

**实践课程报告**

**题目： Dragon 语言编译器实现**

**课程名称： 编译原理实践**

**专业班级： 　　计科 1409 班**

**学 号： U201414800**

**姓 名： 刘 一 龙**

**指导教师： 邵 志 远**

**报告日期： 2017年1月14日**

**计算机科学与技术学院**

**目录**

[1选题背景 1](#_Toc122562485)

[1.1任务 1](#_Toc1206517837)

[1.2目标 1](#_Toc1401373213)

[1.3 源语言定义 1](#_Toc465094345)

[2实验一 词法分析和语法分析 2](#_Toc46756792)

[2.1单词文法描述 2](#_Toc427335327)

[2.2语言文法描述 2](#_Toc944713140)

[2.3 词法分析器的设计 2](#_Toc1102580707)

[2.4 语法分析器设计 2](#_Toc1870873767)

[2.5语法分析器实现结果展示 2](#_Toc766673682)

[3语义分析 3](#_Toc2001746445)

[3.1语义表示方法描述 3](#_Toc698535825)

[3.2符号表结构定义 3](#_Toc888308457)

[3.3错误类型码定义 3](#_Toc1513785930)

[3.4 语义分析实现技术 3](#_Toc1725340612)

[3.5语义分析结果展示 3](#_Toc1468705877)

[4中间代码生成 4](#_Toc563753937)

[4.1中间代码格式定义 4](#_Toc11155315)

[4.2中间代码生成规则定义 4](#_Toc732638317)

[4.3 中间代码生成过程 4](#_Toc503614825)

[4.4代码优化 4](#_Toc1830073691)

[4.5 中间代码生成结果展示 4](#_Toc1912825680)

[5目标代码生成 5](#_Toc1791878498)

[5.1指令集选择 5](#_Toc723712915)

[5.2寄存器分配算法 5](#_Toc894863510)

[5.3 目标代码生成算法 5](#_Toc1516258529)

[5.4 目标代码生成结果展示 5](#_Toc943065791)

[5.5目标代码运行结果展示 5](#_Toc1498212067)

[6结束语 6](#_Toc404377753)

[6.1 实践课程小结 6](#_Toc501391727)

[6.2自己的亲身体会 6](#_Toc2078498149)

[参考文献 7](#_Toc526940238)

[附件：源代码 8](#_Toc1707909564)

# 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

Dragon 语言是基于 Decaf 语言的一门简单的支持面向对象范式的语言。其语言基本定义如下（仿巴库斯-瑙尔范式，并结合扩展正则表达式 + ?）：

program:

class\_defs

;

class\_defs:

(class\_def ‘,’)\*

;

class\_def:

CLASS IDENTIFIER '{' fields '}'

| CLASS IDENTIFIER EXTENDS IDENTIFIER '{' fields '}'

;

fields:

(field ‘,’)\*

;

field:

var\_def

| func\_def

;

var\_def:

var ';'

;

var:

type IDENTIFIER

;

type:

INT

| BOOL

| STRING

| VOID

| CLASS IDENTIFIER

| type '[' ']'

;

func\_def:

type IDENTIFIER '=' '(' formals ')' OP\_ARROW stmt\_block

;

formals:

(var ‘,’)\*

;

stmt\_block:

‘{‘ stmt ‘}’

;

stmts:

stmt+

;

stmt:

var\_def

| simple\_stmt ';'

| if\_stmt

| while\_stmt

| for\_stmt

| return\_stmt ';'

| print\_stmt ';'

| stmt\_block

;

simple\_stmt:

left\_val '=' right\_val

| call

;

left\_val:

receiver '.' IDENTIFIER

| expr '[' expr ']'

;

if\_stmt:

IF '(' expr ')' stmt

| IF '(' expr ')' stmt ELSE stmt

;

while\_stmt:

WHILE '(' bool\_expr ')' stmt

;

for\_stmt:

FOR '(' simple\_stmt ';' expr ';' simple\_stmt ')' stmt

;

return\_stmt:

RETURN (expr)?

;

print\_stmt:

PRINT '(' exprs ')'

;

call:

receiver' IDENTIFIER '(' actuals ')'

;

receiver:

(expr ‘.’)?

;

actuals:

` (exprs )?

;

exprs:

(expr ‘,’)+

expr:

constant

| left\_val

| THIS

| call

| '(' expr ')'

| expr '+' expr

| expr '-' expr

| expr '\*' expr

| expr '/' expr

| expr '%' expr

| '-' expr

| expr '<' expr

| expr OP\_LE expr

| expr '>' expr

| expr OP\_GE expr

| expr OP\_EQ expr

| expr OP\_NE expr

| expr OP\_AND expr

| expr OP\_OR expr

| '!' expr

| READINTEGER '(' ')'

| READLINE '(' ')'

| NEW IDENTIFIER '(' actuals ')'

| NEW type '[' expr ']'

;

constant:

CONSTANT\_INT

| CONSTANT\_BOOL

| CONSTANT\_STRING

| NIL

;

# 2实验一 词法分析和语法分析

2.1单词文法描述

词法分析阶段所需要做的工作为将输入串按照一定的词法规则解析成 token 序列，提供给下一编译阶段（语法分析）使用。在 Dragon 语言中，token 大致分为四类，关键字、标识符、常量、运算符（包括分隔符）。由于 Flex 的最长匹配与最先匹配原则，应将关键字列在词法规则描述的最前部，才可避免将关键字识别成标识符的错误。

确定好 token 的基本分类与解析优先级后，便可以开始编写词法规则描述。其中，需要特别注意各个常量的词法规则与注释规则的解析。

### 2.1.1 常量规则

10进制整型常量：0 或 非 0 开始的数字串，得到正则表达式为 0|[1-9]{0-9}；

16进制整型常量：0x 或 0X 开始的数字串（包括a-f、A-F），得到正则表达式为0[xX]{a-fA-F0-9}+；

字符串常量：以 “ “ 为开头与结尾的任意串，得到正则表达式为 \"(\\.|[^\\"\n])\*\"。值得一提的是，为了编译器实现逻辑上的统一整洁，并没有编写真正的输出函数，而是直接利用 linux C 中的标准格式化输出函数（printf），从而实现了一定程度上（可以说十分完备的）转义字符支持。

e.g string s; s = “\nreturn”; Print(s);

上述程序会将 “\n” 转义为换行符进行输出。

### 2.1.2 注释规则

行注释：由 // 开始直至换行符即可，得到正则表达式为 "//"[^\n]\*；

块注释：块注释的识别借用了函数，当解析到 /\* 后调用识别块注释的函数，如下所示：

while ((ch = yyinput()) != '\0' && ch != EOF) {

if (ch == '\n') {

srcbuf\_append("\n");

}

if (prev == '\*' && ch == '/') {

return;

} else {

prev = ch;

}

}

上述代码实现了一个简单的状态机，用于识别是否遇到了 \*/，即可完成块注释的识别。

### 2.1.3 规则综述

解决了常量识别与注释识别后，其余规则仅是简单地罗列关键字与符号。综合上述几步，即可完成词法规则描述的实现。利用 Flex 将相关的描述文件（scanner.ll）进行编译得到 C 文件，其便是 Flex 为我们生成的一个可识别 Dragon 语言中所有合法 token 的确定性有限状态自动机（DFA）。利用这个文件便可完成所有输入串的识别与 token 序列的生成。

### 2.1.4 调试信息

为了 Dragon 编译器能够有良好的开发体验与用户体验，需要提供 2 种调试信息。一类为面向开发者的 token 序列识别结果信息，另一类为面向用户的 token 位置信息（用于词法/语法/语义错误提示）。

为实现面向开发者的调试信息，仅需编写简单的输出函数，并将其加入至词法规则中的动作中去，便可完成相关调试信息的输出。为此，编写了一个名为 DRAGON\_DEBUG 的宏，如下所示：

/// \macro DRAGON\_DEBUG

/// \brief print debug info for lexical analysis

#ifdef LEX\_DEBUG

#undef DRAGON\_DEBUG

#define DRAGON\_DEBUG(type, value) do { dragon\_debug(type, value); } while (0)

#else

#undef DRAGON\_DEBUG

#define DRAGON\_DEBUG(type, value) do { } while (0)

#endif

可以看到，当开发者在源码（scanner.ll）中加入 LEX\_DEBUG 宏定义，即可打开词法分析调试规则，上述宏会自动调用声明于 src/errors/utils.h 中的调试信息输出函数来完成打印调试信息功能。

为实现面向用户的调试信息，利用 Flex 内置的位置信息宏与位置信息结构体，即可实现位置信息的获取。按照实验指导书上所给的内置宏定义，并加以修改，得到下列宏定义，以获取当前 token 的位置信息：

#define YY\_USER\_INIT \

yy\_push\_state(SRCBUF); \

current\_line = current\_column = 1; \

yylloc.first\_line = yylloc.first\_column = 1; \

srcbuf\_init();

#define YY\_USER\_ACTION \

yylloc.first\_line = yylloc.last\_line = yylineno; \

yylloc.first\_column = current\_column; \

yylloc.last\_column = current\_column + yyleng - 1; \

current\_column += yyleng;

2.2语言文法描述

由于 Bison 默认进行 LALR（1）分析，所以需要将第1章所提到的文法改写为不含任何移进-归约冲突或归约-归约冲突的 LALR（1）文法。

2.3 词法分析器的设计

2.4 语法分析器设计

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