EvalState 类：

该类通过引用通过求值器的递归级别传递，并包含来自求值器可能需要知道的评估环境的信息。在这个版本中，EvalState类维护的惟一信息是一个符号表，它将变量名映射到它们的值中。但是，有几个练习要求您在EvalState类中包含其他信息。

state.setValue(var, value); 设置与指定的var相关的值

int value = state.getValue(var); 获取对应数值

if (state.isDefined(var)) 如果定义了指定的变量，则返回true。

private: Map<std::string , Int> symbolTable;

parser.h：这个文件充当解析器模块的接口

Expression \*exp = parseExp(scanner); 通过从scanner读取tokens来解析表达式，这些tokens必须由客户机提供。扫描仪应该设置为忽略空格和扫描数字。

Expression \*exp = readE(scanner, prec); 从扫描器返回下一个表达式，该表达式只涉及优先级至少为prec的操作符。prec参数是可选的，默认值为0，这意味着函数读取整个表达式。

Expression \*exp = readT(scanner); 返回下一个单独的术语，该术语可以是常量、标识符或带括号的子表达式。

int prec = precedence(token); 返回指定操作符令牌的优先级。如果令牌不是操作符，优先级返回0。

Expression 类： 常数、运算标志、组合式

int value = exp->eval(state); 计算该表达式并在指定的EvalState对象上下文中返回其值 默认为0

string str = exp->toString(); 返回这个式子的string 默认为0

ExpressionType type = exp->getType(); 返回这个式子的类型，常、运、组

ConstantExp 类： 常数表达式

ConstantExp(int value); 构造函数将一个新的整数常量表达式初始化为给定值。

virtual int eval(EvalState & state);

virtual std::string toString();

virtual ExpressionType getType(); 以上三个与Exp类型之中的用法

相同

int value = ((ConstantExp \*) exp)->getValue(); 返回值字段而不调用eval，并且只能应用于已知为ConstantExp的对象。

IdentifierExp 类型：

Expression \*exp = new IdentifierExp(name); 构造函数将一个新的标志表达式初始化为给定值。

virtual int eval(EvalState & state);

virtual std::string toString();

virtual ExpressionType getType(); 以上三个与Exp类型之中的用法

相同

string name = ((IdentifierExp \*) exp)->getName(); 返回标识符节点的名称字段，只能应用于已知为IdentifierExp的对象。

CompoundExp 类型：

Expression \*exp = new CompoundExp(op, lhs, rhs); 构造函数初始化一个新的复合表达式，该表达式由操作符(string op)和左右子表达式(Expression \* lhs和\*rhs)组成。

virtual ~CompoundExp();

virtual int eval(EvalState & state);

virtual std::string toString();

virtual ExpressionType getType(); 以上三个与Exp类型之中的用法

相同

string op = ((CompoundExp \*) exp)->getOp();

Expression \*lhs = ((CompoundExp \*) exp)->getLHS();

Expression \*rhs = ((CompoundExp \*) exp)->getRHS();

Program.h :

该类将行存储在一个基本程序中。程序中的每一行都是按照它的行号顺序存储的。此外，程序中的每一行都与两个组件相关联:

1。源行，即用户输入的完整行(包括行号)。

2。该语句的经过解析的表示形式，它是一个指向语句的指针。

Program 类型：

Program program; 构造初始

program.clear(); 清除所有的存储

program.addSourceLine(lineNumber, line); 用指定的行号向程序添加源行。如果该行已经存在，该行的文本将替换任何现有行的文本，并删除解析后的表示(如果有)。如果行是新的，则按正确的顺序添加到程序中。

program.removeSourceLine(lineNumber); 从程序中删除具有指定数字的行，释放与任何已解析表示相关联的内存。如果不存在这样的行，该方法只返回，不执行任何操作。

string line = program.getSourceLine(lineNumber); 返回对应的的行，如果没有这一行则返回空的string类型

program.setParsedStatement(lineNumber, stmt); 将经过解析的语句表示形式添加到指定行号的语句中。如果不存在这样的行，则此方法将引发错误。如果前面的解析表示存在，则会回收该语句的内存。

Statement \*stmt = program.getParsedStatement(lineNumber); 检索语句在指定行号处的已解析表示。如果没有设置值，该方法返回NULL。

int lineNumber = program.getFirstLineNumber(); 返回这个程序的第一个行数字，如果一行没有则返回-1

int nextLine = program.getNextLineNumber(lineNumber); 返回所给的行数字的下一个行数字，如果没有了返回-1

Statement 类型：

该类用于表示程序中的语句。该类的模型是express .h接口中的表达式。与表达式一样，Statement也是一个抽象类，它具有基本解释器所需的每个语句和命令类型的子类。

stmt->execute(state); 该方法执行一条基本语句。每个子类都定义了自己的执行方法来实现必要的操作。与表达式求值器一样，该方法采用EvalState对象来查找变量或控制解释器的操作。

该文件的其余部分必须包含各个语句表单的子类定义。这些子类都必须定义一个构造函数，该构造函数解析来自扫描器的语句，并定义一个名为execute的方法，该方法执行该语句。如果子类的私有数据包括堆上分配的数据(例如表达式对象)，那么类实现还必须指定自己的析构函数方法来释放该内存。

TokenScanner 类型：

构造函数和析构函数

scanner.setInput(str); scanner.setInput(infile); 将此扫描程序的令牌流设置为指定字符串或输入流。以前的任何令牌流都将被丢弃。

if (scanner.hasMoreTokens()) 返回是否下面还有token 可以读

token = scanner.nextToken(); 返回下一个token，如果没有了返回空string

scanner.saveToken(token); 将指定的令牌推回扫描程序的输入流。在下一次调用nextToken时，扫描程序将返回保存的令牌，而不会从令牌流中读取任何额外字符。

int pos = scanner.getPosition(); 返回扫描器在输入流中的当前位置。如果saveToken已被调用，则此位置对应于已保存令牌的开头。如果saveToken被多次调用，getPosition返回-1。

scanner.ignoreWhitespace(); 告诉扫描程序忽略空格字符。默认情况下，nextToken方法对待空格字符(通常是空格和制表符)就像对待其他标点符号一样，并将它们作为单字符标记返回。更改此行为，以便扫描程序忽略空格字符

scanner.ignoreComments(); 告诉扫描程序忽略注释。扫描器包可以识别来自基于c的语言家族的斜杠-星号和斜杠-斜杠注释格式。更改此行为，以便扫描程序忽略注释

scanner.scanNumbers(); 控制扫描程序如何处理以数字开头的令牌。默认情况下，nextToken方法对数字和字母的处理是相同的，因此不对数字提供任何特殊处理。调用scanner.scanNumbers();更改此行为，以便nextToken返回可解释为实数的最长子字符串。

scanner.scanStrings(); 控制扫描器如何处理括在引号中的令牌。默认情况下，引号(单引号或双引号)的处理方式与其他标点字符相同。调用scanner.scanStrings ();改变这个假设，使nextToken通过匹配的引号返回一个由所有字符组成的令牌。引号作为扫描令牌的一部分返回，以便客户机能够将字符串与其他令牌类型区分开来。

scanner.addWordCharacters(str); 将str中的字符添加到单词令牌中的合法字符集。例如，调用addWordCharacters(“\_”)将下划线添加到作为单词一部分接受的字符集中。

if (scanner.isWordCharacter(ch)) .

\* Returns <code>true</code> if the character is valid in a word.