**DS ASSIGNMENT**

**B.PRITHIRAJ  
B520016**

**COMPUTER ENGINEERING**

*QUESTION 1-*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int update\_condition = 0;

struct BSTnode

{

char word[128], meaning[256];

struct BSTnode \*left, \*right;

};

struct BSTnode \*root = NULL;

struct BSTnode \*createNode(char \*word, char \*meaning)

{

struct BSTnode \*newnode;

newnode = (struct BSTnode \*)malloc(sizeof(struct BSTnode));

strcpy(newnode->word, word);

strcpy(newnode->meaning, meaning);

newnode->left = newnode->right = NULL;

return newnode;

}

void insert(char \*word, char \*meaning)

{

struct BSTnode \*parent = NULL, \*current = NULL, \*newnode = NULL;

int res = 0;

if (!root)

{

root = createNode(word, meaning);

return;

}

for (current = root; current != NULL;

current = (res > 0) ? current->right : current->left)

{

res = strcasecmp(word, current->word);

if (res == 0)

{

printf("Duplicate entry!!\n");

return;

}

parent = current;

}

newnode = createNode(word, meaning);

res > 0 ? (parent->right = newnode) : (parent->left = newnode);

return;

}

void deleteNode(char \*str)

{

struct BSTnode \*parent = NULL, \*current = NULL, \*temp = NULL;

int flag = 0, res = 0;

if (!root)

{

printf("BST is not present!!\n");

return;

}

current = root;

while (1)

{

res = strcasecmp(current->word, str);

if (res == 0)

break;

flag = res;

parent = current;

current = (res > 0) ? current->left : current->right;

if (current == NULL)

return;

}

if (current->right == NULL)

{

if (current == root && current->left == NULL)

{

free(current);

root = NULL;

return;

}

else if (current == root)

{

root = current->left;

free(current);

return;

}

flag > 0 ? (parent->left = current->left) : (parent->right = current->left);

}

else

{

temp = current->right;

if (!temp->left)

{

temp->left = current->left;

if (current == root)

{

root = temp;

free(current);

return;

}

flag > 0 ? (parent->left = temp) : (parent->right = temp);

}

else

{

struct BSTnode \*successor = NULL;

while (1)

{

successor = temp->left;

if (!successor->left)

break;

temp = successor;

}

temp->left = successor->right;

successor->left = current->left;

successor->right = current->right;

if (current == root)

{

root = successor;

free(current);

return;

}

(flag > 0) ? (parent->left = successor) : (parent->right = successor);

}

}

free(current);

return;

}

void findElement(char \*str)

{

struct BSTnode \*temp = NULL;

int flag = 0, res = 0, count = 0;

if (root == NULL)

{

printf("Binary Search Tree is out of station!!\n");

return;

}

temp = root;

while (temp)

{

count++;

if ((res = strcasecmp(temp->word, str)) == 0)

{

if (update\_condition == 0)

{

printf("Word : %s", str);

printf("Meaning: %s", temp->meaning);

if (count == 1)

printf("\nWe found this word in %d comparison\n", count);

else

printf("\nWe found this word in %d comparisons\n", count);

flag = 1;

break;

}

else

{

char new\_meaning[256];

printf("Enter the new meaning to be updated : ");

scanf("%[^\n]", &new\_meaning);

strcpy(temp->meaning, new\_meaning);

printf("The new meaning has been updated.\n");

flag = 1;

break;

}

}

temp = (res > 0) ? temp->left : temp->right;

}

if (!flag)

printf("Search Element not found in Binary Search Tree\n");

return;

}

void inorderTraversal(struct BSTnode \*myNode)

{

if (myNode)

{

inorderTraversal(myNode->left);

printf("Word : %s", myNode->word);

printf("Meaning : %s", myNode->meaning);

printf("\n");

inorderTraversal(myNode->right);

}

return;

}

void update\_the\_meaning(char \*str)

{

}

int main()

{

int ch;

char str[128], meaning[256];

while (1)

{

printf("\n1. Insertion\t2. Deletion\n");

printf("3. Searching\t4. Traversal\n");

printf("5.Update\t6. Exit\nEnter ur choice:");

scanf("%d", &ch);

getchar();

switch (ch)

{

case 1:

printf("Word to insert:");

fgets(str, 100, stdin);

printf("Meaning:");

fgets(meaning, 256, stdin);

insert(str, meaning);

break;

case 2:

printf("Enter the word to delete:");

fgets(str, 100, stdin);

deleteNode(str);

break;

case 3:

printf("Enter the search word:");

fgets(str, 100, stdin);

findElement(str);

break;

case 4:

inorderTraversal(root);

break;

case 5:

update\_condition = 1;

printf("Enter the word whose meaning to be updated:");

fgets(str, 100, stdin);

findElement(str);

update\_condition = 0;

break;

case 6:

exit(0);

default:

printf("You have entered wrong option\n");

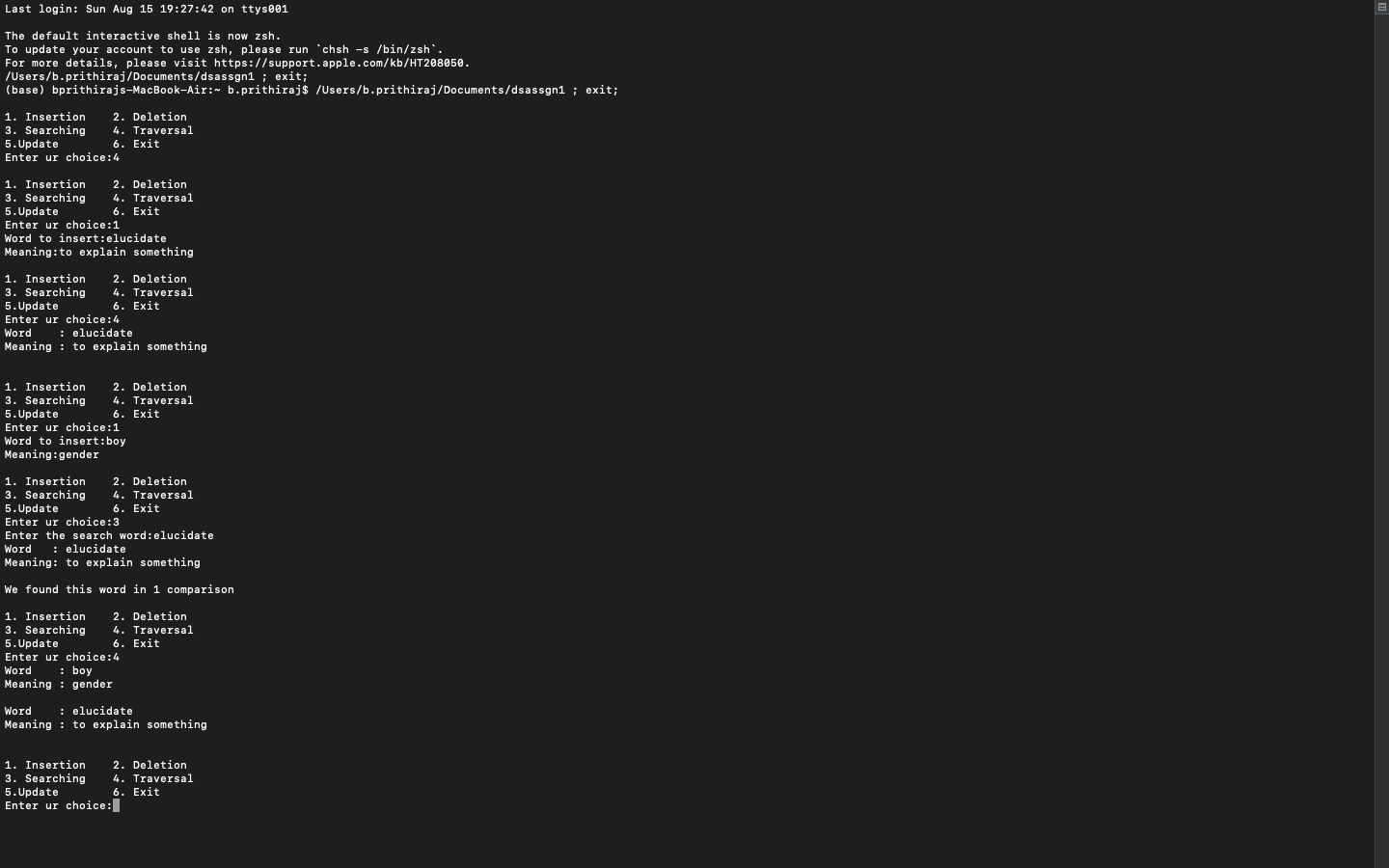
break;

}

}

return 0;

}

**

*OUTPUT-*

Q2:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct node

{

char name[20];

struct node \*next;

struct node \*down;

int flag;

} node;

char ch[20];

int n, i;

node \*head = NULL;

node \*temp = NULL;

node \*t1 = NULL;

node \*t2 = NULL;

node \*create()

{

node \*p = (node \*)malloc(sizeof(node));

p->next = NULL;

p->down = NULL;

p->flag = 0;

printf("\n Enter the name : ");

fflush(stdin);

scanf("%[^\n]", p->name);

return p;

}

void insertb()

{

if (head == NULL)

{

t1 = create();

head = t1;

}

else

{

printf("\n Book exist.\n ");

}

}

void insertc()

{

if (head == NULL)

{

printf("\n There is no book.\n ");

}

else

{

printf("\n How many chapters you want to insert : ");

scanf("%d", &n);

for (i = 0; i < n; i++)

{

t1 = create();

if (head->flag == 0)

{

head->down = t1;

head->flag = 1;

}

else

{

temp = head;

temp = temp->down;

while (temp->next != NULL)

temp = temp->next;

temp->next = t1;

}

}

}

}

void inserts()

{

if (head == NULL)

{

printf("\n There is no book.\n ");

}

else

{

printf("\n Enter the name of chapter on which you want to enter the section : ");

fflush(stdin);

scanf("%[^\n]", ch);

temp = head;

if (temp->flag == 0)

{

printf("\n Their are no chapters on in book.\n ");

}

else

{

temp = temp->down;

while (temp != NULL)

{

if (!strcmp(ch, temp->name))

{

printf("\n How many sections you want to enter : ");

scanf("%d", &n);

for (i = 0; i < n; i++)

{

t1 = create();

if (temp->flag == 0)

{

temp->down = t1;

temp->flag = 1;

printf("\n\*\*\*\*\*\*");

t2 = temp->down;

}

else

{

printf("\n#####");

while (t2->next != NULL)

{

t2 = t2->next;

}

t2->next = t1;

}

}

break;

}

temp = temp->next;

}

}

}

}

void insertss()

{

if (head == NULL)

{

printf("\n There is no book.\n ");

}

else

{

printf("\n Enter the name of chapter on which you want to enter the section : ");

fflush(stdin);

scanf("%[^\n]", ch);

temp = head;

if (temp->flag == 0)

{

printf("\n their are no chapters on in book.\n ");

}

else

{

temp = temp->down;

while (temp != NULL)

{

if (!strcmp(ch, temp->name))

{

printf("\n Enter name of section in which you want to enter the sub section : ");

fflush(stdin);

scanf("%[^\n]", ch);

if (temp->flag == 0)

{

printf("\n Their are no sections.\n ");

}

else

{

temp = temp->down;

while (temp != NULL)

{

if (!strcmp(ch, temp->name))

{

printf("\n How many subsections you want to enter : ");

scanf("%d", &n);

for (i = 0; i < n; i++)

{

t1 = create();

if (temp->flag == 0)

{

temp->down = t1;

temp->flag = 1;

printf("\n\*\*\*\*\*\*");

t2 = temp->down;

}

else

{

printf("\n#####");

while (t2->next != NULL)

{

t2 = t2->next;

}

t2->next = t1;

}

}

break;

}

temp = temp->next;

}

}

}

temp = temp->next;

}

}

}

}

void displayb()

{

if (head == NULL)

{

printf("\n Book not exist.\n ");

}

else

{

temp = head;

printf("\n NAME OF BOOK: %s", temp->name);

if (temp->flag == 1)

{

temp = temp->down;

while (temp != NULL)

{

printf("\n\t\tNAME OF CHAPTER: %s", temp->name);

t1 = temp;

if (t1->flag == 1)

{

t1 = t1->down;

while (t1 != NULL)

{

printf("\n\t\t\t\tNAME OF SECTION: %s", t1->name);

t2 = t1;

if (t2->flag == 1)

{

t2 = t2->down;

while (t2 != NULL)

{

printf("\n\t\t\t\t\t\tNAME OF SUBSECTION: %s", t2->name);

t2 = t2->next;

}

}

t1 = t1->next;

}

}

temp = temp->next;

}

}

}

}

int main()

{

int x;

while (1)

{

printf("\n\n ENTER YOUR CHOICE.\n");

printf("\n 1.INSERT BOOK.\n");

printf("\n 2.INSERT CHAPTER.\n");

printf("\n 3.INSERT SECTION.\n");

printf("\n 4.INSERT SUBSECTION.\n");

printf("\n 5.DISPLAY BOOK.\n");

printf("\n 6.EXIT.\n ");

scanf("%d", &x);

switch (x)

{

case 1:

insertb();

break;

case 2:

insertc();

break;

case 3:

inserts();

break;

case 4:

insertss();

break;

case 5:

displayb();

break;

case 6:

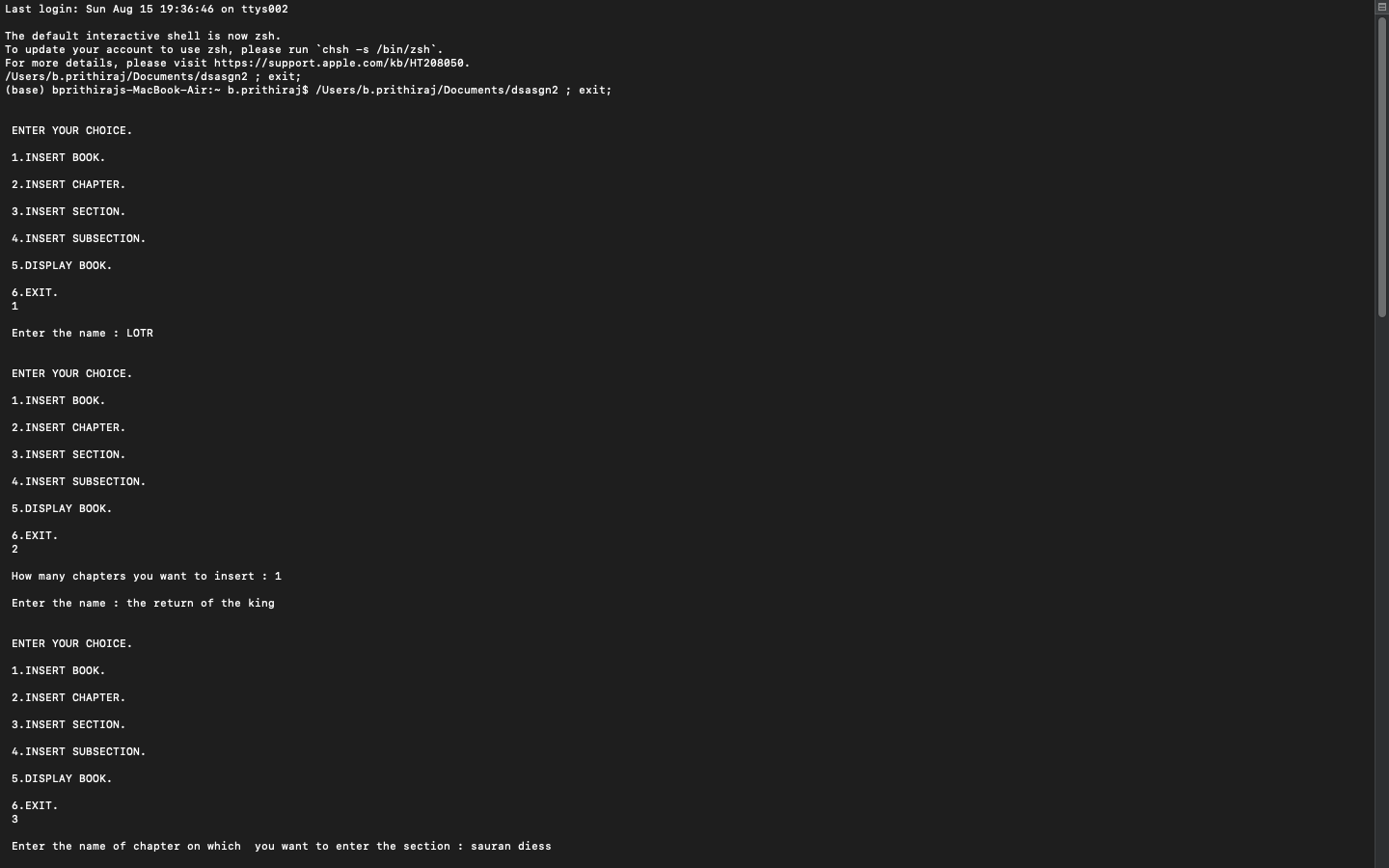
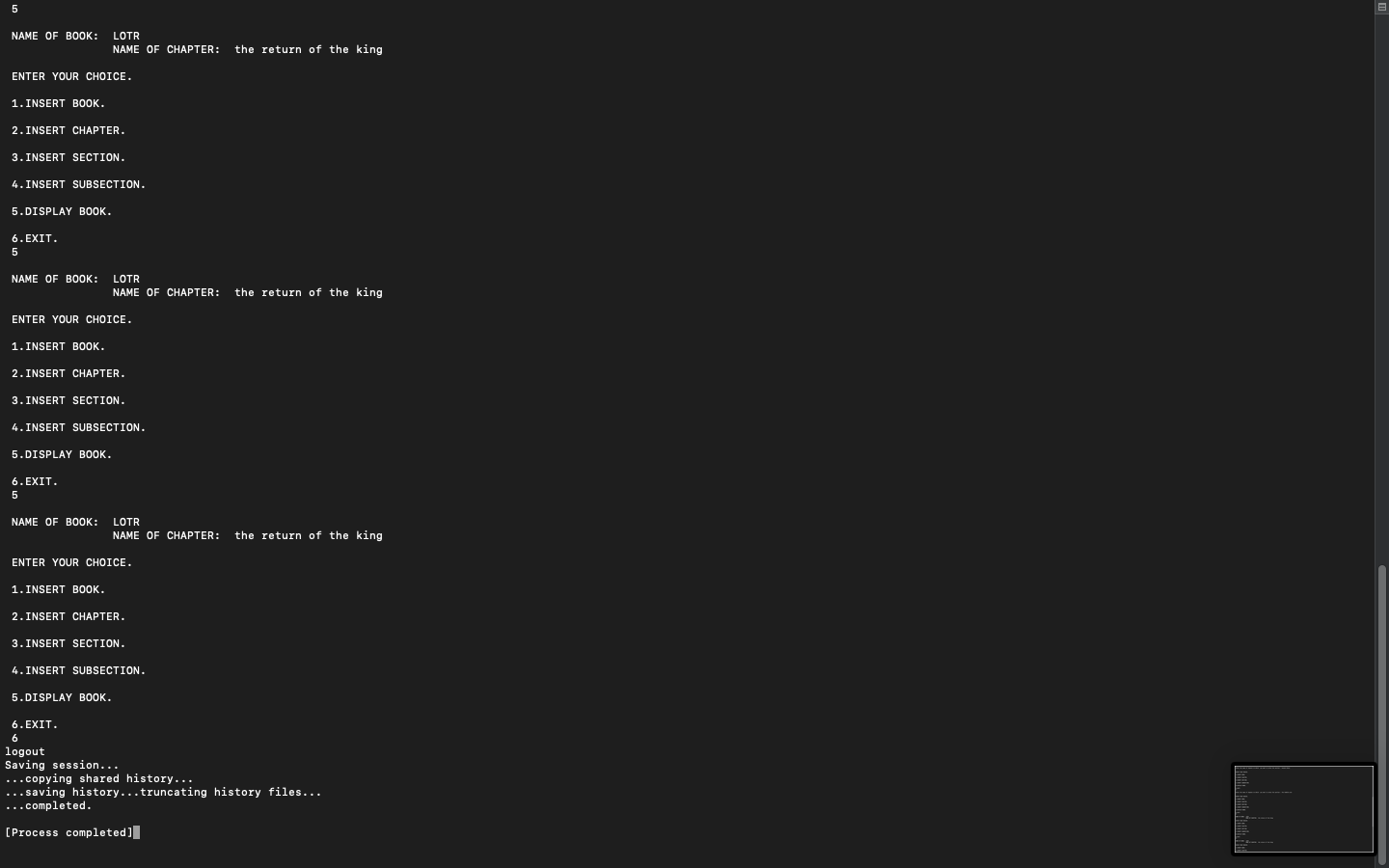
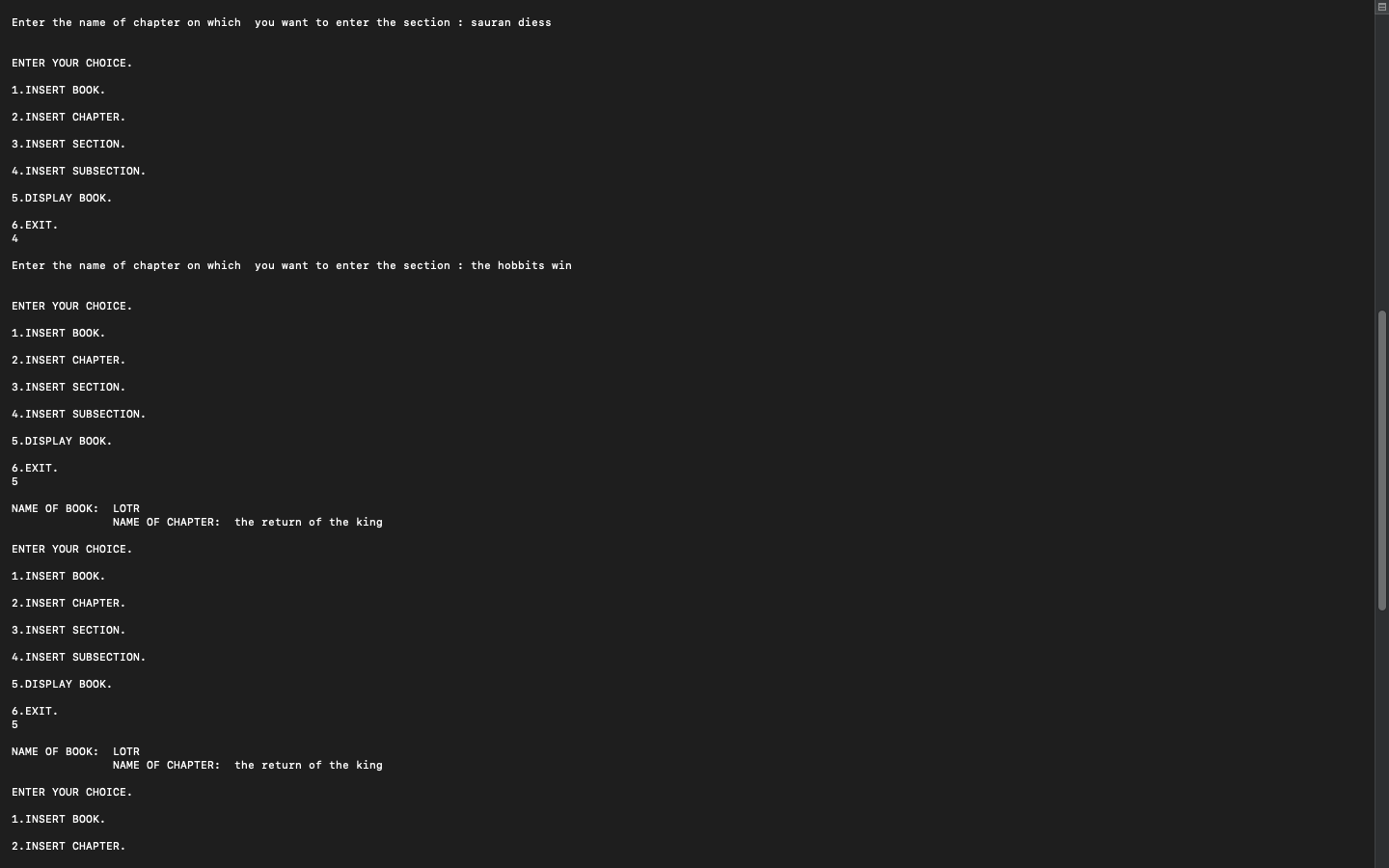
exit(0);

}

}

return 0;

}

*OUTPUT*-

Q3:

#include <stdio.h>

#include <stdlib.h>

#define \_for(i, a, b, c) for (int i = a; i <= b; i += c)

#define MAX 10

int n = 6;

char city[MAX][MAX] = {"CITY", "LUCKNOW", "MUMBAI", "PURI", "WB", "KANPUR", "BBSR"};

void print\_city()

{

\_for(i, 1, n, 1)

{

printf("%d.%s\n", i, city[i]);

}

}

void add\_flight(int graph[MAX][MAX])

{

printf("\nChoose the CITY of DEPARTURE\n");

print\_city();

int x = 0, y = 0, cost = 0;

scanf("%d", &x);

printf("\nChoose the CITY of ARRIVAL\n");

print\_city();

scanf("%d", &y);

printf("\nCOST : ");

scanf("%d", &cost);

graph[x][y] = cost;

printf("Added Sucessfully\n\n");

}

void incoming\_fl(int graph[MAX][MAX], int num)

{

printf("\n++++----++++----++++----++++----++++\n");

\_for(i, 1, n, 1)

{

if (graph[i][num] != 0)

{

printf("%s -> %s\tCOST : %d\n", city[i], city[num], graph[i][num]);

}

}

printf("++++----++++----++++----++++----++++\n");

}

void outgoing\_fl(int graph[MAX][MAX], int num)

{

printf("\n++++----++++----++++----++++----++++\n");

\_for(i, 1, n, 1)

{

if (graph[num][i] != 0)

{

printf("%s -> %s\tCOST : %d\n", city[num], city[i], graph[num][i]);

}

}

printf("++++----++++----++++----++++----++++\n");

}

void add\_city()

{

printf("\n++++----++++----++++----++++----++++\n");

printf("CIty : ");

scanf("%s", city[++n]);

printf("++++----++++----++++----++++----++++\n");

printf("Added Sucessfully\n\n");

}

void city\_details(int graph[MAX][MAX])

{

printf("\nChoose the CITY\n");

print\_city();

int num;

scanf("%d", &num);

int temp = 0;

\_for(i, 1, n, 1)

{

if (graph[i][num] != 0)

{

temp++;

}

}

printf("\n++++----++++----++++----++++----++++\n");

printf("Number of Incoming Flights : %d\n", temp);

temp = 0;

\_for(i, 1, n, 1)

{

if (graph[num][i] != 0)

{

temp++;

}

}

printf("Number of Outgoing Flights : %d\n", temp);

printf("++++----++++----++++----++++----++++\n");

printf("\n1.See all Incoming Flights\n2.See all Outgoing Flight\n3.See all Flights\n4.Continue\n");

scanf("%d", &temp);

if (temp == 1)

{

incoming\_fl(graph, num);

}

else if (temp == 2)

{

outgoing\_fl(graph, num);

}

else if (temp == 3)

{

incoming\_fl(graph, num);

outgoing\_fl(graph, num);

}

}

int main()

{

int graph[MAX][MAX] = {0};

int ch = 0;

while (ch != 9)

{

printf("1.Add City\n2.Add Flight\n3.See City Details\n9.Exit\n");

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

add\_city();

break;

case 2:

add\_flight(graph);

break;

case 3:

city\_details(graph);

break;

case 9:

printf("THANK YOU");

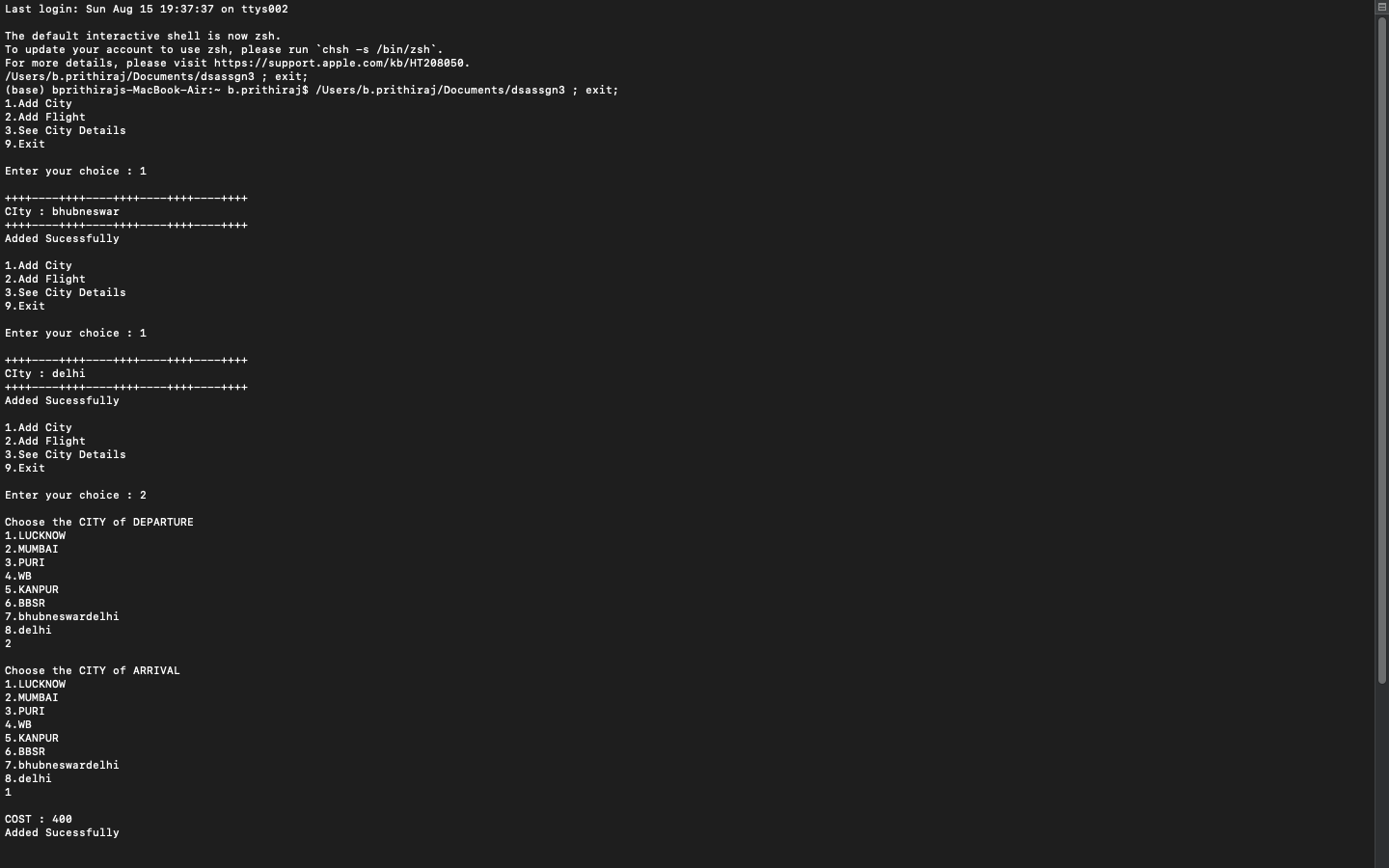
return 0;

default:

printf("Wrong Choice\nPlease try again...\n");

}

}

return 0;

}

*OUTPUT-*

*QUESTION 4-*

#include <stdlib.h>

#include <stdio.h>

#include <ctype.h>

const int MAX = 50;

typedef struct node

{

char data;

struct node \*left, \*right;

} node;

int push(node \*cnode, node \*info[], int top)

{

top = top + 1;

info[top] = cnode;

return top;

}

node \*Top(node \*info[], int top)

{

return info[top];

}

int empty(int top)

{

if (top == -1)

return 1;

else

return 0;

}

int pop(node \*info[], int top)

{

if (!empty(top))

{

return top;

}

return -1;

}

int isFull(int top)

{

if (top == MAX - 1)

return 1;

else

return 0;

}

//==================================================

node \*root;

int priority(char ch)

{

switch (ch)

{

case '+':

case '-':

return 0;

break;

case '\*':

case '/':

return 1;

break;

case '^':

return 2;

break;

}

return -1;

}

void postorderNon()

{

node \*ptr = root;

node \*info[MAX];

int top = -1;

int arr[MAX], i = -1, flag;

LOOP:

while (ptr != NULL)

{

top = push(ptr, info, top);

arr[++i] = 0;

if (ptr->right != NULL)

{

top = push(ptr->right, info, top);

arr[++i] = 1;

}

ptr = ptr->left;

}

if (pop(info, top) != -1)

{

ptr = info[pop(info, top)];

top--;

}

else

{

ptr = NULL;

}

flag = arr[i--];

while (flag == 0 && i >= 0)

{

printf(" %c", ptr->data);

if (pop(info, top) != -1)

{

ptr = info[pop(info, top)];

top--;

}

else

{

ptr = NULL;

}

flag = arr[i--];

}

if (flag == 1 && i >= 0)

{

goto LOOP;

}

printf(" %c", ptr->data);

}

void inorder\_rec(node \*rnode)

{

if (rnode)

{

inorder\_rec(rnode->left);

printf(" %c", rnode->data);

inorder\_rec(rnode->right);

}

}

void create(char str[])

{

node \*info1[MAX];

int top1 = -1;

node \*info2[MAX];

int top2 = -1;

int i = 0;

char ch;

while (str[i] != '\0')

{

ch = str[i];

if (isalpha(ch))

{

node \*temp = (node \*)malloc(sizeof(node));

temp->left = NULL;

temp->right = NULL;

temp->data = ch;

top1 = push(temp, info1, top1);

}

else

{

if (empty(top2))

{

node \*temp = (node \*)malloc(sizeof(node));

temp->left = temp->right = NULL;

temp->data = ch;

top2 = push(temp, info2, top2);

}

else if (priority(ch) > priority(Top(info2, top2)->data))

{

node \*temp = (node \*)malloc(sizeof(node));

temp->left = temp->right = NULL;

temp->data = ch;

top2 = push(temp, info2, top2);

}

else

{

while (!empty(top2) && priority(ch) <= priority(Top(info2, top2)->data))

{

node \*op;

node \*rchild;

node \*lchild;

if (pop(info2, top2) != -1)

{

op = info2[pop(info2, top2)];

top2--;

}

else

{

op = NULL;

}

if (pop(info1, top1) != -1)

{

rchild = info1[pop(info1, top1)];

top1--;

}

else

{

rchild = NULL;

}

if (pop(info1, top1) != -1)

{

lchild = info1[pop(info1, top1)];

top1--;

}

else

{

lchild = NULL;

}

op->right = rchild;

op->left = lchild;

top1 = push(op, info1, top1);

}

node \*x = (node \*)malloc(sizeof(node));

x->data = ch;

x->left = x->right = NULL;

top2 = push(x, info2, top2);

}

}

i++;

}

while (!empty(top2))

{

node \*op;

node \*rchild;

node \*lchild;

if (pop(info2, top2) != -1)

{

op = info2[pop(info2, top2)];

top2--;

}

else

{

op = NULL;

}

if (pop(info1, top1) != -1)

{

rchild = info1[pop(info1, top1)];

top1--;

}

else

{

rchild = NULL;

}

if (pop(info1, top1) != -1)

{

lchild = info1[pop(info1, top1)];

top1--;

}

else

{

lchild = NULL;

}

op->right = rchild;

op->left = lchild;

top1 = push(op, info1, top1);

}

if (pop(info1, top1) != -1)

{

root = info1[pop(info1, top1)];

top1--;

}

else

{

root = NULL;

}

}

void inorder()

{

inorder\_rec(root);

}

int main()

{

root = NULL;

printf("\n");

char exp[] = "a-b\*c-d/e+f";

printf("\nOriginal Expression: %s", exp);

create(exp);

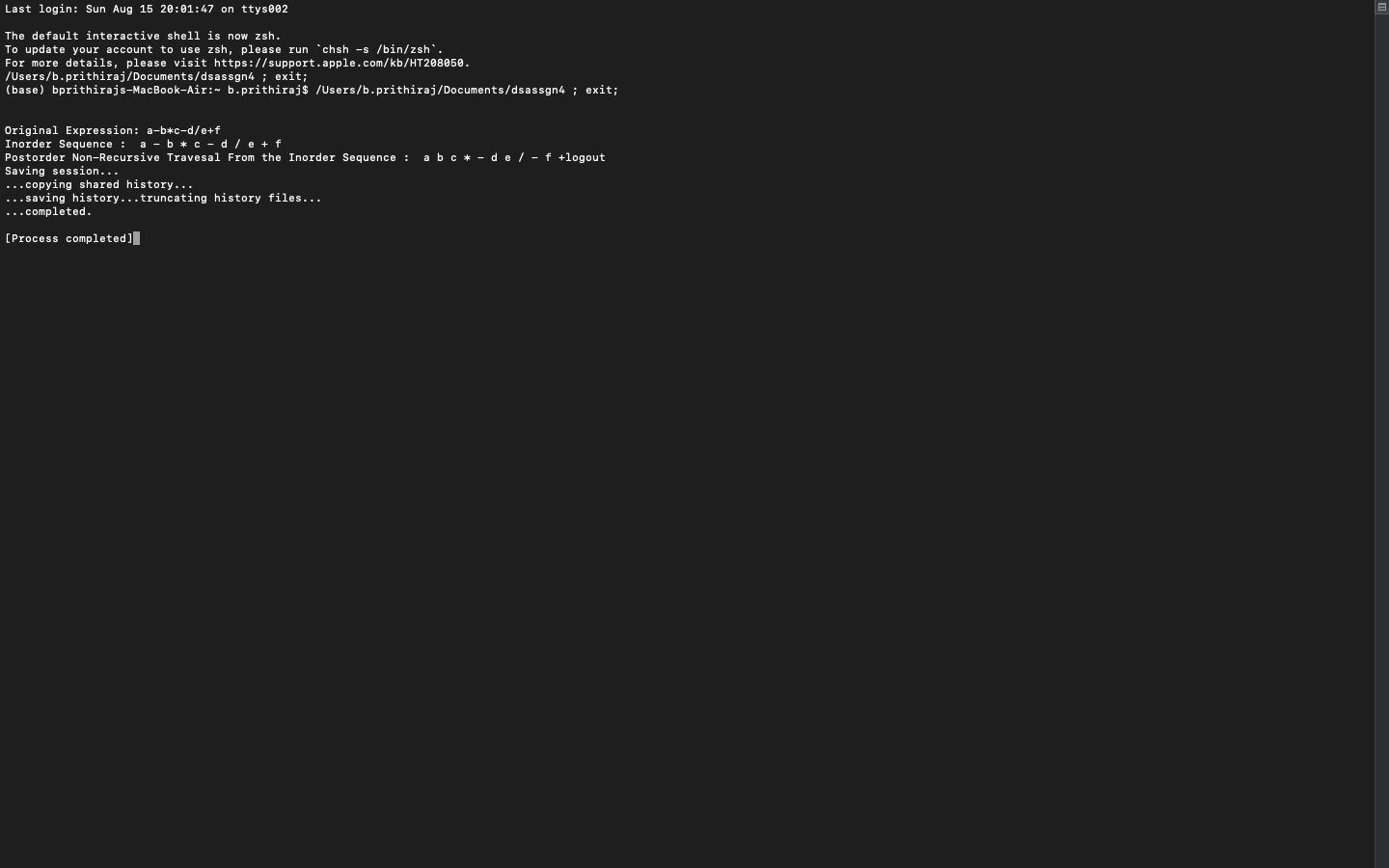
printf("\nInorder Sequence : ");

inorder();

printf("\nPostorder Non-Recursive Travesal From the Inorder Sequence : ");

postorderNon();

return 0;

}

*OUTPUT-*

*SUBMITTED BY-*

*B.PRITHIRAJ*

*B520016*

*COMPUTER ENGINEERING*