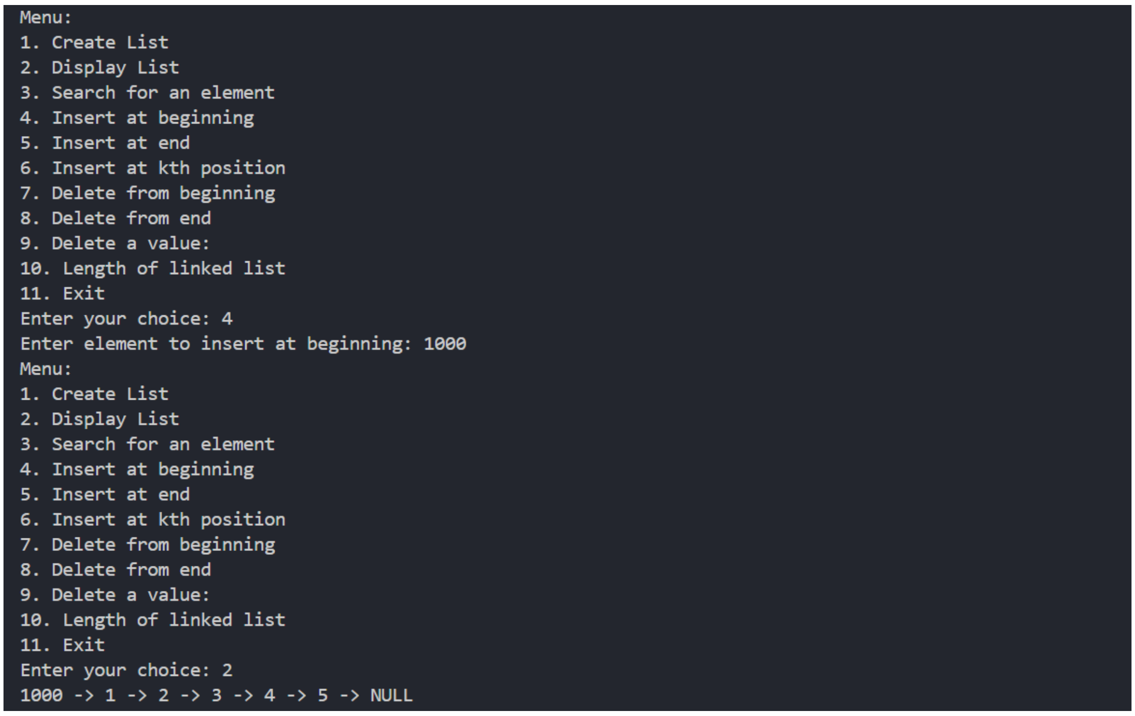
Output Screenshots:



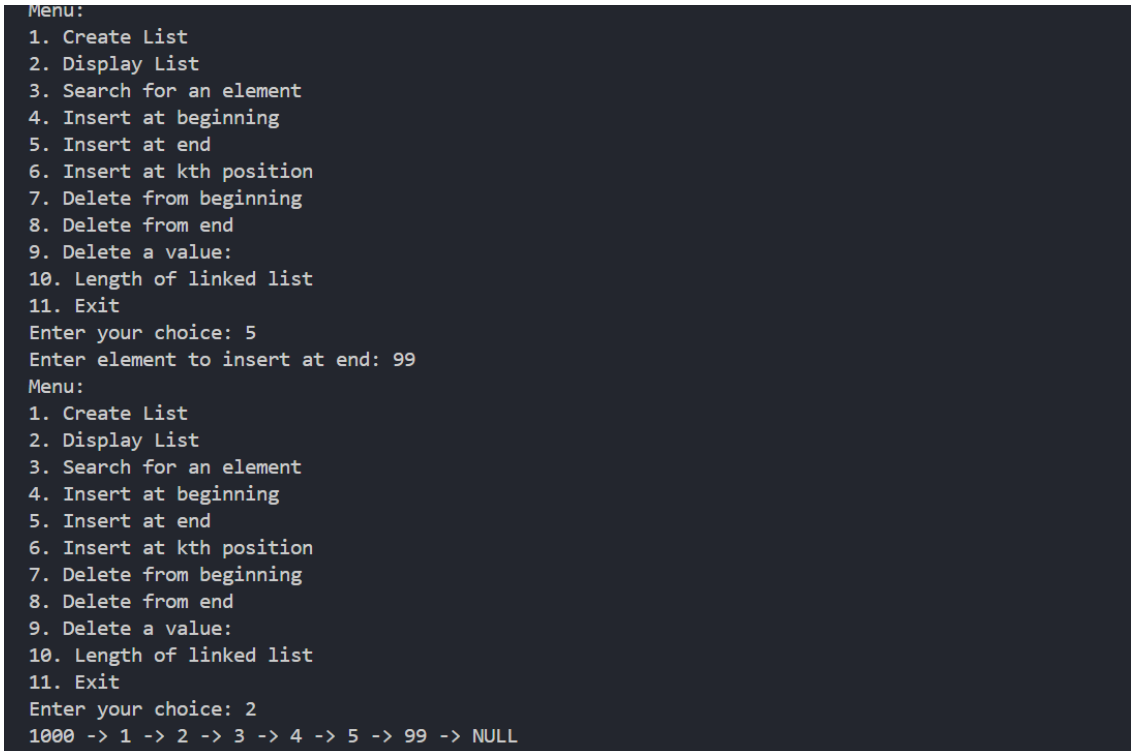
```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs\" ; if ($?) { g
cc linkedlist.c -o linkedlist } ; if ($?) { .\linkedlist }
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 1
Enter data: 1
Do you want to continue? (y/n): y
Enter data: 2
Do you want to continue? (y/n): y
Enter data: 3
Do you want to continue? (y/n): y
Enter data: 4
Do you want to continue? (y/n): y
Enter data: 5
Do you want to continue? (y/n): n
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1 -> 2 -> 3 -> 4 -> 5 -> NULL
```



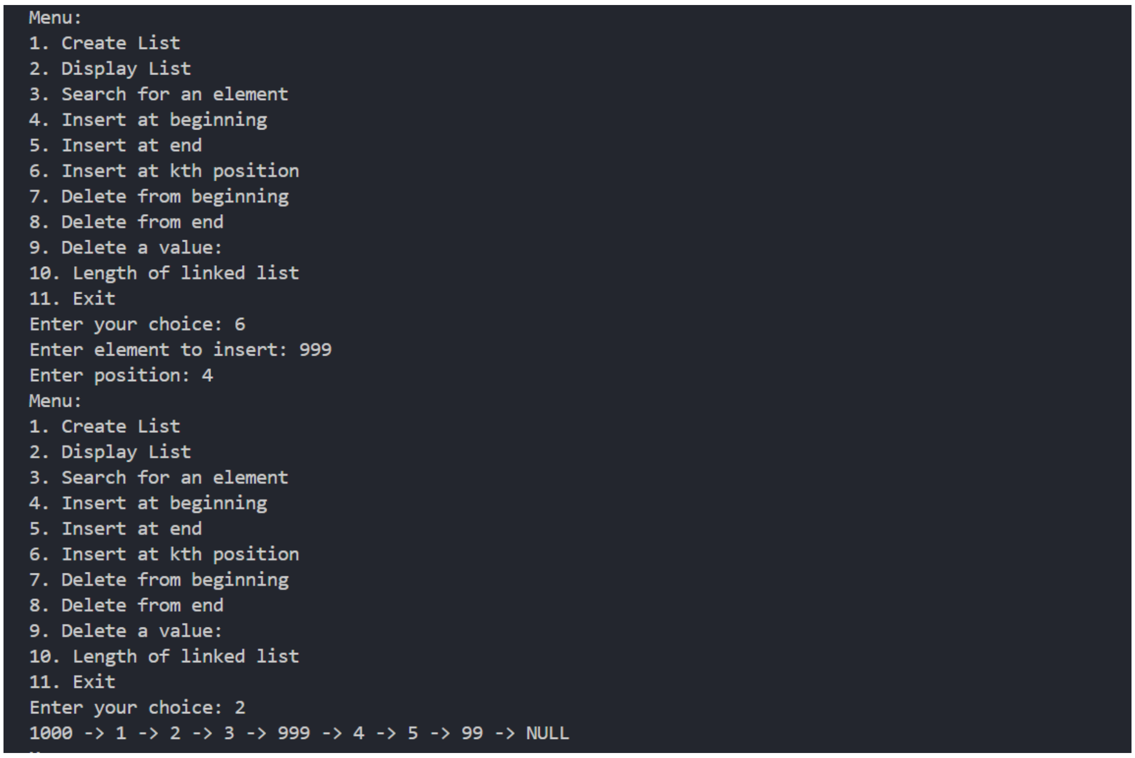
```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 3
Enter element to search: 4
Element found
```



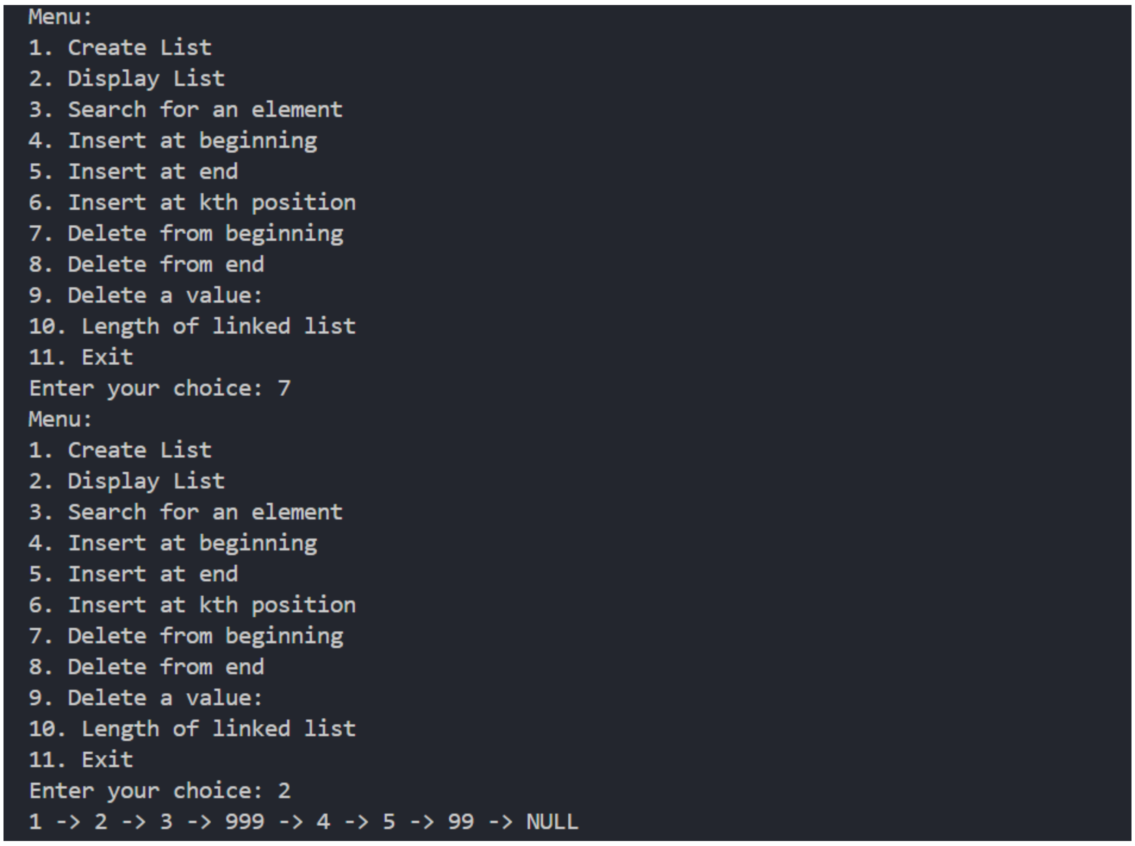
```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 4
Enter element to insert at beginning: 1000
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1000 -> 1 -> 2 -> 3 -> 4 -> 5 -> NULL
```



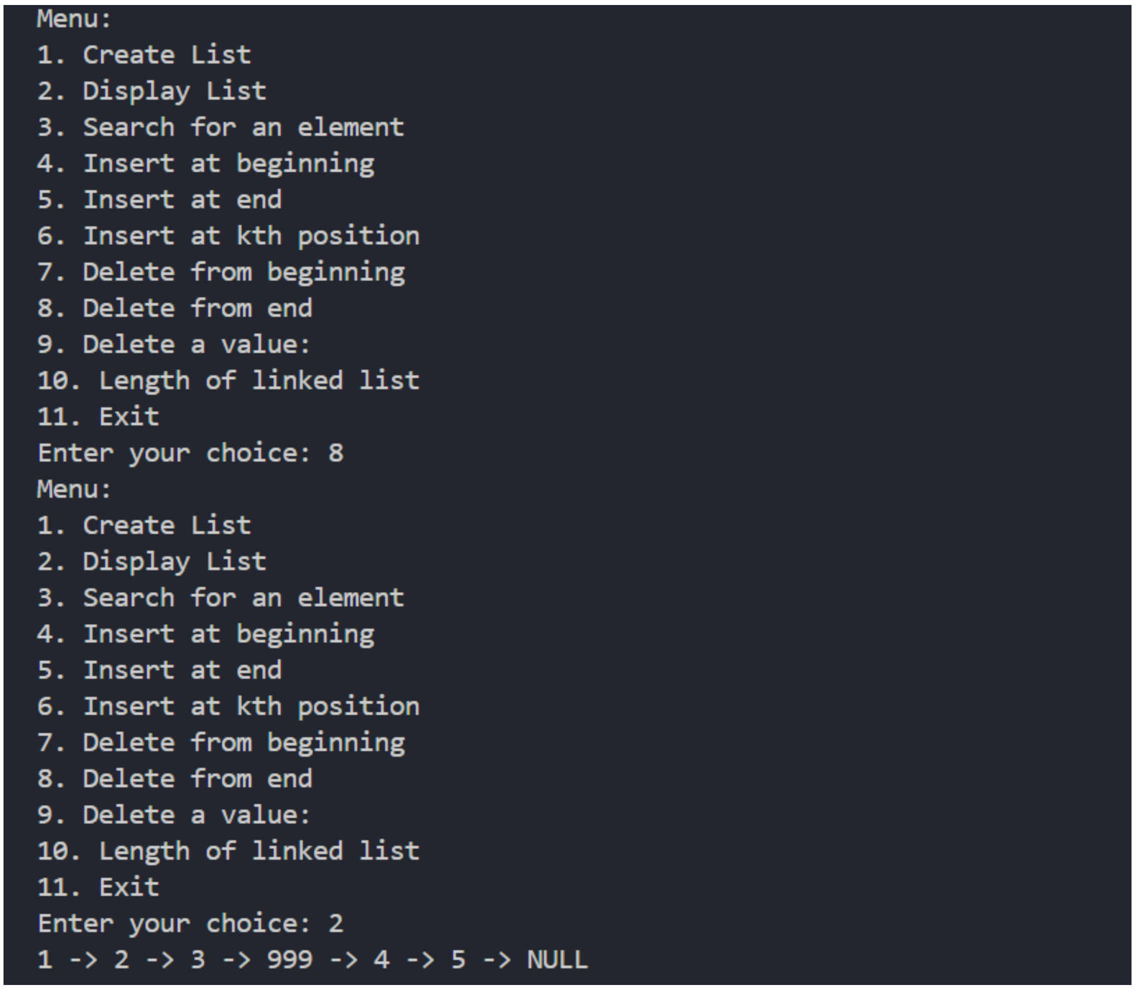
```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 5
Enter element to insert at end: 99
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1000 -> 1 -> 2 -> 3 -> 4 -> 5 -> 99 -> NULL
```



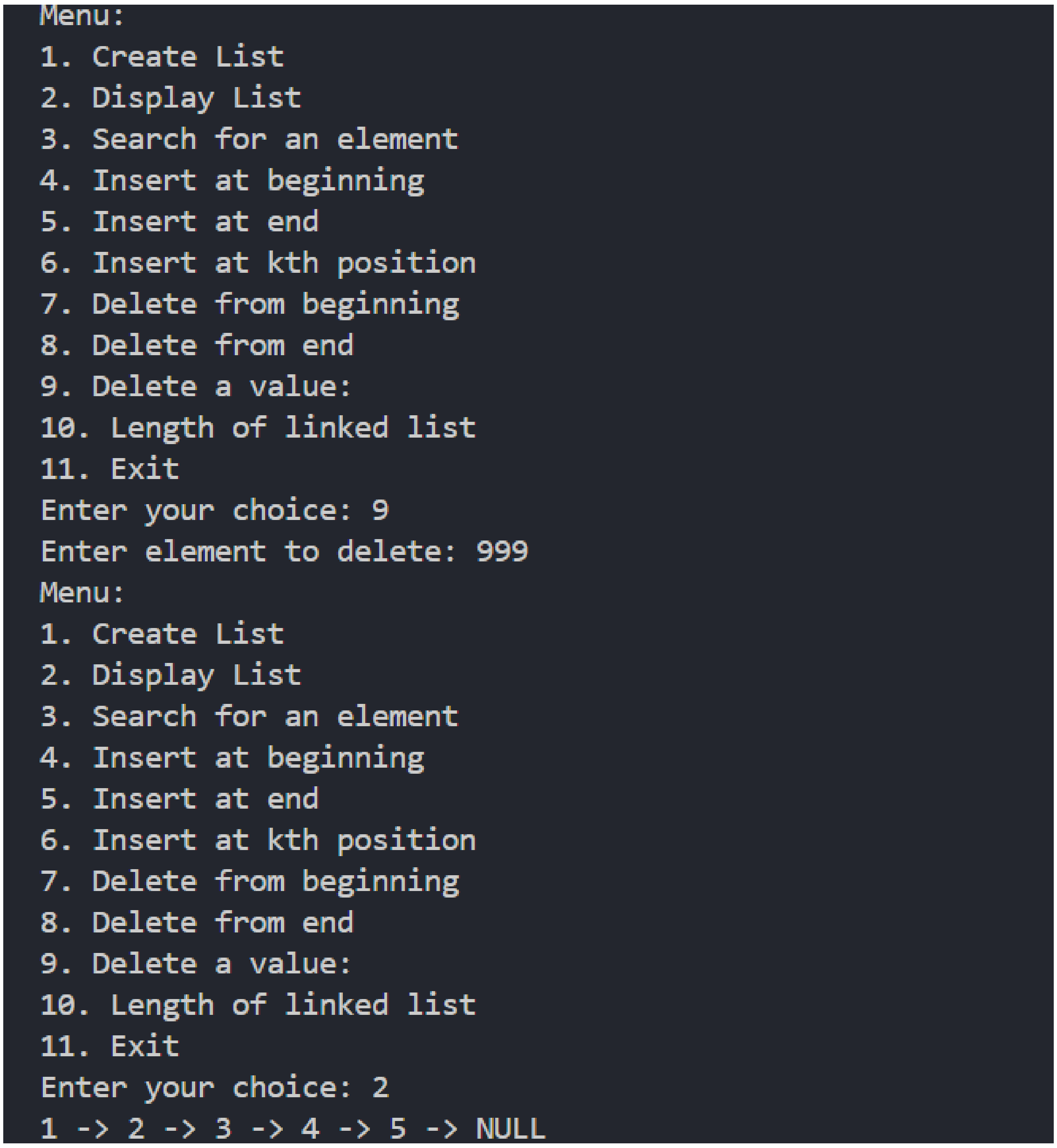
```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 6
Enter element to insert: 999
Enter position: 4
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1000 -> 1 -> 2 -> 3 -> 999 -> 4 -> 5 -> 99 -> NULL
```



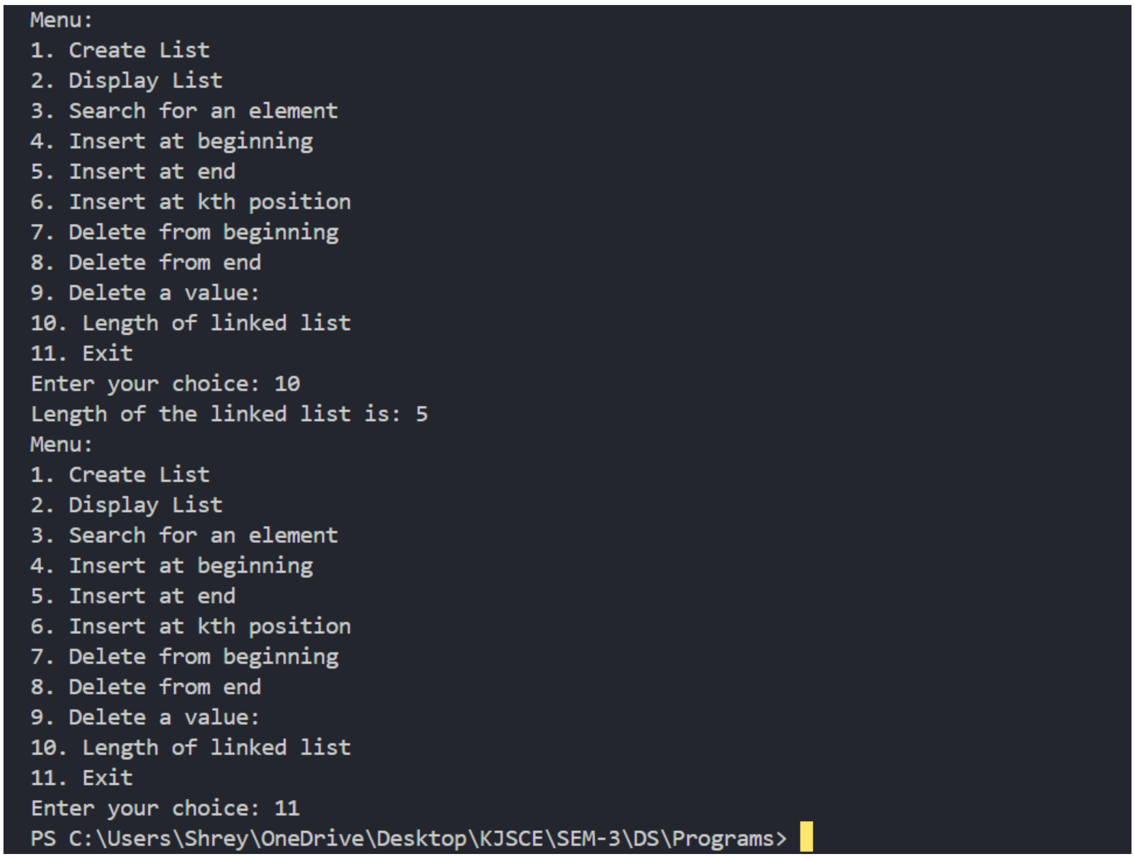
```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 7
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1 -> 2 -> 3 -> 999 -> 4 -> 5 -> 99 -> NULL
```



```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 8
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1 -> 2 -> 3 -> 999 -> 4 -> 5 -> NULL
```



```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 9
Enter element to delete: 999
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 2
1 -> 2 -> 3 -> 4 -> 5 -> NULL
```



```
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 10
Length of the linked list is: 5
Menu:
1. Create List
2. Display List
3. Search for an element
4. Insert at beginning
5. Insert at end
6. Insert at kth position
7. Delete from beginning
8. Delete from end
9. Delete a value:
10. Length of linked list
11. Exit
Enter your choice: 11
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs>
```

Conclusion:-

The C program manages a singly linked list with features for creating, inserting, deleting, displaying, and searching nodes. It provides a menu-driven interface for these operations, demonstrating core linked list functionality.

Post lab questions:

1. Write the differences between a linked list and linear array

Ans:

Linked List:

- Dynamic memory allocation, elements scattered across memory.
- Flexible size, can grow or shrink as needed.
- Efficient insertion and deletion, especially in the middle (O(1)).
- Slower access time, requires traversal (O(n)).
- Extra memory overhead due to pointers.
- Sequential traversal, no random access.

Linear Array:

- Static memory allocation, elements stored contiguously.
- Fixed size, determined at declaration.
- Less efficient insertion and deletion, requires shifting (O(n)).
- Fast access time, direct indexing (O(1)).
- No extra memory overhead per element.
- Direct access via index, easy traversal.

2. Write a C code to to remove the duplicate elements from the Singly linked list (include code and snap shot of output)

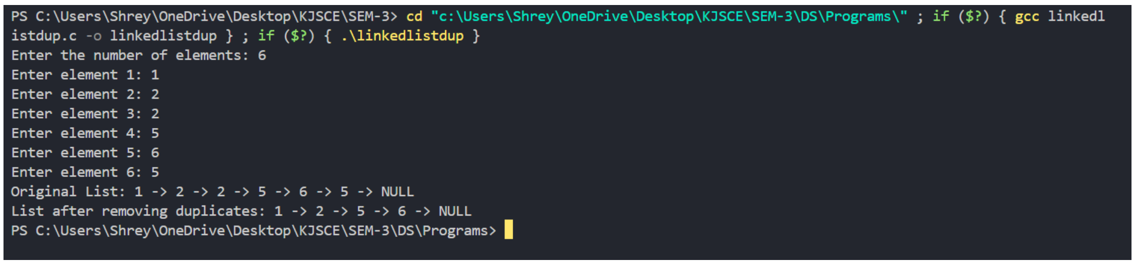
Input- head ->1->2->2->5->6->5

Ouput - head->1→2->5->6

Ans:

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

typedef struct Node {
```

```c
    int data;
    struct Node* next;
} Node;

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

// Function to insert a node at the end of the list
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }

    Node* lastNode = *head;
    while (lastNode->next) {
        lastNode = lastNode->next;
    }

    lastNode->next = newNode;
}

// Function to remove duplicates from the list
void removeDuplicates(Node* head) {
    Node* current = head;
    while (current) {
        Node* runner = current;
        while (runner->next) {
            if (runner->next->data == current->data) {
                Node* temp = runner->next;
                runner->next = runner->next->next;
                free(temp);
            } else {
                runner = runner->next;
            }
        }
        current = current->next;
    }
}

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

int main() {
    Node* head = NULL;
    int n, i, data;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &data);
        insertNode(&head, data);
    }

    printf("Original List: ");
    printList(head);

    // Remove duplicates from the list
    removeDuplicates(head);

    printf("List after removing duplicates: ");
    printList(head);

    return 0;
}
```



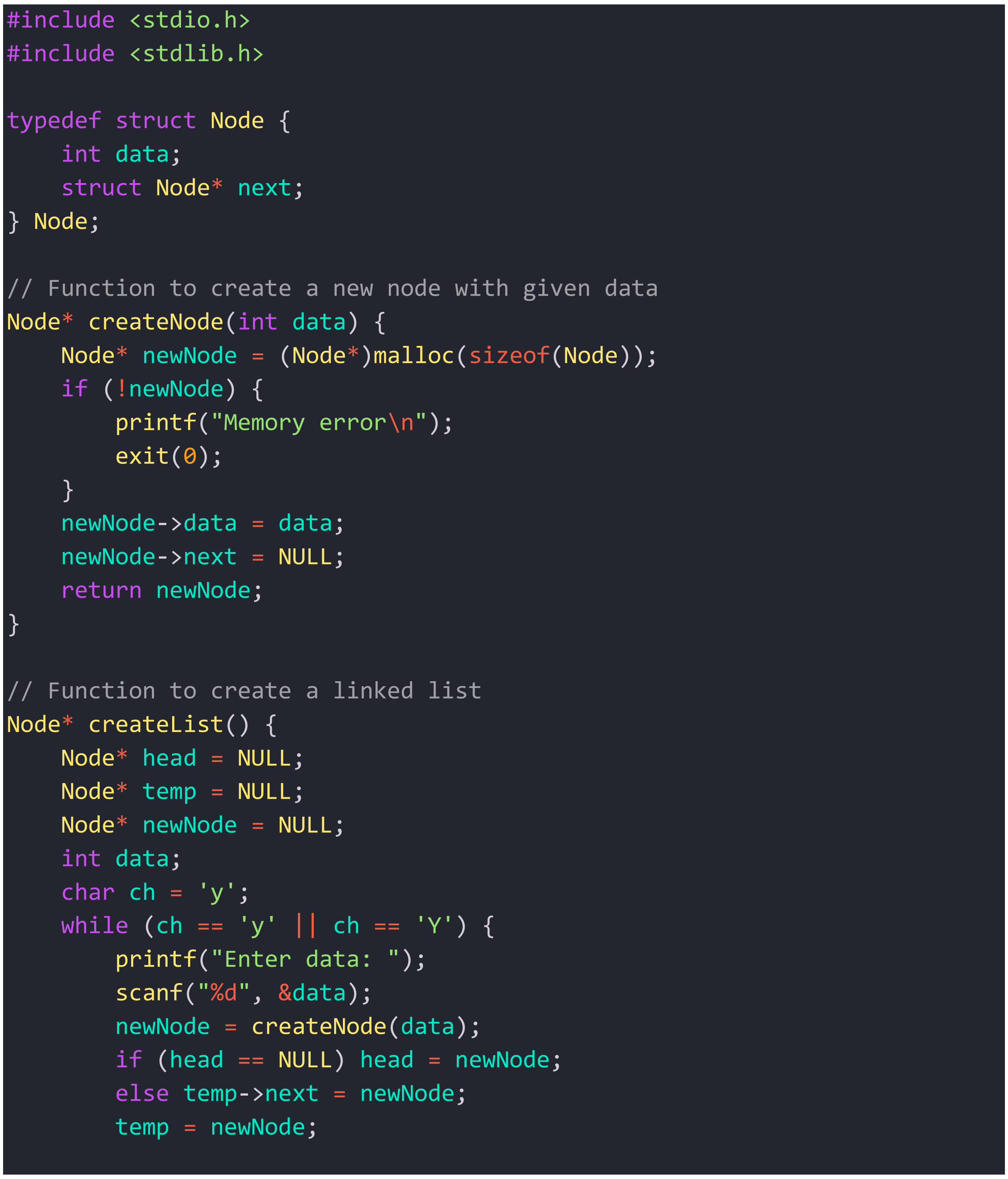
```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs\" ; if ($?) { gcc linkedl
istdup.c -o linkedlistdup } ; if ($?) { .\linkedlistdup }
Enter the number of elements: 6
Enter element 1: 1
Enter element 2: 2
Enter element 3: 2
Enter element 4: 5
Enter element 5: 6
Enter element 6: 5
Original List: 1 -> 2 -> 2 -> 5 -> 6 -> 5 -> NULL
List after removing duplicates: 1 -> 2 -> 5 -> 6 -> NULL
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs>
```

3. Write C code to reverse the linked list

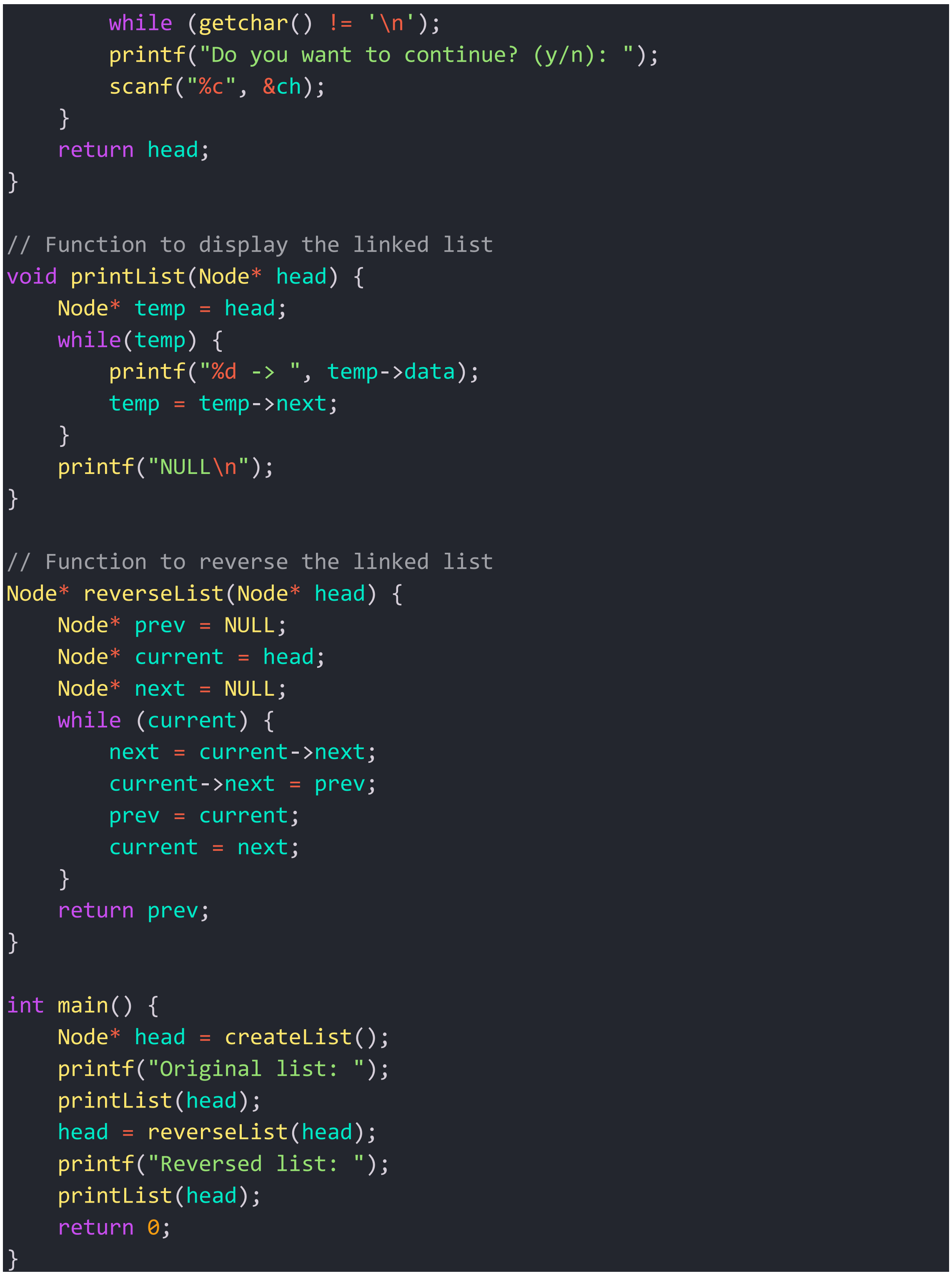
Input- head->1->2->3->4->5

Output- head->5->4->3->2->1

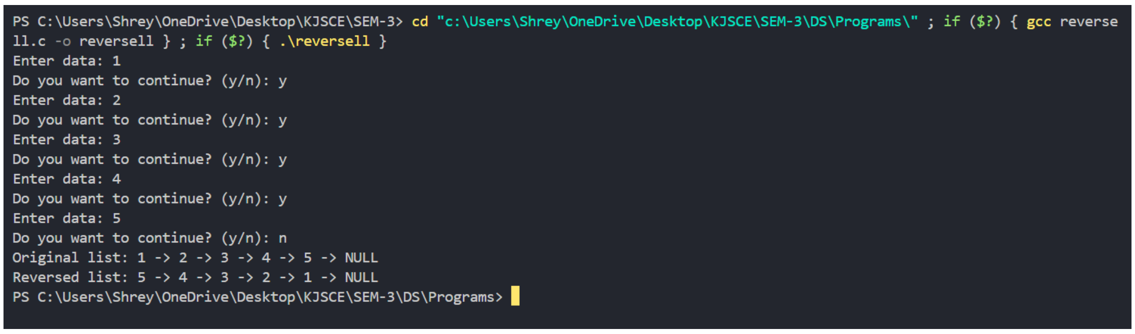
Ans:



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

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

// Function to create a new node with given data
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory error\n");
        exit(0);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create a linked list
Node* createList() {
    Node* head = NULL;
    Node* temp = NULL;
    Node* newNode = NULL;
    int data;
    char ch = 'y';
    while (ch == 'y' || ch == 'Y') {
        printf("Enter data: ");
        scanf("%d", &data);
        newNode = createNode(data);
        if (head == NULL) head = newNode;
        else temp->next = newNode;
        temp = newNode;
```



```c
        while (getchar() != '\n');
        printf("Do you want to continue? (y/n): ");
        scanf("%c", &ch);
    }
    return head;
}

// Function to display the linked list
void printList(Node* head) {
    Node* temp = head;
    while(temp) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to reverse the linked list
Node* reverseList(Node* head) {
    Node* prev = NULL;
    Node* current = head;
    Node* next = NULL;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

int main() {
    Node* head = createList();
    printf("Original list: ");
    printList(head);
    head = reverseList(head);
    printf("Reversed list: ");
    printList(head);
    return 0;
}
```



```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs\" ; if ($?) { gcc reverse
ll.c -o reversell } ; if ($?) { .\reversell }
Enter data: 1
Do you want to continue? (y/n): y
Enter data: 2
Do you want to continue? (y/n): y
Enter data: 3
Do you want to continue? (y/n): y
Enter data: 4
Do you want to continue? (y/n): y
Enter data: 5
Do you want to continue? (y/n): n
Original list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed list: 5 -> 4 -> 3 -> 2 -> 1 -> NULL
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs>
```