SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

Department of Information Technology

COURSE CODE: DJS22ITL502 **DATE:** 21-10-24

COURSE NAME: Advanced Data Structures Laboratory **CLASS:** TY B. TECH

NAME: Anish Sharma **DIV:** IT1-1 **ROLL:** I011

EXPERIME NT NO. 8

CO/LO: Choose appropriate data structure and use it to design algorithm for solving a specific problem

AIM / OBJECTIVE: To implement various operations on a Segment Tree.

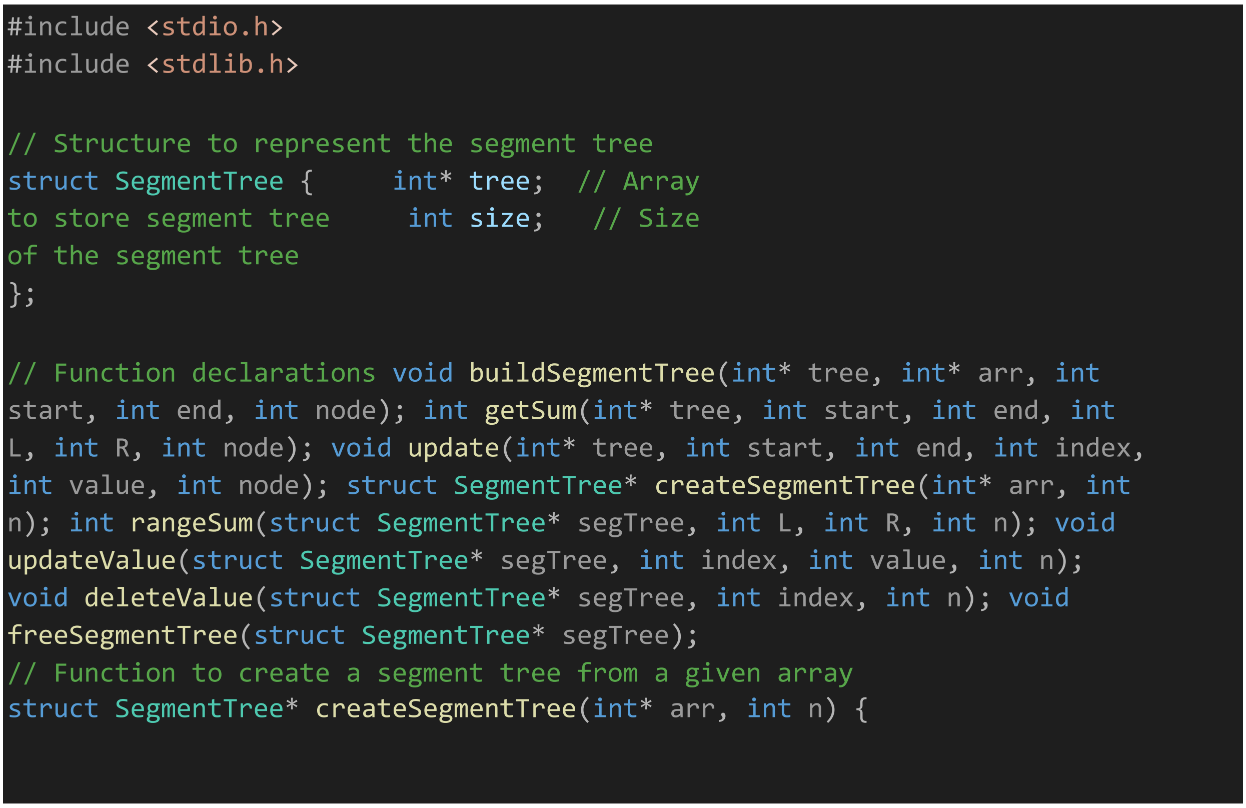
DESCRIPTION OF EXPERIMENT:

Properties of Segment Tree:

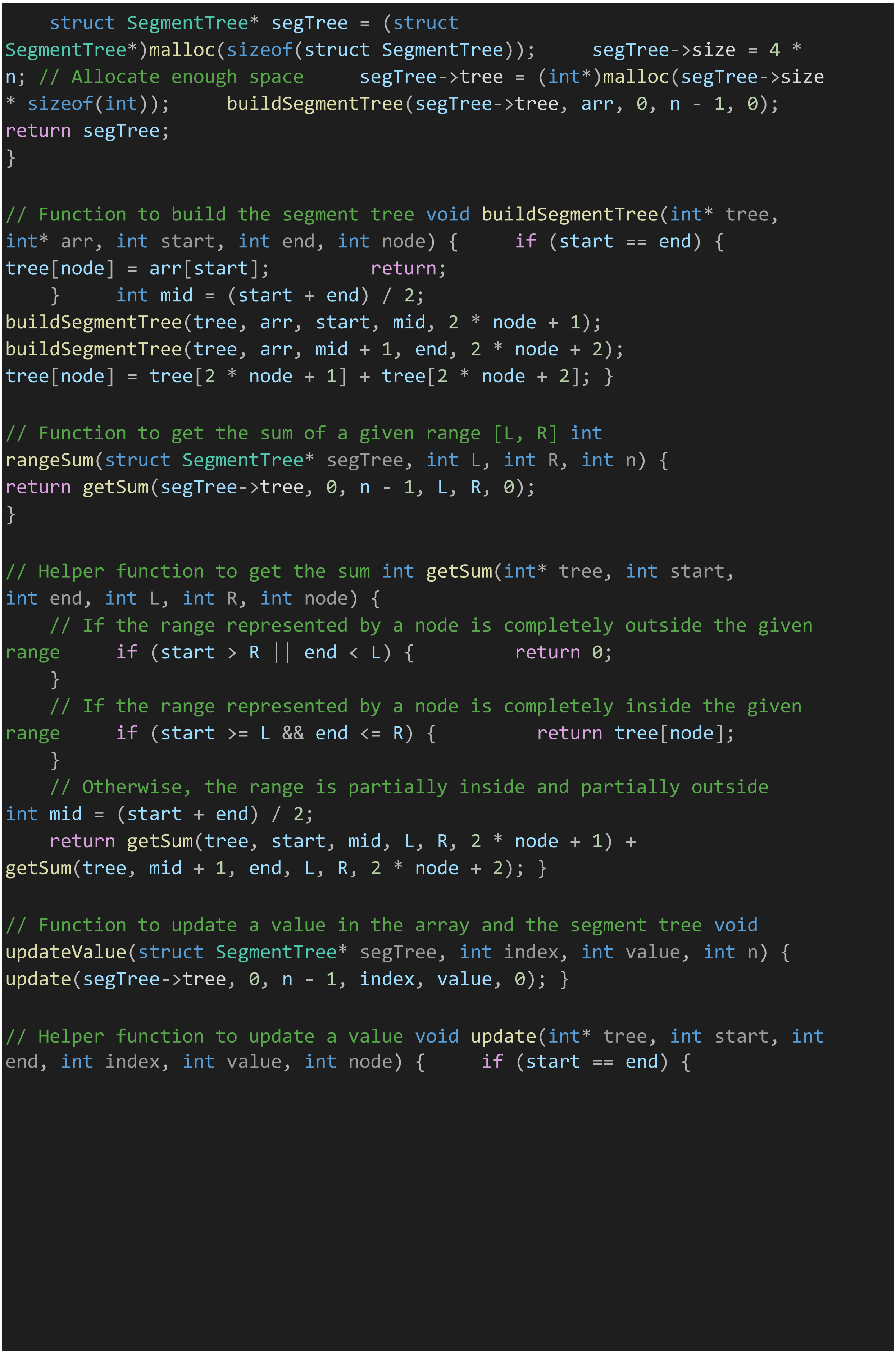
Efficiency: O(log n) time for updates and queries.
Dynamic Updates: Supports efficient modifications of array elements.
Space: Requires O(4n) storage.
Range Queries: Handles various queries like sum, min, and max.
Non-Overlapping Segments: Each node represents a segment of the array.

TECHNOLOGY STACK USED: C

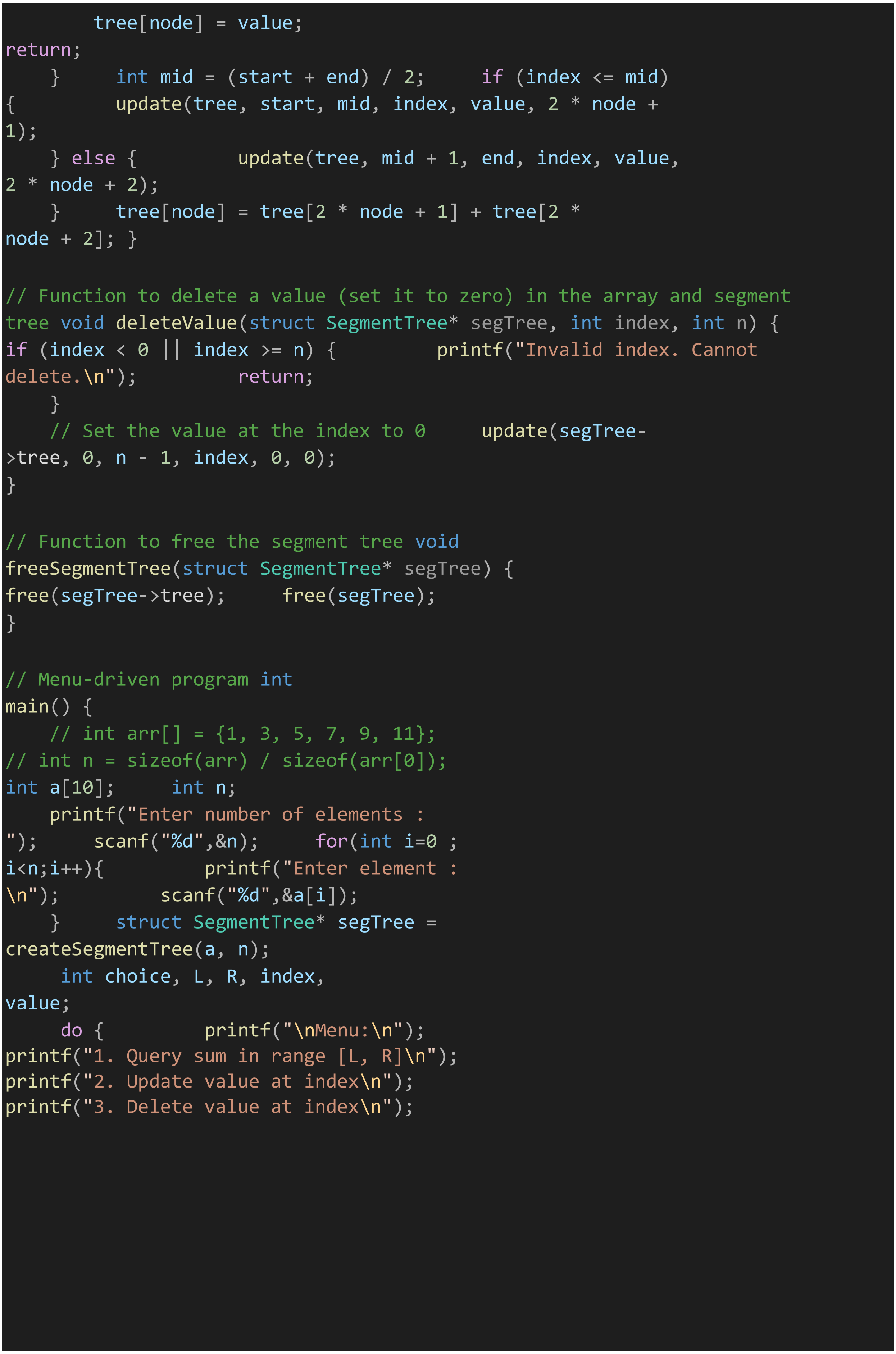
CODE:



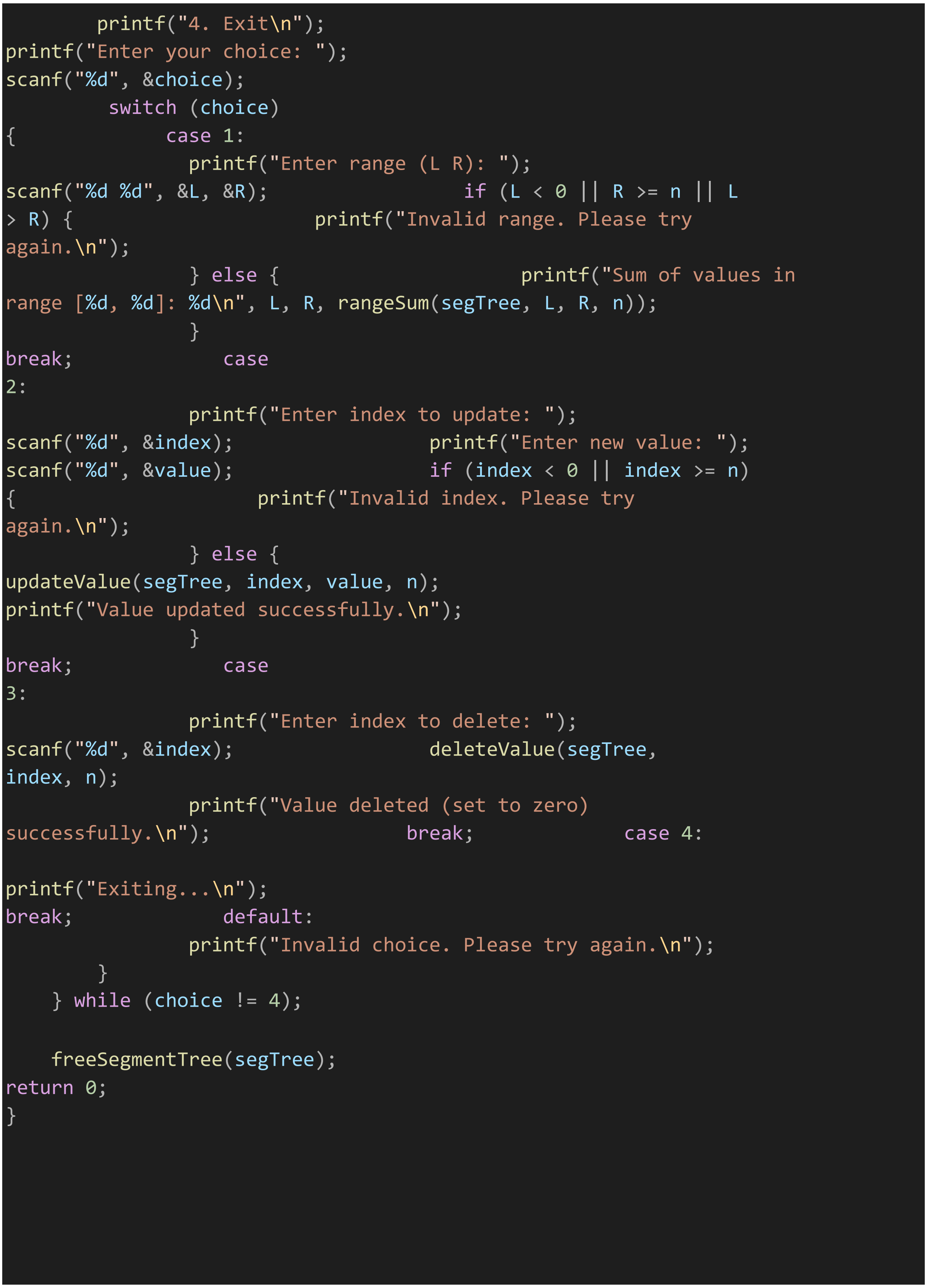
```
#include <stdio.h>
#include <stdlib.h>

// Structure to represent the segment tree
struct SegmentTree {      int* tree;   // Array
to store segment tree      int size;    // Size
of the segment tree
};

// Function declarations void buildSegmentTree(int* tree, int* arr, int
start, int end, int node); int getSum(int* tree, int start, int end, int
L, int R, int node); void update(int* tree, int start, int end, int index,
int value, int node); struct SegmentTree* createSegmentTree(int* arr, int
n); int rangeSum(struct SegmentTree* segTree, int L, int R, int n); void
updateValue(struct SegmentTree* segTree, int index, int value, int n);
void deleteValue(struct SegmentTree* segTree, int index, int n); void
freeSegmentTree(struct SegmentTree* segTree);
// Function to create a segment tree from a given array
struct SegmentTree* createSegmentTree(int* arr, int n) {
```



```
    struct SegmentTree* segTree = (struct
SegmentTree*)malloc(sizeof(struct SegmentTree));     segTree->size = 4 *
n; // Allocate enough space     segTree->tree = (int*)malloc(segTree->size
* sizeof(int));     buildSegmentTree(segTree->tree, arr, 0, n - 1, 0);
return segTree;
}

// Function to build the segment tree void buildSegmentTree(int* tree,
int* arr, int start, int end, int node) {     if (start == end) {
tree[node] = arr[start];         return;
    }     int mid = (start + end) / 2;
buildSegmentTree(tree, arr, start, mid, 2 * node + 1);
buildSegmentTree(tree, arr, mid + 1, end, 2 * node + 2);
tree[node] = tree[2 * node + 1] + tree[2 * node + 2]; }

// Function to get the sum of a given range [L, R] int
rangeSum(struct SegmentTree* segTree, int L, int R, int n) {
return getSum(segTree->tree, 0, n - 1, L, R, 0);
}

// Helper function to get the sum int getSum(int* tree, int start,
int end, int L, int R, int node) {
    // If the range represented by a node is completely outside the given
range     if (start > R || end < L) {         return 0;
    }
    // If the range represented by a node is completely inside the given
range     if (start >= L && end <= R) {         return tree[node];
    }
    // Otherwise, the range is partially inside and partially outside
int mid = (start + end) / 2;
    return getSum(tree, start, mid, L, R, 2 * node + 1) +
getSum(tree, mid + 1, end, L, R, 2 * node + 2); }

// Function to update a value in the array and the segment tree void
updateValue(struct SegmentTree* segTree, int index, int value, int n) {
update(segTree->tree, 0, n - 1, index, value, 0); }

// Helper function to update a value void update(int* tree, int start, int
end, int index, int value, int node) {     if (start == end) {
```

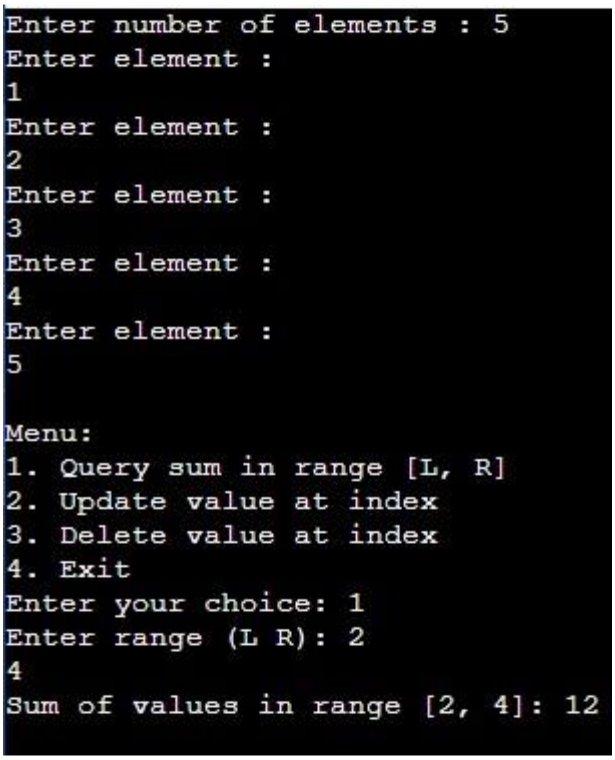


```
        tree[node] = value;
return;
    }     int mid = (start + end) / 2;      if (index <= mid)
{          update(tree, start, mid, index, value, 2 * node +
1);
    } else {          update(tree, mid + 1, end, index, value,
2 * node + 2);
    }     tree[node] = tree[2 * node + 1] + tree[2 *
node + 2]; }

// Function to delete a value (set it to zero) in the array and segment
tree void deleteValue(struct SegmentTree* segTree, int index, int n) {
if (index < 0 || index >= n) {         printf("Invalid index. Cannot
delete.\n");         return;
    }
    // Set the value at the index to 0      update(segTree-
>tree, 0, n - 1, index, 0, 0);
}

// Function to free the segment tree void
freeSegmentTree(struct SegmentTree* segTree) {
free(segTree->tree);     free(segTree);
}

// Menu-driven program int
main() {
    // int arr[] = {1, 3, 5, 7, 9, 11};
// int n = sizeof(arr) / sizeof(arr[0]);
int a[10];     int n;
    printf("Enter number of elements :
");     scanf("%d",&n);     for(int i=0 ;
i<n;i++){          printf("Enter element :
\n");          scanf("%d",&a[i]);
    }     struct SegmentTree* segTree =
createSegmentTree(a, n);
     int choice, L, R, index,
value;
     do {          printf("\nMenu:\n");
printf("1. Query sum in range [L, R]\n");
printf("2. Update value at index\n");
printf("3. Delete value at index\n");
```

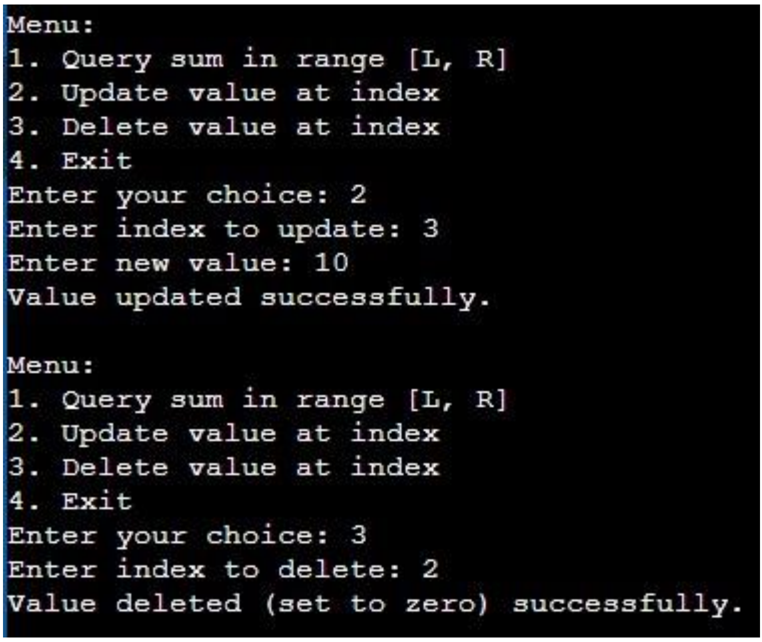


```
        printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
         switch (choice)
{              case 1:
                printf("Enter range (L R): ");
scanf("%d %d", &L, &R);                    if (L < 0 || R >= n || L
> R) {                         printf("Invalid range. Please try
again.\n");
                } else {                        printf("Sum of values in
range [%d, %d]: %d\n", L, R, rangeSum(segTree, L, R, n));
                }
break;               case
2:
                printf("Enter index to update: ");
scanf("%d", &index);                    printf("Enter new value: ");
scanf("%d", &value);                    if (index < 0 || index >= n)
{                     printf("Invalid index. Please try
again.\n");
                } else {
updateValue(segTree, index, value, n);
printf("Value updated successfully.\n");
                }
break;               case
3:
                printf("Enter index to delete: ");
scanf("%d", &index);                    deleteValue(segTree,
index, n);
                printf("Value deleted (set to zero)
successfully.\n");                  break;              case 4:

printf("Exiting...\n");
break;               default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    freeSegmentTree(segTree);
return 0;
}
```

OUTPUT:



```
Enter number of elements : 5
Enter element :
1
Enter element :
2
Enter element :
3
Enter element :
4
Enter element :
5

Menu:
1. Query sum in range [L, R]
2. Update value at index
3. Delete value at index
4. Exit
Enter your choice: 1
Enter range (L R): 2
4
Sum of values in range [2, 4]: 12
```



```
Menu:
1. Query sum in range [L, R]
2. Update value at index
3. Delete value at index
4. Exit
Enter your choice: 2
Enter index to update: 3
Enter new value: 10
Value updated successfully.

Menu:
1. Query sum in range [L, R]
2. Update value at index
3. Delete value at index
4. Exit
Enter your choice: 3
Enter index to delete: 2
Value deleted (set to zero) successfully.
```

Department of Information Technology

CONCLUSION: In this experiment we implemented variouus operations on a Segment Tree

REFERENCES:

1. Peter Brass, “Advanced Data Structures”, Cambridge University Press, 2008

Robert Sedgewick & Kevin Wayne, “Algorithms”, 4th Edition, Addison-Wesley Professional, 2011