Write a C program to implement the Singly linked list using switch case.

Code:

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

struct node
{
  int info;
  struct node *next;
} *start, *q, *r;

void insertatend(int value)
{
  struct node *temp;
  temp = malloc(sizeof(struct node));
  temp->info = value;
  temp->next = NULL;

  if (start == NULL)
  {
    printf("List Empty\n");
    start = temp;
  }
  else
  {
    q = start;
    while (q->next != NULL)
    {
      q = q->next;
    }
    q->next = temp;
  }
}
void displaylist()
{
  q = start;
  if (start == NULL)
  {
    printf("Empty List");
  }
  else
  {
    while (q != NULL)
    {
```

```
      printf("%d ", q->info);
      q = q->next;
    }
    printf("\n");
  }
}
void insertatstart(int value)
{
  struct node *temp;
  temp = malloc(sizeof(struct node));
  temp->info = value;
  temp->next = NULL;
  if (start == NULL)
  {
    printf("List Empty\n");
    start = temp;
  }
  else
  {
    temp->next = start;
    start = temp;
  }
}
void deleteatstart()
{
  if (start == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    q = start;
    start = start->next;
  }
}
void deletatend()
{
  if (start == NULL)
  {
    printf("The list is empty");
  }
  else
  {
    q = start;
    while (q->next != NULL)
    {
```

```
      r = q;
      q = q->next;
    }
    r->next = NULL;
    free(q);
  }
}
void insertbeforenode(int search, int value)
{
  struct node *temp;
  temp = malloc(sizeof(struct node));
  temp->info = value;
  temp->next = NULL;
  if (start == NULL)
  {
    printf("The list is empty");
    start = temp;
  }
  else
  {
    q = start;
    while (q->info != search)
    {
      r = q;
      q = q->next;
    }
    temp->next = q;
    r->next = temp;
  }
}
void insertafternode(int search, int value)
{
  struct node *temp;
  temp = malloc(sizeof(struct node));
  temp->info = value;
  temp->next = NULL;
  if (start == NULL)
  {
    printf("The list is empty");
    start = temp;
  }
  else
  {
    q = start;
    while (q->info != search)
    {
```

```
      q = q->next;
    }
  }
}

int main()
{
  start = NULL;
  int on = 1;
  int choice;
  int data;
  int a;
  while (on == 1)
  {
    printf("Choose a Operation:\n1.Insert at End\n2.Display all
nodes\n3.Insert at the Start\n4.Delete at start.\n5.Delete at
end\n6.Insert after node\n7.Insert before node\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the value: ");
      scanf("%d", &data);
      insertatend(data);
      break;
    case 2:
      displaylist();
      break;
    case 3:
      printf("Enter the value: ");
      scanf("%d", &data);
      insertatstart(data);
      break;
    case 4:
      deleteatstart();
      break;
    case 5:
      deletatend();
      break;
    case 6:
      printf("Enter the search value and node value: ");
      scanf("%d %d", &a, &data);
      insertafternode(a, data);
      break;
    case 7:
      printf("Enter the search value and node value: ");
```

```
      scanf("%d %d", &a, &data);
      insertbeforenode(a, data);
      break;
    default:
      printf("Error 404: Operation not found\n");
      break;
    }
    printf("Enter 1 to continue: ");
    scanf("%d", &on);
  }
}
```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT\DSA> cd "c:\Users\Ajay kumar\Deskto
dList }
Choose a Operation:
1.Insert at End
2.Display all nodes
3.Insert at the Start
4.Delete at start.
5.Delete at end
6.Insert after node
7.Insert before node
1
Enter the value: 3
List Empty
Enter 1 to continue: 1
Choose a Operation:
1.Insert at End
2.Display all nodes
3.Insert at the Start
4.Delete at start.
5.Delete at end
6.Insert after node
7.Insert before node
2
3
Enter 1 to continue: 0
PS C:\Users\Ajay kumar\Desktop\SEIT\DSA\Lab\2>
```

Post Experiment Exercise

1. Write a C program to implement the Singly linked list using switch case.

Code:

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

// Structure for a singly linked list node
struct Node
{
  int data;
  struct Node *next;
};

// Structure for the stack
struct Stack
{
  struct Node *top;
};

// Structure for the queue
struct Queue
{
  struct Node *front;
  struct Node *rear;
};

// Function to create a new node
struct Node *createNode(int data)
{
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  if (newNode == NULL)
  {
    printf("Memory allocation failed!\n");
    exit(1);
  }
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}

// Stack functions
void push(struct Stack *stack, int data)
{
  struct Node *newNode = createNode(data);
```

```
  newNode->next = stack->top;
  stack->top = newNode;
}

int pop(struct Stack *stack)
{
  if (stack->top == NULL)
  {
    printf("Stack is empty!\n");
    exit(1);
  }
  int data = stack->top->data;
  struct Node *temp = stack->top;
  stack->top = stack->top->next;
  free(temp);
  return data;
}

// Queue functions
void enqueue(struct Queue *queue, int data)
{
  struct Node *newNode = createNode(data);
  if (queue->rear == NULL)
  {
    queue->front = newNode;
    queue->rear = newNode;
  }
  else
  {
    queue->rear->next = newNode;
    queue->rear = newNode;
  }
}

int dequeue(struct Queue *queue)
{
  if (queue->front == NULL)
  {
    printf("Queue is empty!\n");
    exit(1);
  }
  int data = queue->front->data;
  struct Node *temp = queue->front;
  queue->front = queue->front->next;
  if (queue->front == NULL)
  {
```

```
    queue->rear = NULL;
  }
  free(temp);
  return data;
}

int main()
{
  // Create a stack
  struct Stack stack;
  stack.top = NULL;

  // Push elements onto the stack
  push(&stack, 10);
  push(&stack, 20);
  push(&stack, 30);

  // Pop and print elements from the stack
  printf("Stack elements: %d, %d, %d\n", pop(&stack), pop(&stack),
pop(&stack));

  // Create a queue
  struct Queue queue;
  queue.front = NULL;
  queue.rear = NULL;

  // Enqueue elements into the queue
  enqueue(&queue, 10);
  enqueue(&queue, 20);
  enqueue(&queue, 30);

  // Dequeue and print elements from the queue
  printf("Queue elements: %d, %d, %d\n", dequeue(&queue),
dequeue(&queue), dequeue(&queue));

  return 0;
}
```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT\DSA> cd "c:\Users\Ajay kumar\Desktop\S
o stack_using_linkedlist } ; if ($?) { .\stack_using_linkedlist }
Stack elements: 10, 20, 30
Queue elements: 30, 20, 10
PS C:\Users\Ajay kumar\Desktop\SEIT\DSA\Lab\2>
```

2. Explain Circular linked list and Doubly linked list with all its operations Write Pseudocodes.

Circular Linked List:

A circular linked list is a type of linked list in which the last node's next pointer points back to the first node, creating a circular structure. This means that there is no NULL pointer at the end of the list. Circular linked lists are used in situations where elements need to be traversed in a circular manner.

Operations on Circular Linked List:

1. Insertion at the Beginning:

```
newNode = CreateNode(data)
if head == NULL:
    head = newNode
    newNode.next = head
else:
    newNode.next = head
    temp = head
    while temp.next != head:
        temp = temp.next
    temp.next = newNode
    head = newNode
```

2. Insertion at the End:

```
Procedure InsertAtEnd(data):
    newNode = CreateNode(data)
    if head == NULL:
        head = newNode
        newNode.next = head
    else:
        temp = head
        while temp.next != head:
            temp = temp.next
        temp.next = newNode
        newNode.next = head
```

3. Deletion from the Beginning:

```
Procedure DeleteFromBeginning():
    if head == NULL:
        Print "List is empty"
    else:
        temp = head
        while temp.next != head:
            temp = temp.next
        temp.next = head.next
        free(head)
        head = temp.next
```

4. Deletion from the End:

```
Procedure DeleteFromEnd():
    if head == NULL:
        Print "List is empty"
    else:
        temp = head
        prev = NULL
        while temp.next != head:
            prev = temp
            temp = temp.next
        prev.next = head
        free(temp)
```

Doubly Linked List:

A doubly linked list is a type of linked list in which each node contains pointers to both its previous and next nodes. This allows for easy traversal in both directions, forward and backward.

Operations on Doubly Linked List:

1. Insertion at the Beginning:

```
Procedure InsertAtBeginning(data):
    newNode = CreateNode(data)
    newNode.next = head
    newNode.prev = NULL
    if head != NULL:
        head.prev = newNode
    head = newNode
```

2. Insertion at the End:

```
Procedure InsertAtEnd(data):
    newNode = CreateNode(data)
    temp = head
    while temp.next != NULL:
        temp = temp.next
    temp.next = newNode
    newNode.prev = temp
    newNode.next = NULL
```

3. Deletion from the Beginning:

```
Procedure DeleteFromBeginning():
    if head == NULL:
        Print "List is empty"
    else:
        temp = head
        head = head.next
        if head != NULL:
```

```
            head.prev = NULL
        free(temp)
```

4. Deletion from the End:

```
Procedure DeleteFromEnd():
    if head == NULL:
        Print "List is empty"
    else:
        temp = head
        while temp.next != NULL:
            temp = temp.next
        if temp.prev != NULL:
            temp.prev.next = NULL
        else:
            head = NULL
        free(temp)
```