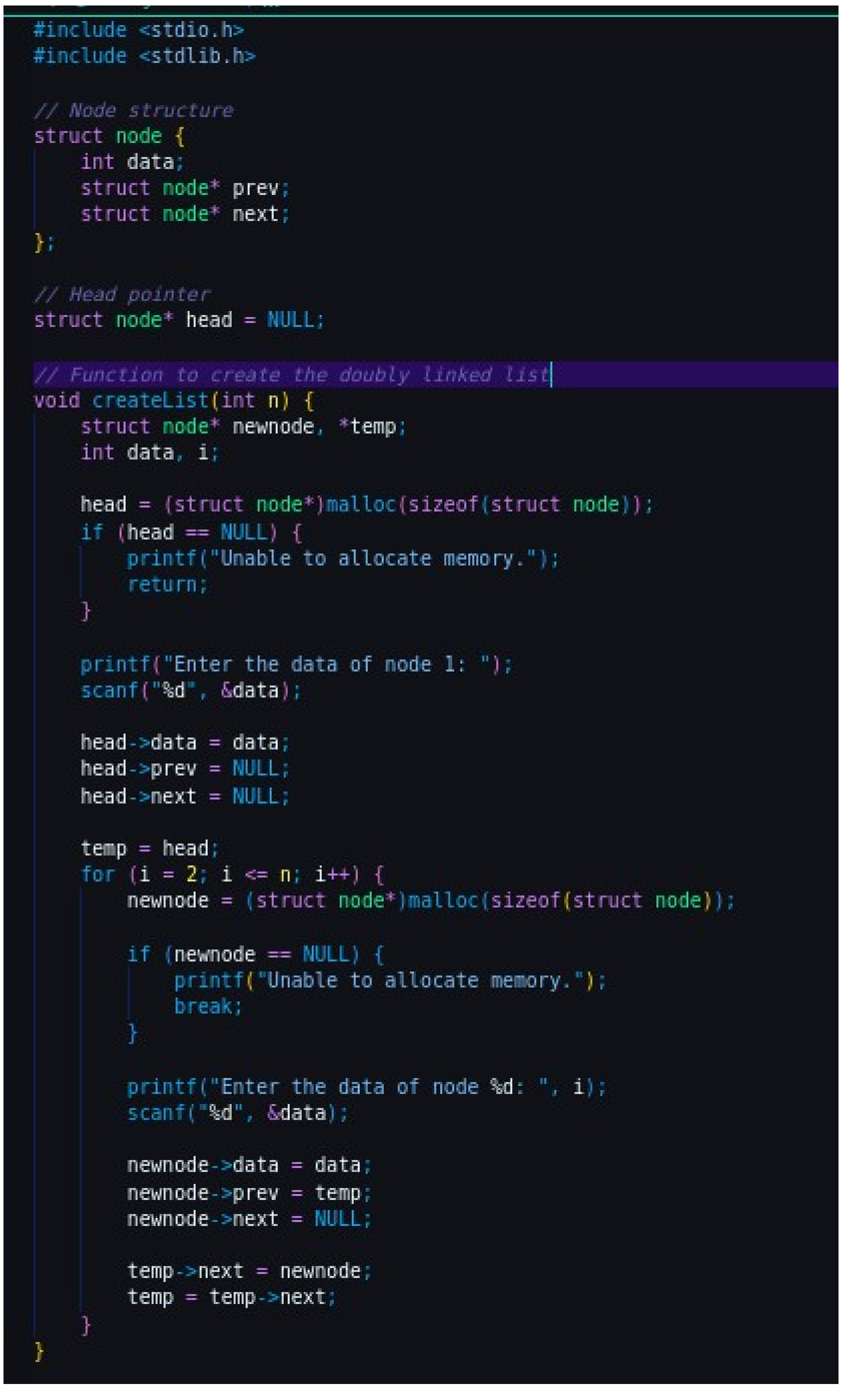
# 22AIE112 Data Structures and Algorithms
# Labsheet 3
# Doubly Linked List

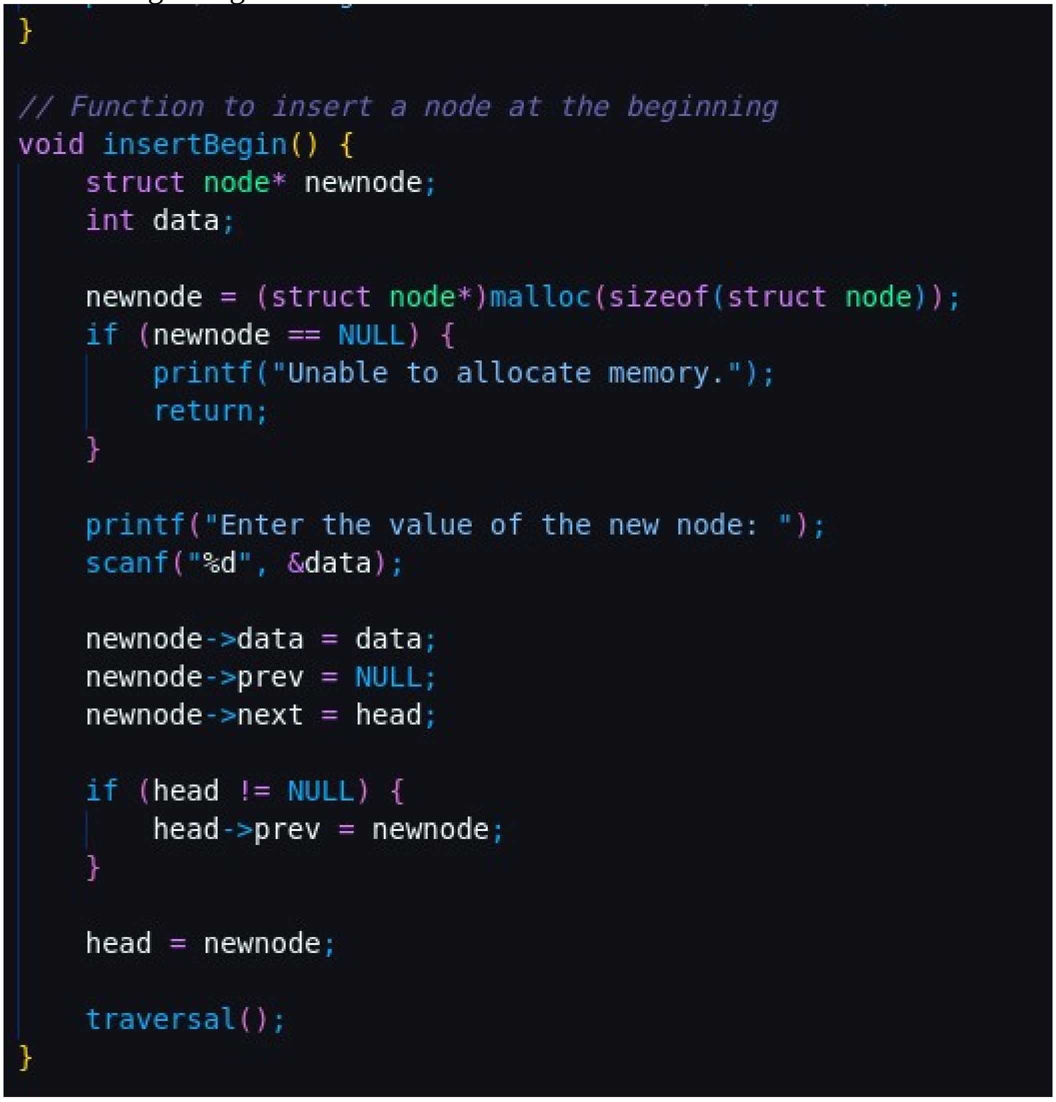
**Date:19//6/2023** **Roll No: AM.EN.U4AIE22009**

**Function to create a doubly linked list**



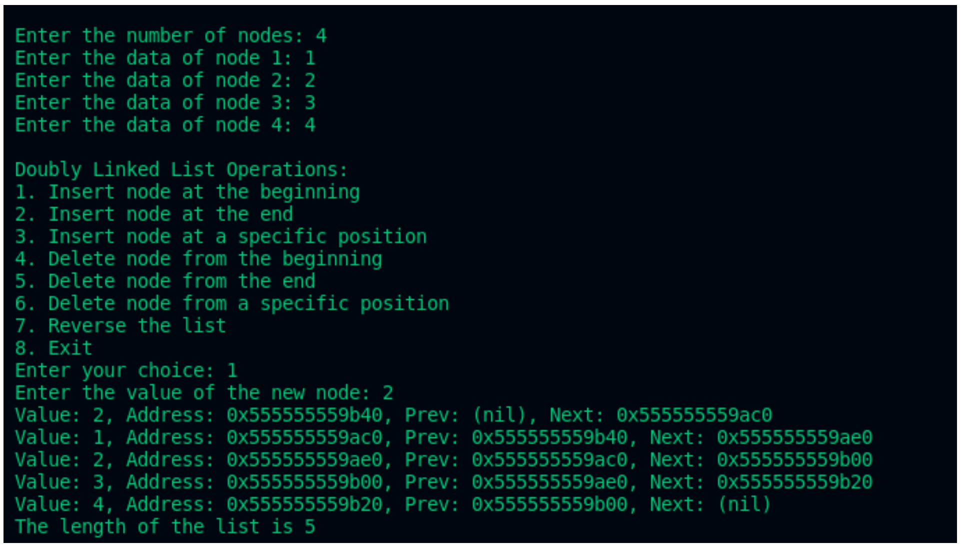
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct node {
    int data;
    struct node* prev;
    struct node* next;
};

// Head pointer
struct node* head = NULL;

// Function to create the doubly linked list
void createList(int n) {
    struct node* newnode, *temp;
    int data, i;

    head = (struct node*)malloc(sizeof(struct node));
    if (head == NULL) {
        printf("Unable to allocate memory.");
        return;
    }

    printf("Enter the data of node 1: ");
    scanf("%d", &data);

    head->data = data;
    head->prev = NULL;
    head->next = NULL;

    temp = head;
    for (i = 2; i <= n; i++) {
        newnode = (struct node*)malloc(sizeof(struct node));

        if (newnode == NULL) {
            printf("Unable to allocate memory.");
            break;
        }

        printf("Enter the data of node %d: ", i);
        scanf("%d", &data);

        newnode->data = data;
        newnode->prev = temp;
        newnode->next = NULL;

        temp->next = newnode;
        temp = temp->next;
    }
}
```

1. Insertion at the beginning



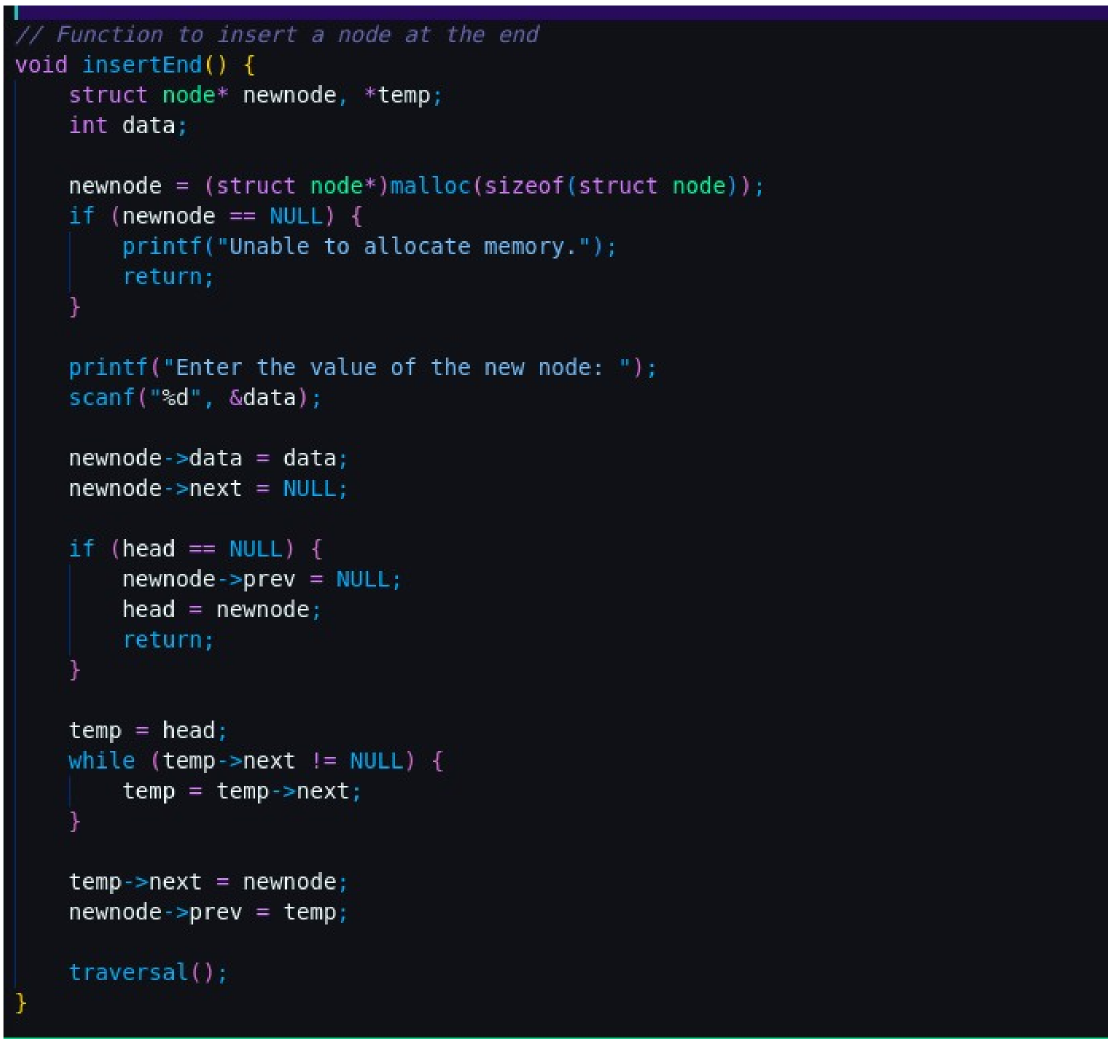
```c
}

// Function to insert a node at the beginning
void insertBegin() {
    struct node* newnode;
    int data;

    newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Unable to allocate memory.");
        return;
    }

    printf("Enter the value of the new node: ");
    scanf("%d", &data);

    newnode->data = data;
    newnode->prev = NULL;
    newnode->next = head;

    if (head != NULL) {
        head->prev = newnode;
    }

    head = newnode;

    traversal();
}
```

Output



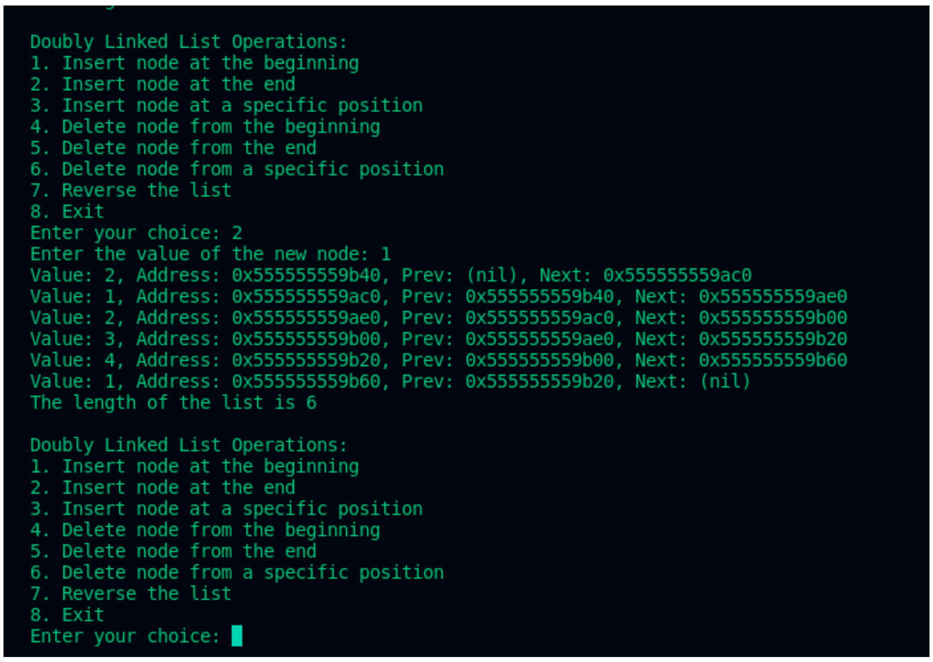
```
Enter the number of nodes: 4
Enter the data of node 1: 1
Enter the data of node 2: 2
Enter the data of node 3: 3
Enter the data of node 4: 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 1
Enter the value of the new node: 2
Value: 2, Address: 0x555555559b40, Prev: (nil), Next: 0x555555559ac0
Value: 1, Address: 0x555555559ac0, Prev: 0x555555559b40, Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559ae0, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: (nil)
The length of the list is 5
```

2.Insertion at end



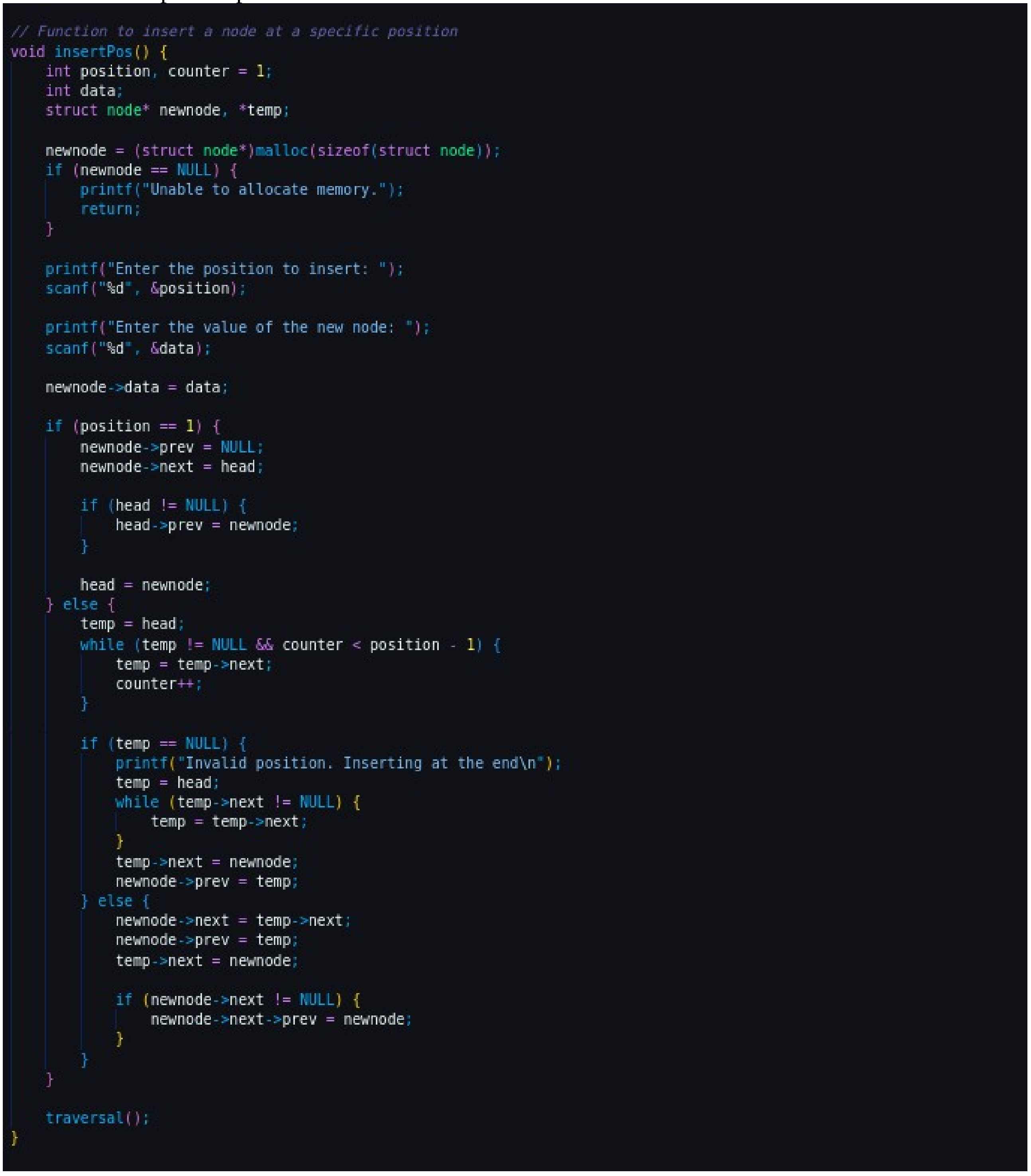
```c
// Function to insert a node at the end
void insertEnd() {
    struct node* newnode, *temp;
    int data;

    newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Unable to allocate memory.");
        return;
    }

    printf("Enter the value of the new node: ");
    scanf("%d", &data);

    newnode->data = data;
    newnode->next = NULL;

    if (head == NULL) {
        newnode->prev = NULL;
        head = newnode;
        return;
    }

    temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    temp->next = newnode;
    newnode->prev = temp;

    traversal();
}
```

Output



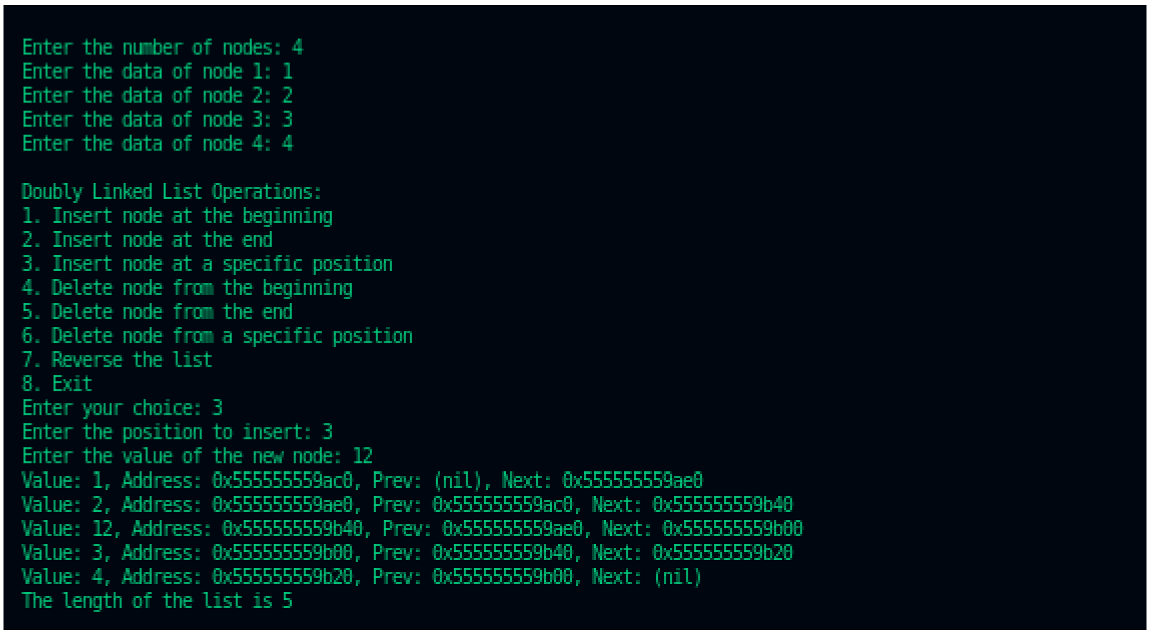
```
Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 2
Enter the value of the new node: 1
Value: 2, Address: 0x555555559b40, Prev: (nil), Next: 0x555555559ac0
Value: 1, Address: 0x555555559ac0, Prev: 0x555555559b40, Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559ae0, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: 0x555555559b60
Value: 1, Address: 0x555555559b60, Prev: 0x555555559b20, Next: (nil)
The length of the list is 6

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 
```

3. Insertion at specific position



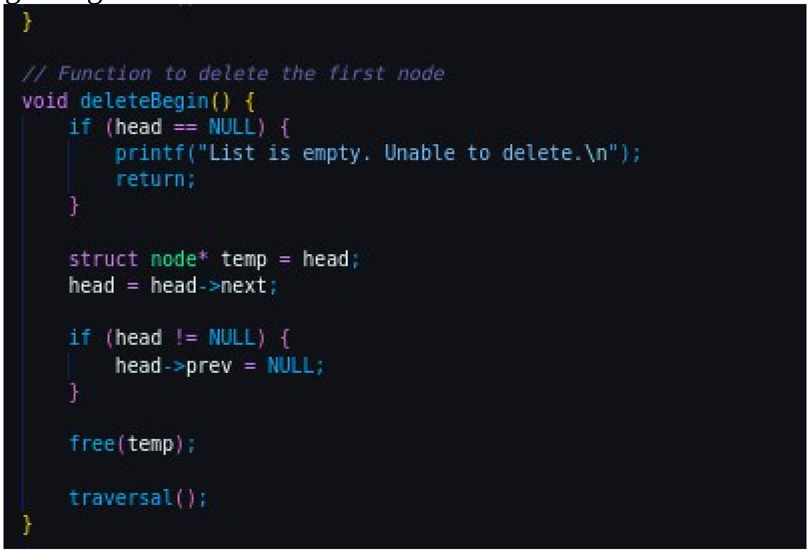
```c
// Function to insert a node at a specific position
void insertPos() {
    int position, counter = 1;
    int data;
    struct node* newnode, *temp;

    newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Unable to allocate memory.");
        return;
    }

    printf("Enter the position to insert: ");
    scanf("%d", &position);

    printf("Enter the value of the new node: ");
    scanf("%d", &data);

    newnode->data = data;

    if (position == 1) {
        newnode->prev = NULL;
        newnode->next = head;

        if (head != NULL) {
            head->prev = newnode;
        }

        head = newnode;
    } else {
        temp = head;
        while (temp != NULL && counter < position - 1) {
            temp = temp->next;
            counter++;
        }

        if (temp == NULL) {
            printf("Invalid position. Inserting at the end\n");
            temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newnode;
            newnode->prev = temp;
        } else {
            newnode->next = temp->next;
            newnode->prev = temp;
            temp->next = newnode;

            if (newnode->next != NULL) {
                newnode->next->prev = newnode;
            }
        }
    }

    traversal();
}
```

Output



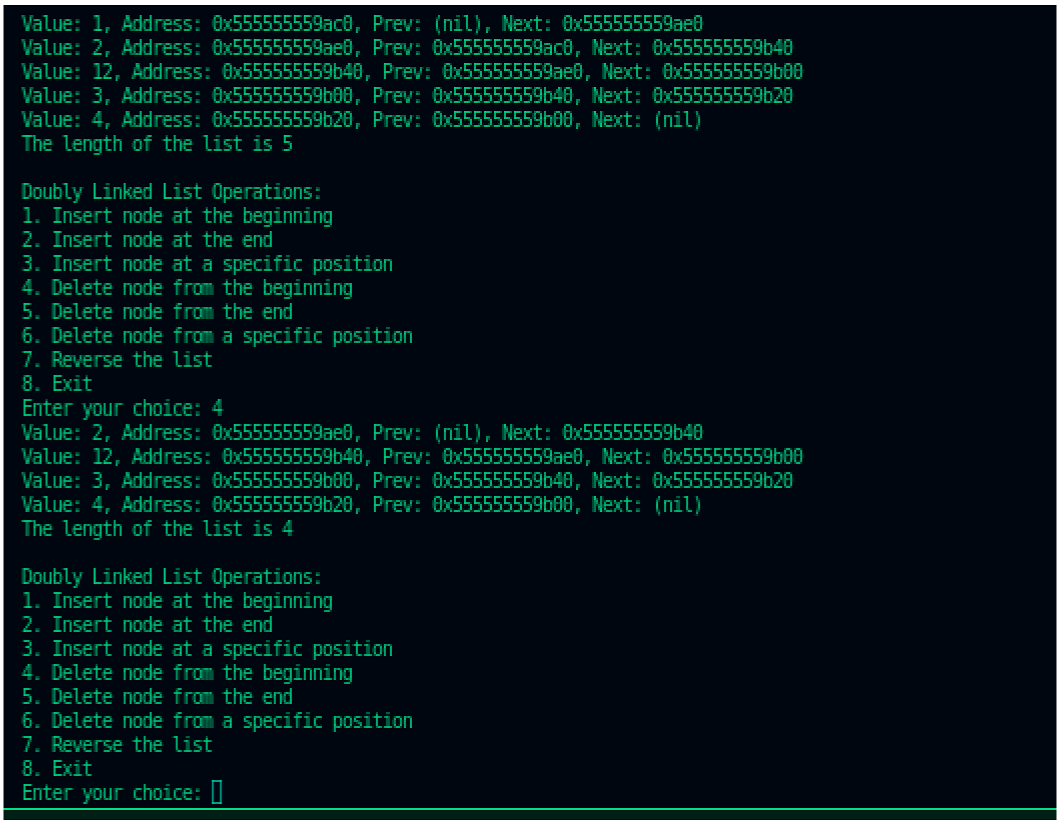
```
Enter the number of nodes: 4
Enter the data of node 1: 1
Enter the data of node 2: 2
Enter the data of node 3: 3
Enter the data of node 4: 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 3
Enter the position to insert: 3
Enter the value of the new node: 12
Value: 1, Address: 0x555555559ac0, Prev: (nil), Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b40
Value: 12, Address: 0x555555559b40, Prev: 0x555555559ae0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559b40, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: (nil)
The length of the list is 5
```

4.Deletion from beginning



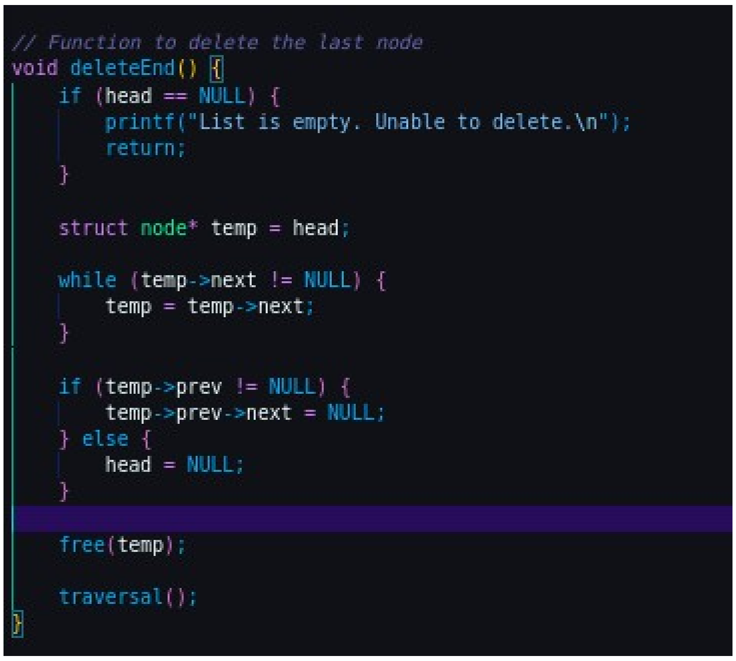
```c
}

// Function to delete the first node
void deleteBegin() {
    if (head == NULL) {
        printf("List is empty. Unable to delete.\n");
        return;
    }

    struct node* temp = head;
    head = head->next;

    if (head != NULL) {
        head->prev = NULL;
    }

    free(temp);

    traversal();
}
```

Output



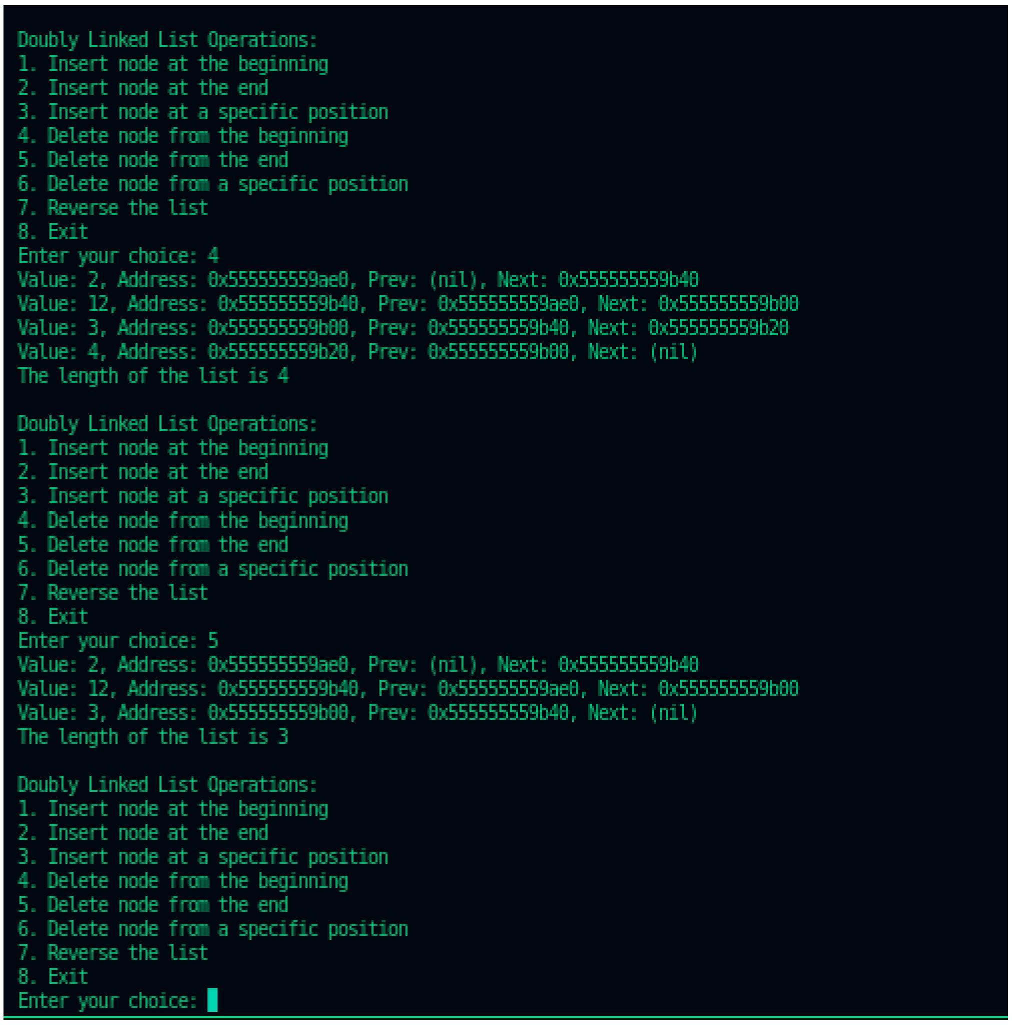
```
Value: 1, Address: 0x555555559ac0, Prev: (nil), Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b40
Value: 12, Address: 0x555555559b40, Prev: 0x555555559ae0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559b40, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: (nil)
The length of the list is 5

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 4
Value: 2, Address: 0x555555559ae0, Prev: (nil), Next: 0x555555559b40
Value: 12, Address: 0x555555559b40, Prev: 0x555555559ae0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559b40, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: (nil)
The length of the list is 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 
```

5. Deletion from end

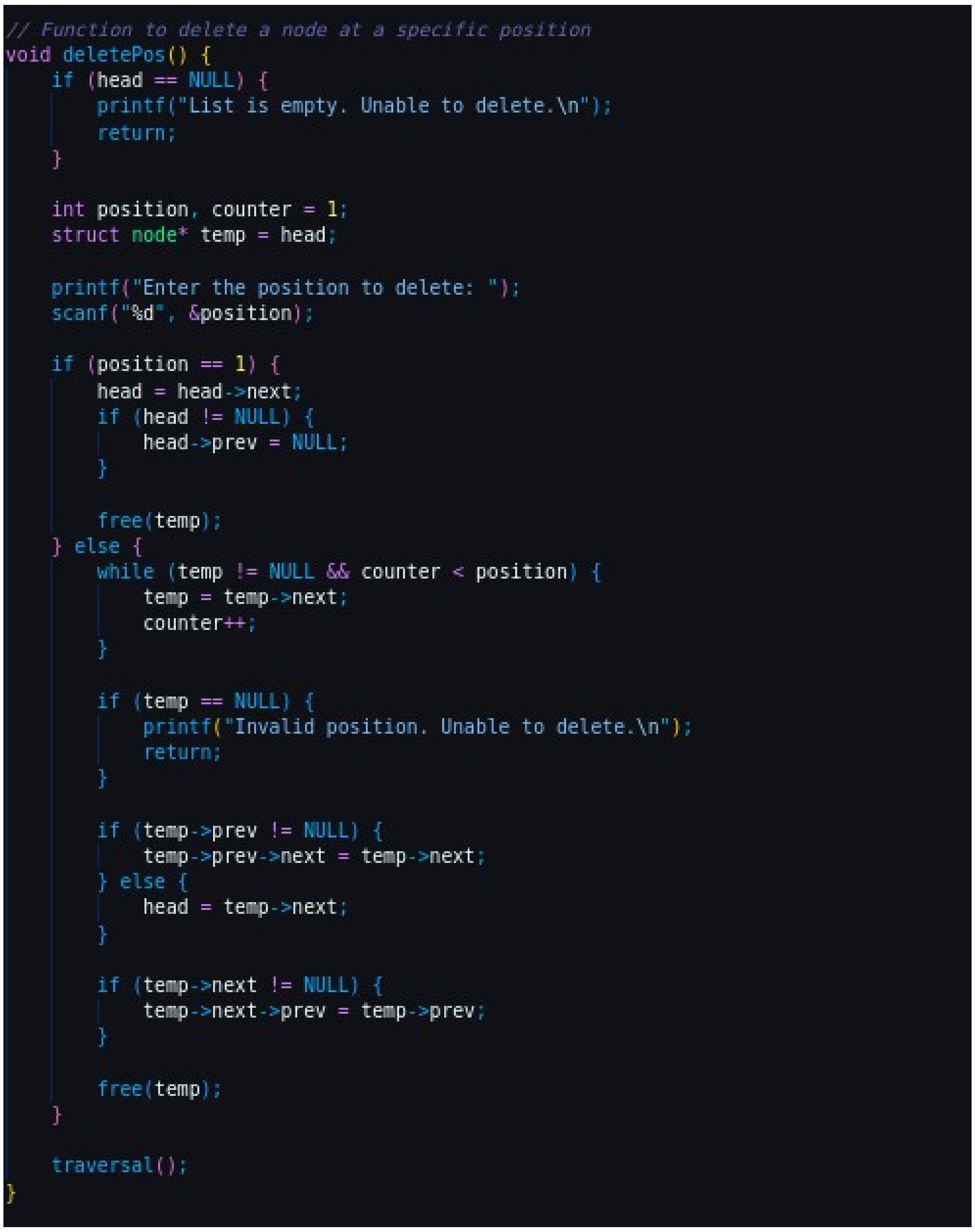


```c
// Function to delete the last node
void deleteEnd() {
    if (head == NULL) {
        printf("List is empty. Unable to delete.\n");
        return;
    }

    struct node* temp = head;

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

    if (temp->prev != NULL) {
        temp->prev->next = NULL;
    } else {
        head = NULL;
    }

    free(temp);

    traversal();
}
```

Output

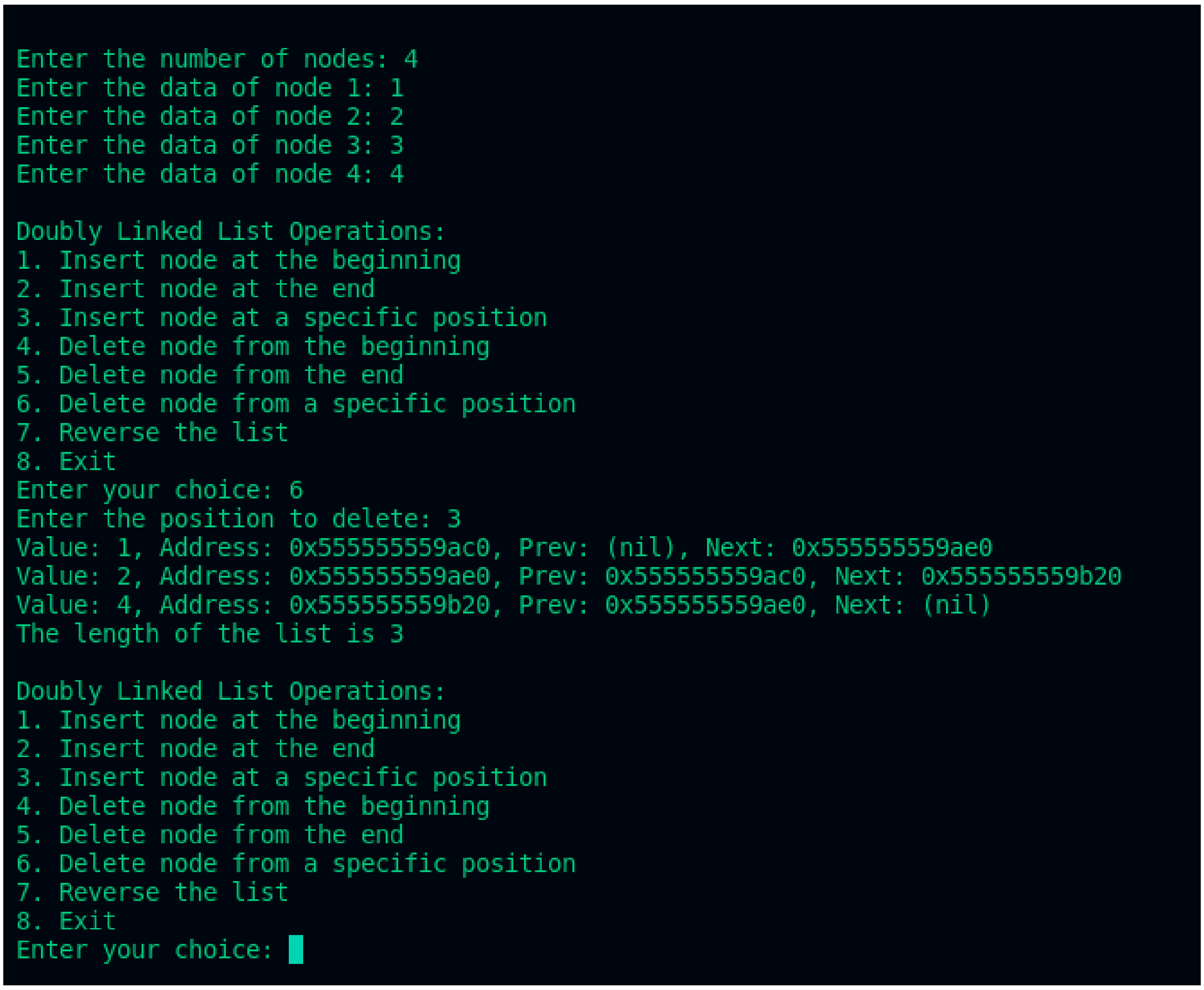

```
Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 4
Value: 2, Address: 0x555555559ae0, Prev: (nil), Next: 0x555555559b40
Value: 12, Address: 0x555555559b40, Prev: 0x555555559ae0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559b40, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559b00, Next: (nil)
The length of the list is 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 5
Value: 2, Address: 0x555555559ae0, Prev: (nil), Next: 0x555555559b40
Value: 12, Address: 0x555555559b40, Prev: 0x555555559ae0, Next: 0x555555559b00
Value: 3, Address: 0x555555559b00, Prev: 0x555555559b40, Next: (nil)
The length of the list is 3

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 
```

6. Deletion from specified position



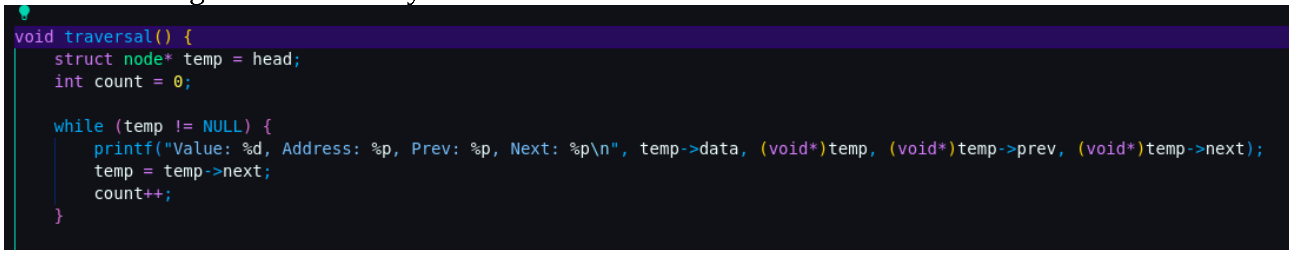
```c
// Function to delete a node at a specific position
void deletePos() {
    if (head == NULL) {
        printf("List is empty. Unable to delete.\n");
        return;
    }

    int position, counter = 1;
    struct node* temp = head;

    printf("Enter the position to delete: ");
    scanf("%d", &position);

    if (position == 1) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }

        free(temp);
    } else {
        while (temp != NULL && counter < position) {
            temp = temp->next;
            counter++;
        }

        if (temp == NULL) {
            printf("Invalid position. Unable to delete.\n");
            return;
        }

        if (temp->prev != NULL) {
            temp->prev->next = temp->next;
        } else {
            head = temp->next;
        }

        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }

        free(temp);
    }

    traversal();
}
```

Output



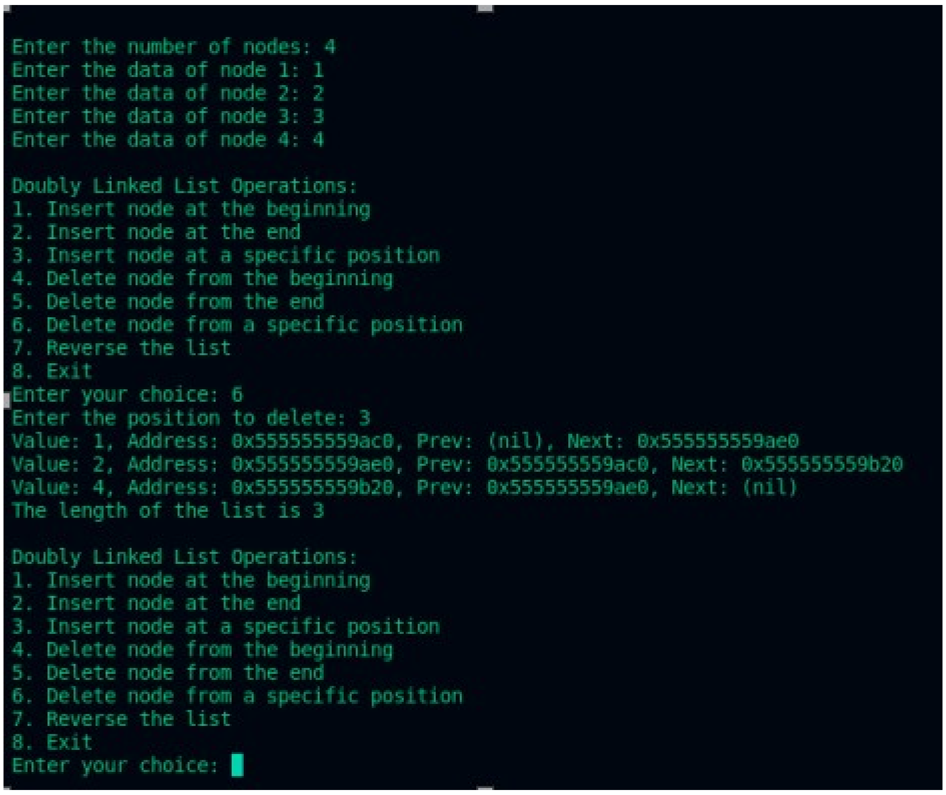
```
Enter the number of nodes: 4
Enter the data of node 1: 1
Enter the data of node 2: 2
Enter the data of node 3: 3
Enter the data of node 4: 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 6
Enter the position to delete: 3
Value: 1, Address: 0x555555559ac0, Prev: (nil), Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559ae0, Next: (nil)
The length of the list is 3

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 
```

7. Find the length of the Doubky Linked List



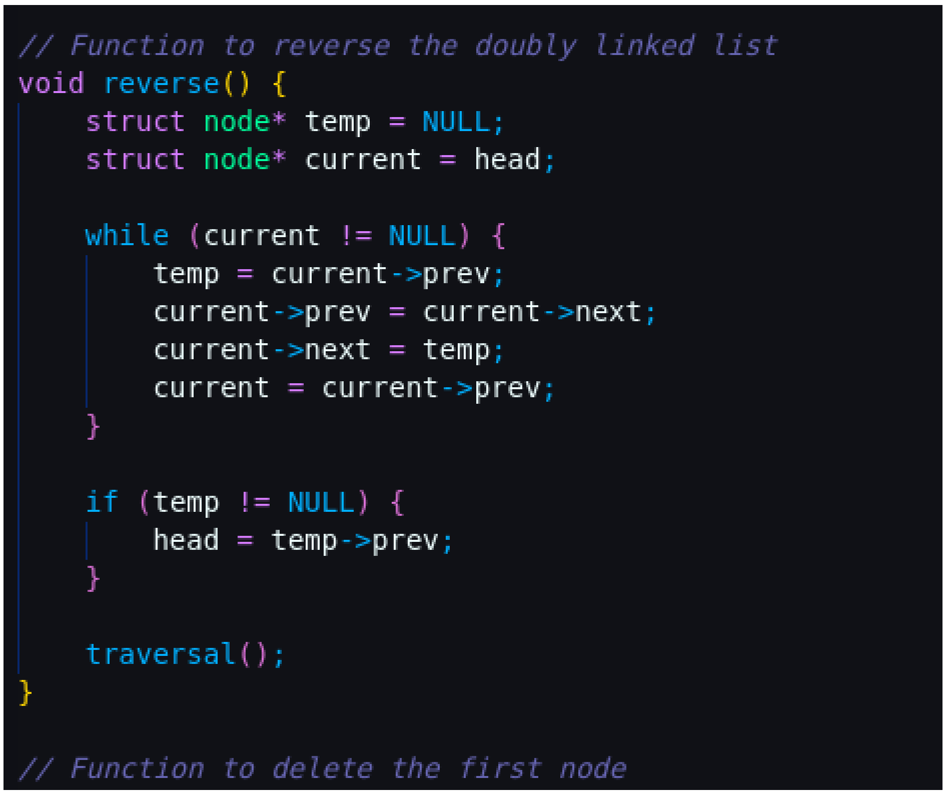
```
void traversal() {
    struct node* temp = head;
    int count = 0;

    while (temp != NULL) {
        printf("Value: %d, Address: %p, Prev: %p, Next: %p\n", temp->data, (void*)temp, (void*)temp->prev, (void*)temp->next);
        temp = temp->next;
        count++;
    }
```

Output



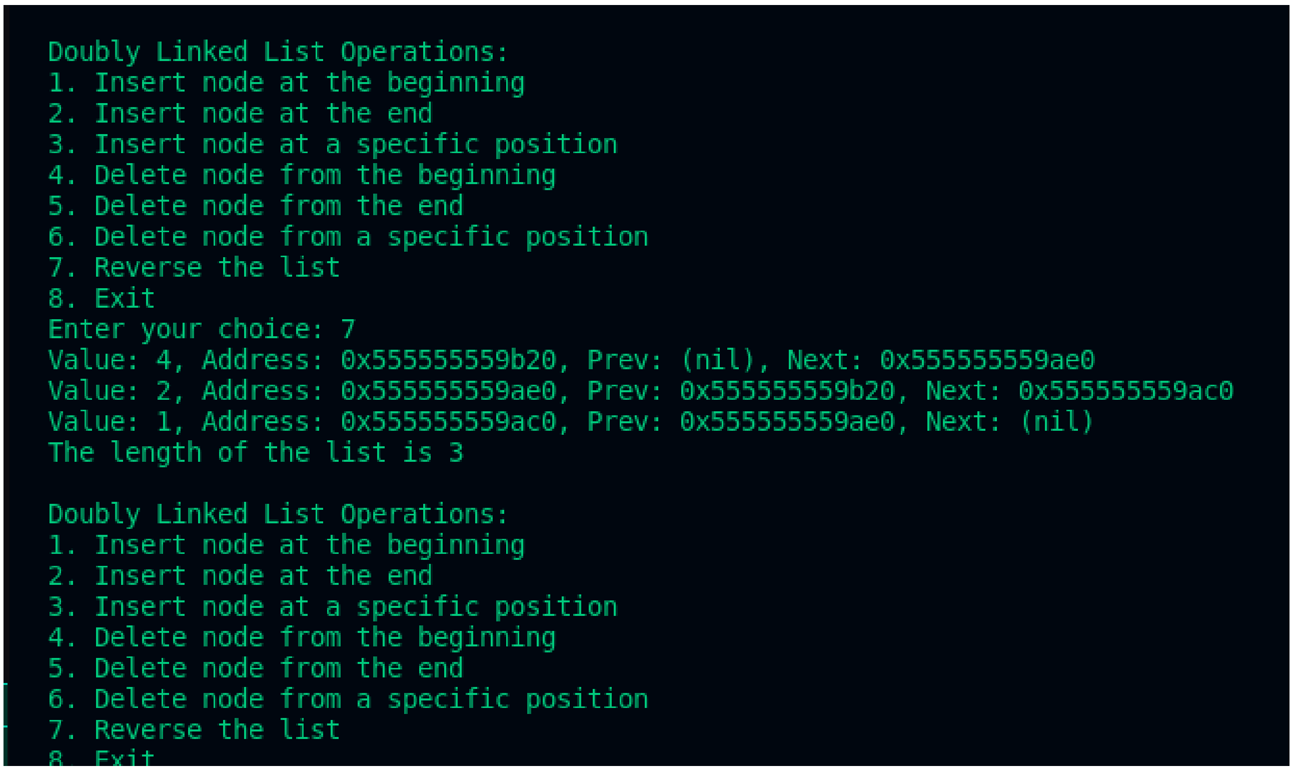
```
Enter the number of nodes: 4
Enter the data of node 1: 1
Enter the data of node 2: 2
Enter the data of node 3: 3
Enter the data of node 4: 4

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 6
Enter the position to delete: 3
Value: 1, Address: 0x555555559ac0, Prev: (nil), Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559ac0, Next: 0x555555559b20
Value: 4, Address: 0x555555559b20, Prev: 0x555555559ae0, Next: (nil)
The length of the list is 3

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 
```

8. Reverse the Doubly Linked List



```c
// Function to reverse the doubly linked list
void reverse() {
    struct node* temp = NULL;
    struct node* current = head;

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

    if (temp != NULL) {
        head = temp->prev;
    }

    traversal();
}

// Function to delete the first node
```

Output


```
Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
Enter your choice: 7
Value: 4, Address: 0x555555559b20, Prev: (nil), Next: 0x555555559ae0
Value: 2, Address: 0x555555559ae0, Prev: 0x555555559b20, Next: 0x555555559ac0
Value: 1, Address: 0x555555559ac0, Prev: 0x555555559ae0, Next: (nil)
The length of the list is 3

Doubly Linked List Operations:
1. Insert node at the beginning
2. Insert node at the end
3. Insert node at a specific position
4. Delete node from the beginning
5. Delete node from the end
6. Delete node from a specific position
7. Reverse the list
8. Exit
```