



中山大學

SUN YAT-SEN UNIVERSITY

实验：编译器构造实验

姓名：葉珺明

学号：19335253

编译器构造实验

目录

实验：编译器构造实验	
姓名：葉琚明	
学号：19335253	
编译器构造实验	
一 实验目的	
二 实验要求	
三 实验设计与运行结果	
3.1 语言文法	
3.2 一个简单文法的编译器前端的设计与实现	
3.2.1 扫描器设计实现	
3.2.2 语法分析器设计实现	
3.2.2 语法分析器设计实现	
3.3 一个简单文法的编译器的设计与实现	
四 实验心得	
五 参考文献	

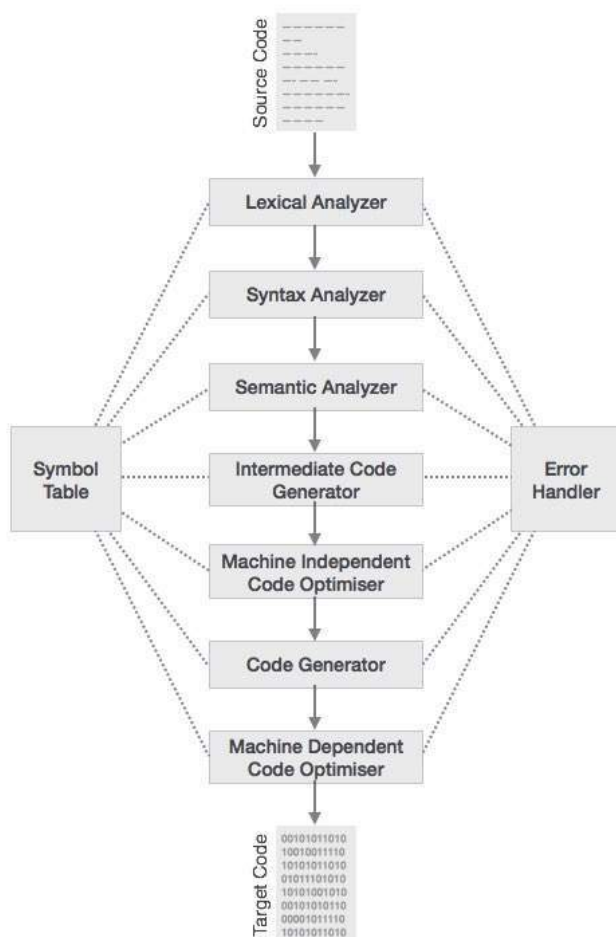
所有文件源码地址：[click here](#)

一 实验目的

- 通过编译器相关子系统的设计，进一步加深对编译器构造的理解
- 培养学生独立分析问题、解决问题的能力，以及系统软件设计的能力
- 提高程序设计能力、程序调试能力

二 实验要求

- 一个简单文法的编译器的设计与实现
 - 定义一个简单程序设计语言文法（包括变量说明语句、算术运算表达式、赋值语句；扩展包括逻辑运算表达式、If语句、While语句等）；
 - 扫描器设计实现；
 - 符号表系统的设计实现；
 - 语法分析器设计实现；
 - 中间代码设计；
 - 中间代码生成器设计实现。
- 一个简单文法的编译器后段的设计与实现
 - 中间代码的优化设计与实现（鼓励）；
 - 目标代码的生成（使用汇编语言描述，指令集自选）；
 - 目标代码的成功运行
- 编译器构造：



三 实验设计与运行结果

3.1 语言文法

- 文法定义:

$\langle \text{程序} \rangle \rightarrow \text{program} \langle \text{标识符} \rangle \langle \text{分程序} \rangle$
 $\langle \text{分程序} \rangle \rightarrow \langle \text{变量说明} \rangle \langle \text{复合语句} \rangle$
 $\langle \text{变量说明} \rangle \rightarrow \text{var} \langle \text{标识符表} \rangle : \langle \text{类型} \rangle ;$
 $\langle \text{标识符表} \rangle \rightarrow \langle \text{标识符} \rangle , \langle \text{标识符表} \rangle \mid \langle \text{标识符} \rangle$
 $\langle \text{复合语句} \rangle \rightarrow \text{begin} \langle \text{语句表} \rangle \text{end}$
 $\langle \text{语句表} \rangle \rightarrow \langle \text{赋值语句} \rangle ; \langle \text{语句表} \rangle \mid \langle \text{赋值语句} \rangle$
 $\langle \text{赋值语句} \rangle \rightarrow \langle \text{标识符} \rangle := \langle \text{算术表达式} \rangle$
 $\langle \text{算术表达式} \rangle \rightarrow \langle \text{算术表达式} \rangle \omega 0 \langle \text{项} \rangle \mid \langle \text{项} \rangle$
 $\langle \text{项} \rangle \rightarrow \langle \text{项} \rangle \omega 1 \langle \text{因子} \rangle \mid \langle \text{因子} \rangle$
 $\langle \text{因子} \rangle \rightarrow \langle \text{算术量} \rangle \mid (\langle \text{算术表达式} \rangle)$
 $\langle \text{算术量} \rangle \rightarrow \langle \text{标识符} \rangle \mid \langle \text{常数} \rangle$
 $\langle \text{类型} \rangle \rightarrow \text{integer} \mid \text{real} \mid \text{char}$

- 按照文法定义消除左递归:

$S \rightarrow \langle \text{PROGRAM} \rangle \langle \text{ID} \rangle A$ ①
 $A \rightarrow BC$ ②
 $B \rightarrow \langle \text{VAR} \rangle D : E;$ ③
 $D \rightarrow \langle \text{ID} \rangle D'$ ④
 $D' \rightarrow , D$ ⑤
 $D' \rightarrow \epsilon$ ⑥
 $E \rightarrow \langle \text{INTEGER} \rangle$ ⑦
 $E \rightarrow \langle \text{REAL} \rangle$ ⑧
 $E \rightarrow \langle \text{CHAR} \rangle$ ⑨
 $C \rightarrow \langle \text{BEGIN} \rangle F \langle \text{END} \rangle$ ⑩
 $F \rightarrow GF'$ ⑪
 $F' \rightarrow ; G$ ⑫
 $F' \rightarrow \epsilon$ ⑬
 $G \rightarrow \langle \text{ID} \rangle := H$ ⑭
 $H \rightarrow IH'$ ⑮
 $H' \rightarrow +IH'$ ⑯
 $H' \rightarrow -IH'$ ⑰
 $H' \rightarrow \epsilon$ ⑱
 $I \rightarrow JI'$ ⑲
 $I' \rightarrow *JI'$ ⑳
 $I' \rightarrow /JI'$ ㉑
 $I' \rightarrow \epsilon$ ㉒
 $J \rightarrow \langle \text{ID} \rangle$ ㉓
 $J \rightarrow \langle \text{NUMBER} \rangle$ ㉔
 $J \rightarrow (H)$ ㉕

- 由自动机实现的文法:

- 上述文法的正则表达式:

3.2 一个简单文法的编译器前端的设计与实现

- 词法阶段要求输出: **tokens**, 关键字**K**表和界符**P**表, 符号表**I**表, 常数**C**表
- 借助**LEX**工具
 - definition** 部分: 定义关键字编号和自动机部分

- **rules** 部分：匹配规则并执行相应操作

- **user code** 部分：在 **user code** 部分，存放用户需要执行的 C 语言代码，主要编写 **main** 函数，读取 C 语言源文件，然后对该文件进行词法分析并输出 **tokens.txt** 文件。该部分所有代码会加入到 **lex.yy.c** 文件的末尾。
如下代码，其中，函数 **yywrap()** 用于判断是否已经扫描完所有的文件，当所有的文件被扫描完时，返回

1, 词法分析器停止分析; 函数 `textout()` 用于储存词法分析器输出的每个单词及其种类枚举值。

```
1 int yywrap (){}
2 void textout(int r){}
3 int main(){}
```

- 单独编译步骤

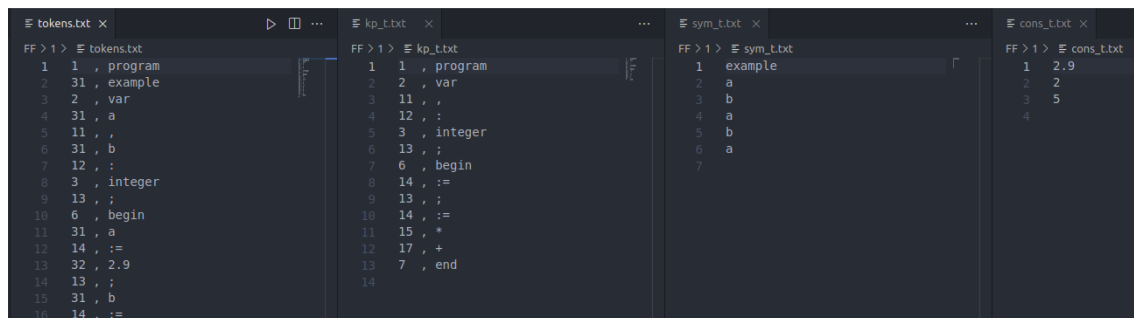
```
1 $ flex -o ana.yy.c ana.l
2 $ gcc -o ana ana.yy.c -lfl
3 $ ./ana demo.txt tokens.txt kp_t.txt sym_t.txt cons_t.txt
```

- 词法分析器实验结果

测试文件: `demo.txt`

```
1 program example
2 var a,b:integer;
3 begin
4 a:=2.9;
5 b:=2*5+a
6 end
```

输出tokens文件, 关键字K表和界符P表, 符号表I表, 常数C表:



tokens.txt	kp_t.txt	sym_t.txt	cons_t.txt
1 1, program	1 1, program	1 example	1 2.9
2 31, example	2 2, var	2 a	2 2
3 2, var	3 11, ,	3 b	3 5
4 31, a	4 12, ;	4 a	4
5 11, ,	5 3, integer	5 b	
6 31, b	6 13, ;	6 a	
7 12, ;	7 6, begin		
8 3, integer	8 14, :=		
9 13, ;	9 13, ;		
10 6, begin	10 14, :=		
11 31, a	11 15, *		
12 14, :=	12 17, +		
13 32, 2.9	13 7, end		
14 13, ;			
15 31, b			
16 14, :=			

其中tokens文件和KP表中的第一列记录的是关键字的编号, 第二列记录对应字符
输出文件均能保存对应部分的内容。

3.2.2 语法分析器设计实现

- 借助JavaCC工具, 根据上述消除左递归后的文法进行语法分析
- 借助JavaCC工具: 递归下降子程序
 - Options 块:

```
1 options {
2     STATIC = false;
3 }
```

- Class 声明块:

```
1 PARSER_BEGIN(IsAriExp)
2     import java.io.PrintStream ;
3     class IsAriExp {
4         ...
5     }
6 PARSER_END(IsAriExp)
```

- 词法和语法解析：

```

1  TOKEN: { < PROGRAM : "program" > }
2  TOKEN: { < VAR : "var" > }
3  ...
4
5  //Recursive subroutine
6  void Start(PrintStream printStream) throws NumberFormatException :{}
7  {
8      (
9          S()
10         { printStream.println( "TRUE" ) ; }
11     ) *
12     <EOF>
13 }
14 ...

```

- 单独编译步骤：

```

1  $ javacc ArithExp.jj
2  $ javac *.java
3  $ java IsAriExp < demo.txt

```

- 语法分析器实验结果

测试文件：demo.txt，输出对应的正确或错误的反馈：

当测试文件语法正确时仅输出： **TRUE**

当测试文件语法错误（修改demo.txt文件第5行为 $*b:=2*a$ ）时，会输出错误位置与正确做法：

```

yip@yip-HBL-WX9:~/Compilers/FInal/Code/FF/2$ java IsAriExp < demo.txt
TRUE
yip@yip-HBL-WX9:~/Compilers/FInal/Code/FF/2$ java IsAriExp < demo.txt
Exception in thread "main" ParseException: Encountered " "+" "+" at line 4, column 10.
Was expecting one of:
    "(" ...
    <NUMBER> ...
    <ID> ...

```

3.2.2 语法分析器设计实现

- 根据消除左递归的文法构造LL(1)预测表

	P	I	V	II	R	C	B	E	,	;	:	+	-	*	/	()	#
S	①																	
A			②															
B			③															
D		④																
D'								⑤		⑥								
E				⑦	⑧	⑨												
C							⑩											
F		⑪																
F'								⑬		⑫								
G		⑭																
H		⑮																
H'												⑯	⑰				⑱	
I		⑲															⑲	
I'												⑳	㉑	㉒	㉓			
J		㉔															㉕	

- 语法栈 SYN，语义栈 SEM，四元式区 QT
- 具体实现：
 - 当产生式(逆序)压栈时，动作符号也不例外；
 - 当动作符号位于栈顶时，执行之；
- 实现代码：
 - 对于算数表达式中四则运算，赋值操作，分程序等需要中间表达形式四元式记录
 - 对于变量定义，需要通过符号表来进行记录，以及各个变量地址的更新
 - 需要记录活动映像，即代码过程中变量的使用 and 更新
 - 常数表记录常数数值

```

1  if syn.peek() == 'geq+':
2      syn.pop()
3      s1 = sem.peek()
4      sem.pop()
5      s2 = sem.peek()
6      sem.pop()
7      s3 = 't'+str(j)
8      sem.push(s3)
9      j = j+1
10     s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
11     qt.push(s4)
12     continue
13     ...
14
15 if oper == 1:
16     status.push(1)
17     syn.pop()
18     syn.push('A')
19     # syn.push('gep_p')

```



```

20     syn.push('J')
21     syn.push('program')
22
23     elif oper == 2:
24         status.push(2)
25         syn.pop()
26         syn.push('C')
27         syn.push('B')
28         ...
29
30

```

- 单独编译步骤:

```
1 python sen.py
```

- 实验结果

对于定义的文法，能够部分准确地判断语义信息并进行记录，在下面截图中第一行代表需要分析的tokens，SYN列为语法栈，STATUS记录状态跳转。

```

vip@vip-HBL-WX9:~/Compilers/Final/Code/FF/3s python sen.py
['program', 'example', 'var', 'a', 'b', ':', 'integer', ';', 'begin', 'a', ':=', '2.9', ';', 'b', ':=', '2', '*', '5', '+', 'a', 'end', '#']
SYN      TOKEN      SEM      STATUS
['S']    program      []      0
['A', 'J'] program      []      1
['A', 'J'] example      []      1
['A', 'push', 'example'] example      []      23
['A', 'push'] var      []      23
['A'] var      ['example'] 23
['C', 'B'] var      ['example'] 2
['C', ':', 'E', ':', 'D', 'var'] var      ['example'] 3
['C', ':', 'E', ':', 'D'] a      ['example'] 3
['C', ':', 'E', ':', 'D1', 'J'] a      ['example'] 4
['C', ':', 'E', ':', 'D1', 'push', 'a'] a      ['example'] 23
['C', ':', 'E', ':', 'D1', 'push'] ,      ['example'] 23
['C', ':', 'E', ':', 'D1'] ,      ['example', 'a'] 23
['C', ':', 'E', ':', 'D', ':'] ,      ['example', 'a'] 5
['C', ':', 'E', ':', 'D'] b      ['example', 'a'] 5

```

对于单独的算数表达式，能够正确判断语义信息并输出四元式：

```

['123', '/', '(', '(', 'a', '*', 'b', ')', '-', '5', '#']
SYN      TOKEN      SEM      STATUS
['E']    123      []      0
['E1', 'T1'] 123      []      1
['E1', 'T1', 'F'] 123      []      5
['E1', 'T1', 'push', '123'] 123      []      9
['E1', 'T1', 'push'] /      []      9
['E1', 'T1'] /      ['123'] 9
['E1', 'T1', 'geq/', 'F', '/'] /      ['123'] 7
['E1', 'T1', 'geq/', 'F'] (      ['123'] 7
['E1', 'T1', 'geq/', ')', 'E', '('] (      ['123'] 10
['E1', 'T1', 'geq/', ')', 'E'] (      ['123'] 10
['E1', 'T1', 'geq/', ')', 'E1', 'T1'] (      ['123'] 1
['E1', 'T1', 'geq/', ')', 'E1', 'T1', 'F'] (      ['123'] 5
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E', '('] (      ['123'] 10
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E'] a      ['123'] 10
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1'] a      ['123'] 1
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'F'] a      ['123'] 5
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push', 'a'] a      ['123'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push'] *      ['123'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1'] *      ['123', 'a'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'F', '*'] *      ['123', 'a'] 6
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'F'] b      ['123', 'a'] 6
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'push', 'b'] b      ['123', 'a'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'push'] )      ['123', 'a'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*'] )      ['123', 'a', 'b'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1'] )      ['123', 't0'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1'] )      ['123', 't0'] 8
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')'] )      ['123', 't0'] 4
['E1', 'T1', 'geq/', ')', 'E1', 'T1'] -      ['123', 't0'] 4
['E1', 'T1', 'geq/', ')', 'E1'] -      ['123', 't0'] 8
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T', '-'] -      ['123', 't0'] 3
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T'] 5      ['123', 't0'] 3
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'F'] 5      ['123', 't0'] 5
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'push', '5'] 5      ['123', 't0'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'push'] #      ['123', 't0'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1'] #      ['123', 't0', '5'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-'] #      ['123', 't0', '5'] 8
['E1', 'T1', 'geq/', ')', 'E1'] #      ['123', 't1'] 8
['E1', 'T1', 'geq/', ')'] #      ['123', 't1'] 4
INCORRECR EXPRESSION!!
The result is: ['(*,a,b,t0)', '(-,t0,5,t1)']

```

- 分析：该算术表达式式不正确的，其圆括号没有匹配上，程序最终也输出“INCORRECT EXPRESSION”提示该表达式是不正确的，前面 $(a * b)$ ，与 $(a * b) - 5$ 的部分是正确输出其四

元式的表达，该语义分析在翻译遇到错误时能够正确判断并退出。

考虑是代码逻辑还是存在问题，主要是翻译文法部分，还没有理清楚该如何执行相关的语义操作。

3.3 一个简单文法的编译器的设计与实现

待实现。

四 实验心得

本次实验是自主设计一个编译器，按照老师给定的简单Pascal的文法，从易到难实现。

利用先前所实现的词法分析器、语法分析器和语义分析Pascal语言进行编译处理，本次实验难度较大，就目前的实现的情况是十分不理想的。

在实验的过程中对文法的理解不够透彻，尤其是翻译部分，从算数表达式到整个语言文法的转变这一过程花费了大量的时间。

现有的实现是基于前面所设计的算数表达式的前端实现部分，用的工具也比较多，包括了LEX和JavaCC工具，使用的语言也比较杂，有C语言和Python，导致最后混合编译难度也比较大。后续应采用CMake混合编译会比较方便或者改写Python实现的代码。

五 参考文献

1. 陈火旺.《程序设计语言编译原理》（第3版）.北京：国防工业出版社.2000.
2. 美 Alfred V.Aho Ravi Sethi Jeffrey D. Ullman著.李建中，姜守旭译.《编译原理》.北京：机械工业出版社.2003.
3. <https://max.book118.com/html/2018/0514/166240038.shtm>