1.main.c

```
#include "header.h"

int main(){

   DLL L1;

   init_DLL(&L1);

   insert_beg(&L1, 5);

   insert_end(&L1, 10);

   insert_pos(&L1, 15, 2);

   insert_pos(&L1, 20, 3);

   insert_end(&L1, 15);

   insert_end(&L1, 10);

   insert_end(&L1, 5);

   displayLR(L1);

   displayRL(L1);

   is_palindrome(&L1);

   sort(&L1);

   printf("After Sorting: ");

   displayLR(L1);

   remove_duplicates(&L1);

   printf("After removing duplicates: ");

   displayLR(L1);

   remove_beg(&L1);

   printf("After removing element from beginning: ");

   displayLR(L1);

   remove_end(&L1);

   printf("After removing element from end: ");

   displayLR(L1);

   remove_pos(&L1, 1);

   printf("After removing element from index 1: ");

   displayLR(L1);

   return 0;
}
2.header.h
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
   int data;
   struct node *next, *prev;
}node;

typedef struct DLL{
   node *front, *rear;
}DLL;

void init_DLL(DLL *l);
int len(DLL *l);
int isEmpty(DLL *l);
void insert_beg(DLL *l, int d);
void insert_end(DLL *l, int d);
void insert_pos(DLL *l, int d, int index);
void remove_beg(DLL *l);
void remove_end(DLL *l);
void remove_pos(DLL *l, int index);
void sort(DLL *l);
void displayLR(DLL l);
void displayRL(DLL l);
void is_palindrome(DLL *l);
void remove_duplicates(DLL *l);
```

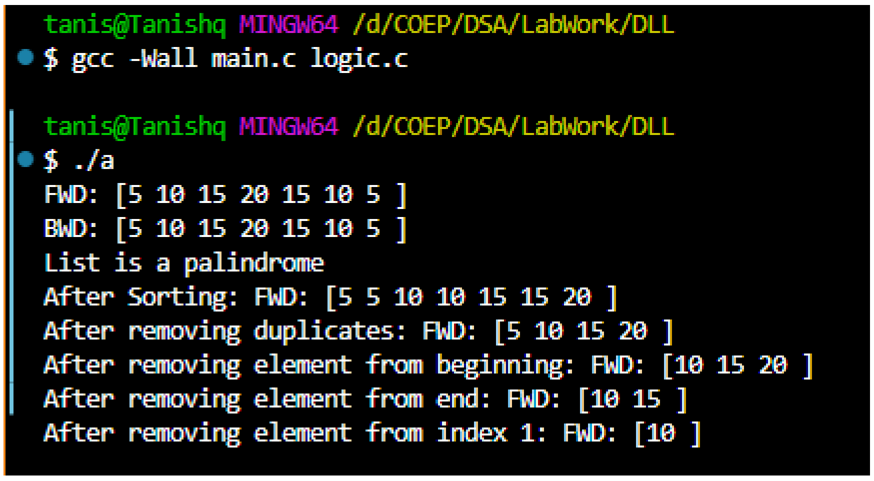
3.logic.c

```
#include <stdlib.h>
#include "header.h"

void init_DLL(DLL *l){
   l -> front = NULL;
   l -> rear = NULL;
}
int len(DLL *l){
   int count = 0;
   node *temp = l -> front;
   while(temp){
      count++;
      temp = temp -> next;
   }
   return count;
}
int isEmpty(DLL *l){
   if(l -> front == NULL){
      return 1;
   }
   return 0;
}
void insert_beg(DLL *l, int d){
   node *newnode = (node *)malloc(sizeof(node));
   newnode -> next = NULL;
   newnode -> prev = NULL;
   newnode -> data = d;
   if(!isEmpty(l)){
      node *temp = l -> front;
      temp -> prev = newnode;
```

```
        newnode -> next = temp;
        l -> front = newnode;
    }
    else{
        l -> front = newnode;
        l -> rear = newnode;
    }
}
void insert_end(DLL *l, int d){
    node *newnode = (node *)malloc(sizeof(node));
    newnode -> next = NULL;
    newnode -> prev = NULL;
    newnode -> data = d;
    if(!isEmpty(l)){
        node *temp = l -> front;
        while(temp -> next != NULL){
            temp = temp -> next;
        }
        temp -> next = newnode;
        newnode -> prev = temp;
        l -> rear = newnode;
    }
    else{
        l -> front = newnode;
        l -> rear = newnode;
    }
}
void insert_pos(DLL *l, int d, int index){
    node *newnode = (node *)malloc(sizeof(node));
    newnode -> next = NULL;
    newnode -> prev = NULL;
    newnode -> data = d;
    if(index == 0){
        insert_beg(l, d);
        return;
    }
    else{
        node *temp = l -> front;
        for(int i = 0; i < index-1 && temp != NULL; i++){
            temp = temp -> next;
        }
        if(temp == NULL || temp -> next == NULL){
            insert_end(l, d);
            return;
        }
        else{
            newnode -> prev = temp;
            newnode -> next = temp -> next;
            temp -> next = newnode;
            newnode -> next -> prev = newnode;
        }
        return;
    }
}
void remove_beg(DLL *l){
    node *temp;
    if(isEmpty(l)){
        printf("List is already empty\n");
        return;
    }
    else if(l -> front  == l -> rear){
        free(l -> front);
        l -> front = NULL;
        l -> rear = NULL;
        return;
    }
    else{
        temp = l -> front;
        l -> front = l -> front -> next;
        l -> front -> prev = NULL;
        free(temp);
    }
}
void remove_end(DLL *l){
    node *temp;
    if(isEmpty(l)){
        printf("List is already empty\n");
        return;
    }
    else if(l -> front  == l -> rear){
        free(l -> front);
        l -> front = NULL;
        l -> rear = NULL;
        return;
    }
    else{
        temp = l -> rear;
        l -> rear = l -> rear -> prev;
        l -> rear -> next = NULL;
        free(temp);
    }
}
void remove_pos(DLL *l, int index){
    node *temp = l -> front;;
    if(isEmpty(l)){
        printf("List is already empty\n");
        return;
    }
    int length = len(l);
    if(index < 0 || index >= length){
        printf("Invalid index\n");
        return;
    }
    if(index == 0){
        remove_beg(l);
        return;
    }
    for(int i = 0; i < index; i++){
        temp = temp -> next;
    }
    if(temp == l -> rear){
        remove_end(l);
    }
    else{
        temp -> prev -> next = temp -> next;
        temp -> next -> prev = temp -> prev;
        free(temp);
    }
}
void sort(DLL *l){
    if(isEmpty(l)){
        return;
    }
    int swap;
    node *temp;
    do{
        swap = 0;
        temp = l -> front;
        while(temp -> next != NULL){
            if(temp -> data > temp -> next -> data){
                int t = temp -> data;
                temp -> data = temp -> next -> data;
                temp -> next -> data = t;
                swap = 1;
            }
            temp = temp -> next;
        }
        l -> rear = temp;
    }while(swap);
}
void displayLR(DLL l){
    node *p;
    printf("FWD: [");
    p = l.front;
    if(!p){
        printf("]\n");
        return;
    }
    while (p != NULL) {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("]\n");
}
void displayRL(DLL l){
    node *p;
    printf("BWD: [");
    p = l.rear;
    if(!p){
        printf("]\n");
        return;
    }
    while (p != NULL) {
        printf("%d ", p->data);
        p = p->prev;
    }
    printf("]\n");
}
void is_palindrome(DLL *l){
    int length = len(l);
    if(isEmpty(l)){
        printf("List is already Empty\n");
        return;
    }
    else if(length == 1){
        printf("List is a palindrome\n");
        return;
    }
    node *p = l -> front;
    node *q = l -> rear;
    while(p != q && p -> next != q){
        if(p -> data != q -> data){
            printf("List is not a palindrome\n");
            return;
        }
        p = p -> next;
        q = q -> prev;
    }
    printf("List is a palindrome\n");
    return;
}
void remove_duplicates(DLL *l){
    node *p = l -> front;
    while(p != NULL){
        node *q = p -> next;
        while(q != NULL){
            if(p -> data == q -> data){
                node *temp = q;
                q = q -> next;
                if(temp == l -> rear){
                    remove_end(l);
                }
                else{
                    temp -> prev -> next = temp -> next;
                    if(temp -> next != NULL){
                        temp -> next -> prev = temp -> prev;
                    }
                    free(temp);
                }
            }
            else{
                q = q -> next;
            }
        }
        p = p -> next;
    }
}
```

OUTPUT:



```
tanis@Tanishq MINGW64 /d/COEP/DSA/LabWork/DLL
$ gcc -Wall main.c logic.c

tanis@Tanishq MINGW64 /d/COEP/DSA/LabWork/DLL
$ ./a
FWD: [5 10 15 20 15 10 5 ]
BWD: [5 10 15 20 15 10 5 ]
List is a palindrome
After Sorting: FWD: [5 5 10 10 15 15 20 ]
After removing duplicates: FWD: [5 10 15 20 ]
After removing element from beginning: FWD: [10 15 20 ]
After removing element from end: FWD: [10 15 ]
After removing element from index 1: FWD: [10 ]
```