**Exp7.6**

#include<stdlib.h>

#include<stdio.h>

#define MaxSize 20

typedef struct node

{

    char data;

    struct node \*lchild;

    struct node \*rchild;

} BTNode;

int n1 = 0;

int n2 = 0;

void CreateBTree(BTNode \*&b,char \*str)*//创建二叉树函数*

{

    BTNode \*St[MaxSize], \*p;

    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 number1(BTNode \*b)*//求二叉树的结点个数*

{

    if(b!=NULL)

    {

        n1++;

        number1(b->lchild);

        number1(b->rchild);

    }

}

void number2(BTNode \*b)*//求叶子结点个数*

{

    if(b!=NULL)

        {

            if(b->lchild==NULL&&b->rchild==NULL)

            n2++;

            number2(b->lchild);

            number2(b->rchild);

        }

}

int number3(BTNode\*b,char x,int h)

{

    int L = 0;

    if(b==NULL)

        return 0;

    else if(b->data==x)

        return h;

    else

    {

        L=number3(b->lchild, x, h + 1);

        if(L!=0)

            return L;

        else

            return number3(b->rchild, x, h + 1);

    }

}

int number4(BTNode\*b)*//利用层次遍历求二叉树b的宽度*

{

    struct

    {

        int level;

        BTNode \*p;

    } Qu[MaxSize];

    int front = 0, rear = 0;

    int level=1;

    int n,i=1;

    int maxwigth;

    if(b!=NULL)

    {

        rear++;

        Qu[rear].p = b;

        Qu[rear].level = level;

        while(rear!=front)

        {

            front++;

            b = Qu[front].p;

            level = Qu[front].level;

            if(b->lchild!=NULL)

            {

                rear++;

                Qu[rear].p = b->lchild;

                Qu[rear].level = level + 1;

            }

            if(b->rchild!=NULL)

            {

                rear++;

                Qu[rear].p = b->rchild;

                Qu[rear].level = level + 1;

            }

        }

        maxwigth = 0;

        level = 1;

        while(i<=rear)

        {

            n = 1;

            while(Qu[i].level==level)

            {

                i++;

                n++;

            }

            if(n>maxwigth)

                maxwigth = n;

            level = Qu[i].level;

            i++;

        }

        return maxwigth;

    }

    else

        return 0;

}

int main()

{

    char str[50] = "A(B(D,E(H(J,K(L,M(,N))))),C(F,G(,I)))";

    BTNode \*b;

    int h=1;

    CreateBTree(b, str);

    number1(b);

    printf("二叉树B的结点个数为%d\n", n1);

    number2(b);

    printf("二叉树的叶子结点个数为%d\n", n2);

    char x;

    printf("想要求层次的结点值为\n");

    scanf(" %c", &x);

    printf("二叉树值为%c的结点的层次为%d\n",x,number3(b,x,h));

    printf("二叉树的宽度为%d\n", number4(b));

    return 1;

}

**Exp7.7**

#include<stdio.h>

#include<stdlib.h>

#define MaxSize 50

typedef struct node

{

    char data;

    struct node \*lchild;

    struct node \*rchild;

} BTNode;

void CreateBTree(BTNode \*&b,char \*str)*//创建二叉树函数*

{

    BTNode \*St[MaxSize], \*p;

    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 AllPath(BTNode \*b,char path[],int pathlen)

{

    if(b!=NULL)

    {

        if(b->lchild==NULL&&b->rchild==NULL)

        {

            printf("    %c叶子结点到根结点的逆路径为%c->", b->data, b->data);

            for (int i = pathlen - 1; i > 0; i--)

            {

                printf("%c->", path[i]);

            }

            printf("%c\n", path[0]);

        }

        else

        {

            path[pathlen] = b->data;

            pathlen++;

            AllPath(b->lchild, path, pathlen);

            AllPath(b->rchild, path, pathlen);

        }

    }

}

void firstlongpath(BTNode\*b,char path[],int pathlen,char longpath[],int &longpathlen)

{

   if(b==NULL)

    {

        if(longpathlen<pathlen)

        {

            for (int i = pathlen-1; i >= 0;i--)

            {

                longpath[i] = path[i];

            }

            longpathlen = pathlen;

        }

    }

        else

        {

            path[pathlen] = b->data;

            pathlen++;

            firstlongpath(b->lchild, path, pathlen, longpath, longpathlen);

            firstlongpath(b->rchild, path, pathlen, longpath, longpathlen);

        }

}

void pathback(BTNode\*b)

{

    BTNode \*st[MaxSize];

    int top = -1;

    BTNode \*p, \*r;

    bool flag;

    p = b;

    do

    {

        while(p!=NULL)

        {

            top++;

            st[top] = p;

            p = p->lchild;

        }

        r = NULL;

        flag = true;

        while(top>-1&&flag)

        {

            p = st[top];

            if(p->rchild==r)

            {

                if(p->lchild==NULL&&p->rchild==NULL)

                {

                    printf("   %c到根结点的逆路径:", p->data);

                    for (int i = top; i > 0;i--)

                        printf("%c->", st[i]->data);

                    printf("%c\n", st[0]->data);

                }

                top--;

                r = p;

            }

            else

            {

                p = p->rchild;

                flag = false;

            }

        }

    } while (top > -1);

}

void path2(BTNode \*b)

{

    struct snode

    {

        BTNode \*node;

        int parent;

    } Qu[MaxSize];

    int front, rear, p;

    front = rear = -1;

    rear++;

    Qu[rear].node = b;

    Qu[rear].parent = -1;

    while(rear>front)

    {

        front++;

        b = Qu[front].node;

        if(b->lchild==NULL&&b->rchild==NULL)

        {

            printf("叶子结点%c到根结点的逆路径为:", b ->data);

            p = front;

            while(Qu[p].parent!=-1)

            {

                printf("%c->", Qu[p].node->data);

                p = Qu[p].parent;

            }

            printf("%c\n", Qu[p].node->data);

        }

        if(b->lchild!=NULL)

        {

            rear++;

            Qu[rear].node = b->lchild;

            Qu[rear].parent = front;

        }

        if(b->rchild!=NULL)

        {

            rear++;

            Qu[rear].node = b->rchild;

            Qu[rear].parent = front;

        }

    }

}

int main()

{

    char str[50] = "A(B(D,E(H(J,K(L,M(,N))))),C(F,G(,I)))";

    BTNode \*b;

    int longpathlen = 0;

    CreateBTree(b, str);

    char path[MaxSize];

    char longpath[MaxSize];

    AllPath(b,path,0);

    firstlongpath(b,path,0,longpath,longpathlen);

    printf("最长的一条逆路径长度为%d\n", longpathlen);

    printf("最长的一条逆路径为:");

    for (int i=longpathlen-1; i >=0;i--)

    {

        printf("%c ", longpath[i]);

    }

    printf("\n");

    pathback(b);

    path2(b);

    return 1;

}

Exp7.8

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MaxSize 50

typedef struct node

{

    char data;

    struct node \*lchild;

    struct node \*rchild;

} BTNode;

double Comp(BTNode \*b)

{

    double v1, v2;

    if(b==NULL)

        return 0;

    if(b->lchild==NULL&&b->rchild==NULL)

        return b->data - '0';

    v1 = Comp(b->lchild);

    v2 = Comp(b->rchild);

    switch(b->data)

    {

        case '+':

            return v1 + v2;

        case "-":

            return (v1 - v2);

        case '\*':

            return v1 \* v2;

        case '/':

        if(v2!=0)

            return v1 / v2;

        else

            abort();

    }

}

BTNode\* CRTree(char str[],int i,int j)*//创建二叉树函数*

{

    BTNode \*p;

    int k, plus = 0, posi;

    if (i = j)

    {

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

        p->data = str[i];

        p->lchild = p->rchild = NULL;

        return p;

    }

    for (k = i; k <= j;k++)

        if(str[k]=='+'||str[k]=='-')

        {

            plus++;

            posi = k;

        }

        if(plus==0)

            for (k = i; k <= j;k++)

            {

                if(str[k]=='\*'||str[k]=='/')

                {

                    plus++;

                    posi = k;

                }

            }

        if(plus!=0)

        {

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

            p->data = str[posi];

            p->lchild = CRTree(str, i, posi - 1);

            p->rchild = CRTree(str, posi + 1, j);

            return p;

        }

        else

            return NULL;

}

int main()

{

    BTNode \*b;

    char str[MaxSize] = "1+2\*3-4/5";

    printf("算术表达式为:%s\n", str);

    b=CRTree(str,0,strlen(str)-1);

    printf("二叉树算术表达式的值为:%g\n", Comp(b));

    return 1;

}