编程语言basic——这个名字是初学者通用符号指令代码的首字母缩写——是由约翰·凯梅尼和托马斯·库尔茨于20世纪60年代中期在达特茅斯学院开发的。 它是最早被设计成易于使用和学习的语言之一。 尽管作为一种教学语言，BASIC现在已经几乎消失了，但它的思想仍然存在于微软的Visual BASIC系统中，并被广泛使用。

在BASIC语言中，程序由一系列有编号的语句组成，如下面的简单程序所示:

10 REM Program to add two numbers

20 INPUT n1

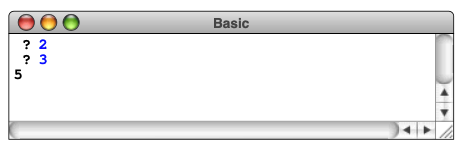
30 INPUT n2

40 LET total = n1 + n2

50 PRINT total

60 END

行首的行号确定了程序中操作的顺序。 在没有任何相反的控制语句的情况下，程序中的语句按照从小到大的数字顺序执行。 例如，在这里，程序执行从第10行开始，这只是一个注释(关键字REM是REMARK的缩写)，表明程序的目的是将两个数字相加。 第20和30行向用户请求两个值，它们分别存储在变量n1和n2中。 第40行中的LET语句是BASIC中赋值的一个例子，它将变量total设置为n1和n2的和。 第50行显示控制台total的值，第60行表示执行的结束。 因此，这个程序的运行示例如下:



行号还用于提供一个简单的编辑机制。 语句不需要按顺序输入，因为行号表示它们的相对位置。 此外，只要用户在数字序列中留下空白，就可以在其他语句之间添加新的语句。 例如，要将两个数字相加的程序改为三个数字相加的程序，你需要做以下更改:

1. Add a new line to read in the third value by typing in the command

35 INPUT n3

1. This statement is inserted into the program between line 30 and line 40. Replace the old line 40 with an update version by typing

40 LET total = n1 + n2 + n3

在经典的BASIC实现中，删除行的标准机制是在行上输入行号后不加任何内容。 注意，此操作实际上删除了该行，而不是简单地将其替换为程序清单中出现的空白行。

表达式的基本

加法程序的第40行演示的LET语句具有一般形式，

LET variable = expression

并且具有将表达式的结果赋值给变量的效果。 在Minimal BASIC中，赋值操作符不再是表达式结构的一部分。 最简单的表达式是变量和整数常量。 这些表达式可以通过将表达式括在括号中或用+、-、\*和/连接两个表达式来组合成更大的表达式。 你只需要在表达式中支持+，-，\*，/，(，)操作符和有符号整数(至少32位)。 (要注意负整数。)

此外，你需要支持表达式中的求幂运算符:

exp1 \*\* exp2

The exponentiation operator returns the result of exp1exp2,，其中exp1和exp2是表达式。 求幂运算符是右结合的，即a \*\* b \*\* c = a \*\* (b \*\* c)，该运算符的优先级高于\*和/。

对于所有表达式和语句，都需要处理额外的空格。**LET a = b + 4 \* (-5 + 4 ).**

BASIC控制语句

加法程序中的语句说明了如何将BASIC用于简单的顺序程序。 如果你想在BASIC程序中表达循环或条件执行，你必须使用GOTO和If语句。 该声明

GOTO *n*

无条件地将控制转移到程序中的第n行。 如果n行不存在，BASIC解释器应该生成一条错误消息，通知用户这一事实。

该声明

IF condition THEN n

执行控制的条件转移。 遇到这样的语句时，BASIC解释器首先求条件，在BASIC的最小版本中，条件由两个算术表达式组成，其中一个运算符是<、>或=。 如果比较结果为真，控制传递到第n行，就像GOTO语句中那样; 如果不是，程序继续按顺序执行下一行。

例如，下面的BASIC程序模拟了从10到0的倒数:

10 REM Program to simulate a countdown

20 LET T = 10

30 IF T < 0 THEN 70

40 PRINT T

50 LET T = T - 1

60 GOTO 30

70 END

尽管GOTO和IF足以表达任何循环结构，但它们代表了比c++中可用的低得多的控制功能，并且往往使BASIC程序更难阅读。 用更高层次的结构(如if/else、while和for)替换这些低级形式代表了软件技术的重大进步，允许程序更接近于程序员的控制结构的心理模型。

在最小的BASIC解释器中可用的语句摘要

最小的BASIC解释器只实现了6种语句形式，见表1。 在不输入行号的情况下，可以直接执行LET、PRINT和INPUT语句，在这种情况下，它们会立即被求值。 因此，如果你输入(就像微软联合创始人保罗·艾伦在第一次演示Altair的BASIC时所做的那样)

PRINT 2 + 2

您的程序应该立即响应4。 语句GOTO、IF、REM和END只有在它们作为程序的一部分出现时才合法，这意味着必须给它们一个行号。

BASIC解释器可以识别的命令

除了表1中列出的语句外，BASIC还接受表2中所示的命令。 这些命令不能是程序的一部分，因此必须在没有行号的情况下输入。

表1。 在BASIC的最小版本中实现的语句

语法树显示

从概念上讲，语法树是程序的一种抽象表示。 更具体地说，程序中的每个语句都可以表示为一个树。 语法树的结构可以看作是语句中表达式计算的步骤。

正如你从以前的编程课程中学到的，一些语句有副作用，在我们的迷你基本语言中，副作用包括赋值和分支。 这些副作用也应该显示在语法树中。

树的节点可以是标识符定义、赋值、函数调用(在此项目中不需要实现函数调用)、表达式计算和条件或无条件分支。 在解释器实现中，可以沿着语法树从叶节点到根节点进行计算。 因为树的连接结构是由计算规则决定的，例如运算符优先级和关联。

在实现中，应该用中缀表示法构造语法树。

为了在GUI窗口中轻松地显示语法树结构，您不需要绘制真正的树。 相反，应该使用缩进来显示每个语句的语法树。

下面是一些具体的例子。 红色边框中的结构是该语句的表达式。

储存

您需要承担的第一个任务是使BASIC解释器能够存储程序。 当您输入以行号开头的行时，例如

你的解释器必须将这一行存储在它的内部数据结构中，这样它就成为了当前程序的一部分。 在输入图2中程序的其余行时，实现中的数据结构必须添加新行并跟踪序列。 特别是当您通过输入来纠正程序时

**Hints on the statement class hierarchy**

表达式的主类应该是Expression，它是一个层次结构的抽象超类，该层次结构包含三个具体的子类，用于三种不同的表达式类型，如下所示:

 这些语句的结构非常相似。 主类应该是Statement，它是对应于每个语句类型的一组子类的抽象超类，如下图所示:

尽管在Statement层次结构中有更多的子类，但它仍然比Expression层次结构更容易实现。 使Expression层次结构复杂但也很强大的原因之一是它是递归的。 复合表达式包含其他表达式，这使得创建任意复杂度的表达式树成为可能。 虽然像c++这样的现代语言中的语句是递归的，但BASIC中的语句不是。

战略和战术

当你完成这个项目时，你可能想要记住以下几点建议:

•比尔•盖茨(Bill Gates)在一份虚构的备忘录中，最后一行鼓励他的团队“尽快开始这个项目”。 我鼓励你们采取同样的战略。

•你需要首先创建一个项目并初始化GUI。 您可以使用Qt Creator的UI编辑器或使用c++代码放置UI小部件。 要完成Minimal BASIC项目，你需要通过精心的分阶段编辑战略性地进行。 无论在何种程度上，您都应该确保BASIC项目在实现的每个阶段完成时继续运行。

•在开始扩展之前，确保项目已经开始工作。 很容易在一开始就过于雄心勃勃，最后得到一堆无法调试的代码。

**Grading**

•(10’)你的解释器应该能够呈现GUI并与用户输入进行交互。

o gui应该包含图2所示的输入和输出接口。

• (20’)你的解释器应该能够加载和编辑基本的程序。

o用户可通过输入框或LOAD按钮添加、更新或删除语句。

o用户输入的语句可按正确顺序存储和显示。

•(50’)你的解释器应该能够正确地解释基本的程序。

o表达式解析(显示语法树，虽然这应该在你存储程序时完成);

o表达式计算和语句执行(如果存在则显示打印结果);

o运行时上下文维护(例如，当前要执行的行，所有变量及其值)。

•(10’)你的解释器应该是健壮的，并且能正确处理输入中的错误。

•(10’)你应该完成面向对象的设计和实现项目; 您的代码应该清晰、易于阅读，并带有适当的注释。