支持功能：

**核心类**

**BiTreeNode 链式二叉树**

**核心函数**

void CreateTree(const char str[])

创建二叉树

格式例：A(B(C),D(,E))

void PreOrderTraverse()

void InOrderTraverse()

void PostOrderTraverse()

void LevelOrderTraverse()

前、中、后、层序遍历

void DispTree()

打印二叉树

打印格式与void CreateTree函数一致

int Depth()

求深度

int LeafCount()

求叶子节点数

**辅助函数**

BiTreeNode(const ElemType \_data = '\0')

默认构造函数

void DestoryTree()

销毁二叉树

bool ifLeaf()

判断是否是叶子节点

BiTreeNode\* copy()

复制二叉树

void buildFullTree(int depth = -1)

构造满二叉树

当参数输入为默认参数-1时

则以当前树层数填满树

void clearEmptyNode()

清除空节点

**OrderTree 顺序二叉树**

**核心函数**

OrderTree(BiTreeNode\* root = NULL)

void Create(BiTreeNode\* root)

构造函数

输入一个链式树 构建顺序树

void output()

顺序输出元素

空节点输出为 #

int Depth()

求深度

BiTreeNode\* ToBiTree()

转化为链式二叉树

void LevelOrder()

层序遍历

void PreOrder (int depth = -1, NodeLoc\* nl = new NodeLoc)

void InOrder (int depth = -1, NodeLoc\* nl = new NodeLoc)

void PostOrder(int depth = -1, NodeLoc\* nl = new NodeLoc)

先、中、后序遍历深度为depth的二叉树

depth为-1时默认遍历整个二叉树

**辅助函数**

void push(const ElemType elem)

在尾部添加节点

void destory()

摧毁顺序树

OrderTree(BiTreeNode\* root = NULL)

默认构造函数

**类ExpressionTreeNode 前缀表达式树**

核心函数

void BuildExpressionTree(const char str[])

构造表达式树

格式例： 10+5\*(6-2)/2

void ExpressionTreeTraverse()

遍历表达式树

格式与构造表达式树一致

int CalcExpressionTree()

计算表达式树

返回计算结果

**辅助类**

**类NodeLoc 二叉树位置编码**

储蓄顺序二叉树路径编码

0为左孩子 1为右孩子

**核心函数**

int LocNum()

返回路径对应的顺序二叉树位置（下标）

NodeLoc\* child(const Branch b)

返回指向子集的路径

**辅助函数**

NodeLoc(Branch b[] = NULL, unsigned int n = 0)

初始化类

unsigned int size()

返回路径长度

void clear()

清除路径

void output()

以0与1的形式顺序输出编码

void push(const Branch b)

在尾缀添加分支（0或1）

bool empty()

判断路径是否为空

NodeLoc operator+(const Branch b)

重置运算符+ 功能与void push函数一致

void operator+=(const Branch& b)

重置运算符+= 功能与void push函数一致

示范代码：

//示范代码1 二叉树基础

//A(B(C),D(,E))

int main(){

    BiTreeNode\* root = new BiTreeNode(); //创建一个根节点

    string str;

    cin >> str;

    root->CreateTree(str.data());

    cout << "PreOrderTraverse:      ";

    root->PreOrderTraverse();

    cout << endl;

    cout << "InOrderTraverse:       ";

    root->InOrderTraverse();

    cout << endl;

    cout << "PostOrderTraverse:     ";

    root->PostOrderTraverse();

    cout << endl;

    cout << "LevelOrderTraverse:    ";

    root->LevelOrderTraverse();

    cout << endl;

    cout << "Tree's Depth:          ";

    cout << root->Depth() << endl;

    cout << "Leaf's Count:          ";

    cout << root->LeafCount() << endl;

    return 0;

}

示范输入：

A(B(C),D(,E))

示范输出：

PreOrderTraverse: A B C D E

InOrderTraverse: C B A D E

PostOrderTraverse: C B E D A

LevelOrderTraverse: A B D C E

Tree's Depth: 3

Leaf's Count: 2

//示范代码2 链式与顺序的互相转换

//A(B(C),D(,E))

int main(){

    BiTreeNode\* root = new BiTreeNode();

    string str;

    cin >> str;

    root->CreateTree(str.data());

    cout << "Old List's DispTree:   ";

    root->DispTree();

    cout << endl;

    OrderTree ot(root);

    root->DestoryTree();

    cout << "Data in Array:         ";

    ot.output();

    root = ot.ToBiTree();

    cout << "New List's DispTree:   ";

    root->DispTree();

    cout << endl;

    system("pause");

    return 0;

}

示范输入

A(B(C),D(,E))

示范输出

Old List's DispTree: A(B(C),D(,E))

Data in Array: ABDC##E

New List's DispTree: A(B(C),D(,E))

//示范代码3 顺序二叉树遍历

//A(B(C),D(,E))

int main(){

    BiTreeNode\* root = new BiTreeNode();

    string str;

    cin >> str;

    root->CreateTree(str.data());

    OrderTree ot(root);

    cout << "OtOutPut:          ";

    ot.output();

    cout << "PreOrderTraverse:  ";

    ot.PreOrder();

    cout << endl;

    cout << "InOrderTraverse:   ";

    ot.InOrder();

    cout << endl;

    cout << "PostOrderTraverse: ";

    ot.PostOrder();

    cout << endl;

    system("pause");

    return 0;

}

示范输入

A(B(C),D(,E))

示范输出

OtOutPut: ABDC##E

PreOrderTraverse: ABCDE

InOrderTraverse: CBADE

PostOrderTraverse: CBADE

//示范代码4 前缀表达式的创建、输出、计算

//10+5\*(6-2)/2

int main(){

    ExpressionTreeNode\* root = new ExpressionTreeNode();

    string str;

    cin >> str;

    root->BuildExpressionTree(str.data());

    cout << "ExpressionTreeTraverse:    ";

    root->ExpressionTreeTraverse();

    cout << endl;

    cout << "Result of the Formula:     ";

    cout << root->CalcExpressionTree() << endl;

    system("pause");

    return 0;

}

示范输入

10+5\*(6-2)/2

示范输出

ExpressionTreeTraverse: 10+5\*(6-2)/2

Result of the Formula: 20