ASSIGNMENT-8

1. Write a menu driven program to perform Multiple Operations on a Circular Linked List

SOLUTION:

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

struct Node {
   int data;
   struct Node *next;
};

struct Node *head = NULL;

struct Node *createNode(int data) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

struct Node* createList(struct Node* head) {
   int n, i;
   printf("Enter the number of nodes: ");
   scanf("%d", &n);
   struct Node* tail = NULL;
   for (i = 0; i < n; i++) {
      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &newNode->data);
      newNode->next = NULL;
      if (head == NULL) {
         head = newNode;
         tail = newNode;
      } else {
         tail->next = newNode;
         tail = newNode;
      }
   }
   if (tail != NULL) {
      tail->next = head;
   }
   return head;
}

void displayList() {
    struct Node *current = head;
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);
    printf("Head (%d)\n", current->data);
}

void addToBeginning(int data) {
    struct Node *newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        struct Node *current = head;
        while (current->next != head) {
            current = current->next;
        }
        newNode->next = head;
        head = newNode;
        current->next = newNode;
    }
    printf("Node added at the beginning.\n");
}

void addToEnd(int data) {
    struct Node *newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        struct Node *current = head;
        while (current->next != head) {
            current = current->next;
        }
        current->next = newNode;
        newNode->next = head;
    }
    printf("Node added at the end.\n");
}
```

```
void addToPosition(int data, int position) {
    struct Node *newNode = createNode(data);
    if (position < 1) {
        printf("Invalid position.\n");
        return;
    }
    if (position == 1) {
        addToBeginning(data);
    } else {
        struct Node *current = head;
        int count = 1;
        while (count < position - 1 && current->next != head) {
            current = current->next;
            count++;
        }
        if (count != position - 1) {
            printf("Invalid position.\n");
        } else {
            newNode->next = current->next;
            current->next = newNode;
            printf("Node added at position %d.\n", position);
        }
    }
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else {
        struct Node *temp = head;
        struct Node *current = head;
        while (current->next != head) {
            current = current->next;
        }
        head = head->next;
        current->next = head;
        free(temp);
        printf("Node deleted from the beginning.\n");
    }
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else {
        struct Node *current = head;
        struct Node *prev = NULL;
        while (current->next != head) {
            prev = current;
            current = current->next;
        }
        prev->next = head;
        free(current);
        printf("Node deleted from the end.\n");
    }
}

void deleteFromPosition(int position) {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else if (position < 1) {
        printf("Invalid position.\n");
    } else if (position == 1) {
        deleteFromBeginning();
    } else {
        struct Node *current = head;
        struct Node *prev = NULL;
        int count = 1;
        while (count < position && current->next != head) {
            prev = current;
            current = current->next;
            count++;
        }
        if (count != position) {
            printf("Invalid position.\n");
        } else {
            prev->next = current->next;
            free(current);
            printf("Node deleted from position %d.\n", position);
        }
    }
}

int nodeCount() {
    int count = 0;
    struct Node *current = head;
    if (head == NULL) {
        return count;
    }
    do {
        count++;
        current = current->next;
    } while (current != head);
    return count;
}
```

```
void sortList() {
    struct Node *current = head, *index = NULL;
    int temp;
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    do {
        index = current->next;
        while (index != head) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
    } while (current != head);
    printf("List sorted in ascending order.\n");
}

void reverseList() {
    struct Node *prev = NULL, *current = head, *next = NULL;
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    do {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    } while (current != head);
    head->next = prev;
    head = prev;
    printf("List reversed.\n");
}

int main() {
    int choice, data, position, num;
    do {
        printf("\nMAIN MENU:\n");
        printf("1: Create a Circular Linked List\n");
        printf("2: Display the list\n");
        printf("3: Add a node at the beginning\n");
        printf("4: Add a node at the end\n");
        printf("5: Add a node at a Specified Position\n");
        printf("6: Delete a node from the beginning\n");
        printf("7: Delete a node from the end\n");
        printf("8: Delete a node from a Specified Position\n");
        printf("9: Node Count\n");
        printf("10: Sorting the List\n");
        printf("11: Reverse the List\n");
        printf("12: EXIT\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                head = createList(head);
                break;
            case 2:
                displayList();
                break;
            case 3:
                printf("Enter data for the new node: ");
                scanf("%d", &data);
                addToBeginning(data);
                break;
            case 4:
                printf("Enter data for the new node: ");
                scanf("%d", &data);
                addToEnd(data);
                break;
            case 5:
                printf("Enter data for the new node: ");
                scanf("%d", &data);
                printf("Enter the position: ");
                scanf("%d", &position);
                addToPosition(data, position);
                break;
            case 6:
                deleteFromBeginning();
                break;
            case 7:
                deleteFromEnd();
                break;
            case 8:
                printf("Enter the position: ");
                scanf("%d", &position);
                deleteFromPosition(position);
                break;
            case 9:
                printf("Number of nodes in the list: %d\n", nodeCount());
                break;
```

```
            case 10:
                sortList();
                break;
            case 11:
                reverseList();
                break;
            case 12:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (1);
    return 0;
}
```

OUTPUT:

```
MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: EXIT
Enter your choice: 1
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
```

```
11: Reverse the List
12: EXIT
Enter your choice: 2
Circular Linked List: 10 -> 20 -> 30 -> Head (10)

MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: EXIT
Enter your choice: 5
Enter data for the new node: 40
Enter the position: 3
Node added at position 3.

MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: EXIT
Enter your choice: 2
Circular Linked List: 10 -> 20 -> 40 -> 30 -> Head (10)

MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
```

```
9: Node Count
10: Sorting the List
11: Reverse the List
12: EXIT
Enter your choice: 8
Enter the position: 3
Node deleted from position 3.

MAIN MENU:
1: Create a Circular Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: EXIT
Enter your choice: 2
Circular Linked List: 10 -> 20 -> 30 -> Head (10)
```

2. Write a menu driven program to perform Multiple Operations on a Doubly Linked List

SOLUTION:

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

struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

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

struct Node* create_linked_list(struct Node* head) {
    int n, i;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    struct Node* tail = NULL;
    for (i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &newNode->data);
        newNode->next = NULL;
        newNode->prev = NULL;
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
    return head;
}

struct Node* addToBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    newNode->next = head;
    head->prev = newNode;
    return newNode;
}

struct Node* addToEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
    return head;
}

struct Node* addToPosition(struct Node* head, int data, int position) {
    if (position < 1) {
        printf("Invalid position\n");
        return head;
    }
    if (position == 1) {
        return addToBeginning(head, data);
    }
    struct Node* newNode = createNode(data);
    struct Node* current = head;
    int currentPosition = 1;
    while (currentPosition < position - 1 && current != NULL) {
        current = current->next;
        currentPosition++;
    }
    if (current == NULL) {
        printf("Invalid position\n");
        return head;
    }
    newNode->next = current->next;
    newNode->prev = current;
    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
    return head;
}

struct Node* deleteFromBeginning(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return NULL;
    }
    struct Node* newHead = head->next;
    free(head);
    if (newHead != NULL) {
        newHead->prev = NULL;
    }
    return newHead;
}
```

```
struct Node* deleteFromEnd(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return NULL;
    }
    if (head->next == NULL) {
        free(head);
        return NULL;
    }
    struct Node* current = head;
    while (current->next->next != NULL) {
        current = current->next;
    }
    free(current->next);
    current->next = NULL;
    return head;
}

struct Node* deleteFromPosition(struct Node* head, int position) {
    if (head == NULL || position < 1) {
        printf("Invalid position\n");
        return head;
    }
    if (position == 1) {
        struct Node* newHead = head->next;
        free(head);
        if (newHead != NULL) {
            newHead->prev = NULL;
        }
        return newHead;
    }
    struct Node* current = head;
    int currentPosition = 1;
    while (currentPosition < position && current != NULL) {
        current = current->next;
        currentPosition++;
    }
    if (current == NULL) {
        printf("Invalid position\n");
        return head;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    current->prev->next = current->next;
    free(current);
    return head;
}
```

```
int countNodes(struct Node* head) {
    int count = 0;
    struct Node* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

void swapData(struct Node* node1, struct Node* node2) {
    int temp = node1->data;
    node1->data = node2->data;
    node2->data = temp;
}

struct Node* sortList(struct Node* head) {
    int swapped;
    struct Node* current;
    struct Node* last = NULL;
    if (head == NULL)
        return NULL;

    do {
        swapped = 0;
        current = head;
        while (current->next != last) {
            if (current->data > current->next->data) {
                swapData(current, current->next);
                swapped = 1;
            }
            current = current->next;
        }
        last = current;
    } while (swapped);
    return head;
}

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

int* search(struct Node* head, int data, int* count) {
    int index = 0;
    int capacity = 10;
    int* indexes = (int*)malloc(capacity * sizeof(int));
    struct Node* current = head;
    *count = 0;
    while (current != NULL) {
        if (current->data == data) {
            indexes[*count] = index;
            (*count)++;
            if (*count >= capacity) {
                capacity *= 2;
                indexes = (int*)realloc(indexes, capacity * sizeof(int));
            }
        }
        current = current->next;
        index++;
    }
    indexes = (int*)realloc(indexes, (*count) * sizeof(int));
    return indexes;
}

int main() {
    struct Node* head = NULL;
    int choice, data, position, count, i;
    while (1) {
        printf("\nMAIN MENU:\n");
        printf("1: Create a Doubly Linked List\n");
        printf("2: Display the list\n");
        printf("3: Add a node at the beginning\n");
        printf("4: Add a node at the end\n");
        printf("5: Add a node at a Specified Position\n");
        printf("6: Delete a node from the beginning\n");
        printf("7: Delete a node from the end\n");
        printf("8: Delete a node from a Specified Position\n");
        printf("9: Node Count\n");
        printf("10: Sorting the List\n");
        printf("11: Reverse the List\n");
        printf("12: Search for an Element in the List\n");
        printf("13: EXIT\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
  case 1:
      head = create_linked_list(head);
      break;
   case 2:
      displayList(head);
      break;
   case 3:
      printf("Enter data for the new node: ");
      scanf("%d", &data);
      head = addToBeginning(head, data);
      break;
   case 4:
      printf("Enter data for the new node: ");
      scanf("%d", &data);
      head = addToEnd(head, data);
      break;
   case 5:
      printf("Enter data for the new node: ");
      scanf("%d", &data);
      printf("Enter the position to add the node: ");
      scanf("%d", &position);
      head = addToPosition(head, data, position);
      break;
   case 6:
      head = deleteFromBeginning(head);
      break;
   case 7:
      head = deleteFromEnd(head);
      break;
   case 8:
      printf("Enter the position to delete the node: ");
      scanf("%d", &position);
      head = deleteFromPosition(head, position);
      break;
   case 9:
      printf("Node count: %d\n", countNodes(head));
      break;
   case 10:
      head = sortList(head);
      break;
   case 11:
      head = reverseList(head);
      break;
   case 12:
      printf("Enter the data to be searched: ");
      scanf("%d", &data);
      int* indexes = search(head, data, &count);
```

```
                if(head==NULL)
                printf("The List is Empty!\n");
                else if (count > 0) {
                    printf("Element found at indexes: ");
                    for (i = 0; i < count; i++) {
                        printf("%d", indexes[i]+1);
                        if (i < count - 1) {
                            printf(", ");
                        }
                    }
                    printf("\n");
                }
                                                else
                printf("Element not found\n");
                free(indexes);
                break;
            case 13:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
MAIN MENU:
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
Enter your choice: 1
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

MAIN MENU:
```

```
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
Enter your choice: 2
NULL <- 10 -> <- 20 -> <- 30 -> NULL

MAIN MENU:
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
Enter your choice: 5
Enter data for the new node: 40
Enter the position to add the node: 3

MAIN MENU:
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
```

```
Enter your choice: 2
NULL <- 10 -> <- 20 -> <- 40 -> <- 30 -> NULL

MAIN MENU:
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
Enter your choice: 8
Enter the position to delete the node: 4

MAIN MENU:
1: Create a Doubly Linked List
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node at a Specified Position
6: Delete a node from the beginning
7: Delete a node from the end
8: Delete a node from a Specified Position
9: Node Count
10: Sorting the List
11: Reverse the List
12: Search for an Element in the List
13: EXIT
Enter your choice: 2
NULL <- 10 -> <- 20 -> <- 40 -> NULL
```