You are required to implement a singly linked list in C with the following functionalities:

1. Insert at the Beginning: Write a function to insert a node at the beginning of the linked list.

2. Insert at the End: Write a function to insert a node at the end of the linked list.

3. Insert After a Certain Node: Write a function to insert a node after a specified node in the linked list. You may assume that the specified node exists in the list.

4. Insert Before a Certain Node: Write a function to insert a node before a specified node in the linked list. You may assume that the specified node exists in the list.

5. Insert at the nth Position: Write a function to insert a node at the nth position in the linked list. If the position is out of bounds, handle it appropriately.

6. Delete the First Node: Write a function to delete the first node of the linked list.

7. Delete the Last Node: Write a function to delete the last node of the linked list.

8. Delete the nth Node: Write a function to delete the nth node of the linked list. If the position is out of bounds, handle it appropriately.

9. Delete a Node After a Node: Write a function to delete a node after a specified node in the linked list. You may assume that the specified node exists in the list and has a successor.

10. Delete a Node Before a Node: Write a function to delete a node before a specified node in the linked list. You may assume that the specified node exists in the list and has a predecessor.

11. Search for a Node: Write a function to search for a node in the linked list by its value. The function should return the position of the node in the list if found, or indicate that the node is not present.

12. Traverse the Linked List: Write a function to traverse the linked list and print the value of each node.

13. Reverse the Linked List: Write a function to reverse the linked list.

14. Sort the Linked List: Write a function to sort the linked list.

1. Insert at the Beginning

c

Copy code

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

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

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

```c
int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 5);
    insertAtBeginning(&head, 8);
    traverseList(head);
    return 0;
}
```

---

2. Insert at the End

c
Copy code

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

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

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

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

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

int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 5);
    insertAtEnd(&head, 8);
    traverseList(head);
    return 0;
}
```

---

3. Insert After a Certain Node

c
Copy code

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

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

void insertAfterNode(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL\n");
        return;
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = NULL;

    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    second->data = 5;
    second->next = NULL;
    head->next = second;

    insertAfterNode(head, 4);
    traverseList(head);
    return 0;
}
```

---

4. Insert Before a Certain Node

c
Copy code

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

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

```
};

void insertBeforeNode(struct Node** head, int target, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    if (*head == NULL) return;

    if ((*head)->data == target) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

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

    if (temp->next == NULL) return;
    newNode->next = temp->next;
    temp->next = newNode;
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = NULL;
    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    second->data = 5;
    second->next = NULL;
    head->next = second;

    insertBeforeNode(&head, 5, 4);
    traverseList(head);
    return 0;
}
```

---

5. Insert at the nth Position

c
Copy code

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

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

void insertAtNthPosition(struct Node** head, int position, int data) {
    if (position < 1) return;

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) return;
    newNode->next = temp->next;
    temp->next = newNode;
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = NULL;

    insertAtNthPosition(&head, 1, 5);
    insertAtNthPosition(&head, 2, 8);
    insertAtNthPosition(&head, 3, 10);
    traverseList(head);
    return 0;
}
```

---

6. Delete the First Node

c

Copy code

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

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

void deleteFirstNode(struct Node** head) {
    if (*head == NULL) return;

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;

    deleteFirstNode(&head);
    traverseList(head);
    return 0;
}
```

---

7. Delete the Last Node

c
Copy code

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

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

void deleteLastNode(struct Node** head) {
    if (*head == NULL) return;

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }

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

    free(temp->next);
    temp->next = NULL;
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;
    deleteLastNode(&head);
    traverseList(head);
    return 0;
}
```

---

8. Delete the nth Node

c

Copy code

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

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

void deleteNthNode(struct Node** head, int position) {
    if (*head == NULL || position < 1) return;

    struct Node* temp = *head;

    if (position == 1) {
        *head = temp->next;
        free(temp);
        return;
    }

    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) return;

    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;

    deleteNthNode(&head, 2);
    traverseList(head);
    return 0;
}
```

---

9. Delete a Node After a Node

c

Copy code

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

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

void deleteNodeAfter(struct Node* prevNode) {
    if (prevNode == NULL || prevNode->next == NULL) return;
    struct Node* temp = prevNode->next;
    prevNode->next = temp->next;
    free(temp);
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 8;
    head->next->next->next = NULL;

    deleteNodeAfter(head);
    traverseList(head);
    return 0;
}
```

---

10. Delete a Node Before a Node

c
Copy code

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

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

```
};

void deleteNodeBefore(struct Node** head, int target) {
    if (*head == NULL || (*head)->next == NULL) return;

    struct Node* temp = *head;

    if ((*head)->next->data == target) {
        *head = temp->next;
        free(temp);
        return;
    }

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

    if (temp->next->next == NULL) return;

    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 8;
    head->next->next->next = NULL;

    deleteNodeBefore(&head, 8);
    traverseList(head);
    return 0;
}
```

---

11. Search for a Node

c
Copy code

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

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

int searchNode(struct Node* head, int value) {
    int position = 1;
    struct Node* temp = head;

    while (temp != NULL) {
        if (temp->data == value) {
            return position;
        }
        temp = temp->next;
        position++;
    }
    return -1; // Value not found
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;

    int position = searchNode(head, 5);
    if (position != -1)
        printf("Value found at position: %d\n", position);
    else
        printf("Value not found.\n");

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

---

12. Traverse the Linked List

c
Copy code

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

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

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;

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

---

13. Reverse the Linked List

c

Copy code

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

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

void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;

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

```c
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 3;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 5;
    head->next->next = NULL;

    reverseList(&head);
    traverseList(head);
    return 0;
}
```

---

14. Sort the Linked List

c
Copy code

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

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

```c
};

void sortList(struct Node* head) {
    struct Node* i = head;
    struct Node* j = NULL;
    int temp;

    if (head == NULL) return;

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

int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 5;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 3;
    head->next->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->next->data = 8;
    head->next->next->next = NULL;

    sortList(head);
    traverseList(head);
    return 0;
}
```