中国矿业大学计算机学院

2019 级本科生实验报告

课程名称 编译技术

报告时间 2021-12-04

学生姓名 王杰永

学　　号 03190886

专　　业 计算机科学与技术

任课教师 刘佰龙

成绩考核

编号	课程教学目标	占比	得分
1	**目标 1：** 掌握编译程序的一般构造原理，包括形式语言基础知识、有限自动机、编译器逻辑功能阶段。（支撑本专业毕业要求 3.3）在掌握基本的算法和硬件架构基础上，理解软硬件资源的管理以及建立在此基础上的各类系统的概念、原理及其在计算机领域的主要体现。	50%	
2	**目标 2：** 从形式语言理论的角度,理解程序设计语言及其与编译程序的联系。编译器各个阶段需要完成的任务及各种分析技术，包括单词构造、语法树、符号表的构造及运行时存储空间的组织等基本方法和主要实现技术。（支撑本专业毕业要求 4.3）针对设计或开发的解决方案，能够基于计算机领域科学原理对其进行分析，并能够通过理论证明、实验仿真或者系统实现等多种科学方法说明其有效性、合理性，并对解决方案的实施质量进行分析，通过信息综合得到合理有效的结论。	50%	
总成绩			
指导教师		评阅日期	

课程名称 编译技术 实验名称 实验一 词法分析

班级 计科19-3 姓名 王杰永 学号 03190886 实验日期 2021.11.19

实验报告要求：

1.实验目的 2.实验内容 3.实验要求与步骤

4.算法分析 5.运行结果 6.实验体会

一、实验目的

1、学会针对DFA转换图实现相应的高级语言源程序。

2、深刻领会状态转换图的含义，逐步理解有限自动机。

3、掌握手工生成词法分析器的方法，了解词法分析器的内部工作原理。

二、实验内容

C语言的编译程序的词法分析部分实现。

从左到右扫描每行该语言源程序的符号，拼成单词，换成统一的内部表示（token）送给语法分析程序。

为了简化程序的编写，有具体的要求如下：

(1)数仅仅是整数。

(2)空白符仅仅是空格、回车符、制表符。

(3)代码是自由格式。

(4)注释应放在花括号之内，并且不允许嵌套

C语言的单词

保留字	特殊符号	其他
if	+	标识符 （一个或更多的字母）
then	-	
else	*	
end	/	
repeat	=	数 （一个或更多的数字）
until	<	
read	{	
write	}	
	;	

除上表以外，拓展实现了部分 C++语言的保留字以及运算符。

三、实验要求

要求实现编译器的以下功能：

(1)按规则拼单词,并转换成二元式形式

(2)删除注释行

(3)删除空白符（空格、回车符、制表符）

(4)显示源程序，在每行的前面加上行号，并且打印出每行包含的记号的二元形式

(5)发现并定位错误。

词法分析进行具体的要求：

(1)记号的二元式形式中种类采用枚举方法定义；其中保留字和特殊字符是每个都一个种类，标示符自己是一类，数字是一类；单词的属性就是表示的字符串值。

(2)词法分析的具体功能实现是一个函数 GetToken()，每次调用都对剩余的字符串分析得到一个单词或记号识别其种类,收集该记号的符号串属性,当识别一个单词完毕,采用返回值的形式返回符号的种类,同时采用程序变量的形式提供当前识别出记号的属性值。

(3)标识符和保留字的词法构成相同，为了更好的实现，把语言的保留字建立一个表格存储，这样可以把保留字的识别放在标示符之后，用识别出的标示符对比该表格，如果存在该表格中则是保留字，否则是一般标识符。

四、算法分析

（1）流读取待处理文件

（2）定义枚举变量 DFASTATE，用于定义存放 DFA 的各种状态；定义保留字集合 _reservedWord，用于定义词法分析程序可以识别的各种保留字；定义整型变量 int _row，用于记录词法分析程序分析至源程序的哪一行

（3） 从待处理文件中按序读取字符

（4） 如果该字符是空格，制表符，换行符或注释内容，则忽略并继续读取下一个

（5） 如果是该字符是字母，则进入判断保留字的子程序 _letterRecognition，判断是否为保留字或变量名

（6） 如果读取字符为数字，则进入判断数字的子程序 _digitRecognition，判断是否是整型数或是浮点数

（7） 以上情况都不是则判断是否为常用的处理符号

（8） 以上情况均不是，则报错。

（9） 否则，继续获取字符串下一个字符，循环执行（4），直至读到待处理文件尾。

部分简略的 DFA 如图 1 所示。

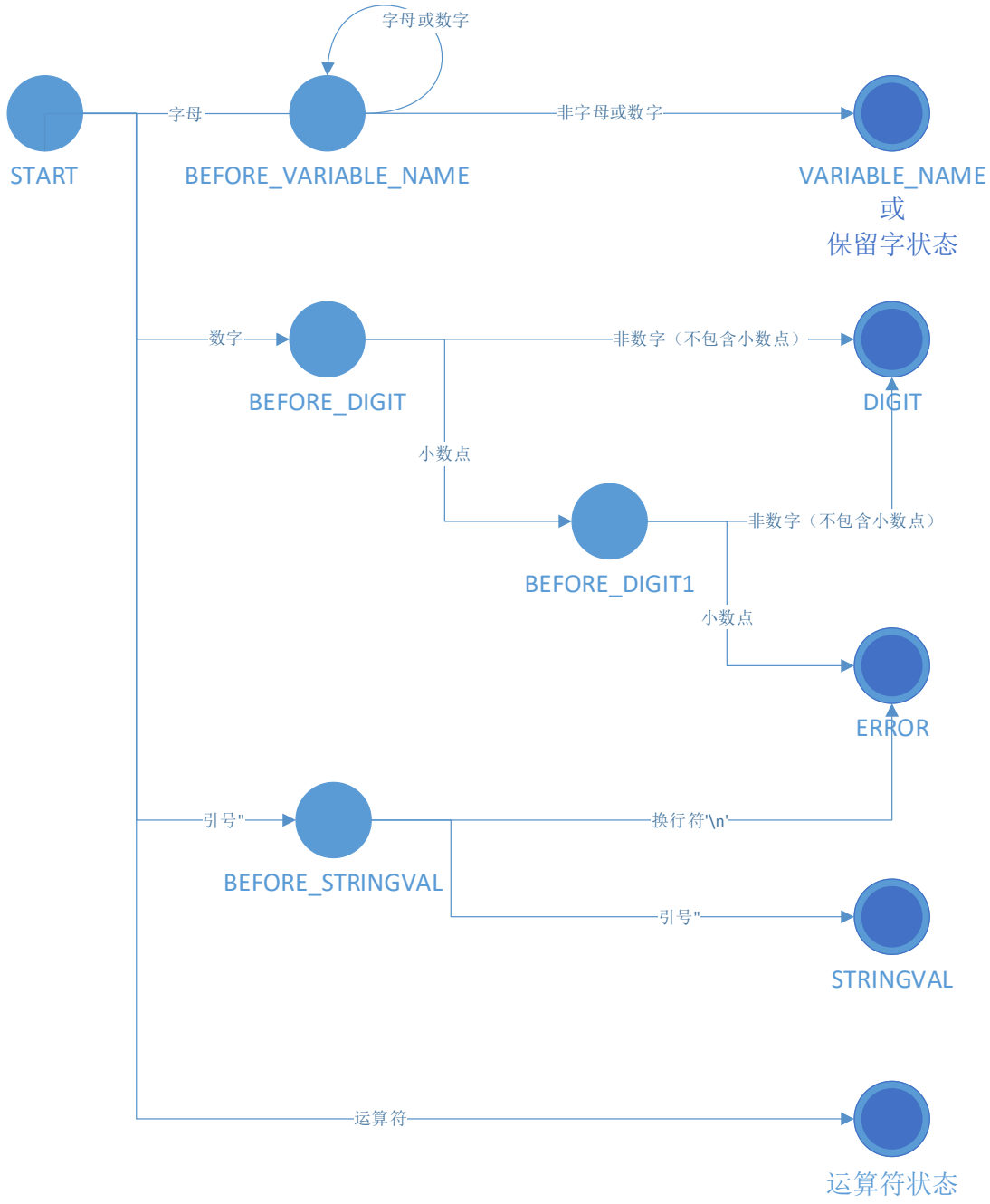

图 1 有限自动机状态转换图

五、实验结果

在程序所在目录下，创建文本文件，内容图 2 所示：

123.txt

compile > 词法分析 > 123.txt

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    //这里是注释
    float a = 1.233;
    string str = "Hello World";
    cout << "hello world" << endl;
    return 0;
}
```

图 2 待进行词法分析的程序

运行词法分析程序，在控制台上打印图 3 所示内容。词法分析结果以二元组为基本单元，二元组第一个元素为当前识别出的单词的类别（DFA 停留的终止态），第二个元素为识别出的单词。

```
[Running] cd "e:\Code WorkSpace\VS CODE\C++ projects\compile\词法分析\" && g++ cf2.0.cpp
行 1    [44, #] [16, include]   [37, <] [20, iostream]  [38, >]
行 2    [44, #] [16, include]   [37, <] [29, vector]    [38, >]
行 3    [17, using] [18, namespace] [19, std]   [39, ;]
行 4    [11, int]   [2, main]   [40, (] [41, )]
行 5    [42, {]
行 7    [13, float] [2, a]  [36, =] [30, 1.233] [39, ;]
行 8    [25, string]    [2, str]    [36, =] [31, "Hello World"] [39, ;]
行 9    [23, cout]  [45, <<]    [31, "hello world"] [45, <<]    [24, endl]  [39, ;]
行 10   [21, return]    [30, 0] [39, ;]
行 11   [43, }]
[Done] exited with code=0 in 0.494 seconds
```

图 3 词法分析结果

我们可以将文本文件中定义的字符串右引号去掉，此时再次运行词法分析程序，程序会将错误精准定位到第 8 行，并识别错误内容：“字符串缺少右引号”。

```
[Running] cd "e:\Code WorkSpace\VS CODE\C++ projects\compile\词法分析\" && g
行 1    [44, #] [16, include]   [37, <] [20, iostream]  [38, >]
行 2    [44, #] [16, include]   [37, <] [29, vector]    [38, >]
行 3    [17, using] [18, namespace] [19, std]   [39, ;]
行 4    [11, int]   [2, main]   [40, (] [41, )]
行 5    [42, {]
行 7    [13, float] [2, a]  [36, =] [30, 1.233] [39, ;]
行 8    [25, string]    [2, str]    [36, =]
错误! 错误位于第[8]行。字符串变量缺少 "

[Done] exited with code=0 in 0.463 seconds
```

图 4 词法分析结果 2

同样，将文本文件第 7 行改为 float a = 1.233.5; 程序同样识别出错误位置，并打印错误信息：

```
[Running] cd "e:\Code WorkSpace\VS CODE\C++ projects\compile\词法分析\" && g
行 1    [44, #] [16, include]   [37, <] [20, iostream]  [38, >]
行 2    [44, #] [16, include]   [37, <] [29, vector]    [38, >]
行 3    [17, using] [18, namespace] [19, std]   [39, ;]
行 4    [11, int]   [2, main]   [40, (] [41, )]
行 5    [42, {]
行 7    [13, float] [2, a]  [36, =]
错误！错误位于第[7]行。数字格式错误
```

图 5 词法分析结果 3

六、实验体会

本次实验，我更好的理解了源程序编译过程中词法分析的地位以及原理。在参考课本上的实验代码后，重新重构了词法分析的源程序代码。拓展了一些功能，同时具有良好的可拓展性以及程序可读性。

对于 DFA 的各种状态，使用 C++语言枚举类型 enum 来定义，大大增强了程序的可读性。收获颇丰。

课程名称 编译技术 实验名称 实验二 递归下降语法分析器的实现

班级 计科 19-3 姓名 王杰永 学号 03190886 实验日期 2021.11.19

实验报告要求：

1. 实验目的 2. 实验内容 3. 实验要求与步骤

4. 算法分析 5. 运行结果 6. 实验体会

一、实验目的

（1）加深对递归下降分析法一种自顶向下的语法分析方法的理解。

（2）根据文法的产生式规则消除左递归，提取公共左因子构造出相应的递归下降分析器。

二、实验内容

根据课堂讲授的形式化算法，编制程序实现递归下降分析器，能对常见的语句进行分析。

三、实验要求

要求实现以下语法的递归下降分析：

```
program  → block
block    → { stmts }
stmts    → stmt stmts | ε
stmt     → id = expr ;
         | if ( bool ) stmt
         | if ( bool) stmt else stmt
         | while (bool) stmt
         | do stmt while (bool )
         | break
         | block
```

```
bool   → expr < expr
       | expr <= expr
       | expr > expr
       | expr >= expr
       | expr
expr   → expr + term
       | expr - term
       | term

term   → term * factor
       | term / factor
       | factor
factor → ( expr ) | id | num
```

图 1 语法分析要求

四、算法分析

(1) 由于要进行递归下降分析，所以要先对文法进行改造。消除左递归并提取公因子。

(2) 使用实验一中的词法分析程序对源文件进行语法分析，将语法分析结果生成的若干二元组单元交给递归下降语法分析器。

(3) 每一个产生式的左侧文法符号对应一个函数，在函数中，根据产生式右部的结构进行不同的代码编写。例如对于最后一个产生式

$$factor->(expr)\mid id\mid num$$

我们可以编写如下函数：

```
void GrammarParser::_factor(){
    if(_flagError)
        return;
    if(_lexicalUnitArray[_unitPointer].state == LEFT){
        cout << "       factor ==> (expr)" << endl;
        _match("(");
        _expr();
        _match(")");
    }
    else if(_lexicalUnitArray[_unitPointer].state == VARIABLE_NAME){
        cout << "       factor ==> id" << endl;
        _match(_lexicalUnitArray[_unitPointer].token);
    }
    else if(_lexicalUnitArray[_unitPointer].state == DIGIT){
        cout << "       factor ==> num" << endl;
        _match(_lexicalUnitArray[_unitPointer].token);
    }
    else{
        _flagError = true;
    }
}
```

图 2 factor 函数

其余函数同理。

(4) 最后可以构造出完整的递归下降语法分析器。

五、实验结果

在程序所在目录下创建源程序文件。如下图所示。

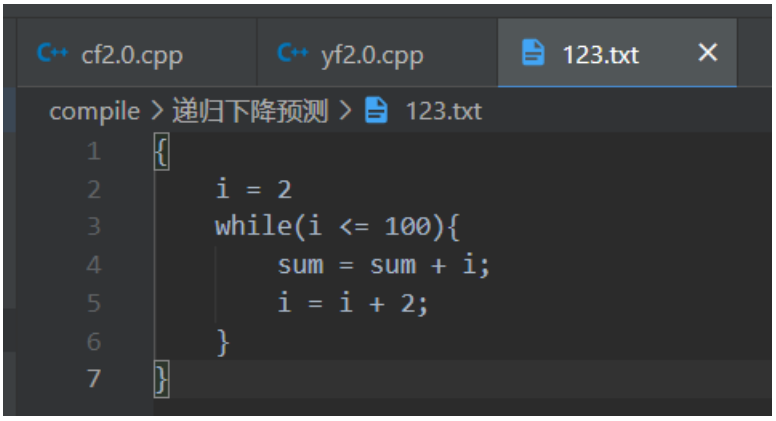

图 3 待语法分析文件

运行递归下降语法分析程序，得到的结果如下图。

```
[Running] cd "e:\Code WorkSpace\VS CODE\C++ projects\compile\递归下降预测\" && g++ yf2.0.cpp -o yf2.0 && "
行 1    [42, {]
行 2    [2, i]  [36, =] [30, 2] [39, ;]
行 3    [15, while] [40, (] [2, i]  [47, <=]    [30, 100]   [41, )] [42, {]
行 4    [2, sum]    [36, =] [2, sum]    [32, +] [2, i]  [39, ;]
行 5    [2, i]  [36, =] [2, i]  [32, +] [30, 2] [39, ;]
行 6    [43, }]
行 7    [43, }]
词法分析结束!
语法分析开始:
      program ==> block
      block ==> {stmts}
      stmts ==> stmt stmts
      stmt ==> id = expr;
      expr ==> term expr1
      term ==> factor term1
      factor ==> num
      term1 ==> null
      expr1 ==> null
      stmts ==> stmt stmts
      stmt ==> while(bool) stmt
      expr ==> term expr1
      term ==> factor term1
      factor ==> id
      term1 ==> null
      expr1 ==> null
      bool ==> expr <= expr
      expr ==> term expr1
      term ==> factor term1
      factor ==> num
      term1 ==> null
      expr1 ==> null
      stmt ==> block
      block ==> {stmts}
      stmts ==> stmt stmts
      stmt ==> id = expr;
      expr ==> term expr1
      term ==> factor term1
      factor ==> id
      term1 ==> null
      expr1 ==> + term expr1
      term ==> factor term1
      factor ==> id
      term1 ==> null
      expr1 ==> null
      stmts ==> stmt stmts
      stmt ==> id = expr;
      expr ==> term expr1
      term ==> factor term1
      factor ==> id
      term1 ==> null
      expr1 ==> + term expr1
      term ==> factor term1
      factor ==> num
      term1 ==> null
      expr1 ==> null
      stmts ==> null
      stmts ==> null
合法语句!
```

图 4 递归下降语法结果

七、实验体会

本次实验让我更深的理解了递归下降语法分析的原理，明白了语法分析的作用，为后面 LR 分析的理解打下了基础。

对于课本上的参考代码，大部分篇幅用来重新实现了一个词法分析，并且还有一部分是用来自己实现了两个栈。我采用实验一语法分析的结果进行递归下降分析，更好理解了词法分析与语法分析的关联所在。

课程名称 编译技术 实验名称 实验三 LR 语法分析器

班级 计科 XX-X 姓名 王杰永 学号 03190886 实验日期 2021.12.3

实验报告要求：

1. 实验目的 2. 实验内容 3. 实验要求与步骤

4. 算法分析 5. 运行结果 6. 实验体会

一、实验目的

（1）掌握有限自动机这一数学模型的结构和理论，并深刻理解下推自动机在 LR 分析法中的应用（即 LR 分析器）。

（2）掌握 LR 分析法的思想，学会特定分析表的构造方法，利用给出的分析表进 行 LR 分析。

二、实验内容

根据课堂讲授的形式化算法，编制程序实现对以下语法进行自底向上语法分析的 LR 分析器，设计分析表，对给出的输入语句进行语法分析，判断是否符合相应的文法要求。

文法如下：

$(1) E' \rightarrow E$

$(2)\ E \rightarrow E + T \mid T$

$(3)\ E \rightarrow T$

$(4)\ T \rightarrow T * F$

$(5)\ T \rightarrow F$

$(6)\ F \rightarrow (E)$

$(7)\ F \rightarrow i$

三、实验要求

要求实现以下功能：

a) 设计分析表和语句的输入；

b) 要实现通用的LR分析思想的源代码；

c) 输出对语句的语法分析判断结果，如果可能给出错误的信息提示。

四、算法分析

（1）构造该文法的LR（1）分析表，并提前写入程序中

（2）将#号添加到栈中以及输入符号串尾

（3）构造状态栈以及符号栈

（4）扫描当前字符，查找LR（1）分析表并执行分析表中的对应操作。

（5）重复执行，直至出现错误或输入串遍历完毕。

五、实验结果

运行LR（1）分析器程序，输入待分析语句 i/(i-i)，可以看出符合语法要求。程序输出结果如下图：

```
PS E:\Code WorkSpace\VS CODE\C++ projects\compile\LR1分析> ./LR2.0.exe
i/(i-i)
步骤                    状态栈                  符号栈                  待分析语句
  1                     0                                               i/(i-i)#
  2                     05                      i                       /(i-i)#
  3                     03                      F                       /(i-i)#
  4                     02                      T                       /(i-i)#
  5                     027                     T/                      (i-i)#
  6                     0274                    T/(                     i-i)#
  7                     02745                   T/(i                    -i)#
  8                     02743                   T/(F                    -i)#
  9                     02742                   T/(T                    -i)#
  10                    02748                   T/(E                    -i)#
  11                    027486                  T/(E-                   i)#
  12                    0274865                 T/(E-i                  )#
  13                    0274863                 T/(E-F                  )#
  14                    0274869                 T/(E-T                  )#
  15                    02748                   T/(E                    )#
  16                    0274811                 T/(E)                   #
  17                    02710                   T/F                     #
  18                    02                      T                       #
  19                    01                      E                       #
语法分析成功!
```

图 1 LR分析结果1

再次输入错误的表达式 i*(i+)，结果如下

```
PS E:\Code WorkSpace\VS CODE\C++ projects\compile\LR1分析> ./LR2.0.exe
i*(i+)
步骤                    状态栈                  符号栈                  待分析语句
  1                     0                                               i*(i+)#
  2                     05                      i                       *(i+)#
  3                     03                      F                       *(i+)#
  4                     02                      T                       *(i+)#
  5                     027                     T*                      (i+)#
  6                     0274                    T*(                     i+)#
  7                     02745                   T*(i                    +)#
  8                     02743                   T*(F                    +)#
  9                     02742                   T*(T                    +)#
  10                    02748                   T*(E                    +)#
  11                    027486                  T*(E+                   )#
error
PS E:\Code WorkSpace\VS CODE\C++ projects\compile\LR1分析>
```

图 2 LR分析结果2

八、实验体会

本次实验让我更好的掌握了LR（1）分析法的原理与应用，在实验要求给定的文法上，拓展了减法与除法运算。同时，每进行一步移进或规约操作程序都会打印状态栈与符号栈的全部信息，利用这种可视化的方式，理解LR分析变得更加容易。