**编译实践文档（申优）**

**13211103 杨帆**

1. 基本环境

本项目采用Python3.4编写，在windows10平台下编译运行通过

项目复现的时候请务必使用python3.0及以上版本，否则将无法正确运行

提交的源文件可能仍有部分bug没有修复，如果在测试中（主要是指报错测试）发现程序进入死循环，请减少源代码的错误数目再进行测试。

1. 理论部分
2. **概述**

这个编译器遵循的语法规范为C0语法。

本部分基本阐述了进行编译程序编写时的程序设计框架和基本原则。

框架即软件各部分的功能设计；基本原则即理论分析，如词法分析和语法分析的基本规则等。

1. **程序设计框架**

基本思想：将语法分析中的最底层看作，作为词法分析的结果。语法分析每次读入的单词都已经是一个最底层，如<标识符>、<字符串>、<整数>都是词法分析的结果，同时也是语法分析的输入。程序按照模块化的思想，分为词法分析块、语法分析块。

主程序功能按照编译的各个步骤分割，主要分为：

主程序：过程编程实现，控制编译程序运行流程的部分，可以控制直接编译或者调用调试函数进行调试

词法分析程序：面向对象编程实现，控制词法分析部分，编写时参考书上例程，分析结果为单词流链表的头指针和尾指针，报错原则为遇错忽略并记录。

语法分析程序：面向对象编程实现，控制语法分析部分，使用递归下降分析法，分析过程结束后程序结束，报错原则为遇错忽略并记录。

1. **C0文法整理规则**

对源文法进行少许改动，使更适应词法分析和语法分析的各自职责。

词法分析的目的为读取单词，包括<标识符>等，语法分析的目的为验证文法正确性，故不考虑读取单个字符，而是直接读取词法分析的结果链表。

1. **C0文法整理结果**

对原文法进行修改，使词法分析语法分析职责划分更为清晰。修改后文法如下

1. 词法分析部分

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-Za-z\_]

<数字>::= [0-9]

<专用符号>::= (｜)｜{｜}｜,｜;｜＝

<字符串> ::= "｛<合法字符> }"

<整数>::= <加法运算符>｛<数字>｝

<标识符>::= <字符>｛<字符>｜<数字>｝

<注释> ::= “/\*”{<合法字符>}”\*/” | “//”{<合法字符>}”\n”

1. 语法分析部分

<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>

<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>::= (<加法运算符>)<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>')' ]

1. **错误处理原则**

错误处理是编译器比较难以实现的地方。我采用的基本原则是：遇错报错并跳过。

关于遇错跳过的这部分，在语法分析到达中间的时候，遇到不同的错误应该跳过不同的单词，但是由于语法本身可以有多重嵌套，在同一个子程序中可能需要跳过的单词不尽相同，所以如何跳过单词成为一个严峻的问题。几经尝试，我最终确定的做法是：在子程序A调用子程序B进行语法分析的时候，需要传递给B一个overlookSet列表变量，这样B程序出错的时候可以根据overlookSet变量跳过单词直至找到一个A程序制定的单词，以便返回到A程序时A程序仍然能够正常处理剩下的源代码。

1. **符号表设计**

符号表的设计相对比较简单。一共有三个类：SymbolTableIndex，SymbolTableItem和SymbolTable类用来管理符号表中的符号表层次索引，符号表项目和符号表本身。

语法分析部分在运行时将会用全程序唯一的一个符号表实例来进行符号表管理

1. **错误类设计**

程序中有一个基本的Error类，用来管理分析中可能遇到的所有错误。在分析过程中，无论是词法分析还是语法分析还是语义分析，如果出现一个Error，那么程序将会生成一个Error并存储之，进行后面的分析，最后在main函数中将会统一将Error打印输出

1. **常量定义**

程序中有唯一的一个const实例供所有过程引用，用来标识程序中可能出现的各种常量。

1. 程序目前功能说明
2. **词法分析**

目前测试中除一个BUG无法修复外，其余词法分析皆并未发现错误。

唯一BUG为：当输入代码的语句中的分号（;）前没有空格时，有可能导致词法分析程序忽略分号，并导致提示缺失分号。目前此BUG的解决办法仍在寻找中。

如果出现词法分析错误，语法分析部分仍会进行，但是代码生成程序不会运行。

如果需要查看词法分析结果，可在main文件中使用WA.printAllVt()语句。

1. **语法分析**

目前的语法分析部分已经能够完全分析C0语法的所有内容，测试并未发现错误。

如果出现语法分析错误，会跳过错误部分并继续分析，但是代码生成程序不会运行。

如果需要查看词法分析结果，可在main文件中使用GA.printAllVn()语句。

1. **符号表处理**

在语法分析的同时，会同步进行符号表的相关操作。目前这部分没有发现错误。

如果需要查看生成的符号表，可在main文件中使用GA.symbolTable.printTable()语句。

1. **P-Code指令**

P-Code指令生成部分目前并没有完全完成，由于时间关系，函数调用，函数定义部分的P-Code指令并不能生成，但是其余部分的P-Code指令都能正确的生成，缺少的部分功能作者将会在申优答辩之前完成。

P-Code会在语法分析的同时生成，最终按照内部结构保存在main文件中的唯一实例instructionStream中。

如果需要打印最终的指令集，可在main文件中使用instructionStream.printAllInstructions()语句

1. 程序使用方法

在命令行中进入项目目录，输入指令：python main.py [sourceFileName] [outFileName]

选项sourceFileName必选，表示源文件的路径

选项outFileName必选，表示输出文件的路径

输出内容为P-Code指令集。

如果想要查看调试内容，请修改main.py添加相关调试调用语句

1. 测试（所有测试文件和输出都可在Sources文件夹下找到）
2. **If语句测试——test1**

源文件：

**void** main()  
{  
 **const** a = 100, b = 200 ;  
 **int** result ;  
 **if**(a > b)  
 result = a \* b ;  
 **else** result = a + b ;  
 **return** ;  
}

输出文件：

No. Lab. Instruction Operator.  
0 LOD a   
1 LOD b   
2 GRT   
3 BRF lab\_1   
4 LDA (result)   
5 LOD a   
6 LOD b   
7 MUL   
8 STN   
9 BR lab\_2   
10 lab\_1 LDA (result)   
11 LOD a   
12 LOD b   
13 ADD   
14 STN   
15 lab\_2 RET

结果：测试通过

1. **While语句测试——test2**

源文件：

**void** main()  
{  
 **const** n = 10;  
 **int** i, result ;  
 i = 1;  
 result = 1;  
 **while**(i <= n)  
 {  
 result = result \* i ;  
 i = i + 1 ;  
 }  
}

输出：

No. Lab. Instruction Operator.  
0 LDA (i)   
1 LOD 1   
2 STN   
3 LDA (result)   
4 LOD 1   
5 STN   
6 lab\_1 LOD i   
7 LOD n   
8 LSE   
9 BRF lab\_2   
10 LDA (result)   
11 LOD result   
12 LOD i   
13 MUL   
14 STN   
15 LDA (i)   
16 LOD i   
17 LOD 1   
18 ADD   
19 STN   
20 BR lab\_1   
21 lab\_2 RET

结果：测试通过

1. **If-else嵌套测试——test3**

源文件：

**void** main()  
{  
 **const** a = 10, b = 20, c = 30 ;  
 **int** result ;  
 result = 0;  
 **if** (b > a + c)  
 result = 1 ;  
 **else  
 if**( b < a + c)  
 result = 2 ;  
 **else  
 if**( b == a + c)  
 result = 3 ;  
 **return** ;  
}

输出：

No. Lab. Instruction Operator.  
0 LDA (result)   
1 LOD 0   
2 STN   
3 LOD b   
4 LOD a   
5 LOD c   
6 ADD   
7 GRT   
8 BRF lab\_1   
9 LDA (result)   
10 LOD 1   
11 STN   
12 BR lab\_2   
13 lab\_1: LOD b   
14 LOD a   
15 LOD c   
16 ADD   
17 LES   
18 BRF lab\_3   
19 LDA (result)   
20 LOD 2   
21 STN   
22 BR lab\_4   
23 lab\_3: LOD b   
24 LOD a   
25 LOD c   
26 ADD   
27 EQL   
28 BRF lab\_5   
29 LDA (result)   
30 LOD 3   
31 STN   
32 lab\_5:lab\_4:lab\_2: RET

结果：测试通过

1. **词法错误、语法错误测试——test4**

源文件：

**void** main()  
{  
 **const** c1= 1, c2 = 2 ;  
 **int** result  
 result = c1 . c2 ;  
}

输出：

Error No.3 In line 5 at 17, Word Analysis Error: Unrecognized Char.  
 result = c1 . c2 ;  
 ^  
Error No.32 In line 4 at 13, Gramma Analysis Error: a ';' after end of sentence is Expected.  
 int result  
 ^  
Error No.71 In line 5 at 10, Symbol Table Error: Try to use a undefined idntifier.  
 result = c1 . c2 ;  
 ^  
Error No.34 In line 5 at 22, Gramma Analysis Error: '=' is expected.  
 result = c1 . c2 ;  
 ^  
No. Lab. Instruction Operator.  
0 RET

结果：主要错误能找出，但是有误报情况

1. **符号表错误测试——test5**

源文件：

void main()  
{  
 const c1= 1, c2 = 2 ;  
 int result ;  
 nonexist = c1 + c2 ;  
 c1 = 10 ;  
}

输出：

Error No.71 In line 5 at 12, Symbol Table Error: Try to use a undefined identifier.  
 nonexist = c1 + c2 ;  
 ^  
Error No.34 In line 5 at 24, Gramma Analysis Error: '=' is expected.  
 nonexist = c1 + c2 ;  
 ^  
Error No.72 In line 6 at 6, Symbol Table Error: Trying to assign to a const.  
 c1 = 10 ;  
 ^  
No. Lab. Instruction Operator.  
0 LOD 10   
1 STN   
2 RET

结果：主要错误能够识别，但是仍有误报情况