```
#include <stdio.h>

#include <stdlib.h>


//////////// Node structure ////////////

struct Node {

       int data;

       struct Node* next;

};

//////////// function prototype ////////////

void operation_select(); // operation selection

int display(struct Node* head); // show the linked list

struct Node* insert_at_head(struct Node* head, int data); //insert the given data at the
begining of the list

struct Node* insert_at_tail(struct Node* head, int data); //insert the given data at the end of
the list

struct Node* delete_at_head(struct Node* head);  //delete the node at the head of the list

struct Node* delete_at_tail(struct Node* head);  //delete the node at the end of the list

struct Node* delete_with_val(struct Node* head, int val);  // find the given val and delete it
from the list

void find_element(struct Node* head, int val); // find the given val

void  count_element(struct Node* head);                                //count the length if the list

struct Node* destroy_list(struct Node* head);    //delete the entire list

//////////// function prototype ////////////



int main(){

   struct Node* head=NULL;

   int operation, value;


   do {
```

```
operation_select();
scanf("%d",&operation);

switch (operation) {
  case 1: display(head);
          break;
  case 2: printf("\nvalue: ");
          scanf("%d", &value);
          head = insert_at_head(head, value);
          break;
  case 3: printf("\nvalue: ");
          scanf("%d", &value);
          head = insert_at_tail(head, value);
          break;
  case 4: head = delete_at_head(head);
          break;
  case 5: head = delete_at_tail(head);
          break;
  case 6: printf("\nwith which value? ");
          scanf("%d", &value);
          head = delete_with_val(head, value);
          break;
  case 7: printf("\nfind which value? ");
          scanf("%d", &value);
          find_element(head, value);
          break;
  case 8: count_element(head);
          break;
  case 9: head = destroy_list(head);
```

```
                break;
            case 10:
                break;
            default: printf("undefined input\n");
        }
    } while(operation!= 10);

    return 0;
}

void operation_select(){
        printf("\n\n************************************\n");
        printf("select your operation (e.g., 2)\n");
        printf("1- Display link list\n");
        printf("2- Insert_at_head\n");
        printf("3- Insert_at_tail\n");
        printf("4- Delete_at_head\n");
        printf("5- Delete_at_tail\n");
        printf("6- Delete_with_val\n");
        printf("7- Find_element\n");
        printf("8- Count_element\n");
        printf("9- Destroy_list\n");
        printf("10- Exit\n");
        printf("Which operation? ");
}

int display(struct Node* head){
        struct Node* curr_node=head;
        printf("\n\n");
        while (curr_node != NULL) {
                printf(" %d --->", curr_node->data);
                curr_node = curr_node->next;
        }
        printf("*NULL*\n\n");
        return 0;
}

/*********************************************
insert_at_head (struct Node* head, int val)
this function adds the val into the head of the
linked list and returns head pointer
Example:
linked list: 10 --->20 --> 30 --> NULL
insert_at_head (head, 42)
linked list: 42 ---> 10 --->20 ---> 30 ---> NULL
*********************************************/
struct Node* insert_at_head(struct Node* head, int val){
   //create a link
   struct Node *link = (struct Node *) malloc(sizeof(struct Node));

   link->data = val;

   //point it to old first node
   link->next = head;

   //point first to new first node
   head = link;
   free(link);
   return head;
}




/*******************************************
insert_at_tail (struct Node* head, int val)
this function adds the val into the tail of the
linked list and returns head pointer
Example:
linked list: 10 --->20 --> 30 --> NULL
insert_at_tail (head, 42)
linked list: 10 --->20 ---> 30 ---> 42 ---> NULL
*******************************************/
struct Node* insert_at_tail(struct Node* head, int val){
    /*if the list was empty*/
    if(head == NULL) {
            //create a link
            struct Node *link = (struct Node *) malloc(sizeof(struct Node));
            link->data = val;
            link->next = NULL;
            return link;
    }

    struct Node *current = head;

    while (current->next != NULL) {
        current = current->next;
    }
    /* now we can add a new variable */

    current->next = (struct Node *) malloc(sizeof(struct Node));
    current->next->data = val;
    current->next->next = NULL;
    return head;
}



/*******************************************
delete_at_head(struct Node* head)
this function deletes the value in the head of the
linked list and returns head pointer
Example:
linked list: 10 --->20 --> 30 --> NULL
delete_at_head(head)
linked list: 20 ---> 30 ---> NULL
Note: if the list is empty print out an appropriate message
*******************************************/
struct Node* delete_at_head(struct Node* head){
        if (head == NULL) {
                printf("the list is empty");
                return head;
        }

   //mark next to first link as first
   head = head->next;
```

```
  //return the deleted link
  return head;
}



/********************************************
delete_at_tail(struct Node* head)
this function deletes the value from the tail of the
linked list and returns head pointer
Example:
linked list: 10 --->20 --> 30 --> NULL
delete_at_tail(head)
linked list: 10 ---> 20 ---> NULL
Note: if the list is empty print out an appropriate message
********************************************/
struct Node* delete_at_tail(struct Node* head){
   if (head == NULL) {
          printf("the list is empty");
          return head;
   }
   /* if there is only one item in the list, remove it */
   if (head->next == NULL) {
      free(head);
      return NULL;
   }

   /* get to the last node in the list */
```

```
    struct Node *current = head;
    while (current->next->next != NULL) {
        current = current->next;
    }

    /* now current points to the last item of the list, so let's remove current->next */
    free(current->next);
    current->next = NULL;
    return head;
}




/*******************************************
delete_with_val(struct Node* head, int val)
this function deletes the node with the selected value
and returns head pointer
Example:
linked list: 10 --->20 --> 30 --> NULL
delete_with_val(head, 20)
linked list: 10 ---> 30 ---> NULL
Note: if the list is empty or the value is not in the
linked list print out an appropriate message
*******************************************/
struct Node* delete_with_val(struct Node* head, int val){
   //start from the first link
   struct Node* current = head;
   struct Node* previous = NULL;
```

```
//if list is empty

if(head == NULL) {

            printf("Not Found! the value is not in the linked list!");

  return NULL;

}


//navigate through list

while(current->data != val) {


  //if it is last node

  if(current->next == NULL) {

          printf("Not Found! the value is not in the linked list!");

    return NULL;

  } else {

    //store reference to current link

    previous = current;

    //move to next link

    current = current->next;

  }

}


//found a match, update the link

if(current == head) {

  //change first to point to next link

  head = head->next;

} else {

  //bypass the current link

  previous->next = current->next;
  }

  return head;
}



/*******************************************
find_element(struct Node* head, int val)
this function finds the node with the selected value

Example:
linked list: 10 --->20 --> 30 --> NULL
find_element(head, 20)
output is : Found! the value is node number 2
find_element(head, 42)
output is: Not Found! the value is not in the linked list!
*******************************************/
void find_element(struct Node* head, int val){
  //start from the first link
  struct Node* current = head;
  int counter = 1;
  //if list is empty
  if(head == NULL) {
    printf("Not Found! the value is not in the linked list!");
    return;
  }

  //navigate through list
  while(current->data != val) {
    //if it is last node
    if(current->next == NULL) {
          printf("Not Found! the value is not in the linked list!");
      return;
    } else {
      //go to next link
      current = current->next;
      counter++;
    }
  }

  //if data found, return the current Link
  printf("Found! the value is node number %i", counter);
}



/*******************************************
count_element(struct Node* head)
this function counts the number of node in the linked list

Example:
linked list: 10 --->20 --> 30 --> NULL
count_element(head)
output is : 3 elements
*******************************************/
void count_element(struct Node* head){
         //start from the first link
  struct Node* current = head;
  int counter = 1;
  //if list is empty
  if(head == NULL) {
    printf("0 element");
    return;
  }

  //navigate through list
  while(current->next != NULL) {
    //if it is last node
    current = current->next;
    counter++;
    }

  //if data found, return the current Link
  printf("%i elements", counter);

}
```

```
/********************************************
destroy_list(struct Node* head)
this function removes all the nodes from the linked list

Example:
linked list: 10 --->20 --> 30 --> NULL
destroy_list(head)
```

```
linked list: NULL
**********************************************/
struct Node* destroy_list(struct Node* head){
    if(head == NULL) {
        return head;
    }
    if(head->next != NULL){
     destroy_list(head->next);
     head->next = NULL;
     free(head);
    }
    return NULL;
}
```