LAB XI

```
//Duplicate keys is not handled
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
char name[100];
char code[20];
struct node *left,*right;
};
typedef struct node node;

node *createb(char *);
void get_cn(char *,char *,char *);
void assign_node(node *,char*,char*);
node *get_insertn(char *,node *);
void inprint(node*);
void preprint(node*);
void postprint(node*);
void insertnode(node **);
void deletenode(node **);
int get_delnode(char *,node *,node **);
node *nodeparent(node*,node*);
node *inorders(node *);
void delete_l(node *,node *);
void delete_lor(node *,node *);
void delete_lar(node *,node *);
int main()
{
int ch,pr;
node *bsr;
bsr=createb("stncode.dat");
printf("The tree made from the data file printed inorder\n");
inprint(bsr);
printf("Enter 1 to insert a node:\n");
printf("Enter 2 to delete a node:\n");
printf("Enter 3 to print the tree:\n");
printf("Enter 0 to stop:\n");
printf("enter your choice:");
scanf("%d",&ch);
while(ch !=0)
{
        switch(ch)
        {
                case 1:
                        insertnode(&bsr);
                        break;
                case 2: deletenode(&bsr);
                        break;
```

```
                case 3: printf("Enter 1 for preorder, 2 for inorder, 3 for postorder print
                        scanf("%d",&pr);
                        if(pr==1)
                             preprint(bsr);
                        else if(pr==2)
                             inprint(bsr);
                        else
                             postprint(bsr);
                        break;
                case 0:
                        return 0;
                default:
                        printf("Unknow input: stop\n");
                        return 0;
        }
printf("Enter 1 to insert a node:\n");
printf("Enter 2 to delete a node:\n");
printf("Enter 3 to print the tree:\n");
printf("Enter 0 to stop:\n");
printf("enter your choice:");
scanf("%d",&ch);
}

return 0;
}

void delete_lar(node *p,node *root)
{
node *parnt,*q;
//find the inorder successor
q=inorders(p->right);
parnt=nodeparent(q,root);
strcpy(p->name,q->name);
strcpy(p->code,q->code);
//delete node has no children
if(q->left==NULL && q->right==NULL)
{
delete_l(q,parnt);
return;
}

//delete node has only one child
if(q->left==NULL || q->right==NULL)
{
delete_lor(q,parnt);
return;
}


}
```

```
void delete_lor(node *p,node *root)
{
node *parnt;

parnt=nodeparent(p,root);
if(p->left!=NULL)
   {
        if(parnt->left==p)
                parnt->left=p->left;
        else
                parnt->right=p->left;
   }
else

   {
        if(parnt->left==p)
                parnt->left=p->right;
        else
                parnt->right=p->right;
   }
        free(p);
        return;


}

void delete_l(node *p,node *root)
{
node *parnt;
  parnt=nodeparent(p,root);

        if(parnt->left==p)
                parnt->left=NULL;
        else
                parnt->right=NULL;
        free(p);
}

void deletenode(node **root)
{
int i,flag;
char ch;
char code[20];
node *p,*parnt,*q;
printf("Enter the stn code to be deleted:");
//Flushing the input
while ((ch = getchar()) != '\n' && ch != EOF);
fgets(code,20,stdin);
//Removes '\n' from stn code
```

```
i=0;
while(code[i] !='\n')
{
  i++;
}
code[i]='\0';

flag=get_delnode(code,*root,&p);
if(flag==0)
{
        printf("The stn code does not exit: stop\n");
        exit(1);
}
//If root is the only node and it is to be deleted

if(p==*root && p->left==NULL && p->right==NULL)
   {
     *root=NULL;
      return;
   }


//delete node is not root and has no children

if(p->left==NULL && p->right==NULL)
{
delete_l(p,*root);
return;
}

//delete node has only one child
if(p->left==NULL || p->right==NULL)
  {
    delete_lor(p,*root);
    return;
  }

//delete node has both children
delete_lar(p,*root);

}


node *inorders(node *root)
{
if(root->left==NULL)
  return root;
return inorders(root->left);
}

node *nodeparent(node *p,node *root)
```

```
{

if(root->left==p || root->right==p)
{
        return root;
}
else
{
                if(strcmp(p->code,root->code)<0)
                        return nodeparent(p,root->left);
                if(strcmp(p->code,root->code)>0)
                        return nodeparent(p,root->right);

}
printf("Some probelm\n");
exit(1);
}




int get_delnode(char *code,node *root,node **p)
{

if(root==NULL)
{
        return 0;
}

        if(strcmp(code,root->code)==0)
{
        *p=root;
        return 1;
}

        if(strcmp(code,root->code)<0)
{
        return get_delnode(code,root->left,p);
}
else
{
                return get_delnode(code,root->right,p);
}


}



void insertnode(node **root)
{
```

```
char ch;
char buff[200],code[20],name[100];
node *p;
printf("Enter the station name followed by stn code:");
//Flushing the input
while ((ch = getchar()) != '\n' && ch != EOF);
fgets(buff,200,stdin);
get_cn(name,code,buff);
p=get_insertn(code,*root);
assign_node(p,name,code);
p->left=NULL;
p->right=NULL;

if(*root==NULL)
{
        *root=p;
}
}

void inprint(node *root)
{
if(root !=NULL)
{
        inprint(root->left);
        printf("%s (%s)\n",root->name,root->code);
        inprint(root->right);
}
}

void preprint(node *root)
{
if(root !=NULL)
{
        printf("%s (%s)\n",root->name,root->code);
        preprint(root->left);
        preprint(root->right);
}
}

void postprint(node *root)
{
if(root !=NULL)
{
        postprint(root->left);
        postprint(root->right);
        printf("%s (%s)\n",root->name,root->code);
}
}
```

```
node *createb(char *file)
{
node *root,*p;
char c,buff[200],code[20],name[100];
int i;
FILE *fp;
fp=fopen(file,"r");
//Read the first line
if(fgets(buff,200,fp)==NULL)
  {
   return NULL;
  }
root=(node *)calloc(1,sizeof(node));
p=root;
//Separate the station name and code
get_cn(name,code,buff);
//Assign name and code to a node
assign_node(p,name,code);
p->left=NULL;
p->right=NULL;

//Read other lines
  while(fgets(buff,200,fp)!=NULL)
  {
  get_cn(name,code,buff);
// Get the address of the newly created node
  p=get_insertn(code,root);
  assign_node(p,name,code);
  p->left=NULL;
  p->right=NULL;
  }
return root;
}

node *get_insertn(char *code,node *root)
{
if(root==NULL)
{
        return (node *)calloc(1,sizeof(node));

}

if(strcmp(code,root->code)<0)
{

    if(root->left == NULL)
      {
       root->left=(node *)calloc(1,sizeof(node));
       return root->left;
      }
    return get_insertn(code,root->left);
```

```
}


if(strcmp(code,root->code)>0)
{

    if(root->right == NULL)
      {
       root->right=(node *)calloc(1,sizeof(node));
       return root->right;
      }
    return get_insertn(code,root->right);

}

printf("Duplicate key not allowed\n");
exit(1);
}




void assign_node(node *p,char *name,char *code)
{
strcpy(p->name,name);
strcpy(p->code,code);
}

void get_cn(char *name,char *code,char *buff)
{
int i,j;
char c;
//Move to the end of the string
//Note that buff has '\n' before '\0'
i=strlen(buff);
c=buff[i];
while(c!=' ')
 {
  i--;
  c=buff[i];
 }
j=i+1;
//j has the starting index of the stn code
//assign the code to the node
c=buff[j];
while(c!='\n')
 {
  code[j-i-1]=c;
  j++;
  c=buff[j];
```

```
 }
code[j-i-1]='\0';
//Skip the white space between station name and station code
j=i;
c=buff[j];
while(c==' ')
 {
  j--;
  c=buff[j];
 }
//assign the station name to the name
i=0;
while(i<=j)
 {
  name[i]=buff[i];
  i++;
  }
name[i]='\0';

}
```