



|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 18040400054 | **姓名** | **周师扬** |
| **班级** | **1803054** | **任课教师** | **张淑平** |
| **实验名称** | 类的基本概念与编程 | | |
| **实验学期** | **2019 – 2020 学年第2学期** | | |
| **实验日期** | 选择日期 | **实验地点** |  |
| **报告成绩** |  | | |

西安电子科技大学计算机科学与技术学院

# 实验目的

[使用“类”相关机制来设计并实现一些程序，以熟悉 C++提供的面向对象基本概念和机制，掌握数据抽象的基本手段，用类型上的操作来封装数据结构，为面向对象程序设计奠定基础。]

# 实验环境

操作系统：[MacOS]

开发工具：[Xcode，clion]

# 实验内容

**定义表示二叉树及其结点的类型**

Rewrite Tnode from §7.10[7] as a class with constructors, destructors, etc.

Define a tree of Tnodes as a class with constructors, destructors, etc.

## **定义算术表达式类型**

Define a class for analyzing, storing, evaluating, and printing simple arithmetic expressions consisting of integer constants and the operators +, -, \*, and /. The public interface should look like this:

class Expr {

// …

public:

Expr(char\*);

double eval();

void print();

};

The string argument for the constructor Expr::Expr() is the expression. The function Expr::eval() returns the value of the expression, and Expr::print() prints a representation of the expression on cout. A program might look like this:

Expr x("123.2/(4+123)\*43.8");

cout << "x = " << x.eval() << "\ n ";

x.print();

Experiment with different ways of printing the expression: fully parenthesized, postfix notation, prefix notation,etc.

# 数据结构与算法设计

**定义表示二叉树及其结点的类型**

**模块结构及文件组织设计：**

模块1：主控模块，仅包括文件main.cpp ，定义了 main()函数。

**关键数据结构设计：**

数据结构1：定义了一个Tnode类，用于存储树中节点。

class Tnode {  
 string word;  
 int count;  
 Tnode \*left;  
 Tnode \*right;  
public:  
 Tnode(string data = "=0");  
 void showTnode();  
 Tnode \*get\_Left();  
 Tnode \*get\_Right();  
 string get\_Word();  
 void set\_Left(Tnode \*);  
 void set\_Right(Tnode \*);  
 void add\_Count();  
};

数据结构2:定义了一个Tree类，用于实现一些有关于树的操作，给用户提供操作接口

class Tree {  
 Tnode \*root;  
public:  
 Tree();  
 ~Tree();  
 bool searchNode(Tnode \*bst, string key, Tnode \*f, Tnode \*\*p);  
 bool regist(string key);  
 void print();  
};

**算法1.1 int main()**

作 用：主控函数，也实现对题目所需其他内容的测试。

参 数：无参数。

返回值：总是返回0。

计算过程：

生成一个Tree类型的指针指向一个Tree类型的节点，向树中存储信息并且进行有序化输出，测试相关函数。

**算法1.2 bool Tree::searchNode()**

作 用：在树中寻找key值对应的节点，有则返回空，无则返回最终位置。

参 数：Tnode \*bst, string key, Tnode \*f, Tnode \*\*p

返回值：true或false

计算过程：

1. 检测根节点是否为空，若为空，则直接返回f指针NULL，并且返回false；
2. 如果根节点不为空，则检测该节点的word值是否为查找的key值，如果是，则返回true，直接返回f指针为根节点；如果key值大于word，则向右子树查找，否则，向左子树查找。

**算法1.3 bool Tree::regist()**

作 用：向树中贮存节点信息，无则存储，有则对应节点的计数器+1

参 数：string key

返回值：true或false。

计算过程：

1. 检测当前树的根节点是否为空，如果是，则立即申请一个新的节点并且将key值贮存在其中；
2. 如果根节点不为空，则点用算法1.2的搜索函数进行搜索，如果找到，则对应的节点的count值+1，如果没有找到，则在最终位置申请一个新节点，初始化word为key值并且插入该位置；

**算法1.4 void Tree::print()**

作 用：中序打印当前树

参 数：无参数

返回值：无返回值。

计算过程：

1. 检测当前树是否为空，如果为空，则立即输出“The tree is empty！”，然后停止输出；
2. 如果不为空，则一直将左子树压栈，直到没有左子树，然后弹栈进行输出节点，再将右子树压栈直到没有右子树，然后弹栈输出节点，直到栈为空，停止输出；

**算法1.4 Tree::~Tree()**

作 用：析构销毁当前树

参 数：无参数

返回值：无返回值。

计算过程：

1. 判断当前树是否为空，如果是，则停止销毁；
2. 如果不为空，则一直将左子树压栈，直到没有左子树，然后弹栈进行销毁节点，再将右子树压栈直到没有右子树，然后弹栈销毁节点，直到栈为空，停止销毁；
3. 销毁用于读取的指针和根节点；

## **定义定义算数表达式类型**

**模块结构及文件组织设计：**

模块1：主控模块：仅包含文件main.cpp，定义了main()函数，用于测试所编写的头文件。

模块2：类文件模块：包含Expr.cpp、Expr.h和Tnode.h

Expr()：析构函数

void change()：转换函数

Tnode \*build\_tree()：生成树函数

double eval()：返回值函数

void cal()：分析函数

void print()：打印函数

void print\_tree()：打印树函数

void print\_Postorder\_tree()：前序遍历函数

void print\_Proorder\_tree()：后序遍历函数

void print\_Midorder\_tree()：中序遍历函数

string lexer(char str[])：语法解析函数

**关键数据结构设计：**

数据结构1:class Expr {  
private:  
 Tnode translation[100];  
 int count;  
 Tnode \*head = new Tnode;  
 stack<double> num;  
 char str[100];

......

}

类Expr中的变量，用于记录转换的表达式与求值

数据结构2:class Tnode {  
private:  
 Tnode \*Lchild;  
 Tnode \*Rchild;  
 char sign;  
 double num;

......

}

类Tnode中的变量，用于存储遍历树的信息，同时也记录了部分生成的值

**算法2.1 int main()**

作 用： 测试编写的头文件

参 数： 无参数

返回值： 总是返回0

计算过程：

1. 输入一个表达式
2. 测试树的前序遍历、中序遍历和后序遍历
3. 测试树的求值功能

**算法2.2 void change()**

作 用： 转换string字符串为并且放到translation中

参 数： string const str

返回值： 无返回值

计算过程：

1. 设置一个Tnode类型的变量temp、15长度的字符数组和一个标记值。
2. 对字符串进行翻译转换并且number数组末位置零。
3. 设置char值和number值为零。

**算法2.3 void cal()**

作 用： 对生成树进行计算

参 数： stack<double> &num, Tnode \*T

返回值： 无返回值

计算过程：

1. 向左向右遍历直到节点为空。
2. 将对应节点的number值入栈。
3. 全部完成后弹栈并且进行计算。

**算法2.4 string Lexer()**

作 用： 分析器

参 数： char s[]

返回值： string

计算过程：

1. 分析s[]。
2. 对各类函数进行分析，如果无误，调用insert函数进行插入，如果有错误，则输出错误”error”，并且停止运算。
3. 返回字符串str。

**算法2.4 void Expr::print\_Postorder\_tree()**

**void Expr::print\_Proorder\_tree()   
 void Expr::print\_Midorder\_tree()**

作 用： 前序中序后序遍历树

参 数： Tnode \*t

返回值： 无返回值

计算过程：

1. 如果根节点为空，则返回空树。
2. 如果根节点不为空，则进行前序/中序/后序遍历并且输出值。

**算法2.5 double Expr::eval()**

作 用：利用后缀表达式求值

参 数：无

返回值：无

计算过程：

（1） 当队列q不为空时，取出队首元素，判断其flag，若是数字，则压入栈s中，否则取出栈s顶上的两个数字，进行q队首的操作，并将结果再次压入栈中

（2） 最后返回栈顶的值

# 测试用例与测试结果

**定义表示二叉树及其结点的类型**

**测试用例：**

myTree->regist("peach");  
myTree->regist("banana");  
myTree->regist("apple");  
myTree->regist("pear");  
myTree->regist("apple");  
myTree->regist("pineapple");

**输出结果：**

按名字小计:

apple 2

banana 1

peach 1

pear 1

pineapple 1

**定义算术表达式**

**测试用例：**

(1+2)\*3

1\*(3+2\*3)

2+3/2+4\*(2/3)

**输出结果：**

\*+123

(((1)+(2))\*(3))

12+3\*

9

\*1\*+323

((1)\*(((3)+(2))\*(3)))

132+3\*\*

15

/+23\*+24/23

(((2)+(3))/(((2)+(4))\*((2)/(3))))

23+24+23/\*/

1.25

# 实验总结

这次实验主要是对类的理解和使用。

其中第一道题是二叉搜索树，在之前的实验中已经实现过，但这次是使用类来实现，又开发了新的思路，通过对同一个问题的多种方法实现，可以帮助我更好的理解各种算法，通过树和结点的类来实现封装，对用户使用来说更加方便，也增加了代码的可读性和封装性。 其中第二道题是计算器的再次实现，在这道题中我使用了string和stack，学会了利用压栈的方式计算和遍历前缀、中缀、后缀表达式，得到了很多的锻炼。第一次使用类来实现计算器，有不少的难点出现，今后还需要多加练习，不仅加深对计算器的理解，也加强对类编程的理解。