

# 让我们构建一个简单的解释器。第 9 部分。 (<https://ruslanspivak.com/lbasi-part9/>)

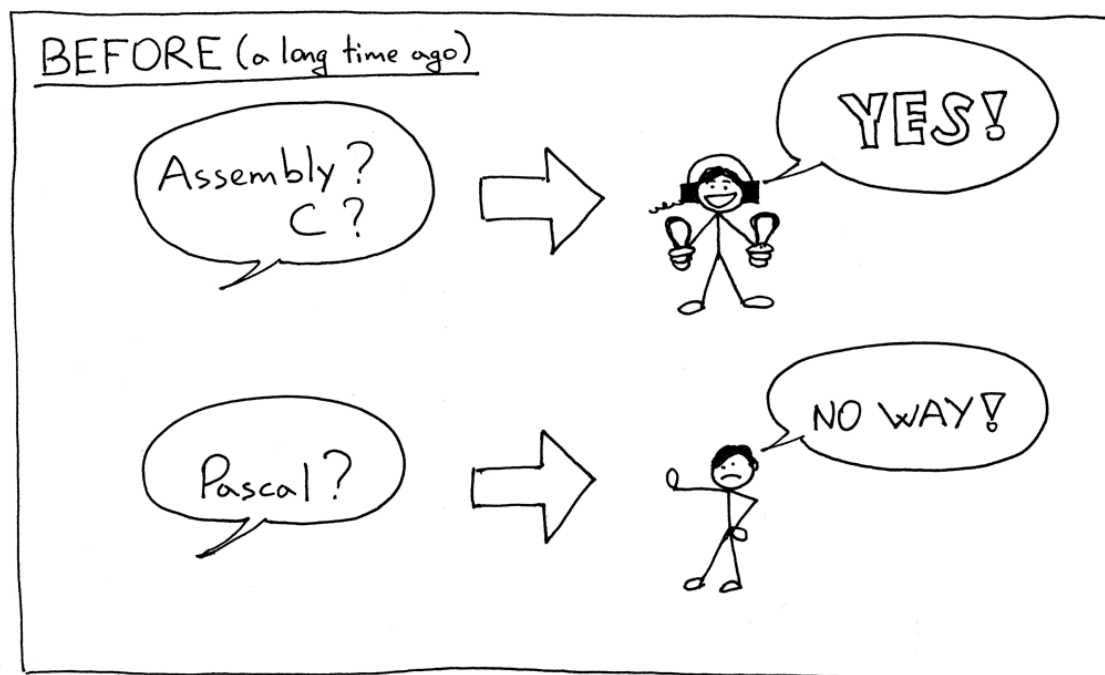
日期 2016 年 5 月 1 日, 星期日

我记得当我在大学（很久以前）学习系统编程时，我相信唯一“真正的”语言是汇编和 C。而 Pascal 是——怎么说好一点——一种非常高级的语言不想知道幕后发生了什么的应用程序开发人员。

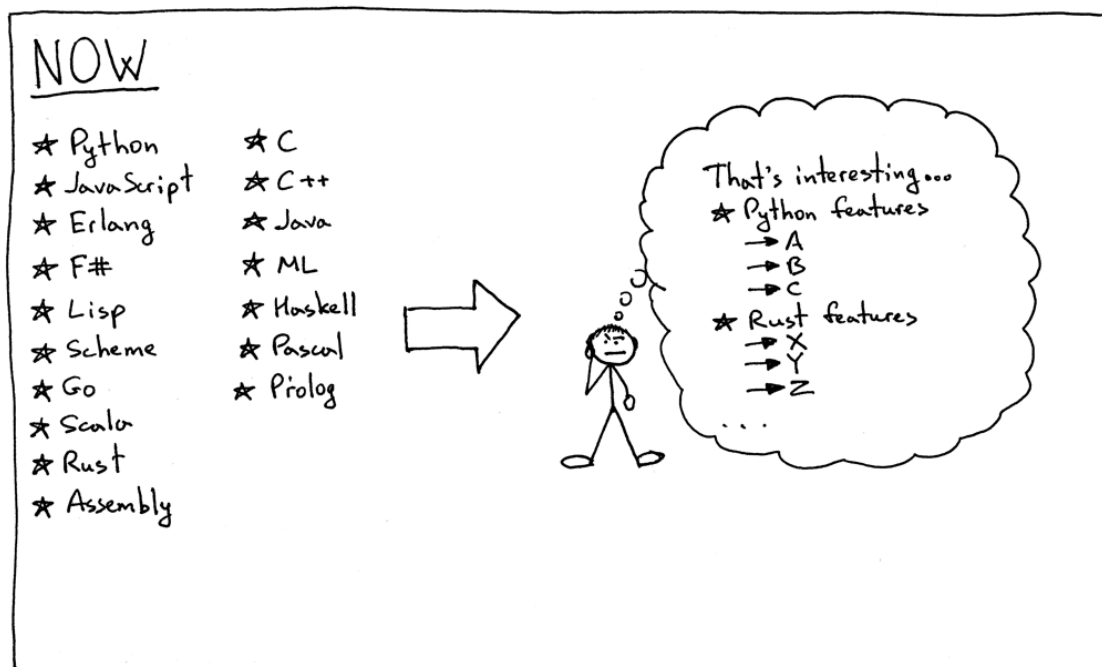
那时我几乎不知道我会用 Python 编写几乎所有东西（并且喜欢它的每一点）来支付我的账单，而且我还会因为我在第一篇文章中 ([/lbasi-part1/](#))提到的原因为 Pascal 编写解释器和编译器该系列的 ([/lbasi-part1/](#))。

这些天，我认为自己是一个编程语言爱好者，我对所有语言及其独特的功能都很着迷。话虽如此，我必须指出，我比其他语言更喜欢使用某些语言。我有偏见，我将是第一个承认这一点的人。:)

这是我之前：



现在：



好的，让我们进入正题。以下是您今天要学习的内容：

1. 如何解析和解释 Pascal 程序定义。
2. 如何解析和解释复合语句。
3. 如何解析和解释赋值语句，包括变量。
4. 关于符号表以及如何存储和查找变量的一些知识。

我将使用以下示例 Pascal-like 程序来介绍新概念：

```
BEGIN
  BEGIN
    号 := 2 ;
    一个 := 数字;
    b := 10 * a + 10 * 数字 / 4 ;
    c := a - - b
  END ;
  x := 11 ;
结束。
```

您可以说，按照本系列的前几篇文章，到目前为止您编写的命令行解释器是一个很大的跳跃，但我希望这种跳跃会带来兴奋。它不再“只是”一个计算器，我们在这里变得认真了，Pascal 是认真的。:)

让我们深入了解新语言结构的语法图及其相应的语法规则。

在你的标记上：准备好了。放。走！

## GRAMMAR RULE

3/21

1. 我将从描述什么是 Pascal 程序开始。Pascal 程序由一个以点结尾的复合语句组成。下面是一个程序示例：

“开始结束。”

我必须注意，这不是一个完整的程序定义，我们将在本系列的后面对其进行扩展。

2. 什么是复合语句？甲**复合语句**是标有块 BEGIN 和 END 语句包括其它复合语句，它可以包含一个列表（可能为空）。复合语句中的每一条语句，除了最后一条，都必须以分号结束。块中的最后一条语句可能有也可能没有终止分号。以下是一些有效复合语句的示例：

“开始结束”  
 “开始一个 := 5; x := 11 结束”  
 “开始一个 := 5; x := 11; 结尾”  
 “BEGIN BEGIN a := 5 END; x := 11 结束”

3. 一个**语句列表**是一个复合语句中的零条或多个语句的列表。有关一些示例，请参见上文。  
 4. 甲**语句**可以是一个复合语句，一个赋值语句，或者它可以是一个空的 语句。  
 5. 一个**赋值语句**是一个变量后面是 ASSIGN 令牌（两个字符，“:” 和 “=”），随后的表达式。

“一个: = 11”  
 “b := a + 9 - 5 \* 2”

6. 甲**变量**是一个标识符。我们将使用 ID 令牌作为变量。令牌的值将是变量的名称，如 “a”、“number” 等。在以下代码块中，'a' 和 'b' 是变量：

“开始一个 := 11; b := a + 9 - 5 \* 2 结束”

7. 一个**空的**声明表示，没有进一步的制作语法规则。我们使用 empty\_statement 语法规则来指示 解析器中 statement\_list 的结尾，并允许像 “BEGIN END” 中那样的空复合语句。  
 8. 该**系数**的规则进行更新处理变量。

现在让我们来看看我们完整的语法：

程序：复合语句 DOT

Compound\_statement : BEGIN statement\_list END

statement\_list : 语句  
                   | statement SEMI statement\_list

语句：复合语句  
       | 赋值语句  
       | 空的

assignment\_statement : 变量 ASSIGN expr

空的 :

expr: term ((PLUS | MINUS) term)\*

术语: 因子 ((MUL | DIV) 因子)\*

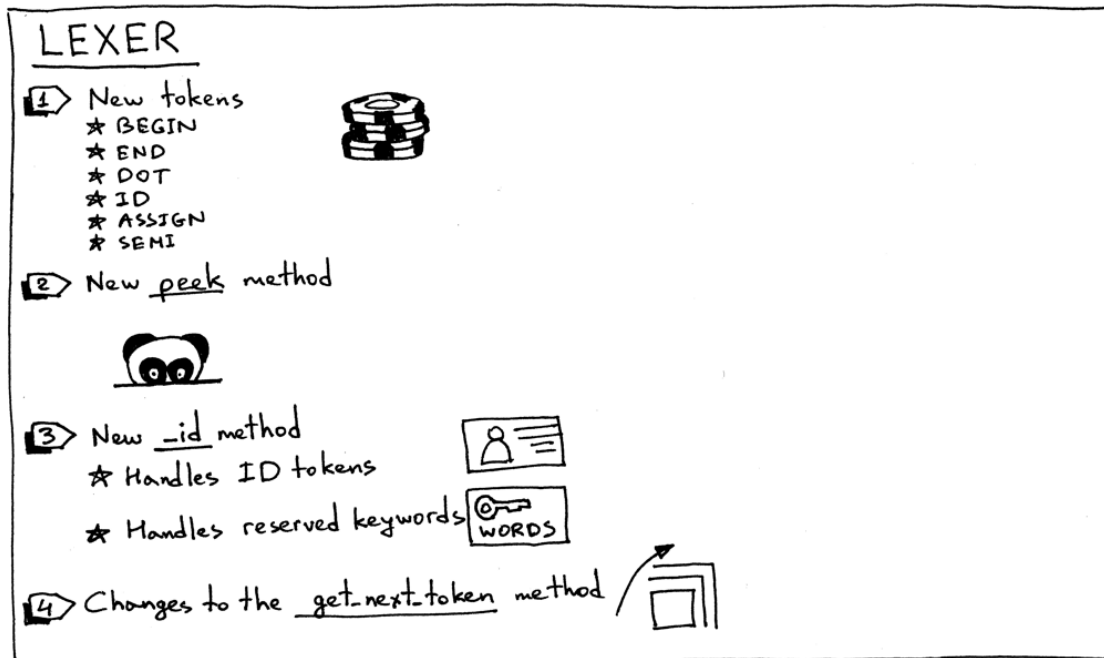
因素: 加因素  
       | 减因子  
       | 整数  
       | LPAREN expr RPAREN  
       | 多变的

变量: ID

您可能已经注意到，我没有在复合语句规则中使用星号 “\*” 来表示零次或多次重复，而是明确指定了 `statement_list` 规则。这是表示 “零或多个” 操作的另一种方式，当我们查看本系列后面的 `PLY` (<http://www.dabeaz.com/ply/>) 等解析器生成器时，它会派上用场。我还将 “( PLUS | MINUS ) factor ” 子规则拆分为两个单独的规则。 (<http://www.dabeaz.com/ply/>)

In order to support the updated grammar, we need to make a number of changes to our lexer, parser, and interpreter. Let's go over those changes one by one.

Here is the summary of the changes in our lexer:



1. To support a Pascal program's definition, compound statements, assignment statements, and variables, our lexer needs to return new tokens:

- BEGIN (to mark the beginning of a compound statement)
- END (to mark the end of the compound statement)
- DOT (a token for a dot character '.' required by a Pascal program's definition)
- ASSIGN (a token for a two character sequence ':=' ). In Pascal, an assignment operator is different than in many other languages like C, Python, Java, Rust, or Go, where you would use single character '=' to indicate assignment
- SEMI (a token for a semicolon character ';' that is used to mark the end of a statement inside a compound statement)
- ID (A token for a valid identifier. Identifiers start with an alphabetical character followed by any number of alphanumerical characters)

2. Sometimes, in order to be able to differentiate between different tokens that start with the same character, ( '.' vs ':=' or '==' vs '>=' ) we need to peek into the input buffer without actually consuming the next character. For this particular purpose, I introduced a peek method that will help us tokenize assignment statements. The method is not strictly required, but I thought I would introduce it earlier in the series and it will also make the `get_next_token` method a bit cleaner. All it does is return the next character from the text buffer without incrementing the `self.pos` variable. Here is the method itself:

```
def peek(self):
    peek_pos = self.pos + 1
    if peek_pos > len(self.text) - 1:
        return None
    else:
        return self.text[peek_pos]
```

3. Because Pascal variables and reserved keywords are both identifiers, we will combine their handling into one method called `_id`. The way it works is that the lexer consumes a sequence of alphanumerical characters and then checks if the character sequence is a reserved word. If it is, it returns a pre-constructed token for that reserved keyword. And if it's not a reserved keyword, it returns a new ID token whose value is the character string (lexeme). I bet at this point you think, "Gosh, just show me the code." :) Here it is:

```
RESERVED_KEYWORDS = {
    'BEGIN': Token('BEGIN', 'BEGIN'),
    'END': Token('END', 'END'),
}

def _id(self):
    """Handle identifiers and reserved keywords"""
    result = ''
    while self.current_char is not None and self.current_char.isalnum():
        result += self.current_char
        self.advance()

    token = RESERVED_KEYWORDS.get(result, Token(ID, result))
    return token
```

4. And now let's take a look at the changes in the main lexer method `get_next_token`:

```
def get_next_token(self):
    while self.current_char is not None:
        ...
        if self.current_char.isalpha():
            return self._id()

        if self.current_char == ':' and self.peek() == '=':
            self.advance()
            self.advance()
            return Token(ASSIGN, ':=')

        if self.current_char == ';':
            self.advance()
            return Token(SEMI, ';')

        如果 自. current_char == '.' :
            自己. 提前()
            返回 令牌( DOT , '.' )

        ...
```

是时候看看我们闪亮的新词法分析器的所有荣耀和动作了。从 [GitHub](https://github.com/rspivak/lbasi)

(<https://github.com/rspivak/lbasi/blob/master/part9/python>) 下载源代码并保存 `spi.py`

(<https://github.com/rspivak/lbasi/blob/master/part9/python/spi.py>) 文件的同一目录启动 Python shell :

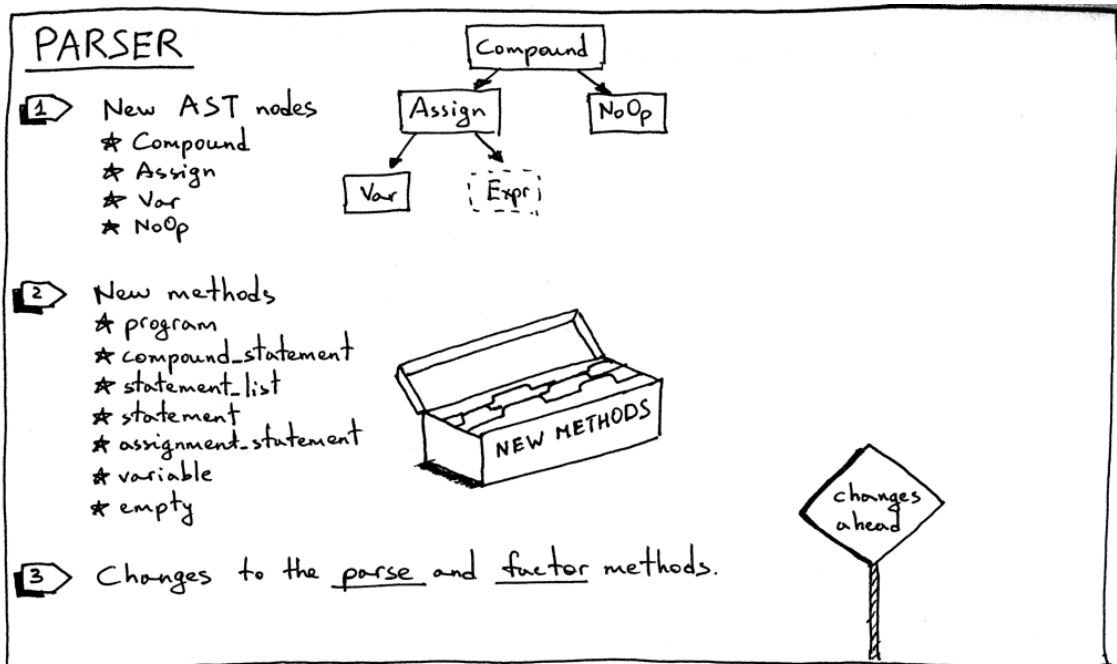
```

>>> 从 spi 导入词法分析器
>>> lexer = Lexer ( 'BEGIN a := 2; END.' )
>>> lexer.get_next_token ()
Token ( BEGIN, 'BEGIN' )
>>> lexer.get_next_token ()
Token ( ID, 'a' )
>>> lexer.get_next_token ()
令牌( ASSIGN, ':' )
>>> lexer.get_next_token ()
令牌( INTEGER, 2 )
>>> lexer.get_next_token ()
令牌( SEMI, ';' )
>>> lexer.get_next_token()
令牌( END, 'END' )
>>> lexer.get_next_token ()
令牌( DOT, '.' )
>>> lexer.get_next_token ()
令牌( EOF, None )
>>>

```

继续解析器更改。

以下是我们的解析器更改的摘要：



1. 让我们从新的AST 节点开始：

- 复合 AST节点表示复合语句。它在其children 变量中包含一个语句节点列表。

```

class Compound ( AST ):
    """代表一个 'BEGIN ... END' 块"""
    def __init__ ( self ):
        self.儿童 = []

```

- Assign AST节点代表一个赋值语句。它的左变量用于存储一个Var节点，它的右变量用于存储由 expr 解析器方法返回的节点：

```

class Assign ( AST ):
    def __init__ ( self , left , op , right ):
        self.左 = 左
        self.令牌 = 自我.操作 = 操作
        self.正确 = 正确

```

- Var AST节点（你猜对了）代表一个变量。该self.value持有变量的名称。

```
class Var ( AST ):  
    """Var 节点是由 ID 标记构建的。"""  
    def __init__ ( self , token ):  
        self 。令牌 = 令牌  
        self 。价值 = 令牌。价值
```

- NoOp节点用于表示空语句。例如, ' BEGIN END ' 是一个没有语句的有效复合语句。

```
类 NoOp ( AST ) :  
    通过
```

2. 您还记得, 语法中的每个规则在我们的递归下降解析器中都有一个对应的方法。这次我们添加了七个新方法。这些方法负责解析新的语言结构和构建新的AST节点。它们非常简单:



```

def program ( self ):
    """program : Compound_statement DOT"""
    node = self . Compound_statement ()
    自我。吃 ( DOT )
    返回 节点

高清 compound_statement (个体经营):
    """
    compound_statement: BEGIN STATEMENT_LIST END
    """
    自我。吃 ( BEGIN )
    节点 = 自我。statement_list ()
    自我。吃 ( 完 )

    root = Compound ()
    用于 节点 中的 节点:
        root 。孩子们。追加 ( 节点 )

    返回 根

def statement_list ( self ):
    """
    statement_list : statement
                    / statement SEMI statement_list
    """
    node = self . 声明()

    结果 = [ 节点 ]

    而 自我。current_token 。类型 == 半:
        自我。吃 ( SEMI )
        结果。追加 ( 自我。声明 ( ) )

    如果 自。current_token 。类型 == ID :
        self 。错误()

    返回 结果

def 语句( self ):
    """
    语句 : Compound_statement
        / assignment_statement
        / 空
    """
    if self 。current_token 。type == BEGIN :
        node = self 。compound_statement ( )
    ELIF 自我。current_token 。类型 == ID :
        节点 = self 。assignment_statement ( )
    else :
        node = self 。空 ( )
    返回 节点

def assignment_statement ( self ):
    """
    assignment_statement : 变量 ASSIGN expr
    """
    left = self 。变量()
    token = self 。current_token
    自我。吃 ( 分配 )
    权 = 自我。expr ( )
    node = Assign ( left , token , right )
    返回 节点

def 变量( self ):
    """

```

```

变量: ID
"""
node = Var ( self . current_token )
self . 吃 ( ID )
返回 节点

def empty ( self ):
    """An empty production"""
    return NoOp ()

```

3. 我们还需要更新现有的factor方法来解析变量:

```

def factor ( self ):
    """factor : PLUS factor
               / MINUS factor
               / INTEGER
               / LPAREN expr RPAREN
               / variable
    """
    token = self . current_token
    如果是 token . 类型 == PLUS :
        自我。吃 ( PLUS )
        节点 = UnaryOp ( 令牌, 自我。因子 ( ) )
        返回 节点
    ...
    别的:
        节点 = 自我。变量()
        返回 节点

```

4. 解析器的parse方法更新为通过解析程序定义来启动解析过程:

```

def 解析 ( self ) :
    node = self . program ()
    如果是 self . current_token . 输入 != EOF :
        self . 错误()

    返回 节点

```

这是我们的示例程序:

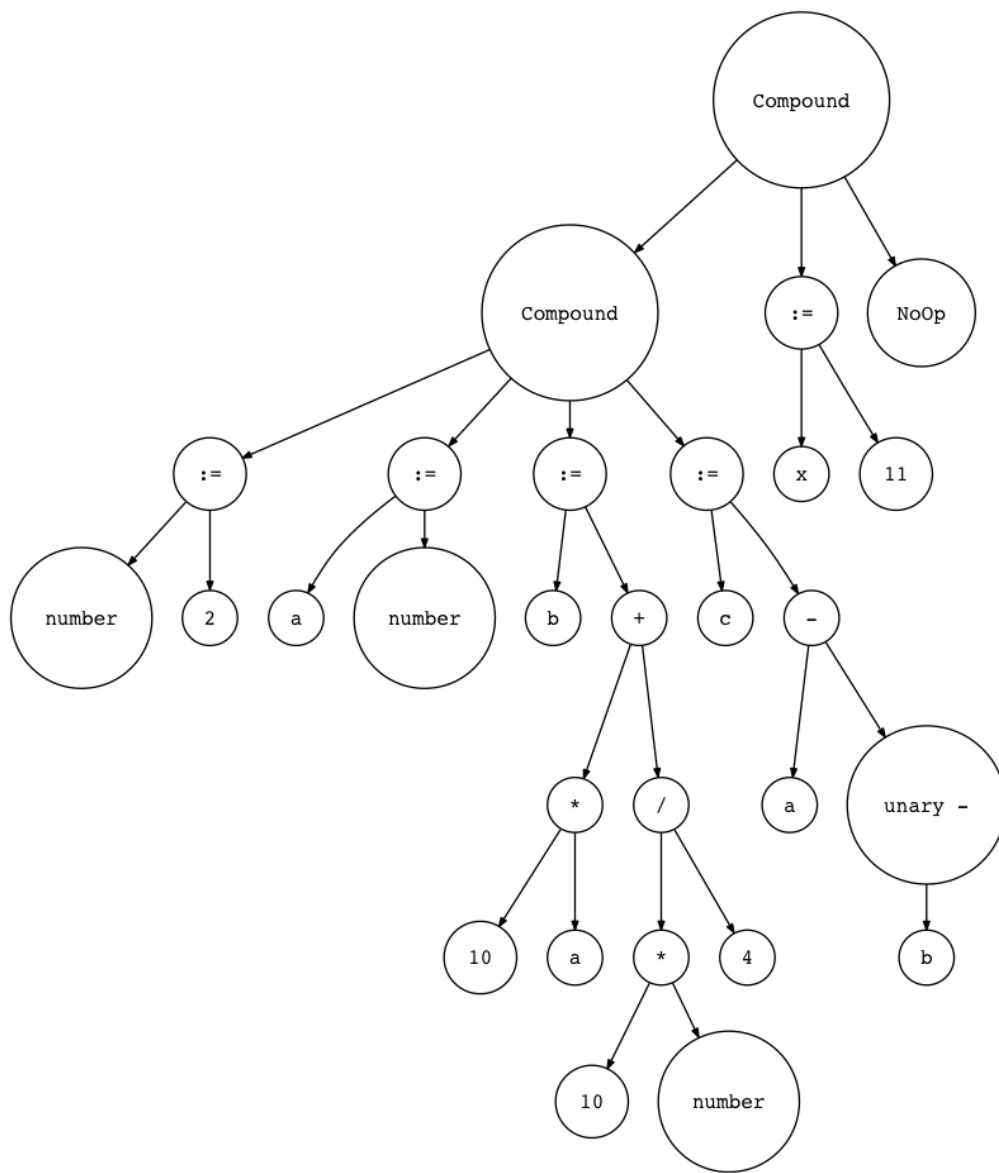
```

BEGIN
  BEGIN
    号 := 2 ;
    一个 := 数字;
    b := 10 * a + 10 * 数字 / 4 ;
    c := a - - b
  END ;
  x := 11 ;
结束。

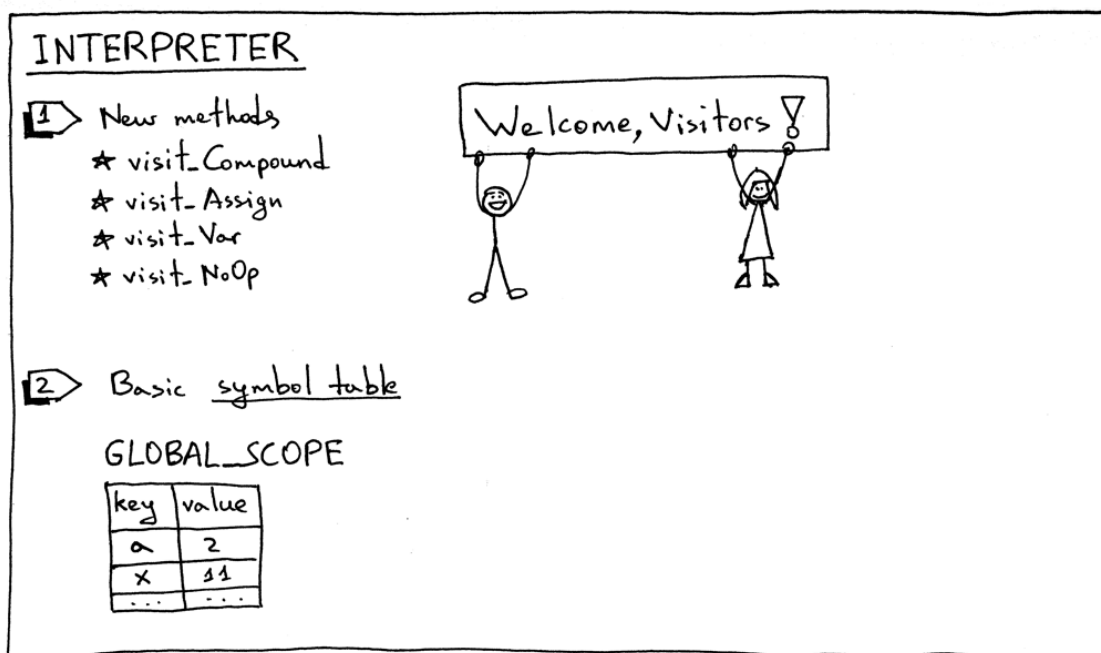
```

让我们用genastdot.py (<https://github.com/rspivak/lbasi/blob/master/part9/python/genastdot.py>)对其进行可视化 (为简洁起见, 当显示Var节点时, 它只显示节点的变量名称, 当显示一个 Assign 节点时, 它显示 ':' 而不是显示 'Assign' 文本) :

```
$ python genastdot.py assignments.txt > ast.dot && dot -Tpng -o ast.png ast.dot
```



最后，这里是所需的解释器更改：



要解释新的AST节点，我们需要向解释器添加相应的访问者方法。有四种新的访问者方法：

- 访问\_化合物

- 访问\_分配
- 访问\_变量
- 访问\_NoOp

Compound和NoOp访问者方法非常简单。该visit\_Compound方法遍历它的孩子和参观各一转，和visit\_NoOp方法不起作用。

```

高清 visit_Compound (自我, 节点):
    为 孩子 的 节点。孩子:
        自我。拜访 (孩子)

def visit_NoOp ( self , node ):
    通过
  
```

在分配和瓦尔游客方法值得仔细研究。

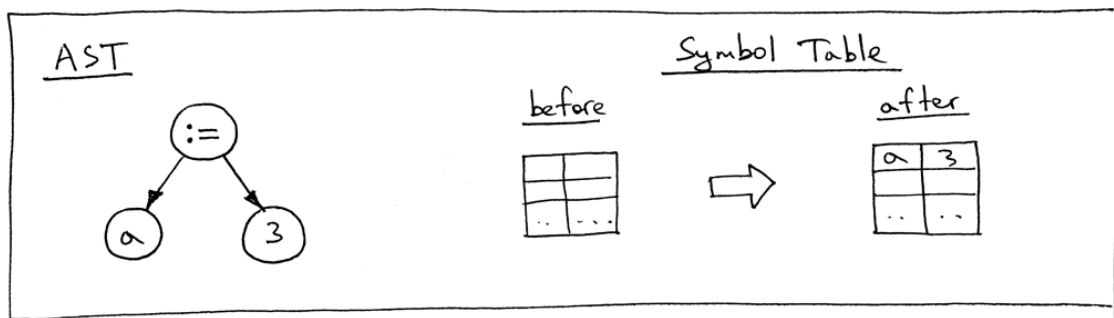
当我们为变量赋值时，我们需要将该值存储在某个地方以备日后需要时使用，这正是visit\_Assign方法所做的：

```

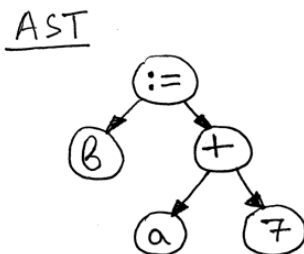
def visit_Assign ( self , node ):
    var_name = node 。离开了。重视
    自我。GLOBAL_SCOPE [ var_name ] = self 。访问 (节点。右)
  
```

该方法在符号表GLOBAL\_SCOPE 中存储键值对（变量名和与变量关联的值）。什么是符号表？**符号表**是一个抽象数据类型（ADT用于在源代码中追踪各种符号）。我们现在唯一的符号类别是变量，我们使用 Python 字典来实现符号表ADT。现在我只想说，本文中符号表的使用方式非常“hacky”：它不是一个具有特殊方法的单独类，而是一个简单的 Python 字典，它还作为内存空间执行双重任务。在以后的文章中，我将更详细地讨论符号表，我们还将一起删除所有的黑客。

让我们看一下语句 “a := 3;” 的AST 和visit\_Assign方法之前和之后的符号表完成它的工作：



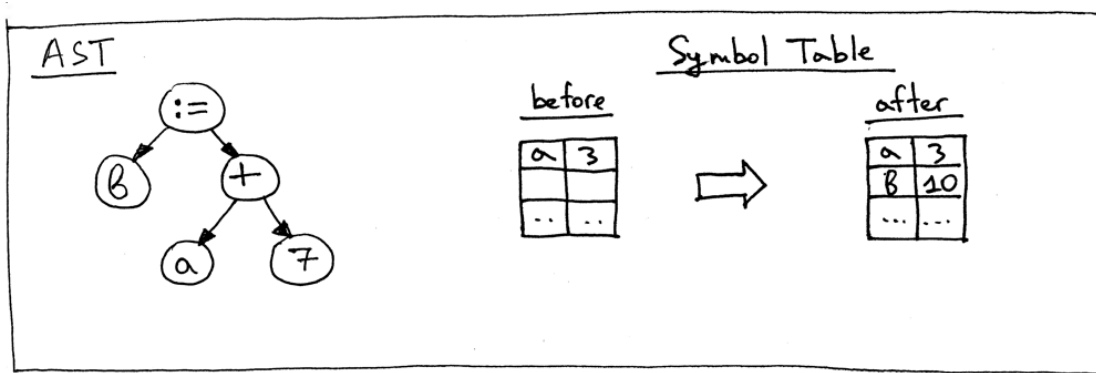
现在让我们看一下语句 “b := a + 7;” 的AST



如您所见，赋值语句的右侧 - “a + 7” - 引用了变量 'a'，因此在我们评估表达式 “a + 7” 之前，我们需要找出 'a' 的值，这是visit\_Var 方法的职责：

```
def visit_Var ( self , node ) :
    var_name = node . 价值
    val = 自我 . GLOBAL_SCOPE . get ( var_name )
    如果 val 是 None :
        引发 NameError ( repr ( var_name ) )
    else :
        return val
```

当该方法访问上面AST图中的Var节点时，它首先获取变量的名称，然后使用该名称作为GLOBAL\_SCOPE字典中的键来获取变量的值。如果它可以找到该值，则返回该值，否则会引发NameError异常。以下是在评估赋值语句“b := a + 7;”之前的符号表内容：



这些都是我们今天需要做的改变，以使我们的解释器打勾。在主程序结束时，我们简单地将符号表GLOBAL\_SCOPE的内容打印到标准输出。

让我们从 Python 交互式 shell 和命令行中使用我们更新的解释器作为驱动器。确保在测试之前下载了解释器的源代码和assignments.txt

(<https://github.com/rspivak/lbasi/blob/master/part9/python/assignments.txt>)文件：

启动你的 Python shell：

```
$蟒蛇
>>> from spi 导入词法分析器、解析器、解释器
>>> text = """\
... BEGIN
...
... BEGIN
... number := 2;
... a := number;
... b := 10 * a + 10 * number / 4;
... c := a - - b
... END;
...
... x := 11;
... END.
... """
>>> lexer = Lexer ( text )
>>> parser = Parser ( lexer )
>>> interpreter = Interpreter ( parser )
>>> interpreter.interpret ()
>>> 打印(interpreter.GLOBAL_SCOPE )
{ 'a' : 2 , 'x' : 11 , 'c' : 27 , 'b' : 25 , 'number' : 2 }
```

从命令行，使用源文件作为我们解释器的输入：

```
$ python spi.py assignments.txt
{ 'a' : 2 , 'x' : 11 , 'c' : 27 , 'b' : 25 , 'number' : 2 }
```


如果您还没有尝试过，现在就尝试一下，亲眼看看解释器是否正确地完成了它的工作。

让我们总结一下你在这篇文章中扩展 Pascal 解释器需要做的事情：

1. 向语法添加新规则
2. 向词法分析器添加新标记和支持方法并更新 `get_next_token` 方法
3. 为新的语言结构向解析器添加新的AST节点
4. 将与新语法规则相对应的新方法添加到我们的递归下降解析器中，并在必要时更新任何现有方法（因子方法，我在看着你。:）
5. 向解释器添加新的访问者方法
6. 添加用于存储变量和查找变量的字典

在这一部分中，我不得不介绍一些“技巧”，我们将随着系列的推进而将其删除：

## HACKS



- 1 Incomplete program definition
- 2 Variables have no declared types
- 3 No type checking
- 4 A basic symbol table that also does double duty as a memory space
- 5 Using the character `'/'` for integer division

1. 该程序的语法规则是不完整的。稍后我们将使用其他元素对其进行扩展。
2. Pascal 是一种静态类型语言，您必须在使用它之前声明一个变量及其类型。但是，正如您所看到的，本文中的情况并非如此。
3. 到目前为止没有类型检查。在这一点上这没什么大不了的，但我只是想明确地提到它。例如，一旦我们向解释器添加更多类型，当您尝试添加字符串和整数时，我们将需要报告错误。
4. 这部分中的符号表是一个简单的 Python 字典，它具有双重存储空间的功能。不用担心：符号表是一个非常重要的主题，我将专门针对它们撰写几篇文章。内存空间（运行时管理）本身就是一个话题。
5. 在我们之前文章中的简单计算器中，我们使用正斜杠字符 `"/"` 来表示整数除法。但是，在 Pascal 中，您必须使用关键字 `div` 来指定整数除法（参见练习 1）。
6. 我还特意引入了一个 hack，以便您可以在练习 2 中修复它：在 Pascal 中，所有保留关键字和标识符都不区分大小写，但本文中的解释器将它们视为区分大小写。

为了让你保持健康，这里有新的练习给你：



1. Pascal 变量和保留关键字不区分大小写，这与许多其他编程语言不同，因此BEGIN、begin和BeGin它们都引用相同的保留关键字。更新解释器，使变量和保留关键字不区分大小写。使用以下程序对其进行测试：

开始

开始

编号 := 2 ;

a := 数字;

B := 10 \* a + 10 \* NUMBER / 4 ;

c := a - - b

结束;

x := 11 ;

结束。

2. 我之前在“hacks”部分提到过，我们的解释器使用正斜杠字符“/”来表示整数除法，但它应该使用 Pascal 的保留关键字div进行整数除法。更新解释器以使用div关键字进行整数除法，从而消除其中一种技巧。
3. 更新解释器，以便变量也可以以下划线开头，如 '\_num := 5'。

这就是今天的全部内容。请继续关注，很快就会见到你。

以下是我推荐的书籍清单，它们将帮助您学习解释器和编译器：

1. 语言实现模式：创建您自己的特定领域和通用编程语言（实用程序员）  
([http://www.amazon.com/gp/product/193435645X/ref=as\\_li\\_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=193435645X&linkCode=as2&tag=russblo0b-20&linkId=MP4DCXDV6DJMEJBL](http://www.amazon.com/gp/product/193435645X/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=193435645X&linkCode=as2&tag=russblo0b-20&linkId=MP4DCXDV6DJMEJBL))
2. 编译器：原理、技术和工具（第 2 版）  
([http://www.amazon.com/gp/product/0321486811/ref=as\\_li\\_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0321486811&linkCode=as2&tag=russblo0b-20&linkId=GOEGDQG4HIHU56FQ](http://www.amazon.com/gp/product/0321486811/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0321486811&linkCode=as2&tag=russblo0b-20&linkId=GOEGDQG4HIHU56FQ))

如果您想在收件箱中获取我的最新文章，请在下方输入您的电子邮件地址，然后单击“获取更新”！

输入您的名字 \*

输入您最好的电子邮件 \*

获取更新!

## 本系列所有文章:

- 让我们构建一个简单的解释器。第1部分。 (/lsbasi-part1/)
- 让我们构建一个简单的解释器。第2部分。 (/lsbasi-part2/)
- 让我们构建一个简单的解释器。第 3 部分。 (/lsbasi-part3/)
- 让我们构建一个简单的解释器。第 4 部分。 (/lsbasi-part4/)
- 让我们构建一个简单的解释器。第 5 部分。 (/lsbasi-part5/)
- 让我们构建一个简单的解释器。第 6 部分。 (/lsbasi-part6/)
- 让我们构建一个简单的解释器。第 7 部分: 抽象语法树 (/lsbasi-part7/)
- 让我们构建一个简单的解释器。第 8 部分。 (/lsbasi-part8/)
- 让我们构建一个简单的解释器。第 9 部分。 (/lsbasi-part9/)
- 让我们构建一个简单的解释器。第 10 部分。 (/lsbasi-part10/)
- 让我们构建一个简单的解释器。第 11 部分。 (/lsbasi-part11/)
- 让我们构建一个简单的解释器。第 12 部分。 (/lsbasi-part12/)
- 让我们构建一个简单的解释器。第 13 部分: 语义分析 (/lsbasi-part13/)
- 让我们构建一个简单的解释器。第 14 部分: 嵌套作用域和源到源编译器 (/lsbasi-part14/)
- 让我们构建一个简单的解释器。第 15 部分。 (/lsbasi-part15/)
- 让我们构建一个简单的解释器。第 16 部分: 识别过程调用 (/lsbasi-part16/)
- 让我们构建一个简单的解释器。第 17 部分: 调用堆栈和激活记录 (/lsbasi-part17/)
- 让我们构建一个简单的解释器。第 18 部分: 执行过程调用 (/lsbasi-part18/)
- 让我们构建一个简单的解释器。第 19 部分: 嵌套过程调用 (/lsbasi-part19/)

## 注释

ALSO ON RUSLAN'S BLOG

<b>Let's Build A Simple Interpreter. Part 17: ...</b> 2 years ago • 8 comments You may have to fight a battle more than once to win it. - Margaret Thatcher	<b>Let's Build A Web Server. Part 1.</b> 7 years ago • 88 comments Out for a walk one day, a woman came across a construction site and saw ...	<b>Let's Build A Simple Interpreter. Part 5.</b> 6 years ago • 19 comments How do you tackle something as complex as understanding how to ...	<b>Let's Build A Simple Interpreter. Part 15.</b> 2 years ago • 14 comments "I am a slow walker, but I never walk back." — Abraham Lincoln And ...	<b>Let's Interp</b> 6 years Have y learnir these :
---	--	---	--	--

42 Comments Ruslan's Blog  Disqus' Privacy Policy Login ▾ Recommend 11  Tweet  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Aleksandr Zhuravlev • 5 years ago • edited

Thanks for the series, Ruslan. By following it I write my own implementation of the interpreter in Java. And I have a question, why do we need this 'if' statement in statement\_list?

```
if self.current_token.type == ID:  
    self.error()
```



It seems redundant to me, because we already have eat(END) on the above level.

2 ^ | v • Reply • Share ›



**thulani mtetwa** → Aleksandr Zhuravlev • 5 years ago

Do you mind sharing the code. I am finding it hard to follow since I do not really understand Python

2 ^ | v • Reply • Share ›



**Aleksandr Zhuravlev** → thulani mtetwa • 5 years ago

You are welcome!

<https://github.com/zelark/spi>

^ | v • Reply • Share ›



**thulani mtetwa** → Aleksandr Zhuravlev • 5 years ago

Thanks Alexandr. You just made a ton of things easier

^ | v • Reply • Share ›



**rspivak** Mod → Aleksandr Zhuravlev • 5 years ago

Thanks, Alexander. You're right, it is redundant in this case. I'll update the code to reflect that.

^ | v • Reply • Share ›



**Robert H** • 2 years ago

This series is really great. I'm following it and implementing it in C#. During my studies at school (a long time ago) I actually tried to program a Basic interpreter in Pascal. The Borland Pascal IDE always looked magical to me. I started with pushing and popping operators and operands on a stack, but eventually it became too complex for a complete language. So it's very nice to understand it after all these years. Thanks a lot.

Robert

1 ^ | v • Reply • Share ›



**Maddy** → Robert H • a year ago

yeah dude ! this guy is GOLD

^ | v • Reply • Share ›



**Diego Marcia** • 5 years ago

WHOA.

Took me one week, but I managed to reproduce all 9 parts... And now I can't help but beg you:

Part ten, part ten, part ten, part ten, please, please, please, please, please, please PLEEEEEEEEEAAAAAASEEEEEEE

1 ^ | v • Reply • Share ›



**rspivak** Mod → Diego Marcia • 5 years ago

Way to go, Diego. Thanks for reading and actually implementing the interpreter! You're super fast. :) Part 10 is already in the making, should be out in the next two weeks.

6 ^ | v • Reply • Share ›



**Andy** • 5 years ago

I love this series, thank you so much!

1 ^ | v • Reply • Share ›



**rspivak** Mod → Andy • 5 years ago

Thank you!

^ | v • Reply • Share ›



**Jonathan** • 5 years ago

YEEEEAAAAaaah, finally! <3 Thank you

1 ^ | v • Reply • Share ›



**rspivak** Mod → Jonathan • 5 years ago

You're welcome. :)

^ | v • Reply • Share ›



**Maddy** • a year ago



I thank you so much my man ! I here is such a huge lack of this kind of content over the internet :) I am from india and you have helped me a lot

^ | v • Reply • Share ›



**Anig** • 2 years ago

I have a question that why just we need a class for Compound but not for statement,program and ...?Thx !

^ | v • Reply • Share ›



**周景锦** • 3 years ago

Thanks you so much. You must be the best programmer among painters all over the world~ haha!

^ | v • Reply • Share ›



**Octree** • 3 years ago

Thank you, bro! I learn a lot from your posts

^ | v • Reply • Share ›



**lonelywaiting** • 3 years ago

thanks very much~

^ | v • Reply • Share ›



**Christian Alexander (TheNewCom)** • 4 years ago

In python 3.6, pylint is telling me that the Interpreter class has no member called GLOBAL\_SCOPE.How do you get this?

^ | v • Reply • Share ›



**evanxg852000** • 4 years ago • edited

I wish i had this wonderful resource when I started learning about compilers. Still this is really a nice and important article series. I really like the way this is structured. at the end of each part you have something working. this helps keep motivation. I believe this is the best part of this series. Thanks for this series. Any plan on OS dev or Database engines implementation ?

^ | v • Reply • Share ›



**Raju Choudhari** • 5 years ago

Thanks. Very useful.

^ | v • Reply • Share ›



**ER** • 5 years ago

I would love you as my teacher

^ | v • Reply • Share ›



**Humoyun Ahmedov** • 5 years ago • edited

Awesome, maybe more than awesome :)

^ | v • Reply • Share ›



**Hilde** • 5 years ago

Thank you so much for these series! You made it very clear and understandable for me. I'm enjoying programming as a hobby and also a small part of my scientific work, but I have no formal education in it. I managed to do your tasks, and found out the recursive implementation of the nested expression before you explained it. I was so proud for trying ;)

Well, I am going to interpret not a programming language, but something that might resemble XML/HTML in that it has tags with names and various values. Your tutorial did not get me there, but you gave me a solid foundation to work on. Thank you!

^ | v • Reply • Share ›



**Sniper435** • 5 years ago

Finding the series really useful and informative, keep going over parts to try and make sure I fully understand them.

could I ask how you're doing with the next part and also whether you'll touch on how to implement an interpreter in a language that uses strict data types as python seems to make this "easy" for you.

^ | v • Reply • Share ›



**Sniper435** → Sniper435 • 5 years ago

FYI, as I've been going through I've been implementing every step of the series in another language to solidify my

understanding, one part that really interests me how to implement functions - I hope this comes up soon.

^ | v • Reply • Share ›



**rspivak** Mod → Sniper435 • 5 years ago

Thanks for reading and yes, procedures and functions are coming soon.

^ | v • Reply • Share ›



**Matthew Tolman** • 5 years ago

Thanks for this. Looking forward to the rest of the series! Don't stop now!

^ | v • Reply • Share ›



**Daniel Kurashige-Gollub** • 5 years ago

Thanks for this blog post series about Interpreter/Compiler implementation. I am enjoying it a great deal and am following it along in the D programming language, implementing a very rudimentary BASIC dialect (so no Pascal for me; reason for this is, I want to implement something simpler than Pascal that I can use to teach my daughters programming when they are a bit older). Please continue your series, as it is pretty entertaining and very informative. Thank you.

^ | v • Reply • Share ›



**gabodev** • 5 years ago

It is helping to improve the implementation of the interpreter of my project :)

<https://github.com/centauri...>

Thank you so much!

^ | v • Reply • Share ›



**Venkatesh Pitta** • 5 years ago

Excellent work and writing. Thank you. I just finished one reading of all the 9 articles in two evenings. Starting to reread. Repetition :). One thing that stands out to me is, why use type for a field name? Because type(node) of Python is used as well. Perhaps a bit verbose, and a better descriptive name token\_type could be used. I will send you a PR on GitHub. Cheers!

^ | v • Reply • Share ›



**Mars Cheng** • 5 years ago

I'm studying interpreter recently and really like this series. Would you mind I translate your articles to Traditional Chinese? It would help others a lot. :-)

^ | v • Reply • Share ›



**rspivak** Mod → Mars Cheng • 5 years ago

Go ahead and good luck! :)

^ | v • Reply • Share ›



**Connor Stack** • 5 years ago

Just finished part 9! As an extra challenge I also made error messages include line numbers. Can't wait for part 10 :D

^ | v • Reply • Share ›



**Dimas Salazar** • 5 years ago

Excelent articles Ruslan. It helps me a lot to understand some things about interpreters and to learn some concepts for some things I do instinctively. I can't wait for part 10 and beyond.

How many parts are you planning for this serie.

I will implement this in other languages to really learn it all.

Thanks!!!

(Pascal was my first programming language)

^ | v • Reply • Share ›



**Ike S. Ma** • 5 years ago

Thanks soooo much! These tutorials are so awesome! They are way too addictive and I end up finishing the entire series (1-9) in two days! I've been implementing along the series and I can hardly feel my wrists now....

I audited the compiler course on coursera last summer for fun (but did not attempt any of the program assignments). I

ended up just knowing lots of terms and sort of lost in the theory... By following along the tutorials so far, I have better grasp and understanding the what and why of the theory. So many things I thought I knew before had been relearned!

It's also mind-blowing to see all the stuff I learnt at various places come together here! Visitor pattern from design pattern, tree traversal from data structure, object-oriented design from a C++ course, and context free grammar from automaton course etc. Just WOW! Lots of things I didn't quite understand before make so much sense now!

I love how the theories have been brought into practice with beautiful python code! I am fairly new to python, and I have picked quite a few python hacks and implementation tricks from these tutorials. Thanks so much for your time writing up these awesome tutorials!

How many more tutorials are we gonna have for the interpreter topic? Any spoilers what's gonna come up next? Are we gonna touch on implementations such as control flow, functions, pointers, class, garbage collection, and so forth? Anyways, I can't wait to see whatever comes up next !!!

^ | v • Reply • Share ›



**Aleksandr Zhuravlev** • 5 years ago

To be more honest, we need to add a little bit to our grammar, here it is:

```
program : compound_statement DOT EOF
```

I am looking forward to the next part.

^ | v • Reply • Share ›



**Alex Emelyanov** → Aleksandr Zhuravlev • 8 months ago • edited

I tried to add random chars after `end.` in example Pascal program, Free Pascal successfully compiled it, looks like it just ignore anything after the dot symbol.

^ | v • Reply • Share ›



**tuvi** • 5 years ago

it is an awesome article...Thank you so much..

^ | v • Reply • Share ›



**Michael Karotsieris** • 5 years ago

Awesome material ! Thank you! :D

^ | v • Reply • Share ›



**PubSubPenguin** • 5 years ago

I'm trying to follow this guide in rust. And up untill now I could always look to your github if I got stuck. But you've stoped updating the rust example code and I've gotten stuck with the symbol table.

Would you consider updating it again? I would appreciate it a lot if you would! Thank you for the helpful guide.

^ | v • Reply • Share ›



**rspivak** Mod → PubSubPenguin • 5 years ago

You're welcome and thank you for reading. I can't promise you the date, but I will add Rust code eventually.

^ | v • Reply • Share ›

✉ Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

🏠 社会的

🐙 github (<https://github.com/rspivak/>)

🐦 推特 (<https://twitter.com/rspivak>)

🌐 链接 (<https://linkedin.com/in/ruslanspivak/>)

## 🏠 热门帖子

让我们构建一个 Web 服务器。第1部分。 (<https://ruslanspivak.com/lbaws-part1/>)

让我们构建一个简单的解释器。第1部分。 (<https://ruslanspivak.com/lbasi-part1/>)

让我们构建一个 Web 服务器。第2部分。 (<https://ruslanspivak.com/lbaws-part2/>)

让我们构建一个 Web 服务器。第 3 部分。 (<https://ruslanspivak.com/lbaws-part3/>)

让我们构建一个简单的解释器。第2部分。 (<https://ruslanspivak.com/lbasi-part2/>)

## 免责声明

这个网站上的一些链接有我的亚马逊推荐 ID，它为我提供了每次销售的小额佣金。感谢您的支持。