```
#include<stdio.h>
#include<stdlib.h>
typedef struct nodetype
{
    int e,l,m,type;
    struct nodetype *child1, *child2, *child3, *forthchild;
}node;
node * root=NULL;
typedef struct stack
{
    node *data;
    int chno;
}st;
st s[30],k[30];
int top,top2;
int f=0;
node * deletion(node * ptr,int x);
node * insert(node * ptr,int item);
void display(node *ptr);
void balance(node * ptr);
int findmax(node * ptr); /* returns the max value in the subtree pointed by ptr */
void main()
{
    int ch,item,del;
    printf("\n********** 2-3 tree operations *************\n");
    printf("1----------insertion\n2----------deletion\n3------------Display the tree\n4-------
---Exit\n");
    printf("Enter your choice\n");
    scanf("%d",&ch);
    while(ch!=4)
    {
        switch(ch)
        {
            case 1:
                    printf("Enter the inserting item\n");
                    scanf("%d",&item);
                    top=0;
                    s[top].data=root;
                    s[top].chno=0;
                    root=insert(root,item);
                    display(root);
                    break;
            case 2:
                       printf("Enter the deleting item\n");
                       scanf("%d",&del);
                       top2=0;
                       k[top2].data=root;
                       k[top2].chno=0;
                       root=deletion(root,del);
                       printf("After deletion the tree is \n");
                       if(root!=NULL)
                       {
                           balance(root);
                           display(root);
                       }
                       else
                   printf("Empty tree\n");
                       break;
            case 3:
                    printf("The 2-3 tree is\n");
                    display(root);
                    break;
         }
         printf("Enter your next choice\n");
         scanf("%d",&ch);
    }
}
node *insert(node *ptr,int item)
{
    node *p, *parent, *p1;
    if((ptr==NULL)||(ptr->type==0)||((ptr->child1)->type==0))
    {
        /* insertion is to be performed now */
        p=(node *)malloc(sizeof(node));
        p->e=item;
        p->type=0;
        if(root==NULL)
            ptr=p;  /* !st insertion */
        else if(root!=NULL && ptr->type==0)
        {
            /* 2nd insertion */
            parent=(node *)malloc(sizeof(node));
```

```
        if(root->e<item)
        {
         parent->child1=ptr;
         parent->l=root->e;
         parent->child2=p;
         parent->m=item;
        }
        else
        {
          parent->child1=p;
         parent->l=item;
         parent->child2=ptr;
         parent->m=ptr->e;
        }
        parent->type=1;
        parent->child3=NULL;
        parent->forthchild=NULL;
     ptr=parent;
    }
    else
    {
        /* neither 2nd nor 1st insertion , but now ptr points to the node which
        has either 2 or 3 children already and the new one is to be placed as its
        child */
        if(item<ptr->l)
        {
            /* p will be the leftmost child of ptr */
            if(ptr->child3!=NULL)
             ptr->forthchild=ptr->child3;
            ptr->child3=ptr->child2;
            ptr->child2=ptr->child1;
            ptr->child1=p;
            ptr->l=item;
            ptr->m=(ptr->child2)->e;
        }
        else if((item>ptr->l)&&(item<ptr->m))
        {
            /* p will be the 2nd child */
            if(ptr->child3!=NULL)
                ptr->forthchild=ptr->child3;
            ptr->child3=ptr->child2;
            ptr->child2=p;
            ptr->m=item;
        }
        else
        {
            /* p will be the 3rd or 4th child */
            if(ptr->child3!=NULL)
            {
                if(item>(ptr->child3)->e)
                    ptr->forthchild=p;
                else
                {
                    ptr->forthchild=ptr->child3;
                    ptr->child3=p;
                }
            }
            else
             ptr->child3=p;
        }
    }
}
else if(item<ptr->l)
{
    top++;
    s[top].data=ptr;
    s[top].chno=1;
    ptr->child1=insert(ptr->child1,item);
}
else if((item>ptr->l)&&(item<ptr->m))
{
    top++;
  s[top].data=ptr;
    s[top].chno=2;
    ptr->child2=insert(ptr->child2,item);
}
else if(item>ptr->m)
{
    top++;
    s[top].data=ptr;
    if(ptr->child3!=NULL)
    {
```

```
            s[top].chno=3;
                ptr->child3=insert(ptr->child3,item);
        }
        else
        {
            s[top].chno=2;
            ptr->child2=insert(ptr->child2,item);
        }
    }
    /* now the neasure for the forth child is to be taken */
    if((ptr->type==1)&&(ptr->forthchild!=NULL))
    {
        p1=(node *)malloc(sizeof(node));
        p1->type=1;
        p1->child1=ptr->child3;
        p1->child2=ptr->forthchild;
        p1->child3=NULL;
        p1->l=findmax(p1->child1);
        p1->m=findmax(p1->child2);
        p1->forthchild=NULL;
        ptr->child3=NULL;
        ptr->forthchild=NULL;
        ptr->l=findmax(ptr->child1);
      ptr->m=findmax(ptr->child2);
        /* now make p1 a brother of ptr */
        parent=s[top].data;
        if(parent==ptr)
        {   /* when root has 4 children --- here the parent is the new root */
            parent=(node *)malloc(sizeof(node));
            parent->type=1;
            parent->l=ptr->m;
            parent->m=p1->m;
            parent->child1=ptr;
            parent->child2=p1;
            parent->child3=NULL;
            parent->forthchild=NULL;
         ptr=parent;
        }
        else
        {
            if(s[top].chno==3)
                parent->forthchild=p1;
            else if(s[top].chno==2)
            {
                parent->forthchild=parent->child3;
                parent->child3=p1;
            }
            else
            {
                parent->forthchild=parent->child3;
                parent->child3=parent->child2;
                parent->child2=p1;
            }
        }
    }
    if(ptr->type==1)
    {
        ptr->l=findmax(ptr->child1);
        ptr->m=findmax(ptr->child2);
    }
    return(ptr);
}
void display(node * ptr)
{
    node *a[30];
    int f=0,r=-1;
    printf("node\tchild1\tchild2\tchild3\n");
   printf("-------------------------------------------------\n");
    r++;
    a[r]=ptr;
    while(r!=-1)
    {
        ptr=a[f];
        if(f==r)
        {
         f=0;
         r=-1;
        }
        else
         f++;
        if(ptr->type==0)
            printf("%d\tx\tx\tx\n",ptr->e);
```

```
        else
        {
            printf("(%d:%d)\t",ptr->l,ptr->m);
            if((ptr->child1)->type==1)
            {
                printf("(%d:%d)\t(%d:%d)\t",(ptr->child1)->l,(ptr->child1)->m,(ptr->child2)->l
,(ptr->child2)->m);
                r++;
                a[r]=ptr->child1;
                r++;
                a[r]=ptr->child2;
                if(ptr->child3!=NULL)
                {
                    printf("(%d:%d)\n",(ptr->child3)->l,(ptr->child3)->m);
                    r++;
                    a[r]=ptr->child3;
                }
                else
                    printf("x\n");
            }
            else
            {
                printf("%d\t%d\t",(ptr->child1)->e,(ptr->child2)->e);
                r++;
                a[r]=ptr->child1;
                r++;
                a[r]=ptr->child2;
            if(ptr->child3!=NULL)
                {
                    printf("%d\n",(ptr->child3)->e);
                    r++;
                    a[r]=ptr->child3;
                }
                else
                    printf("x\n");
            }
        }
    }/* end of while */
}/* end of the display function */
int findmax(node *ptr)
{
  if(ptr->type==1)
  {
        if(ptr->child3!=NULL)
            return(findmax(ptr->child3));
        else
            return(findmax(ptr->child2));
  }
  else
    return(ptr->e);
}
node * deletion(node * ptr,int x)
{
    node * parent,*pp,*g;
    parent=k[top2].data;
    if(ptr==NULL)
        printf("deleting item not found \n");
  else if((ptr->type==0)&&(ptr->e==x))
    {
        /* this node is to be deleted */
        if(ptr==root)
        {
            free(ptr);
            ptr=NULL;
        }
        else if(parent->child3!=NULL)
        {
            if(k[top2].chno==1)
            {
                ptr=parent->child2;
                parent->child1=parent->child2;
                parent->child2=parent->child3;
                parent->child3=NULL;

            }
            else if(k[top2].chno==2)
            {
            ptr=parent->child3;
            parent->child2=parent->child3;
                parent->child3=NULL;

            }
```

```
    else
        ptr=NULL;
}
else if(parent==root)
{

    if(k[top2].chno==1)
        ptr=parent->child2;
    else
        ptr=parent->child1;
    parent->type=ptr->type;
    parent->child1=ptr->child1;
    parent->child2=ptr->child2;
    parent->child3=ptr->child3;
    if(k[top2].chno==1)
     ptr=ptr->child1;
    else
    ptr=ptr->child2;
}
else
{
    pp=k[top2-1].data;
    if(k[top2-1].chno==1)
    {
        g=pp->child2;
        if(g->child3==NULL)
        {
            g->child3=g->child2;
            g->child2=g->child1;
            if(k[top2].chno==1)
            {
                g->child1=parent->child2;
                parent->child2=NULL;
            }
            else
            {
                g->child1=parent->child1;
                parent->child1=NULL;
            }
            free(ptr);
            ptr=NULL;
        }
        else
        {
            if(k[top2].chno==1)
            {
                ptr=parent->child2;
                parent->child1=parent->child2;
            }
            else
                ptr=g->child1;
            parent->child2=g->child1;
            g->child1=g->child2;
            g->child2=g->child3;
            g->child3=NULL;
        }
    }
    else if(k[top2-1].chno==2)
    {
        g=pp->child1;
        if(g->child3==NULL)
        {
            if(k[top2].chno==1)
            {
                g->child3=parent->child2;
                parent->child2=NULL;
            }
            else
            {
                g->child3=parent->child1;
                parent->child1=NULL;
            }
            free(ptr);
            ptr=NULL;
        }
        else
        {
            if(k[top2].chno==2)
            {
                ptr=parent->child1;
                parent->child2=parent->child1;
            }
```

```
                else
                    ptr=g->child3;
                parent->child1=g->child3;
                g->child3=NULL;
            }
        }
        else
        {
            g=pp->child2;
        if(g->child3==NULL)
            {
                if(k[top2].chno==1)
                {
                    g->child3=parent->child2;
                    parent->child2=NULL;
                }
                else
                {
                    g->child3=parent->child1;
                    parent->child1=NULL;
                }
                free(ptr);
                ptr=NULL;
            }
            else
            {
                if(k[top2].chno==2)
                {
                    ptr=parent->child1;
                    parent->child2=parent->child1;
                }
                else
                    ptr=g->child3;
                parent->child1=g->child3;
                g->child3=NULL;
            }
        }
    }
}
 else
 {
        if(x<=ptr->l)
        {
            top2++;
            k[top2].data=ptr;
            k[top2].chno=1;
            ptr->child1=deletion(ptr->child1,x);
        }
        else if((x>ptr->l)&&(x<=ptr->m))
        {
            top2++;
            k[top2].data=ptr;
            k[top2].chno=2;
            ptr->child2=deletion(ptr->child2,x);
        }
        else
        {
             top2++;
            k[top2].data=ptr;
            k[top2].chno=3;
            ptr->child3=deletion(ptr->child3,x);
        }
        if((ptr->type==1)&&(ptr->child1==NULL)&&(ptr->child2==NULL))
        {
            ptr->e=x;
            ptr->type=0;
            f=1;
            printf("**%d,%d\n",(k[top2].data)->l,(k[top2].data)->m);
            if(k[top2].chno==1)
            {
                (k[top2].data)->child1=deletion(ptr,x);
                ptr=(k[top2].data)->child1;
            }
            else if( k[top2].chno==2)
            {
                (k[top2].data)->child2=deletion(ptr,x);
                ptr=(k[top2].data)->child2;
            }
            else
            {
                  (k[top2].data)->child3=deletion(ptr,x);
                  ptr=(k[top2].data)->child3;
```

```
                }

                f=0;
            }
     }
    if(f==0)
        top2--;
    return(ptr);
}

void balance(node * ptr)
{
    if(ptr->type==1)
    {
        balance(ptr->child1);
        balance(ptr->child2);
        if(ptr->child3!=NULL)
            balance(ptr->child3);
        ptr->l=findmax(ptr->child1);
        ptr->m=findmax(ptr->child2);
    }
}
```