Output:

List: 2 → 3 → 4 → (back to head)

Date.....

Expt. No. ..... Page No. .....

Lab Sheet 6

1. Writ a C program to implement a Circular Singly linked list using First and last pointers.
Implement the following operations:
(i) Insertion at the end of the list using First & last pointers.
(ii) Deletion from the beginning or end using First and last pointers.
(iii) Display the list after each operation.

Source Code:

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

struct Node {
    int data;
    struct Node *next;
} *head = NULL, *tail = NULL;

void append(int data) {
    struct Node *newNode = (struct Node *)
malloc(sizeof(struct Node));
    if(!newNode) {
```

```
        perror("Failed to dynamically allocate
memory\n");
        return;
    }
    newNode->data = data;
    newNode->next = newNode;
    if(!head) {
        head = tail = newNode;
        return;
    }
    tail->next = newNode;
    newNode->next = head;
    tail = newNode;
}

void delete_begin() {
    if(!head) return;
    if(head == tail) {
        free(head);
        head = tail = NULL;
        return;
    }
    struct Node *temp = head;
    tail->next = head->next;
    head = head->next;
    free(temp);
}
```

```
void delete_end() {
  if(!head) return;
  if(head == tail) {
     free(head);
     head = tail = NULL;
     return;
  }
  struct Node *temp = head;
  for(; temp->next != tail; temp = temp->next);
  temp->next = head;
  free(tail);
  tail = temp;
}

void display() {
  if(!head) {
     printf("List is empty\n");
     return;
  }
  struct Node *temp = head;
  printf("List: ");
  do {
     printf("%d -> ", temp->data);
     temp = temp->next;
  } while(temp != head);
  printf("(back to head)\n");
}
```

Date.....

Expt. No. .....

Page No. .....

```
int main() {
    append(1);
    append(2);
    append(3);
    append(4);
    append(5);
    delete_begin();
    delete_end();
    display(); while(head) delete_end();
    return 0;
}
```

Output:

```
Enter the number of terms in polynomial A: 2
Enter the coefficient of term 1: 1
Enter the exponent of term 1: 2
Enter the coefficient of term 2: 2
Enter the exponent of term 2: 1
Enter the number of terms in polynomid B: 2
Enter the coefficient of term 1: 2
Enter the exponent of term 1: 1
Enter the coefficient of term 2: 4
Enter the exponent of term 2: 0
1.00x^2.00 + 2.00x^1.00
+
2.00x ^1.00 + 4.00x ^0.00
=
1.00x ^2.00 + 4.00x^1.00 + 4.00x^0.00
```

Date.....

Expt. No. ..... Page No. .....

2. Add two polynomials represented as doubly linked lists.
(i) Represent each polynomial using a doubly linked list.
(ii) Write a function to add two polynomials by merging terms with equal exponents. The resulting polynomial should be stored in a new doubly linked list, maintaining the order of terms in descending powers of exponents.
(iii) Display all three polynomials: the two input polynomials & their sum. Ensure dynamic memory allocation is used for all node operations and that both prev & next pointers are maintained correctly.

Source Code:

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

struct Node {
    float coeff;
    float expo;
    struct Node *next, *prev;
};
```

Date.....

Expt. No. ..... Page No. .....

```
void sort(struct Node *head) {
    int swapped;
    struct Node *temp;
    float swap_expo, swap_coeff;
    do {
        swapped = 0;
        for(temp = head; temp->next; temp = temp->next) {
            if(temp->expo < temp->next->expo){
                swap_expo = temp->expo;
                temp->expo = temp->next->expo;
                temp->next->expo = swap_expo;

                swap_coeff = temp->coeff;
                temp->coeff = temp->next->weff;
                temp->next->weff = swap_coeff;
                swapped = 1;
            }
        }
    }while(swapped);
}

void append(float coeff, float expo, struct
Node **head, struct Node **tail) {
    struct Node *newNode = (struct Node *)
malloc(sizeof(struct Node));
    if(!newNode) {
```

```
        perror("Failed to dynamically allocate
memory!\n");
        return;
    }
    newNode->coeff = coeff;
    newNode->expo = expo;
    newNode->next = newNode->prev = NULL;
    if(!(*head)) {
        *head = *tail = newNode;
        return;
    }
    (*tail)->next = newNode;
    newNode->prev = *tail;
    *tail = newNode;
}

void take_input(struct Node **head, struct
Node **tail, char c) {
    int i, n;
    float coeff, expo;
    printf("Enter the number of terms in
the polynomial %c: ", c);
    scanf("%d", &n);
    for(i=0; i<n; i++) {
        printf("Enter the coefficient of term
%d: ", i+1);
        scanf("%f", &coeff);
```

```
        printf("Enter the exponent of term %d: ", i+1);
        scanf("%f", &expo);
        append(coeff, expo, head, tail);
    }
    sort(*head);
}

void add(struct Node **head3, struct Node **tail3, struct Node *head1, struct Node *head2) {
    while (head1 && head2) {
        if(head1->expo == head2->expo) {
            append(head1->coeff + head2->coeff, head1->expo, head3, tail3);
            head1 = head1->next;
            head2 = head2->next;
        } else if(head1->expo > head2->expo){
            append(head1->coeff, head1->expo, head3, tail3);
            head1 = head1->next;
        } else if (head2->expo > head1->expo){
            append(head2->coeff, head2->expo, head3, tail3);
            head2 = head2->next;
        }
    }
```

```
    if(!head1) head1 = head2;
    for(; head1; head1 = head1->next) append(
head1->coeff, head1->expo, head3, tail(3);
}

void empty(struct Node *head){
    struct Node *temp;
    while(head) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

void display(struct Node *head) {
    struct Node *temp = head;
    for(; temp->next; temp=temp->next) {
        printf("%lfx^%.1f+", temp->coeff, temp
->expo);
    }
    printf("%.2fx^%.2f\n", temp->coeff,
temp->expo);
}

int main() {
    struct Node *head1, *tail1, *head2, *tail2,
*head3, *tail3;
```

```
    head1 = tail1 = head2 = tail2 = head3 = tail3
= NULL;
    take_input(&head1, &tail1, 'A');
    take_input(&head2, &tail2, 'B');
    display(head1);
    printf(" + \n");
    display(head2);
    printf(" = \n");
    add(&head3, &tail3, head1, head2);
    display(head3);
    empty(head1);
    empty(head2);
    empty(head3);
    return 0;
}
```