编译原理与设计

实验报告

实验名称： Lab 1：语言认知实验

姓名/学号： 李昊阳/1120203053

一、 实验目的和内容

实验目的：了解程序设计语言的发展历史，了解不同程序设计语言的各自特点；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。

实验内容：给定一个特定的功能，分别使用 C/C++、Java、Python、Haskell 和一种汇编语言实现该功能，对采用这几种语言实现的编程效率，程序的规模，程序的运行效率进行对比分析。汇编语言可以是 X86、MIPS、ARM 或者 RISC-V 等。例如分别使用上述几种语言实现一个简单的矩阵乘法程序，输入两个矩阵，输出一个矩阵，并分析相应的执行效果。

二、 实验环境

设备：RedmiBook 14 锐龙版

操作系统：Windows 10 Pro，64-bit（Build 19043.2006）10.0.19043

IDE：C++：Microsoft Visual Studio Community 2019

Java：IntelliJ IDEA 2022.2.2（Community Edition）

Python：PyCharm 2022.2.3（Community Edition）

Haskell 编译器：The Glasgow Haskell Compiler（GHC）9.4.2

CPU 核数：4　CPU 主频：2.10 GHz　RAM：8.00 GB（5.93 GB 可用）

L1\L2\L3 Cache 大小：384KB\2.0MB\4.0MB

三、 实现的具体过程和步骤

参阅实验指导文件后，综合考虑，选择了题目（1）：使用上述各种语言分别实现矩阵相乘。下面分别为使用 C++、Java、Python、Haskell、汇编实现的具体过程，顺序即实验时

编写的先后顺序。

C++：

C++作为本人计算机学习中最先掌握也是最熟练的编程语言，考虑使用 C++来实现第一个矩阵相乘算法，其他语言的实现可以参考 C++实现的思路，加快编写速度。

实验要求进行程序运行性能对比，对每个程序运行 5~10 次取其平均值，因此源数据/样本的选择也应具有多样性与随机性，现对实验重复步骤定义如下：样本通过随机程序生成 $n = 2\text{^}k\ (k = 1,2,\ldots,10)$ 的一系列随机矩阵，每个程序对样本重复运行 3 次，求取不同 n 时每个程序的平均运行时间与每个程序的总体平均运行时间。

如此重复实验可以横向比较不同语言下、不同规模的矩阵乘法的运算速度，包括算术运算、文件读写的效率等等。

首先编写随机矩阵生成算法，文件命名为 lab1_random.cpp，规范矩阵格式：第一行为矩阵大小 q，表面接下来 2q+1 行为两个 q*q 的随机矩阵，以空行隔开，每个矩阵元素以两个制表符“\t\t”隔开。最终生成样本 testdata.txt，生成算法此处不再赘述。

然后编写矩阵相乘算法，命名为 lab1_C++.cpp。在算法中，使用文件流 ifstream 读入 testdata.txt，然后按行循环读取文件，将每行内容保存在 string 中，直至串为空。若 string 为数字 q，则说明接下来 2q+1 行为两个矩阵，建立左右矩阵 left/rightMatrix 以及 resultVector（计算乘法时使用），后按行循环 q 次读取文件，对于每行字符串，使用正则表达式“\\s+”进行拆分，依次存储进对应矩阵元素中，略过空行后再重复一次。读取完毕后进行矩阵乘法计算，采用交换次序的 ikj 循环，可以极大地减少指针跳跃次数（即 cache 不命中次数），每一行（i）计算完毕后，使用 string 将该行计算结果整合起来，录入文件流 ofstream 中。该矩阵计算完毕后，释放矩阵资源，将文件流写入 result_C++.txt 中。最终关闭所有文件流。

此外，在程序开始时与每个矩阵计算开始时均开启计时，并在程序结束前与矩阵计算结束后截止，结果也将计入到 result_C++.txt 中，以此来统计总的与不同规模矩阵相乘的运行时长。

Java：

Java 语言本身就与 C++语言具有相似性，同为面向对象的命令式语言，因此 Java 算法实现的具体过程与步骤与 C++极为类似，可直接参考 lab1_C++.cpp 进行编写，命名为

lab1_Java.java，具体实现步骤如上。

文件的读写同样采取流的方式，使用了 BufferedReader 与 BufferedWriter 类，读取 testdata.txt，计算结果缓存在输出文件流中，后写入到 result_Java.txt 中。有所不同的是，在构造输出字符串时，使用了 StringBuilder 类，加快字符串的连接速度。同时，Java 的 String 类自带 split 方法，可将字符串拆分为一系列子串，保存在 String[]中。

需注意的时，Java 自带有垃圾回收机制，因此新分配的对象在超出作用范围后会被自动回收，不需要再释放矩阵空间。而 C++则需要自己维护堆，每次将分配给矩阵的空间手动释放。

Python：

Python 与 C 具有很大不同，C 属编译型语言，Python 属解释型语言。且 Python 在语法上更为简洁，变量不需要事先声明类型，具有更多可用的库。因此在编写 Python 的矩阵相乘算法时引用了 numpy 库：NumPy(Numerical Python)是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

lab1_Python.py 的算法思路大致相同：使用 open 方法打开读写文件，在 while true 循环中按行读取 testdata.txt 文件，若为空则退出循环，若为数字，则使用 np.zeros 建立两个 q*q 的空矩阵（效率高于 list 建立的二维数组），读取 2q+1 行内容，将 split 后的结果存入左右矩阵中。有两种方法可以计算矩阵乘法，一种是同样使用 ikj 的三层循环计算，另一种则直接调用 np.dot 方法，返回相乘后的矩阵。计算完毕后，将结果矩阵写入到 result_Python.txt 中。同时也计算运行时间。

Haskell：

Haskell 是我从未了解也未曾接触过的高级编程语言，因此要使用 Haskell 语言编写矩阵乘法算法，对我来说具有相当大的困难。如此难的原因，是因为 Haskell 和 C 语言有着非常大的区别。Haskell 是纯函数式编程语言，而 C 属于过程式编程语言。两者不具有互通性。且函数式语言不存在变量，不存在循环，只能通过递归实现。因此要实现矩阵的录入与相乘需要对 C++的循环体进行高度的抽象。

考虑便捷性与实用性，未下载 Haskell 的“全家桶套餐”。在下载简单的 Haekell 编译器 GHC 后，配置环境变量，使用记事本与命令行，即可进行简单代码的编写、编译与运行。

先整体读入文件 testdata02.txt，按行分为[String]，再根据空行分为两个[[Int]]

矩阵。对于函数式语言 Haskell 而言，需将循环转化为递归形式。进行矩阵乘法计算时，重载运算(*.)，先将右边的矩阵转置，进行嵌套的列表解析，类似于双层循环，返回结果矩阵[[Int]]，最后写入 result_Haskell01.txt，并输出运行用时。

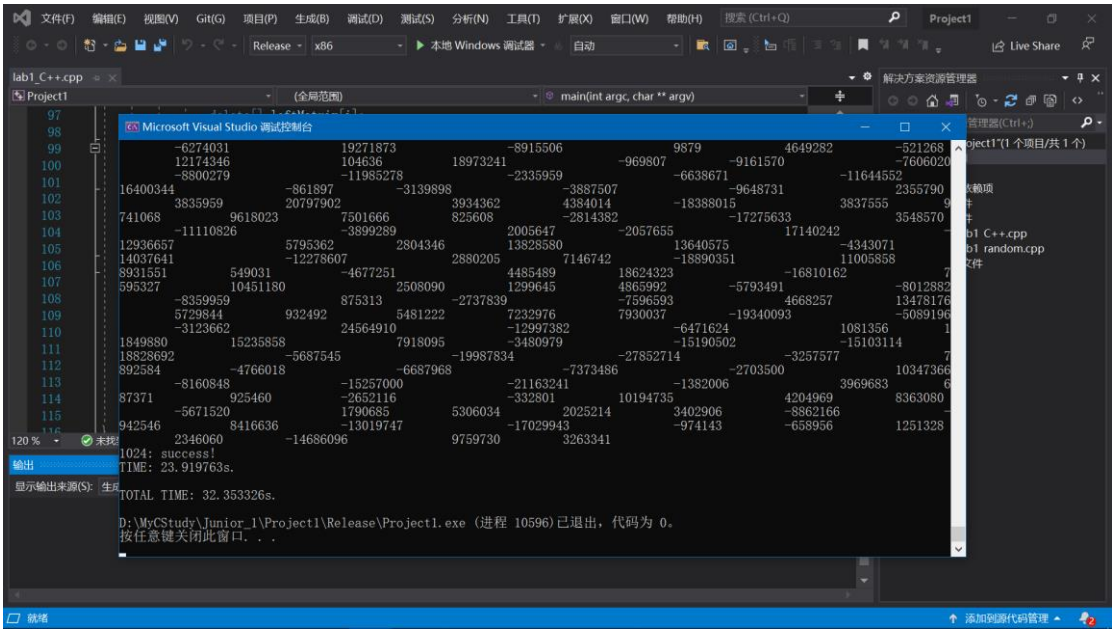
需注意的是，Haskell 是一门惰性语言，即表达式是在其值需要使用时才被求值。因此使用 getCPUTime 计时时，其范围内的核心代码需得到执行，才能进行准确计时。

汇编：

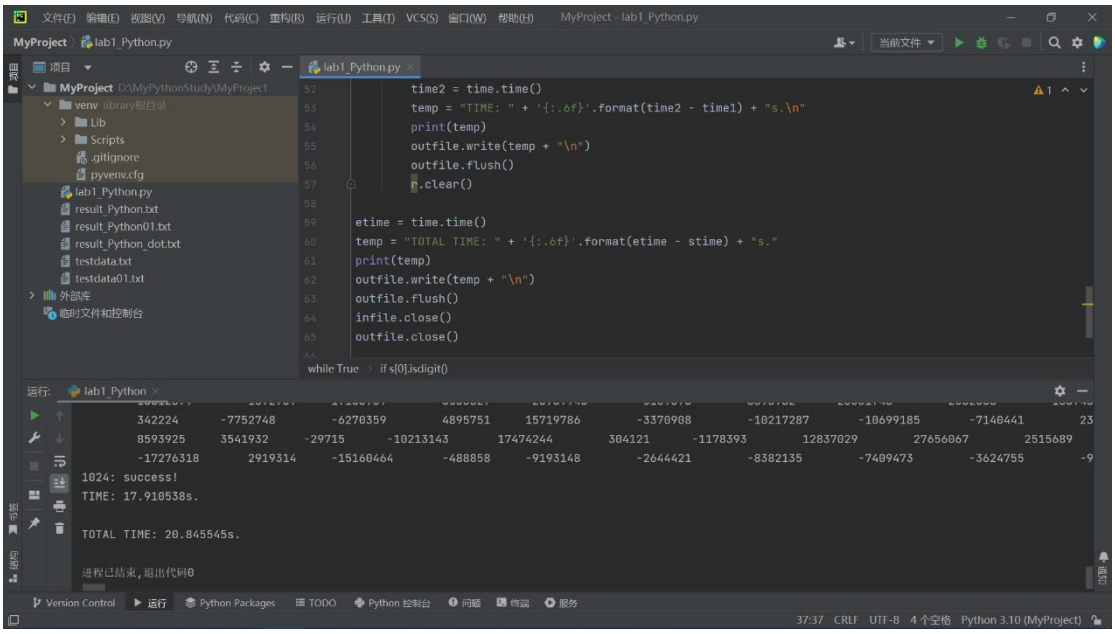
待写。

四、 运行效果截图

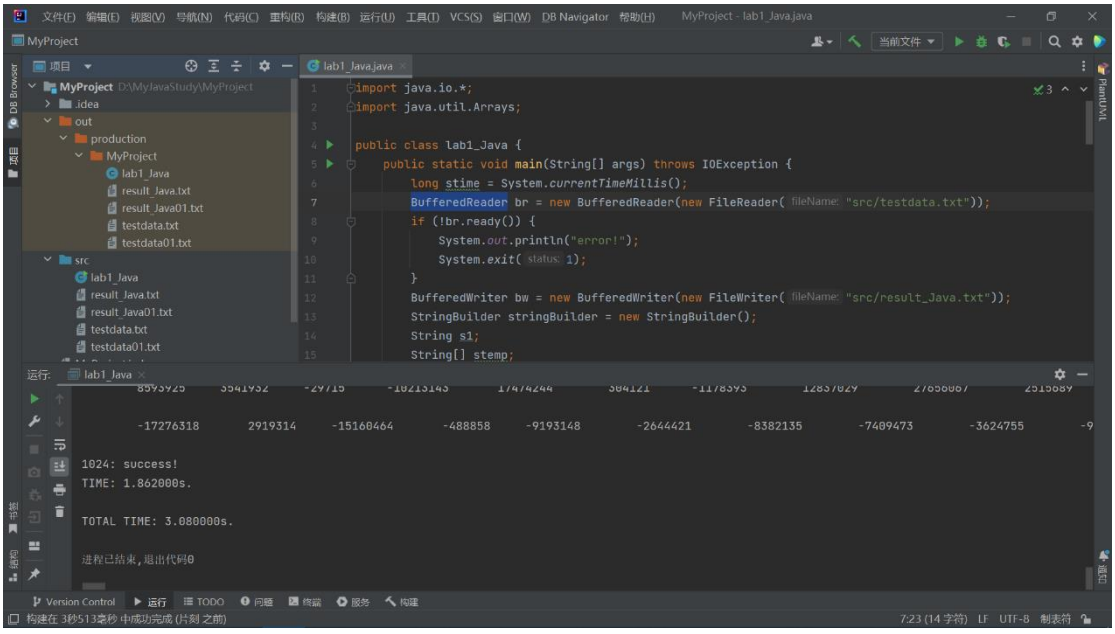
C++：



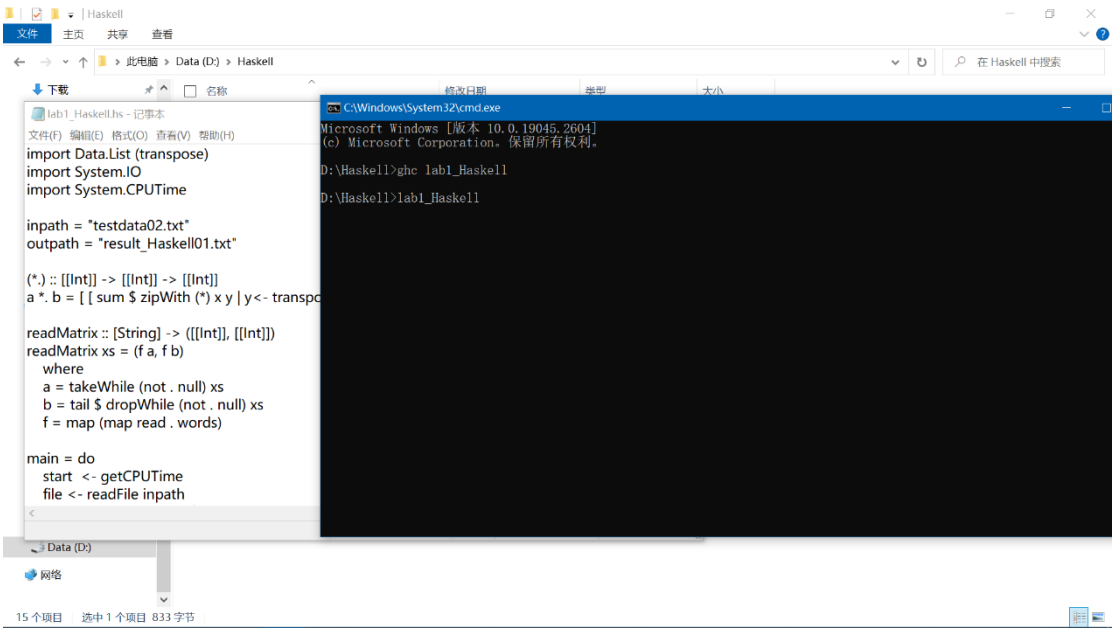
Java:


MyProject - lab1_Python.py
lab1_Python.py
time2 = time.time()
temp = "TIME: " + '{:.6f}'.format(time2 - time1) + "s.\n"
print(temp)
outfile.write(temp + "\n")
outfile.flush()
n.clear()
etime = time.time()
temp = "TOTAL TIME: " + '{:.6f}'.format(etime - stime) + "s."
print(temp)
outfile.write(temp + "\n")
outfile.flush()
infile.close()
outfile.close()
1024: success!
TIME: 17.910538s.
TOTAL TIME: 20.845545s.
进程已结束,退出代码0

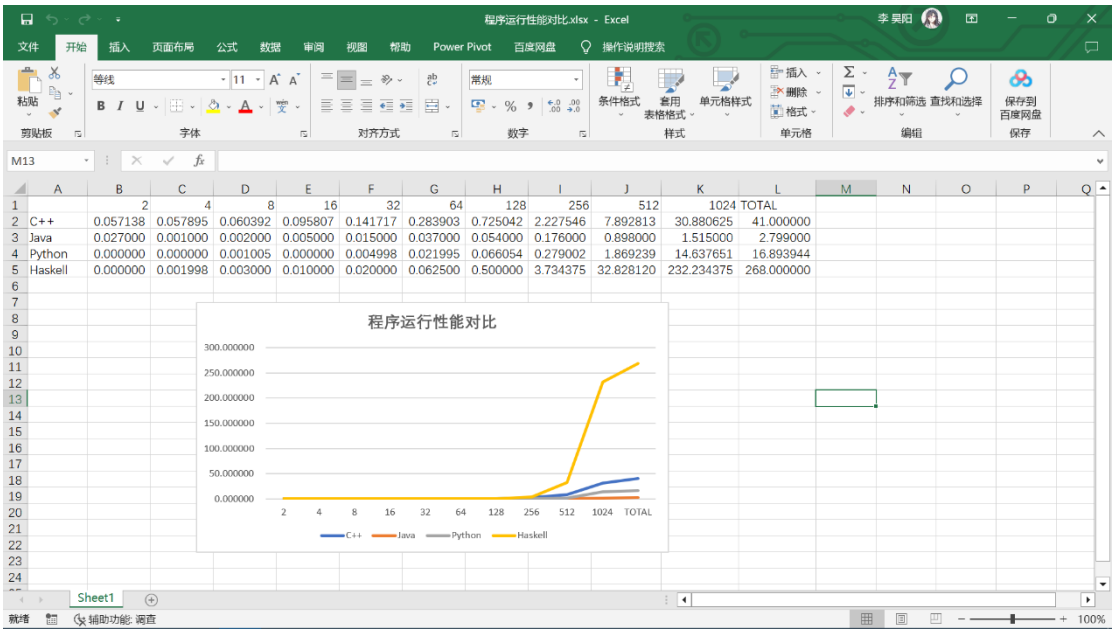
Python:


MyProject - lab1_Java.java
lab1_Java.java
import java.io.*;
import java.util.Arrays;
public class lab1_Java {
public static void main(String[] args) throws IOException {
long stime = System.currentTimeMillis();
BufferedReader br = new BufferedReader(new FileReader(fileName: "src/testdata.txt"));
if (!br.ready()) {
System.out.println("error!");
System.exit(status: 1);
}
BufferedWriter bw = new BufferedWriter(new FileWriter(fileName: "src/result_Java.txt"));
StringBuilder stringBuilder = new StringBuilder();
String s1;
String[] stemp;
1024: success!
TIME: 1.862000s.
TOTAL TIME: 3.080000s.
进程已结束,退出代码0

Haskell：



数据统计：



	2	4	8	16	32	64	128	256	512	1024	TOTAL
C++	0.057138	0.057895	0.060392	0.095807	0.141717	0.283903	0.725042	2.227546	7.892813	30.880625	41.000000
Java	0.027000	0.001000	0.002000	0.005000	0.015000	0.037000	0.054000	0.176000	0.898000	1.515000	2.799000
Python	0.000000	0.000000	0.001005	0.000000	0.004998	0.021995	0.066054	0.279002	1.869239	14.637651	16.893944
Haskell	0.000000	0.001998	0.003000	0.010000	0.020000	0.062500	0.500000	3.734375	32.828120	232.234375	268.000000

五、 语言易用性和程序规模对比分析

语言易用性：先考虑高级语言，从不同的语言分类来说，其中 C++、Java 和 Python 为命令式语言，Haskell 为纯函数式语言。显而易见的，对于不熟悉函数式语言的使用者来说，Haskell 无疑是噩梦。而 C++与 Java 则对初学者来说温和得多，但是想要深入学习 C++与

Java，对于两门极其庞大而规则与特性众多的语言来说，也是非常困难的。

Java 与 Python 都是解释型语言，C++为编译型语言。C++具有学习 C 语言的基础用户，历史悠久，语言易用性强，但是由于历史遗留问题（早先对语言的设计不当等），致使 C++仍有诸多问题。Java 由于为解释型语言，其面向对象编程思想与可跨平台的特性，让其极受欢迎，同时 Java 实现了垃圾回收系统，相较于 C++，不再需要手动维护堆。Python 由其简单的语法规则与众多可调用的库，让编写人员只用注重算法，而在当下人工智能板块等领域越来越受到欢迎。

汇编语言则为低一级的语言，常常与机器硬件有关而不具有跨平台的特性。C、C++语言都会编译为汇编语言再汇编机器语言。因此，汇编语言更适用于底层，在某些情况下，使用汇编语言会达到更快的效率。

在鄙人看来，综合语言易用性为：Python > Java > C++ > Haskell > 汇编

从学习难度来看：Python < C++ < Java < 汇编 < Haskell

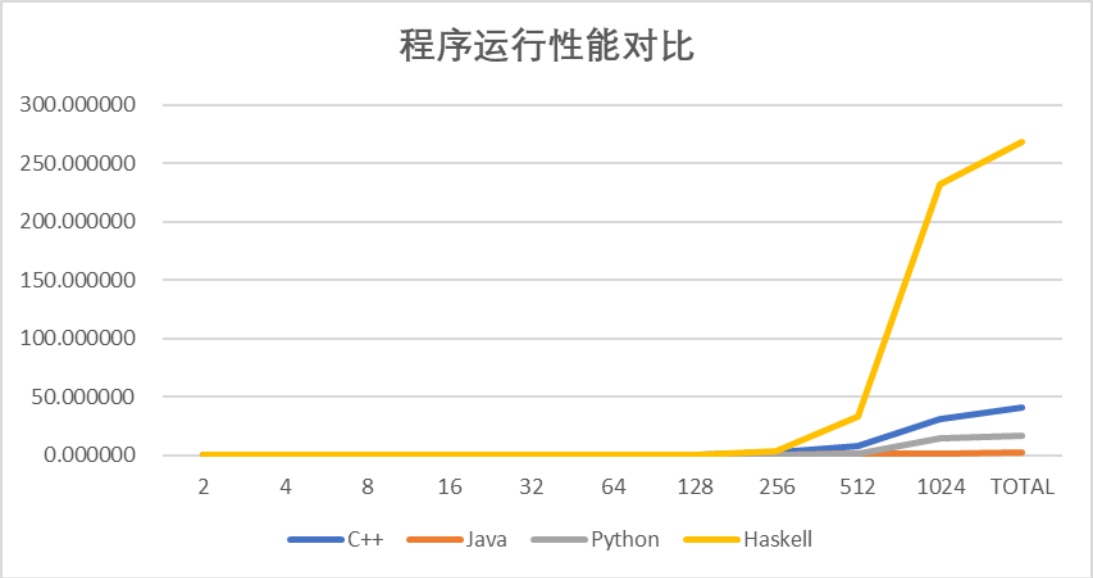
从编程效率来看：Python > Haskell > Java > C++ > 汇编

程序规模对比：

从程序编写/源代码规模来看：Haskell 30 行 < Python 66 行 < Java 80 行 < C++ 116 行 < 汇编

从执行程序大小来看：py(Python) 1.94KB < class(Java) 3.88KB < exe(C++) 1.45MB < exe(Haskell) 12.4MB，但是执行程序由于对应虚拟机（解释型语言）的存在，大小对比意义不大。

六、 程序运行性能对比分析



由于优化与算法实现问题，可以发现，Haskell 只注重该做什么，而不注重如何实现，因此算法时间远大于其他算法。而 Java 优化明显好于 C++与 Python。Python 在使用了库函数矩阵乘法 dot 后，运算速度提升明显。C++则使用了-O2 级别优化，在算法相同的情况下，速度也有了极大提升。

综合程序运行性能对比：Java > Python > C++ > 汇编 > Haskell

七、 实验心得体会

本次实验是《编译原理与设计》课程第一次实验《Lab1-程序设计语言认知实验》，意义非凡，收获颇丰。与以往做过的所有实验有所不同的是，本次实验对五门编程语言进行了回顾、算法实现与横向对比。

对于本人而言，很久没有实际回顾过 C++、Java、Python，因此最开始算法实现颇慢，需要多次回顾语法与参考资料。在进行 Java 与 Python 的编写时，还会由于函数的相似造成误用的尴尬。但总体进展顺利。本次实验的难点在于学习并使用 Haskell 进行编程，Haskell 是一种纯函数型的语言，语法学习起来颇为困难，阅读起来也由于抽象而极为费劲。最终在不断的查阅与尝试后，堪堪写出了一份矩阵相乘算法，但无法读取不同规模的矩阵，因此还需手动修改输入数据(过于愚蠢)。算法的汇编语言实现也由于时间问题而未能完成，只能于实验后进行。