Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

(Autonomous Institute Affiliated to University of Mumbai)

Name: Jhaveri Varun Nimitt

UID: 2023800042

Batch: CSE A Batch C

Experiment No.:3

Aim: Singly Linked List application

Problem:

Remove Duplicates from Sorted Linked

List

Given a random list, delete all duplicates such that each element appears only once.

For example, Given:2,4,1,3,2 sorted list created :1->2->2->3->4, return 1->2->3->4

Task1: Creation of sorted list

Rask2: Removal of duplicates

METHOD 1 (single pointer aka returning head so its updated in main):

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

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

struct Node* sortedInsert(struct Node* head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (!head || head->data >= data) {
       newNode->next = head;
       head = newNode;
   }
   else {
       struct Node* current = head;
       while (current->next && current->next->data < data) {
           current = current->next;
       }
       newNode->next = current->next;
       current->next = newNode;
   }
   return head;
}

struct Node* removeDuplicates(struct Node* head) {
   struct Node* current = head;

   while (current && current->next) {
       if (current->data == current->next->data) {
           struct Node* temp = current->next;
           current->next = current->next->next;
           free(temp);
       } else {
           current = current->next;
       }
   }
   return head;
}

void printList(struct Node* node) {
   while (node) {
```

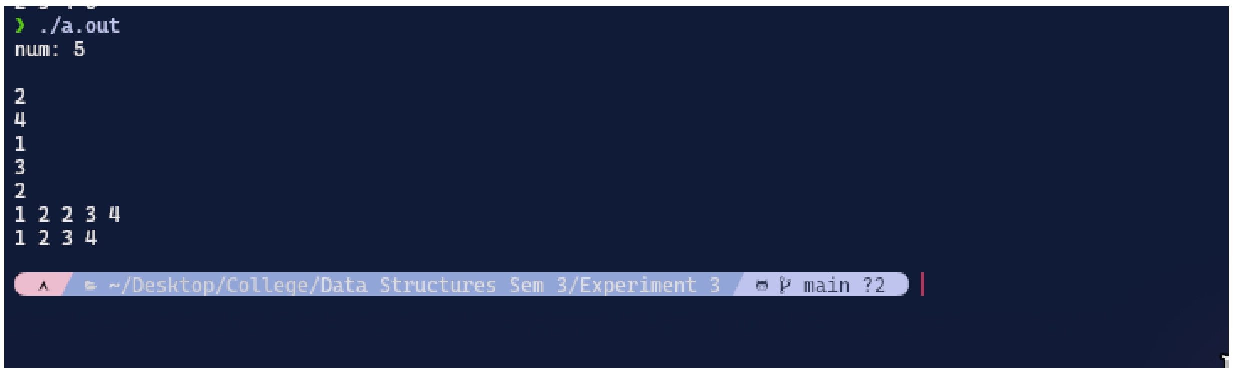
```c
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int n, data;

    printf("list length: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("element %d: ", i + 1);
        scanf("%d", &data);
        head = sortedInsert(head, data);
    }

    printf(":\n");
    printList(head);
    head = removeDuplicates(head);
    printf("\n");
    printList(head);
}
```

METHOD 2 (double pointer aka modify head directly):

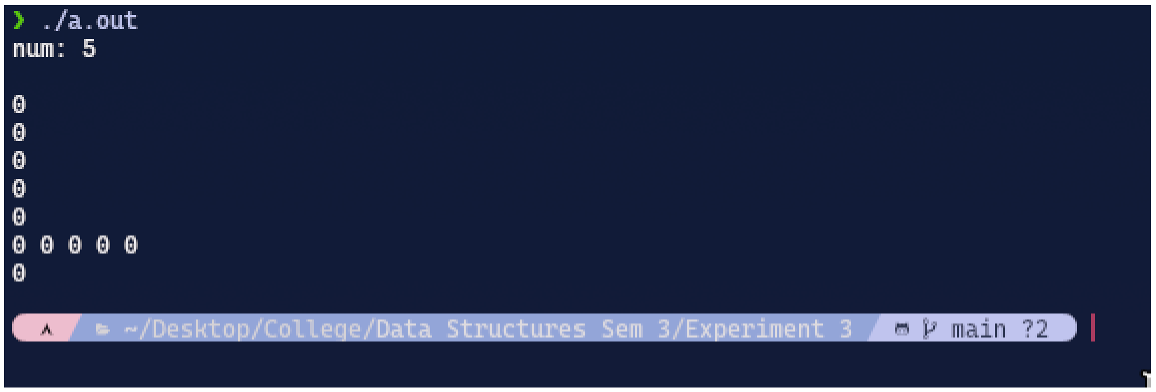
```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void sortedInsert(struct Node** headRef, int data) {

   struct Node* newNode = createNode(data);
   struct Node* current;

   if (*headRef == NULL || (*headRef)->data >= newNode->data) {
       newNode->next = *headRef;
       *headRef = newNode;
   } else {
       current = *headRef;
       while (current->next != NULL && current->next->data < newNode->data) {
           current = current->next;
       }
       newNode->next = current->next;
       current->next = newNode;
   }
}

void removeDuplicates(struct Node* head) {
   struct Node* current = head;
   struct Node* next_next;
   if (current == NULL)
       return;
   while (current->next != NULL) {
       if (current->data == current->next->data) {
           next_next = current->next->next;
           free(current->next);
           current->next = next_next;
       } else {
           current = current->next;
       }
   }
}

void printList(struct Node* node) {
   while (node != NULL) {
       printf("%d ", node->data);
       node = node->next;
   }
   printf("\n");
}

int isValidInteger(const char* str) {
   char* endptr;
   strtol(str, &endptr, 10);
   if (*str == '\0' || *endptr != '\0') return 0;
   return 1;
}

void trimNewline(char* str) {
   char* pos;
   if ((pos = strchr(str, '\n')) != NULL) {
       *pos = '\0';
   }
}

int main() {
   struct Node* head = NULL;
   int n;
   char input[100];

   printf("num: ");
   fgets(input, sizeof(input), stdin);
   trimNewline(input);
   if (!isValidInteger(input)) {
       printf("input messed up please try again.\n");
       return 1;
   }
   n = atoi(input);
```

```
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("element %d: ", i + 1);
        fgets(input, sizeof(input), stdin);
        trimNewline(input);
        if (!isValidInteger(input)) {
            printf("input messed up please try again.\n");
            i--;
            continue;
        }
        int value = atoi(input);
        sortedInsert(&head, value);
    }
    printList(head);
    removeDuplicates(head);
    printList(head);

    return 0;
}
```

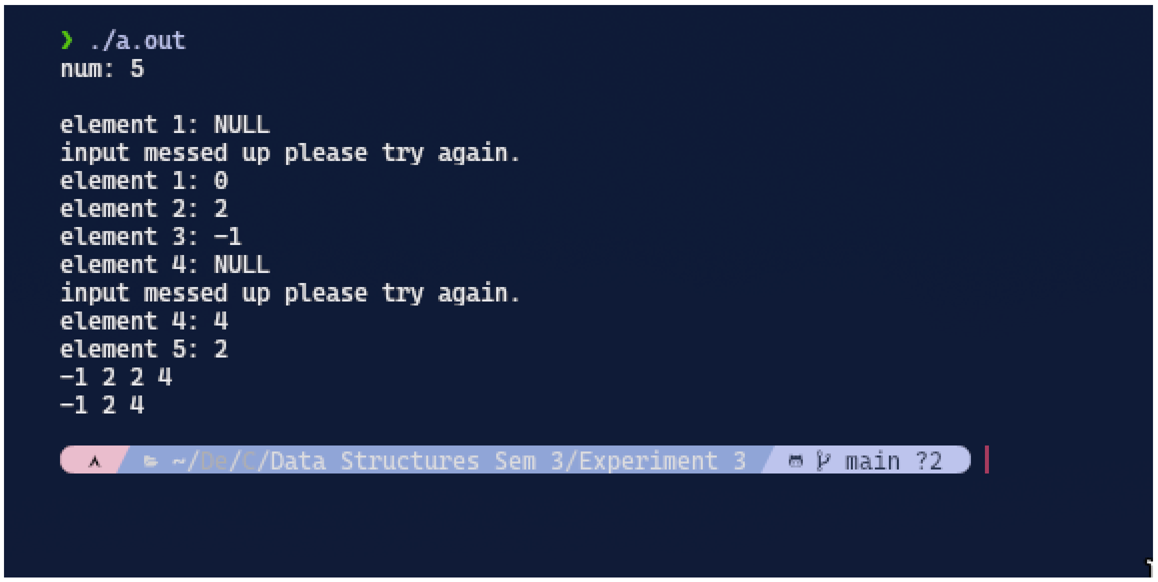
OUTPUT (using the 2nd method) :



```
❯ ./a.out
num: 5

2
4
1
3
2
1 2 2 3 4
1 2 3 4

 ~/Desktop/College/Data Structures Sem 3/Experiment 3   main ?2
```

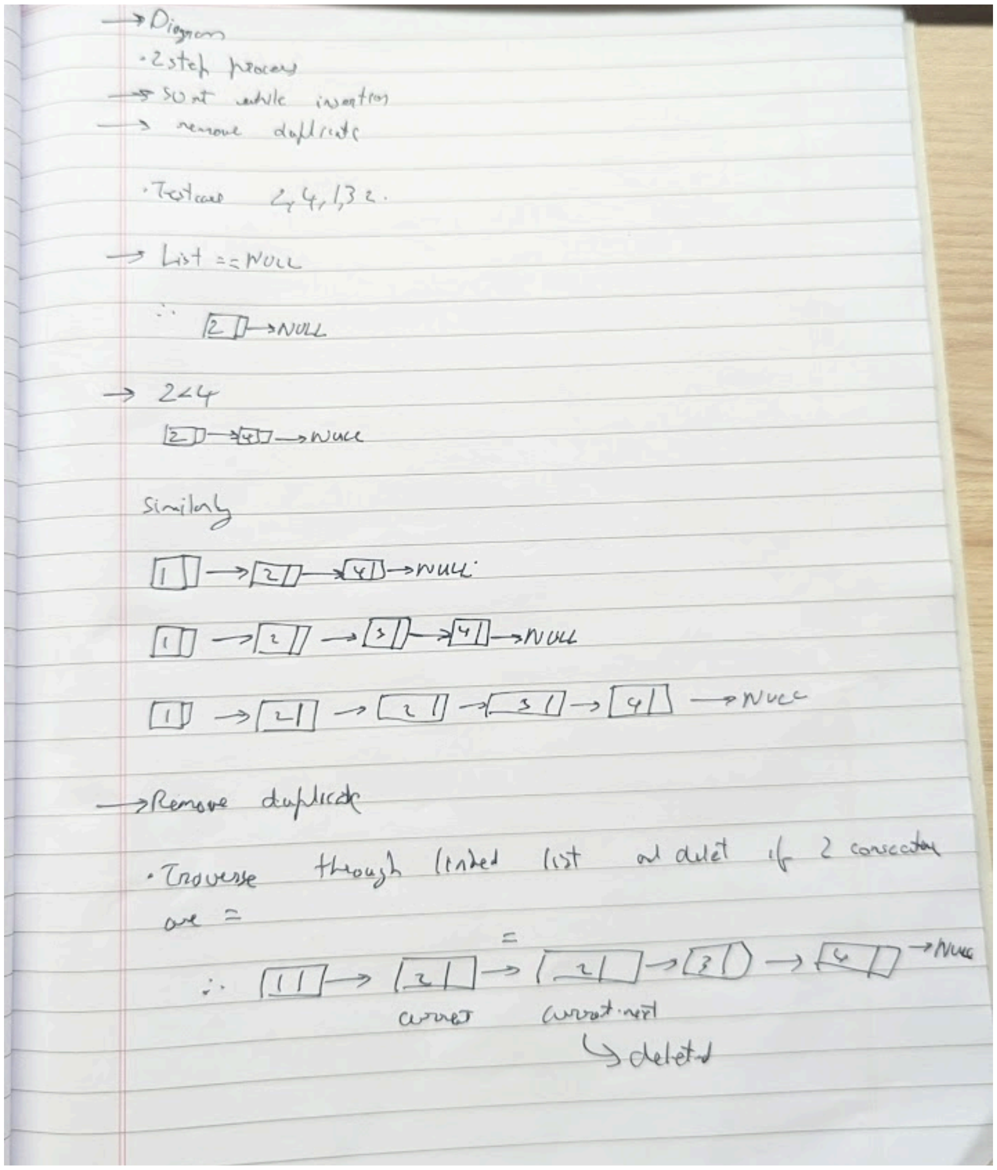


```
❯ ./a.out
num: 5

0
0
0
0
0
0 0 0 0 0
0

 ~/Desktop/College/Data Structures Sem 3/Experiment 3   main ?2
```



```
❯ ./a.out
num: 5

element 1: NULL
input messed up please try again.
element 1: 0
element 2: 2
element 3: -1
element 4: NULL
input messed up please try again.
element 4: 4
element 5: 2
-1 2 2 4
-1 2 4

 ~/De/C/Data Structures Sem 3/Experiment 3   main ?2
```

Handwritten explanation part :



→ Diagram

- 2 step process

→ sort while insertion

→ remove duplicate

- Testcase 2, 4, 1, 3, 2.

→ List == NULL

$\therefore \boxed{2} \rightarrow NULL$

→ 2 < 4

$\boxed{2} \rightarrow \boxed{4} \rightarrow NULL$

Similarly

$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow NULL$

$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{3} \rightarrow \boxed{4} \rightarrow NULL$

$\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{2} \rightarrow \boxed{3} \rightarrow \boxed{4} \rightarrow NULL$

→ Remove duplicate

- Traverse through linked list and delete if 2 consecutive are =

$$\therefore \boxed{1} \rightarrow \underset{current}{\boxed{2}} \overset{=}{\rightarrow} \underset{\substack{current.next \\ \hookrightarrow deleted}}{\boxed{2}} \rightarrow \boxed{3} \rightarrow \boxed{4} \rightarrow NULL$$