编译原理与设计实验报告

姓名：卜梦煜 学号：1120192419 班级：07111905

# 1. 实验名称

程序设计语言认知实验

# 2. 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。

# 3. 实验内容

分别使用C/C++、Java、Python 和 Haskell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。

# 4. 实验环境

## 4.1硬件配置

CPU核数：8

CPU主频：3.2GHz

内存：16GB

Cache：L1：512KB，L2：4MB，L3：16MB

## 4.2 软件配置

Visual Studio Code 1.64.2.0

C++：gcc 8.1.0

Python：python 3.9.7

Java：jdk-17.0.2

Haskell：ghc-8.0.2

# 5. 实现过程与步骤

四种语言在矩阵乘法功能的实现逻辑上是相似的，可以分为“输入模块”、“运算模块”、“输出模块”、“计时模块”四个部分，输入模块从文件读入矩阵，运算模块进行矩阵乘法运算，输出模块将运算结果存入，计时模块记录程序前三模块运行时间，以毫秒为单位输出。

编写程序完成后，对编译型语言，C++需要利用gcc编译成.exe文件，然后直接执行；Java需要利用jdk编译生成.class文件，然后在JVM运行。对解释型语言，Python调用解释器Python解释执行文件代码。

Haskell语言相比其他三种较特殊，既有编译器ghc，也有解释器ghci。编译运行时，调用编译器ghc编译生成.o文件、.hi文件、.exe文件，然后直接执行.exe文件；解释运行时，ghci会解释.hs文件，并加载Main模块。

## 5.1 C++

输入模块：利用fstream库打开文件“data.txt”，读入矩阵，以int型二维数组存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为。

输出模块：利用fstream库打开文件“result\_C++.txt”，写入运算结果。

计时模块：利用ctime库记录程序开始时间、结束时间，相减得到程序运行时间。

## 5.2 Python

为更好地感受语言本身的效率，没有使用优化较好的Numpy库的Matmul函数，而是使用Python基本的列表操作进行运算。

输入模块：利用内置的open()函数打开文件“data.txt”，读入矩阵，以列表形式存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为。

输出模块：利用open()函数打开文件“result\_python.txt”，写入运算结果。

计时模块：利用time模块记录程序开始时间、结束时间，相减得到程序运行时间。

## 5.3 Java

输入模块：利用java.util.Scanner类打开文件“data.txt”，读入矩阵，以int型二维数组存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为。

输出模块：利用java.io类打开文件“result\_java.txt”，写入运算结果。

计时模块：利用System类记录程序开始时间、结束时间，相减得到程序运行时间。

## 5.4 Haskell

输入模块：利用readFile函数打开文件“data.txt”，编写函数读入矩阵，并处理成Int类型存储。

运算模块：按照矩阵乘法定义编写相应函数，复杂度为。

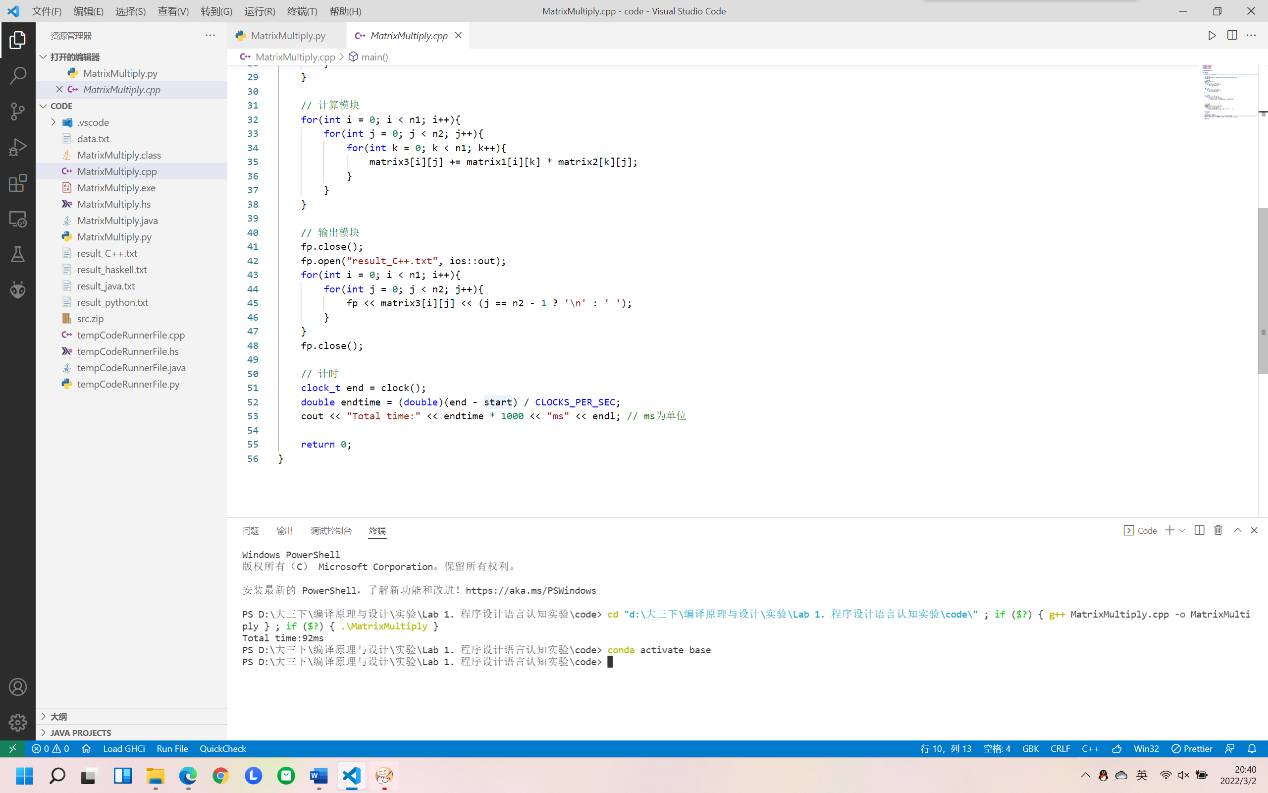
输出模块：将结果加工成字符串，利用writeFile函数打将运算结果写入文件“result\_haskell.txt”。

计时模块：利用Data.Time.Clock库记录程序开始时间、结束时间，利用diffUTCTime函数得到程序运行时间。

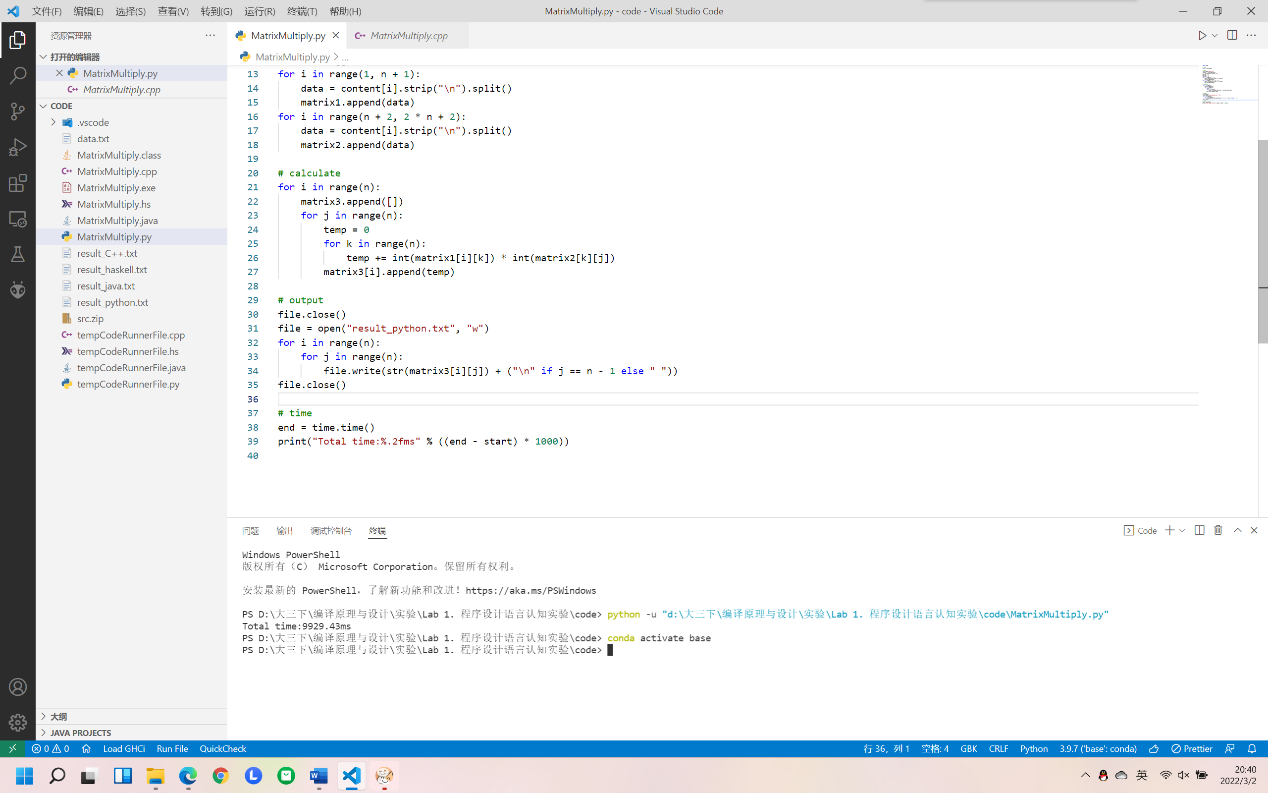
# 6. 运行效果截图

本实验的输入矩阵为300\*300的方阵，由随机数生成。

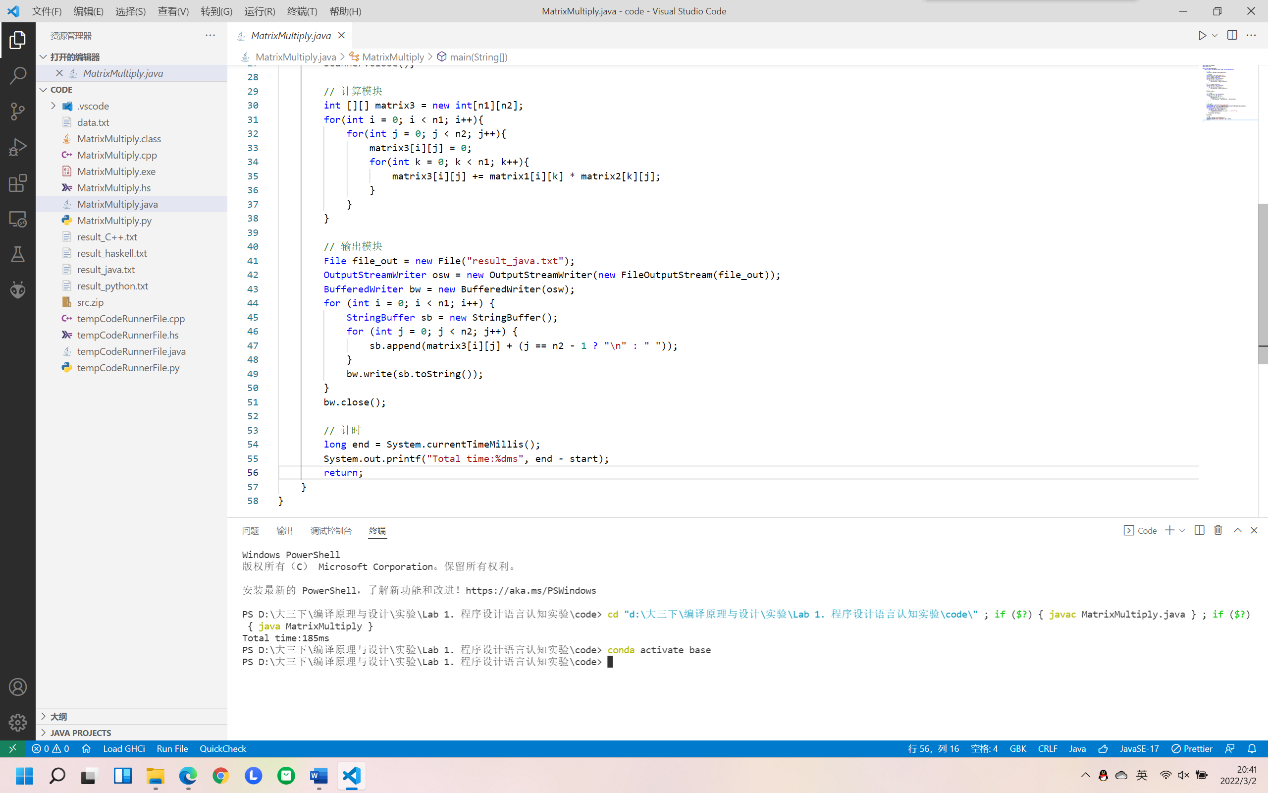
（1）C++运行92ms，运行截图：



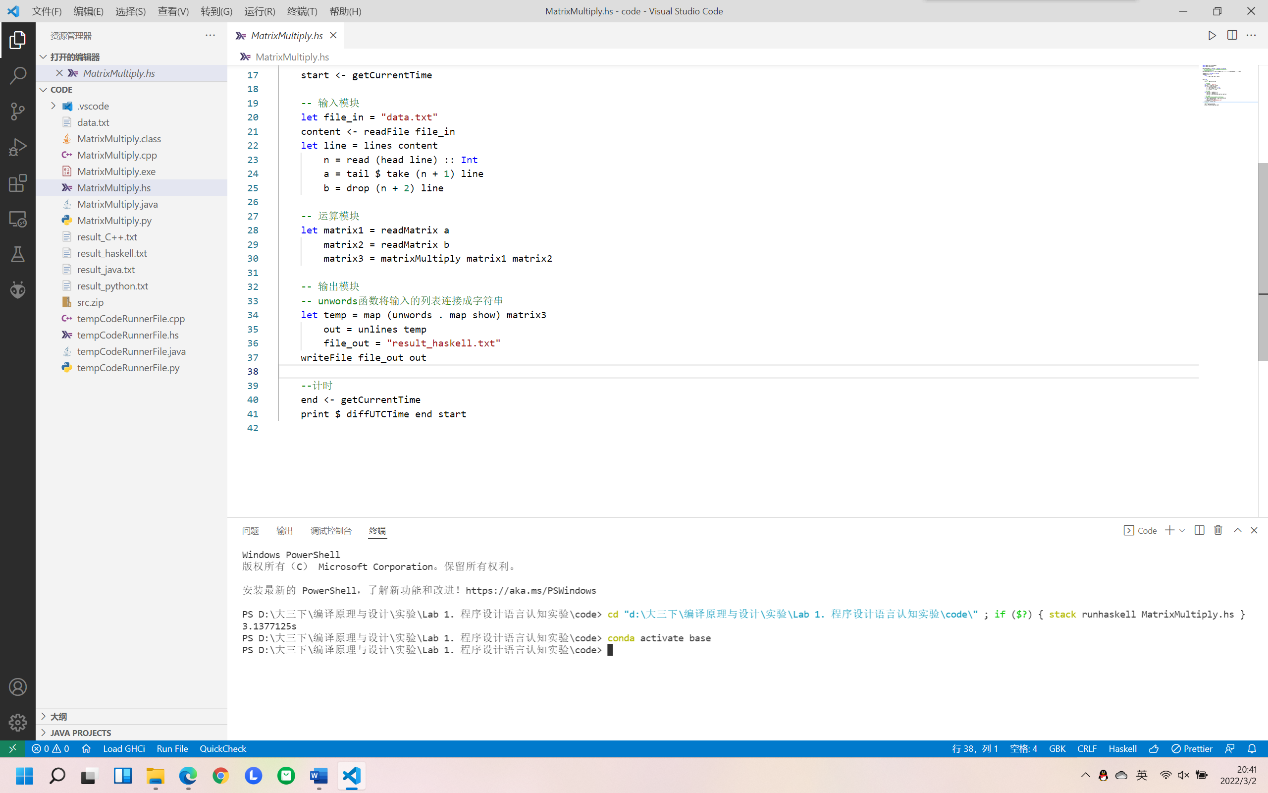
（2）Python运行9.93s，运行截图：



（3）Java运行185ms，运行截图：



（4）Haskell运行3.14s，运行截图：



# 7. 语言易用性与程序规模对比分析

## 7.1 语言易用性

学习难度方面。由于之前学过C++、Python、Java，没学过Haskell，网络上关于C++、Python、Java的教程远比Haskell的教程详细的多，并且Haskell为此前没有接触过的函数式编程。因此，从学习难度与学习成本上看，Haskell的学习难度最大，学习成本最高，函数式编程思想、Haskell语法、Haskell编译环境的安装都需要较多时间。

语言编程效率方面。从执行时间来看，由快到慢顺序依次为C++、Java、Haskell、Python。

## 7.2 程序规模

考虑代码行数，C++为43行，Python为29行，Java为47行，Haskell为25行。因此本实验中，C++、Java程序规模较大，Python、Haskell程序规模较小。

# 8. 程序运行性能对比分析

## 8.1 运行结果

实验条件：输入矩阵为随机数生成300\*300的方阵，重复运行10次取平均值，作为程序运行时间。实验中用到的Python为原生Python，并未使用Numpy库加速。

实验结果如下：



由结果可知，运行速度由快到慢依次为C++、Java、Haskell、Python。

## 8.2 结果分析

1. 编译型语言比解释型语言快。C++、Java、Haskell为编译型语言，程序运行计时并未包括编译二进制文件所需时间，因此速度上比边解释边执行的Python更快。

2. C++比Java快。 C++编译生成的为.exe文件为二进制文件，可直接在机器上运行；而Java编译生成的.class文件需要在JVM虚拟机上运行，因此C++运行速度比Java快。

3. C++比Haskell快。查阅资料得知，这与CPU在函数返回方面的优化有关。gcc会调用指令集里现有的返回指令ret，CPU的分支预测可以准确预测返回地址，流水线不会出现停滞情况；而ghc并不能调用指令集原生的返回指令，而是使用jmp \*指令完成，导致CPU不能准确预测返回地址，流水线经常停滞。这就导致Haskell运行速度比C++慢。

# 9. 实验心得体会

通过这次实验，我有如下收获：

1. 对解释型语言、编译型语言的执行过程有了更深刻的认识，尤其是不同编译型语言生成的中间文件，以及中间文件的运行方式。例如C++与Haskell编译生成的.exe文件是可以直接在机器上运行的，而Java编译生成的.class文件必须在JVM虚拟机上运行。

2. 对四种语言有了更深入的了解，包括代码效率、语法特性、库的调用、运行环境配置等。程序运行速度不仅与运行方式有关，更与程序设计语言本身有关。例如，Haskell函数式编程在表示循环时常用递归实现，而不恰当的递归可能导致运行时间大大增加；Java对各种操作都有封装完整的模块，使用前需多了解，查阅相关文档。

3. 了解了函数式编程的思想与基本方法，能够编写简单代码。