```c
// Project Name: Org Hierarchy and Escalation Levels

// Platform: C Programming

// Owner: Rajesh D. Nagawade, 8308788572 (Mb), rajesh.nagawade@gmail.com

// Purpose: Provides representation of hierarchy with insert, delete, traversal and escalation

// Learnings: BST Applications, Array of Pointers, Recursive Functions

// Complexity Level: Moderate to High

// Uses: Any Organization May use this core logic for escalation of services, org restructuring

//Scalable To: Payslip Generation, Effort Tracking & Performance Appraisal

//Duration: 28 Oct to 6 Nov 2024

//Team comprises of: The owner only


# include <stdio.h>

# include <stdlib.h>

# include <string.h>


struct Node

{

    int         id;

    char name[10];

    char skill[20];

    int mb;

    struct Node * parent;


    struct Node * Reportees[5];


    struct Node* left;

    struct Node* right;
```

```
} ;

struct Node * root = NULL;
struct Node * L1 = NULL;

struct Node s1 = {20,"KC","Own",111,NULL};
struct Node s2 = {10,"Nishant","CEO",222,NULL};
struct Node s3 = {30,"Abhi","Acnts",333,NULL};
struct Node s4 = {25,"Mounika","HR",444,NULL};
struct Node s5 = {35,"Vaibhav","Conten",555,NULL};
struct Node s6 = {5,"Sravan","Ops",666,NULL};
struct Node s7 = {15,"Sagar","Ops",777,NULL};
struct Node s8 = {3,"Vishnu","Ops",888,NULL};
struct Node s9 = {151,"Meghana","ML",771,NULL};
struct Node s10 = {152,"Mahi","DAA",772,NULL};
struct Node s11 = {153,"Rajesh","DSA",773,NULL};
struct Node s12 = {7,"PavanKo","Ops",999,NULL};
struct Node s13 = {71,"AmitPat","Java",991,NULL};
struct Node s14 = {72,"Kuldeep","Java",992,NULL};
struct Node s15 = {31,"Mahendra","ERP",881,NULL};
struct Node s16 = {32,"Pravindr","ERP",882,NULL};

struct Node * getnode(struct Node * s1)
{
  int i;

  struct Node * k;
```

```
    k = (struct Node*) malloc (sizeof(struct Node));
    *k = *s1;

    k->left = NULL;
    k->right = NULL;

    for(i=0; i<5; i++)
        k->Reportees[i] = NULL;

    return k;
}

void PrintEmployee(struct Node * p)
{
    printf("%3d \t %7s \t %7s %7d \t", p->id, p->name, p->skill, p->mb);
    if(p->parent != NULL)
        printf("%3d \n", p->parent->id);
    else
        printf("ROOT \n");
}

void InOrder(struct Node * p)
{
    if(!p) return ;

    InOrder(p->left);
    PrintEmployee(p);


    InOrder(p->right);
}


void insert(struct Node* n)
{
    struct Node* p = root;
    struct Node* q;
    while (p != NULL)
    {
        q = p;
        if (n->id < p->id)
            p = p->left;
        else
            p = p->right;
    }
    if (n->id < q->id)
    {


        printf("inserting %d on left of %d \n",n->id, q->id);
        q->left = n;
        n->parent = q;
    }
    else
    {
        printf("inserting %d on right of %d \n",n->id,q->id);

        q->right = n;

        n->parent = q;

    }

}


void DeleteManager(int no)

{


    struct Node* p=root;

    struct Node* q=root;

    int id;


    while(no!=p->id)

    {

        if(no<p->id)

        {

            q=p;

            p=p->left;

        }

        else

        {

            q=p;

            p=p->right;

        }

    } // here p reaches the node to delete
    if(p->left==NULL && p->right==NULL)// delete if it has no child

    {

        printf("Deleting %d %s \n", p->id, p->name);

        printf("Give the new manager for his reportees\n");

        scanf("%d",&id);

        printf("Transferring Reportees From %d to %d\n",p->id, id);


        if((q->left) && (q->left->id==no))

            q->left=NULL;

        else

            q->right=NULL;

        printf("Deleted Successfully %d \n",p->name);

    }

    else // don't delete

        printf("OPS Manager having Other Manager Reportees: Delete Pending\n");


    free(p);

}


void PrintOrg()

{

printf("\nThe Organization Hierarchy is as follows \n\n");

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

printf("ID \t\t Name \t\t Skill \t Mb \t Parent \n");

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


    InOrder(root);
```

```
}

struct Node * SearchManager(int id)
{
   struct Node* p = root;
   struct Node* q;
   while (p != NULL)
   {
      q = p;
      if (id < p->id)
         p = p->left;
      else if (id > p->id)
         p = p->right;
      else
         return p;
   }
   return p;
}
int IsReportee(struct Node * p, int id)
{
   int i=0;

   while(p->Reportees[i] != NULL )
   {
      if(p->Reportees[i]->id == id)
         return 1;
      i++;
```

```
    }
    return 0;
}
void SearchReportee(struct Node * p, int id)
{
    if(!p)
        return;

    SearchReportee(p->left,id);

    if(p->id == id)
        L1 = p->parent;
    else if(IsReportee(p,id) == 1)
        L1 = p;

    SearchReportee(p->right,id);
}
void PrintReportee(struct Node * m)
{
    int i = 0;
    struct Node * k;

    printf("Reportees of %s are \n", m->name);
    printf("ID \t\t Name \t\t Skill \t Mb \t Parent \n");
printf("----------------------------------------------\n");

    k = m->Reportees[i];
    while(k != NULL)
    {
        PrintEmployee(k);
        i++;
        k = m->Reportees[i];
    }
    printf("\n");
}

void NewEmployee()
{
    struct Node *k, *m;

    k = getnode(&s9);
    m = SearchManager(15);
    m->Reportees[0] = k;
    k->parent = m;
    printf("New Employee %s added successfully for %s \n", k->name,m->name);

    k = getnode(&s10);
    m = SearchManager(15);
    m->Reportees[1] = k;
    k->parent = m;
    printf("New Employee %s added successfully for %s \n", k->name,m->name);

    k = getnode(&s11);
    m = SearchManager(15);
```

```
m->Reportees[2] = k;

k->parent = m;

printf("New Employee %s added successfully for %s \n", k->name,m->name);


k = getnode(&s13);

m = SearchManager(7);

m->Reportees[0] = k;

k->parent = m;

printf("New Employee %s added successfully for %s \n", k->name,m->name);


k = getnode(&s14);

m = SearchManager(7);

m->Reportees[1] = k;

k->parent = m;

printf("New Employee %s added successfully for %s \n", k->name,m->name);


k = getnode(&s15);

m = SearchManager(3);

m->Reportees[0] = k;

k->parent = m;

printf("New Employee %s added successfully for %s \n", k->name,m->name);


k = getnode(&s16);

m = SearchManager(3);

m->Reportees[1] = k;

k->parent = m;

printf("New Employee %s added successfully for %s \n", k->name,m->name);
```

```
}


void Menu()

{

   int opt = 1,id;

   struct Node * k;


while(opt)

{

printf("<1> New Employee <2> Escalation <3> Print Org\n");

printf("<4> Delete Manager <5> XYZ <6> XYZ \n");

printf("<7> Search Manager <8> XYZ \n");

printf("<9> Exit \n");


scanf("%d", &opt);


   switch(opt)

   {

      case 1:

            NewEmployee();

            break;

      case 2:

            printf("Give ID for Escalation \n");

            scanf("%d", &id);

            SearchReportee(root,id);

            if(L1)

               printf("%d: Reports To: %s \n", id, L1->name);
```

```
        else
           printf("No Escalation 1 For %d \n",id);
        if((L1) && (L1->parent))
           printf("%s: Reports to: %s \n", L1->name, L1->parent->name);
        else
           printf("No Escalation 2 For %d \n",id);
        break;
case 3:
        PrintOrg();
        break;
case 4:
        DeleteManager(15);
        break;
case 5:
        // XYZ
        break;
case 6:
        // XYZ();
        break;
case 7:
        printf("Give ID of manager \n");
        scanf("%d",&id);
        k = SearchManager(id);
        PrintReportee(k);
        break;
case 8:
        // XYZ();
```

```
            break;
        case 9:
            opt = 0;
            break;
    };
}
}

int main()
{
    static int i = 0;

    struct Node * k, * m;

    root = getnode(&s1);
    root->parent = NULL;

    k = getnode(&s2);
    insert(k);

    k = getnode(&s3);
    insert(k);
    k = getnode(&s4);
    insert(k);
    k = getnode(&s5);
    insert(k);
    k = getnode(&s6);
    insert(k);

    k = getnode(&s7);

    insert(k);

    k = getnode(&s8);

    insert(k);

    k = getnode(&s12);

    insert(k);


    Menu();


    return 0;

}
```