/\*

\* C Program to Construct a Binary Search Tree and perform deletion, inorder traversal on it

\*/

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

#include<conio.h>

#include<windows.h>

struct Parking

{

char value[100]; //Value of the car number

struct Parking\* l; //Left pointer of the tree

struct Parking\* r; //Right pointer of the tree

}\*root = NULL,\*temp = NULL,\*t2,\*t1; /\*Root is the starting point of the tree and temp is a temporary pointer the tree to

continue.\*/

//Pointers t1 and t2 are used for

//t1 is used to delete or find leaf node with 0 children or 1 child(left or right)

//t2 is used to delete 2 children in the tree and for finding smallest and largest

//element in tree

void insert(); //To insert a node into the tree

void create(); //To create a node

void search(struct Parking\*t); //To search for another node position if the root node is present

void display(struct Parking\*t); //For traversal of the nodes

void delete(); //To delete a node form the tree

void search1(struct Parking\*t,int data); //To search for the position of the desired root to be deleted

void delete1(struct Parking\*t); //To delete the node it can be of 0,1 or 2 children

int smallest(struct Parking\*t); //To calculate the smallest node on the left

int largest(struct Parking\*t); //To calculate the largest node on the right

void count\_Cars(); //To count the number of cars present in the Parking Lot

struct node

{

char info[100]; //The number of the car

struct node \*ptr; //The pointer pointing o the next element in the linked list

}\*front,\*rear,\*TEMP,\*front1; //The front pointer represents the car at the starting of the queue

//The rear pointer represents the car at the end of the queue

//TEMP is the pointer used to make

//The front1 is used for dequeue cars

void frontelement(); //The car present at the first position in the queue

void enq(char data[100]); //The insertion of the cars in the queue

void deq(); //The deletion of the cars in the queue

void empty(); //Checking for the queue is empty or not

void searching1(char data); //To insert car into the tree from the queue

void create11(); //To start the queue Linked list

void queuesize(); //To see the number of cars present in the queue

int flag=1,i;

int c=0,k = 97,k1=1,k11=1,h=0;

int qs=0;

int j=0;

int count = 0;

char s[100],s1[100];

char e[100],f[10];

void main()

{

char pass[10],password[10]="om123";

int i=0;

system("color 4f"); //The following is a predefined function of the windows.h library which is used to impart

// color to the console input/output

//The number can be used to specify the desired

printf("\n\n\t\tEnter the password to login:");

scanf("%s",pass);

if(strcmp(pass,password)==0)

{

printf("\n\nPassword Match!\nLOADING");

Sleep(1000);

printf("\xDB\xDB\xDB\xDB"); //This is a unicode operation that I have taken from net to that helps in loading!!

Sleep(1000); //This delays the time interval

printf("\xDB\xDB\xDB\xDB");

Sleep(1000);

printf("\xDB\xDB\xDB\xDB");

system("cls");

start();

}

else

{

printf("\n\nWrong Password!");

exit(0);

}

}

void start()

{

//printf("\n---CAR PARKING IN INDIA---");

printf("\nFirst the Parking lot is created!!!");

int ch;

while(1)

{ system("color 0f");

printf("\n---CAR PARKING IN INDIA ---");

printf("\n1 - Insert a car into the Parking lot\n");

printf("2 - Removal of a car from the Parking lot\n");

printf("3 - Traversal of the Parking Lot\n");

printf("4 - Number of cars in the Parking Lot\n");

printf("5 - Go to Queue as the parking lot is ready \n");

printf("6 - Exit\n");

printf("\nEnter your choice : ");

scanf("%d",&ch);

system("cls");

switch(ch)

{

case 1:

if(qs==0)

{

insert();

}

else

{

printf("The size of the car parking system is being made fixed.");

}

break;

case 2:

if(c!=0)

delete();

if(c<=0)

{

free(root);

printf("\nNo cars in the parking lot to display are present!!");

}

break;

case 3:

display(root);

break;

case 4:

count\_Cars();

break;

case 5:

qs++;

Queue\_System();

break;

case 6:

printf("Thank you for visiting");

exit(0);

default:

printf("Wrong choice, Please enter correct choice : ");

break;

}

}

}

void Queue\_System()

{

printf("\nNow the parking lot is occupied and this considered to be the size of the parking lot.");

printf("\nThe bulking of cars outside of the parking lot starts,i.e, the queue operation is started.\n");

int check,e,ch;

char no[100];

while (1)

{

printf("\n 1 - Enter the cars present in the queue");

printf("\n 2 - To check the car present at the first position ");

printf("\n 3 - To check the number of cars present in the queue");

printf("\n 4 - Go back to the BST ");

printf("\n Enter choice : ");

scanf("%d", &ch);

system("cls");

switch (ch)

{

case 1:

printf("Enter the car number : ");

scanf("%s", &no);

j++;

enq(no);

break;

case 2:

frontelement();

break;

case 3:

queuesize();

break;

case 4:

for(k=0;k<j;k++)

deq();

start();

j=0;

count=0;

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

//The part belongs to insertion of the car

/\* To insert a node in the tree \*/

void insert()

{

create();

if(root == NULL)

root = temp;

else

search(root);

}

/\* To create a node \*/

void create()

{

char data[100];

if(qs==0)

{

strcpy(e,"a");

printf("\nEnter the car number that has arrived : ");

scanf("%s",&data);

if((strcmp(data,"space")==0)||(strcmp(data,"Space")==0))

{

f[0] = (char)k;

f[1] = '\0';

strcpy(e,f);

strcat(data,e);

k=k+1;

--c;

}

c++;

temp =(struct Parking\*)malloc(sizeof(struct Parking));

strcpy(temp->value,data);

temp->l = temp->r = NULL;

system("cls");

}

else

{

printf("\nEnter the car number that has arrived again please : ");

scanf("%s",&data);

strcpy(s1,data);

if((strcmp(data,"space")==0)||(strcmp(data,"Space")==0))

{

f[0] = (char)k;

f[1] = '\0';

strcpy(e,f);

strcat(data,e);

k=k+1;

--c;

}

c++;

temp =(struct Parking\*)malloc(sizeof(struct Parking));

strcpy(temp->value,data);

temp->l = temp->r = NULL;

system("cls");

}

}

/\* Function to search the appropriate position to insert the new node \*/

void search(struct Parking\*t)

{

if((strcmp(temp->value,t->value)>0)&&(t->r != NULL))/\* value more than root node value insert at right \*/

search(t->r);

else if((strcmp(temp->value,t->value)>0)&&(t->r == NULL))

t->r = temp;

else if((strcmp(temp->value,t->value)<0)&&(t->l != NULL))/\* value less than root node value insert at left \*/

search(t->l);

else if((strcmp(temp->value,t->value)<0)&&(t->l == NULL))

t->l=temp;

}

/\* recursive function to perform inorder traversal of tree \*/

void display(struct Parking\*t)

{

if(root == NULL)

{

printf("No cars are present in the parking lot.");

return;

}

if(t->l != NULL)

display(t->l);

printf("%s -> ", t->value);

if(t->r != NULL)

display(t->r);

}

/\* To check for the deleted node \*/

void delete()

{

char data[100];

if(root == NULL)

{

printf("No cars are present in the parking lot.");

return;

}

else

{

printf("Enter the car number that will depart : ");

scanf("%s",&data);

system("cls");

t1 = root;

t2 = root;

search1(root, data);

}

}

/\* Search for the appropriate position to insert the new node \*/

void search1(struct Parking\*t,int data)

{

if(qs==0)

{

if(strcmp(data,t->value)>0)

{

k11=1;

if(t->r==NULL)

{

printf("\nNo such car is present in the parking lot.");

start();

}

else

{

t1 = t;

search1(t->r, data);

}

}

else if(strcmp(data,t->value)<0)

{

k11=1;

if(t->l==NULL)

{

printf("\nNo such car is present in the parking lot.");

start();

}

else

{

t1 = t;

search1(t->l, data);

}

}

else

{

k11=0;

printf("The car which left the parking lot is : %s",data);

delete1(t);

}

}

else

{

if(strcmp(data,t->value)>0)

{

k11=1;

if(t->r==NULL)

{

printf("\nNo such car is present in the parking lot.");

start();

}

else

{

t1 = t;

search1(t->r, data);

}

}

else if(strcmp(data,t->value)<0)

{

k11=1;

if(t->l==NULL)

{

printf("\nNo such car is present in the parking lot.");

start();

}

else

{

t1 = t;

search1(t->l, data);

}

}

else

{

k11=0;

printf("The car which left the parking lot is : %s",data);

strcpy(data,"space");

if((strcmp(data,"space")==0)||(strcmp(data,"Space")==0))

{

f[0] = (char)k;

f[1] = '\0';

strcpy(e,f);

strcat(data,e);

k=k+1;

}

strcpy(t->value,data);

c--;

}

}

}

/\* To delete a node \*/

void delete1(struct Parking\*t)

{

char k[100];

/\* To delete leaf node \*/

if((t->l == NULL)&&(t->r == NULL))

{

if(t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

c--;

return;

}

/\* To delete node having one left hand child \*/

else if((t->r == NULL))

{

if(t1 == t)

{

root = t->l;

t1 = root;

}

else if(t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

c--;

return;

}

/\* To delete node having right hand child \*/

else if(t->l == NULL)

{

if(t1 == t)

{

root = t->r;

t1 = root;

}

else if(t1->r == t)

{

t1->r = t->r;

}

else

{

t1->l = t->r;

}

t == NULL;

free(t);

c--;

return;

}

/\* To delete node having two child \*/

else if((t->l != NULL)&&(t->r != NULL))

{

t2 = root;

if(t->r != NULL)

{

strcpy(k,smallest(t->r));

flag =1;

}

else

{

strcpy(k,largest(t->l));

flag =2;

}

search1(root, k);

strcpy(t->value,k);

}

}

/\* To find the smallest element in the right sub tree \*/

int smallest(struct Parking\*t)

{

t2 = t;

if(t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return(t->value);

}

/\* To find the largest element in the left sub tree \*/

int largest(struct Parking\*t)

{

if(t->r != NULL)

{

t2 = t;

return(largest(t->r));

}

else

return(t->value);

}

/\*Counts the number of cars in the parking lot. \*/

void count\_Cars()

{

printf("\nThe number of cars present in the parking lot is : %d",c);

}

/\* Using queue to store the in coming cars into the parking place Now!!!\*/

/\*

\* C Program to Implement Queue Data Structure using Linked List

\*/

/\* Create an empty queue \*/

void create11()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

printf("\n Queue size : %d", count);

}

/\* Enqueing the queue \*/

void enq(char data[100])

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

strcpy(rear->info,data);

front = rear;

searching1(rear->info);

}

else

{

TEMP=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = TEMP;

strcpy(TEMP->info,data);

TEMP->ptr = NULL;

rear = TEMP;

searching1(TEMP->info);

}

count++;

}

/\* Dequeing the queue \*/

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %s", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %s", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

/\* Returns the front element of queue \*/

void frontelement()

{

if ((front != NULL) && (rear != NULL))

printf("Front element : %s ",(front->info));

else

printf("\nNo front element in the queue as it is empty.");

}

/\* This is used to insert the car from queue into the tree \*/

void searching1(char data)

{

insert();

}