

北京交通大学
BEIJING JIAOTONG UNIVERSITY

《编译原理》 实验报告

实验名称:	编译器前端实现
学 号:	
姓 名:	
学 院:	计算机与信息技术学院
日 期:	2022 年 12 月 09 日

目录

1. 实验目的	3
2. 实验要求	3
3. 程序实现	3
3.1. 相关环境介绍	3
3.2. 主要数据结构	3
3.3. 程序结构描述	3
3.3.1. 设计方法	3
3.3.2. 函数定义	4
4. 程序测试	6
5. 实验汇总	12
5.1. 技术难点及解决方案	12
5.2. 实验感想和经验总结	12

1. 实验目的

编译器前端的组成，理解编译程序的整体概念，语法制导翻译的基本概念和实现方法。将词法分析程序设计原理与实现（专题 1）和基于 SLR(1)分析法的语法制导翻译及中间代码生成程序设计原理与实现（专题 5）形成一个程序，即程序的输入为符号串源程序，输出为中间代码四元式序列。

2. 实验要求

- (1) 词法分析的输入为符号串，词法分析的输出为二元式形式的中间文件；
- (2) 语法制导翻译读取二元式形式的中间文件并生成中间代码四元式形式的中间文件；
- (3) 设计两个测试用例（尽可能完备），并给出程序执行结果。

3. 程序实现

3.1. 相关环境介绍

操作系统：window 10 21H2

开发环境：Clion-2022.2.1-Windows

编译器：mwing-10.0

3.2. 主要数据结构

所有重要数据结构同实验 5。

3.3. 程序结构描述

3.3.1. 设计方法

考虑到程序的通用性，程序的输入流全部都是 txt 文件。首先读入 grammar 文件，其中包括文法和 V_n 以及 V_t 集的定义。之后，在程序中创建 V_n 集合 V_t 集，然后求的 first 和 follow 集。在这基础上构建 DFA 和分析表程序。构建成功后，按格式化输出，这样可以方便手动验证正确性。

接下来是和词法分析程序的联动，本程序可以直接调用词法分析程序，因此测试样例直接准备源代码待分析即可。但其实在实验 2 开始，程序的设计理念就是完整前端，并且所有成熟模型一直沿用。

对于 SLR 分析法，本人自信已经完成的十分健壮，可以直接修改 grammar 中的文件更改文法，完成不同 SLR 文法的语法判断；但同时，对于中间文法的生成，本程序只实现了赋值文法的生成，为了不影响其他非赋值文法的语法分析，我们加入了 mode 选项，可以关闭中间代码的生成，符合开闭原则。

本文大量引入面向对象编程思想、相应设计模式思想和单元测试思想，在前几次实验编码的基础上，大量进行函数封装，以类为中心，尽量使用类成员函数完成编写。

当然，语法分析部分的输入当然还是词法分析的结果，也就是二元式，同时本程序为了报错的方便，在二元式的基础上加入了行号 line，变成了三元式。其中的相应 config 配置依赖文件参考实验 1，包括各种文件依赖。

对于中间代码的输出，首先是逆波兰式，后边才是每一个四元式计算，临时变量采用 T 加上下标的形式。

程序的依赖文件和待分析的字符创文件，在 config 文件中进行更改。对于词法分析文件的依赖见实验 1 技术文档。

与实验 1 词法分析的重要联通，采用的是系统调用的方式。也就是说，本前端分为两个 exe 可执行文件，本前端 exe 调用了词法分析器程序。其中核心的就是 config_init 函数。在下一节给出。

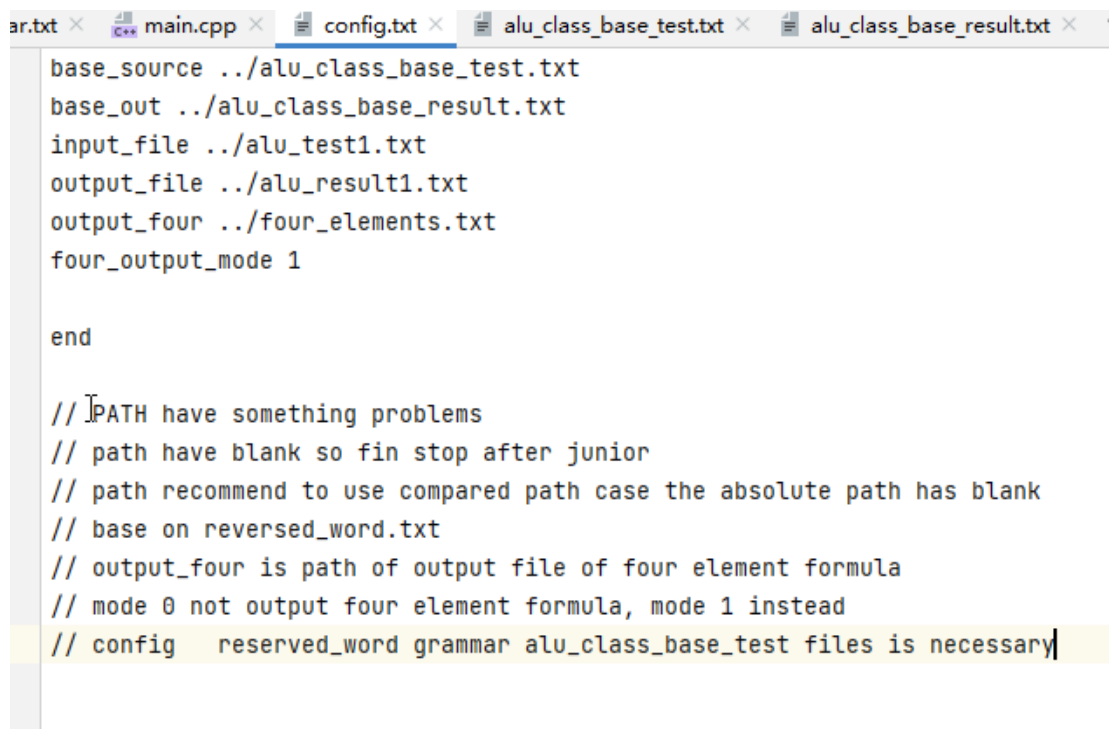
3.3.2. 函数定义

create_Vn_Vt_grammar(grammar_path); 创建 Vn Vt 集和文法表。
 create_first(); 创建 firs 集。
 create_last(); 创建 lastVt 集。
 create_analysis_table(); 创建分析表
 analyzer_stack(need_to_analysis); 分析栈入口，即分析驱动程序。

int correct_prom() 语句符合文法提示。
 int error_prom() 语句不合文法提示。
 DFA_state_show() DFA 输出打印函数。
 Analysis_table_show() 分析表输出打印函数。
 还有一众辅助函数，不表。

Config_init 函数过程如下。

打开文本文件 config.txt，进行读操作。依次匹配读入 base_out、input_file、output_file、output_four、four_output_mode 文件路径。指定词法程序的文件依赖，好为词法中的相应符号指定号编号。



```

base_source ../alu_class_base_test.txt
base_out ../alu_class_base_result.txt
input_file ../alu_test1.txt
output_file ../alu_result1.txt
output_four ../four_elements.txt
four_output_mode 1

end

// PATH have something problems
// path have blank so fin stop after junior
// path recommend to use compared path case the absolute path has blank
// base on reversed_word.txt
// output_four is path of output file of four element formula
// mode 0 not output four element formula, mode 1 instead
// config reserved_word grammar alu_class_base_test files is necessary

```

图表 3-1 config 文件定义

由 system 函数调用词法分析函数，进行词法分析。
最后返回词法分析的结果文件路径等。

```

01: ConfigResult config_init(string &config_path){
02:     ifstream fin;
03:     fin.open(config_path);
04:     if(!fin){
05:         cerr << "config file open error\n";
06:         return {};
07:     }
08:     string tem_string;
09:     string base_source;
10:     string base_out;
11:     string input_file;
12:     string output_file;
13:     ConfigResult result;
14:
15:     while(fin >> tem_string){
16:         if(tem_string == "base_source"){
17:             fin >> tem_string;
18:             base_source = tem_string;
19:         }
20:         else if(tem_string == "base_out"){
21:             fin >> tem_string;

```

```
22:         base_out = tem_string;
23:     }
24:     else if(tem_string == "input_file"){
25:         fin >> tem_string;
26:         input_file = tem_string;
27:     }
28:     else if(tem_string == "output_file"){
29:         fin >> tem_string;
30:         result.need_handle = tem_string;
31:         output_file = tem_string;
32:     }
33:     else if(tem_string == "output_four"){
34:         fin >> tem_string;
35:         result.output_four = tem_string;
36:     }
37:     else if(tem_string == "four_output_mode"){
38:         fin >> result.mode;
39:     }
40:     else if(tem_string == "end")
41:         break;
42: }
43:
44: string command = "Lexical_Analyzer.exe "+base_source + " " +
base_out;
45: system(command.c_str());
46: command = "Lexical_Analyzer.exe "+input_file + " " + output_file;
47: system(command.c_str());
48:
49: fin.close();
50:// base out is the standard class num for func class_num_init()
51: class_num_init(base_out);
52:// return output_file from lexical analyzer is the file need to
grammar analyzer.
53:// output_file is for alu_anlysis
54: return result;
55:}
```

剩余部分为实验 5 内容，不再表述。

4. 程序测试

Alu_test1.txt

这里一个文件中包含了 4 个测试样例。

```
56:
57:k=a- b * c #
58:223 = 23434+34324+(a*b / c) +(a*b/ (2-2)) #
59:a = ( a+ b -c ) * 32 / a #
60:b=(a+c * b* #
```

这是本程序的最初输入，由于是从词法分析开始，下边给出中间的结果的输出：

notation	class_num	line	
k	0	1	
=	26	1	
a	0	1	
-	36	1	
b	0	1	
*	39	1	
c	0	1	
#	24	1	
223	1	2	I
=	26	2	
23434	1	2	
+	33	2	
34324	1	2	
+	33	2	
(45	2	
a	0	2	
*	39	2	
b	0	2	
/	42	2	

图表 4-1 test1 中词法分析结果

alu_class_base_result.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

notation	class_num	line
+	33	1
-	36	1
*	39	1
/	42	1
(45	1
)	46	1
#	24	1
=	26	1

图表 4-2 符号类号文件（词法分析程序的输出结果的一部分）

```
base_source ../alu_class_base_test.txt
base_out ../alu_class_base_result.txt
input_file ../alu_test1.txt
output_file ../alu_result1.txt
output_four ../four_elements.txt
four_output_mode 1

end

// PATH have something problems
// path have blank so fin stop after junior
// path recommend to use compared path case the absolute path has blank
// base on reversed_word.txt
// output_four is path of output file of four element formula
// mode 0 not output four element formula, mode 1 instead
// config reserved_word grammar alu_class_base_test files is necessary
```

图表 4-3 config 文件

其中包括了一众依赖文件的路径还有默认待分析文件路径，最下边是书写格式。相比于前几个实验，config 文件多了四元式输出文件和模式选项。


```

Vn E T F V S
Vt + - * / = ( ) i n #

S -> V = E
E -> E + T | E - T | T
T -> T * F | T / F | F
F -> ( E ) | i | n
V -> i

start S
end
// e is epilog
// each line ends with nothing, no blank
// Vt i is identifier num
|

```

图表 4-4 grammar 文件

Grammar 文件主要是 Vn Vt 集还有文法的输入，下方是特殊说明。同时，由于在分析表中有接受操作，所以要特殊指定开始符号。同时，我们对文法进行了拓展，等号右部可以出现数字 n，原来为只有标识符 i。

词法分析程序输出仍然采用了三元组，多了一个行号，方便错误提示，更多的我们直接用了 Keyword 结构，这对结果没有影响。

```

line 1 syntax correct
line 2 has syntax error
line 3 syntax correct
line 4 has syntax error

Process finished with exit code 0

```

图表 4-5 本程序结果，信息提示

通过信息提示可以看出，程序对 4 个测试样例完成的很好，结果正确。等式中，实验指导书原文法标识符位置，可以出现数字常量。这是对文法的拓展。同时为了方便判断程序正确与否，还输出了 DFA 和分析表。

↓	id	0
⌕	S ->	. V = E
⇅	V ->	. i
🖨	next	
🗑	V	1
	i	2
	id	1
	S ->	V . = E
	next	
	=	3
	id	2
	V ->	i .
	next	
	id	3
	E ->	. E + T
	E ->	. E - T

图表 4-6 DFA 结果

DFA 输出的结果包括状态号，产生式情况，还有不同输入对应的下一个状态。

analysis_table is following
0 for move inside, 1 for jump goto, 2 for acc, 3 for guiyue

state_id	input	op
0		
	V	1
	i	0
1		
	=	0
2		
	=	3
3		
	(0
	E	1
	F	1
	T	1
	i	0
	n	0
4		
	#	2
	+	0
	-	0
5		
	#	3
)	3
	*	0
	+	3
	-	3
	/	0

图表 4-7 SLR 分析表

分析表包括状态号，不同输入的操作，分 4 种不同操作，对应操作号。最后一列为相应的补充，如 goto 操作则是转向的状态；规约为使用的产生式。

```

line 1
k a b c * - =

  op  left  right  result
  *   b     c     T0
  -   a     T0    T1
  =   k     T1    T2

line 3
a a b + c - 32 * a / =

  op  left  right  result
  +   a     b     T0
  -   T0    c     T1
  *   T1    32    T2
  /   T2    a     T3
  =   a     T3    T4

```

图表 4-8 四元式的输出

输出包括行号、逆波兰式、四元式。由于是追加写模式打开文件，所以上次的分析结果不会被覆盖。

5. 实验汇总

5.1. 技术难点及解决方案

所有难点实验 2 时已解决。

5.2. 实验感想和经验总结

其实一开始所有的实验的构想都是一个完整的前端。一开始只是为了验证语法分析程序正确性方便。试想，人工当然只能给出一个测试样例的字符串，如果手动运行词法分析程序，然后再把结果作为语法分析程序的输入，这样会严重拖慢单纯的语法分析程序 debug 的进度。于是，一开始，从实验 2 起，所有的语法实验都是一个完整的前端程序，其输入都是字符串，输出为相应要求。实验六的代码甚至是直接从实验 5 拷贝过来的没有任何修改。

其实我们也可以采用拷贝代码的方式，或者是在程序链接上下功夫，又或者是引用词

法分析的头文件方式进行前端代码编写，但是由于实验 2 开始的路径依赖，我们选择了配置好实验 1 词法分析程序的调用规则直接进行系统调用的方式进行，其过程，如果再封装一层就是一个调用函数，十分简单，不需要一次次重新编译。

对于六个实验的总结和心得体会，其实每个分实验文档中写的比较详尽了。现在还记得比较清楚的就是算法配合不同数据结构的时候需要有调整，因为上课教授的主要是基于 fortune 语言的伪代码或者程序框图，有许多 goto 语句需要避免，同时还有许多封装的十分高级几乎近似自然语言的算法表述，这让我在数据结构的设计上着实花了不少功夫。或者说算法和数据结果的关系通过这六个实验表现出来就是相互配合，互相迁就，先设计好了算法思想，然后想程序框图，对算法进行分解，这个过程中有时也要想想具体的实现，于是就联系上了数据结构。反过来数据结果敲定后，算法也要有相应的调整，进行配合。

总的来说，这六个实验收获很多，是对编程能力巨大的锻炼。特别欣慰的是，在编程中更多地尝试了面向对象方法和相应设计模型的加入，在 debug 时还使用了软件工程中学习到的测试方法去找 bebug。最关键的是，在实验中对课堂上教授的对语言分析的处理方法和相应的思想理解的更加深入，如 DFA 等等。

最后感谢在实验过程中帮助过我的老师、同学和相关技术文档博主。特别感谢丁丁老师的指导。