可编程计算器 实验报告

(使用LaTeX编辑)

题目: 按照实验要求, 设计一个多功能可编程计算器.

姓名: **崔冠宇**　　学号: **2018202147**　　完成日期: **2019.11.22**

1 需求分析

1. 设计任务: 用顺序表、链表、栈和数组等数据结构, 设计一个多功能计算器, 要求至少具有向量计算功能(含加减法, 夹角余弦值), 多项式计算功能 (含加减法、乘法, 分别使用顺序表和链表实现), 表达式定义解析执行功能, 以及选做矩阵运算等功能.

2. 输入输出形式: 以交互式界面实现, 用户按程序所给提示输入命令, 程序执行相应命令后将结果输出在控制台中.

3. 达到的功能: 基本实现“设计任务”部分的要求.

4. 测试数据: 已经形成单独的测试报告, 详见测试报告.

2 概要设计

这里只给出最主要的抽象数据类型定义, 其他小的数据结构(如多项式节点, 函数等)详见代码.

1. 顺序表抽象数据类型定义:

   ADT SqList<T>{

   数据对象: $D = \{a_i | a_i \in TSet, i = 1, 2, \ldots, n, n \in \mathbb{N}_+\}$

数据关系: $R = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D, i = 2, 3, \ldots, n\}$

基本操作:

(a) SqList()

初始条件: 无.

操作结果: 构造函数, 构造一个顺序表.

(b) ~SqList()

初始条件: 顺序表已经存在.

操作结果: 析构函数, 销毁一个顺序表.

(c) clear()

初始条件: 顺序表已经存在.

操作结果: 将顺序表清空.

(d) isEmpty()

初始条件: 顺序表已经存在.

操作结果: 若顺序表为空表, 返回true, 否则返回false.

(e) length()

初始条件: 顺序表已经存在.

操作结果: 返回顺序表中元素个数.

(f) getElem(i, &e)

初始条件: 顺序表已经存在, $1 \leq i \leq$ length().

操作结果: 用e返回第i个元素的值.

(g) putElem(i, e)

初始条件: 顺序表已经存在, $1 \leq i \leq$ length().

操作结果: 将e替换第i个元素.

(h) locateElem(e, compare())

初始条件: 顺序表已经存在.

操作结果: 返回第一个与e满足compare的元素的位序. 若不存在, 返回0.

(i) priorElem(cur_e, &pre_e)

初始条件: 顺序表已经存在.

操作结果: 若cur_e是数据元素, 且不是第一个, 则用pre_e返回前驱, 否则没有定义.

(j) nextElem(cur_e, &next_e)

初始条件: 顺序表已经存在.

操作结果: 若cur_e是数据元素, 且不是最后一个, 则用next_e返回后继, 否则没有定义.

(k) insertElem(i, e)

初始条件: 顺序表已经存在, $1 \leq i \leq$ length()+1.

操作结果: 在第i个位置之前插入新的数据元素e, L的长度加1.

(l) deleteElem(i, &e)

初始条件: 顺序表已经存在且非空, $1 \leq i \leq$ length().

操作结果: 删除第i个位置的元素, 并用e返回, L的长度减1.

(m) traverse(visit())

初始条件: 顺序表已经存在.

操作结果: 依次对顺序表的每个元素调用visit(), 一旦visit()失败, 则操作失败.

(n) print()

初始条件: 顺序表已经存在.

操作结果: 打印顺序表.

}**ADT** SqList<T>

2. 链表抽象数据类型定义:

   ADT LinkedList<T>{

   数据对象: $D = \{a_i | a_i \in TSet, i = 1, 2, \ldots, n, n \in \mathbb{N}_+\}$

   数据关系: $R = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D, i = 2, 3, \ldots, n\}$

   基本操作: (和顺序表基本一致, 不再详细写出)

   }**ADT** LinkedList<T>

3. 栈抽象数据类型定义:

   ADT Stack<T>{

   数据对象: $D = \{a_i | a_i \in TSet, i = 1, 2, \ldots, n, n \in \mathbb{N}_+\}$

   数据关系: $R = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D, i = 2, 3, \ldots, n\}$, 约定$a_n$端为栈顶, a_1端为栈底.

   基本操作:

   (a) Stack()

   初始条件: 无.

   操作结果: 构造函数, 构造一个栈.

   (b) ∼Stack()

   初始条件: 栈已经存在.

   操作结果: 析构函数, 销毁一个栈.

   (c) isEmpty()

   初始条件: 栈已经存在.

   操作结果: 若为空栈, 返回true, 否则返回false.

   (d) top(&e)

初始条件: 栈已经存在, 且不空.

操作结果: 把栈顶元素复制到e.

(e) push(e)

初始条件: 栈已经存在.

操作结果: 把e压到栈顶.

(f) pop()

初始条件: 栈已经存在, 且不空.

操作结果: 弹出栈顶元素.

}**ADT** Stack<T>

4. 矩阵抽象数据类型定义:

ADT Matrix{

数据对象: $D = \{a_{i,j} | i = 1, 2, \ldots, m; j = 1, 2, \ldots, n; a_{i,j} \in \mathbb{R}, m$和$n$分别称为矩阵的行数和列数$\}$

数据关系:

$R = \{Row, Col\}$

$Row = \{< a_{i,j}, a_{i,j+1} > | 1 \leq i \leq m, 1 \leq j \leq n-1\}$

$Col = \{< a_{i,j}, a_{i+1,j} > | 1 \leq i \leq m-1, 1 \leq j \leq n\}$

基本操作:

(a) Matrix(m, n)

初始条件: 无.

操作结果: 构造函数, 构造一个$m \times n$的零矩阵.

(b) ∼Matrix()

初始条件: 矩阵已经存在.

操作结果: 析构函数, 销毁一个矩阵.

(c) readData()

初始条件: 矩阵已经存在.

操作结果: 读取输入的数据更改矩阵内容.

(d) operator+(&m)

初始条件: 矩阵与矩阵m行数列数对应相等.

操作结果: 返回二者之和.

(e) operator-(&m)

初始条件: 矩阵与矩阵m行数列数对应相等.

操作结果: 返回二者之差.

(f) operator*(&m)

初始条件: 矩阵的列数与矩阵m行数相等.

操作结果: 返回二者之积.

(g) operator=(&m)

初始条件: 矩阵与矩阵m行数列数对应相等.

操作结果: 用矩阵m赋值给本矩阵.

(h) transpose()

初始条件: 矩阵已经存在.

操作结果: 返回矩阵的转置.

(i) trace()

初始条件: 矩阵行数列数相等.

操作结果: 返回矩阵的迹.

(j) rank()

初始条件: 矩阵已经存在.

操作结果: 返回矩阵的秩.

(k) det()

初始条件: 矩阵行数列数相等.

操作结果: 返回矩阵的行列式.

(l) inverse()

初始条件: 矩阵行数列数相等, 且可逆.

操作结果: 返回矩阵的逆矩阵.

(m) QRDecomp(&Q, &R)

初始条件: 矩阵已经存在.

操作结果: 将矩阵QR分解的结果分别放到Q矩阵和R矩阵.

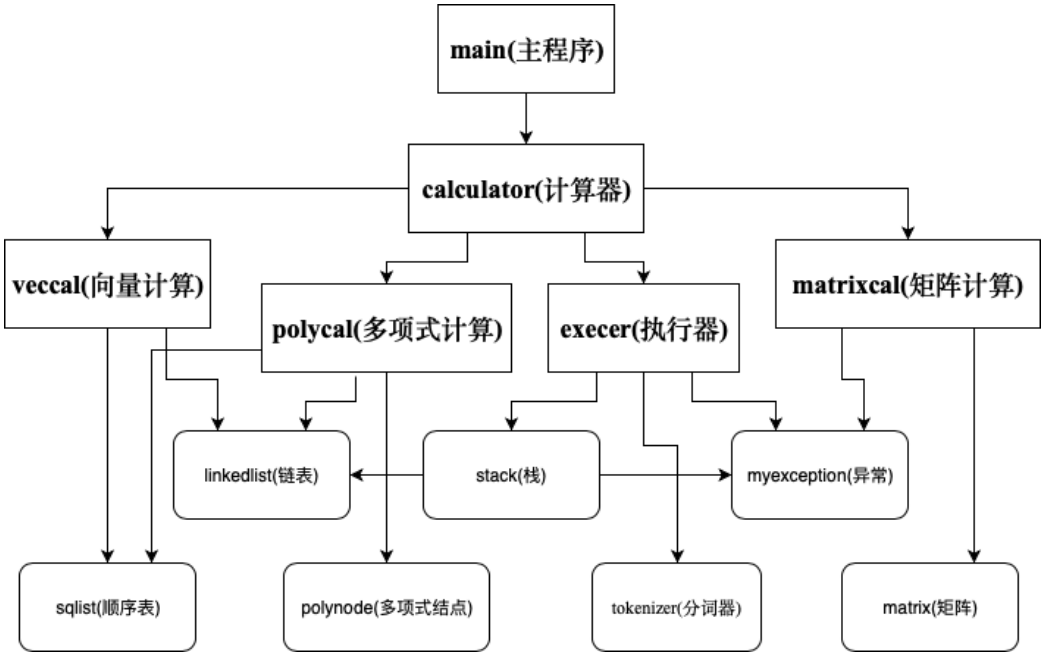
(n) eigen_QR(iterTimes)

初始条件: 矩阵行数列数相等.

操作结果: 使用QR算法迭代iterTimes次, 输出矩阵的特征值与特征向量.

}**ADT** Matrix

5. 模块设计: 本程序共由6个主要模块组成, 它们分别是: 主程序、计算器模块、向量计算模块、多项式计算模块、表达式执行模块、矩阵计算模块. 模块之间的调用关系如下图:



3 详细设计

这一部分由于篇幅所限, 详细内容请参看代码, 下面仅列举一些个人认为比较有亮点的设计.

1. 在之前的实验中, 助教学长提到了“正则表达式”这一技术, 说它能够简化代码, 比较方便. 我之前听说过regex, 但一直没有机会了解, 于是我就趁这次实验, 稍微学习了一下. 所以这次的分词模块中, 在函数表达式处理的时候, 简单运用了一下.

2. 除了正则表达式, 我还了解了一下lambda表达式, 并尝试使用.

3. 另外值得一提的一点是, 之前一直看到C++有异常处理的功能(try-catch语句块), 稍做了解后, 我尝试在这次的实验中使用了一下, 权当为之后工作等方面积累经验.

4. 最后一个亮点, 就是(扩展要求的)矩阵运算部分. 这一部分中, 我不仅写了矩阵加减乘这三种基本运算, 还加入了行列式、秩、逆矩阵等功能. 除此之外, 我还

参考了一些有关数值分析与数值计算方面的文献[1], 尝试着实现了矩阵QR分解与特征值特征向量计算两个功能.

4 调试分析

1. 调试与问题解决: 偶尔会出现“手误”的情况, 或是逻辑不严密, 导致程序未能按预期运行. 这时, 我主要有两种解决方案: 一是利用lldb的单步执行以及变量查看, 观察代码执行情况; 二是增加输出语句, 观察在哪里出错.

2. 复杂度分析: 关于几类数据结构及相关算法的复杂度分析, 书上已经明确了, 在这里我只简单分析一下矩阵几个算法的复杂度.

   (方便起见, 均设矩阵是$n \times n$的.)

   - 迹: 只需要对主对角线上的元素进行累加, 故时间复杂度为$O(n)$.
   - 加、减、数乘、转置: 显然它们是双循环算法, 时间复杂度为$O(n^2)$.
   - 乘法: 结果矩阵(n^2个元素)的每一个元素的确定, 都要做n次乘法和n次加法, 时间复杂度为$O(n^3)$ [2].
   - 秩、行列式、逆矩阵: 使用初等变换, 根据《高等代数》教材所写, 时间复杂度为$O(n^3)$.
   - QR分解: 首先最外层循环n次, 在每个循环中, 有获取列向量($O(n)$), 向量内积($O(n)$), 向量模长($O(n)$), 矩阵加法($O(n^2)$)和矩阵乘法($O(n^3)$). 可见循环内最高复杂度$O(n^3)$, 故整个算法的时间复杂度为$O(n^4)$.
   - 特征值与特征向量: 最外层循环iterTimes次, 在每个循环中, 最高复杂度为QR分解的$O(n^4)$, 故整个算法的时间复杂度为$O(n^4 \times$ iterTimes)(输出部分的复杂度小, 被忽略).

[1] Peter J. Olver, University of Minnesota. Orthogonal Bases and the QR Algorithm.
[2] 另有文献指出, Strassen算法(1969)为$O(n^{2.81})$, 目前最优算法(2014)为$O(n^{2.373})$左右.

3. 一些感想: 通过这一系列实验, 我感到了软件设计中“模块化”的重要性. 一开始, 我的程序只有一个计算器大类和数据结构几个类, 随着代码量不断增加, 函数的定位、修改愈发困难起来, 最终我通过把计算器模块相比之下更精细的分为几个更小的模块, 解决了这些问题.

5 用户手册

这一部分根据实验要求, 已经拆分成独立的用户手册, 请参阅“用户手册.pdf”.

6 测试结果

这一部分根据实验要求, 已经拆分成独立的测试报告, 请参阅“测试报告.pdf”.

7 附录

源程序文件名清单(按字典序)如下:

1. calculator.h //计算器总模块
2. execer.h //表达式执行器模块
3. linkedlist.h //链表数据结构
4. main.cpp //主程序
5. matrix.h //矩阵数据结构
6. matrixcal.h //矩阵运算器模块
7. myexception.h //自定义异常类

8. polycal.h　　//多项式运算模块

9. polynode.h　　//多项式节点类

10. sqlist.h　　//顺序表数据结构

11. stack.h　　//栈数据结构

12. tokenizer.h　　//分词器分模块

13. unary_function.h　　//一元函数类

14. veccal.h　　//向量运算模块

总代码量超过3500行.