LINKED LIST _ASSIGNMENTS

1. Refer the example code of single linked list.
a. Modify existing data structure to include below additional data members. Update all relevant existing functions to accommodate this member update and test.

```
{
char id[MAX_ID_LEN]; //let max id length be 5
int val;
}
```

b. Add a function below to perform search and update based on name (str) input.

```
extern int search_update_name(Node **, char *search, char *replace);
```

c. Add functions below() to perform a sorted insert operation (ascending order) on the list based on name (str) and on val.

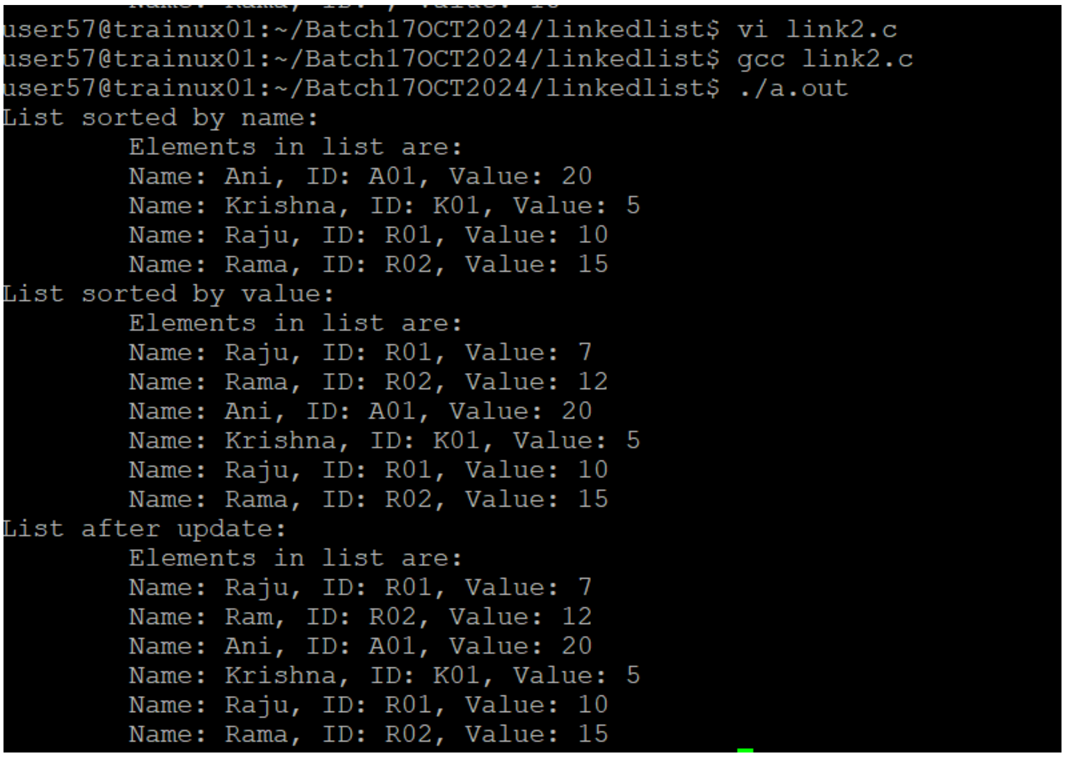
```
//insert based on name (str member) in increasing order
extern int sorted_insert_name(Node **, char *, int val);
//insert based on val in increasing order
extern int sorted_insert_val(Node **, char *, int val);
```

Sol:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define SIZE 100

#define MAX_ID_LEN 6

#define SUCCESS 0

#define FAILURE -1


typedef struct Node {

   char str[SIZE];

   char id[MAX_ID_LEN];

   int val;

   struct Node* next;

} Node;


int create_node(Node **new_node, int data_len)
```

```
{
    *new_node = (Node *)malloc(sizeof(Node));
    if (NULL == *new_node)
    {
        perror("Error while creating node");
        return FAILURE;
    }

    (*new_node)->next = NULL;
    return SUCCESS;
}

void myfflush(void)
{
    while ('\n' != getchar());
}

void get_string_input(char *input_string)
{
    while (1)
    {
        fgets(input_string, SIZE - 1, stdin);
        int len = strlen(input_string);
        if (1 == len)
        {
            printf("\tEmpty string, enter again: ");
            memset(input_string, 0, SIZE);
            continue;
        }
        if ('\n' == input_string[len - 1])
        {
            input_string[len - 1] = '\0';
        }
        else
        {
            myfflush();
        }
        break;
    }
}

void display_list(Node *head)
{
    if (NULL == head)
    {
        printf("\tList is empty\n");
        return;
    }
    else
    {
        printf("\tElements in list are:\n");
    }
    while (head != NULL)
    {
        printf("\tName: %s, ID: %s, Value: %d\n", head->str, head->id, head->val);
        head = head->next;
    }
}

void free_list(Node **head)
{
    Node *tmp = *head;
    while (tmp != NULL)
    {
        Node *free_node = tmp;
        tmp = tmp->next;
        free(free_node);
    }
    *head = NULL;
}

int sorted_insert_name(Node **head, char *name, int val, char *id)
{
    Node *new_node = NULL;
    if (create_node(&new_node, sizeof(Node)) == FAILURE)
    {
        return FAILURE;
    }

    strcpy(new_node->str, name);
    new_node->val = val;
    strcpy(new_node->id, id);

    if (NULL == *head || strcmp((*head)->str, name) > 0) {
        new_node->next = *head;
        *head = new_node;
        return SUCCESS;
    }

    Node *temp = *head;
    while (temp->next != NULL && strcmp(temp->next->str, name) < 0)
    {
        temp = temp->next;
    }
    new_node->next = temp->next;
    temp->next = new_node;
    return SUCCESS;
}

int sorted_insert_val(Node **head, char *name, int val, char *id)
{
    Node *new_node = NULL;
    if (create_node(&new_node, sizeof(Node)) == FAILURE)
    {
        return FAILURE;
    }

    strcpy(new_node->str, name);
    new_node->val = val;
    strcpy(new_node->id, id);

    if (NULL == *head || (*head)->val > val) {
        new_node->next = *head;
        *head = new_node;
        return SUCCESS;
    }

    Node *temp = *head;
    while (temp->next != NULL && temp->next->val < val)
    {
        temp = temp->next;
    }
    new_node->next = temp->next;
    temp->next = new_node;
    return SUCCESS;
}

int search_update_name(Node **head, char *search, char *replace)
{
    Node *temp = *head;

    while (temp != NULL)
    {
        if (strcmp(temp->str, search) == 0)
        {
            strcpy(temp->str, replace);
            return SUCCESS;
        }
        temp = temp->next;
    }
    return FAILURE;
}

int main()
{
    Node *head = NULL;

    // Example insertions with name, value, and id
    sorted_insert_name(&head, "Raju", 10, "R01");
    sorted_insert_name(&head, "Rama", 15, "R02");
    sorted_insert_name(&head, "Krishna", 5, "K01");
    sorted_insert_name(&head, "Ani", 20, "A01");

    printf("List sorted by name:\n");
    display_list(head);
    sorted_insert_val(&head, "Raju", 7, "R01");

    sorted_insert_val(&head, "Rama", 12, "R02");


    printf("List sorted by value:\n");

    display_list(head);


    search_update_name(&head, "Rama", "Ram");


    printf("List after update:\n");

    display_list(head);


    free_list(&head);


    return 0;

}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/linkedlist$ vi link2.c
user57@trainux01:~/Batch17OCT2024/linkedlist$ gcc link2.c
user57@trainux01:~/Batch17OCT2024/linkedlist$ ./a.out
List sorted by name:
        Elements in list are:
        Name: Ani, ID: A01, Value: 20
        Name: Krishna, ID: K01, Value: 5
        Name: Raju, ID: R01, Value: 10
        Name: Rama, ID: R02, Value: 15
List sorted by value:
        Elements in list are:
        Name: Raju, ID: R01, Value: 7
        Name: Rama, ID: R02, Value: 12
        Name: Ani, ID: A01, Value: 20
        Name: Krishna, ID: K01, Value: 5
        Name: Raju, ID: R01, Value: 10
        Name: Rama, ID: R02, Value: 15
List after update:
        Elements in list are:
        Name: Raju, ID: R01, Value: 7
        Name: Ram, ID: R02, Value: 12
        Name: Ani, ID: A01, Value: 20
        Name: Krishna, ID: K01, Value: 5
        Name: Raju, ID: R01, Value: 10
        Name: Rama, ID: R02, Value: 15
```