

**实践课程报告**

**题目：**

**课程名称： 编译原理实践**

**专业班级： CS1409**

**学 号： U201414795**

**姓 名： 王卓焱**

**指导教师： 邵志远**

**报告日期： 2016年12月25日**

**计算机科学与技术学院**

**目录**

# 1选题背景

1.1任务

主要是通过对简单编译器的完整实现，加深课程中关键算法的理解，提高学生对系统软件编写的能力。

1.2目标

本次课程实践目标是构造一个高级语言的子集的编译器，目标代码是汇编语言。按照任务书，实现的方案可以有很多种选择。

1.3 源语言定义

本次实验选用的文法是来自实验书的C -- 文法，文法的特点是精简版的C语言。

# 2词法分析和语法分析

本实验的任务是编写一个程序对使用C – 语言书写的代码进行词法和语法分析，并且打印分析结果。实验要求使用词法分析工具GUN Flex 和语法分析工具GUN Bison， 并且使用C语言完成。

2.1单词文法描述

1. **Lex文件中使用的描述正规表达式**

x匹配除换行符以外的任何单个字符

\* 匹配前面表达式的零个或多个拷贝

[]匹配括号中的任意字符的字符类.第一个符号是("^"),表示匹配除括号中的字符以外的任意字符.

^ 作为正则表达式的第一个字符匹配行的开头

$ 作为正则表达式的最后一字符匹配行的结尾

{}指示前面的模式被允许匹配多少次

\ 用于转义字符

+ 匹配前面的正则表达式的一次或多次出现

? 匹配前面的正则表达式的零次或一次出现

| 匹配前面的正则表达式或随后的下面表达式

""引号中的每个字符解释为字面意义

/ 只有在后面跟有指定的正则表达式时才匹配前面的正则表达式

{}将一系列的正则表达式组成一个新的正则表达式

1. **单词文法的定义约定：**

digit [0-9]//即单个数字用digit来表示

number [-]?({digit}+)//定义了int|long型的数字（可以匹配正负号）

letter [a-zA-Z]//匹配了字母A-Z的大写和小写情况

identifier {letter}({digit}|{letter}|\_)\*//根据letter来定义出变量的表达式

blank [\t\x0B\f\r\x20]+//将空格 制表符 换行符等进行匹配（它们地位相同）

notes \{.\*\}//匹配注释

1. **文法构造的规则：**

词法分析器构造的时候，根据的是由高到低的顺序进行规则制定，所谓的由高到低指的是，根据先将注释使用规则匹配掉，然后在剩下的文法中进行关键字匹配，具体的关键字见下表格：

|  |  |  |
| --- | --- | --- |
| 关键字 | 匹配规则 | 关键字的描述 |
| INT | number | 代表一切的整型数字（正负都有） |
| FLOAT | [+-]?({digit}+"."{digit}\*|{digit}\*"."{digit}+) | 代表一切的浮点型数字（正负都有） |
| SEMI | ; | 代表分号 |
| COMMA | , | 代表句号 |
| ASSIGNOP | = | 代表等号 |
| RELOP | >|<|>=|<=|==|!= | 代表各种判断符号 |
| PLUS | + | 四则运算的加号 |
| MINUS | — | 四则运算的减号 |
| STAR | \* | 四则预算的乘号 |
| DIV | / | 四则运算的除号 |
| AND | && | 与运算 |
| OR | || | 或运算 |
| DOT | . | 结构体的点 |
| NOT | ! | 求非 |
| TYPE | int|float | 类型（整型，浮点） |
| LP | ( | 左括号 |
| RP | ) | 右括号 |
| LB | [ | 左中括号 |
| RB | ] | 右中括号 |
| LC | { | 左大括号 |
| RC | } | 右大括号 |
| STRUCT | struct | 结构体标识 |
| RETURN | return | 返回标识 |
| IF | if | 判断语句 |
| ELSE | else | 判断分支 |
| WHILE | while | 循环语句 |

1. **构造的优先级简述：**

构造时候，要根据一定的规则进行匹配，在匹配时候，将int和float分开进行了匹配，然后指定出的优先级从高到底规则如下：

1. 统计换行符

在词法分析阶段最开始统计行数，这个时候统计出来的行数是最准确的，因为文本在最开始时候没有被进行任何修改。统计换行符的个数来确定行数，每一个\n代表一行，在开始时定义一个整型的yycolum=1，每次匹配到一个换行符就累加，可以实现行号的精确计数。

1. 匹配注释

因为注释中可能有任何文字内容，如果这些文字内容是关键字或者数字等的话，会将注释中的内容是被为关键字或者各种符号，导致分析出错。

注释分为两种，一种是//开头的行注释，另一种是/\*注释注释\*/的格式；

第一种注释的匹配方法是直接判断//到行的结尾符号\n,中间可以包含任意字符的※闭包。

第二种注释是匹配/\* 与\*/号，两者之间的所有文字为注释（如果一个/\*后有多个\*/则会有限匹配第一个，在gcc编译器进行常规C语言编译时亦是如此。）

1. 匹配各类型关键字

关键字的匹配理论可以在任何地方进行，但是在实际编写中放到ID（变量符号）之前匹配，可以保证匹配规则的正确。

1. 匹配ID
2. 匹配数字

数分为整数和浮点数，优先匹配浮点数，因为如果先匹配整数的话，会导致浮点数的整数部分和小数部分都被匹配为一个整数，而中间的点被匹配为DOT，其中浮点数的规则为：[+-]?({digit}+"."{digit}\*|{digit}\*"."{digit}+)，整数的匹配规则为：[+-]?{digit}+

1. 匹配各类型符号

将各类型的符号进行匹配，从而得出最后结果

1. 报错处理

如果在上6步之后还存在没有匹配的短语，则该短语不是我们的C--文法能够识别的，从而在上述六步后，匹配到的任何字符都会转入出错处理。

2.2语言文法描述

1. **语法分析的传入参数：**

语法分析的参数是通过词法分析后传入的，所以需要将词法分析中的匹配后行为规则进行进一步的修改，在lexical.l中增加对syntax.tab.h的引用，并且让flex源代码中的规则部分每一条都返回一个相对应的词法单元，如下所示：

%{

#include”syntax.tab.h”

…

}

…

%%

“+” {return PLUS;}

“-” {return MINUS;}

“+24{return ID}”

….

1. **Bison编写时语法规则**

Bison文件的编写和上边词法分析的flex文件的编写格式相同，我们只需要关注语法规则即可

语法规则来自实验参考书附录，主要分为如下六部分：

1. High-level Definitions

这一部分包含了本语言中的高层（全局变量和函数定义）语法。

1. Specifiers

这一部分主要与变量的类型有关

1. Declarators

这一部分的产生式主要与变量和函数的定义有关

1. Statements

这一部分主要和语句有关

1. Local Definitions

这一部分的产生式主要与局部变量的定义有关

1. Exp

主要和表达式有关

1. **冲突处理**

Bison采用的是LR分析法，而且存在一些递归调用，在编写的时候不可避免的会出现一些移进-规约冲突和规约-规约冲突，故在编写时需要注意根据优先级更改结合性，消除直接递归等措施来避免冲突。

1. **结合性处理**

大部分的移进规约冲突和规约规约冲突可以通过修改文法的结合性的优先级来进行避开，例如本实验中的C—文法中的Exp生成部分，是冲突的重灾区，解决的办法就是定义文法的结合性，根据C—文法的定义表，得出结合性如下：

%left <a>LP RP LB RB DOT

%right <a>NOT

%left <a>MINUS DIV PLUS STAR

%left <a>RELOP AND OR

%right <a>ASSIGNOP

其余符号的结合性不需要进行修改，顺序写下即可。

1. **判断语句的移进-规约冲突的解决**

判断语句存在的冲突是通过修改优先级结局不了的，此时需要进行文法的局部修改(增加优先级的判断)，修改前后对照如下：

修改前：

Stmt: IF LP Exp RP Stmt

| IF LP Exp RP Stmt ELSE Stmt

修改后：

%nonassoc NOELSE

%nonassoc ELSE

Stmt: IF LP Exp RP Stmt ELSEFU

ELSEFU: ELSE Stmt|%prec NOELSE;

当语法分析程序读到IF LP Exp RP时候，优先选择移入else 从而避免bison中报告冲突

1. **Bison工作方式**

bison编写程序后，规约的步骤一般如下：

1. 读取/解析命令行选项 Main.getargs(), 打开文件 Files.openfiles()

这些是程序基本的准备工作, 一般不属于算法范畴, 我们略去不述.

2. 读入文法文件 Reader.reader(), 文法文件一般以 .y 为后缀.

读取时建立内存的一些中间数据结构(主要是单链结构), 后面详述.

3. 将第2步读入的数据进行检查, 消除无用的,错误的产生, 建立/转换为适合

计算 LR 的数据结构(主要是单链=>数组结构). 预先计算一些辅助数据.

4. 计算 LR0 状态集, 结果可能是一个非确定的(有冲突的)有限状态机.

5. 转变第 4 步的状态机为确定的 LALR 状态机.

6. 如果第 5 步中有 s/r, r/r 冲突, 则解决冲突.

7. 输出及别的收尾工作. 一般略去不细述了.

1. **语法树的生成**

语法树在语法分析时候是一个重要的组成即代表了整个文法的框架组成，又将各个有用的信息存储到各个节点上，对后边的语义分析起到了至关重要的作用，在语法分析时候，生成一个语法树，可以清晰的看出规约过程是否正确，文法中各部分间的联系等，所以语法树的设计对于整个实验是至关重要的。

语法树的生成方式是利用数据结构中的树的模型，每次扫描都将信息加入到一个节点中，最后生成语法树，遍历语法树。

树中每个节点都是一个子节点结构体，定义如下：

struct ast

{

int line; //行号

char\* name;//语法单元的名字

struct ast \*l;//左孩子

struct ast \*r;//右孩子

union//共用体用来存放ID/TYPE/INTEGER/FLOAT结点的值

{

char\* idtype;

int intgr;

float flt;

};

};

每次生成都会将各个信息填入，然后将节点串联，最后生成一个语法树。

1. **语法树的概述**

语法树是在进行词法分析后，语法分析的时候，一边进行语法分析，一边生成的语法树，每规约一条语句，都将相应的有用信息加入到语法树的相应结点中。叶子节点代表了一个有实际含义的终结符，而中间结点可以看做是规约的过程。

创建结点的方式如下（举例）：

Exp: Exp ASSIGNOP Exp{$$=newest("Exp",3,$1,$2,$3);}

1. **错误恢复（多处报错）**

当输入文件中出现语法错误的时候，Bison总是会让他生成的语法分析程序尽早的报告错误。每当语法分析程序从yylex()得到了一个词法单元，如果当前状态没有针对这个词法单元的动作，那么久认为输入文件里出现了语法错误。

要实现在多个地方报错，首先的目标就是实现“错误恢复”，即在及时已经检测到错误的情况下，还是继续进行语法分析，此时就需要在规则中添加匹配错误信息的规则，添加例子如下：

| error MINUS Exp{yyerrok;yytest\_zy=1;}

例如此例子可实现在减法的第一个操作数出现错误的情况下，讲该错误匹配，然后继续进行后边的匹配，检查后续代码的语法情况，直到遇到error都无法匹配或者结束，然后再进行错误输出，即可实现多处报错。

1. **传入信息到树的结点**

因为如果一个一个数值传入结点的各类信息是很复杂的事情，所以在编写的时候听取了别人的建议，采用了变长列表的方法，将信息通过各个规约语句的所包含符号个数，使用结构体的方式进行变长传值。

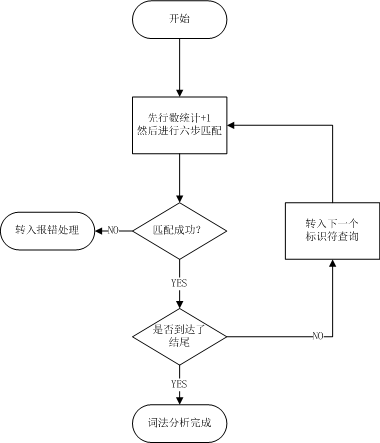
结构体的参数定义如下：

struct ast \*newest(char\* name,int num,...);

其中name为传入的语法单元的名字，num为边长结点的个数，后边跟变长结点。

2.3 词法分析器的设计

词法分析器的结构设计框架图如下：



词法分析器的设计采用flex工具完成，我们只需要进行规则的写入，出错的处理即可，flex的书写规则如下：

{definitions}

%%

{rules}

%%

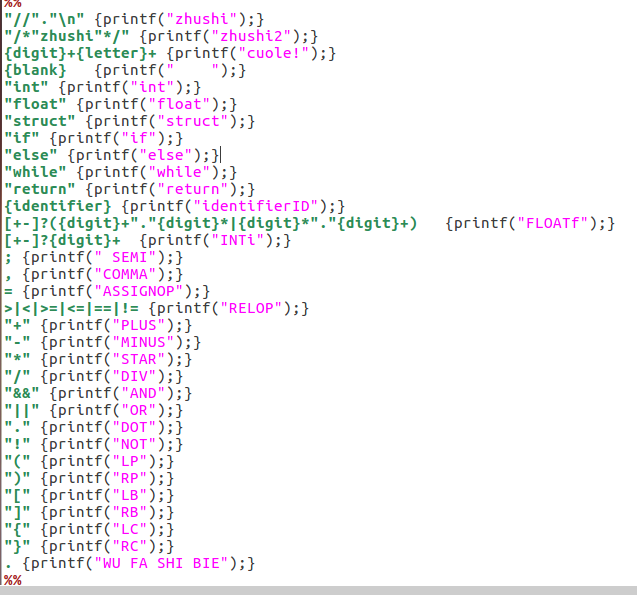
{user subroutines}

书写内容则按照2.2中的描述进行编写。

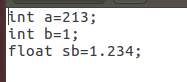
除了int float id需要自己想规则来进行匹配外，其余均在实验指导书上给出，没有难度。

在对C—语法有了了解后，先进行规则查询，使用printf函数输出匹配情况，查看匹配是否成功。

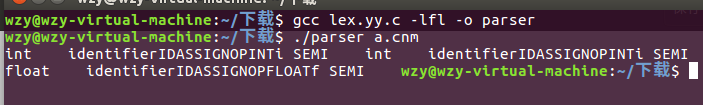
设计代码如下：



尝试输出结果如下：



如果对上述语句进行词法分析，有如下结果：



可以看出，分析成功，则设计方案可行

2.4 语法分析器设计

（1）语法分析器是利用bison直接生成的，采用的是自底向上的上下文无关文法的LALR(1)分析方法，输入的接口来自于上边的词法分析的结果，此时需要将词法分析中的输出到屏幕的匹配信息换成return XXX以便yylex()带出该函数供语法分析使用。

所以词法分析部分代码修改如下：

"int" { yylval.a = newest("DEF\_INT",0,yylineno);return DEF\_INT;}

"float" { yylval.a = newest("DEF\_FLOAT",0,yylineno);return DEF\_FLOAT;}

"struct" { yylval.a = newest("STRUCT",0,yylineno);return STRUCT;}

"if" { yylval.a = newest("IF",0,yylineno);return IF;}

将返回值写为一个个的标识符，通过return带出。

（2）在完成了词法分析器修改后，syntax.y的语法分析文件中应该将相应的返回标识符按照优先级顺序进行列出（根据不同的结合性）（%token %left %right）

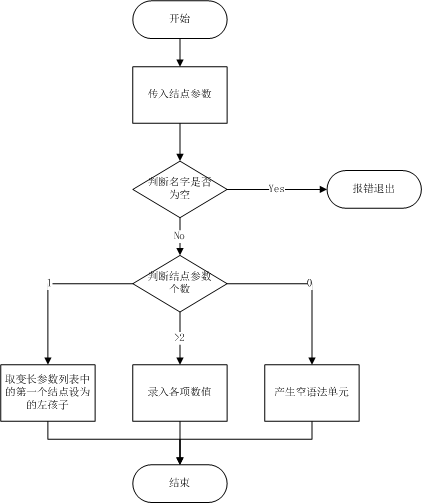
然后将C--文法规则写入到代码段的对应规则部分，每条规则后边加{}备后边生成语法树和语义分析使用。

（3）将错误恢复加入

（4）加入树的节点生成

树的结点生成是通过加入到语义中形成的，每次规约完毕一个句子，则调用节点生成函数。

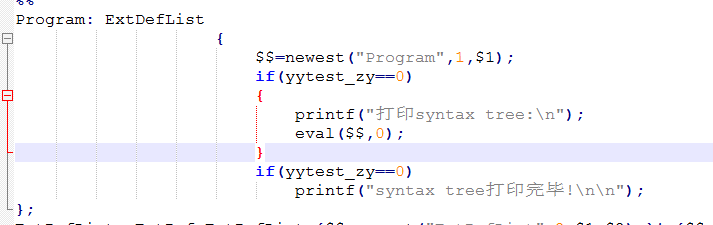
语法生成函数构造方式：



（4）构造完毕后，需要进行语法树的遍历输出，语法树的遍历输出有一个前提条件，就是语法分析必须通过，而语法树是在语法分析的时候同步生成的，所以如果报错，也可能存在语法树的输出（但是不完全），这是我们不希望看到的，所以在语法规则匹配的时候，需要做错误检查。

在这儿错误检查的方法很简单，就是添加一个全局变量error\_flag，如果规则中匹配到了错误恢复，则将error\_flag的值置1，这样，在遍历语法树的时候，就可以查看语法树是否生成完整，如果成功则打印，否则不打印，输出相应错误信息。

输出语法树是在从下到上的语法分析的最后一步分析完成后进行，所以对应的就是规约的开始符，即遍历添加到此处（遍历函数为eval()）：



遍历函数eval递归打印左右结点，输出整个树之后退出。

至此，语法分析完成。

2.5语法分析器实现结果展示

# 3语义分析

3.1语义表示方法描述

3.2符号表结构定义

3.3错误类型码定义

3.4 语义分析实现技术

3.5语义分析结果展示

# 4中间代码生成

### 4.1中间代码格式定义

### 4.2中间代码生成规则定义

### 4.3 中间代码生成过程

### 4.4代码优化

### 4.5 中间代码生成结果展示

# 5目标代码生成

### 5.1指令集选择

### 5.2寄存器分配算法

### 5.3 目标代码生成算法

### 5.4 目标代码生成结果展示

### 5.5目标代码运行结果展示

# 6结束语

6.1 实践课程小结

6.2自己的亲身体会

**参考文献**

[1] 吕映芝等. 编译原理(第二版). 北京：清华大学出版社，2005

[2] 胡伦俊等. 编译原理(第二版). 北京：电子工业出版社，2005

[3] 王元珍等. 80X86汇编语言程序设计. 武汉：华中科技大学出版社,2005

[4] 王雷等. 编译原理课程设计. 北京：机械工业出版社，2005

[5] 曹计昌等. C语言程序设计. 北京：科学出版社，2008

**附件：源代码**