```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *next;

   struct Node *prev;

};

struct Node *header = NULL;

struct Node *createNode(int data) {

   struct Node *newnode;

   newnode = (struct Node *)malloc(sizeof(struct Node));

   newnode->data = data;

   newnode->next = NULL;

   newnode->prev = NULL;

   return newnode;

}
```

```
void checkIfEmpty() {
    if (header == NULL) {
        printf("The list is currently empty.\n");
    }
}

void insertAtFront(int data) {
    struct Node *newnode = createNode(data);
    if (header == NULL) {
        header = newnode;
    } else {
        newnode->next = header;
        header->prev = newnode;
        header = newnode;
    }
    printf("Node with value %d inserted at the front.\n", data);
}

void insertAtEnd(int data) {
    struct Node *newnode = createNode(data);
    if (header == NULL) {
        header = newnode;
        printf("Node with value %d inserted at position 1.\n", data);
    } else {
        struct Node *current = header;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newnode;
        newnode->prev = current;
        printf("Node with value %d inserted at the end.\n", data);
    }
}

void insertAtAny_secondMethod(int data, int key) {
    struct Node *newnode = createNode(data);
    struct Node *ptr = header;
    if (header == NULL) {
        header = newnode;
    } else {
        while (ptr != NULL && ptr->data != key) {
            ptr = ptr->next;
        }
        if (ptr == NULL) {
            printf("Key not found in the list \n");
            free(newnode);
            return;
        }
        newnode->next = ptr->next;
        ptr->next = newnode;
        newnode->prev = ptr;

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

```
void insertAtAny(int data, int pos) {
    struct Node *newnode = createNode(data);
    struct Node *ptr = header, *prev;
    int currentPos = 1;
    if (pos == 1) {
        newnode->next = header;
        header = newnode;
    } else if (pos == 0) {
        printf("Node Starts at 1\n");
    } else {
        while (ptr != NULL && currentPos < pos) {
            prev = ptr;
            ptr = ptr->next;
            currentPos++;
        }
        if (currentPos == pos) {
            newnode->prev = prev;
            newnode->next = ptr;
            prev->next = newnode;
            if (ptr != NULL)
                ptr->prev = newnode;
        } else {
            printf("Position not found\n");
            free(newnode);
        }
    }
}
```

```
void deleteAtFront() {
    if (header == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    struct Node *temp = header;
    printf("Node with value %d deleted from the front.\n", temp->data);
    header = header->next;
    if (header != NULL) {
        header->prev = NULL;
    }
    free(temp);
}

void deleteAtEnd() {
    if (header == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    struct Node *current = header;
    if (current->next == NULL) {
        printf("Node with value %d deleted from the end.\n", current->data);
        free(header);
        header = NULL;
        return;
    }
    while (current->next != NULL) {
        current = current->next;
    }
    printf("Node with value %d deleted from the end.\n", current->data);
    current->prev->next = NULL;
    free(current);
}

void deleteAtAny(int position) {
    if (header == NULL) {
        checkIfEmpty();
        return;
    }
    if (position == 1) {
        deleteAtFront();
        return;
    }
    struct Node *current = header;
    int i;
    for (i = 1; i < position && current != NULL; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Position out of bounds. Nothing to delete.\n");
        return;
    }
    printf("Node with value %d deleted from position %d.\n", current->data, position);
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    if (current->prev != NULL) {
        current->prev->next = current->next;
    }
    free(current);
}

int search(int key) {
    struct Node *current = header;
    int position = 1;
    while (current != NULL) {
        if (current->data == key) {
            return position;
        }
        current = current->next;
        position++;
    }
    return -1;
}

void traversal() {
    if (header == NULL) {
        printf("The list is currently empty.\n");
        return;
    }
    struct Node *ptr = header;
    int position = 1;
    while (ptr != NULL) {
        printf("%d (%d) ", ptr->data, position);
        ptr = ptr->next;
        position++;
    }
    printf("\n");
}

int main() {
    int choice, data, position, key;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Any Position\n");
        printf("4. Delete at Front\n");
        printf("5. Delete at End\n");
        printf("6. Delete at Any Position\n");
        printf("7. Search in List\n");
        printf("8. Display List\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {

    case 1:

        printf("Enter data to insert at front: ");

        scanf("%d", &data);

        insertAtFront(data);

        break;

    case 2:

        printf("Enter data to insert at end: ");

        scanf("%d", &data);

        insertAtEnd(data);

        break;

    case 3:

        printf("Enter key to insert after: ");

        scanf("%d", &key);

        printf("Enter data: ");

        scanf("%d", &data);

        insertAtAny_secondMethod(data, key);

        break;

    case 4:

        deleteAtFront();

        break;

    case 5:

        deleteAtEnd();

        break;

    case 6:

        printf("Enter the position to delete: ");

        scanf("%d", &position);

        deleteAtAny(position);

        break;
```

```
            case 7:
                printf("Enter data to search: ");
                scanf("%d", &data);
                position = search(data);
                if (position == -1) {
                    printf("Data not found\n");
                } else {
                    printf("Data found at position %d\n", position);
                }
                break;
            case 8:
                printf("Current List: ");
                traversal();
                break;
            case 9:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```