linkedlist.h

```
using namespace std;
class Node {
private:
    int data;
    Node* next;
public:
    Node(){ }
    ~Node(){destroy_list(); }
    void operation_select();          //operation selection
    int display();                    //show the linked list
    Node *insert_at_head(int data);   //insert the given data at the begining of the list
    Node *insert_at_tail(int data);   //insert the given data at the end of the list
    Node *delete_at_head();           //delete the node at the head of the list
    Node *delete_at_tail();           //delete the node at the end of the list
    Node *delete_with_val(int val);   //find the given val and delete it from the list
    void find_element(int val);       //find the given val
    void  count_element();            //count the length if the list
    Node *destroy_list();             //delete the entire list
};
```

```
#include "linkedlist.h"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

int main(){
    Node* head = new Node();
    head = head->destroy_list();
    int operation, value;

    do {
        head->operation_select();
        scanf("%d",&operation);

        switch (operation) {
          case 1: head->display();
                  break;
          case 2: printf("\nvalue: ");
                  scanf("%d", &value);
                  head = head->insert_at_head(value);
                  break;
          case 3: printf("\nvalue: ");
                  scanf("%d", &value);
                  head = head->insert_at_tail(value);
                  break;
    case 4: head = head->delete_at_head();
                  break;
          case 5: head = head->delete_at_tail();
                  break;
          case 6: printf("\nwith which value? ");
                  scanf("%d", &value);
                  head = head->delete_with_val(value);
                  break;
          case 7: printf("\nfind which value? ");
                  scanf("%d", &value);
                  head->find_element(value);
                  break;
          case 8: head->count_element();
                  break;
          case 9: head = head->destroy_list();
                  break;
          case 10:
                  break;
          default: printf("undefined input\n");
        }
    } while(operation!= 10);

    return 0;
}
```

```
#include <cstddef>
#include "linkedlist.h"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

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

void Node::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 Node::display(){
	Node* curr_node = this;
	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 (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 (42)
linked list: 42 ---> 10 --->20 ---> 30 ---> NULL
*****************************************/
Node* Node::insert_at_head(int val){
  //create a link
  Node *link = (Node *) malloc(sizeof(Node));

  link->data = val;

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



  return link;
}
```

```
/*****************************************
insert_at_tail (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 (42)
linked list: 10 --->20 ---> 30 ---> 42 ---> NULL
*****************************************/
Node* Node::insert_at_tail(int val){
   /*if the list was empty*/
   if(this == NULL) {
          //create a link
     Node *link = (Node *) malloc(sizeof(Node));
          link->data = val;
          link->next = NULL;
          return link;
   }

   Node *current = this;

   while (current->next != NULL) {
      current = current->next;
   }

   /* now we can add a new variable */

   current->next = (Node *) malloc(sizeof(Node));
```

```
   current->next->data = val;
   current->next->next = NULL;
   return this;
}



/*****************************************
delete_at_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()
linked list: 20 ---> 30 ---> NULL
Note: if the list is empty print out an appropriate message
*****************************************/
Node* Node::delete_at_head(){
        if (this == NULL) {
                printf("the list is empty");
                return this;
        }

  //return next to first link as first
  return this->next;
}
```

```
/*****************************************
delete_at_tail()
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()
linked list: 10 ---> 20 ---> NULL
Note: if the list is empty print out an appropriate message
*****************************************/
Node* Node::delete_at_tail(){
   if (this == NULL) {
          printf("the list is empty");
          return this;
   }
   /* if there is only one item in the list, remove it */
   if (this->next == NULL) {
      free(this);
      return NULL;
   }

   /* get to the last node in the list */
   Node *current = this;
   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 this;
}




/*****************************************
delete_with_val(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(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
*****************************************/
Node* Node::delete_with_val(int val){
  //start from the first link
  Node *head = this;
  Node *current = head;
  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 head;
    } 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(int val)
this function finds the node with the selected value

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

Example:
linked list: 10 --->20 --> 30 --> NULL
count_element()
output is : 3 elements
*******************************************/
void Node::count_element(){
          //start from the first link
  Node *current = this;
  int counter = 1;
  //if list is empty
  if(this == 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()
this function removes all the nodes from the linked list

Example:
linked list: 10 --->20 --> 30 --> NULL
destroy_list()
linked list: NULL
*********************************************/
Node* Node::destroy_list(){
   if(this == NULL) {
      return this;
   }
```

```
    if(this->next != NULL){
      this->next->destroy_list();
     this->next = NULL;
     free(this);
    }
    return NULL;
}
```