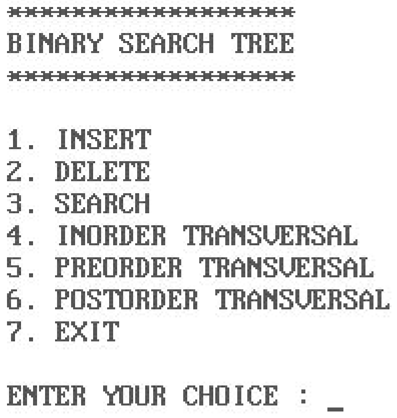
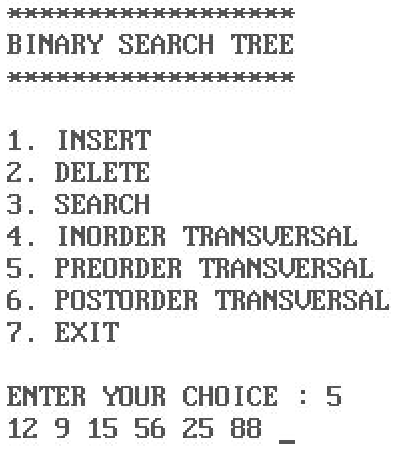
BINARY SEARCH TREE

```
/*
**********************
* BINARY SEARCH TREE *
**********************
*/

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node {
           int data;
           struct node *lchild;
           struct node *rchild;
           } * root;

void postorder(struct node *temp) {
 if(root == NULL) {
     printf("TREE IS EMPTY!");
     return;
 }
 if(temp!=NULL) {
     postorder(temp->lchild);
     postorder(temp->rchild);
     printf("%d ",temp->data);
 }
}

void preorder(struct node *temp) {
 if(root == NULL) {
     printf("TREE IS EMPTY!");
     return;
 }
 if(temp!=NULL) {
     printf("%d ",temp->data);
     preorder(temp->lchild);
     preorder(temp->rchild);
 }
}


void inorder(struct node *temp) {
 if(root == NULL) {
     printf("TREE IS EMPTY!");
     return;
 }
 if(temp!=NULL) {
     inorder(temp->lchild);
     printf("%d ",temp->data);
     inorder(temp->rchild);
 }
}

void find(int data, struct node **loc, struct node **par) {
 struct node *ptr, *ptrpar;
 if(root==NULL) {
     *loc = NULL;
     *par = NULL;
     return;
 }
 if(data == root->data) {
     *loc = root;
     *par = NULL;
     return;
 }
 ptr = root;
 ptrpar = NULL;
 while(ptr!=NULL) {
     if(data == ptr->data) {
     *loc = ptr;
     *par = ptrpar;
     return;
 }
 ptrpar = ptr;
 if(data < ptr->data) {
     ptr = ptr->lchild;
 } else {
      ptr = ptr->rchild;
     }
 }
 *loc = NULL;
 *par = ptrpar;
}

void case_a(struct node *par,struct node *loc) {
 if(loc == root) {
     root = NULL;
     return;
 }
 if(loc == par->lchild) {
     par->lchild = NULL;
 } else {
     par->rchild = NULL;
 }
}

void case_b(struct node *par,struct node *loc) {
 struct node *child;
 if(loc->lchild!=NULL) {
     child = loc->lchild;
 } else {
          child = loc->rchild;
 }
 if(loc == root) {
     root = child;
     return;
 }

 if(loc == par->lchild) {
     par->lchild = child;
 } else {
          par->rchild = child;
 }
}

void case_c(struct node *par,struct node *loc) {
 struct node *ptr,*ptrpar,*suc,*parsuc;
 ptr = loc->rchild;
 ptrpar = loc;
 while(ptr->lchild!=NULL){
     ptrpar = ptr;
     ptr = ptr->lchild;
 }
 suc = ptr;
 parsuc = ptrpar;
 if(ptr->lchild == NULL && ptr->rchild == NULL) {
     case_a(ptrpar,ptr);
 } else {
          case_b(ptrpar,ptr);
 }
 if(loc == root) {
     root = suc;
 } else {
          if(loc == par->lchild) {
           par->lchild = suc;
          } else {
                par->rchild = suc;
          }
 }
 suc->lchild = loc->lchild;
 suc->rchild = loc->rchild;
}

void deletef() {
 int data;
 struct node *parent,*location;
 if(root == NULL) {
     printf("TREE IS EMPTY!");
     return;
 }
 printf("ENTER DATA TO BE DELETED : ");
 scanf("%d",&data);
 find(data,&location,&parent);
 if(location == NULL) {
     printf("DATA NOT FOUND!");
     return;
 }
 if(location->lchild == NULL && location->rchild == NULL) {
     case_a(parent,location);
     return;
 }
 if(location->lchild != NULL && location->rchild == NULL) {
     case_b(parent,location);
     return;
 }
 if(location->lchild == NULL && location->rchild != NULL) {
     case_b(parent,location);
     return;
 }
 case_c(parent,location);
 }

void insert() {
 int data;
 struct node *location,*parent,*temp;
 printf("ENTER A NUMBER :");
 scanf("%d",&data);
 find(data,&location,&parent);
 if(location != NULL) {
     printf("DATA ALREADY EXISTS!");
     return;
 }
 temp = malloc(sizeof(struct node));
 temp->data = data;
 temp->rchild = NULL;
 temp->lchild = NULL;
 if(parent == NULL) {
     root = temp;
     return;
 }
 if(data < parent->data) {
     parent->lchild = temp;
 } else {
          parent->rchild = temp;
 }
}

void search(int data, struct node * temp) {
 if(temp == NULL) {
     printf("DATA NOT FOUND");
     return;
 }
 if(data > temp -> data) {
     search(data, temp -> rchild);
 } else if(data < temp -> data) {
     search(data, temp -> lchild);
 } else {
          printf("DATA FOUND!");
 }
}

void main() {
 int ch, num;
 while(1) {
     clrscr();
     printf("******************\n");
     printf("BINARY SEARCH TREE\n");
     printf("******************\n\n");
     printf("1. INSERT\n");
     printf("2. DELETE\n");
     printf("3. SEARCH\n");
     printf("4. INORDER TRANSVERSAL\n");
     printf("5. PREORDER TRANSVERSAL\n");
     printf("6. POSTORDER TRANSVERSAL\n");
     printf("7. EXIT\n");
     printf("\nENTER YOUR CHOICE : ");
     scanf("%d",&ch);
     switch(ch) {
      case 1  : insert();
                break;
      case 2  : deletef();
                break;
      case 3  : printf("ENTER DATA TO BE SEARCHED : ");
                scanf("%d",&num);
                search(num,root);
                break;
      case 4  : inorder(root);
                break;
      case 5  : preorder(root);
                break;
      case 6  : postorder(root);
                break;
      case 7  : exit(0);
      default : printf("WRONG CHOICE!");
     }
 getch();
 }
}
```

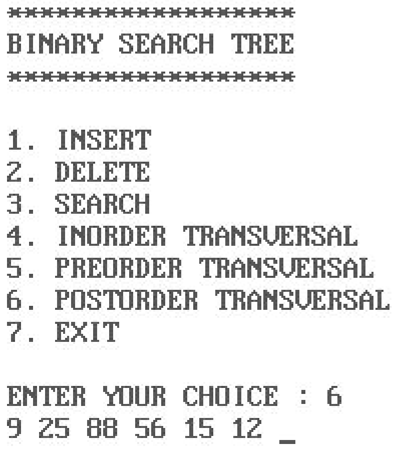
OUTPUT



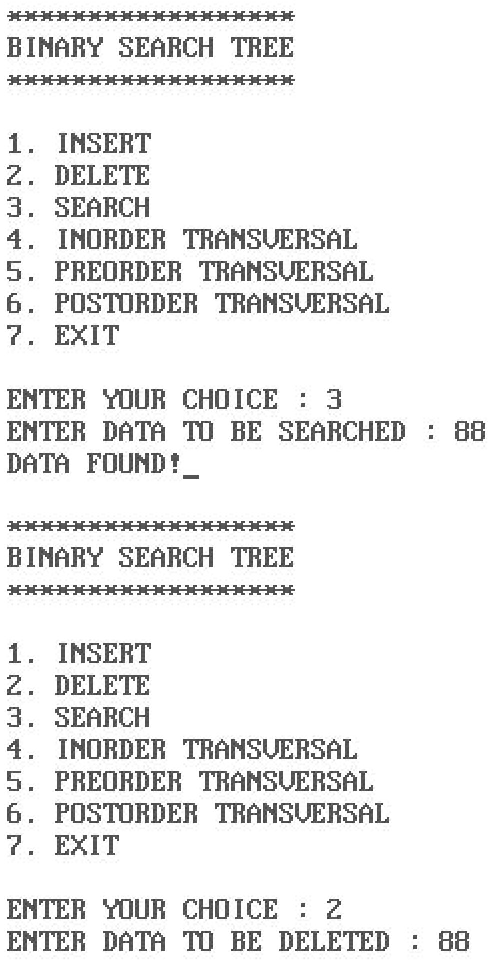
```
******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : _
```



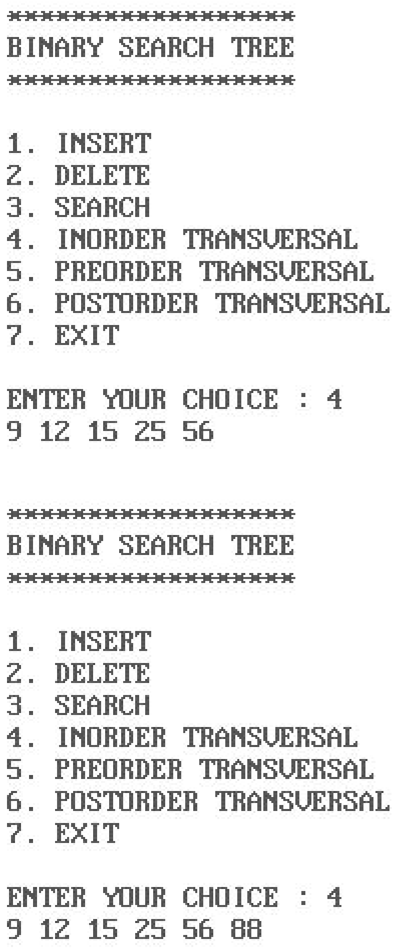
```
******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 5
12 9 15 56 25 88 _
```



```
******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 6
9 25 88 56 15 12 _
```



```
******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 3
ENTER DATA TO BE SEARCHED : 88
DATA FOUND!_

******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 2
ENTER DATA TO BE DELETED : 88
```



```
******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 4
9 12 15 25 56

******************
BINARY SEARCH TREE
******************

1. INSERT
2. DELETE
3. SEARCH
4. INORDER TRANSVERSAL
5. PREORDER TRANSVERSAL
6. POSTORDER TRANSVERSAL
7. EXIT

ENTER YOUR CHOICE : 4
9 12 15 25 56 88
```

END OF BINARY SEARCH TREE