2.1 language

计算器语言同样简单。它不包含任何说明

用于输入，仅用于输出。有可以分配给的变量

并从中读取，但它们仅限于a-z的小写字母的单个字母标识符，因此一个变量不能超过26个

该程序计算732和2684之间的最大公约数

3

使用Euclid的原始算法打印结果（在这种情况下为244）。给定的基于堆栈的编译器将其转换为：

如您所见，堆栈语言同样简单。输出包含“推入/弹出”指令，这些指令将变量a中的值压入堆栈或将值弹出堆栈顶部并将其存储在变量中。运算符同样简单。上面的“ sub”之类的算术运算符从堆栈中弹出两个值，将它们相减并将结果推回堆栈中。因此，计算器语言中的语句“ a = a-b”被转换为；推一个；推b;子弹出a”，以正确的顺序将值“ a”和“ b”放入堆栈中，将它们相减，然后将结果存储回变量“ a”中。由于x86-64'sub'指令不会从堆栈中获取其参数并且不会将其结果放入堆栈中，因此在处理减法情况时，您将不得不添加额外的指令来执行此操作。控制结构包括比较指令“ compGT”表示“ compare than”，它在计算机中的某处设置一个不可见的true或false标志，随后的“ jz”-“如果为零则跳转”指令可以作用，这是唯一的条件跳转指令。因此，适当的编译器（在包含的文件中的calc3b.c文件中）可以通过简单的递归策略来完成其工作，该策略不需要考虑比当前输入令牌更多的内容，即，仅需要检查是否有先前的比较指令即可。返回true或false。

3.requirment

如上所述，您的任务是将calc3b.c更改为calc3i.c，而不是发出伪汇编代码，而是发出实际的x86-64汇编程序指令（使用x86-64堆栈）。您的编译器应发出处理64位带符号整数的代码。

由于提供的编译器仅生成翻译calc语言代码所需的指令，因此它本身不会创建可被编译为可执行文件的汇编程序。之所以如此，是因为它没有定义数据和文本段以及符号表，而这些是处理变量所需的。此外，编译器也不会调用退出函数/系统调用来终止生成的程序。

为了产生可以被编译成可执行表的汇编程序，您将必须编写一个名为“ x86-64-driver.sh”的shell脚本，该脚本将带有“ .calc”的文件作为输入。结束然后：

1.将所需的序言（例如，定义数据和文本段，定义符号表）写入以“ .s”结尾的新文件

2.为给定的“ .calc”文件在文件中附加编译器的输出（即calc3i.exe）

3.附加一个结尾（例如，用于调用退出函数/系统调用）3

之后，驱动程序应调用“ gcc”（或分别为“ as”和“ ld”）进行汇编并链接汇编文件以生成可执行文件。

例如，当我按如下方式运行您的shell脚本：“ x86-64-driver.sh bcd.calc”时，我希望将名为“ bcd.s”的文件作为输出，其中包含该文件产生的x86-64汇编代码“ bcd.calc”以及一个名为“ bcd”的文件，该文件是一个可执行程序，它执行在“ bcd.calc”中编写的内容。

3.3附加功能

该实验室要求您实现对Lex和Yacc原始教程中未描述的三个新“指令”的支持。这些必须以函数调用的形式实现，也就是说，只要编译器处理这些新的指令情况之一，编译器就应该发出对库函数的调用。当然，您还必须提供计算这些新指令的结果的例程的实现（以汇编语言）。

如果要考虑使用更高的标记，则必须将它们实现为适当的外部库（编译为“ .so”或“ .a”文件），发出适当的调用，并将生成的二进制文件与包含实现的库链接（参见第5节）。否则，可以在序言中仅包含每个函数的实现，而无论如何，序言都是程序的一部分。这三个新指令是：

1.事实-接受一个论点并返回该论点的阶乘。例如：0！ = 1。

2. lntwo-接受一个参数，然后返回该参数的二进制对数。例如：lntwo 32 = 5。

3. gcd-接受两个参数，并返回两个参数之间的最大公约数。例如：36 gcd 24 = 12。

您可以自由选择这些（和其他）算法所需的任何明智的实现。