**exp7-9**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MaxSize 30

#define NAMEWIDTH 10

typedef struct fnode

{

    char father[NAMEWIDTH];

    char wife[NAMEWIDTH];

    char son[NAMEWIDTH];

}FamType;

typedef struct tnode

{

    char name[NAMEWIDTH];

    struct tnode \*lchild,\*rchild;

}Btree;

int n;

FamType fam[MaxSize];

Btree \*CreateBTree(char \*root)

{

    int i=0,j;

    Btree \*b,\*p;

    b=(Btree \*)malloc(sizeof(Btree));

    strcpy(b->name,root);

    b->lchild=b->rchild=NULL;

    while(i<n&&strcmp(fam[i].father,root)!=0)

        i++;

    if(i<n)

    {

        p=(Btree \*)malloc(sizeof(Btree));

        p->lchild=p->rchild=NULL;

        strcpy(p->name,fam[i].wife);

        b->lchild=p;

        for(j=0;j<n;j++)

        {

            if(strcmp(fam[j].father,root)==0)

            {

                p->rchild=CreateBTree(fam[j].son);

                p=p->rchild;

            }

        }

    }

    return(b);

}

void DispTree(Btree \*b)

{

    if(b!=NULL)

    {

        printf("%s",b->name);

        if(b->lchild!=NULL||b->rchild!=NULL)

        {

            printf("(");

            DispTree(b->lchild);

            if(b->rchild!=NULL)

            {

                printf(",");

            }

            DispTree(b->rchild);

            printf(")");

        }

    }

}

Btree \*FindNode(Btree \*b,char xm[])

{

    Btree \*p;

    if(b==NULL)

        return(NULL);

    else

    {

        if(strcmp(b->name,xm)==0)

            return(b);

        else

        {

            p=FindNode(b->lchild,xm);

            if(p!=NULL)

                return p;

            else

                return(FindNode(b->rchild,xm));

        }

    }

}

void Findson(Btree \*b)

{

    char xm[NAMEWIDTH];

    Btree \*p;

    printf(" >>父亲名字:");

    scanf("%s",xm);

    p=FindNode(b,xm);

    if(p==NULL)

        printf(" >>不存在%s的父亲!\n",xm);

    else

    {

        p=p->lchild;

        if(p==NULL)

            printf(" >>%s没有妻子!\n",xm);

        else

        {

           p=p->rchild;

           if(p==NULL)

                printf(" >>%s没有儿子!\n",xm);

            else

            {

                printf(" >>%s的儿子:",xm);

                while(p!=NULL)

                {

                    printf("%10s",p->name);

                    p=p->rchild;

                }

                printf("\n");

            }

        }

    }

}

int Path(Btree \*b,Btree \*s)

{

    Btree \*St[MaxSize];

    Btree \*p;

    int i,top=-1;

    bool flag;

    do

    {

        while(b)

        {

           top++;

           St[top]=b;

           b=b->lchild;

        }

        p=NULL;

        flag=true;

        while(top!=-1&&flag)

        {

            b=St[top];

            if(b->lchild==p)

            {

                if(b==s)

                {

                    printf(" >>所有祖先:");

                    for(i=0;i<top;i++)

                        printf("%s",St[i]->name);

                    printf("\n");

                    return 1;

                }

                else

                {

                    top--;

                    p=b;

                }

            }

            else

            {

                b=b->rchild;

                flag=false;

            }

        }

    } while (top!=-1);

    return 0;

}

void Ancestor(Btree \*b)

{

    Btree \*p;

    char xm[NAMEWIDTH];

    printf(" >>输入姓名:");

    scanf("%s",xm);

    p=FindNode(b,xm);

    if(p!=NULL)

        Path(b,p);

    else

        printf(" >>不存在%s\n",xm);

}

void DestroyBTree(Btree \*b)

{

    if(b!=NULL)

    {

        DestroyBTree(b->lchild);

        DestroyBTree(b->rchild);

        free(b);

    }

}

void DelAll()

{

    FILE \*fp;

    if((fp=fopen("fam.dat","wb"))==NULL)

    {

        printf(" >>不能打开家谱文件\n");

        return;

    }

    n=0;

    fclose(fp);

}

void ReadFile()

{

    FILE \*fp;

    long len;

    int i;

    if((fp=fopen("fam.dat","rb"))==NULL)

    {

        n=0;

        return;

    }

    fseek(fp,0,2);

    len=ftell(fp);

    n=len/sizeof(FamType);

    for(i=0;i<n;i++)

        fread(&fam[i],sizeof(FamType),1,fp);

    fclose(fp);

}

void SaveFile()

{

    int i;

    FILE \*fp;

    if((fp=fopen("fam.bat","wb"))==NULL)

    {

        printf(" >>数据家谱文件不能打开\n");

        return;

    }

    for(i=0;i<n;i++)

    {

        fwrite(&fam[i],sizeof(FamType),1,fp);

    }

    fclose(fp);

}

void InputFam()

{

    printf(" >>输入父亲、母亲和儿子姓名:");

    scanf("%s %s %s",fam[n].father,fam[n].wife,fam[n].son);

    n++;

}

void OutputFile()

{

    int i;

    if(n<=0)

    {

        printf(" >>没有任何记录\n");

        return;

    }

    printf("        父亲        母亲        儿子\n");

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

    for(i=0;i<n;i++)

        printf("  %10s%10s%10s\n",fam[i].father,fam[i].wife,fam[i].son);

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

}

void Fileop()

{

    int sel;

    do

    {

        printf(">1:输入 2:输出 9:全清 0:存盘返回 请选择:");

        scanf("%d",&sel);

        switch(sel)

        {

            case 9:

                DelAll(); break;

            case 1:

                InputFam(); break;

            case 2:

                OutputFile(); break;

            case 0:

                SaveFile(); break;

        }

    } while (sel!=0);

}

void BTreeop()

{

    Btree \*b;

    int sel;

    if(n==0) return;

    b=CreateBTree(fam[0].father);

    do

    {

        printf(">1:括号表示法 2.找某人所有儿子 3.找某人所有祖先 0:返回 请选择");

        scanf("%d",&sel);

        switch(sel)

        {

            case 1:

                printf(" >>"); DispTree(b); printf("\n"); break;

            case 2:

                Findson(b); break;

            case 3:

                printf(" >>"); Ancestor(b); break;

        }

    } while (sel!=0);

    DestroyBTree(b);

}

int main()

{

    Btree \*b;

    int sel;

    ReadFile();

    do

    {

        printf("\*1.文件操作 2:家谱操作 0:退出 请选择：");

        scanf("%d",&sel);

        switch (sel)

        {

        case 1:

            Fileop(); break;

        case 2:

            BTreeop(); break;

        }

    } while (sel!=0);

    return 0;

}

**exp7-10**

#include<stdio.h>

#include<malloc.h>

#include<string.h>

#include<stdlib.h>

#define MaxSize 100

#define MaxSons 10

typedef struct

{

    char fname[20];

    char sname[20];

}RecType;

typedef struct node

{

    char data[20];

    struct node \*sons[MaxSize];

}TSonNode;

void ReadFile(RecType R[],int &n)

{

    FILE \*fp;

    n=0;

    if((fp=fopen("abc.txt","r"))==NULL)

    {

        printf("不能打开文件abc.txt");

        return;

    }

    while(!feof(fp))

    {

        fscanf(fp,"%s",&R[n].fname);

        fscanf(fp,"%s",&R[n].sname);

        n++;

    }

    fclose(fp);

}

TSonNode \*CreateTree(char root[],RecType R[],int n)

{

    int i,j,k;

    TSonNode \*t;

    t=(TSonNode \*)malloc(sizeof(TSonNode));

    strcpy(t->data,root);

    for(k=0;k<MaxSons;k++)

        t->sons[k]=NULL;

    i=0,j=0;

    while(i<n)

    {

        if(strcmp(R[i].fname,root)==0)

        {

            t->sons[j]=CreateTree(R[i].sname,R,n);

            j++;

        }

        i++;

    }

    return t;

}

void DispTree(TSonNode \*t)

{

    int i;

    if(t!=NULL)

    {

        printf("%s",t->data);

        if(t->sons[0]!=NULL)

        {

            printf("(");

            for(i=0;i<MaxSons;i++)

            {

                DispTree(t->sons[i]);

                if(t->sons[i+1]!=NULL)

                    printf(",");

                else

                    break;

            }

            printf(")");

        }

    }

}

void DestroyTree(TSonNode \*&t)

{

    int i;

    if(t!=NULL)

    {

        for(i=0;i<MaxSons;i++)

        {

            if(t->sons[i]!=NULL)

                DestroyTree(t->sons[i]);

            else

                break;

        }

    }

    free(t);

}

TSonNode \*FindNode(TSonNode \*t,char x[])

{

    int i;

    TSonNode \*p;

    if(t==NULL)

        return NULL;

    else if (strcmp(t->data,x)==0)

        return t;

    else

    {

        for(i=0;i<MaxSons;i++)

            if(t->sons[i]!=NULL)

            {

                p=FindNode(t->sons[i],x);

                if(p!=NULL) return p;

            }

            else break;

        return NULL;

    }

}

int ChildCount(TSonNode \*p)

{

    int i,num=0;

    for(i=0;i<MaxSons;i++)

        if(p->sons[i]!=NULL)

            num++;

        else

            break;

    return num;

}

int Sonnum(TSonNode \*t,char x[])

{

    TSonNode \*p;

    p=FindNode(t,x);

    if(p==NULL)

        return 0;

    else

        return ChildCount(p);

}

int LeafCount(TSonNode \*t)

{

    int i,num=0;

    if(t==NULL)

        return 0;

    else

    {

        if(t->sons[0]==NULL)

            num++;

        else

        {

            for(i=0;i<MaxSons;i++)

                if(t->sons[i]!=NULL)

                    num+=LeafCount(t->sons[i]);

                else break;

        }

        return num;

    }

}

int Classnum(TSonNode \*t,char x[])

{

    TSonNode \*p;

    p=FindNode(t,x);

    if(p==NULL)

        return 0;

    else

        return LeafCount(p);

}

int LeafSum(TSonNode \*t)

{

    int i,sum=0;

    if (t==NULL)

        return 0;

    else

    {

        if(t->sons[0]==NULL)

            return atoi(t->data);

        else

        {

            for(i=0;i<MaxSons;i++)

                if(t->sons[i]!=NULL)

                    sum+=LeafCount(t->sons[i]);

                else break;

        }

        return sum;

    }

}

int Studnum(TSonNode \*t,char x[])

{

    TSonNode \*p;

    p=FindNode(t,x);

    if(p==NULL)

        return 0;

    else

        return LeafSum(p);

}

int main()

{

    TSonNode \*t;

    RecType R[MaxSize];

    int n;

    printf("(1)从abc.txt文件读取数据到R数组中\n");

    ReadFile(R,n);

    if(n==0) return 1;

    printf("(2)由数组R创建树t的孩子链存储结构\n");

    t=CreateTree(R[0].fname,R,n);

    printf("(3)输出树t:"); DispTree(t); printf("\n");

    printf("(4)求计算机学院的专业数:%d\n",Sonnum(t,"计算机学院"));

    printf("(5)求计算机学院的班数:%d\n",Classnum(t,"计算机学院"));

    printf("(6)求电信学院的学生数:%d\n",Studnum(t,"电信学院"));

    printf("(7)销毁树t");

    DestroyTree(t);

    return 1;

}

**exp7-11**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MaxSize 100

typedef struct

{

char data[MaxSize];

int length;

} SqString;

typedef char ElemType;

typedef struct node

{

ElemType data;

struct node \*lchild;

struct node \*rchild;

} BTNode;

void DispStr(SqString s)

{

if (s.length>0)

{ for (int i=0;i<s.length;i++)

printf("%c",s.data[i]);

printf("\n");

}

}

void DispBTree(BTNode \*b)

{

if (b!=NULL)

{ printf("%c",b->data);

if (b->lchild!=NULL || b->rchild!=NULL)

{ printf("(");

DispBTree(b->lchild);

if (b->rchild!=NULL) printf(",");

DispBTree(b->rchild);

printf(")");

}

}

}

void DestroyBTree(BTNode \*&b)

{ if (b!=NULL)

{ DestroyBTree(b->lchild);

DestroyBTree(b->rchild);

free(b);

}

}

void CreateBTree(BTNode \* &b,char str[])

{

BTNode \*St[MaxSize],\*p=NULL;

int top=-1,k,j=0;

char ch;

b=NULL;

ch=str[j];

while (ch!='\0')

{

switch(ch)

{

case '(':top++;St[top]=p;k=1; break;

case ')':top--;break;

case ',':k=2; break;

default:p=(BTNode \*)malloc(sizeof(BTNode));

p->data=ch;p->lchild=p->rchild=NULL;

if (b==NULL)

b=p;

else

{

switch(k)

{

case 1:St[top]->lchild=p;break;

case 2:St[top]->rchild=p;break;

}

}

}

j++;

ch=str[j];

}

}

void StrAssign(SqString &s,char cstr[])

{

int i;

for (i=0;cstr[i]!='\0';i++)

s.data[i]=cstr[i];

s.length=i;

}

SqString Concat(SqString s,SqString t) {

SqString str;

int i;

str.length=s.length+t.length;

for (i=0;i<s.length;i++) str.data[i]=s.data[i];

for (i=0;i<t.length;i++)

str.data[s.length+i]=t.data[i];

return str;

}

SqString PreOrderSeq(BTNode \*b)

{

SqString str,strl,strr;

if(b==NULL)

{

StrAssign(str, "#");

return str;

}

str.data[0] = b->data;

str.length = 1;

strl = PreOrderSeq(b->lchild);

strr = PreOrderSeq(b->rchild);

str = Concat(str, strl);

str = Concat(str, strr);

return str;

}

int i;

BTNode\* CreatePreSeq(SqString str)

{

char letter;

if(i>=str.length)

return NULL;

letter = str.data[i];

i++;

if(letter=='#')

return NULL;

BTNode \*p;

p = (BTNode \*)malloc(sizeof(BTNode));

p->data = letter;

p->lchild = CreatePreSeq(str);

p->rchild = CreatePreSeq(str);

return p;

}

int main()

{

BTNode \*b, \*b1;

SqString str;

printf("(1)创建二叉链b\n");

CreateBTree(b,"A(B(D,E(,G)),C(,F(H,I)))");

printf("(2)采用括号表示输出二叉树b：\n");

DispBTree(b);

printf("\n(3)对二叉链b采用先序遍历产生先序序列化序列str:\n");

str=PreOrderSeq(b);

printf("(4)输出先序序列化序列str:\n");

DispStr(str);

printf("(5)由str构建二叉链b1(反序列化)\n");

b1=CreatePreSeq(str);

printf("(6)采用括号表示输出二叉链b1\n");

DispBTree(b1);

printf("\n(7)销毁二叉树b和b1\n");

DestroyBTree(b);

DestroyBTree(b1);

}

**exp7-12**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MaxSize 100

typedef struct

{

char data[MaxSize];

int length;

} SqString;

typedef char ElemType;

typedef struct node

{

ElemType data;

struct node \*lchild;

struct node \*rchild;

} BTNode;

void DispStr(SqString s)

{

if (s.length>0)

{ for (int i=0;i<s.length;i++)

printf("%c",s.data[i]);

printf("\n");

}

}

void DispBTree(BTNode \*b)

{

if (b!=NULL)

{ printf("%c",b->data);

if (b->lchild!=NULL || b->rchild!=NULL)

{ printf("(");

DispBTree(b->lchild);

if (b->rchild!=NULL) printf(",");

DispBTree(b->rchild);

printf(")");

}

}

}

void DestroyBTree(BTNode \*&b)

{ if (b!=NULL)

{ DestroyBTree(b->lchild);

DestroyBTree(b->rchild);

free(b);

}

}

void CreateBTree(BTNode \* &b,char str[])

{

BTNode \*St[MaxSize],\*p=NULL;

int top=-1,k,j=0;

char ch;

b=NULL;

ch=str[j];

while (ch!='\0')

{

switch(ch)

{

case '(':top++;St[top]=p;k=1; break;

case ')':top--;break;

case ',':k=2; break;

default:p=(BTNode \*)malloc(sizeof(BTNode));

p->data=ch;p->lchild=p->rchild=NULL;

if (b==NULL)

b=p;

else

{

switch(k)

{

case 1:St[top]->lchild=p;break;

case 2:St[top]->rchild=p;break;

}

}

}

j++;

ch=str[j];

}

}

void StrAssign(SqString &s,char cstr[])

{

int i;

for (i=0;cstr[i]!='\0';i++)

s.data[i]=cstr[i];

s.length=i;

}

SqString Concat(SqString s,SqString t) {

SqString str;

int i;

str.length=s.length+t.length;

for (i=0;i<s.length;i++) //s.data[0..s.length-1] str.data[i]=s.data[i];

for (i=0;i<t.length;i++) //t.data[0..t.length-1] str.data[s.length+i]=t.data[i];

return str;

}

SqString PreOrderSeq(BTNode \*b)

{

SqString str,strl,strr;

if(b==NULL)

{

StrAssign(str, "#");

return str;

}

str.data[0] = b->data;

str.length = 1;

strl = PreOrderSeq(b->lchild);

strr = PreOrderSeq(b->rchild);

str = Concat(str, strl);

str = Concat(str, strr);

return str;

}

int i;

BTNode\* CreatePreSeq(SqString str)

{

char letter;

if(i>=str.length)

return NULL;

letter = str.data[i];

i++;

if(letter=='#')

return NULL;

BTNode \*p;

p = (BTNode \*)malloc(sizeof(BTNode));

p->data = letter;

p->lchild = CreatePreSeq(str);

p->rchild = CreatePreSeq(str);

return p;

}

void GetNext(SqString t,int next[]) ֵ

{ int j,k;

j=0;k=-1;next[0]=-1;

while (j<t.length-1)

{ if (k==-1 || t.data[j]==t.data[k]) { j++;k++;

next[j]=k;

}

else k=next[k];

}

}

int KMPIndex(SqString s,SqString t)

{

int next[MaxSize],i=0,j=0;

GetNext(t,next);

while (i<s.length && j<t.length)

{ if (j==-1 || s.data[i]==t.data[j])

{ i++;

j++;

}

else j=next[j];

}

if (j>=t.length)

return(i-t.length); else

return(-1); }

bool compere(BTNode \*b1,BTNode \*b2)

{

SqString s1 = PreOrderSeq(b1);

SqString s2 = PreOrderSeq(b2);

if(KMPIndex(s1,s2)!=-1)

return true;

else

return false;

}

int main()

{

bool find;

BTNode \*b1, \*b2;

CreateBTree(b1,"A(B(D,E(,G)),C(,F(H,I)))");

printf("b1:");

DispBTree(b1);

printf("\n");

CreateBTree(b2,"C(,F(H,I))");

printf("b2:");

DispBTree(b2);

find=compere(b1, b2);

if(find)

printf("\n有相同子树");

else

printf("\n没有相同子树");

}

**exp7-13**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MaxSize 100

typedef struct

{

char data[MaxSize];

int length;

} SqString;

typedef char ElemType;

typedef struct node

{

ElemType data;

struct node \*lchild;

struct node \*rchild;

} BTNode;

void DispStr(SqString s)

{

if (s.length>0)

{ for (int i=0;i<s.length;i++)

printf("%c",s.data[i]);

printf("\n");

}

}

void DispBTree(BTNode \*b)

{

if (b!=NULL)

{ printf("%c",b->data);

if (b->lchild!=NULL || b->rchild!=NULL)

{ printf("(");

DispBTree(b->lchild);

if (b->rchild!=NULL) printf(",");

DispBTree(b->rchild);

printf(")");

}

}

}

void DestroyBTree(BTNode \*&b)

{ if (b!=NULL)

{ DestroyBTree(b->lchild);

DestroyBTree(b->rchild);

free(b);

}

}

void CreateBTree(BTNode \* &b,char str[])

{

BTNode \*St[MaxSize],\*p=NULL;

int top=-1,k,j=0;

char ch;

b=NULL;

ch=str[j];

while (ch!='\0')

{

switch(ch)

{

case '(':top++;St[top]=p;k=1; break;

case ')':top--;break;

case ',':k=2; break;

default:p=(BTNode \*)malloc(sizeof(BTNode));

p->data=ch;p->lchild=p->rchild=NULL;

if (b==NULL)

b=p;

else

{

switch(k)

{

case 1:St[top]->lchild=p;break;

case 2:St[top]->rchild=p;break;

}

}

}

j++;

ch=str[j];

}

}

void StrAssign(SqString &s,char cstr[])

{

int i;

for (i=0;cstr[i]!='\0';i++)

s.data[i]=cstr[i];

s.length=i;

}

SqString Concat(SqString s,SqString t)

{

SqString str;

int i;

str.length=s.length+t.length;

for (i=0;i<s.length;i++)

str.data[i]=s.data[i];

for (i=0;i<t.length;i++)

str.data[s.length+i]=t.data[i];

return str;

}

SqString PreOrderSeq(BTNode \*b)

{

SqString str,strl,strr;

if(b==NULL)

{

StrAssign(str, "#");

return str;

}

str.data[0] = b->data;

str.length = 1;

strl = PreOrderSeq(b->lchild);

strr = PreOrderSeq(b->rchild);

str = Concat(str, strl);

str = Concat(str, strr);

return str;

}

int i;

BTNode\* CreatePreSeq(SqString str)

{

char letter;

if(i>=str.length)

return NULL;

letter = str.data[i];

i++;

if(letter=='#')

return NULL;

BTNode \*p;

p = (BTNode \*)malloc(sizeof(BTNode));

p->data = letter;

p->lchild = CreatePreSeq(str);

p->rchild = CreatePreSeq(str);

return p;

}

void GetNext(SqString t,int next[])

{ int j,k;

j=0;k=-1;next[0]=-1;

while (j<t.length-1)

{ if (k==-1 || t.data[j]==t.data[k])

{ j++;k++;

next[j]=k;

}

else k=next[k];

}

}

int KMPIndex(SqString s,SqString t)

{

int next[MaxSize],i=0,j=0;

GetNext(t,next);

while (i<s.length && j<t.length)

{ if (j==-1 || s.data[i]==t.data[j])

{ i++;

j++;

}

else j=next[j];

}

if (j>=t.length)

return(i-t.length);

else

return(-1);

}

SqString PreOrderSeq1(BTNode \*b)

{

SqString str, str1, leftstr, rightstr;

if(b==NULL)

{

StrAssign(str, "#");

return str;

}

str.data[0] = '@';

str.length = 1;

leftstr = PreOrderSeq1(b->lchild);

str1 = Concat(str, leftstr);

rightstr = PreOrderSeq1(b->rchild);

str = Concat(str1, rightstr);

return str;

}

bool compere(BTNode \*b1,BTNode \*b2)

{

SqString s1 = PreOrderSeq1(b1);

SqString s2 = PreOrderSeq1(b2);

if(KMPIndex(s1,s2)!=-1)

return true;

else

return false;

}

int main()

{

bool find;

BTNode \*b1, \*b2;

CreateBTree(b1,"A(B(D,E(,G)),C(,F(H,I)))");

printf("b1:");

DispBTree(b1);

printf("\n");

CreateBTree(b2,"C(,F(H,I))");

printf("b2:");

DispBTree(b2);

if(compere(b1, b2))

printf("\n有树形相同子树");

else

printf("\n没有树形相同子树");

return 1;

}