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 | 2023.11.27 | |

|  |  |
| --- | --- |
| 实 验 名 称 | 基于PLY的Python解析(2) |

1. 实验目的

掌握基于 PLY 的解析技术: 通过使用 Python Lex-Yacc(PLY)，深入理解编译原理中的语法分析过程。 理解并实现简单的解析器: 学习如何构建一个能够处理基本 Python 语句(赋值、四则运算、print 语句) 的解析器。

语法树的构建与理解: 学习如何从解析过程中构建语法树，并理解其结构与用途。 实现语法制导翻译: 理解并实践如何通过语法树进行语法制导翻译，包括变量值的存储和运算结果的计算。

1. 实验内容

设计语言规范：定义解释器支持的简易编程语言的语法规则和特性。

实现词法分析器（py\_lex.py）：使用 PLY 的 Lex 工具或类似工具构建词法分析器，将源代码文本分解为标记（tokens）。

实现语法分析器（py\_yacc.py）：使用 PLY 的 Yacc 工具或类似工具构建语法分析器，根据词法分析器的输出构建抽象语法树（AST）。

节点定义（node.py）：定义不同类型的节点以构建 AST，包括非终结符、左值、数字、标识符和终结符等。

实现语法翻译（translation.py）：实现翻译函数，将 AST 节点转换为可执行代码。

主程序（main.py）：实现解释器的主入口，负责读取源代码文件、调用词法分析器和语法分析器，执行解析出的代码。

测试脚本（select\_sort.py）：编写特定语法的脚本，作为解释器的测试用例。

1. 实验步骤和结果

项目结构图:

**experiment11/**

**|**

**|-- python\_parser/**

**| |-- main.py # 主程序入口**

**| |-- node.py # 定义抽象语法树（AST）的节点**

**| |-- translation.py # 语法树翻译和执行**

**| |-- py\_yacc.py # 语法分析器**

**| |-- py\_lex.py # 词法分析器**

**| |-- select\_sort.py # 示例程序**

**| |-- binary\_search.py #示例程序**

**|**

**|-- readme.md # 项目的readme文档**

**|-- requirements.txt # 项目需求**

项目运行方式:

终端输入:

**python3 main.py {需要分析的py程序}**

当需要分析的py程序未给出时,程序会报错,显示” 不正确的用法”

项目的运行结果:

当项目运行配置如下时:

电脑萤幕的截图

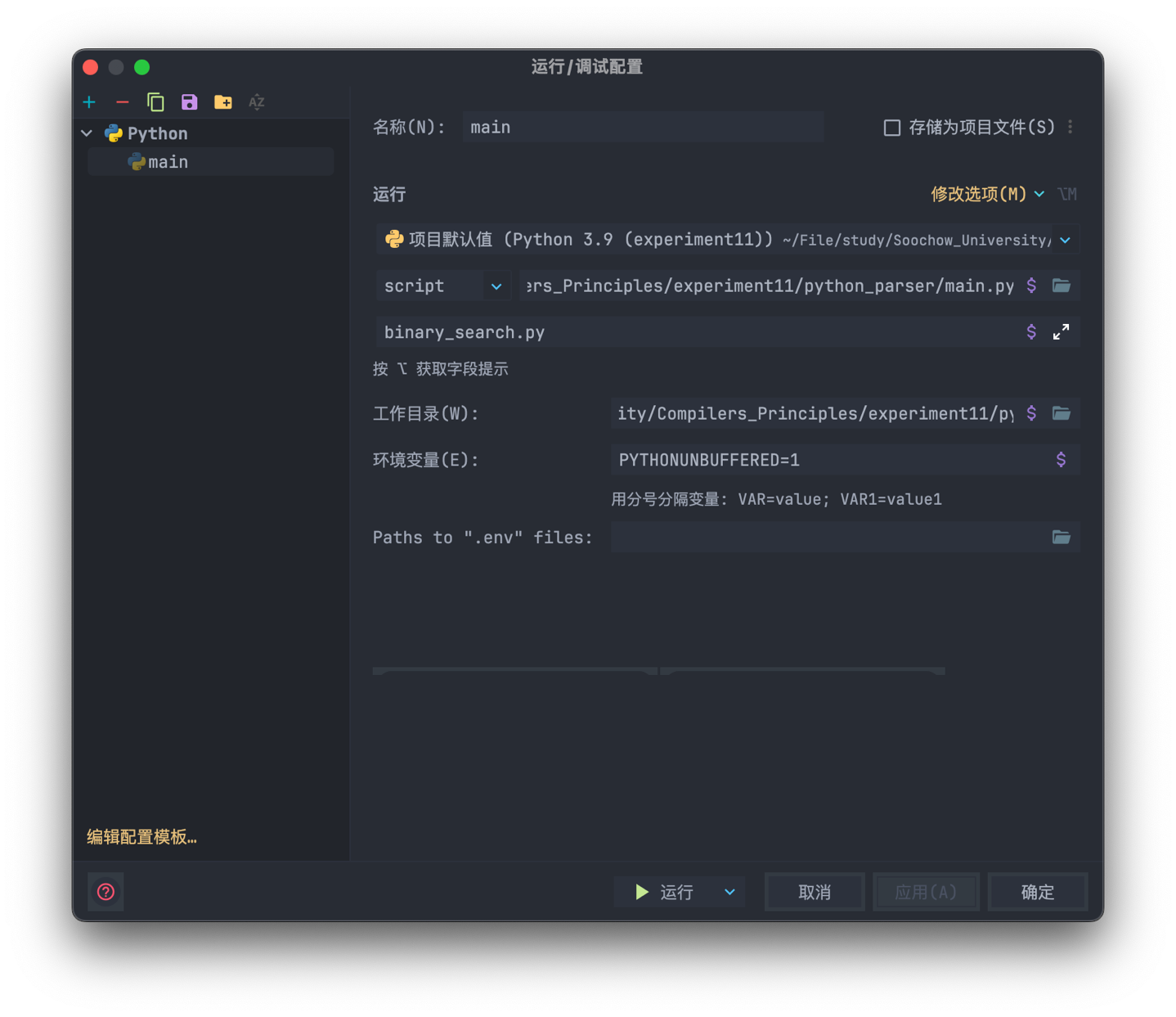
描述已自动生成

此时运行结果:

文本

描述已自动生成

项目运行配置如下:



运行结果:

文本

描述已自动生成

定义的语法规则:

**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 assignment -> leftval ASSIGN expr**

**Rule 12 assignment -> leftval ASSIGN array**

**Rule 13 leftval -> leftval LBRACKET expr RBRACKET**

**Rule 14 leftval -> ID**

**Rule 15 expr -> expr PLUS term**

**Rule 16 expr -> expr MINUS term**

**Rule 17 expr -> term**

**Rule 18 term -> term TIMES factor**

**Rule 19 term -> term DIVIDE factor**

**Rule 20 term -> term EDIVIDE factor**

**Rule 21 term -> factor**

**Rule 22 factor -> leftval**

**Rule 23 factor -> NUMBER**

**Rule 24 factor -> len**

**Rule 25 factor -> LPAREN expr RPAREN**

**Rule 26 exprs -> exprs COMMA expr**

**Rule 27 exprs -> expr**

**Rule 28 len -> LEN LPAREN leftval RPAREN**

**Rule 29 print -> PRINT LPAREN exprs RPAREN**

**Rule 30 print -> PRINT LPAREN RPAREN**

**Rule 31 array -> LBRACKET exprs RBRACKET**

**Rule 32 array -> LBRACKET RBRACKET**

**Rule 33 selfvar -> leftval DPLUS**

**Rule 34 selfvar -> leftval DMINUS**

**Rule 35 condition -> expr LT expr**

**Rule 36 condition -> expr LE expr**

**Rule 37 condition -> expr GT expr**

**Rule 38 condition -> expr GE expr**

**Rule 39 condition -> expr EQ expr**

**Rule 40 condition -> expr NE expr**

**Rule 41 condition -> expr**

**Rule 42 if -> IF LPAREN condition RPAREN LBRACE statements RBRACE**

**Rule 43 if -> IF LPAREN condition RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE**

**Rule 44 if -> IF LPAREN condition RPAREN LBRACE statements RBRACE ELIF LPAREN condition RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE**

**Rule 45 while -> WHILE LPAREN condition RPAREN LBRACE statements RBRACE**

**Rule 46 for -> FOR LPAREN assignment SEMICOLON condition SEMICOLON selfvar RPAREN LBRACE statements RBRACE**

**Rule 47 break -> BREAK**

其中,node.py的内容为:

1. \_node 类

作用：所有节点的基类，为其他特定节点类型提供基础结构。

属性：

\_data：存储节点的数据。

\_children：子节点列表。

\_value：节点的值，初始化为 NIL。

方法：

\_\_init\_\_：构造函数，初始化数据、子节点和值。

value：属性装饰器，用于获取和设置节点值。

child：获取指定索引的子节点。

children：返回所有子节点。

add：添加一个子节点。

2. NonTerminal 类

作用：表示非终结符的节点，例如表达式或语句。

属性：

type：非终结符的类型。

方法：

\_\_str\_\_：返回节点的字符串表示，包括类型和子节点。

3. LeftValue 类

作用：表示左值，即可以被赋值的实体，例如变量。

属性：

id：引用的变量名。

方法：

\_\_init\_\_：初始化左值节点。

value：不允许直接访问 LeftValue 的 value 属性，而是通过符号表实现。

\_\_str\_\_：返回节点的字符串表示。

4. Number 类

作用：表示数字。

属性：

\_value：数字的值。

方法：

\_\_init\_\_：构造函数，将传入的数据转换为整数。

\_\_str\_\_：返回节点的字符串表示，显示数字值。

5. ID 类

作用：表示标识符，例如变量名。

属性：

id：标识符名称。

方法：

\_\_init\_\_：初始化标识符节点。

\_\_str\_\_：返回节点的字符串表示。

6. Terminal 类

作用：表示除标识符以外的终结符节点，如运算符、括号等。

属性：

text：终结符的文本内容。

方法：

\_\_str\_\_：返回节点的字符串表示，特定符号进行字符替换。

translate.py内容:

全局变量

\_\_DEBUG\_MODE: 一个布尔变量，用于控制是否打印调试信息。当设置为 True 时，程序会在执行过程中打印额外的调试信息，有助于理解程序的执行流程和状态。

loop\_flag: 用于跟踪当前的循环层级。它在处理嵌套循环时特别有用，以确定 break 语句应该跳出哪个循环层级。

break\_flag: 一个标志，用于指示是否遇到 break 语句。当在循环中遇到 break 时，这个标志会被设置为 True，导致循环提前终止。

函数 get\_value

作用: 从变量表（var\_table）中获取指定变量或数组元素的值。

参数: tb（变量表），vid（变量或数组元素的标识符）。

功能: 能够处理简单变量和嵌套变量（如数组元素）的值获取。

函数 set\_value

作用: 在变量表（var\_table）中设置指定变量或数组元素的值。

参数: tb（变量表），vid（变量或数组元素的标识符），val（要设置的值）。

功能: 类似于 get\_value，但用于设置而非获取值。

函数 translate

作用: 遍历和翻译 AST，执行程序。

参数: tree（AST的节点）。

功能: 根据节点类型（如 If, While, For, Break, Assignment 等）执行相应的操作。

If 语句处理

解析 if 及其变体（else, elif）语句。

根据条件表达式的值决定执行哪个代码块。

递归调用 translate 来处理嵌套的语句。

While 语句处理

解析 while 循环。

根据条件表达式的值决定是否继续循环。

通过 break\_flag 控制循环的退出。

For 语句处理

解析 for 循环，包括初始化、条件判断和迭代表达式。

实现循环的逻辑，包括对循环变量的更新。

Break 语句处理

当遇到 break 时，设置 break\_flag，导致最近的外层循环结束。

赋值、表达式和其他语句

对赋值语句执行变量值的更新。

计算表达式的值，如算术运算和逻辑运算。

递归处理复合表达式和嵌套语句。

程序运行过程

初始化: 当 main.py 调用 translate 函数时，它传入了解析好的 AST 的根节点。

递归遍历: translate 函数递归地遍历每个节点，根据节点类型执行不同的操作。

条件判断: 对于 if、while、for 等节点，根据条件表达式的值决定执行路径。

赋值和表达式计算: 对于赋值节点，更新变量表中相应的值；对于表达式节点，计算并返回表达式的值。

循环控制: 利用 loop\_flag 和 break\_flag 控制循环的执行和退出。

1. 实验总结

编译原理理解: 通过实践深入理解了编译原理的基本概念，如词法分析、语法分析、抽象语法树（AST）的构建和遍历，以及如何将这些理论应用到实际的编程语言解释器中。

编程技能提升: 加强了 Python 编程能力，特别是在处理复杂数据结构和算法方面。同时，对 Python 中类的继承、多态和封装等面向对象的概念有了更深入的理解。

工具应用: 学会了使用诸如 PLY（Python Lex-Yacc）之类的工具，这些工具在编写词法分析器和语法分析器时极为有用。