```c
// Starting code for Q1 Sample Questions for Week-11 Lab Practical Exam
// Base code by Elizabeth Willer and modified by Braedon Wooding

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

struct node {
    struct node *next;
    int          data;
};

struct node *strings_to_list(int len, char *strings[]);
void print_list(struct node *head);
int sum_list(struct node *head);
struct node *find_item(struct node *head, int item);
int find_last(struct node *head);
int find_nth(struct node *head, int nth);
struct node *add_last(struct node *head, int item);
struct node *remove_first(struct node *head);
struct node *remove_last(struct node *head) ;
struct node *remove_item(struct node *head, int item);
void free_list(struct node *head);
void test_strings(void);
void test_sum(void);
void test_find(void);
void test_find_last(void);
void test_find_nth(void);
void test_add_last(void);
void test_remove_first(void);
void test_remove_last(void);
void test_remove_item(void);

int main(int argc, char *argv[]) {
    // UNCOMMENT THESE AS YOU COMPLETE THEM to test them
    test_strings();
    test_sum();
    test_find();
    test_find_last();
    test_find_nth();
    test_add_last();
    test_remove_first();
    test_remove_last();
    test_remove_item();
    return 0;
}

// create linked list from array of strings
struct node *strings_to_list(int len, char *strings[]) {
    struct node *head = NULL;
    int i = len - 1;
    while (i >= 0) {
        struct node *n = malloc(sizeof (struct node));
        assert(n != NULL);
        n->next = head;
        n->data = atoi(strings[i]);
        head = n;
        i = i-1;
    }
    return head;
}

// print linked list in python style (i.e. [1, 2, 3, 4])
void print_list(struct node *head) {
    printf("[");
    struct node *cur = head;
    while (cur != NULL && cur->next != NULL) {
        printf("%d, ", cur->data);
        cur = cur->next;
    }
    printf("%d ]\n",cur->data);
}
```

```c
// free the memory used by the linked list
void free_list(struct node *head) {
    struct node *cur = head;
    if (cur != NULL){
        free_list(cur->next);
        free(cur);
    }

}

//calculate sum of a list
// return 0 for empty list.
int sum_list(struct node *head) {
    int sum = 0;
    while(head != NULL) {
        sum = sum + head->data;
        head = head->next;
    }
    return sum;
}

//find 'item' in a list (return pointer to that item)
//return NULL if can't find item.
struct node *find_item(struct node *head, int item) {
    struct node *cur = head;
    while (cur != NULL) {
        if (cur->data == item){
            return cur;
        }
        cur = cur->next;
    }

    return NULL;
}

//find last item in a list (return the value of that item)
//return -1 if empty list
int find_last(struct node *head) {
    struct node *cur = head;
    if (head == NULL) {
        return -1;
    }else{
        while (cur->next != NULL) {
            cur = cur->next;
        }
    }
    return cur->data;
}

// find nth item in a list (return the value of that item)
// i.e. find_nth(head, 1) should return the head of the list
// and find_nth(head, 2) should return the second item of the list (head->next)
// return -1 if out of bounds.
int find_nth(struct node *head, int nth) {
    struct node *cur = head;
    int count = 0;
    while (cur != NULL) {
        count ++;
        cur = cur->next;
    }
    if (count < nth) {
        return -1;
    } else{
        count = 0;
        cur = head;
        while (count < nth - 1) {
            cur = cur->next;
            count++;
        }
        return cur->data;
    }
```

```
}

//add item at the end of the list (return a pointer to the head of the list)
//if head is NULL then return the created node
struct node *add_last(struct node *head, int item) {
    struct node *new = malloc(sizeof(struct node));
    new->data = item;
    new->next = NULL;
    struct node *cur = head;
    if (head == NULL) {
        return new;
    }
    while (cur->next != NULL) {
        cur = cur->next;
    }
    cur->next = new;
    return head;

}

// remove first item from a list (return a pointer to the head of the list)
// do nothing if head is empty
// if last element return NULL;
struct node *remove_first(struct node *head) {
    if (head == NULL) {
        return NULL;
    }else {
        struct node *cur = head;

        head = head->next;
        free(cur);

        return head;
    }
}

//remove last item from a list (return a pointer to the head of the list)
//do nothing if head is empty
//if last element return null
struct node *remove_last(struct node *head) {
    struct node *cur = head;
    if (head == NULL || head->next == NULL) {
        return NULL;
    }
    struct node *temp = cur->next;
    while (cur != NULL && temp->next != NULL) {

            cur = cur->next;
            temp = temp->next;

        }

        cur->next = NULL;
        free(temp);



    return head;
}

// remove all items with value given (item) (say 25) from a list
// (return a pointer to the head of the list)
// do nothing if can't find item.
// i.e. 2 -> 2 -> 5, remove_item(head, 2) gives us 5
struct node *remove_item(struct node *head, int item) {
    if (head == NULL){
       return NULL;
    }
/*  while (head != NULL && head->data == item) {
        head = head->next;
    }
    if (head == NULL){
```

```
        return NULL;
    }
    struct node *cur = head;
    struct node *temp = cur->next;

    while (temp != NULL) {
        if (temp->data == item) {
            cur->next = temp->next;
            free(temp);
            temp = cur->next;
        } else {
            temp = temp->next;
            cur = cur->next;
        }
    }
*/
    struct node *cur = head;
    while (cur != NULL && cur->data == item) {
        head = head->next;
        free(cur);
        cur = head;
    }
    head = cur;
    if (head == NULL) {
        return NULL;
    }
    cur = head;
    while (cur != NULL && cur->next != NULL) {
        if (cur->next->data == item){
            struct node *temp = cur->next;
            cur->next = cur->next->next;
            free(temp);
        } else {
        cur = cur->next;
        }
    }
    return head;
}


void test_strings(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Strings ==\n");
    char *list1[] = {"2", "4", "8", "16"};
    struct node *head1 = strings_to_list(4, list1);
    print_list(head1);
    free_list(head1);
    char *list2[] = {"1", "2", "3", "4"};
    struct node *head2 = strings_to_list(4, list2);
    print_list(head2);
    free_list(head2);
    printf("Print Successful\n");
}

void test_sum(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Sum ==\n");
    char *list1[] = {"2", "4", "8", "16"};
    struct node *head = strings_to_list(4, list1);
    assert(sum_list(head) == 30);
    free_list(head);
    char *list2[] = {"1", "2", "3", "4"};
    head = strings_to_list(4, list2);
    assert(sum_list(head) == 10);
    free_list(head);
    assert(sum_list(NULL) == 0);
    printf("Sum Successful\n");
}

void test_find(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Find ==\n");
```

```
    char *list1[] = {"2", "2", "8", "16"};
    struct node *head = strings_to_list(4, list1);
    assert(find_item(head, 2)->data == 2);
    assert(find_item(head, 12) == NULL);
    assert(find_item(head, 2) == head);
    assert(find_item(NULL, 1) == NULL);
    free_list(head);
    printf("Find Successful\n");
}

void test_find_last(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Find Last ==\n");
    char *list1[] = {"2", "2", "8", "16"};
    struct node *head = strings_to_list(4, list1);
    assert(find_last(head) == 16);
    assert(find_last(NULL) == -1);
    free_list(head);
    printf("Find Last Successful\n");
}

void test_find_nth(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Find Nth ==\n");
    char *list1[] = {"2", "2", "8", "16"};
    struct node *head = strings_to_list(4, list1);
    assert(find_nth(head, 2) == 2);
    assert(find_nth(head, 12) == -1);
    assert(find_nth(head, 1) == 2);
    assert(find_nth(NULL, 1) == -1);
    free_list(head);
    printf("Find Nth Successful\n");
}

void test_add_last(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Add Last ==\n");
    char *list1[] = {"2", "2", "8", "16"};
    struct node *head = strings_to_list(4, list1);
    head = add_last(head, 2);
    assert(head->next->next->next->next != NULL);
    assert(head->next->next->next->next->data == 2);
    free_list(head);
    head = add_last(NULL, 1);
    assert(head != NULL);
    assert(head->data == 1);
    free_list(head);
    printf("Add Last Successful\n");
}

void test_remove_first(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Remove First ==\n");
    char *list1[] = {"2", "5", "3"};
    struct node *head = strings_to_list(3, list1);
    head = remove_first(head);
    assert(head != NULL && head->data == 5);
    head = remove_first(head);
    assert(head != NULL && head->data == 3);
    head = remove_first(head);
    assert(head == NULL);
    head = remove_first(head);
    assert(head == NULL);
    printf("Remove first Successful\n");
}

void test_remove_last(void) {
    // TODO: Write some test cases for your functions
    printf("== Testing Remove Last ==\n");
    char *list1[] = {"2", "5", "3"};
    struct node *head = strings_to_list(3, list1);
    head = remove_last(head);
```

```
    assert(head != NULL && head->next != NULL &&
           head->next->next == NULL && head->next->data == 5 && head->data == 2);
    head = remove_last(head);
    assert(head != NULL && head->next == NULL && head->data == 2);
    head = remove_last(head);
    assert(head == NULL);
    head = remove_last(head);
    assert(head == NULL);
    printf("Remove Last Successful\n");
}

void test_remove_item(void) {
     // TODO: Write some test cases for your functions
    printf("== Testing Remove Item ==\n");
    char *list1[] = {"2", "5", "3", "7"};
    struct node *head = strings_to_list(4, list1);
    head = remove_item(head, 0);
    assert(head != NULL                    && head->data == 2);
    assert(head->next != NULL              && head->next->data == 5);
    assert(head->next->next != NULL        && head->next->next->data == 3);
    assert(head->next->next->next != NULL && head->next->next->next->data == 7);
    assert(head->next->next->next->next == NULL);

    head = remove_item(head, 2);
    assert(head != NULL              && head->data == 5);
    assert(head->next != NULL        && head->next->data == 3);
    assert(head->next->next != NULL && head->next->next->data == 7);
    assert(head->next->next->next == NULL);

    head = remove_item(head, 3);
    assert(head != NULL              && head->data == 5);
    assert(head->next != NULL        && head->next->data == 7);
    assert(head->next->next == NULL);

    head = remove_item(head, 7);
    assert(head != NULL && head->data == 5);
    assert(head->next == NULL);

    head = remove_item(head, 5);
    assert(head == NULL);
    head = remove_item(head, 0);
    assert(head == NULL);

    char *list2[] = {"2", "2", "5"};
    head = strings_to_list(3, list2);
    head = remove_item(head, 2);
    assert(head->data == 5);
    assert(head->next == NULL);
    free_list(head);
    printf("Remove Item Successful\n");
}
```