DAY 5 ASSIGNMENT

WAP to implement a skip list.

CODE

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

#define PLUS_INFINITY INT_MAX
#define MINUS_INFINITY INT_MIN

int c = 0; // Global comparison counter

typedef struct node  //Quad-node
{
    int key;
    struct node *f;
    struct node *b;
    struct node *u;
    struct node *d;
} node;

typedef struct skiplist // list
{
    node *head;
    node *tail;
    int level;
    int size;
} skiplist;

int toss()
{
    return rand() % 2; //genrates 2 values 0 and 1
}

node *createNode(int key) //will create a quad node pointing to null on all four
sides
{
    node *newNode = (node *)malloc(sizeof(node));
    if (newNode == NULL)
    {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->key = key;
    newNode->f = NULL;
    newNode->b = NULL;
    newNode->u = NULL;
    newNode->d = NULL;
    return newNode;
}
```

```c
skiplist *createSkiplist() //will initialize a list at levvel 0
{
    skiplist *sl = (skiplist *)malloc(sizeof(skiplist));
    if (sl == NULL)
    {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }

    node *p2 = createNode(PLUS_INFINITY); //node for +∞
    node *p1 = createNode(MINUS_INFINITY); //node for -∞

    p1->f = p2;
    p2->b = p1;
    sl->head = p1;
    sl->tail = p2;
    sl->level = 0;
    sl->size = 0;

    return sl;
}

void addEmptyList(skiplist *sl) //will add an empty list to a level up of current 
level
{
    node *p1 = createNode(MINUS_INFINITY);
    node *p2 = createNode(PLUS_INFINITY);

    p1->f = p2;
    p1->d = sl->head;

    p2->b = p1;
    p2->d = sl->tail;

    sl->head->u = p1;
    sl->tail->u = p2;

    sl->head = p1;
    sl->tail = p2;

    sl->level++;
}

node *search(skiplist *sl, int value ) //searching for the element
{
    node *t = sl->head;
    while (1)
    {
        while (t->f->key != PLUS_INFINITY && t->f->key <= value) //if next element 
is not +∞ , and next element is lesser than key then ...
        {
            t = t->f; // move tail
            c++; // incrementing comparision counter
        }
        if (t->d)
        {
            t = t->d;
            c++; // one counter gets increment when compared with the next greater 
element and traversal moves down
        }
        else
            break;
    }
    return t;
}

void insert(skiplist *sl, int value)
{
    c = 0;
    node *p = search(sl, value); //searching for the value, returns a pointer to
the node <= the value
    if (p->key == value) // if duplicates found
    {
        printf("Value %d already exists in the skiplist. Comparisons: %d\n",
value, c);
        return;
    }
    node *newNode = createNode(value); // quad node created
    newNode->f = p->f; // setting up the node pointers
    newNode->b = p;
    p->f->b = newNode;
    p->f = newNode;

    int current_level = 0;
    node *curr_node = newNode;
    while (toss()) //tossing the coin
    {
        if (current_level >= sl->level)
            addEmptyList(sl); //adding one extra list on top

        while (p->b && p->u == NULL)
            p = p->b;

        if (p->u == NULL)
            break;
        p = p->u;

        node *newNodeUp = createNode(value); //adding the node to new levels if
tosses a head (1)
        newNodeUp->f = p->f;
        newNodeUp->b = p;
        newNodeUp->d = curr_node;
        p->f->b = newNodeUp;
        p->f = newNodeUp;
        curr_node->u = newNodeUp;
        curr_node = newNodeUp;

        current_level++; //incrementing the level
    }
    // Always ensure the top level is empty.
    // If the top level contains a promoted node (i.e. it’s not just sentinels),
add one extra level.
    if (sl->head->f != sl->tail)
    {
        addEmptyList(sl);
    }
    sl->size++; // incrementing the size of list
    printf("Value %d inserted. Comparisons: %d\n", value, c);
}

void deleteExtras(skiplist *sl)
{
    // While there is a level below and the current top level is empty (only
sentinels)
    while (sl->head->d != NULL && sl->head->f->key == PLUS_INFINITY)
    {
        node* old_left = sl->head;      // current top level - left sentinel
        node* old_right = sl->tail;    // current top level - right sentinel

        // Move down one level
        sl->head = old_left->d;
        sl->tail = old_right->d;

        // Remove upward links from the new top level
        if (sl->head)
            sl->head->u = NULL;
        if (sl->tail)
            sl->tail->u = NULL;

        // Free the sentinel nodes of the old top level
        free(old_left);
        free(old_right);

        sl->level--;
    }
}

void deleteValue(skiplist *sl, int value)
{
    c = 0;
    node *p = search(sl, value); //searching for the value, returns a pointer to
the node <= the value
    if (p->key != value)  // checks whether present or not
    {
        printf("Value %d not found. Comparisons: %d\n", value, c);
        return;
    }

    while (p)
    {
        p->b->f = p->f; //adjusting the pointer so to detach and hence delete the
node
        p->f->b = p->b;
        node *temp = p;
        p = p->u; //going up and deleting all instances
        free(temp);
    }
    sl->size--;
    printf("Value %d deleted. Comparisons: %d\n", value, c);
    deleteExtras(sl);//deleting the extra lists if present after deletion  (as
only 1 list should be present in extra)

    if (sl->head->f != sl->tail) {
        addEmptyList(sl);
    }
}

void searchValue(skiplist *sl, int value)
{
    c = 0;
    node *p = search(sl, value); // will also return the pointer pointing to key
<= value
    if (p->key == value)
        printf("Value %d found. Comparisons: %d\n", value, c);
    else
        printf("Value %d not found. Comparisons: %d\n", value, c);
}

void display(skiplist *sl) // for displaying the skiplist
```

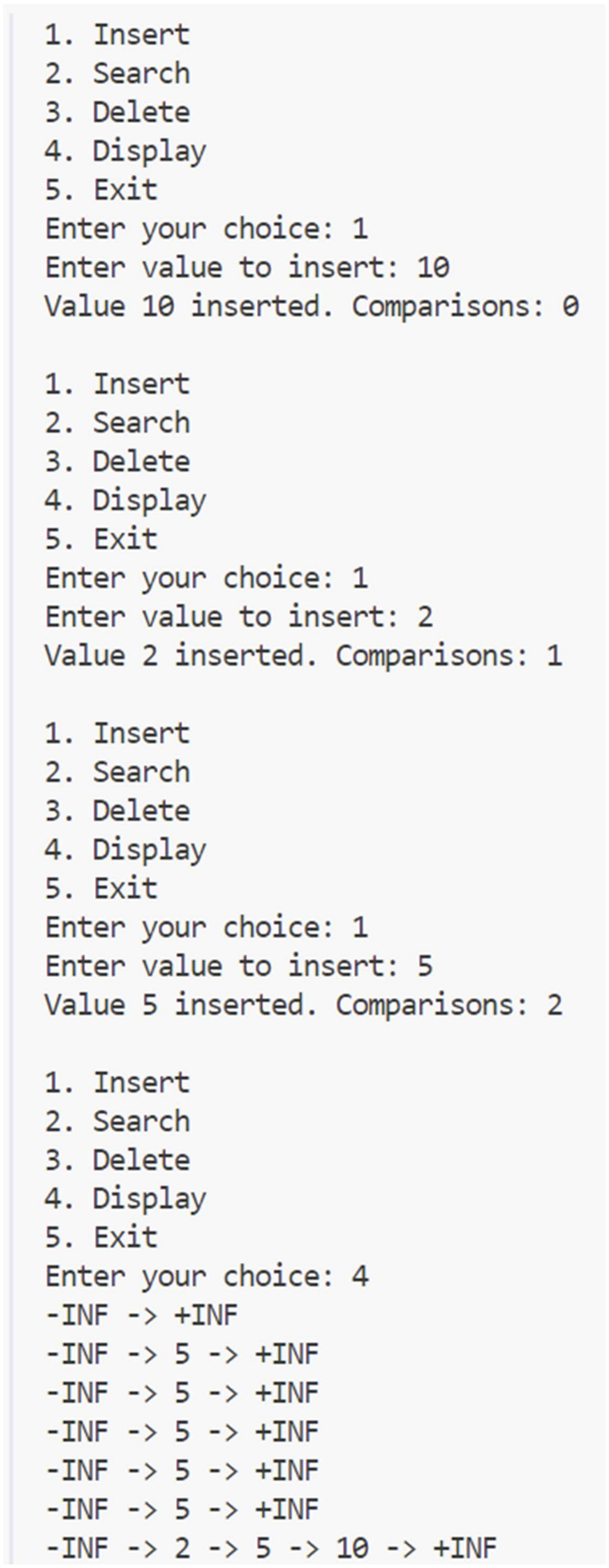
```
{
    node *t = sl->head;
    while (t)
    {
        node *q = t;
        printf("-INF -> ");
        while (q->f->key != PLUS_INFINITY)
        {
            printf("%d -> ", q->f->key);
            q = q->f;
        }
        printf("+INF\n");
        t = t->d; //stops when t->d points to null
    }
}

int main()
{
    srand(time(NULL));
    skiplist *sl = createSkiplist();

    int choice, value;
    do
    {
        printf("\n1. Insert\n2. Search\n3. Delete\n4. Display\n5. Exit\nEnter your 
choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insert(sl, value);
            break;
        case 2:
            printf("Enter value to search: ");
            scanf("%d", &value);
            searchValue(sl, value);
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteValue(sl, value);
            break;
        case 4:
            display(sl);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

OUTPUT

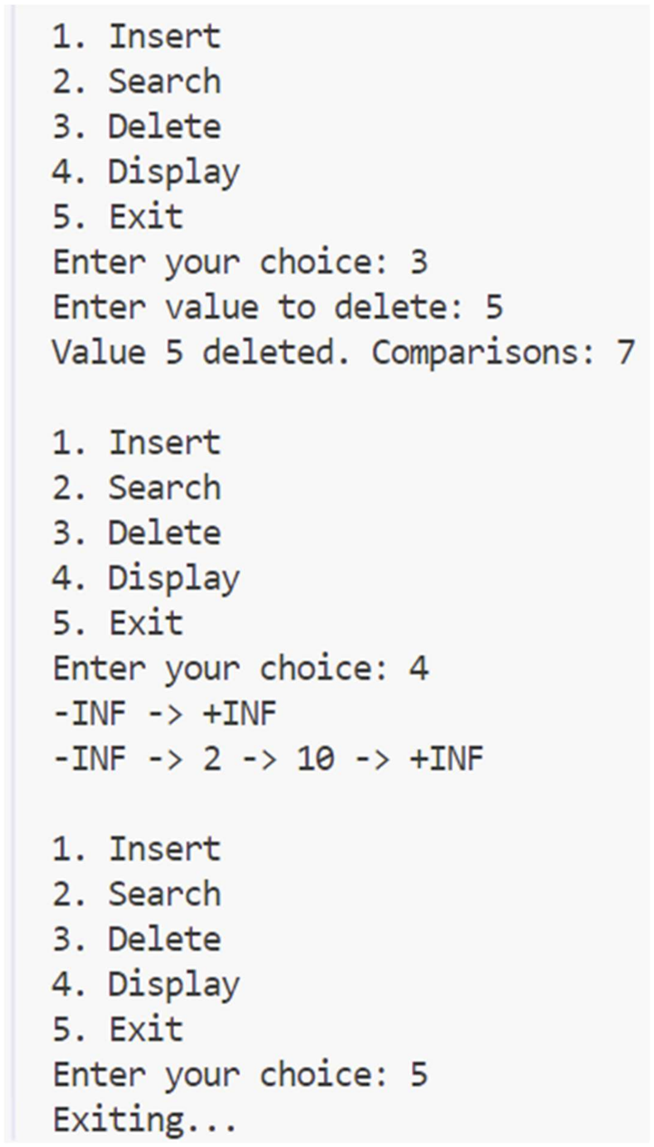


```
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter value to insert: 10
Value 10 inserted. Comparisons: 0

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter value to insert: 2
Value 2 inserted. Comparisons: 1

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter value to insert: 5
Value 5 inserted. Comparisons: 2

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 4
-INF -> +INF
-INF -> 5 -> +INF
-INF -> 5 -> +INF
-INF -> 5 -> +INF
-INF -> 5 -> +INF
-INF -> 5 -> +INF
-INF -> 2 -> 5 -> 10 -> +INF
```



```
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 3
Enter value to delete: 5
Value 5 deleted. Comparisons: 7

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 4
-INF -> +INF
-INF -> 2 -> 10 -> +INF

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 5
Exiting...
```