Event Management System Documentation

Performed By - Divya Patel 19BCE167
Het Patel 19BCE172
Hrushi Patel 19BCE175

Problem Statement – A program to store details of participants for an event. It has functions like add, delete, edit, search, print data etc. Moreover generation of unique id for each entry.

Explanation of Implementation –

Data Structure Used – **Binary Search Tree with Doubly Linked List.**

As we need to store names, and other information in sorted order and need to have edit, delete, search etc. like functions, the data structure best suitable was Binary Search Tree.

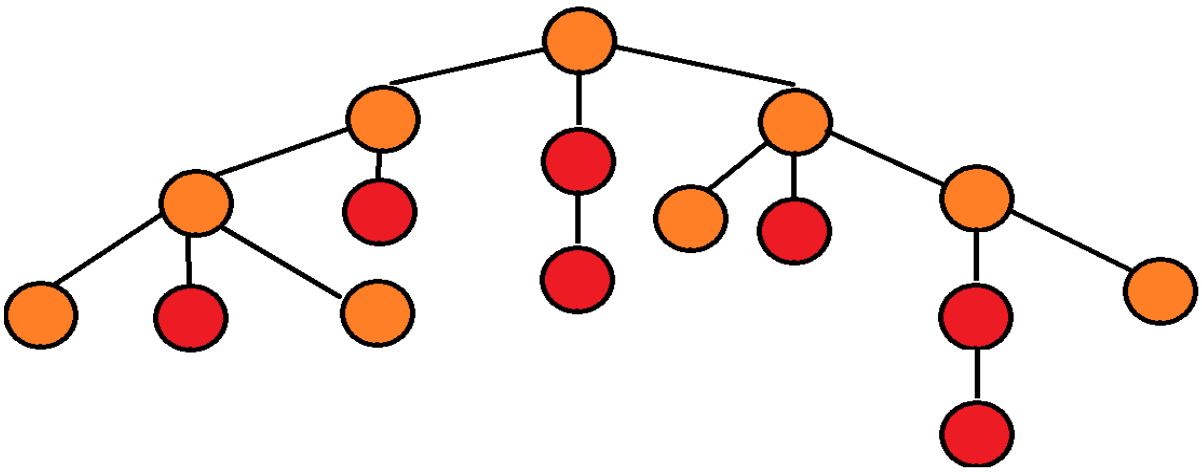
Still, we faced a problem of having duplicate keys (name). Hence, to resolve this problem, we added implementation of a doubly linked list in nodes containing duplicate names.

That is, a node of our BST had following contents of address,

1. **Left**
2. **Right**
3. **Parent**
4. **Middle**

While inserting data (keys and values) if we find out common key (name) already use, then we use its middle address to store data with same keys. Hence, creating a linked list. But as we also need to optimize the delete and edit functions, we used an extra address called parent which stored the address of just previous nodes or parent nodes. This gave rise to doubly linked list and indeed increasing the performance of deletion and edits.

The data structure that would be formed in the background will be as follows



The **orange part forms BST** here which forms backbone and divides the data uniformly in sorted arrangement, whereas the **red part forms doubly linked list** which is basically meant for storing same name.

Source Code -

```c
/* <Libaries> */
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include <Windows.h>
#include<string.h>
/* </Libaries> */

/* <Clears buffers on Std.Input Stream> */
#define flush fflush(stdin)
/* </Clears buffers on Std.Input Stream> */

/* <Node Structure of BST Typedef as Node> */
typedef struct node {
    char name[50];
    int number;
    int event_registered[3];
    int unique_id;
    struct node *left;
    struct node *right;
    struct node *middle;
    struct node *parent;
} node;
/* </Node Structure of BST Typedef as Node> */

/* <Some global varibles> */
int ans = 0;
int del = 0;
node *global_root;
FILE *fptr;
int unique_id_in = 0;
void print_pattern();
/* </Some global varibles> */

/* <Intro Screen> */
void intro()
{
    printf("Hello!\nWelcome to the event management system!\n");
    printf("This program is created by \n");
    printf("\t19bce167 Divya Patel\n");
    printf("\t19bce172 Het Patel\n");
    printf("\t19bce175 Hrushi Patel\n");
    printf("Press 1 to continue\n>>>");
    int key = 0;
    while (key != 1)
    {
```

```
        scanf("%d", &key);
    }
    // system("cls");
    return;
}
/* </Intro Screen> */

/* <Exit Confirmation Screen> */
char confirm_on_leave()
{
    printf("Are you sure you want to leave?\n");
    printf("Enter Y/y for Yes and another key for No\n>>>");
    char key;
    flush;
    scanf("%c", &key);
    if (key == 'Y' || key == 'y')
    {
        printf("Exiting program...\n");
        return '0';
    }
    else
        return '1';
}
/* </Exit Confirmation Screen> */

/* Function to Clear Array Which Is Used For Taking Input of Event Registered
or Not*/
void clear_array(int *arr)
{
    for (int i = 0; i < 3; i++)
        arr[i] = 0;
}

/*Function to allocate space in heap memory for a node*/
node *newNode(char name_in[], int number_in, int *arr)
{
    node *temp =  (node *)malloc(sizeof(node));
    strcpy(temp->name, name_in);
    temp->number = number_in;
    for (int i = 0; i < 3; i++)
    {
        temp->event_registered[i] = arr[i];
    }
    temp->unique_id = unique_id_in;
    temp->left = temp->right = temp->middle = temp->parent = NULL;
    return temp;
}
```

```
/*Function to insert a entry*/
//Working -
> Goes to left or right according to higher or lower value of key(key is here
name);
//If same node with key (name) is found then the new entry is inserted in a li
nkedlist type structure which will is created
//by the component node_name->middle of node structure
node * insert(node* root, char name_in[], int number_in, int *arr)
{
    if (root == NULL)
        return newNode(name_in, number_in, arr);
    if (strcmpi(name_in, root->name) == 0)
    {
        root->middle = insert(root->middle, name_in, number_in, arr);
        node *temp = root->middle;
        temp->parent = root;
    }
    else if (strcmpi(name_in, root->name) < 0)
    {
        root->left  = insert(root->left, name_in, number_in, arr);
        node *temp = root->left;
        temp->parent = root;
    }
    else if (strcmpi(name_in, root->name) > 0)
    {
        root->right = insert(root->right, name_in, number_in, arr);
        node *temp = root->right;
        temp->parent = root;
    }
    return root;
}

//Given a non-empty binary search tree, return the node with minimum
//key value found in that tree. Note that the entire tree does not
//need to be searched.
node * minValueNode(node* root)
{
    node* current = root;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

//Function to delete entries with multiple names
//Working - It goes deep in the list and gives unique id of each node, and ask
s user to insert
//unique id to delete that node. After deletion is called by user, the other n
ode adjusts accordingly
```

```
//to the TO-BE-
DELETED node position with address present in their structure and after arrang
ements,
//frees the memory of deleted node
void delete_multiple(node* root)
{
    printf("There are many entries of same name.\n");
    node *temp = root;
    while (root != NULL)
    {
        printf("\t%s %d\n", root->name, root->unique_id);
        root = root->middle;
    }
    while (1)
    {
        int flag = 0;
        root = temp;
        printf("Enter unique id to get the entry deleted.\n>>>");
        int id_in;
        flush;
        scanf("%d", &id_in);
        flush;
        while (root != NULL && root->unique_id != id_in)
        {
            root = root->middle;
        }
        if (root == NULL)
        {
            printf("Enter valid id.\n>>>");
            continue;
        }
        else
        {
            if (root == temp)
            {
                root->middle->right = root->right;
                root->middle->left = root->left;
                if (root == global_root)
                {
                    global_root = root->middle;
                }
                else
                {
                    if (root->parent->left == root)
                    {
                        root->parent->left = root->middle;
                    }
                    else if (root->parent->right == root)
```

```
                    {
                        root->parent->right = root->middle;
                    }
                }
                free(root);
            }
            else
            {
                root->parent->middle = root->middle;
                temp = root->middle;
                free(root);
            }
            break;
        }
    }
}
// Generel Function to delete entries
//Working - Explained in line comments
void deleteNode(node *t, char key[])                                //deletenum
ber function  for deleting node from tree
{
    del = 0;
    while (t != NULL)                                               //loop will
 run till last node
    {
        if (strcmpi(key, t->name) == 0 && t-
>middle == NULL)       //if node found in tree
        {
            if (t->left == NULL && t-
>right == NULL)               //if both node don't have both child
            {
                if (t == global_root)                               //if it is
root and alone in tree
                {
                    global_root = NULL;                             //make root
 null
                }
                else if (t->parent-
>left == t)                     //if it is left child of its parent node
                {
                    t->parent-
>left = NULL;                         //make null parent's lchild to null
                }
                else                                                //if it is
right child of parent node
                {
                    t->parent-
>right = NULL;                        //then make parent's right child to null
```

```
                }
                free(t);
            }
            else if (t->left != NULL && t-
>right != NULL)          //if it has both the child
            {
                struct node *e;
                e = t-
>right;                                     //shift t to its right child
                if (e != NULL)                                          //if it is
 not null
                {
                    while (e-
>left != NULL)                          //then goto left most child of t
                    {
                        e = e->left;
                    }
                }

                if (t == global_root)
                {

                    if (t->right == e)
                    {
                        e->left = t->left;
                        e->parent = NULL;
                    }
                    else
                    {
                        e->parent->left = NULL;
                        e->parent = NULL;
                        e->left = t->left;
                        e->right = t->right;
                    }
                    global_root = e;

                }
                else
                {
                    int y = 0;
                    if (e->parent->left == e)
                    {
                        e->parent->left = NULL;
                    }
                    else
                    {
                        e->parent->right = NULL;
                        y = 1;
```

```
                    }
                    e->parent = t->parent;
                    if (t->parent->left == t)
                        t->parent->left = e;
                    else
                        t->parent->right = e;
                    if (y == 1)
                        e->left = t->left;
                    else
                    {
                        e->left = t->left;
                        e->right = t->right;
                    }
                }

                free(t);
                if (e->left)
                    e->left->parent = e;
                if (e->right)
                    e->right->parent = e;

            }
            else                                                    //else part
 if it has only one child
            {
                if (t == global_root)                               //if node f
ound is root
                {
                    if (t-
>left)                                  //if node has left child
                    {
                        global_root = t-
>left;                     //update root to its left child
                    }
                    else                                            //if it has
 right child
                    {
                        global_root = t-
>right;                    //update root to its right child
                    }
                    free(t);
                    global_root->parent = NULL;
                }
                else if (t->parent-
>left == t)                      //if it is not root and it is left child of pa
rent node
                {
```

```
                    if (t-
>left == NULL)                          //if it has right child
                    {
                        t->parent->left = t-
>right;                //make parent's left child to current node's right chil
d
                        t->right->parent = t->parent;
                    }
                    else                                                //if it has
 left child
                    {
                        t->parent->left = t-
>left;                //make parent's left child to current node's left child
                        t->left->parent = t->parent;
                    }
                    free(t);
                }
                else if (t->parent-
>right == t)                    //else part if it is right child of parent nod
e
                {
                    if (t-
>left == NULL)                          //if current node's has right child
                    {
                        t->parent->right = t-
>right;              //then make current nodes right child as parent's richil
d
                        t->right->parent = t->parent;
                    }
                    else
                    {
                        t->parent->right = t-
>left;               //else make it left child of parent node
                        t->left->parent = t->parent;
                    }
                    free(t);
                }
            }
            del = 1;                                                    //make i to
 1 to know that we deleted node
            break;                                                      //break the
 loop after deletion
        }
        else if (strcmpi(key, t->name) == 0 && t->middle != NULL)
        {
            delete_multiple(t);
            del = 1;
            break;
```

```
        }
        else if (strcmpi(key, t-
>name) < 0)                        //if our name is greater then current node'
s name
        {

            t = t-
>left;                                          //and update t to its left ch
ild
        }
        else                                                        //else part
 if it is lesser then current node
        {

            t = t-
>right;                                         //and shift t to its right ch
ild
        }
    }
    if (del == 0)                                                   //if after
completion of loop name not found then print message
        printf("System Message::::%s Not Found.\n", key);

}

//Function to edit data of a entry
void edit(node *root)
{
    int key = -1;
    while (key != 0)
    {
        printf("Enter 1 to edit mobile number.\n");
        printf("Enter 2 to edit events registered.\n");
        printf("Enter 0 if you dont want to edit further.\n>>>");
        scanf("%d", &key);
        if (key > 2)
        {
            printf("Enter valid choice.\n>>>");
            continue;
        }
        if (key == 1)
        {
            printf("Enter new number.\n>>>");
            int number_in;
            scanf("%d", &number_in);
            root->number = number_in;
            printf("System Message::::Mobile number changed sucessfully!\n");
        }
```

```
        else if (key == 2)
        {
            printf("Enter which index number of event to view current status.\
n>>>");
            int event_in;
            scanf("%d", &event_in);
            printf("%d event is ", event_in);
            if (root->event_registered[event_in] == 0)
            {
                printf("unregistered.\n");
            }
            else
            {
                printf("registered.\n");
            }
            printf("Press Y/y to alternate, any other key to skip.\n>>>");
            char c[10];
            flush;
            gets(c);
            flush;
            if (c[0] == 'Y' || c[0] == 'y')
            {
                root->event_registered[event_in] = !root-
>event_registered[event_in];
                printf("System Message::::Alternated sucessfully!\n");
            }
            else
            {
                printf("System Message::::Not Changed!\n");
            }
        }
        else
        {
            break;
        }
    }
}

//In case there are mulitple nodes of same name, then this function allows to
change any of them
//with help of unique id
void edit_multiple(node *root)
{
    printf("There are many enteries of same name.\nEnter unique id to change p
articular information.\n>>>");
    node *temp = root;
    while (root != NULL)
    {
```

```
        printf("%s %d\n", root->name, root->unique_id);
        root = root->middle;
    }
    root = temp;
    int id;
    scanf("%d", &id);
    while (1)
    {
        while (root != NULL && root->unique_id != id)
        {
            root = root->middle;
        }
        if (root == NULL)
        {
            printf("Enter valid id.\n>>>");
            scanf("%d", &id);
        }
        else break;
    }
    edit(root);
}

//Funtion to search a node with particular name pass that node
// to edit or edit_multiple function according to needs
void edit_search(node* root, char name_in[])
{
    if (root == NULL)
        return;
    if (strcmpi(name_in, root->name) > 0)
        edit_search(root->right, name_in);
    else if (strcmpi(root->name, name_in) < 0)
        edit_search(root->left, name_in);
    else if (root->middle != NULL)
        edit_multiple(root);
    else
        edit(root);
}

//General function to search by name
void search(node* root, char name_in[])
{

    if (root == NULL)
        return;
    if (strcmpi(name_in, root->name) > 0)
        search(root->right, name_in);
    else if (strcmpi(root->name, name_in) < 0)
        search(root->left, name_in);
```

```
    else if (root != NULL && strcmpi(root->name, name_in) == 0)
    {
        printf("\t%s %d\n", root->name, root->number);
        ans++;
        search(root->middle, name_in);
    }
}

//General Funtion to Search by Number
void search_by_n(struct node *root, int ns)
{

    if (root != NULL)
    {
        if (root->number == ns)
        {
            printf("\t%s %d\n", root->name, root->number);
            ans++;
        }
        search_by_n(root->left, ns);
        search_by_n(root->middle, ns);
        search_by_n(root->right, ns);
    }
}

//Function which performs in-order traversal and prints data in
// ascending order of name entries
void inorder(node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        inorder(root->middle);
        printf("\t%-10s %-10d %-10d \n", root->name, root->number, root-
>unique_id);
        inorder(root->right);
    }
}

//Fucntion to save data in a file called output.txt
void file_print_all(node *root)
{

    if (root != NULL)
    {
        file_print_all(root->left);
        file_print_all(root->middle);
        fprintf(fptr, "%s %d\n", root->name, root->number);
```

```
        file_print_all(root->right);
    }
}

//Menu Function to simulate all the functions as per user needs
void menu()
{
    node *root = NULL;
    char name_in[50];
    int number_in;
    int event_registered_in[3];
    while (1)
    {
        // Sleep(5000);
        // system("cls");
        printf("---------Menu---------\n");
        printf("Enter 1 to Add New Participant.\n");
        printf("Enter 2 to Remove Participant.\n");
        printf("Enter 3 to Edit Existing Participant.\n");
        printf("Enter 4 to Veiw All Participants.\n");
        printf("Enter 5 to Print All Participants.\n");
        printf("Enter 6 to Search Participants.\n");
        printf("Enter 0 to Exit.\n");
        printf("----------------------\n>>>");

        char key[100];
        scanf("%s", key);
        if (key[0] == '0')
        {
            printf("System Message::::Saving your data before you plan to leav
e!\n...\n");
            fptr = fopen("output.txt", "w");
            file_print_all(root);
            fclose(fptr);
            printf("System Message::::Data saved!\n");
            key[0] = confirm_on_leave();
            if (key[0] == '0' )
                return;
            else
                continue;
        }
        else if (key[0] == '1')
        {
            clear_array(event_registered_in);
            printf("Enter name.\n>>>");
            flush;
            gets(name_in);
            flush;
```

```
            printf("Enter number.\n>>>");
            scanf("%d", &number_in);
            printf("Enter 1 to register in an event, any other key to skip the
 event.\n");
            for (int i = 0; i < 3; i++)
            {
                printf(">>>Event %2d : ", i + 1);
                int tmp;
                scanf("%d", &tmp);
                if (tmp == 1)
                    event_registered_in[i] = 1;
                else
                    event_registered_in[i] = 0;
            }
            if (root == NULL)
            {
                root = insert(root, name_in, number_in, event_registered_in);
                global_root = root;
            }
            else
                insert(root, name_in, number_in, event_registered_in);
            unique_id_in += 1;
        }
        else if (key[0] == '2')
        {
            if (global_root == NULL)                                    //if
 list is empty then print message
                printf("System Message::::Please first enter number, Your list
 is empty.\n");
            else
            {
                printf("Enter a name which you want to delete.\n");
                flush;
                gets(name_in);                                  //user input to f
ind name in list and delete
                flush;
                flush;
                deleteNode(root, name_in);                              //call to
 deletenumber function
                root = global_root;
                if (del != 0)
                    printf("System Message::::Deleted Successfully.\n");
                del = 0;
            }
            continue;

        }
        else if (key[0] == '3')
```

```
        {
            if (global_root == NULL)                                    //if
 list is empty then print message
                printf("System Message:::Please first enter number, Your list
 is empty\n");
            else
            {
                printf("Enter a name which you want to edit\n>>>");
                flush;
                gets(name_in);                              //user input to f
ind name in list and delete
                flush;
                edit_search(root, name_in);
            }
            continue;
        }
        else if (key[0] == '4')
        {
            if (global_root)
            {
                printf("\tName      Phone      Unique ID\n");
                inorder(root);
            }
            else
                printf("System Message:::Event List is empty\n");
        }
        else if (key[0] == '5')
        {
            fptr = fopen("output.txt", "w");
            file_print_all(root);
            fclose(fptr);
        }
        else if (key[0] == '6')
        {
            printf("Enter 1 to search by name.\nEnter 2 to search by number.\n
>>>");
            int x;
            scanf("%d", &x);
            if (x == 1)
            {
                printf("Enter name\n>>>");
                flush;
                gets(name_in);
                search(root, name_in);
                if (ans == 0)
                {
                    printf("System Message:::No matches found\n");
                }
```

```
                else
                {
                    printf("System Message:::Finished viewing results\n");
                    ans = 0;
                }
            }
            else
            {
                printf("Enter number.\n>>>");
                scanf("%d", &number_in);
                search_by_n(root, number_in);
                if (ans == 0)
                {
                    printf("System Message:::No matches found\n");
                }
                else
                {
                    printf("System Message:::Finished viewing results\n");
                    ans = 0;
                }
            }
        }
        else
        {
            printf("System Message::::Please Enter A Valid Choice\n");
            continue;
        }
    }
}

//Function to print message after exiting
void exit_confirm()
{
    printf("Underlying Data Structures\n\tMain - BST\n\tSecondary - Linkedlist
\n");
    printf("Time Complexities of Each Fucntions are below\n");
    printf("\t Insertion: Average-O(Log(n)), Worst-O(n)");
    printf("\t Deletion:  Average-O(Log(n)), Worst-O(n)");
    printf("\t Searching: Average-O(Log(n)), Worst-O(n)");
    printf("\t Searching: Average-O(Log(n)), Worst-O(n)");
    printf("The Worst Cases are the ones such that all users will have same na
me and ultimately linkedlist will be formed.\n");
    printf("The program has ended!\nFor the reference, the data is been stored
 in output.txt file!\n");
}

//Main Functions to call other functions
int main()
```

```
{
    intro();
    menu();
    exit_confirm();
}
```