Assignment 6.1

1.Implement Circular Linked List and related operations like insertion, deletion, display, reverse and sort in C.

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

   int data;

   struct Node* next;

};

struct Node* insertAtBegin(struct Node* head, int value) {

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

   newNode->data = value;

   if (head == NULL) {

      newNode->next = newNode;

      printf("Inserted %d at beginning.\n", value);

      return newNode;

   }

   struct Node* curr = head;

   while (curr->next != head)

      curr = curr->next;

   newNode->next = head;

   curr->next = newNode;

   printf("Inserted %d at beginning.\n", value);

   return newNode;

}

struct Node* insertAtEnd(struct Node* head, int value) {

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

   newNode->data = value;

   if (head == NULL) {

      newNode->next = newNode;

      printf("Inserted %d at end.\n", value);

      return newNode;

   }

   struct Node* curr = head;

   while (curr->next != head)

      curr = curr->next;

   curr->next = newNode;
```

```c
    newNode->next = head;
    printf("Inserted %d at end.\n", value);
    return head;
}
struct Node* insertAtPosition(struct Node* head, int value, int pos) {
    if (pos <= 1 || head == NULL) {
        return insertAtBegin(head, value);
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    struct Node* curr = head;
    int count = 1;
    while (count < pos - 1 && curr->next != head) {
        curr = curr->next;
        count++;
    }
    newNode->next = curr->next;
    curr->next = newNode;
    printf("Inserted %d at position %d.\n", value, pos);
    return head;
}
struct Node* deleteAtBegin(struct Node* head) {
    if (head == NULL) return NULL;
    int deletedValue = head->data;
    if (head->next == head) {
        free(head);
        printf("Deleted %d from beginning.\n", deletedValue);
        return NULL;
    }
    struct Node* curr = head;
    while (curr->next != head)
        curr = curr->next;
    struct Node* temp = head;
    head = head->next;
    curr->next = head;
    free(temp);
    printf("Deleted %d from beginning.\n", deletedValue);
    return head;
```

```
}

struct Node* deleteAtEnd(struct Node* head) {

    if (head == NULL) return NULL;

    struct Node* curr = head;

    struct Node* prev = NULL;


    if (head->next == head) {

        int deletedValue = head->data;

        free(head);

        printf("Deleted %d from end.\n", deletedValue);

        return NULL;

    }

    while (curr->next != head) {

        prev = curr;

        curr = curr->next;

    }

    int deletedValue = curr->data;

    prev->next = head;

    free(curr);

    printf("Deleted %d from end.\n", deletedValue);

    return head;

}

struct Node* deleteAtPosition(struct Node* head, int pos) {

    if (head == NULL) return NULL;

    if (pos == 1) {

        return deleteAtBegin(head);

    }

    struct Node* curr = head;

    struct Node* prev = NULL;

    int count = 1;

    while (count < pos && curr->next != head) {

        prev = curr;

        curr = curr->next;

        count++;

    }

    if (count != pos) {

        printf("Position %d does not exist.\n", pos);

        return head;

    }
    int deletedValue = curr->data;
    prev->next = curr->next;
    free(curr);
    printf("Deleted %d from position %d.\n", deletedValue, pos);
    return head;
}
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* curr = head;
    do {
        printf("%d -> ", curr->data);
        curr = curr->next;
    } while (curr != head);
    printf("(back to head)\n");
}
struct Node* reverse(struct Node* head) {
    if (head == NULL || head->next == head) return head;
    struct Node* prev = NULL;
    struct Node* curr = head;
    struct Node* next = NULL;
    struct Node* first = head;
    do {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    } while (curr != head);
    first->next = prev;
    return prev;
}
struct Node* sort(struct Node* head) {
    if (head == NULL || head->next == head) return head;
    int swapped;
    struct Node* ptr1;
    struct Node* lptr = NULL;
    do {
        swapped = 0;
        ptr1 = head;
        while (ptr1->next != lptr && ptr1->next != head) {
            if (ptr1->data > ptr1->next->data) {
                int temp = ptr1->data;
                ptr1->data = ptr1->next->data;
                ptr1->next->data = temp;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
    return head;
}
void freeList(struct Node* head) {
    if (head == NULL) return;
    struct Node* curr = head->next;
    struct Node* next;
    while (curr != head) {
        next = curr->next;
        free(curr);
        curr = next;
    }
    free(head);
}
int main() {
    struct Node* head = NULL;
    int choice, value, pos;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Beginning\n");
        printf("5. Delete at End\n");
        printf("6. Delete at Position\n");
        printf("7. Display\n");
printf("8. Reverse\n");
printf("9. Sort\n");
printf("10. Exit\n");

printf("Enter your choice: ");
if (scanf("%d", &choice) != 1) break;

switch (choice) {
  case 1:
    printf("Enter value: ");
    if (scanf("%d", &value) != 1) break;
    head = insertAtBegin(head, value);
    break;
  case 2:
    printf("Enter value: ");
    if (scanf("%d", &value) != 1) break;
    head = insertAtEnd(head, value);
    break;
  case 3:
    printf("Enter value: ");
    if (scanf("%d", &value) != 1) break;
    printf("Enter position: ");
    if (scanf("%d", &pos) != 1) break;
    head = insertAtPosition(head, value, pos);
    break;
  case 4:
    head = deleteAtBegin(head);
    break;
  case 5:
    head = deleteAtEnd(head);
    break;
  case 6:
    printf("Enter position: ");
    if (scanf("%d", &pos) != 1) break;
    head = deleteAtPosition(head, pos);
    break;
  case 7:
    display(head);
    break;
```

```
            case 8:

                head = reverse(head);

                printf("List reversed.\n");

                break;

            case 9:

                head = sort(head);

                printf("List sorted.\n");

                break;

            case 10:

                freeList(head);

                printf("Exiting.\n");

                return 0;

            default:

                printf("Invalid choice.\n");

        }

        while(getchar() != '\n');

    }


    freeList(head);

    return 0;

}
```