程序设计语言认知实验报告

——谭超 1120161874

一、 实验目的和内容

了解程序设计语言的发展历史，了解不同程序设计的各自特点；感受编译执行和解释执行两种不同的执行方式，初步体验语言对编译器设计的影响，为后续编译程序的设计和开发奠定良好的基础。

二、 实验的具体过程和步骤

运行环境：

4 核 CPU 主频 2.30GHz 内存 8G

共同的实验步骤：

我新建了三个 txt 文件。其中 vector.txt 用来存储向量，第一行是两个整数，分别是 1 和 vectorColumn(向量的长度)，为了格式相同，我将向量视为一个 1*vectorColumn 的矩阵，第二行则是通过随机数产生的 vectorColumn 个 0 到 100 的整数。matrix.txt 用来存储矩阵，第一行是 matrixRom(矩阵的行数)和 matrixColumn(矩阵的列数)，后面的 matrixRom 行每一行有 matrixColumn 个通过随机数产生的 0 到 100 的整数。最后是 result.txt,该文件用来存储运算结果，存储格式与 vector.txt 相同。

C++：

一共分为三个函数：inputMatrix()用来从文件中读入矩阵和向量并以二维数组的方式存放，vectorMultiplyMatrix()实现向量与矩阵的相乘，并将结果以二维数组的方式返回给主函数，main()则调用 inputMatrix()和 vectorMultiplyMatrix()以及打印结果、将结果写入 result.txt 文件和程序运行计时。

JAVA：

一共定义了两个类，一个主类和一个 Matrix 类，Matrix 内部的属性有行数 row、列数 column、以及用列表存储的矩阵 matrix， Matrix 类内部实现的方法有从文件读取矩阵的函数 inputMatrixFromFile()，将矩阵写入文件的函数 outputMatrixToFile() 以及实现矩阵相乘的函数 matrixMultiply()。首先创建两个 Matrix 对象 vector 和 matrix，然后分别从 vector.txt 和 matrix.txt 读入向量与矩阵，通过调用 vector 对象的 matrixMultiply()函数实现向量与矩阵相乘的结果即新的 Matrix 对象 result,通过调用 result 对象的 outputMatrixToFile()将结果写入到文件中。在程序开始和结束的时候分别调用 currentTimeMillis()获取当前时间，结束后通过二者的差获得程序运行时间。

Python：

我定义了一个 Matrix 类，Matrix 类实现的功能与上述 JAVA 的 Matrix 类部份相同，之后的运行过程也和上述 JAVA 的运行过程相同。运行结束后通过调用 time 模块的 process_time()函数得到程序运行的时间。

Haskell：

Haskell 实现的向量乘以矩阵也是从分别文件中 vector.txt 和 matrix.txt 读入向量与矩阵，然后利用定义的三个函数来实现向量矩阵的运算，第一个函数取出矩阵的第一列，第二个函数去除矩阵的除了第一列的后面所有列，最后一个函数便是计算向量与矩阵相乘的函数，主要的思想是递归，每次取出剩余矩阵的第一行与向量相乘并求和，然后将结果加入到向量与剩余矩阵列相乘的列表中去，直至最后一列。计算时间我使用了 getCPUTime 这个函数，分别在计算开始和结束的时候调用此函数，最后相减得到计算时间。

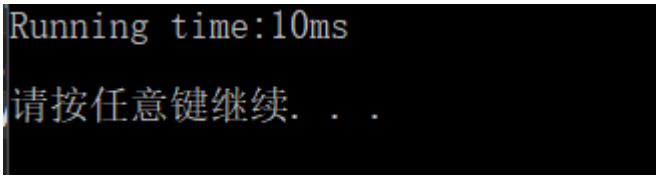
注意：本实验中的程序计时记录的是进行矩阵运算的时间，对文件的读写操作没有计入时间。所以实际运行的时间要比记录的时间长。

三、 运行效果截图

C++：

运算结果由于太大无法截图，请打开/C++/bin/result.txt 查看。

第一次运行时间结果截图

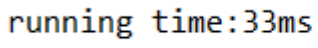


运行时间

编号	运行时间（ms）
1	10
2	9
3	8
4	8
5	8
6	8
7	8
8	8
9	9
10	8
平均时间	8.4

JAVA：

第一次运行时间截图



部分运行结果截图（由于结果输出到 result.txt 是乱码，而且我始终没找出 bug，故而截了部分结果的图）

```
result row:1  result column:1000
2330316
2469708
2400241
2393810
2375213
2425206
2462960
2367764
2356506
2391086
2403190
2378957
2409560
```

运行时间

编号	运行时间（ms）
1	33
2	31
3	27
4	29
5	28
6	36
7	27
8	28
9	27
10	28
平均时间	28.9

Python：

运算结果由于太大无法截图，请打开/Python/src/result.txt 查看。

第一次运行时间结果截图

```
In [4]: runfile('D:/SourceCode/PythonSo
running time: 0.28470396995544434 s
```

运行时间

编号	运行时间（ms）
1	284.70396
2	279.78125
3	280.9375
4	282.109375
5	283.3125
6	284.46875
7	285.640625
8	286.75

9	287.984375
10	289.15625
平均时间	284.86526

Haskell：
第一次运行时间以及结果

```
result[999]: 2445428
Running time(size: 1000): 6734.375 ms
```

运行时间

编号	运行时间（ms）
1	6734.375
2	6718.75
3	6640.625
4	6859.375
5	6671.875
6	6781.25
7	6734.375
8	6812.5
9	6718.75
10	6703.125
平均时间	6737.5

四、 语言易用性和程序规模对比分析

语言使用的难易程度：
由于我们最开始学的就是面向过程编程的 C，然后才学了面向对象编程，而函数式编程对于我来说较为陌生，所以个人使用起来的难易程度为 C/C++ < Python < Java < Hakell，但就数学运算的角度来讲，Haskell 无疑是非常方便的，Python 其次。
各种语言计算向量与矩阵相乘的代码行数

语言	行数
C++	4
JAVA	4
Python	4
Haskell	4

C++、Java、Python 实现向量与矩阵的相乘的思路都几乎一样，都是通过两个循环完成，所以代码规模也相差不大。而 Hakell 虽然思路不同，通过递归调用的函数也十分简单，故而代码规模也比较小。

五、 程序运行性能对比分析

从实验结果来看，C/C++无疑是最优的，Java 其次，Python 耗时更长，Haskell 耗时最长。C/C++是编译型语言，执行速度较快，而 Java 和 Python 是解释型语言，先编译生成字节码文件，再从字节码文件中读一行解释一

行， 故而执行速度较慢。

六、 实验心得体会

不同的语言有不同的思考模式，这便造成了不同的语言擅长于处理不同的问题，在本次实验中，由于不了解函数式编程，导致我很难理解 Haskell，即使只是写一个小程序也较为困难。另外，编译执行和解释执行的速度真的差异较大。