

# 同济大学计算机系

## 编译原理-中间代码生成实验报告



学 号 1952651

姓 名 杨凡

专 业 计算机科学与技术

# 编译原理实验报告

## 设计说明

### 1、运行和开发环境

操作系统:MacOS Big Sur 11.5.2

内存:8 GB 2133 MHz LPDDR3

处理器:2.4 GHz 四核 Intel Core i5

开发语言: Html CSS JavaScript 开发框架和外部库: Bootstrap 4.3.1 jQuery 3.6.0

编辑器:VSCODE

编译器:Google chrome

运行环境:使用 Google chrome 浏览器直接打开对应的 html 文件即可运行。

采用的编码格式:本程序采用 UTF-8 的编码格式

### 2、程序说明

#### 2.1 能识别的单词

能识别的单词如下：

关键字：int | void | if | else | while | return

标识符：字母（字母 | 数字）\* （不与关键字相同）

数值：数字（数字）\*

赋值号： =

算符： + | - | \* | / | = | == | > | >= | < | <= | !=

界符： ；

分隔符： ，

注释号： /\* \*/ | //

左括号： (

右括号： )

左大括号： {

右大括号： }

字母： | a | ... | z | A | ... | Z |

数字： 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

结束符： #

## 2.2 能分析的语法

能分析的语法如下：

Program ::= <类型> < ID> '(' ' ' )' <语句块>

<类型

> ::= int | void

::= 字母(字母 | 数字)\*

<语句块> ::= '{' <内部声明> <语句串> '}'

<内部声明> ::= 空 | <内部变量声明> '{' <内部变量声明> }

<内部变量声明> ::= int （注： { } 中的项表示可重复若干次）

<语句串> ::= <语句> { <语句> }

<语句> ::= <if语句> | <while语句> | <return语句> | <赋值语句>

<赋值语句> ::= = <表达式>;

**<return语句> ::= return ( <表达式> )** (注: ( )中的项表示可选)

$$\langle \text{while 语句} \rangle ::= \text{while } \langle \text{表达式} \rangle \langle \text{语句块} \rangle$$
$$\langle \text{if语句} \rangle ::=$$

if ‘(‘<表达式>’)' <语句块> ( else <语句块> ) (注: ( )中的项表示可选)

$$\langle \text{表达式} \rangle ::= \langle \text{加法表达式} \rangle \{ \text{relop } \langle \text{加法表达式} \rangle \} \quad (\text{注: relop} \rightarrow < | <= | > | >= | == | !=)$$
$$\langle \text{加法表达式} \rangle ::= \langle \text{项} \rangle \{ + \langle \text{项} \rangle \mid - \langle \text{项} \rangle \}$$
$$\langle \text{项} \rangle ::= \langle \text{因子} \rangle \{ * \langle \text{因子} \rangle \mid / \langle \text{因子} \rangle \}$$
$$\langle \text{因子} \rangle ::= \text{ID} \mid \text{num} \mid '(' \langle \text{表达式} \rangle ')'$$

### 3、分析算法的主程序框图

算法采用了自上而下的分析方法，通过递归下降对词法分析得到的串进行了语法分析，同时对其进行了语义分析，得到了四元式列表。

基本逻辑就是通过对于每个非终结符实现一个函数：

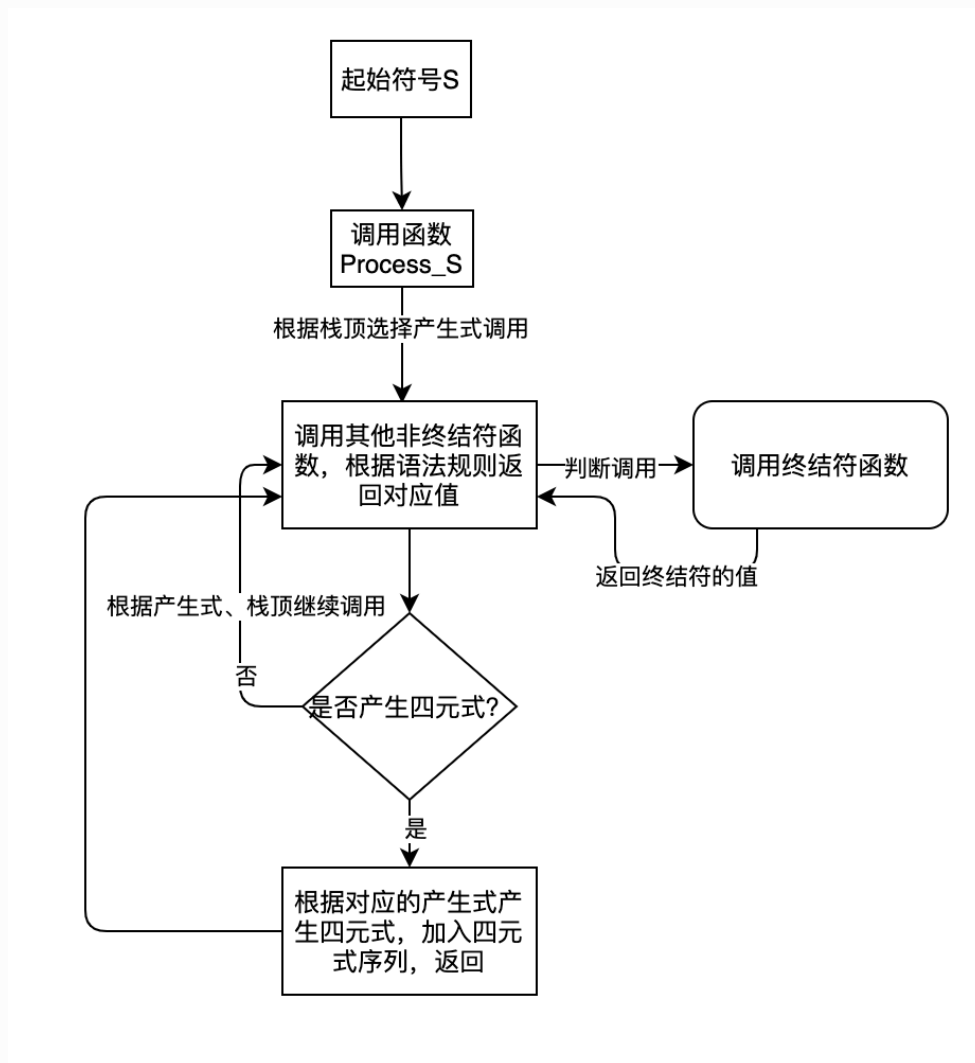
```

process_S() // S → T ID ( D' S' ...
}
process_SPLUS() // S' → B # ...
}
process_T() // T → INT | VOID ...
}
process_B() // B → { C E } ...
}
process_C() // C → INT ID ; C | ε ...
}
process_D() // D → INT ID ...
}
process_DPLUS() // D' → D D_ | ε ...
}
process_DPLUSPLUS() // D_ → , D D_ | ε ...
}
process_E() // E → F V ...
}
process_V() // V → F V | ε ...
}
process_F() // F → G | H | I | J ...
}
process_J() // J → ID = K ; ...
}
process_I() // I → RETURN W ...
}
process_W() // W → K ; | ; ...
}
process_H() // H → WHILE ( K ) B ...
}
process_G() // G → IF ( K ) B G' |

```

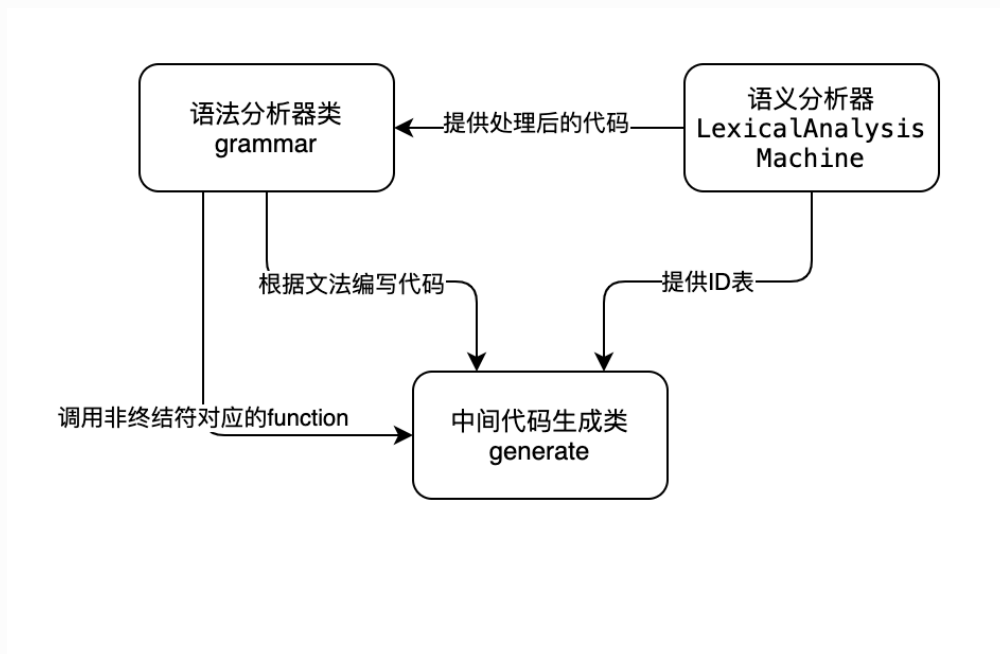
如上图，基本每个过程就根据分析栈栈顶的元素进行函数的调用，而出栈入栈所遵循的规则，都为实验1-2语法分析器中实现的预测分析表中保留栈的过程，这里就不再描述。

程序总体分析的逻辑如下:



如上图，程序从起始符号S开始不断调用其他的符号对应的函数。调用的规则根据产生式决定。对应函数的返回也根据产生规定。同时，根据符号返回的综合属性和符号对应的产生式，来决定产生对应的语义动作。同时，符号对应也存在一定的继承属性，这里通过函数的参数来进行传递。

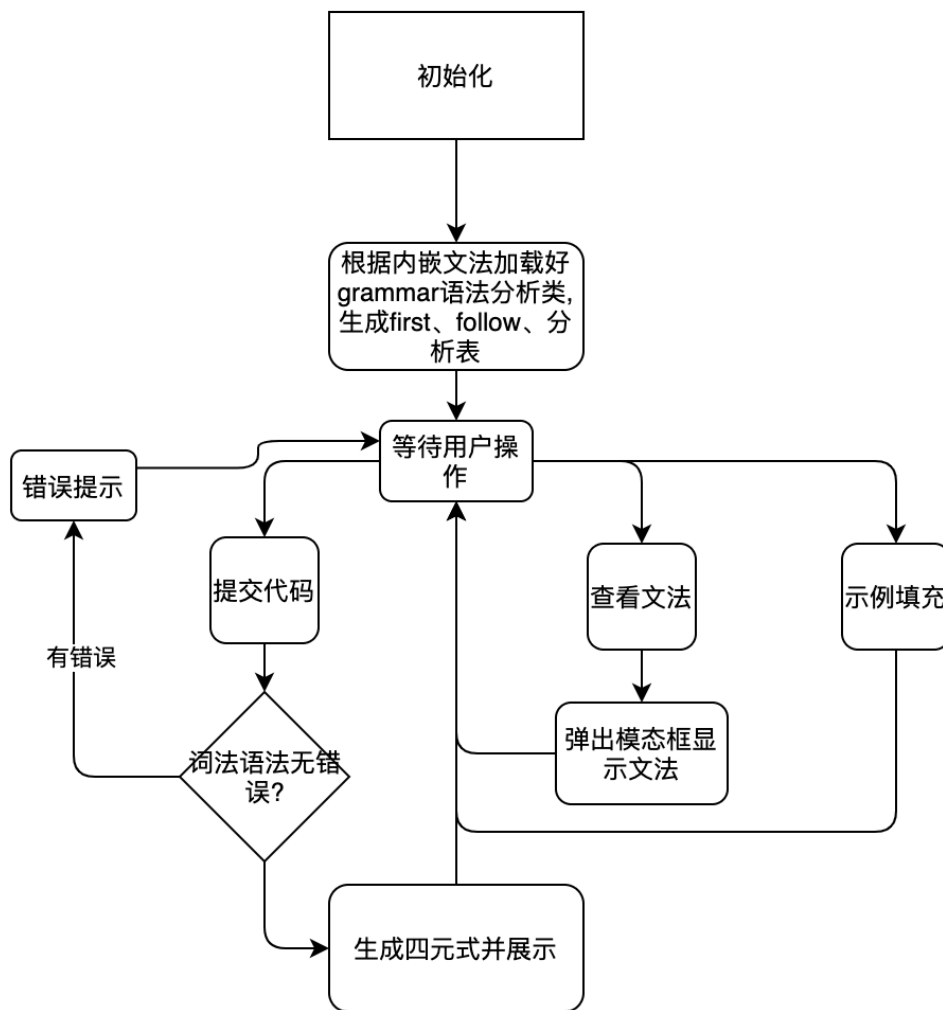
语义分析、语法分析、词法分析这里使用了JavaScript的三个class来实现：class直接的关系为下图：



在generate里面采用了一个JavaScript内置的数据结构map用来作为对已经声明的变量进行记录的数据结构，具体为ID->Type的键值类型，具体结构如下所示：

```
< ▼ Map(6) {'i' => 'INT', 'j' => 'INT', 'k' => 'INT', 'a' => 'INT', 'b' => 'INT', ...}
  ▼ [[Entries]]
    ▶ 0: {"i" => "INT"}
    ▶ 1: {"j" => "INT"}
    ▶ 2: {"k" => "INT"}
    ▶ 3: {"a" => "INT"}
    ▶ 4: {"b" => "INT"}
    ▶ 5: {"c" => "INT"}
    size: 6
    ▶ [[Prototype]]: Map
```

程序总系统功能流程图如下：



## 错误处理：

下面有本程序对错误的处理的截图，这里只讲述本程序对错误处理的基本逻辑。本程序通过对已定义的变量构建一张key-value的map，在后续每次对变量进行访问的时候，都要查询这张map，查看是否存在对应我需要的ID，如果不存在，那么会报错，并在前端界面进行显示。

## 4、运行结果截图

### 正常运行结果：

通过文法输入框、代码输入框输入文法，程序结果如下：

编译原理课程实验

语义分析器

1952651 杨凡

输入您的代码和文法(在下方的操作框或者通过文件)

■输入代码

```
/*这是用于测试的代码123*/
/*123你好*/
//123, 国庆节快乐
int main()
{
    int i; //这是i
    int j;
    int k;
    int a;
    int b;
    int c;
    i = 10;
    j = 1*2*4*i+2/i/3*4;

    if(a>(b+c))
    {
        j=a+(b*c+1);
    }
    else
    {
```

选择文件

未选择任何文件

语义分析

查看文法

示例填充

语义分析结果:

```
0: (:=,10,_,i)
1: (*,4,i,t14)
2: (*,2,t14,t15)
3: (*,1,t15,t16)
4: (*,3,4,t17)
5: (/i,t17,t18)
6: (/2,t18,t19)
7: (+,t16,t19,t20)
8: (:=,t20,_,j)
9: (+,b,c,t21)
10: (J>,a,t21,12)
11: (J,_,_,17)
12: (*,b,c,t22)
13: (+,t22,1,t23)
14: (+,a,t23,t24)
15: (:=,t24,_,i)
16: (J,_,_,18)
17: (:=,a,_,j)
18: (+,100,3,t25)
19: (J<=,i,t25,21)
20: (J,_,_,24)
21: (*,j,2,t26)
22: (:=,t26,_,i)
23: (J,_,_,18)
```

本实验基于先前的词法分析器和语法分析器，因此结合了先前内容的对于语法的错误处理，这里简单介绍先前对于语法分析和词法分析的错误处理。

输入代码的词法分析错误，会进行错误提示，错误提示如下：

错误提示

×

注释未闭合

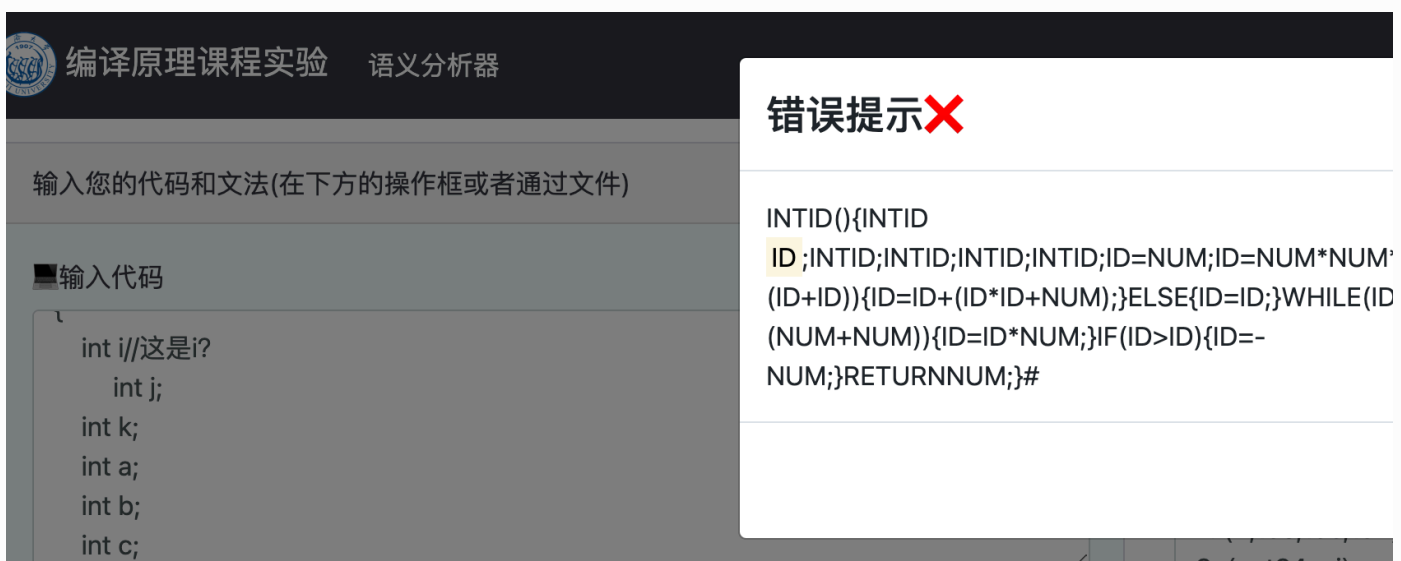
关闭

或者如下：

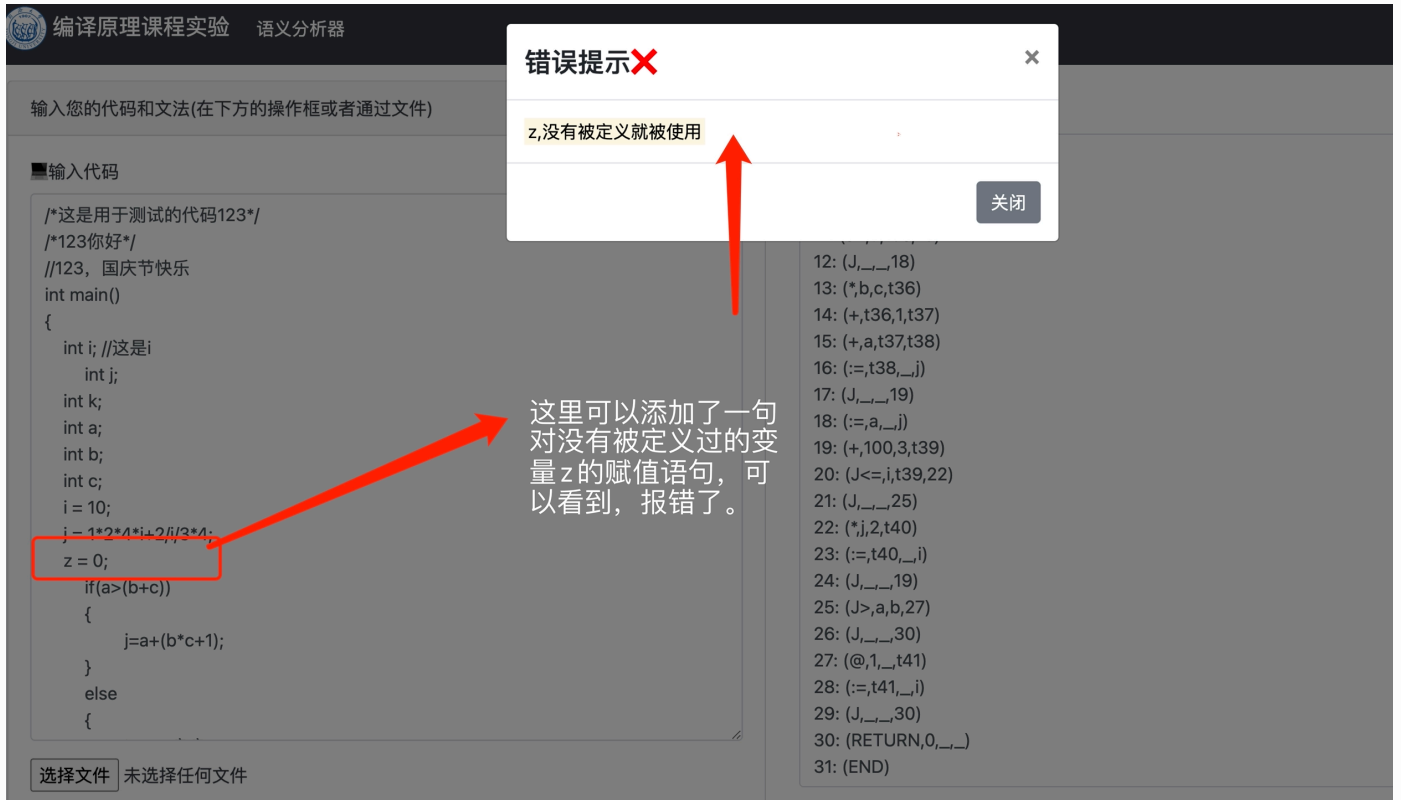




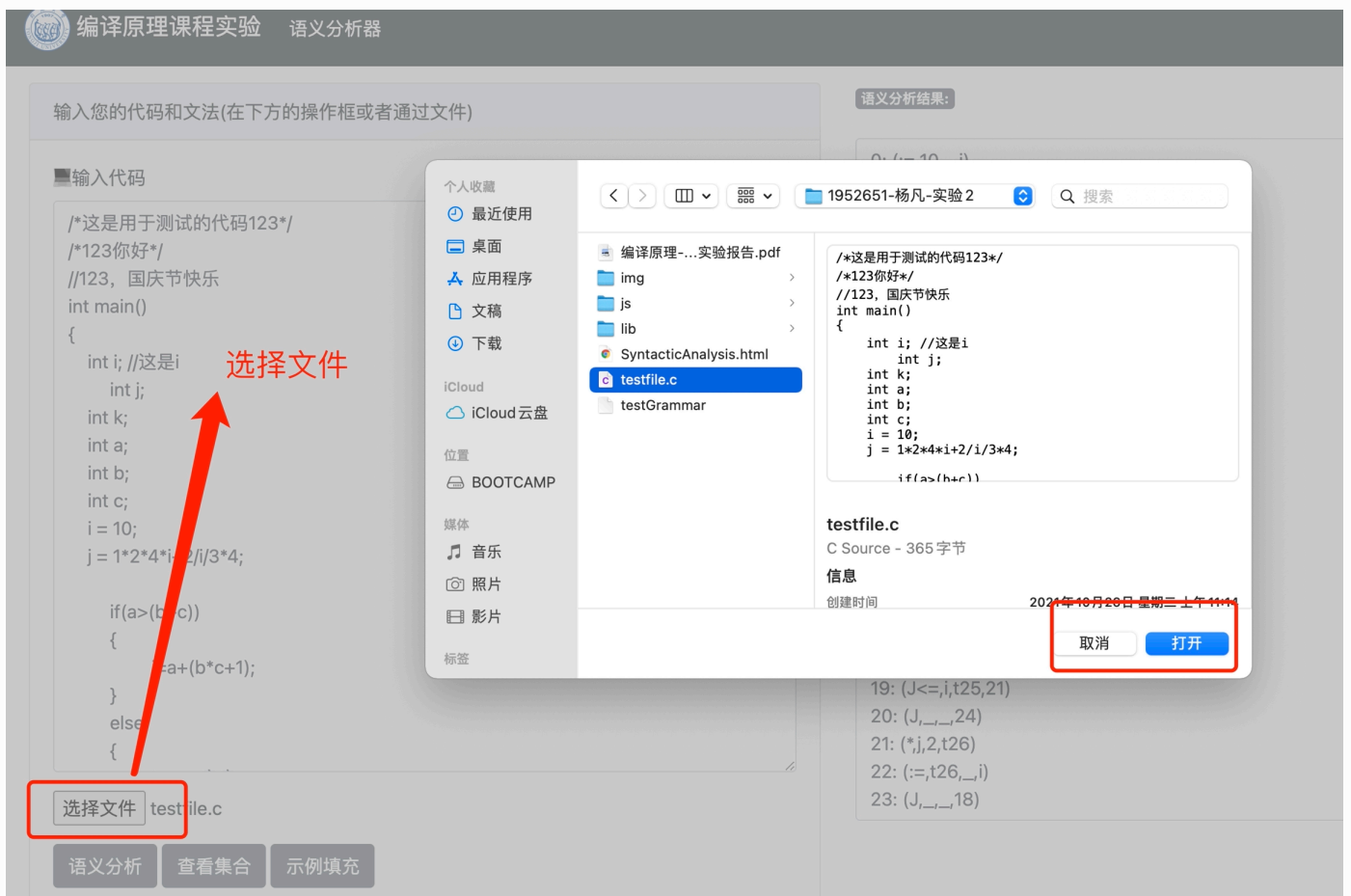
对语法分析的错误处理如下：



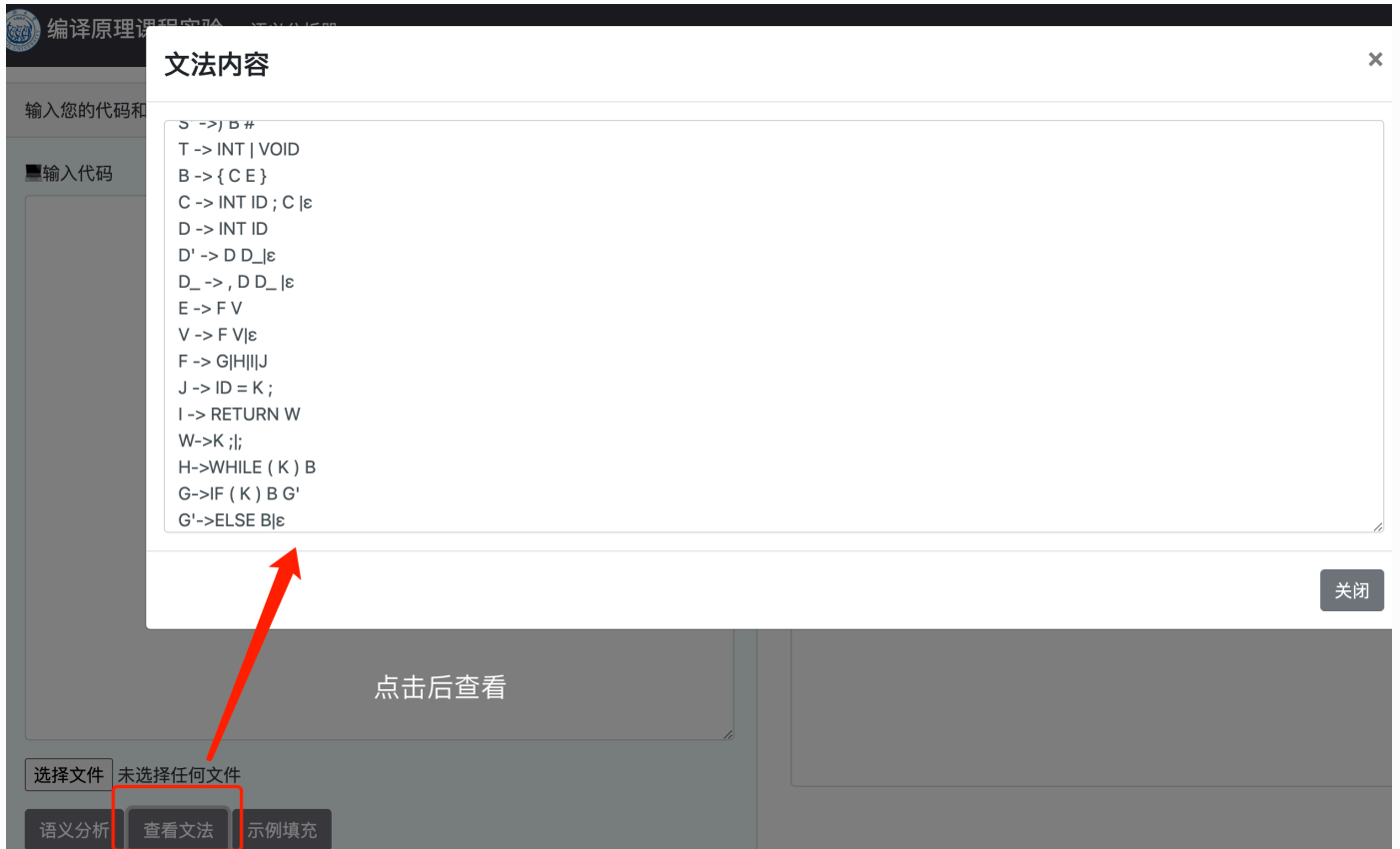
语义的错误提示：



## 文件输入:



## 查看内嵌文法:



## 测试用例

下面对程序进行测试，由于程序大体在词法分析、语法分析阶段沿用了实验1-1、实验1-2词法分析器、语法分析器的设置，因此这里就不再对相关的内容进行测试，下面用下述的类C语言代码和文法进行语义分析：

类C语言代码：

```
/*这是用于测试的代码123*/  
/*123你好*/  
//123，国庆节快乐  
int main()  
{  
    int i; //这是i  
    int j;  
    int k;  
    int a;
```

```

    int b;
    int c;
    i = 10;
    j = 1*2*4*i+2/i/3*4;

    if(a>(b+c))
    {
        j=a+(b*c+1);
    }
    else
    {
        j=a; //
    }
    while(i<=(100+3))
    {
        i=j*2;
    }
    if(a>b)
    {
        i=-1;
    }
    return 0;//返回0!!!
}

```

## 使用的文法：

S S' T B C D D' D\_ E V F J I W H G G' K L M N N' Q Y R U  
 ID INT VOID ; , ( ) { } = < <= > >= == != + - \* / IF WHILE RETURN ELSE NUM #  
 S -> T ID ( D' S'  
 S' -> ) B #  
 T -> INT | VOID  
 B -> { C E }  
 C -> INT ID ; C |  $\epsilon$   
 D -> INT ID  
 D' -> D D |  $\epsilon$   
 D -> , D D\_ |  $\epsilon$   
 E -> F V  
 V -> F V |  $\epsilon$   
 F -> G | H | I | J  
 J -> ID = K ;

$I \rightarrow \text{RETURN } W$   
 $W \rightarrow K ; | ;$   
 $H \rightarrow \text{WHILE } ( K ) B$   
 $G \rightarrow \text{IF } ( K ) B G'$   
 $G' \rightarrow \text{ELSE } B | \epsilon$   
 $K \rightarrow L Q$   
 $L \rightarrow M R$   
 $M \rightarrow N U$   
 $N \rightarrow N' | \epsilon$   
 $N' \rightarrow \text{ID} | \text{NUM} | ( K )$   
 $Q \rightarrow Y L Q | \epsilon$   
 $Y \rightarrow < | < = | > | > = | = = | ! =$   
 $R \rightarrow + M R | - M R | \epsilon$   
 $U \rightarrow * N' U | / N' U | \epsilon$

产生的四元式结果：


0: ( $:=, 10, i$ )  
 1: ( $, 4, i, t14$ )  
 2: ( $, 2, t14, t15$ )  
 3: ( $, 1, t15, t16$ )  
 4: ( $, 3, 4, t17$ )  
 5: ( $/, i, t17, t18$ )  
 6: ( $/, 2, t18, t19$ )  
 7: ( $+, t16, t19, t20$ )  
 8: ( $:=, t20, j$ )  
 9: ( $+, b, c, t21$ )  
 10: ( $J>, a, t21, 12$ )  
 11: ( $J, ,, 17$ )  
 12: ( $, b, c, t22$ )  
 13: ( $+, t22, 1, t23$ )  
 14: ( $+, a, t23, t24$ )  
 15: ( $:=, t24, j$ )  
 16: ( $J, ,, 18$ )  
 17: ( $:=, a, j$ )  
 18: ( $+, 100, 3, t25$ )  
 19: ( $J<=, i, t25, 21$ )

```

20: (J,,,24)
21: (j,2,t26)
22: (:=,t26,,i)
23: (J,,,18)
24: (J>,a,b,26)
25: (J,,,29)
26: (@,1,,t27)
27: (:=,t27,,i)
28: (J,,,29)
29: (RETURN,0,,_)
30: (END)

```

## 程序界面的运行结果：


编译原理课程实验 语义分析器
1952651 杨凡

输入您的代码和文法(在下方的操作框或者通过文件)

■ 输入代码

```

/*这是用于测试的代码123*/
/*123你好*/
//123, 国庆节快乐
int main()
{
    int i; //这是i
    int j;
    int k;
    int a;
    int b;
    int c;
    i = 10;
    j = 1*2*4*i+2//3*4;

    if(a>(b+c))
    {
        j=a+(b*c+1);
    }
    else
    {

```

选择文件

未选择任何文件

语义分析

查看文法

示例填充

语义分析结果:

```

0: (:=,10,_,i)
1: (*,4,i,t14)
2: (*,2,t14,t15)
3: (*,1,t15,t16)
4: (*,3,4,t17)
5: (/,i,t17,t18)
6: (/,2,t18,t19)
7: (+,t16,t19,t20)
8: (:=,t20,_,i)
9: (+,b,c,t21)
10: (J>,a,t21,12)
11: (J,_,_,17)
12: (*,b,c,t22)
13: (+,t22,1,t23)
14: (+,a,t23,t24)
15: (:=,t24,_,i)
16: (J,_,_,18)
17: (:=,a,_,i)
18: (+,100,3,t25)
19: (J<=,i,t25,21)
20: (J,_,_,24)
21: (*,j,2,t26)
22: (:=,t26,_,i)
23: (J,_,_,18)

```

根据课堂所学的知识，和基本类C文法判断，生成的四元式符合文法、类C语言的要求。测试正确。

下面刻意添加错误的代码，观察程序是否能够发现并报错，具体说明如图所示：



## 源代码

所有源代码都在附录中，因此只对文件进行说明。

文件说明:

**lib** - 使用的第三方前端库的文件

**js/Action.js** - 前端js交互文件

**js/LexicalAnalysis.js** - 词法分析器文件

**Js/Grammar.js** - 语法分析+中间代码生成文件

**SyntacticAnalysis.html** - 程序网页主体文件，用浏览器打开即可运行程序

**testfile.c** - 测试文件，可以用来进行测试程序的正确性

# 可执行代码:

## 文件说明:

**SyntacticAnalysis.html** - 程序网页主体文件，用浏览器打开即可运行程序