**安徽师范大学**

**数学计算机科学学院实验报告**

|  |  |
| --- | --- |
| **年级专业** | **2014级计算机科学与技术** |
| **实验地点** | **2060401** |
| **实验课程** | **数据结构实验** |
| **实验名称** | **数据结构实验8** |
| **姓 名** | **王昊** |
| **学 号** | **14111204056** |
| **实验日期** | **2015-12-10** |

**实验目的和要求**

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】

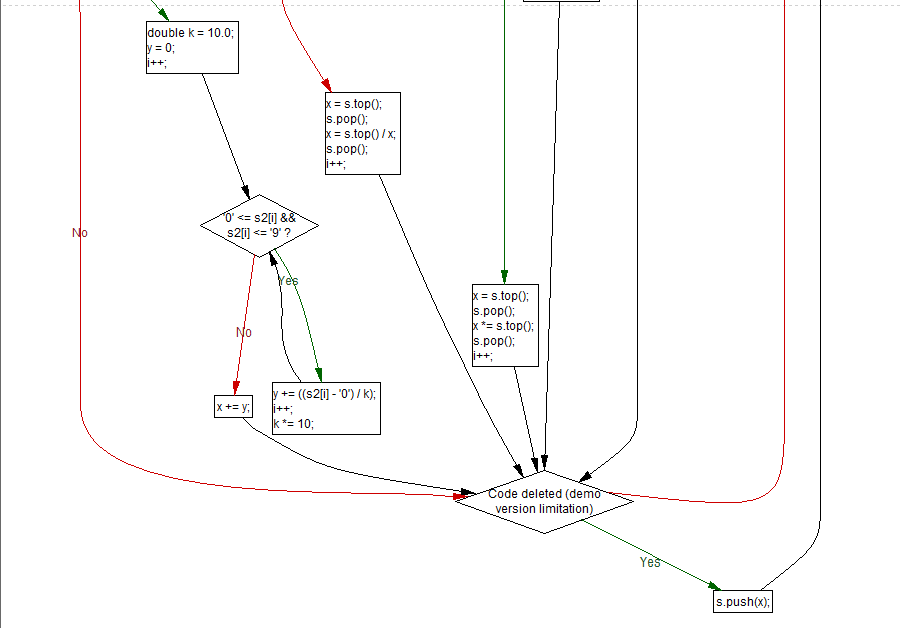
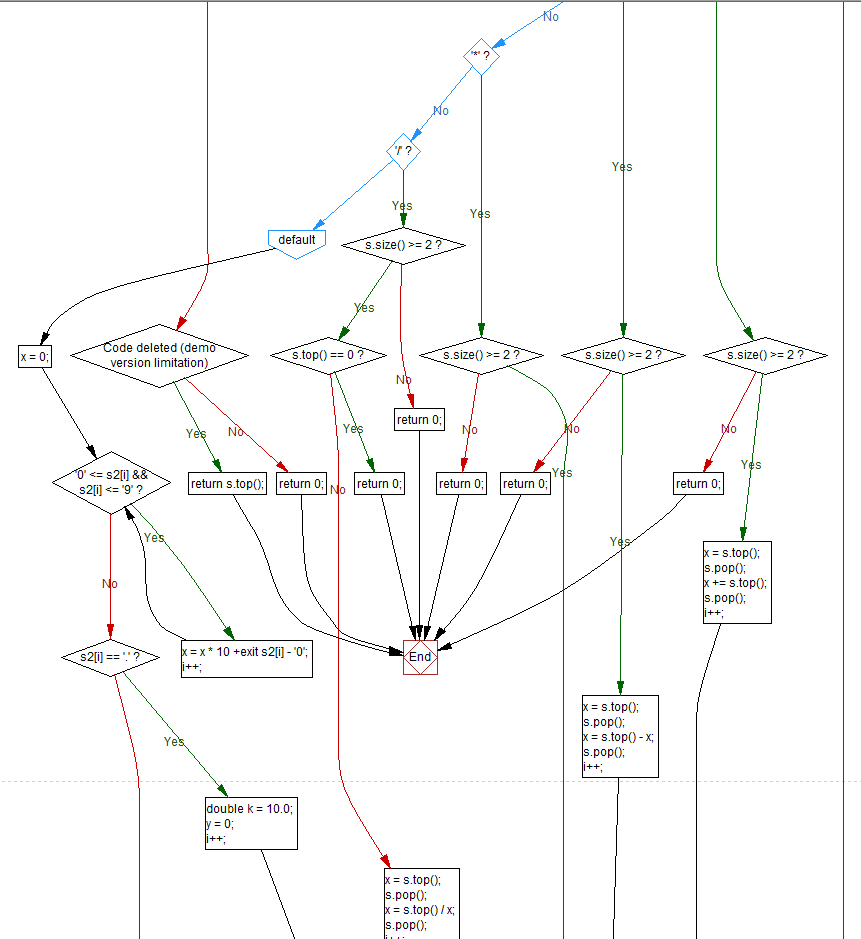
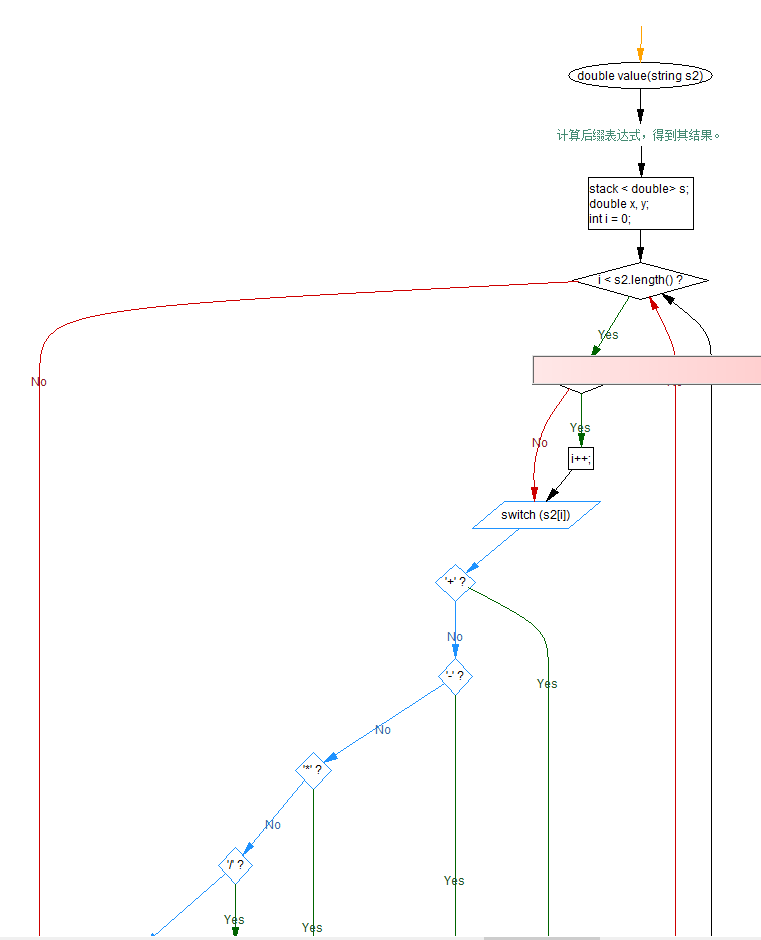
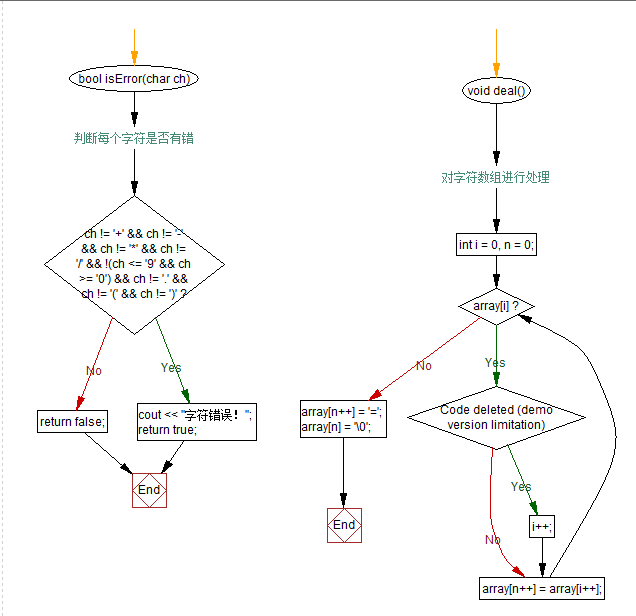
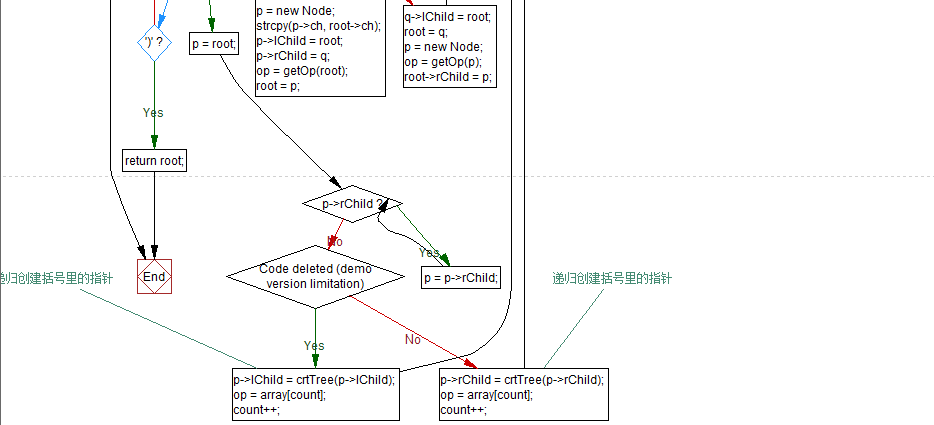
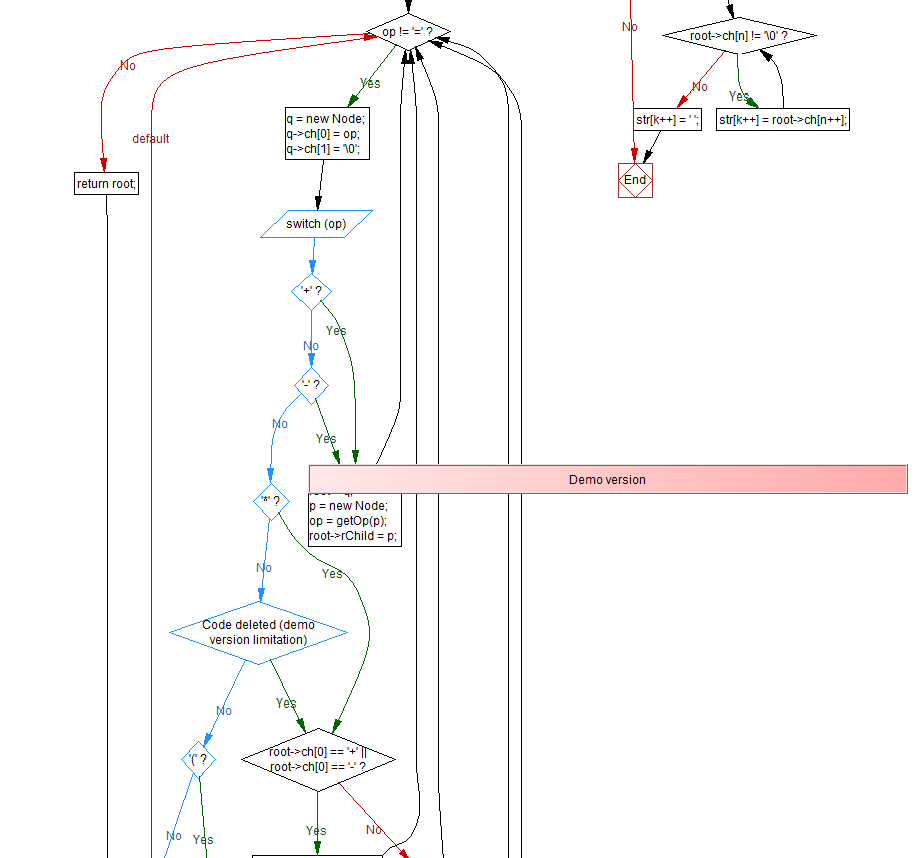
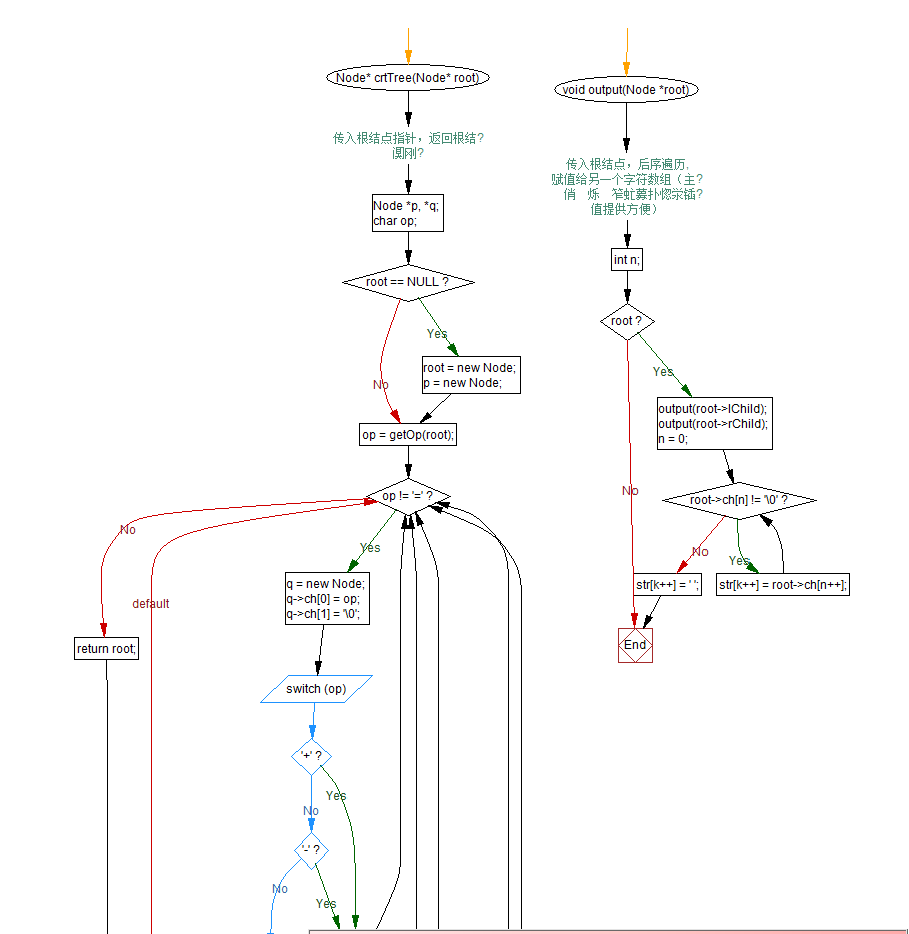
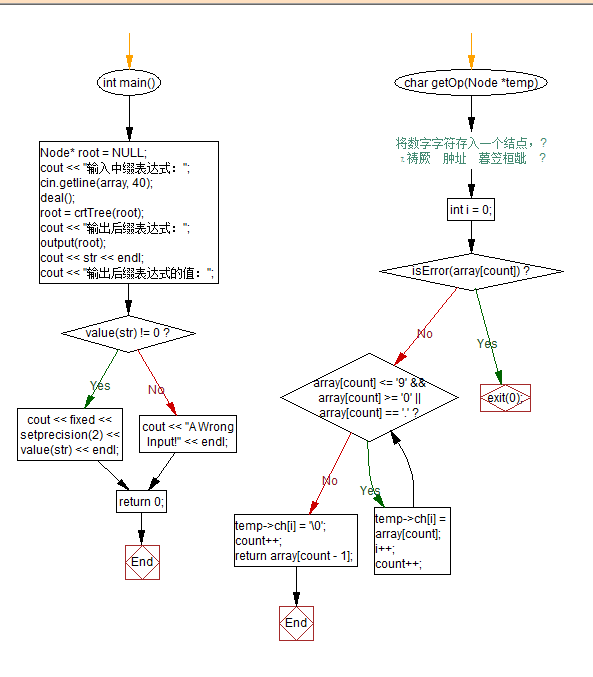
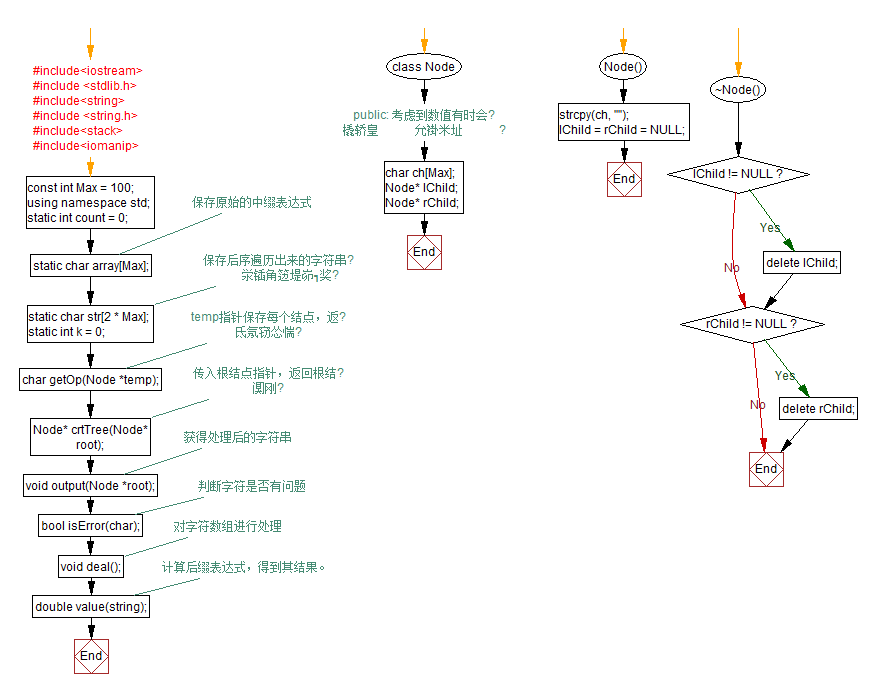
|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

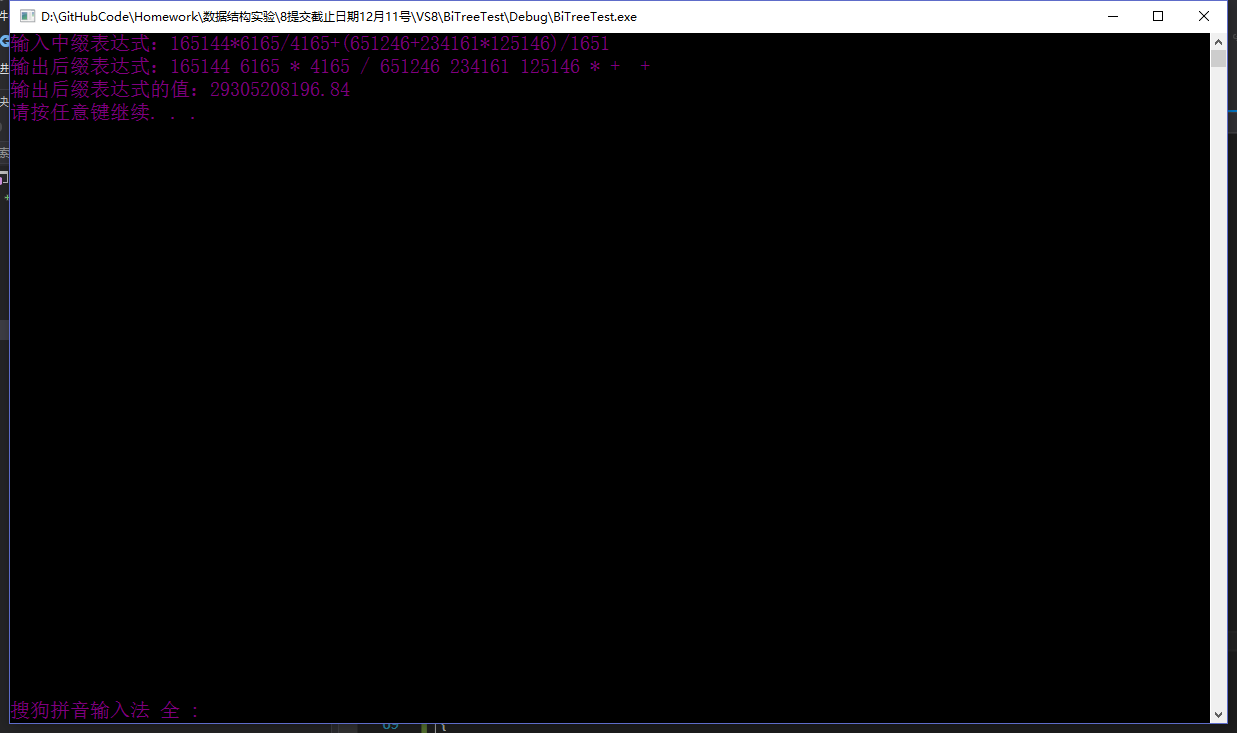


1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】



|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】

#include<iostream>

#include <stdlib.h>

#include<string>

#include <string.h>

#include<stack>

#include<iomanip>

const int Max = 100;

using namespace std;

class Node

{

public:

char ch[Max]; //考虑到数值有时会是两位数，所以使用字符串数组

Node\* lChild;

Node\* rChild;

Node()

{

strcpy(ch, "");

lChild = rChild = NULL;

}

~Node()

{

if (lChild != NULL) {

delete lChild;

}

if (rChild != NULL) {

delete rChild;

}

}

};

static int count = 0;

static char array[Max]; //保存原始的中缀表达式

static char str[2 \* Max]; //保存后序遍历出来的字符串，为表达式求值提供方便

static int k = 0;

char getOp(Node \*temp); //temp指针保存每个结点，返回的是运算符

Node\* crtTree(Node\* root); //传入根结点指针，返回根结点指针

void output(Node \*root); //获得处理后的字符串

bool isError(char); //判断字符是否有问题

void deal(); //对字符数组进行处理

double value(string); // 计算后缀表达式，得到其结果。

int main()

{

Node\* root = NULL;

cout << "输入中缀表达式：";

cin.getline(array, 40);

deal();

root = crtTree(root);

cout << "输出后缀表达式：";

output(root);

cout << str << endl;

cout << "输出后缀表达式的值：";

if (value(str) != 0) {

cout << fixed << setprecision(2) << value(str) << endl;

} else {

cout << "A Wrong Input!" << endl;

}

return 0;

}

//将数字字符存入一个结点，并返回数字字符的后一个符号

char getOp(Node \*temp)

{

int i = 0;

if ( isError(array[count]) ) {

exit(0);

}

while (array[count] <= '9' && array[count] >= '0' || array[count] == '.') {

temp->ch[i] = array[count];

i++;

count++;

}

temp->ch[i] = '\0';

count++;

return array[count - 1];

}

//传入根结点指针，返回根结点指针

Node\* crtTree(Node\* root)

{

Node \*p, \*q;

char op;

if (root == NULL) {

root = new Node;

p = new Node;

}

op = getOp(root);

while (op != '=') {

q = new Node;

q->ch[0] = op;

q->ch[1] = '\0';

switch (op) {

case '+':

case '-':

q->lChild = root;

root = q;

p = new Node;

op = getOp(p);

root->rChild = p;

break;

case '\*':

case '/':

if (root->ch[0] == '+' || root->ch[0] == '-') {

p = new Node;

strcpy(p->ch, root->ch);

p->lChild = root;

p->rChild = q;

op = getOp(root);

root = p;

} else {

q->lChild = root;

root = q;

p = new Node;

op = getOp(p);

root->rChild = p;

}

break;

case '(':

p = root;

while (p->rChild) {

p = p->rChild;

}

if (p->lChild == NULL) {

p->lChild = crtTree(p->lChild); //递归创建括号里的指针

op = array[count];

count++;

break;

} else {

p->rChild = crtTree(p->rChild); //递归创建括号里的指针

op = array[count];

count++;

break;

}

case ')':

return root;

}

}

return root;

}

//传入根结点，后序遍历,赋值给另一个字符数组（主要是为了给后序的计算表达式值提供方便）

void output(Node \*root)

{

int n;

if (root) {

output(root->lChild);

output(root->rChild);

n = 0;

while (root->ch[n] != '\0') {

str[k++] = root->ch[n++];

}

str[k++] = ' ';

}

}

bool isError(char ch) //判断每个字符是否有错

{

if (ch != '+' && ch != '-' && ch != '\*' && ch != '/' && !(ch <= '9' && ch >= '0') && ch != '.' && ch != '(' && ch != ')') {

cout << "字符错误！";

return true;

}

return false;

}

void deal() //对字符数组进行处理

{

int i = 0, n = 0;

while (array[i]) {

if (array[i] == ' ' || array[i] == '=') {

i++;

}

array[n++] = array[i++];

}

array[n++] = '=';

array[n] = '\0';

}

double value(string s2) // 计算后缀表达式，得到其结果。

{

stack < double> s;

double x, y;

int i = 0;

while (i < s2.length() ) {

if (s2[i] == ' ') {

i++;

}

switch (s2[i]) {

case '+':

if (s.size() >= 2) {

x = s.top();

s.pop();

x += s.top();

s.pop();

i++;

break;

} else {

return 0;

}

case '-':

if (s.size() >= 2) {

x = s.top();

s.pop();

x = s.top() - x;

s.pop();

i++;

break;

} else {

return 0;

}

case '\*':

if (s.size() >= 2) {

x = s.top();

s.pop();

x \*= s.top();

s.pop();

i++;

break;

} else {

return 0;

}

case '/':

if (s.size() >= 2) {

if ( s.top() == 0) {

return 0;

} else {

x = s.top();

s.pop();

x = s.top() / x;

s.pop();

i++;

break;

}

} else {

return 0;

}

default :

x = 0;

while ('0' <= s2[i] && s2[i] <= '9') {

x = x \* 10 + exit s2[i] - '0';

i++;

}

if (s2[i] == '.') {

double k = 10.0;

y = 0;

i++;

while ('0' <= s2[i] && s2[i] <= '9') {

y += ((s2[i] - '0') / k);

i++;

k \*= 10;

}

x += y;

}

break;

}

if (x != 0) {

s.push(x);

}

}

if ( s.size() == 1 ) {

return s.top();

} else {

return 0;

}

}

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】

1. **概要设计**

【定义所有抽象数据类型、自定义函数间的调用关系图，自定义函数的功能描述和流程图，以及主程序的流程图。】

1. **调试分析**

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

1. **测试数据与结果**

【列出你的测试结果，包括输入和输出。测试数据应该完整和严格，最好多于需求分析中所列。】

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **输入** | **输出** | **说明** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

1. **附录**

【给出每部分的源代码（必须要有一定量的注释）。】