苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 编译原理课程实践 | | | | | | | 成绩 |  |
| 指导教师 | | 王中卿 | | 同组实验者 | | 无 | | 实验日期 | 2023.12.11 | |

|  |  |
| --- | --- |
| 实 验 名 称 | 实验 |

1. 实验目的

本实验的主要目的是深入理解Python语言的解析原理，并实践编译原理的基础知识。通过手动实现一个简易的Python解释器，旨在加深对词法分析、语法分析、抽象语法树（AST）构建、以及语法制导翻译等编译过程的理解。

1. 实验内容

项目概览:

实现了一个基于PLY（Python Lex-Yacc）库的简易Python解释器。

包含词法分析器（py\_lex.py）、语法分析器（py\_yacc.py）、节点定义（node.py）和翻译执行模块（translation.py）。

词法分析 (py\_lex.py):

定义了Python语言的基本词法元素（tokens），如关键词、操作符和其他符号。

使用PLY的lex模块来识别和生成这些tokens。

语法分析 (py\_yacc.py):

建立了符合Python语法的解析规则。

使用PLY的yacc模块根据词法tokens构建AST。

AST节点定义 (node.py):

定义了用于构建AST的各种节点类型。

包括非终结符、终结符、变量、数字等节点类。

语法制导翻译 (translation.py):

实现了从AST到可执行代码的转换逻辑。

处理了变量声明、赋值、表达式计算等功能。

测试和示例:

使用main.py作为解释器的入口，测试了包括但不限于算术运算、条件语句和函数定义等特性。

实现了快速排序算法（quick\_sort.py）作为实用示例。

1. 实验步骤和结果

依赖项

要运行此项目，需要安装以下依赖项：

**ply~=3.11**

您可以通过运行以下命令来安装这些依赖项：

**pip install -r requirements.txt**

使用方法

安装完项目依赖后,在终端执行:

python3 main.txt {待分析的文件}

该项目中各个文件的作用:

python\_parser/: 主要的Python解释器项目目录。main.py: 解释器的主入口文件，负责启动解释过程。

node.py: 定义AST（抽象语法树）的节点，用于构建和处理解释器的语法树。

parser.out: PLY库生成的解析器调试文件，包含解析器的内部信息。

parsetab.py: PLY库生成的解析器表格文件，用于存储语法分析的状态。

py\_lex.py: 词法分析器的定义，用于将源代码分解为tokens。

py\_yacc.py: 语法分析器的定义，用于根据tokens构建AST。

quick\_sort.py: 快速排序算法的Python实现，用作测试。

translation.py: 负责将AST转换为可执行代码的翻译器。

项目结构图:

**基于PLY的Python解析(3)**

**├── docs**

**│   ├── 第12次课.docx**

**│   └── 第12次课实验报告.docx**

**├── python\_parser**

**│   ├── main.py # 解释器的主入口文件，负责启动解释过程**

**│   ├── node.py # 定义AST（抽象语法树）的节点，用于构建和处理解释器的语法树。**

**│   ├── parser.out # PLY库生成的解析器调试文件，包含解析器的内部信息。**

**│   ├── parsetab.py # PLY库生成的解析器表格文件，用于存储语法分析的状态。**

**│   ├── py\_lex.py # 词法分析器的定义，用于将源代码分解为tokens。**

**│   ├── py\_yacc.py # 语法分析器的定义，用于根据tokens构建AST。**

**│   ├── quick\_sort.py # 快速排序算法的Python实现，用作测试。**

**│   └── translation.py # 负责将AST转换为可执行代码的翻译器。**

**├── readme.md**

**└── requirements.txt # 包含项目所需Python库的列表，用于设置项目环境。**

该项目的流程图:

图形用户界面, 应用程序

描述已自动生成

项目定义的语法规则:

**Grammar**

**Rule 0 S' -> program**

**Rule 1 program -> statements**

**Rule 2 statements -> statements statement**

**Rule 3 statements -> statement**

**Rule 4 statement -> assignment**

**Rule 5 statement -> expr**

**Rule 6 statement -> print**

**Rule 7 statement -> if**

**Rule 8 statement -> while**

**Rule 9 statement -> for**

**Rule 10 statement -> break**

**Rule 11 statement -> function**

**Rule 12 statement -> return**

**Rule 13 assignment -> variable ASSIGN expr**

**Rule 14 assignment -> variable MINEQUAL expr**

**Rule 15 assignment -> variable PLUSEQUAL expr**

**Rule 16 assignment -> variable DPLUS**

**Rule 17 assignment -> variable DMINUS**

**Rule 18 variable -> variable LBRACKET expr RBRACKET**

**Rule 19 variable -> ID**

**Rule 20 expr -> expr PLUS term**

**Rule 21 expr -> expr MINUS term**

**Rule 22 expr -> term**

**Rule 23 expr -> array**

**Rule 24 term -> term TIMES factor**

**Rule 25 term -> term DIVIDE factor**

**Rule 26 term -> term EDIVIDE factor**

**Rule 27 term -> factor**

**Rule 28 factor -> variable**

**Rule 29 factor -> NUMBER**

**Rule 30 factor -> len**

**Rule 31 factor -> call**

**Rule 32 factor -> LPAREN expr RPAREN**

**Rule 33 exprs -> exprs COMMA expr**

**Rule 34 exprs -> expr**

**Rule 35 len -> LEN LPAREN variable RPAREN**

**Rule 36 print -> PRINT LPAREN exprs RPAREN**

**Rule 37 print -> PRINT LPAREN RPAREN**

**Rule 38 array -> LBRACKET exprs RBRACKET**

**Rule 39 array -> LBRACKET RBRACKET**

**Rule 40 condition -> condition OR join**

**Rule 41 condition -> join**

**Rule 42 join -> join AND equality**

**Rule 43 join -> equality**

**Rule 44 equality -> equality EQ rel**

**Rule 45 equality -> equality NE rel**

**Rule 46 equality -> rel**

**Rule 47 rel -> expr LT expr**

**Rule 48 rel -> expr LE expr**

**Rule 49 rel -> expr GT expr**

**Rule 50 rel -> expr GE expr**

**Rule 51 rel -> expr**

**Rule 52 if -> IF LPAREN condition RPAREN LBRACE statements RBRACE**

**Rule 53 if -> IF LPAREN condition RPAREN LBRACE statements RBRACE else**

**Rule 54 else -> ELIF LPAREN condition RPAREN LBRACE statements RBRACE**

**Rule 55 else -> ELIF LPAREN condition RPAREN LBRACE statements RBRACE else**

**Rule 56 else -> ELSE LBRACE statements RBRACE**

**Rule 57 while -> WHILE LPAREN condition RPAREN LBRACE statements RBRACE**

**Rule 58 for -> FOR LPAREN assignment SEMICOLON condition SEMICOLON assignment RPAREN LBRACE statements RBRACE**

**Rule 59 break -> BREAK**

**Rule 60 function -> DEF ID LPAREN args RPAREN LBRACE statements RBRACE**

**Rule 61 function -> DEF ID LPAREN RPAREN LBRACE statements RBRACE**

**Rule 62 args -> args COMMA ID**

**Rule 63 args -> ID**

**Rule 64 call -> ID LPAREN exprs RPAREN**

**Rule 65 call -> ID LPAREN RPAREN**

**Rule 66 return -> RETURN**

**Rule 67 return -> RETURN exprs**

node.py模块内容:

该模块定义了用于构建抽象语法树（AST）的各种节点类型。

主要包含的类有：

- \_node：所有节点的基类，提供基本的节点数据结构。

- NonTerminal：非终结符节点，用于表示具有特定类型和可选值的非终结符。

- Variable：左值节点，表示引用的变量，包含类型和标识符。

- Number：数字节点，直接包含一个数字值。

- ID：标识符节点，包含标识符的名称和值。

- Terminal：终结符节点，用于表示除标识符外的其他终结符，包含其文本内容。

这些节点类型在解析Python代码并构建其AST时发挥核心作用。

translation.py模块的作用:

该模块负责将解析得到的抽象语法树（AST）转换为可执行的代码。

主要功能包括：

- 执行Python代码的翻译和运行。

- 处理函数定义和调用。

- 管理运行时环境和变量。

此模块是简易Python解释器的核心部分，实现了基本的语言特性和执行逻辑。

其中用来表示函数的类:

表示Python函数的类。

这个类用于表示一个Python函数，包括它的名称、参数列表和函数体。

它提供了执行函数体的方法，并能够处理参数传递和局部作用域。

Attributes:

name (str): 函数的名称。

arg\_names (list): 参数名列表。

body (\_node): 函数体，表示为一个AST节点。

代码中定义的关键词:

reserved\_words = {

'print': 'PRINT',

'if': 'IF',

'elif': 'ELIF',

'else': 'ELSE',

'for': 'FOR',

'while': 'WHILE',

'len': 'LEN',

'break': 'BREAK',

'and': 'AND',

'or': 'OR',

'def': 'DEF',

'return': 'RETURN',

}

tokens = ['NUMBER', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN', 'LBRACKET', 'RBRACKET', 'ASSIGN',

'LBRACE', 'RBRACE', 'SEMICOLON', 'COMMA', 'DPLUS', 'DMINUS', 'ID', 'EDIVIDE', 'MINEQUAL', 'PLUSEQUAL',

'LT', 'LE', 'GT', 'GE', 'EQ', 'NE', ] + list(reserved\_words.values())

# Define of tokens

t\_PLUSEQUAL = r'\+='

t\_MINEQUAL = r'-='

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_EDIVIDE = r'//'

t\_DIVIDE = r'/'

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

t\_LBRACE = r'\{'

t\_RBRACE = r'\}'

t\_LBRACKET = r'\['

t\_RBRACKET = r'\]'

t\_ASSIGN = r'='

t\_DPLUS = r'\+\+'

t\_DMINUS = r'--'

t\_COMMA = r','

t\_SEMICOLON = r';'

t\_LT = r'<'

t\_LE = r'<='

t\_GT = r'>'

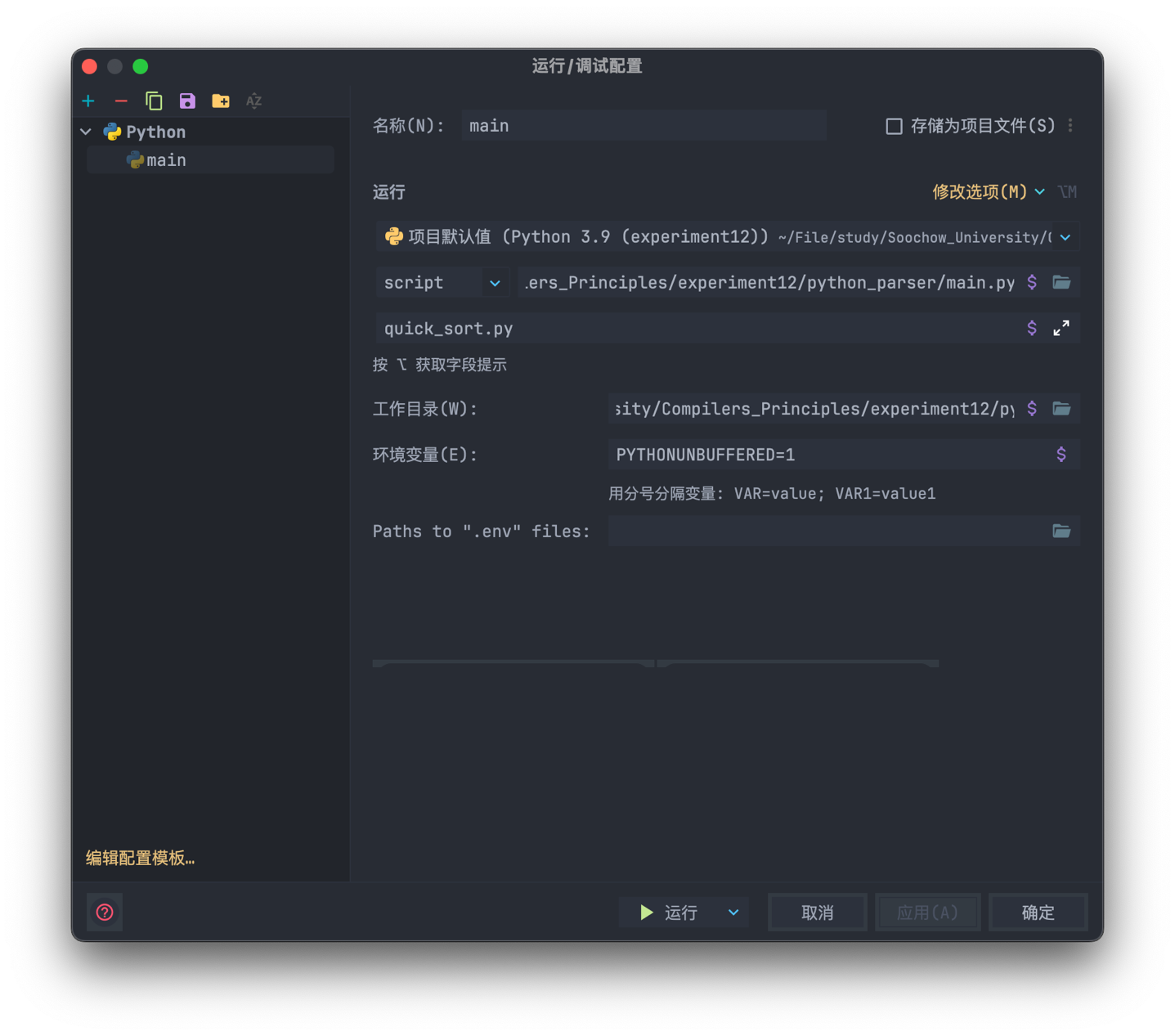
t\_GE = r'>='

t\_EQ = r'=='

t\_NE = r'!='

代码运行结果:

当代码的运行配置如下时:



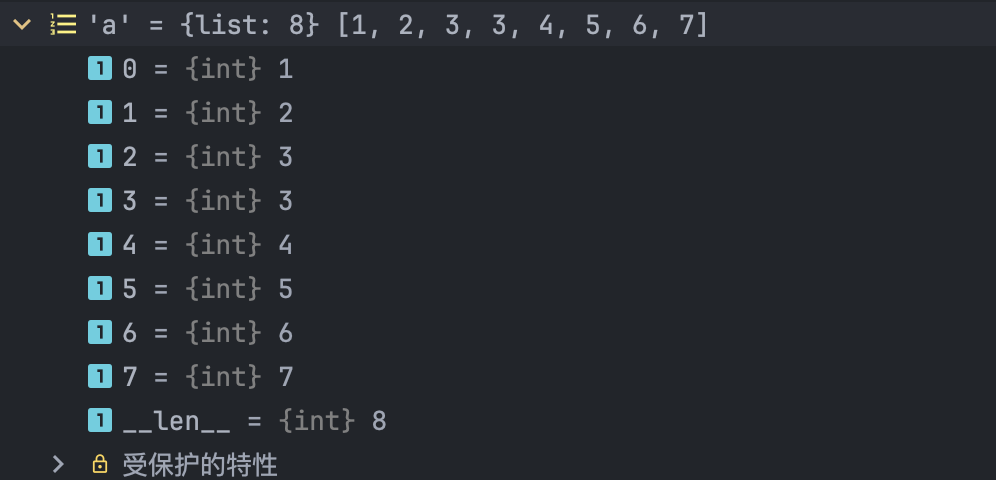
代码运行后变量表中的内容:

文本

描述已自动生成

可以看到,此时函数以及数组已经被添加到了变量表中了

代码运行后数组中的各个数的内容:



可以看到数组中的各个数已经被成功修改

1. 实验总结

通过本实验，我深入了解了Python解释器的工作原理和编译原理的基本概念。在实现解释器的过程中，我学习了如何使用PLY库进行词法和语法分析，并理解了AST在语言解析中的核心作用。此外，实验过程中对代码结构的规划和模块化设计也提升了我的软件工程技能。通过编写和测试代码，我加强了自己的调试能力和代码优化能力。总的来说，这个实验不仅加深了我对编译原理的理解，也锻炼了我的编程实践能力。