《编译原理》课设

文 档

学号：\_\_\_\_\_\_16041049\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_陈天宇\_\_\_\_\_\_\_\_\_\_\_

2018年 12 月 9 日

## 需求说明

### 文法说明

关键字: const, int, void, if, else, while, main, return, printf, scanf

＜加法运算符＞ ::= +｜－

＜乘法运算符＞ ::= \* ｜／

＜关系运算符＞ ::= ＜｜＜＝｜＞｜＞＝｜!＝｜＝＝

＜字符＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= 0｜＜非零数字＞

＜非零数字＞ ::= 1｜．．．｜9

＜专用符号＞ ::= (｜)｜{｜}｜，｜；｜＝

<字符串> ::= "｛<合法字符> }"

＜程序＞ ::= 〔＜常量说明部分＞〕〔＜变量说明部分＞〕｛＜函数定义部分＞｝＜主函数＞

＜常量说明部分＞ ::= const ＜常量定义＞｛,＜常量定义＞};

＜常量定义＞ ::= ＜标识符＞＝＜整数＞

＜整数＞ ::= 〔＋｜－〕＜非零数字＞｛＜数字＞｝｜０

＜标识符＞ ::= ＜字符＞｛＜字符＞｜＜数字＞｝

＜声明头部＞ ::= int　＜标识符＞

＜变量说明部分＞ ::= ＜声明头部＞｛，＜标识符＞｝；

＜函数定义部分＞ ::= （＜声明头部＞｜void ＜标识符＞）＜参数＞＜复合语句＞

＜复合语句＞ ::= ‘{’〔＜常量说明部分＞〕〔＜变量说明部分＞〕＜语句序列＞‘}’

＜参数＞ ::= ‘(’＜参数表＞‘)’

＜参数表＞ ::= int ＜标识符＞｛，int ＜标识符＞} | 空

//参数表可以为空

＜主函数＞ ::= ( void ｜int ) main ＜参数＞＜复合语句＞

＜表达式＞ ::= 〔＋｜－〕＜项＞｛＜加法运算符＞＜项＞｝

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

＜因子＞ ::= ＜标识符＞｜‘（’＜表达式＞‘）’｜＜整数＞｜＜函数调用语句＞

＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜‘{’<语句序列>‘}’｜＜函数调用语句＞;

｜＜赋值语句＞; | <返回语句>;｜＜读语句＞;｜＜写语句＞;｜＜空＞

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞

＜条件语句＞ ::= if‘（’＜条件＞‘）’＜语句＞〔else＜语句＞〕

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

＜循环语句＞ ::= while‘（’＜条件＞‘）’＜语句＞

＜函数调用语句＞ ::= ＜标识符＞‘（’＜值参数表＞‘）’

＜值参数表＞ ::= ＜表达式＞｛，＜表达式＞｝｜＜空＞

＜语句序列＞ ::= ＜语句＞｛＜语句＞｝

＜读语句＞ ::= scanf‘(’＜标识符＞‘）’

＜写语句＞ ::= printf‘(’[<字符串>,][＜expression ＞]‘）’

　　　　　 //当出现字符串时，就加印字符串, 之后打印表达式的值；

＜返回语句＞ ::= return [ ‘(’＜表达式＞’)’ ]

### 目标代码说明

* **P-code**

一种栈式机的语言，这类栈式机没有累加器和通用寄存器，有四个控制寄存器（指令寄存器I， 指令寄存器P，栈式寄存器T和基址寄存器B），运算都在栈顶进行。

* **指令格式**

|  |  |  |
| --- | --- | --- |
| **f** | **x** | **y** |

* **f： 操作码**
* **x： x域，第一个操作数**
* **y： y域，第二个操作数**
* **设计使用指令：**

**表格 1 指令表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **助记符** | **f** | **x** | **y** | **功能** |
| LDA | 0 | x | y | 把变量地址装入栈顶（x为层次，y为相对地址） |
| LOD | 1 | x | y | 装入值 |
| DIS | 3 | x | y | 更新display |
| JMP | 10 |  | y | 无条件转移到y |
| JPC | 11 |  | y | 如栈顶内容为假，转移到y |
| MKS | 18 |  | y | 标记栈，调用函数前使用，保存现场 |
| CAL | 19 |  | y | 调用用户过程或函数 |
| LDC | 24 |  | y | 装入字面常量 |
| RED | 27 |  | y | 读入整型数 |
| WRS | 28 |  | y | 写字符串 |
| WRW | 29 |  | y | 写栈顶整型数 |
| EXP | 32 |  |  | 退出过程 |
| EXF | 33 |  |  | 退出函数 |
| STO | 38 |  |  | 将栈顶内容存入以栈顶次高元为地址的单元 |
| EQL | 45 |  |  | 整型相等比较 |
| NEQ | 46 |  |  | 整型不等比较 |
| LSS | 47 |  |  | 整型小于比较 |
| LER | 48 |  |  | 整型小于等于比较 |
| GRT | 49 |  |  | 整型大于比较 |
| GEQ | 50 |  |  | 整型大于等于比较 |
| ADD | 52 |  |  | 整型加 |
| SUB | 53 |  |  | 整型减 |
| MUL | 57 |  |  | 整型乘 |
| DIV | 48 |  |  | 整型除 |

## 详细设计

### 程序结构

C0 Online编译程序采用以递归下降子程序法为语法分析核心，进行一遍扫描的编译。词法分析和语法分析程序相互独立，语法分析程序将词法分析程序的结果作为输入。在语法分析的同时，提供了出错报告的功能。在源程序没有错误，编译通过的情况下，将生成对应的P-code代码。

程序采用了Django作为模板引擎来渲染前端，用户在网页端提交代码或者提交代码文件之后，会向服务器发起POST请求，POST调用C0编译器程序对代码进行处理。处理后的结果将生成H5表格代码，通过Django引擎返回request，浏览器渲染页面，将表格呈现给用户。

首先，词法分析special\_lexer继承自词法分析作业中的C0lexer，因而初始化的时候先调用类C0lexer生成父类对象，如果是本地对象，需要指明测试代码文件的路径（默认为网络服务器调用服务），接着完成一系列私有属性的初始化。

1. **def** \_\_init\_\_(self, INPUT\_FILE\_NAME: str):
2. '''''
3. '''
4. self.TOKEN = ""
5. self.INPUT\_FILE\_NAME = INPUT\_FILE\_NAME
6. self.SOURCE\_TEXT = ""
7. self.p = 0
8. self.RESULT = []
9. self.REPLY = ""
10. self.KEYWORDS = ["const", "int", "void", "if", "else",
11. "while", "main", "return", "printf", "scanf"]

然后，我们开始对输入代码进行词法分析，如果是本地的词法分析的话，可以调用word\_analyze接口，如果是服务器端进程，则调用web\_word\_analyze接口，这两个接口都是自动执行getsym()函数直到出错或者读到文件结尾。

1. **def** word\_analyze(self):
2. '''''
3. 自动词法分析至出错或到文件末尾
4. '''
5. self.read()
6. length = len(self.SOURCE\_TEXT)
7. **while** self.p < length **and** self.getsym():
8. **continue**
9. **if** self.p >= length:
10. **print**("WordAnalysis Completed!")
11. **else**:
12. **print**("Unknown invalid syntax!")
14. **def** web\_word\_analyze(self):
15. '''''
16. 用于服务器的词法分析接口
17. '''
18. length = len(self.SOURCE\_TEXT)
19. self.RESULT = []
20. **while** self.p < length **and** self.getsym():
21. **continue**
22. **if** self.p >= length:
23. self.REPLY += "WordAnalysis Completed!" + "\n"
24. **else**:
25. self.REPLY += "Unknown invalid syntax!" + "\n"

getsym（）函数每次会对接下来的字符串进行分析处理，如果发现有符合条件的单词，就会把它和分析结果一起作为一条记录增加到RESULT这个list对象中。

当getsym读到保留字时，就会查找保留字字典，看该保留字是否有定义，如果有定义就判定为保留字，否则就把这个字符串识别为一个标识符。由于需求文法中的数字只有整型数字，因而程序也只识别整型数。对于字符串，则看字符串前后是否存在约定的专用符号，如果满足，则判断为字符串。其他类型的运算符合专用符号同理。

在分析完成后，lexer对象输出RESULT，可以调用output（）接口输出到指定文件，也可以直接将结果输入给下一个对象norm\_C0\_compiler()，compiler执行s\_program()过程，对程序进行递归下降分析，如果分析成功，就可以调用一系列表格生成接口输出HTML5代码，或者在shell里直接打印表格。如果分析失败，也可以输出详细的编译时错误信息（未定义，重复定义， 语法错误等）。