Singly Linked List:

```c
// Define a structure for a node in the singly linked list
struct Node {
    int data;
    struct Node* next;
}

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert data at the end of the linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

// Function to insert data at the beginning of the linked list
void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

```

```c
// Function to check if the linked list is empty
int isEmpty(struct Node* head) {
    return head == NULL;
}

// Function to delete data from the start of the linked list
void deleteAtStart(struct Node** head) {
    if (*head == NULL) {
        printf("Linked list is empty. Nothing to delete.\n");
        return;
    }

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

```c
// Function to insert data at a specific index position
void insertAtIndex(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        insertAtStart(head, data);
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;
    int currentPos = 0;
    while (current != NULL && currentPos < position - 1) {
        current = current->next;
        currentPos++;
    }

    if (current == NULL) {
        printf("Invalid position\n");
        free(newNode);
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
}
```

```c
// Function to delete data from the end of the linked list
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("Linked list is empty. Nothing to delete.\n");
        return;
    }

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

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

    free(current->next);
    current->next = NULL;
}
```

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

```c
// Function to delete data at a specific index position
void deleteAtIndex(struct Node** head, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        deleteAtStart(head);
        return;
    }

    struct Node* current = *head;
    int currentPos = 0;
    while (current != NULL && currentPos < position - 1) {
        current = current->next;
        currentPos++;
    }

    if (current == NULL || current->next == NULL) {
        printf("Invalid position\n");
    } else {
        struct Node* temp = current->next;
        current->next = temp->next;
        free(temp);
    }
}
```

```c
int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("1. Insert at the end\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at a specific position\n");
        printf("4. Check if the linked list is empty\n");
        printf("5. Delete from the beginning\n");
        printf("6. Delete from the end\n");
        printf("7. Delete from a specific position\n");
        printf("8. Display the linked list\n");
        printf("9. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```c
        switch (choice) {
            case 1:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 2:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtStart(&head, data);
                break;
```

```
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtIndex(&head, data, position);
                break;

            case 4:
                if (isEmpty(head)) {
                    printf("Linked list is empty.\n");
                } else {
                    printf("Linked list is not empty.\n");
                }
                break;

            case 5:
                deleteAtStart(&head);
                break;

            case 6:
                deleteAtEnd(&head);
                break;

            case 7:
                printf("Enter the position to delete from: ");
                scanf("%d", &position);
                deleteAtIndex(&head, position);
                break;

            case 8:
                displayList(head);
                break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    }

    return 0;
}
```

Doubly Linked List:

```c
// Define a structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert data at the end of the doubly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

```

```c
// Function to insert data at the beginning of the doubly linked list
void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}

// Function to check if the doubly linked list is empty
int isEmpty(struct Node* head) {
    return head == NULL;
}
```

```c
// Function to insert data at a specific index position
void insertAtIndex(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        insertAtStart(head, data);
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;
    int currentPos = 0;
    while (current != NULL && currentPos < position - 1) {
        current = current->next;
        currentPos++;
    }

    if (current == NULL) {
        printf("Invalid position\n");
        free(newNode);
    } else {
        newNode->prev = current;
        newNode->next = current->next;
        if (current->next != NULL) {
            current->next->prev = newNode;
        }
        current->next = newNode;
    }
}
```

```c
// Function to delete data from the start of the doubly linked list
void deleteAtStart(struct Node** head) {
    if (*head == NULL) {
        printf("Doubly linked list is empty. Nothing to delete.\n");
        return;
    }

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

// Function to delete data from the end of the doubly linked list
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("Doubly linked list is empty. Nothing to delete.\n");
        return;
    }

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

    if (current->prev != NULL) {
        current->prev->next = NULL;
    } else {
        *head = NULL;
    }
    free(current);
}
```

```c
// Function to delete data at a specific index position
void deleteAtIndex(struct Node** head, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        deleteAtStart(head);
        return;
    }

    struct Node* current = *head;
    int currentPos = 0;
    while (current != NULL && currentPos < position) {
        current = current->next;
        currentPos++;
    }

    if (current == NULL) {
        printf("Invalid position\n");
    } else {
        if (current->prev != NULL) {
            current->prev->next = current->next;
        } else {
            *head = current->next;
        }
        if (current->next != NULL) {
            current->next->prev = current->prev;
        }
        free(current);
    }
}
```

```c
// Function to print the doubly linked list in both forward and backward directions
void displayList(struct Node* head) {
    struct Node* current = head;

    printf("Forward: ");
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");

    current = head;
    printf("Backward: ");
    while (current->next != NULL) {
        current = current->next;
    }
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->prev;
    }
    printf("NULL\n");
}

```

```c
int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("1. Insert at the end\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at a specific position\n");
        printf("4. Check if the doubly linked list is empty\n");
        printf("5. Delete from the beginning\n");
        printf("6. Delete from the end\n");
        printf("7. Delete from a specific position\n");
        printf("8. Display the doubly linked list\n");
        printf("9. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```c
        switch (choice) {
            case 1:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 2:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtStart(&head, data);
                break;

            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtIndex(&head, data, position);
                break;

            case 4:
                if (isEmpty(head)) {
                    printf("Doubly linked list is empty.\n");
                } else {
                    printf("Doubly linked list is not empty.\n");
                }
                break;

            case 5:
                deleteAtStart(&head);
                break;
```

```
            case 6:
                deleteAtEnd(&head);
                break;

            case 7:
                printf("Enter the position to delete from: ");
                scanf("%d", &position);
                deleteAtIndex(&head, position);
                break;

            case 8:
                displayList(head);
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Circular LinkedList:

```
// Define a structure for a node in the circular linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert data at the end of the circular linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
    } else {
        struct Node* tail = (*head)->next;
        while (tail->next != *head) {
            tail = tail->next;
        }
        tail->next = newNode;
        newNode->next = *head;
    }
}
```

```c
// Function to insert data at the beginning of the circular linked list
void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
    } else {
        struct Node* tail = (*head)->next;
        while (tail->next != *head) {
            tail = tail->next;
        }
        newNode->next = *head;
        tail->next = newNode;
        *head = newNode;
    }
}
```

```c
// Function to insert data at a specific index position
void insertAtIndex(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        insertAtStart(head, data);
        return;
    }

    struct Node* newNode = createNode(data);
    if (*head == NULL) {
    printf("List is empty. Insertion at the specified position is not possible.\n");
        free(newNode);
        return;
    }

    struct Node* current = *head;
    int currentPos = 0;
    while (current != *head && currentPos < position - 1) {
        current = current->next;
        currentPos++;
    }

    if (current == *head) {
        printf("Invalid position\n");
        free(newNode);
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
}

```

```c
// Function to check if the circular linked list is empty
int isEmpty(struct Node* head) {
    return head == NULL;
}

// Function to delete data from the start of the circular linked list
void deleteAtStart(struct Node** head) {
    if (*head == NULL) {
        printf("Circular linked list is empty. Nothing to delete.\n");
        return;
    }

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

    if (*head == tail) {
        free(*head);
        *head = NULL;
    } else {
        struct Node* temp = *head;
        *head = (*head)->next;
        tail->next = *head;
        free(temp);
    }
}

```

```c
// Function to delete data from the end of the circular linked list
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("Circular linked list is empty. Nothing to delete.\n");
        return;
    }

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

    if (prev == NULL) {
        free(*head);
        *head = NULL;
    } else {
        prev->next = *head;
        free(current);
    }
}

```

```c
// Function to delete data at a specific index position
void deleteAtIndex(struct Node** head, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        deleteAtStart(head);
        return;
    }

    if (*head == NULL) {
        printf("Circular linked list is empty. Nothing to delete.\n");
        return;
    }

    struct Node* current = *head;
    struct Node* prev = NULL;
    int currentPos = 0;
    while (current->next != *head && currentPos < position) {
        prev = current;
        current = current->next;
        currentPos++;
    }

    if (current == *head) {
        printf("Invalid position\n");
    } else {
        prev->next = current->next;
        free(current);
    }
}

```

```c
// Function to print the circular linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("Circular linked list is empty.\n");
        return;
    }

    struct Node* current = head;
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);
    printf("Head\n");
}

```

```c
int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("1. Insert at the end\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at a specific position\n");
        printf("4. Check if the circular linked list is empty\n");
        printf("5. Delete from the beginning\n");
        printf("6. Delete from the end\n");
        printf("7. Delete from a specific position\n");
        printf("8. Display the circular linked list\n");
        printf("9. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 2:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtStart(&head, data);
                break;
```

```c
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtIndex(&head, data, position);
                break;

            case 4:
                if (isEmpty(head)) {
                    printf("Circular linked list is empty.\n");
                } else {
                    printf("Circular linked list is not empty.\n");
                }
                break;

            case 5:
                deleteAtStart(&head);
                break;

            case 6:
                deleteAtEnd(&head);
                break;
```

```
            case 7:
                printf("Enter the position to delete from: ");
                scanf("%d", &position);
                deleteAtIndex(&head, position);
                break;

            case 8:
                displayList(head);
                break;

            case 9:
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Circular Doubly Linked List:

```
// Define a structure for a node in the circular doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert data at the end of the circular doubly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    } else {
        struct Node* tail = (*head)->prev;
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = *head;
        (*head)->prev = newNode;
    }
}

```

```c
// Function to insert data at the beginning of the circular doubly linked list
void insertAtStart(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    } else {
        newNode->next = *head;
        newNode->prev = (*head)->prev;
        (*head)->prev->next = newNode;
        (*head)->prev = newNode;
        *head = newNode;
    }
}

```

```c
// Function to insert data at a specific index position
void insertAtIndex(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }
    if (position == 0) {
        insertAtStart(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        printf("List is empty. Insertion at the specified position is not possible.\n"
        free(newNode);
        return;
    }
    struct Node* current = *head;
    int currentPos = 0;
    while (current != *head && currentPos < position - 1) {
        current = current->next;
        currentPos++;
    }
    if (current == *head) {
        printf("Invalid position\n");
        free(newNode);
    } else {
        newNode->next = current->next;
        newNode->prev = current;
        current->next->prev = newNode;
        current->next = newNode;
    }
}

```

```c
// Function to check if the circular doubly linked list is empty
int isEmpty(struct Node* head) {
    return head == NULL;
}

```

```c
// Function to delete data from the start of the circular doubly linked list
void deleteAtStart(struct Node** head) {
    if (*head == NULL) {
        printf("Circular doubly linked list is empty. Nothing to delete.\n");
        return;
    }
    struct Node* tail = (*head)->prev;
    if (*head == tail) {
        free(*head);
        *head = NULL;
    } else {
        *head = (*head)->next;
        (*head)->prev = tail;
        tail->next = *head;
    }
}

// Function to delete data from the end of the circular doubly linked list
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("Circular doubly linked list is empty. Nothing to delete.\n");
        return;
    }

    struct Node* tail = (*head)->prev;
    if (*head == tail) {
        free(*head);
        *head = NULL;
    } else {
        tail->prev->next = *head;
        (*head)->prev = tail->prev;
        free(tail);
    }
}
```

```c
// Function to delete data at a specific index position
void deleteAtIndex(struct Node** head, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }
    if (position == 0) {
        deleteAtStart(head);
        return;
    }
    if (*head == NULL) {
        printf("Circular doubly linked list is empty. Nothing to delete.\n");
        return;
    }
    struct Node* current = *head;
    int currentPos = 0;
    while (current != *head && currentPos < position) {
        current = current->next;
        currentPos++;
    }
    if (current == *head) {
        printf("Invalid position\n");
    } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
        free(current);
    }
}

```

```c
// Function to print the circular doubly linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("Circular doubly linked list is empty.\n");
        return;
    }

    struct Node* current = head;
    do {
        printf("%d <-> ", current->data);
        current = current->next;
    } while (current != head);
    printf("Head\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("1. Insert at the end\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at a specific position\n");
        printf("4. Check if the circular doubly linked list is empty\n");
        printf("5. Delete from the beginning\n");
        printf("6. Delete from the end\n");
        printf("7. Delete from a specific position\n");
        printf("8. Display the circular doubly linked list\n");
        printf("9. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 2:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtStart(&head, data);
                break;

            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtIndex(&head, data, position);
                break;
```

```
207             case 4:
208                 if (isEmpty(head)) {
209                     printf("Circular doubly linked list is empty.\n");
210                 } else {
211                     printf("Circular doubly linked list is not empty.\n");
212                 }
213                 break;
214             case 5:
215                 deleteAtStart(&head);
216                 break;
217             case 6:
218                 deleteAtEnd(&head);
219                 break;
220             case 7:
221                 printf("Enter the position to delete from: ");
222                 scanf("%d", &position);
223                 deleteAtIndex(&head, position);
224                 break;
225             case 8:
226                 displayList(head);
227                 break;
228             default:
229                 printf("Invalid choice. Please try again.\n");
230         }
231     }
232     return 0;
233 }
```