中山大學 SUN YAT-SEN UNIVERSITY

# 编译原理

| 作业名称 | Oberon-0 逆向工程工具 Rose<br>实验四：手工编写递归下降预测分析程序 | | 任课老师 | 万老师 | | |
|---|---|---|---|---|---|---|
| 院系 | 软件学院 | | 方向 | 通信软件 | 姓名 | 李明宽 |
| 学号 | 11331173 | | 完成日期 | 2014 年 01 月 05 日 | | |
| QQ | 736459905 | | E-mail | limkuan@mail2.sysu.edu.cn | | |

## 实验四：手工编写递归下降预测分析程序

# 一、 实验目的

本实验要求使用 Java 语言手工编写一个 Oberon-0 语言的语法分析程序，完成对 Oberon-0 语言的语法分析过程，然后生成 Oberon-0 语言的流程图。并且了解使用递归下降预测语法分析技术，使用语法制导翻译思想完成语法分析程序。

# 二、 实验过程

## 1. 设计 Oberon-0 语言的翻译模式

由于使用递归下降预测的语法分析方法，因此原来给定的上下文无关文法中会出现左递归，为了避免死循环，这种左递归是必须消除的。在实现过程中，我设计的翻译模式如下：（详见文件<trans_mode.txt>）

PS：语法动作由于比较复杂就没有在下面的翻译模式中列举出来了。

```
parse                -> module ;
module               -> "MODULE" IDENTIFIER ";" declaration module_key "END" ".";
module_key           ->  | "BEGIN" statement;

declaration          -> const_declarations type_declarations var_declarations
procedure_declarations;
const_declarations   ->  | "CONST" const_declarations_key;
const_declarations_key ->  | IDENTIFIER "=" expression ";" const_declarations_key;
type_declarations    ->  | "TYPE" type_declarations_key;
type_declarations_key  ->  | IDENTIFIER "=" type ";" type_declarations_key;
var_declarations     ->  | "VAR" var_declarations_key;
var_declarations_key   ->  | identifier_list ":" type ";" var_declarations_key;

procedure_declarations ->  | "PROCEDURE" procedure_heading procedure_body procedure_declarations;
procedure_body       -> declaration procedure_body_key "END" IDENTIFIER ";" ;
procedure_body_key   ->  | "BEGIN" statement;
procedure_heading    -> IDNETIGIER formal_parameters ";" ;
formal_parameters    ->  | "(" fp_section fp_section_key ")" ;
fp_section           -> var_key identifier_list ":" type ;
var_key              ->  | "VAR";
fp_section_key       ->  | ";" fp_section fp_section_key ;
```

```
type                  -> INTEGER
                       | BOOLEAN
                       | array_type
                       | record_type
                       | IDENTIFIER;
array_type            -> "ARRAY" expression "OF" type;
record_type           -> "RECORD" field_list field_list_key "END";
field_list            -> identifier_list ":" type ;
field_list_key        ->  | ";" field_list field_list_key ;

statement             -> if_statement statement_key
                       | while_statement statement_key
                       | selector ":=" expression statement_key
                       | procedure_call statement_key;
statement_key         ->  | ";" statement;

procedure_call        -> IDENTIFIER "(" actual_parameters actual_parameters_key ")";
actual_parameters     ->  | expression;
actual_parameters_key ->  | ";" actual_parameters actual_parameters_key;

if_statement          -> "IF" expression "THEN" statement elseif_statement else_statement "END";
elseif_statement      ->  | "ELSIF" expression "THEN" statement elseif_statement else_statement;
else_statement        ->  | "ELSE" statement;

while_statement       -> "WHILE" expression "DO" statement "END";

identifier_list       -> IDENTIFIER identifier_list_key;
identifier_list_key   ->  | "," IDENTIFIER identifier_list_key;

expression            -> simple_expression
                       | simple_expression operator_1 simple_expression;

simple_expression     -> operator_2 term simple_expression_key;
simple_expression_key ->  | operator_3 term simple_expression_key;

term                  -> factor term_key;
term_key              -> operator_4 factor term_key;

operator_1            -> "=" | "#" | ">" | ">=" | "<" | "<=";
operator_2            ->  | "+" | "-";
operator_3            -> "+" | "-" | "OR";
operator_4            -> "*" | "DIV" | "MOD" | "&";
```

```
factor              -> NUMBER
                     | "(" expression ")"
                     | "~" factor
                     | selector;
selector            -> IDENTIFIER selector_key;
selector_key         ->  | "." IDENTIFIER selector_key
                     | "[" expression "]" selector_key;
```

# 2. 编写递归下降预测分析程序

根据上面得到的翻译模式，使用 Java 语言完成 Oberon-0 语言的语法分析程序，实现自动绘制程序中相应的流程图的功能。其中 Oberon-0 的词法分析程序依然使用 JFlex 生成工具生成。

根据翻译模式能够有比较程序的启发式规则生成递归下降预测分析程序，对于每一个非终端符号都对应一个方法，根据向前看符号决定下一步应该使用哪一个文法推导式，每一个继承属性对应一个形式参数，综合属性对应返回值等等，在这些规则的帮助下设计出递归下降预测分析程序的主题，然后再加上错误判断，流程图的输出等功能，完成了对 Oberon-0 语言的语法分析。

在实际设计过程中，感觉使用了很多函数递归来对应相应的文法，然后每一步通过 lookahead 判断下一个识别到的符号应该是什么，然后根据这个符号得到下一步操作。

# 3. 语法分析讨论：自顶向下 VS 自底向上

实验三使用了 JAVACUP 工具生成了一个 LALR 语法分析器，而实验四中使用 Java 语言实现了一个类似于 LL(1)的递归下降预测分析程序，从以下几个方面对比自顶向下和自底向上这两种不同的语法分析方法：

- 分析技术的简单性：对于设计语法分析程序过程来说，很明显是递归下降预测分析程序编写起来更加简单，而且比自底向上的 LALR 分析更加容易调试，毕竟是使用 Java 语言编写，出现错误或者异常的时候比较容易定位到错误的位置，并且有 IDE 的支持使得在编写代码过程中出现的错误会比较少。对于 JavaCUP 抛出的异常其实还不是很明白是什么意思，如下图所示：

```
Syntax error at character 120 of input
Couldn't repair and continue parse at character 120 of input
Can't recover from previous error(s)
```

如果要调试的时候就要输出 parser 里输出当前的状态，然后找到相应状态的非终结符以及语法推导式，再来判断错误类型。而是用 Java 编写的自顶向下语法分析程序如果出现错误直接可以通过 Exception 中的 Stack Trace 快速定位到出错的位置，调试起来比较方便。

- 分析技术的通用性：自顶向下的递归下降预测分析程序能处理的文法差不多是LL(1)型的文法，必须保证文法没有左递归，否则会出现死循环。而自底向上的语法分析程序能处理的文法范围更大，其中包括LR(0),SLR(1),LR(1),LALR(1)等，适用范围更广。
- 是否便于完成语法制导翻译：自顶向下的预测分析程序中需要使用函数的返回值将综合属性传回，使用参数形式把继承属性传递，而使用自底向上的分析过程是都是使用综合属性，这样处理起来相对比较简单。但递归下降预测分析程序也相对来说比较灵活。
- 出错恢复：由于在具体的实现过程中，如果遇到错误都直接使用 throw 抛出异常终止程序，而没有进行恢复的过程。但感觉上来说使用自底向上的语法分析错误恢复会简单点，通过操作堆栈就可以恢复到错误出现之前的状态，而自顶向下使用函数回溯比较困难。对于错误的判断，相对来说自底向上的语法分析程序能更早发现错误。
- 表格驱动的优劣：如果不考虑表格所占用的空间的大小的话，表格驱动是一个效率比较高的解决方法，就是通过当前状态以及向前看的符号，判断应该使用哪一个文法推导式进行展开，这样能够直接通过查表的方式进行语法分析，效率会更快也会更好理解。编译原理的 Project2-ExprEval 就是通过 OPP 表，然后在分析过程中通过查表判断移入、规约、完成还是异常处理，就是建表格的时候比较烦，需要耗费比较长时间，实现起来还算是比较简单。
- 分析速度：相对来说自底向上的语法分析速度比较快。
- 总的来说，自顶向下的递归下降预测分析程序对于手工编写来说是比较好的，因为实现简单并且容易调试，但对于程序的性能可能没有自底向上的语法分析工具好。但是自底向上的语法分析方法如 LR(0)等手工实现会很复杂，那个分析表很大如果手工建立基本上是不可能的，在程序的调试方面也会比较麻烦。

# 三、 关键问题及解决办法

## 1. 已实现的功能

### 1) 流程图的绘制

在声明主程序或者 Procedure 过程中 BEGIN 的时候往 flowchart 类库中添加一个过程，并且使用一个栈记录当前语句外层的结构，主要针对 if 语句以及 while 语句嵌套这方面的问题，调用在识别到相应的语句后使用 flowchart 中的方法往 Module 添加语句就可以画出来流程图。

要注意的是，流程图中的文本是 HTML 语言格式的，因此如果要表示小于符号<,需要使用 HTML 转移<否则就会被当作标签而不显示出来。

### 2） 类型不匹配异常的判断及处理

处理的方法与第三部分使用 JavaCUP 编写语法分析程序中的类似，都是使用一个 HashMap<String, Variable>来记录每一个 ID 对应的类型，进行运算的时候先获取到相应 ID 对应的类型，然后判断类型是否匹配再进行运算。

### 3） 变量作用域的使用规则

与第三部分处理方法类似，使用一个栈来存储所用的 ID，栈顶中的 HashMap 就是当前作用域下所有的变量，在 declaration 结束的时候把上一层的中在没有与当前作用域冲突的变量加到当期啊作用域，然后把当前作用域的 HashMap 压入栈中，作用域结束的时候就把栈顶的 Map 弹出

### 4） 函数调用中参数个数不匹配异常

使用 ArrayList 记录函数调用的时候所输入的参数，然后对比函数声明中 heading 的参数列表，如果大小不一致那就出现了参数个数不一致的错误。

### 5） 异常抛出时记录问题的位置

由于不适用 JavaCUP 的资源，因此需要自己实现一个 Token 类，在这个类里存储了当前 Token 出现的行号以及列号，抛出异常的时候调用 getter 获取到当前 Token 的位置即可，但是好像行是从 0 开始的，列的位置总是有点问题 o(╯□╰)o

### 6） 表达式的处理

经过较长时间的思考，感觉在语法分析过程中把所有的表达式的值都求出来不是一件简单的事情，首先嵌套数组就不知道怎么解决。在第四部分，我是用原表达式字符串来代表一个表达式，这样得到的效果更好并且更加符合程序的需要。接下来修改第三部分的时候也相应表达式改变成字符串输出。

## 2. 有待后续实现的功能

### 1） 数组类型嵌套处理

依旧这个问题，对于处理类型是否匹配方面是没事的，因为数组的每一个元素的类型都一样，只需要存储这个类型就可以判断了，但是如果多维数组的话应该需要增加多一个变量记录数组定义迭代了多少层，这个比较麻烦就没有实现了 o(╯□╰)o

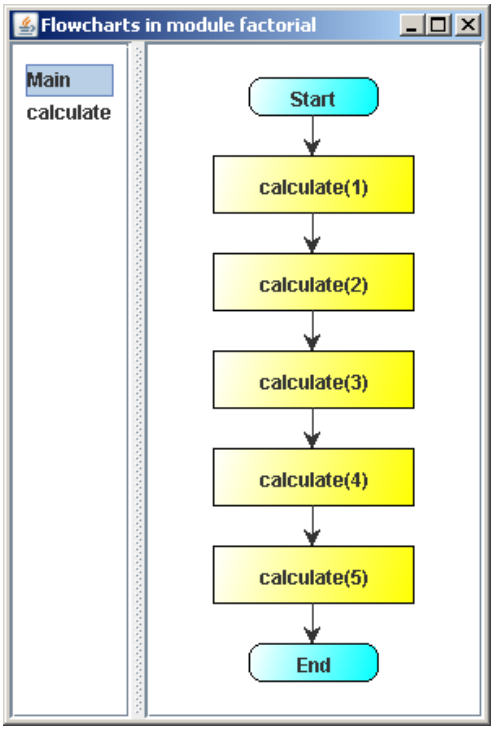
# 四、 实验成果展示

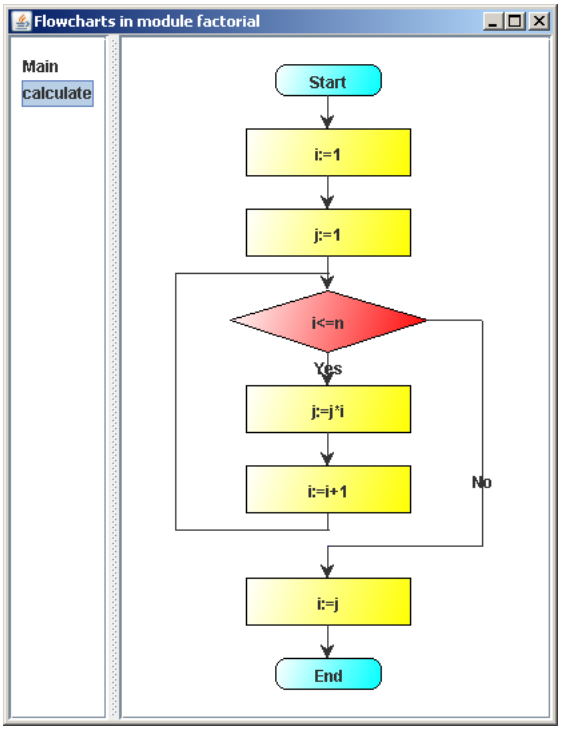
## 1. 目录结构与文件说明

```
➜  ex4  tree
.
├── bin……编译输出字节码文件夹
│   …
│   └── Variable.class
├── doc……JavaDOC 工具生成的文档
│   …
│   └── Variable.html
├── lib……实验中需要到的类库
│   ├── callgraph.jar
│   ├── flowchart.jar
│   └── jgraph.jar
├── src……所有源代码存放的文件夹
│   ├── exceptions……定义的异常类
│   │   …
│   │   └── TypeMismatchedException.java
│   ├── IdType.java……定义 Oberon-0 语言中出现的类型
│   ├── OberonMain.java……运行语法分析生成流程图的主函数
│   ├── OberonParser.java……递归下降预测分析程序
│   ├── OberonScanner.java……修改 JFlex 生成的词法分析程序
│   ├── Symbol.java……Oberon-0 语言中出现的符号
│   ├── Token.java……词法分析返回的 Token 类
│   ├── Variable.java……记录声明的变量的类型以及值
│   └── testcases……测试样例存放的文件夹
│       ├── GivenException……实验软装置中提供的测试样例
│       │   …
│       │   └── SyntacticErrors
│       ├── LexicalException……自己编写的词法错误变异程序
│       │   ├── Sample.001
│       │   …
│       │   └── Sample.010
│       ├── SemanticException……自己编写的语义错误变异程序
│       │   ├── Sample.001
│       │   …
│       │   └── Sample.008
│       └── SyntacticException……自己编写的语法错误变异程序
│           ├── Sample.001
│           …
│           └── Sample.010
│       └── Sample.obr……正确的 Oberon-0 样例程序
├── readme.txt……说明文件
├── test.txt……运行 test.bat 输出
├── run.bat……程序运行脚本
├── doc.bat……生成 JavaDOC 脚本
├── build.bat……项目构建脚本
├── clean.bat……项目清理脚本
├── test.bat……运行异常测试程序运行脚本
├── test_lexical_error.bat……测试词法错误变异程序脚本
├── test_semantic_error.bat……测试语义错误变异程序脚本
├── test_syntactic_error.bat……测试语法错误变异程序脚本
├── scheme.pdf……实验报告 PDF 格式
├── scheme.doc……实验报告 DOC 格式
└── trans_mode.txt……适用自顶向下分析的翻译模式
```

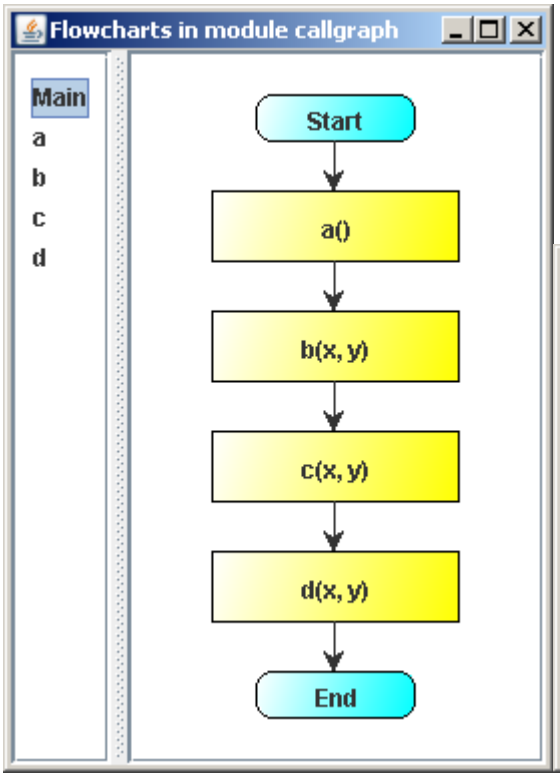
# 2. 程序运行截图：

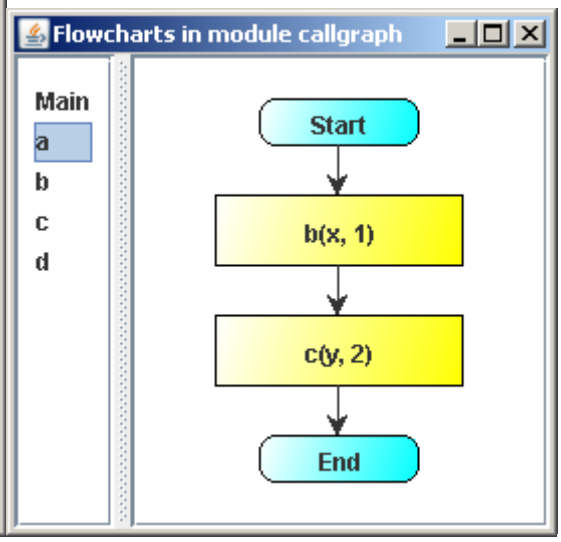
- 运行 Run.bat 脚本分析 factorial.obr 源程序结果：



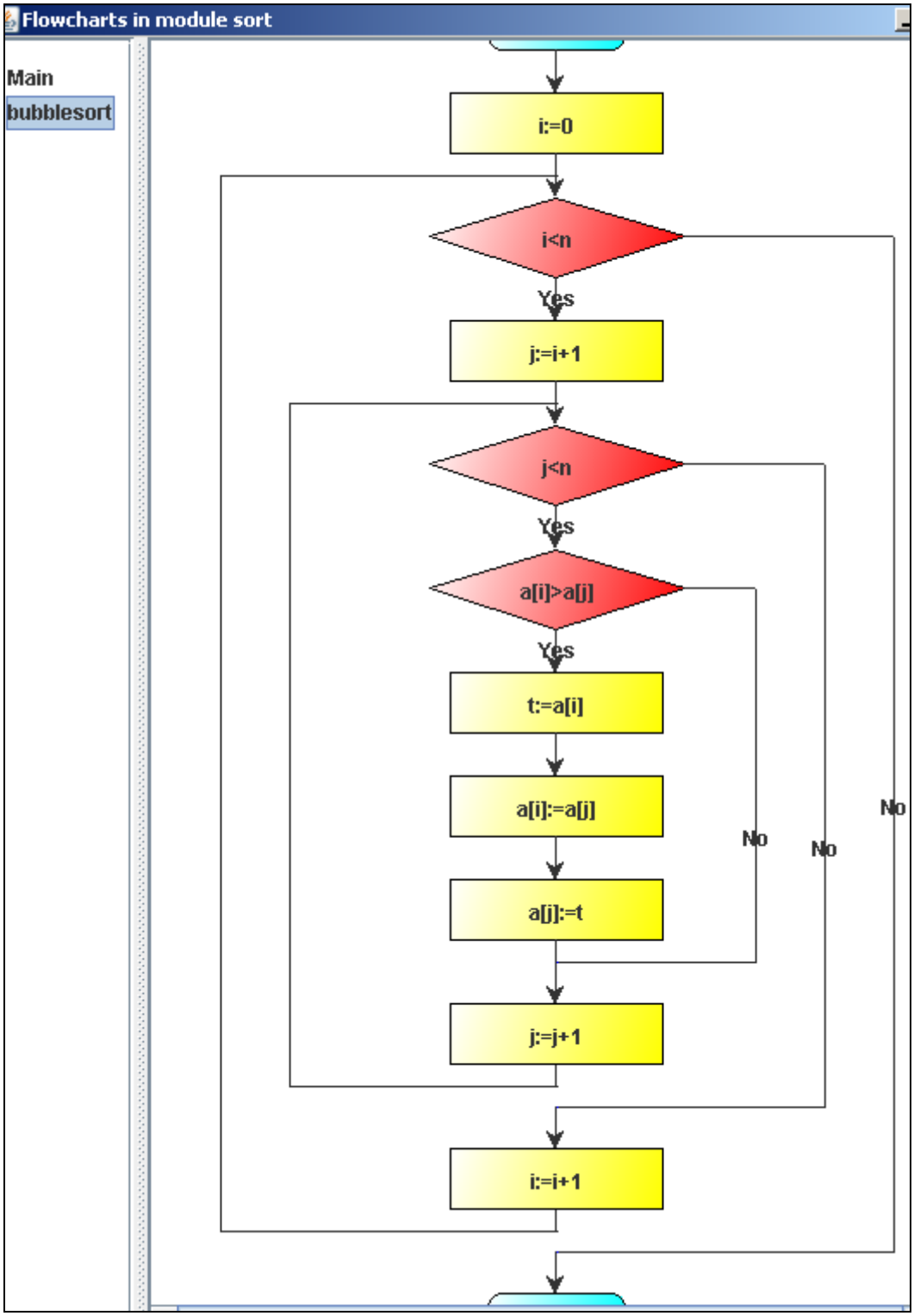




- 运行 Run.bat 脚本分析 callgraph.obr 源程序结果(部分)：



- 运行 `Run.bat` 脚本分析 `sort.obr` 源程序结果(部分)：




- 运行我编写的 `Oberon-0` 源程序 `Sample.obr`……太大了就不贴出来了……

- 测试变异程序（测试源程序见 testcases 以及 testcases/GivenException）：

  由于测试样例比较多，test.bat 运行具体详细实验结果见<test.txt>

  1. 词法异常

```
Test-001
Begin parsering...
Module name is test

Illegal Symbols have been found.
The error position is Line 3 Column 0
Test-002
Begin parsering...
Module name is test

The integer is illegal.
The error position is Line 3 Column 17
Test-003
Begin parsering...
Module name is test

The integer's length is too long.
The error position is Line 3 Column 17
Test-004
Begin parsering...
Module name is test

The integer's length is too long.
The error position is Line 3 Column 17
```

  2. 语法异常

```
Test-001
Begin parsering...
Module name is factorial

Type has been mismatched.
The error position is Line 8 Column 23
Test-002
Begin parsering...
Module name is factorial
Const variable must be integer

Type has been mismatched.
The error position is Line 3 Column 22
Test-003
Begin parsering...
Module name is factorial
Missing end of module

Syntax error has been found.
The error position is Line 29 Column 11
Test-004
Begin parsering...
Module name is factorial
Unexpected ";"

Syntax error has been found.
The error position is Line 29 Column 12
```

3. 语义异常

```
Test-001
Begin parsering...
Module name is factorial

Identifier is Conflict.You have define the id before.
The error position is Line 4 Column 15
Test-002
Begin parsering...
Module name is factorial

The number of parameters is incorrect.
The error position is Line 19 Column 17
Test-003
Begin parsering...
Module name is factorial

Identifier is Conflict.You have define the id before.
The error position is Line 18 Column 4
Test-004
Begin parsering...
Module name is factorial

Identifier is Conflict.You have define the id before.
The error position is Line 4 Column 25
```

# 五、 实验心得

完成了这次实验后对递归下降的语法分析技术又有了新的认识，在实现语法分析的过程中也锻炼了我的代码实现能力。完成第四部分的时候在第三部分的基础上有重新思考了一些异常的处理以及程序的结构，较好地解决了一些第三部分没有解决的问题，并且这个解决办法还能对第三部分的 JavaCUP 生成文件进行修改，尽量努力把以前有待解决的问题解决了……（期末考试后……）

准备考试了，花了不少复习的时间在完成 Project3 上面，应该来说没有浪费吧，对编译原理有所帮助，剩下时间好好复习希望期末考试能考个好成绩啦~

Thanks\(^o^)/~