



|  |  |  |  |
| --- | --- | --- | --- |
| **学号** | 17130130312 | **姓名** | 曹晨瑶 |
| **班级** | **1713013** | **任课教师** | **王献青** |
| **实验名称** | 类的基本概念与编程 | | |
| **实验学期** | **2018 – 2019 学年第2学期** | | |
| **实验日期** | 2019年5月25日 | **实验地点** | **G334** |
| **报告成绩** |  |  |  |

西安电子科技大学软件学院

# 实验目的

本次实验要求使用“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：Windows10

开发工具: Code::Blocks

# 实验内容

## 定义并实现日期类型

Complete and test Date.

Reimplement it with ‘‘number of days after 1/1/1970’’ representation.

## 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.

## 定义算术表达试类型

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

## 定义并实现类型日期

**模块结构及文件组织设计：**

模块1：仅包括main.cpp文件,定义了main（）函数，对日期的相关操作进行了测试。

模块2：仅包含Tnode.hpp文件，定义了二叉树的结点类型Tnode

模块3：仅包含Tnoode.cpp文件，实现二叉树的结点类型Tnode

模块4：仅包含BSTree.hpp 文件，定义了二叉排序树类型 BSTree

模块5：仅包含BSTree.cpp文件，实现了二叉排序树类型BSTree。

**关键数据结构设计：**

class Date{

public:

//缺省处理

Date(int y,int m,int d);//日、月、年 初始化

Date(int m, int d);//日、月、本年

Date(int dd);//日、本年、本月

Date();//本日本月本年

//修改Date的函数

void add\_day(int n);//加n天

void add\_month(int n);//加n月

void add\_year(int n);//加n年

void print\_date();//日期输出

private:

int days;//自1970年1月1日以来的天数

};

**算法1.1 int main()**

作 用：主控函数,，测试函数，测试相关操作

参 数：无

返回值：总是返回0。

计算过程：

1. 调用算法1.2，传入规范（包括年月日）参数
2. 调用算法1.3
3. 调用算法1.2，传入只有年月的参数
4. 调用算法1.4
5. 调用算法1.2，传入只有日的参数
6. 调用算法1.5
7. 调用算法1.2（无年月日）
8. 返回0.

**算法1.2 Date()**

作 用：构造函数，初始化，将输入日期转化距离1970.1.1的天数并输出日期

参 数：根据传入参数数目的不同，选择不同参数数目的初始化

返回值：无

计算过程：

1. 首先判断输入是否合法，再进行缺省处理（若缺少年，设置为本年，缺少月，设置为本月，缺少日，设置为本日）
2. 调用算法1.9并赋值给days
3. 调用算法1.6

**算法1.3 void add\_day()**

作 用：给当前日期的天数加上n天

参 数：int n

返回值：无

计算过程：

1. 给天数days 加n
2. 输出add：n 天
3. 调用算法1.6

**算法1.4 void add\_month()**

作 用：给当前日期的天数加上n月

参 数：int n

返回值：无

计算过程：

1. 定义 y, m, d
2. 调用算法1.9转化为规范日期，赋值给y,m,d
3. 月份从当前月份开始逐次加1直到加了n个月（若当前月份为12月，则下一个月为从1月开始，同时年份加1）
4. 若旧月为2，日为29或28；判断新年是否为闰年，是，则新日为29，不是为28
5. 调用算法1.6

**算法1.5 void add\_year()**

作 用：给当前日期的天数加上n年

参 数：int n

返回值：无

计算过程：

1. 定义 y, m, d
2. 调用算法1.9转化为规范日期，赋值给y,m,d
3. 年份加n
4. 若旧月为2，日为29或28；判断新年是否为闰年，是，则新日为29，不是为28；月不变
5. 调用算法1.6

**算法1.6 void print\_date()**

作 用：输出规范日期

参 数：无

返回值：无

计算过程：

1. 定义表示年、月、日的int型整数
2. 调用算法1.8获取年月日信息。
3. 打印日期信息。

**算法1.7 int isleapyear()**

作 用：判断是否为闰年

参 数：int year

返回值：1或者0

计算过程：

1. 如果能被4整除而不能被100整除的为闰年，返回1
2. 否则返回0

**算法1.8** **void days\_to\_date()**

作 用：将表示日期的天数转化为日期

参 数：int new\_days（天数）,int\* y,int\* m,int\* d

返回值：无

计算过程：

1. 循环1 从1970年开始，判断是否为闰年（若是，天数大于366，则天数减去366，年份加1，天数小于366，停止循环；若不是，天数大于365，则天数减去365，年份加1，天数小于365，停止循环），直到天数小于366（闰年）或者365（平年），获得年
2. 循环2 从1月开始，（若是1、3、5、7、8、10、12月，减去31天；若是2月，根据年份判断2月天数，减去28或者29天；其他月份减去30天），月份加1，判断下一个月份是天数，直到总天数小于下一个月的天数，获得月，日
3. 将获得的年月日赋值给\*y \*m \*d

**算法1.9 int date\_to\_days()**

作 用：将日期转化为天数

参 数：int y,int m,int d

返回值：整型 new\_days（表示距离1970.1.1的天数）

计算过程：

1. 定义int new\_days并初始化
2. 循环1 从1970年开始到当前年份的前一年，若为闰年，天数加366，若为平年，天数加 355
3. 循环2 从1月开始到当前月份的前一个月，如果有闰年的2月加29天，否则加28天；若是1、3、5、7、8、10、12月，天数加31天；若是4、6、9、11月，则天数加30天
4. 天数加当前表示的日并减1（1970.1.1为0）

## 定义表示二叉树及其节点的类型

**模块结构及文件组织设计：**

模块1：仅包括main.cpp文件,定义了main（）函数。

模块2：仅包含function.hpp文件，对regist\_word()、deleteBST()、search()、print\_tree()、print\_tree\_byorder()四个函数进行了声明。

模块3：仅包含function.cpp文件，对regist\_word()、deleteBST()、search()、print\_tree()、print\_tree\_byorder()四个函数进行了功能实现。

**关键数据结构设计：**

class Tnode

{

public: // public interface for operations

Tnode(const char\* str =0); //default constructor

Tnode(const std::string& str);

virtual ~Tnode(); // this 'virtual' is introduced in Section 15.6 of Text book

// 仅输出当前结点自身的信息。而整颗树的输出则由 BSTree 负责.

// 参数 outFile 为输出目标文件

void print(std::ostream& outFile) const;

public:

void setWord(const std::string& val) { word = val; }

const std::string& getWord() const { return word; }

std::string& getWord() { return word; }

void setLeft(Tnode\* val) { left = val; }

// return the pointer to const left child of const Tnode

const Tnode\* getLeft() const { return left; }

// return the non-const reference to left child of non-const Tnode

Tnode\*& getLeft() { return left; }

void setRight(Tnode\* val) { right = val; }

const Tnode\* getRight() const { return right; }

Tnode\*& getRight() { return right; }

void setCount(int val) { count = val; }

int getCount() const { return count; }

int& getCount() { return count; }

private: //这两个对象复制操作声明为私有的，以阻止 Tnode 对象被复制.

Tnode(const Tnode&); // 仅需声明，无需定义，因为没有使用这两个函数

Tnode& operator=(const Tnode&);

private:

std::string word; // the text in this node

int count; // appear times of word in input

Tnode\* left; // left child

Tnode\* right; // right child

};

class BSTree

{

public:

BSTree();

virtual ~BSTree();

void destroy();//销毁树中的结点

bool regist(const std::string& newWord);

bool regist(const char\* newWord);

private: //这两个操作定义为私有的，以阻止 BSTree 对象进行复制.

BSTree(const BSTree&);// 仅需声明，无需定义，因为没有使用这两个函数

BSTree& operator=(const BSTree&);

private:

Tnode\* Root;//

bool deleteBST(Tnode\* root);

public:

//辅助函数

int regist\_word(Tnode\* & root,const std::string& newWord);

void print\_pre\_order(const Tnode\* root,unsigned int indent );

void print\_middle\_order(const Tnode\* root);

};

**算法2.1 int main()**

作 用: 主控函数 输入数据、初始化、输出结果

参 数： 无参数

返回值： 0

计算过程：

1. 建立一个树，缺省处理
2. 循环输入新文本
3. 调用算法2.4
4. 调用算法2.5
5. 调用算法2.6

**算法2.2 Tnode::Tnode()**

作 用：结点成员初始化

参 数：const char\* str或者const string& str

返回值：无

计算过程：

1. 将 word 的值从 初始的空串 修改为 str 相同的值
2. 将 count 的值从 初始的随机值 修改为 1
3. 左右子树为空子树

**算法2.3 bool BSTree::regist()**

作 用：建立树

参 数：const string&/char\* newWord

返回值：true或者 false

计算过程

1. 忽略空串，返回false
2. 调用算法2.4，将文本插入树的合适位置
3. 返回true

**算法2.4 int BSTree::regist\_word()**

作 用：给树插入数据（辅助函数）

参 数：Tnode\* & root, const string& newWord

返回值：0

计算过程：

1. 若结点为空，建立新结点
2. 若树不为空，查找树中是否已经存入
3. 若没有存入相同的数据，小值放到左孩子，递归调用函数本身
4. 大值放到右孩子，递归调用函数本身
5. 返回0

**算法2.5** **void BSTree::print\_pre\_order()**

作 用：先序遍历 打印信息

参 数：const Tnode\* root,unsigned int indent

返回值：无

计算过程：

1. 若树为空，则终止函数；
2. 打印若干空格，向右缩进；
3. 输出根结点；
4. 递归调用，打印左子树；
5. 若根结点只有左子树没有右子树，向右缩进打印<NULL>;
6. 递归调用，打印右子树.

**算法2.6 void BSTree::print\_middle\_order()**

作 用：中序遍历树 按字母顺序输出

参 数：const Tnode\* root

返回值：无

计算过程：

1. 若树为空，则终止函数；
2. 递归调用，打印左子树；
3. 输出根结点；
4. 递归调用，打印右子树.

**算法2.7 bool BSTree::deleteBST**

作 用：销毁树

参 数：Tnode \*root

返回值：true false

计算过程：

1. 若树为空，返回false
2. 删除左子树，递归调用函数自身
3. 删除右子树，递归调用函数自身
4. 返回true

## 定义算术表达式和类型

**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，用于对类进行测试。

模块2： Expression.hpp，Expression.cpp：

定义并实现表达式类，表达式可以是数字或者操作符，表达式按照语法树进行存储；定义并实现了语法分析以及遍历输出的操作。

模块3： Exception.hpp：定义并实现了异常类

模块4： tool.hpp,tool.cpp,定义并实现了一个工具类，用来对用户的输入等进行处理

**关键数据结构设计：**

语法树：

Operator=“+”||“-”||“\*”||“/”

Expression=<Number>|| “Operator”

如下图为1+2和1\*（2+3）的语法树

+

1

2

\*

1

+

2

3

对结果的求解采用中序遍历，依次将左子树计算的值与根节点的操作数和右子树的值进行相应的操作。

**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

返回值：总是返回0。

计算过程：

1. 获取用户输入的字符串
2. 调用Expression类的成员方法GetExp(),对字符串进行处理，处理后判断异常
3. 分别用三种顺序遍历语法树，输出对应式子。

**算法1.2 Expression::Expression(double aNumber)**

作 用：Expression类的一个构造方法，给成员变量中的Number赋值

参 数：double aNumber

返回值：一个Expression类的对象

计算过程：

给成员变量IsNumber赋值为true，表明这是一个数字表达式，给Number赋值aNumber，其余初始化为0.

**算法1.3 Expression::Expression(char aOperator , Expression\* aLeft , Expression\* aRight)**作 用：Expression类的一个构造方法，给成员变量中的Number赋值

参 数：char aOperator , Expression\* aLeft , Expression\* aRight

返回值：一个Expression类对象

计算过程：

给成员变量IsNumber赋值为false，表明这是一个操作符表达式，给Number赋值0，Operator为aOperator，左右子树初始化为aLeft，aRight。

**算法1.4 Expression::~Expression()**

作 用：Expression类的析构函数

参 数：无

返回值：无

计算过程：

判断左右子树是否为空，分别删除

**算法1.5**  **Expression\* Expression::GetNumber(char\*& Stream)**

作 用：获得输入字符传中的数字，

参 数：char\*& Stream

返回值：一个Expression对象的指针。

计算过程：逐个扫描输入的字符，一直到不是数字，将获得的数字字符转化为对应的十进制数字，生成新的对象返回。如果在该出现数字（操作符后面）的位置没有出现，抛出异常。

**算法1.6**  **Expression\* Expression::GetTerm(char\*& Stream)**

作 用：判断表达式是否括弧表达式

参 数：char\*& Stream

返回值：返回Expression类对象的指针

计算过程：

1. 尝试返回数字表达式
2. 如果检测到了左括弧，对后面的字符串进行处理，碰到右括弧，停止，若一直没有碰到，抛出异常
3. 在没有做括弧的情况下，先碰到右括弧，抛出异常

**算法1.7 Expression\* Expression::GetFactor(char\*& Stream)**

作 用：判断字符串是否是一个因子

参 数：char\*& Stream

返回值：返回Expression类对象的指针

计算过程：

（1）获得括弧表达式的结果

（2）判断是否碰到乘号或除号，有的话，就转入下一层的判断，如果发生异常，先删除当前结果，并转发异常

**算法1.8 Expression\* Expression::GetExp(char\*& Stream)**

作 用：判断是否求和作差

参 数：char\*& Stream

返回值：返回Expression类对象的指针

计算过程：

1. 获得乘除表达式的结果
2. 进入循坏,判断是否碰到加减号，进行相关运算或者抛出异常

**算法1.9 double Expression::calculate()**

作 用：计算表达式的值

参 数：无

返回值：双精度浮点型

计算过程：

1. 如果是数字，直接返回
2. 如果是操作符表达式，判断是哪一个，递归用左子树的值和右子树的值用这个操作符进行计算

**算法1.10 void Expression::PrintDLR()**

作 用：先序遍历输出

参 数：无

返回值：无

计算过程：

（1）如果是数字，直接输出；如果是操作符表达式，输出后递归输出左右子树

**算法1.11 void Expression::PrintLDR()**

作 用：中序遍历输出

参 数：无

返回值：无

计算过程：

如果是数字，直接输出；如果是操作符表达式，递归输出左子树，输出操作符，在递归输出右子树

**算法1.12 void Expression::PrintDLR()**

作 用：后序遍历输出

参 数：无

返回值：无

计算过程：

如果是数字，直接输出；如果是操作符表达式，递归输出左右子树 ，再输出操作符

**算法1.12 bool Is(char\*& Stream , char\* Text);**

作 用：判断读入的是否为对应的字符串

参 数：无

返回值：无

计算过程：

（1）保存输入的字符串的参数

（2）过滤空格

（3）再和期望的值比较

# 测试用例与测试结果

## 5.1定义并实现日期类型

|  |  |  |
| --- | --- | --- |
| 序号 | 初始化 | 输出 |
| 1 | 1970.1.1  5.25  3  2.19.5.31 | the date is :1970.1.1  add: 31 day  the date is :1970.2.1  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  the date is :2019.5.25  add: 4 month  the date is :2020.9.25  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  the date is :2019.5.3  add: 3 year  the date is :2022.5.3  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  the date is :2019.5.31 |
| 2 | 2019.13.6 | 输入错误！ |
| 3. | 2007.2.28 | 2008.2.29 |

## 5.3定义表示二叉树及其结点的类型

|  |  |  |
| --- | --- | --- |
| 序号 | 测试数据 | 输出 |
| 1 | b  a  g  f  e  e  d  c | \*\*\* 本程序按【二叉排序树】存储单词 \*\*\*  请输入若干单词，用空格/换行/制表符分隔.  \*\*\* 输入 ^z 结束输入 \*\*\*  b: 1  a: 1  g: 1  f: 1  e: 2  d: 1  c: 1  <NULL>  <NULL>  <NULL>  <NULL>  a: 1  b: 1  c: 1  d: 1  e: 2  f: 1 |
| 2 | k  z  a  g  f  a | \*\*\* 本程序按【二叉排序树】存储单词 \*\*\*  请输入若干单词，用空格/换行/制表符分隔.  \*\*\* 输入 ^z 结束输入 \*\*\*  k :1  a:2  g:1  f:1  <NULL>  z:1 |

## 5.3定义算术表达式类型

|  |  |  |
| --- | --- | --- |
| 序号 | 测试数据 | 测试结果 |
| 1 | 3+5 | 先序遍历：+ 3 5  中序遍历：（3+5）  后序遍历：3 5 +  8 |
| 2 | 2\*4 | 先序遍历：\* 2 4  中序遍历：（2\*4）  后序遍历：2 4 \*  8 |
| 3 | 3/0 | 先序遍历：/ 3 0  中序遍历：（3/0）  后序遍历：3 0 \*  错误：除数为零 nan |
| 4 | 1@@3 | 发生错误  信息：发现多余的字符 |
| 5 | +3 | 发生错误  信息：此处需要表达式 |

# 实验总结

通过这次实验，我了解了一些C++的基本概念和机制，掌握了数据抽象的基本手段，用类型的操作来封装数据结构。在实验过程中，采用多个源代码文件，我们要先定接口再考虑实现。同时缺省处理和边界处理在编程过程中也十分重要。我们也要在以后的编程中使用库函数，减少编程，提高效率。在实验过程中，我并不太会使用“封装”这一概念，在第二题中一直编程错误。重新学习相关理论和模仿老师的代码，解决了相关问题。由于时间仓促，第一题中对日期的相关操作比较少，在以后可以慢慢补充。我还需要加深对接口和实现的理解，通过实际操作提高自己的编程水平。