//Singly Circular Linked list

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

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

// Create 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;
        newnode->next = NULL;

        if (head == NULL) {
            head = newnode;
            newnode->next = head;
        } else {
            last->next = newnode;
            newnode->next = head;
        }
        last = 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) {
        head = newnode;
        newnode->next = head;
    } else {
        struct node *temp = head;
        while (temp->next != head)
            temp = temp->next;
        newnode->next = head;
        temp->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 = head;
   } else {
      struct node *temp = head;
      while (temp->next != head)
         temp = temp->next;
      temp->next = newnode;
      newnode->next = head;
   }
   printf("Inserted %d at the end.\n", data);
}

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

```
    if (position == 0) {
        insertAtBeginning(data);
        return;
    }

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

    newnode->next = temp->next;
    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;
        while (last->next != head)
            last = last->next;
        head = head->next;
        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 *temp = head;

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

    while (temp->next->next != head)
        temp = temp->next;
    struct node *del = temp->next;
    temp->next = head;
    printf("Deleted %d from end.\n", del->data);
    free(del);
}

// Delete from 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 - 1; i++) {
        if (temp->next == head) {
            printf("Position out of bounds.\n");
            return;
        }
        temp = temp->next;
    }

    struct node *del = temp->next;
    if (del == head) {
        printf("Position out of bounds.\n");
        return;
    }
    temp->next = del->next;
    printf("Deleted %d from position %d.\n", del->data, position);
    free(del);
}

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

    struct node *temp = head;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to head)\n");
}

int main() {
    int mainChoice, subChoice, data, pos;

    while (1) {
        printf("\n--- Main Menu ---\n");
        printf("1. Create List\n");
        printf("2. Insert\n");
        printf("3. Delete\n");
        printf("4. Display\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &mainChoice);

switch (mainChoice) {
  case 1:
    create();
    break;
  case 2:
    printf("\n-- Insert Menu --\n");
    printf("1. Insert at Beginning\n");
    printf("2. Insert at End\n");
    printf("3. Insert at Position\n");
    printf("Enter your choice: ");
    scanf("%d", &subChoice);
    printf("Enter data to insert: ");
    scanf("%d", &data);
    switch (subChoice) {
      case 1: insertAtBeginning(data); break;
      case 2: insertAtEnd(data); break;
      case 3:
        printf("Enter position: ");
        scanf("%d", &pos);
        insertAtPosition(data, pos);
        break;
      default: printf("Invalid insert choice.\n");
    }
    break;
  case 3:
    printf("\n-- Delete Menu --\n");
    printf("1. Delete from Beginning\n");
```

```
            printf("2. Delete from End\n");
            printf("3. Delete from Position\n");
            printf("Enter your choice: ");
            scanf("%d", &subChoice);
            switch (subChoice) {
              case 1: deleteFromBeginning(); break;
              case 2: deleteFromEnd(); break;
              case 3:
                printf("Enter position to delete: ");
                scanf("%d", &pos);
                deleteAtPosition(pos);
                break;
              default: printf("Invalid delete choice.\n");
            }
            break;
          case 4:
            display();
            break;
          case 5:
            printf("Exiting program.\n");
            exit(0);
          default:
            printf("Invalid choice. Please try again.\n");
        }
      }

      return 0;
    }
```

Sample Output:

```
--- Main Menu ---
1. Create List
2. Insert
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter a number (-1 to stop): 4
Enter a number (-1 to stop): 4
Enter a number (-1 to stop): 2
Enter a number (-1 to stop): 6
Enter a number (-1 to stop): 7
Enter a number (-1 to stop): -1

--- Main Menu ---
1. Create List
2. Insert
3. Delete
4. Display
5. Exit
Enter your choice: 4
Circular Linked List: 4 -> 4 -> 2 -> 6 -> 7 -> (back to head)
```