```
//Doubly circular linked list
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *prev;
    struct node *next;
} *head = NULL;

// Create the doubly circular linked list
void create() {
    int num;
    struct node *newnode, *last = NULL;

    while (1) {
        printf("Enter a number (-1 to stop): ");
        scanf("%d", &num);
        if (num == -1)
            break;

        newnode = (struct node *)malloc(sizeof(struct node));
        if (!newnode) {
            printf("Memory allocation failed.\n");
            return;
        }

        newnode->data = num;

        if (head == NULL) {
            head = newnode;
            newnode->prev = newnode->next = newnode;
        } else {
            last = head->prev;

            newnode->next = head;
            newnode->prev = last;

            last->next = newnode;
            head->prev = newnode;
        }
    }
}

// Insert at beginning
void insertAtBeginning(int data) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (!newnode) {
        printf("Memory allocation failed!\n");
        return;
    }
    newnode->data = data;

    if (head == NULL) {
```

```c
        head = newnode;
        newnode->next = newnode->prev = newnode;
    } else {
        struct node *last = head->prev;

        newnode->next = head;
        newnode->prev = last;
        head->prev = newnode;
        last->next = newnode;
        head = newnode;
    }

    printf("Inserted %d at the beginning.\n", data);
}

// Insert at end
void insertAtEnd(int data) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (!newnode) {
        printf("Memory allocation failed!\n");
        return;
    }

    newnode->data = data;

    if (head == NULL) {
        head = newnode;
        newnode->next = newnode->prev = newnode;
    } else {
        struct node *last = head->prev;

        newnode->next = head;
        newnode->prev = last;
        last->next = newnode;
        head->prev = newnode;
    }

    printf("Inserted %d at the end.\n", data);
}

// Insert at position
void insertAtPosition(int data, int position) {
    if (position == 0) {
        insertAtBeginning(data);
        return;
    }

    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (!newnode) {
        printf("Memory allocation failed!\n");
        return;
    }

    newnode->data = data;
    struct node *temp = head;

    for (int i = 0; i < position - 1; i++) {
        temp = temp->next;
        if (temp == head) {
            printf("Position out of bounds.\n");
            free(newnode);
            return;
        }
    }

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

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

    printf("Inserted %d at position %d.\n", data, position);
}
```

```
// Delete from beginning
void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *temp = head;

    if (head->next == head) {
        head = NULL;
    } else {
        struct node *last = head->prev;

        head = head->next;
        head->prev = last;
        last->next = head;
    }

    printf("Deleted %d from beginning.\n", temp->data);
    free(temp);
}
```

```
// Delete from end
void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *last = head->prev;

    if (last == head) {
        printf("Deleted %d from end.\n", last->data);
        free(last);
        head = NULL;
        return;
    }

    struct node *secondLast = last->prev;
    secondLast->next = head;
    head->prev = secondLast;

    printf("Deleted %d from end.\n", last->data);
    free(last);
}
```

// **Delete from specific position**

```
void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if (position == 0) {
        deleteFromBeginning();
        return;
    }

    struct node *temp = head;

    for (int i = 0; i < position; i++) {
        temp = temp->next;
        if (temp == head) {
            printf("Position out of bounds.\n");
            return;
        }
    }
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    printf("Deleted %d from position %d.\n", temp->data, position);
    free(temp);
}
```

// **Display the list**

```
void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *temp = head;
    printf("Doubly Circular Linked List: ");
    do {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    } while (temp != head);
```