

简易C编译器报告

编译原理课程作业

姓名 韩兴华 | 学号 201992394 | Date

# 词法分析器

## 原理

使用正则表达式匹配输入的字符序列，并识别出字符。

## 规定

### 符号字面值

单个符号匹配： **"'+-\*/()!\"#$%&\',.:;<>?@[]^\_`{}|~=\\"**

多个符号匹配：

t\_PLUSSLF = **r'\+\+'**t\_SUBSLF = **r'--'**t\_GRTREQL = **r'>='**t\_LESSEQL = **r'<='**t\_LSHIFT = **r'<<'**t\_RSHIFT = **r'>>'**t\_EQUAL = **r'=='**t\_NEQUAL = **r'!='**t\_BOOLAND = **r'&&'**t\_BOOLOR = **r'\|\|'**t\_PLUSASSIGN = **r'\+='**t\_SUBASSIGN = **r'-='**t\_MULASSIGN = **r'\\*='**t\_DIVIDEASSIGN = **r'/='**t\_MODASSIGN = **r'%='**t\_XORASSIGN = **r'^='**t\_ORASSIGN = **r'\|='**t\_ANDASSIGN = **r'&='**t\_LSHIFTASSIGN = **r'<<='**t\_RSHIFTASSIGN = **r'>>='**

### 关键字/保留字

**'auto'**: **'AUTO'**,  
**'double'**: **'DOUBLE'**,  
**'int'**: **'INT'**,  
**'struct'**: **'STRUCT'**,  
**'break'**: **'BREAK'**,  
**'else'**: **'ELSE'**,  
**'long'**: **'LONG'**,  
**'switch'**: **'SWITCH'**,  
**'case'**: **'CASE'**,  
**'enum'**: **'ENUM'**,  
**'register'**: **'REGISTER'**,  
**'typedef'**: **'TYPEDEF'**,  
**'char'**: **'CHAR'**,  
**'extern'**: **'EXTERN'**,  
**'return'**: **'RETURN'**,  
**'union'**: **'UNION'**,  
**'const'**: **'CONST'**,  
**'float'**: **'FLOAT'**,  
**'short'**: **'SHORT'**,  
**'unsigned'**: **'UNSIGNED'**,  
**'continue'**: **'CONTINUE'**,  
**'for'**: **'FOR'**,  
**'signed'**: **'SIGNED'**,  
**'void'**: **'VOID'**,  
**'default'**: **'DEFAULT'**,  
**'goto'**: **'GOTO'**,  
**'sizeof'**: **'SIZEOF'**,  
**'volatile'**: **'VOLATILE'**,  
**'do'**: **'DO'**,  
**'if'**: **'IF'**,  
**'while'**: **'WHILE'**,  
**'static'**: **'STATIC'**

### 词法规则匹配

t\_DECIMAL(t): *r'\d\*\.\d+(e\d+)?'* # 小数匹配  
t\_NUMBER(t): *r'\d+'* #整数匹配  
t\_newline(t): *r'\n+' #换行匹配*  
t\_STRINGLITERAL(t): *r'".\*[^\\]"' #字符串字面值（const char \* 类型）匹配*

### 预处理忽略内容

t\_ignore = **' \t' #忽略空格和制表符**t\_ignore\_COMMENT = **r'/\\*[\s\S]\*?\\*/' #忽略注释及注释内内容**

## 实现

使用python库ply.lex，具体代码如下：

**import** ply.lex **as** lex  
  
reserved = {  
 **'auto'**: **'AUTO'**,  
 **'double'**: **'DOUBLE'**,  
 **'int'**: **'INT'**,  
 **'struct'**: **'STRUCT'**,  
 **'break'**: **'BREAK'**,  
 **'else'**: **'ELSE'**,  
 **'long'**: **'LONG'**,  
 **'switch'**: **'SWITCH'**,  
 **'case'**: **'CASE'**,  
 **'enum'**: **'ENUM'**,  
 **'register'**: **'REGISTER'**,  
 **'typedef'**: **'TYPEDEF'**,  
 **'char'**: **'CHAR'**,  
 **'extern'**: **'EXTERN'**,  
 **'return'**: **'RETURN'**,  
 **'union'**: **'UNION'**,  
 **'const'**: **'CONST'**,  
 **'float'**: **'FLOAT'**,  
 **'short'**: **'SHORT'**,  
 **'unsigned'**: **'UNSIGNED'**,  
 **'continue'**: **'CONTINUE'**,  
 **'for'**: **'FOR'**,  
 **'signed'**: **'SIGNED'**,  
 **'void'**: **'VOID'**,  
 **'default'**: **'DEFAULT'**,  
 **'goto'**: **'GOTO'**,  
 **'sizeof'**: **'SIZEOF'**,  
 **'volatile'**: **'VOLATILE'**,  
 **'do'**: **'DO'**,  
 **'if'**: **'IF'**,  
 **'while'**: **'WHILE'**,  
 **'static'**: **'STATIC'**,  
 **'print'**: **'PRINT'**}  
  
*# List of token names. This is always required*tokens = [  
**'PLUSSLF'**,**'SUBSLF'**,**'GRTREQL'**,**'LESSEQL'**,**'LSHIFT'**,**'RSHIFT'**,  
**'EQUAL'**,**'NEQUAL'**,**'BOOLAND'**,**'BOOLOR'**,**'PLUSASSIGN'**,**'SUBASSIGN'**,  
**'MULASSIGN'**,**'DIVIDEASSIGN'**,**'MODASSIGN'**,**'XORASSIGN'**,**'ORASSIGN'**,**'ANDASSIGN'**,  
**'LSHIFTASSIGN'**,**'RSHIFTASSIGN'**,  
 **'NUMBER'**,**'ID'**,**'STRINGLITERAL'**,**'DECIMAL'**] + list(reserved.values())  
  
*# Regular expression rules for simple tokens*literals = **"'+-\*/()!\"#$%&\',.:;<>?@[]^\_`{}|~=\\"**t\_PLUSSLF = **r'\+\+'**t\_SUBSLF = **r'--'**t\_GRTREQL = **r'>='**t\_LESSEQL = **r'<='**t\_LSHIFT = **r'<<'**t\_RSHIFT = **r'>>'**t\_EQUAL = **r'=='**t\_NEQUAL = **r'!='**t\_BOOLAND = **r'&&'**t\_BOOLOR = **r'\|\|'**t\_PLUSASSIGN = **r'\+='**t\_SUBASSIGN = **r'-='**t\_MULASSIGN = **r'\\*='**t\_DIVIDEASSIGN = **r'/='**t\_MODASSIGN = **r'%='**t\_XORASSIGN = **r'^='**t\_ORASSIGN = **r'\|='**t\_ANDASSIGN = **r'&='**t\_LSHIFTASSIGN = **r'<<='**t\_RSHIFTASSIGN = **r'>>='  
  
def** t\_ID(t):  
 *r'[a-zA-Z\_][a-zA-Z\_0-9]\*'* t.type = reserved.get(t.value, **'ID'**) *# Check for reserved words* **return** t  
  
  
*# A regular expression rule with some action code***def** t\_DECIMAL(t):  
 *r'\d\*\.\d+(e\d+)?'* t.value = float(t.value)  
 **return** t  
  
**def** t\_NUMBER(t):  
 *r'\d+'* t.value = int(t.value)  
 **return** t  
  
*# Define a rule so we can track line numbers***def** t\_newline(t):  
 *r'\n+'* t.lexer.lineno += len(t.value)  
  
**def** t\_STRINGLITERAL(t):  
 *r'".\*[^\\]"'* **return** t  
  
  
*# A string containing ignored characters (spaces and tabs)*t\_ignore = **' \t'**t\_ignore\_COMMENT = **r'/\\*[\s\S]\*?\\*/'***# Error handling rule***def** t\_error(t):  
 print(**"Illegal character '%s'"** % t.value[0])  
 t.lexer.skip(1)  
  
  
*# Build the lexer*lexer = lex.lex()

# 语法处理器

## 原理

构建上下文无关文法，使用带有优先级规定的LALR(1)文法分析器进行分析。

## 规定

Rule 0 S' -> Program  
Rule 1 EMPTY -> <empty>  
Rule 2 RELOP -> > | < | GRTREQL | LESSEQL | EQUAL | NEQUAL  
Rule 8 TYPE -> INT | SHORT | CHAR | LONG | FLOAT | DOUBLE  
Rule 14 Program -> ExtDefList  
Rule 15 ExtDefList -> ExtDef ExtDefList  
Rule 16 ExtDefList -> EMPTY  
Rule 17 FunHead -> Specifier FunDec  
Rule 18 ExtDecHead -> Specifier VarDec  
Rule 19 ExtDecList -> ExtDecList , VarDec  
Rule 20 ExtDecList -> ExtDecHead  
Rule 21 ExtDef -> ExtDecList ;  
Rule 22 ExtDef -> Specifier ;  
Rule 23 ExtDef -> FunHead CompSt  
Rule 24 Specifier -> TYPE  
Rule 25 Specifier -> StructSpecifier  
Rule 26 StructSpecifier -> STRUCT OptTag { DefList }  
Rule 27 StructSpecifier -> STRUCT Tag  
Rule 28 OptTag -> ID  
Rule 29 OptTag -> EMPTY  
Rule 30 Tag -> ID  
Rule 31 VarDec -> ID  
Rule 32 VarDec -> ( VarDec )  
Rule 33 VarDec -> VarDec [ NUMBER ]  
Rule 34 VarDec -> FunDec  
Rule 35 VarDec -> \* VarDec  
Rule 36 FunDec -> ID ( VarList )  
Rule 37 FunDec -> ID ( )  
Rule 38 VarList -> ParamDec , VarList  
Rule 39 VarList -> ParamDec  
Rule 40 ParamDec -> Specifier VarDec  
Rule 41 CompSt -> { DefList StmtList }  
Rule 42 StmtList -> Stmt StmtList  
Rule 43 StmtList -> EMPTY  
Rule 44 Stmt -> RETURN Exp ; | PRINT ( Exp ) ;   
Rule 46 FlowCtrl -> IF ( Exp ) Stmt | IF ( Exp ) Stmt ELSE Stmt  
Rule 48 FlowCtrl -> WHILE ( Exp ) Stmt  
Rule 49 Stmt -> Exp ; | CompSt | ; | FlowCtrl  
Rule 53 DefList -> Def ; DefList | EMPTY  
Rule 55 Def -> DecList  
Rule 56 DecHead -> Specifier Dec  
Rule 57 DecList -> DecHead | DecList , Dec  
Rule 59 Dec -> VarDec | VarDec = Exp  
Rule 61 PrefixedExp -> \* Exp | & Exp  
Rule 63 SubTypeSpecifier -> EMPTY  
Rule 64 SubTypeSpecifier -> ( SubTypeSpecifier )  
Rule 65 SubTypeSpecifier -> \* SubTypeSpecifier  
Rule 66 SubTypeSpecifier -> SubTypeSpecifier [ NUMBER ]  
Rule 67 SubTypeSpecifier -> SubTypeSpecifier ( TypeList )  
Rule 68 SubTypeSpecifier -> SubTypeSpecifier ( )  
Rule 69 TypeSpecifier -> TYPE SubTypeSpecifier  
Rule 70 TypeList -> TypeSpecifier  
Rule 71 TypeList -> TypeList , TypeSpecifier  
Rule 72 PrefixedExp -> - Exp | + Exp | PLUSSLF Exp | SUBSLF Exp  
Rule 76 PrefixedExp -> ( TypeSpecifier ) Exp  
Rule 77 Exp -> ( Exp ) | ID | NUMBER | DECIMAL | STRINGLITERAL  
Rule 82 Exp -> Exp = Exp | Exp + Exp | Exp – Exp | Exp \* Exp  
Rule 86 Exp -> Exp / Exp  
Rule 87 Exp -> FuncCall | PrefixedExp  
Rule 89 Exp -> Exp BOOLAND Exp | Exp BOOLOR Exp | ! Exp  
Rule 92 Exp -> Exp RELOP Exp | Exp [ Exp ] | Exp . ID  
Rule 95 FuncCall -> ID ( Args ) | ID ( )   
Rule 97 Args -> Exp , Args | Exp

优先级（由上至下为从高到低）：

(**'left'**,**'('**),  
(**'left'**,**'['**),  
(**'left'**, **'.'**),  
(**'right'**,**'plusslf'**,**'subslf'**),  
(**'right'**,**'UPLUS'**,**'UMINUS'**),  
(**'right'**,**'MEM'**,**'ADDR'**),  
(**'left'**,**'+'**,**'-'**),  
(**'left'**,**'\*'**,**'/'**),  
(**'right'**,**'='**),  
(**'right'**,**','**)

## 实现

使用python库ply.yacc。该python库根据提供的上下文无关文法以及规定优先级将自动匹配规则，我们使用匹配的规则函数来构建语法分析树。

针对输入程序构建语法分析树：

输入程序：

**/\* 输入程序 \*/  
int i\_add(int a,int b){  
 return a+b;  
}  
  
float f\_add(float a,float b){  
 return a+b;  
}  
  
int main(){  
 int a = 3;  
 float fa = 2.0;  
 f\_add(i\_add(a+a,a\*a),fa);  
 return 0;  
}**

构建语法树：

<**Program optr='ExtDefList'**>  
 <**Generic optr='append'**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['int']'"**/>  
 <**Generic optr='append'**>  
 <**ID val='i\_add'**/>  
 <**Generic optr='append'**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['int']'"**/>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='a'**/>  
 </**Generic**>  
 </**LocalDec**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['int']'"**/>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='b'**/>  
 </**Generic**>  
 </**LocalDec**>  
 </**Generic**>  
 </**Generic**>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**Empty**/>  
 <**Generic optr='append'**>  
 <**Stmt optr='return'**>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='b'**/>  
 </**Calc**>  
 </**Stmt**>  
 <**Empty**/>  
 </**Generic**>  
 </**CompStmt**>  
 </**Ext**>  
 <**Generic optr='append'**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['float']'"**/>  
 <**Generic optr='append'**>  
 <**ID val='f\_add'**/>  
 <**Generic optr='append'**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['float']'"**/>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='a'**/>  
 </**Generic**>  
 </**LocalDec**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['float']'"**/>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='b'**/>  
 </**Generic**>  
 </**LocalDec**>  
 </**Generic**>  
 </**Generic**>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**Empty**/>  
 <**Generic optr='append'**>  
 <**Stmt optr='return'**>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='b'**/>  
 </**Calc**>  
 </**Stmt**>  
 <**Empty**/>  
 </**Generic**>  
 </**CompStmt**>  
 </**Ext**>  
 <**Generic optr='append'**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['int']'"**/>  
 <**Generic optr='append'**>  
 <**ID val='main'**/>  
 </**Generic**>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**Generic optr='append'**>  
 <**LocalDec optr='dec'**>  
 <**Type list="['int']'"**/>  
 <**Generic optr='append'**>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='a'**/>  
 </**Generic**>  
 <**Calc optr='='**>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='a'**/>  
 </**Generic**>  
 <**Val val='3'**/>  
 </**Calc**>  
 </**Generic**>  
 </**LocalDec**>  
 <**Generic optr='append'**>  
 <**LocalDec optr='dec'**>  
 <**Type list="['float']'"**/>  
 <**Generic optr='append'**>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='fa'**/>  
 </**Generic**>  
 <**Calc optr='='**>  
 <**Generic optr='append'**>  
 <**Empty**/>  
 <**ID val='fa'**/>  
 </**Generic**>  
 <**Literal val='2.0'**/>  
 </**Calc**>  
 </**Generic**>  
 </**LocalDec**>  
 <**Empty**/>  
 </**Generic**>  
 </**Generic**>  
 <**Generic optr='append'**>  
 <**FuncCall optr='call'**>  
 <**Identifier val='f\_add'**/>  
 <**Generic optr='append'**>  
 <**FuncCall optr='call'**>  
 <**Identifier val='i\_add'**/>  
 <**Generic optr='append'**>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='a'**/>  
 </**Calc**>  
 <**Calc optr='\*'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='a'**/>  
 </**Calc**>  
 </**Generic**>  
 </**FuncCall**>  
 <**Identifier val='fa'**/>  
 </**Generic**>  
 </**FuncCall**>  
 <**Generic optr='append'**>  
 <**Stmt optr='return'**>  
 <**Val val='0'**/>  
 </**Stmt**>  
 <**Empty**/>  
 </**Generic**>  
 </**Generic**>  
 </**CompStmt**>  
 </**Ext**>  
 <**Empty**/>  
 </**Generic**>  
 </**Generic**>  
 </**Generic**>  
</**Program**>

对树进行优化：

<**Program optr='ExtDefList'**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['int']'"**/>  
 <**ID val='i\_add'**/>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['int']'"**/>  
 <**ID val='a'**/>  
 </**LocalDec**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['int']'"**/>  
 <**ID val='b'**/>  
 </**LocalDec**>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**Stmt optr='return'**>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='b'**/>  
 </**Calc**>  
 </**Stmt**>  
 </**CompStmt**>  
 </**Ext**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['float']'"**/>  
 <**ID val='f\_add'**/>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['float']'"**/>  
 <**ID val='a'**/>  
 </**LocalDec**>  
 <**LocalDec optr='param\_dec'**>  
 <**Type list="['float']'"**/>  
 <**ID val='b'**/>  
 </**LocalDec**>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**Stmt optr='return'**>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='b'**/>  
 </**Calc**>  
 </**Stmt**>  
 </**CompStmt**>  
 </**Ext**>  
 <**Ext optr='extdef\_func'**>  
 <**FunDef optr='funhead\_def'**>  
 <**Type list="['int']'"**/>  
 <**ID val='main'**/>  
 </**FunDef**>  
 <**CompStmt optr='compst'**>  
 <**LocalDec optr='dec'**>  
 <**Type list="['int']'"**/>  
 <**ID val='a'**/>  
 <**Calc optr='='**>  
 <**ID val='a'**/>  
 <**Val val='3'**/>  
 </**Calc**>  
 </**LocalDec**>  
 <**LocalDec optr='dec'**>  
 <**Type list="['float']'"**/>  
 <**ID val='fa'**/>  
 <**Calc optr='='**>  
 <**ID val='fa'**/>  
 <**Literal val='2.0'**/>  
 </**Calc**>  
 </**LocalDec**>  
 <**FuncCall optr='call'**>  
 <**Identifier val='f\_add'**/>  
 <**FuncCall optr='call'**>  
 <**Identifier val='i\_add'**/>  
 <**Calc optr='+'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='a'**/>  
 </**Calc**>  
 <**Calc optr='\*'**>  
 <**Identifier val='a'**/>  
 <**Identifier val='a'**/>  
 </**Calc**>  
 </**FuncCall**>  
 <**Identifier val='fa'**/>  
 </**FuncCall**>  
 <**Stmt optr='return'**>  
 <**Val val='0'**/>  
 </**Stmt**>  
 </**CompStmt**>  
 </**Ext**>  
</**Program**>

# 中间代码生成

.globl i\_add

i\_add:

T1 = a + b

retVal = T1

ret

.globl f\_add

f\_add:

T2 = a + b

retVal = T2

ret

.globl main

main:

T3 = a = 3

T4 = fa = 2.0

T5 = a + a

T6 = a \* a

call i\_add

call f\_add

retVal = 0

ret

# 目标代码生成

## 原理

对应的每个终结符/非终结符都有一个节点对象（Node），所有匹配的对象都保存在节点的子节点列表中，并按照匹配的顺序保存。当节点无子节点时，该节点将为Leaf类型（继承自Node）。

字段及其含义：

targetCode: 目标代码，将于对语法树的前向遍历生成的目标代码。

targetCode\_post: 目标代码，将对于语法树的后向遍历生成的目标代码。

Asm\_val: 该节点值的汇编码表示（通常为寄存器值（%eax、%ecx）、内存引用值（4(%esp)、-8(%ebp)）、代码/常量标签（LC0、L0））

Storage\_unit: 记录当前节点所在变量作用域信息。

Type: 记录当前节点数据类型。

方法及其含义：

set\_program方法/set\_program\_post方法：初始化节点，通过设置asm\_val以及type字段、对存储空间的分配等。Set\_program\_post方法用于当子节点的set\_program以及set\_program\_post方法均已调用完毕后调用。

Get\_targetCode/get\_targetCode\_post方法：生成目标代码。带有post后缀的方法同上。

.section .data

LC0:

.float 2.0

.section .text

.globl \_i\_add

\_i\_add:

pushl %ebp

movl %esp,%ebp

movl 12(%ebp),%eax

addl 8(%ebp),%eax

leave

ret

.globl \_f\_add

\_f\_add:

pushl %ebp

movl %esp,%ebp

flds 12(%ebp)

flds 8(%ebp)

faddp %st(0),%st(1)

leave

ret

.globl \_main

\_main:

pushl %ebp

movl %esp,%ebp

subl $20,%esp

movl $3,-4(%ebp)

flds LC0

fsts -8(%ebp)

movl -4(%ebp),%eax

addl -4(%ebp),%eax

movl -4(%ebp),%eax

imull -4(%ebp),%eax

movl %eax,(%esp)

movl %eax,4(%esp)

call \_i\_add

movl %eax,8(%esp)

fildl 8(%esp)

fsts (%esp)

flds -8(%ebp)

fsts 4(%esp)

call \_f\_add

movl $0,%eax

leave

ret