**编译原理与设计实验报告**

**姓名/学号**： 鲍吴迪/1120173588

# 实验名称

语法分析实验

# 实验目的

(1)熟悉C语言的语法规则，了解编译器语法分析器的主要功能;

(2)熟练掌握典型语法分析器构造的相关技术和方法，设计并实现具有一

定复杂度和分析能力的C语言语法分析器;

(3)了解ANTLR的工作原理和基本思想，学习使用工具自动生成语法分析

器;

(4)掌握编译器从前端到后端各个模块的工作原理，语法分析模块与其他

模块之间的交互过程。

# 实验内容

该实验选择C语言的一个子集，基于BIT-MiniCC构建C语法子集的语法分

析器，该语法分析器能够读入词法分析器输出的存储在文件中的属性字符流，进

行语法分析并进行错误处理，如果输入正确时输出JSON 格式的语法树，输入不.

正确时报告语法错误。

# 实验过程

本实验基于BIT-MINICC中的ExampleParser，对以下语句进行了语法分析实现：

语句块、混合语句、表达式语句、if-else选择语句、for循环语句、返回语句、声明语句（变量声明、数组声明）

常量表达式、赋值表达式、关系表达式、二元运算表达式、条件表达式

1. 文法：
2. <语句>→<混合语句>| <表达式语句> |<选择语句> |<循环语句>｜<返回语句>
3. <混合语句>→{<语句块列表>}

<语句块列表>→<语句块>| < 语句块列表><语句块>

<语句块>→<声明><语句>

<声明>→ <说明符> <初始化列表>

<初始化列表> → <声明符> <表达式列表>

<表达式列表>→<表达式>| < 表达式列表><表达式>

<声明符>→ <数组声明符>｜<变量声明符>｜<函数声明符>

<数组声明符> → <标识符> [ 表达式 ]

<变量声明符> → <标识符>

<函数声明符> → <变量声明符><参数>

<参数> → <说明符> <声明符>

<说明符>→int

1. <表达式语句>→<表达式>;

<表达式>→<基本表达式>|<赋值表达式>｜<关系表达式>｜<二元运算表达式>｜条件表达式

<基本表达式>→<标识符>｜<常量表达式>

<赋值表达式>→<标识符> = <表达式>

<关系表达式>→<表达式> < <表达式>｜<表达式> > <表达式>

<二元运算表达式>→<表达式> + <表达式>｜<表达式> - <表达式> | <表达式> \* <表达式>｜<表达式> / <表达式>

<条件表达式>→<关系表达式>?<表达式>:<表达式>

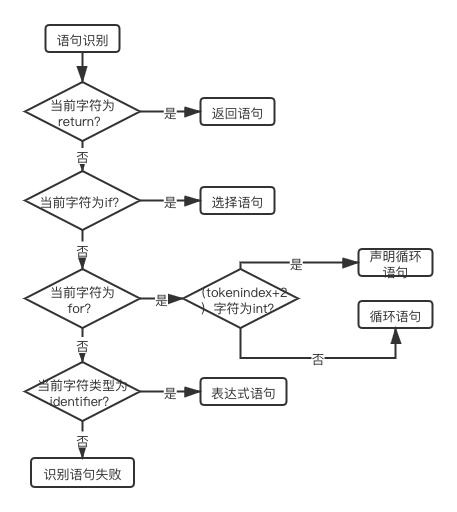
1. <选择语句>→if(<表达式>)<语句> | if(<表达式>) <语句> else<语句>
2. <循环语句>→FOR(<表达式>;<表达式>;<表达式>)<语句>|FOR(<声明>;<表达式>;<表达式>)<语句>
3. <返回语句>→return <表达式>;
4. 算法流程

主要采用bit-minicc中定义好的框架，自己定义了ASTAssignmentExpression, ASTRelationalRxpression两个结点用于识别赋值表达式和关系表达式。

实现时采用递归下降分析方法，为不同类型的语法树结点ASTNode构造识别函数，利用函数间相互调用关系，自顶向下为输入的代码串找到识别序列，主要采用LL(1)算法进行识别，必要时添加了对当前字符后1-2个字符的判断，以在避免回溯的情况下产生语法树。

对于确定的终结符如 {}、；、()、identifier、int、if、for等,将其与词法分析产生的token结果进行匹配，再进行下一个token的识别（对应归约到非终结符，以及符号的移进栈操作）

部分算法流程图如下：

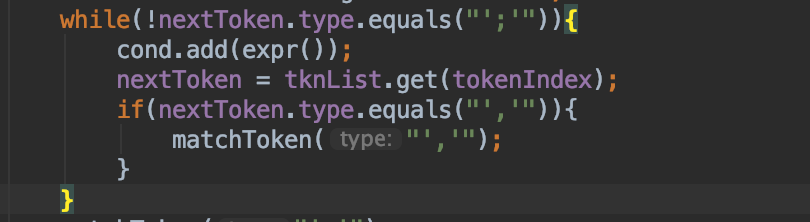


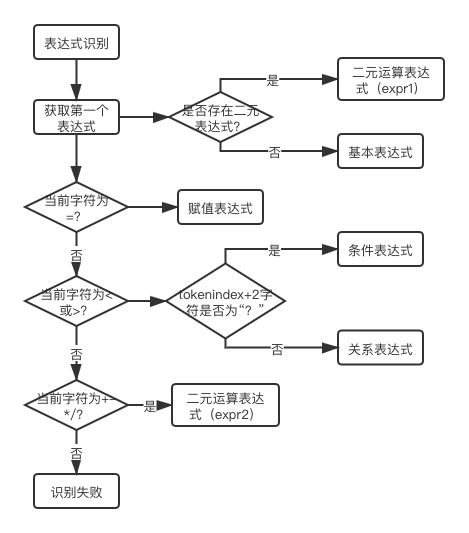
语句识别

在循环语句识别中，对for(后的符号类型进行判断FOLLOW( for ( ),如果是int类型，则判断为声明循环语句，其init属性为声明语句（ASTDeclaration），如果是identifier类型，则判断为普通循环语句，其init属性为表达式（ASTExpression）。



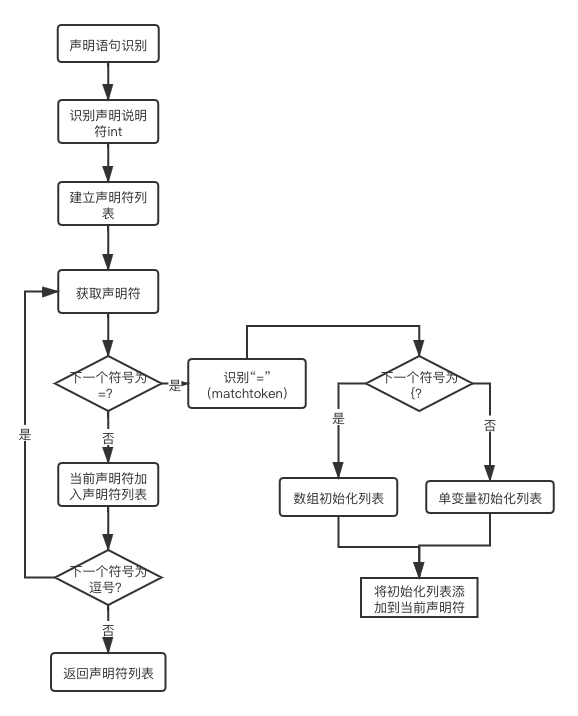
两种循环语句中，init属性后的判断（cond，step，stat属性）均相同，在识别到本属性结束符； 、 ） 、 }前重复添加语句或表达式到当前list中。





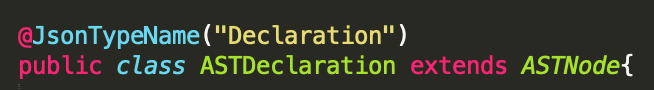
表达式识别

其中所有带符号的表达式都采取第一个表达式与（符号、第二个表达式）分次识别后再合成的方式。

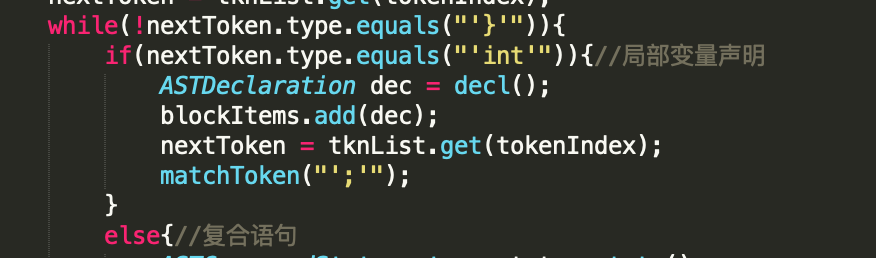


声明语句识别

声明语句一开始被我放在了语句识别函数stmt（）中，判断当前识别字符是否为int，但后来发现ASTDeclaration是不继承于ASTStatement的独立结点，因此在语句识别函数中无法返回一个语句。

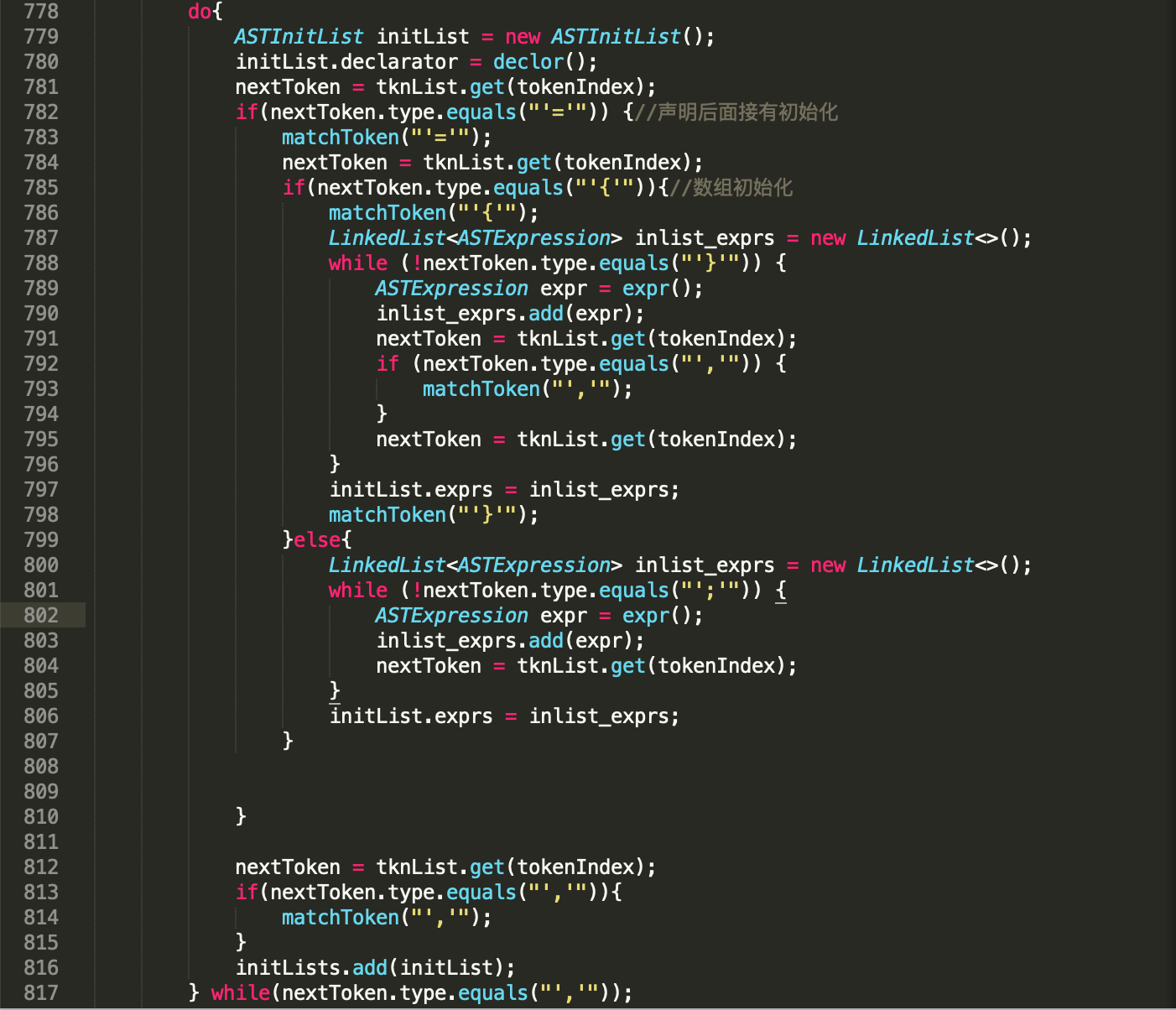


因此将它作为独立的blockitem类型（语句块中的一项），直接将结点添加到语句块中。



其次，理解声明中declaration、declarator的区别，specifier、initlists的具体含义一开始也比较困难。首先一个declaration是一整个声明语句，它包含的（声明的类型如int）🡪specifier，以及（初始化列表如a、如b=1）🡪initlist，而declarator是所要声明的变量（如int a中的a），它包含在声明语句的初始化列表中，除此之外，初始化列表还可能包含表达式（如int b=1中的b=1）

因此，我在declaration的识别函数decl（）中先对声明说明符（int）进行了识别和匹配，然后在识别到表示声明未结束的符号“，”时，不断调用declarator的识别函数declor（）以实现对与一次声明多个声明符时初始化列表的构造（构造时首先识别是对单变量进行声明还是对数组进行声明）。



1. 识别测试

采用如下代码进行语法分析测试

int sum(int a, int b) {

int a; //单变量声明

int b[9]; //数组变量声明

int c[3] = {1,2,3}; //数组初始化声明

int a,b=1; //多变量声明

a<b?a+1:b+1; //条件表达式（包括关系表达式）

if(a < b){ //选择语句

return a + b;

} else{

return a - b;

}

if(c)

return c;

a=b;

for(i=b;i<b;i+b){ //循环语句

i+1;

}

for(int i=0;i<1;i+b){

i+1;

}

for(i=0;i<1;i+1)

a+b;

}

1. 实验结果

输出json格式语法树能与内置分析器达到基本一致

对返回语句return a+b;的识别：



对选择语句的识别：

if(a < b){

return a + b;

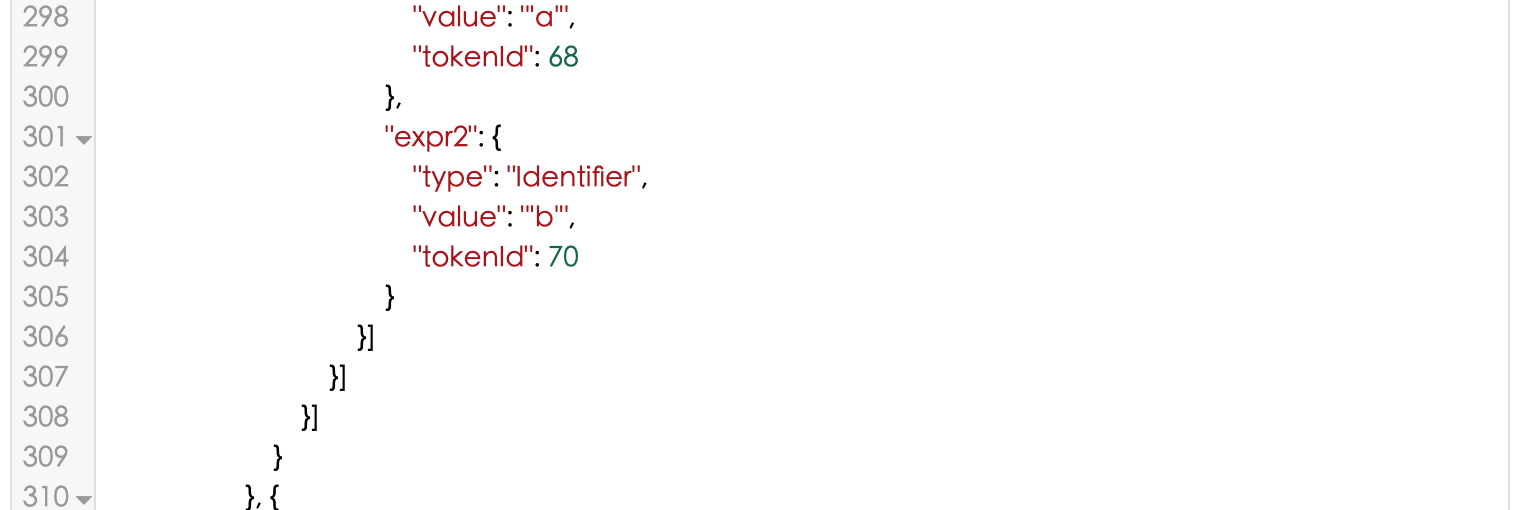
} else{

return a - b;

}







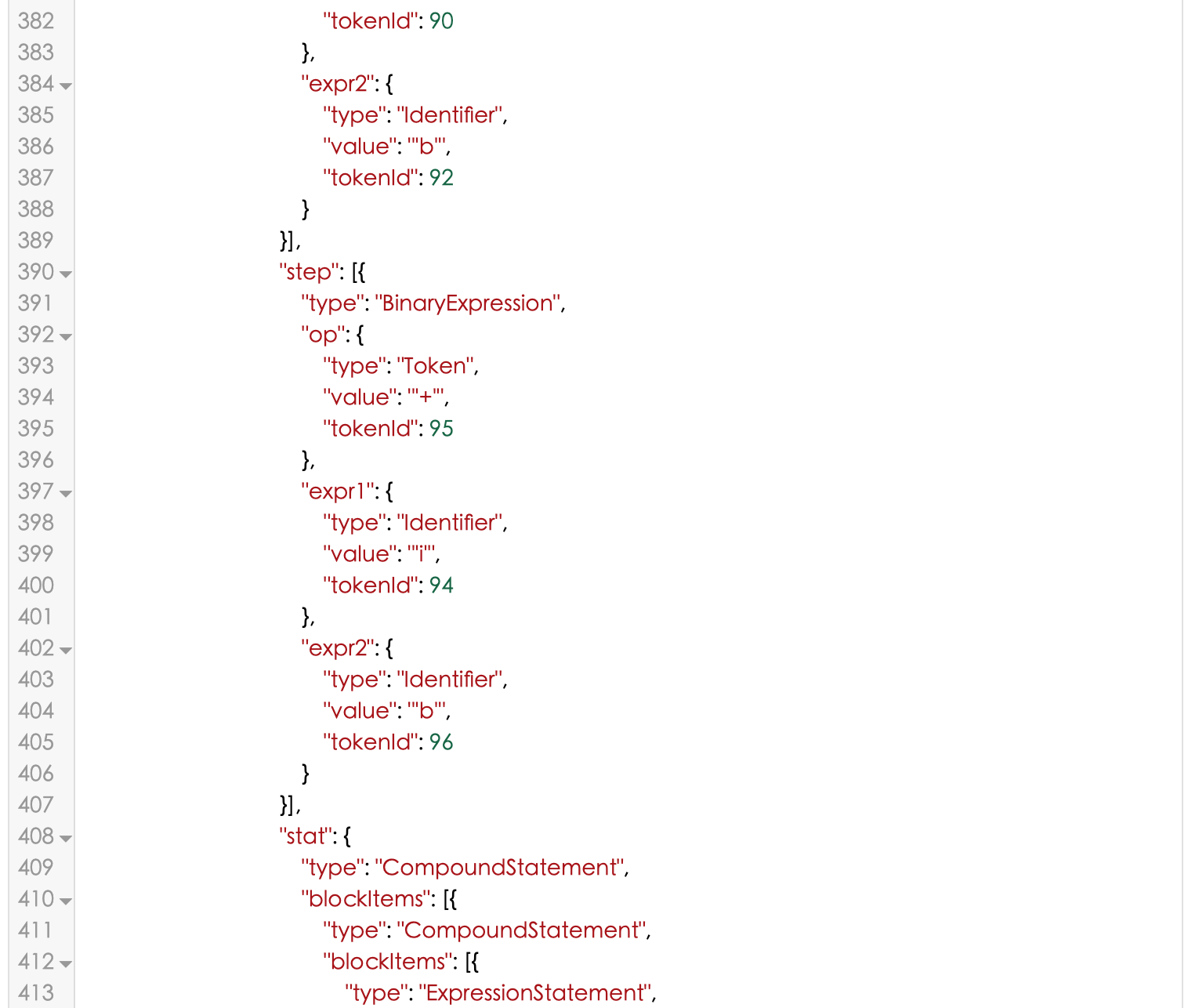
对循环语句的识别：

for(int i=0;i<1;i+b){

i+1;

}







对数组初始化声明语句的识别：

int c[3] = {1,2,3};



对条件表达式（语句）的识别：

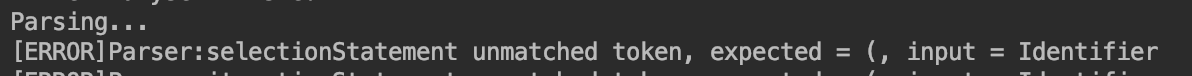
a<b?a+1:b+1;



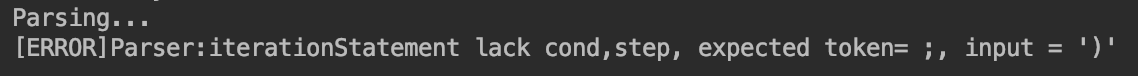
1. 错误处理

能对基本语法错误进行输出，并报告错误出现的语句类型、错误位置等。

如if语句缺少括号时：



For循环语句缺少判断条件、执行语句时：



# 实验心得与体会

这次实验中，主要是很深切的体会了“万事开头难”的道理，从下载新的bitminicc框架开始，两天都没有写出一行代码，（因为没有看懂示例的语法分析器），到后来通过一步步调试去理解代码，再到自己可以修改代码，对语法分析整个流程的掌握也有了很大的提高。

在编写自己的语法分析器的过程中，对我来说最大的困难应该是文法之间的相互调用和包含关系，很多时候在用新的代码段进行测试时，比如刚开始写赋值语句时，自己也不是很清楚应该从语句类别去识别还是从表达式类别，导致后来写了ASTAssignmentExpression和ASTAssignmentStatement两个结点，然后在语句中添加了表达式语句后，ASTAssignmentStatement就彻底废弃了。做这种无用功的原因其实是我自己没有把文法设计好就直接开始编写代码和测试导致的结构混乱的问题，后来重新选取语法集、设计文法，既避免了更多的bug，也能让自己的代码框架更清楚和有条理。

即使这次实验时间加长了，但仍有许多没有完成好的地方没来得及做，比如变量声明只支持int类型，循环语句、选择语句只支持一种形式等，需要进行后期的完善。