Exp (20): write a c program to perform the following operations:

a) insert an element into a AVL tree

b) Delete an element from a AVL tree

c) search for a key element in an AVL tree

Aim: To write a c program to perform the following operations:

a) Insert an element into a AVL tree

b) Delete an element from a AVL tree

c) search for a key element in an AVL tree.

Algorithm:

*. start.

*. create AVL tree node structure.

*. Implement insert, delete, and search with balancing.

*. Display tree (inorder).

*. Stop.

program:

```
#include <stdio.h>
#include <stdlib.h>
  struct Node{
          int key, h;
          struct Node *l , *r;

  };
  int height(struct Node* n){ return n? n->h : 0;}
  int bf(struct Node* n){ return height(n->l)-height(n->
                                                  r); }

  struct Node* newNode(int k){
     struct Node* n = malloc(sizeof(struct Node));
     n->key =k; n->l = n->r = NULL; n->h=1;
     return n;
  }
  struct Node* rightRotate(struct Node* y){
    struct Node *x = y->l, *T = x->r;
     x->r = y; y->l = T;
    y->h = 1 +(height(y->l)>height(y->r)?
                height(y->l): height(y->r));
```

ing - 3

Q. Insert the [illegible]

```
x->h = 1+ (height (x->l) > height (x->r) ? height (x->l):
                            height (x->r));

  return x;

}
struct Node * left Rotate (struct Node* x) {
  struct Node *y = x->r, * T = y->l;
  y->l = x; x->r = T;
  x->h = 1+(height(x->l) > height(x->r) ? height (x->l):
                            height ( x->r));

  y->h = 1+ (height (y->l)>height (y->r) ? height (y->l):
                            height (y->r));

  return y;

}
struct Node* insert (struct Node*n, int k){
  if(!n) return newNode(k);
  if (k<n->key) n->l = insert (n->l, k);
  else if(k>n->key) n->r = insert (n->r , k);
  n->h = 1+ (height (n->l)>height(n->r)? height (n->l):
                            height ( n->r));

  int b= bf (n);
  if (b>1 && k<n->l->key) return rightRotate (n);
  if (b<-1 && k>n->r->key) return left Rotate (n);
  if ( b>1 && k>n->l->key) { n->l =left Rotate (n->l);
                    return right Rotate (n); }

  if (b<-1 && k<n ->r->key) { (n->r= right Rotate(n->r)
               return left Rotate (n); }
  return n;

}
struct Node *  minNode (struct Node * n) {
  while (n->l) n= n->l;
  return n;
}
```

```
struct node* delete (struct node* n, int k){
    if (!n) return n;
    if (k<n->key) n->l = delete (n->l, k);
    else if (k>n->key) n->r = delete (n->r, k);
    else {
        if (!n->l || !n->r){
            struct node* t= n->l ? n->l : n->r;
            free(n); return t;
        }
        struct node *t = minNode (n->r);
         n->key = t->key;
         n->r = delete (n->r, t->key);
    }
    n->h = 1+(height (n->l) > height (n->r) ? height(n->l):
                                        height(n->r));
    int b=bf(n);
    if (b>1 && bf(n->l)>=0) return rightRotate(n);
    if (b>1 && bf(n->l)<0){ n->l = leftRotate(n->l);
        return rightRotate(n); }
    if (b<-1 && bf(n->r)<=0) return leftRotate(n);
    if (b<-1 && bf(n->r)>0){ n->r = rightRotate(n->r);
         return leftRotate(n); }
    return n;
}

void inorder(struct node* r){ if (r){ inorder(r->l);
  printf("%d ", r->key); inorder(r->r); } }
int main(){
    struct node *root = NULL;
    root = insert(root, 30);
     root = insert(root, 20);
      root = insert(root, 40);
       root = insert(root, 10);
printf("Inorder:"); inorder(root);
 root = delete(root, 20);
 printf("\nAfter delete 20:"); inorder(root);
 printf("\n search 40: %s", search(root,40)?
      "found":"Not found");
}
```

output:

```
   inorder : 10   20   30   40

   After delete 20 :  10   30   40

   search  40 :  Found
```

Result: Thus, the program executed successfully