图标

描述已自动生成

**编译原理实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| **专业** | **：** | **计算机科学与技术** |  |
| **班级** | **：** | **ACM1901班** |  |
| **学号** | **：** | **U201915035** |  |
| **姓名** | **：** | **邹雅** |  |
| **电话** | **：** | **15058667378** |  |
| **邮箱** | **：** | **1542527211@qq.com** |  |

**独创性声明**

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签名：

日期：2022年 7月 5日

|  |  |
| --- | --- |
| 成绩 |  |
| 教师签名 |  |

目 录

[1 概述 1](#_Toc107830387)

[1.1 实验目的 1](#_Toc107830388)

[1.2 实验任务 1](#_Toc107830389)

[1.3 实验要求 1](#_Toc107830390)

[2 编译工具链的使用 3](#_Toc107830391)

[2.1 实验任务 3](#_Toc107830392)

[2.2 实验实现 3](#_Toc107830393)

[3 词法分析器的设计与实现 4](#_Toc107830394)

[3.1 SysY语言词法分析器的设计 4](#_Toc107830395)

[3.2 SysY语言词法分析器的实现 5](#_Toc107830396)

[4 语法分析器的设计与实现 9](#_Toc107830397)

[4.1 SysY语言语法分析器的设计 9](#_Toc107830398)

[4.2 SysY语言语法分析器的实现 11](#_Toc107830399)

[5 一个L-翻译模式的递归下降翻译程序的实现 13](#_Toc107830400)

[5.1 L-翻译模式自顶向下语义计算 13](#_Toc107830401)

[5.2 一个L-翻译模式递归下降翻译程序的实现 13](#_Toc107830402)

[6 PL/0编译系统分析与修改 17](#_Toc107830403)

[6.1 PL/0编译系统分析 17](#_Toc107830404)

[6.2 PL/0编译系统的修改 19](#_Toc107830405)

[7 总结与展望 20](#_Toc107830406)

[7.1 实验总结 20](#_Toc107830407)

[7.2 实验感想 20](#_Toc107830408)

[7.3 实验展望 21](#_Toc107830409)

[参考文献 22](#_Toc107830410)

# 概述

## 实验目的

《编译原理实验》是配合《编译原理》课程理论教学独立开设的实验课程。目的在于通过一系列专门设计的实验，完成一个高级语言-SysY(C语言的一个子集)的定义和编译。熟悉业界经典的工具软件，从零开始完成一个完整的编译器，将该语言源程序编译为ARM汇编语言程序。

通过实验培养大型工程项目的设计与管理能力，提升系统软件实践能力。

## 实验任务

实验任务包括一个完整的编译器的构造过程，和三个独立的实验。完整的编译器构造过程即构造SysY语言的编译器，又细分为：词法分析，语法分析，符号表管理，语义分析，中间代码生成，代码优化和目标代码生成等阶段。三个独立的实验包括：

1. 编译工具链的使用；
2. 一个L-翻译模式的递归下降翻译程序的实现；
3. PL/0编译器的分析与修改。

## 实验要求

以上实验任务并不需要全部完成，可根据情况选做其中的部分实验(必做的实验除外)。

实验一（编译工具链的使用），实验二（词法分析），实验三（语法分析）的第１关（即语法正确性检查）是必做部分，第２关（抽象语法树的构造和输出）的６个测试用例除用例５（该用例含for语句）以外，原则上应能基本正确地输出其它５个用例的抽象语法树（输出形式不限）。实验三(语法分析)、实验七(中间代码生成)、实验九(代码优化)、实验十(功能评测)等四个实验的得分以老师最后打分为准，其它实验的成绩直接采纳平台评测的分数。

根据自己实际完成情况，据实撰写实验报告。

# 编译工具链的使用

## 实验任务

掌握gcc, clang, arm-linux-gnueabihf-gcc,qemu-arm等编译工具的使用。

## 实验实现

1. GCC编译器的使用

用gcc将def-test.c 与alibaba.c 编连，又因为代码中需要定义BILIBILI才能实现对话，因此在命令行中用-D定义BILIBILI。具体代码如下：

gcc -DBILIBILI def-test.c alibaba.c -o def-test

1. CLANG编译器的使用

用clang编译器把上述程序bar.c“翻译”成优化的(优化级别O2)armv7架构，linux系统，符合gnueabihf嵌入式二进制接口规则，并支持arm硬浮点的汇编代码。汇编代码文件名为bar.clang.arm.s。

具体代码如下：

clang -O2 -S -target armv7-linux-gnueabihf bar.c -o bar.clang.arm.s

1. 交叉编译器arm-linux-gnueabihf-gcc和qemu-arm虚拟机的使用

用交叉编译器arm-linux-gnueabihf-gcc将源程序“翻译”成arm汇编代码，再将汇编代码汇编并与Sysy2022运行时库连接，生成arm可执行代码，然后用qemu-arm虚拟机运行arm可执行程序。

具体代码如下：

arm-linux-gnueabihf-gcc iplusf.c -o iplusf.arm.s -S

arm-linux-gnueabihf-gcc iplusf.arm.s sylib.a -o iplusf.arm

qemu-arm -L /usr/arm-linux-gnueabihf/ ./iplusf.arm

# 词法分析器的设计与实现

## SysY语言词法分析器的设计

实现词法分析程序的自动构造通常是用正规表达式或正规文法作为词法规则的形式描述，然后通过转化为等价的有限自动机来设计相应的单词识别过程。自动构造工具按照一定的控制策略生成词法分析程序中扫描程序的代码，该扫描程序可以选择对每一单词种别所对应的有限自动机进行模拟运行，并从当前输入符号序列中识别下一个单词，然后返回相应的单词符号。

flex工具要求以正则表达式的形式给出词法规则，遵循上述技术线路，flex自动生成给定的词法规则的词法分析程序。于是，设计能准确识别各类单词的正则表达式就是关键。

本章节实验要求利用 flex 工具生成SysY2022语言的词法分析器，要求输入一个SysY2022语言源程序文件，词法分析器能输出该程序的token以及token的种别。一个高级语言的词法分析器，需要识别五类单词：关键字、运算符、界符、常量和标识符。

使用工具flex生成词法分析程序时，按照其规定的格式，生成一个flex文件，flex的文件扩展名为.l的文本文件，其语法被分为三个部分：

/\* 辅助定义部分 \*/

%option yylineno noyywrap

%{C代码}%

正则式定义

%%

/\* 规则部分 \*/

正则式  {对应的动作：C代码}

%%

/\* 用户子程序段 \*/

辅助定义部分部分一般是一些声明及选项设置等。C 语言的注释、头文件包含等一般就放在 %{ %} 之间；定义段还可以定义正则式，这些正则式可以在其定义后被后续的正则式所引用，引用之前定义的正则式，要用{}将先前定的的正则式括起来。

规则部分为一系列匹配模式和动作，模式一般使用正则表达式书写，动作部分为C代码：模式 1 {动作 1 (C代码)}，在输入和模式 1 匹配的时候，执行动作部分的代码。

用户子程序部分的代码会原封不动的被复制到词法分析器源程序lex.yy.c中。

## SysY语言词法分析器的实现

flex 是通过处理其源文件来生词法和语法分析器的，按照其三个部分分别构建SysY语言词法分析器：

1. 辅助定义部分：

首先需要定义单词符号的独立种别，包括数据类型、保留字、无符号整数、标识符、运算符以及界符。其中比较特别的是 LEX\_ERR 表示词法错误。

接着给出一些正则式。

数字为0到9组成且任选的其中一个。字母包括小写字母a到z和大写字母A到Z。标识符为开头为字母，后续可以有任意个数字或字母的串。后续的正则式还包括了真值串EXP、不合法的字符串Invaild、多行注释MultilineComment、当行注释SingleLineComment、int类型数字 INT\_LIT、float类型数字FLOAT\_LIT。具体定义如下代码所示：

%option noyywrap

%option yylineno

%{

#include <stdio.h>

enum Token {

    INT=258, FLOAT, VOID, CONST, RETURN, IF, ELSE, FOR, WHILE, DO, BREAK, CONTINUE,

    LP, RP, LB, RB, LC, RC, COMMA, SEMICOLON, QUESTION, COLON,

    MINUS, NOT, TILDE, ASSIGN, ADD, MUL, DIV, MOD, AND, OR,

    EQ, NE, LT, LE, GT, GE,

    ID, INT\_LIT, FLOAT\_LIT,

    LEX\_ERR

};

%}

DIGIT [0-9]

LETTER [A-Za-z]

ID [a-z\_A-Z][a-z\_A-Z0-9]\*

INT\_LIT ([1-9][0-9]*\*|0[0-7]\**|(0x|0X)[0-9a-fA-F]+)

EXP ([Ee][-+]?[0-9]+)

FLOAT\_LIT (([0-9]\*\.[0-9]+|[0-9]+\.){EXP}?[fF]?)|[0-9]+{EXP}[fF]?

MultilineComment "/*\*"([^\\**]|(\*\*)\**[^\*\*/])\**(\*\*)\**"\*/"

SingleLineComment "//".\*$

Invalid {DIGIT}+{LETTER}+{DIGIT}\*

1. 规则部分：

一条规则的组成为：

**正规表达式 动作**

表示词法分析器一旦识别出**正规表达式**所对应的单词，就执**行动**作所对应的操作，返回单词的种类码。在这里可写代码显示（种类编码，单词的自身值），观察词法分析每次识别出来的单词。

"int" {printf("%s : INT\n", yytext); return INT; }

"float" {printf("%s : FLOAT\n", yytext); return FLOAT; }

"void" {printf("%s : VOID\n", yytext); return VOID; }

"const" {printf("%s : CONST\n", yytext); return CONST; }

"return" {printf("%s : RETURN\n", yytext); ; return RETURN; }

"if" {printf("%s : IF\n", yytext); return IF; }

"else" {printf("%s : ELSE\n", yytext); return ELSE; }

"for" {printf("%s : FOR\n", yytext); return FOR; }

"while" {printf("%s : WHILE\n", yytext); return WHILE; }

"do" {printf("%s : DO\n", yytext); return DO; }

"break" {printf("%s : BREAK\n", yytext); return BREAK; }

"continue" {printf("%s : CONTINUE\n", yytext); return CONTINUE; }

"(" {printf("%s : LP\n", yytext); return LP; }

")" {printf("%s : RP\n", yytext); return RP; }

"[" {printf("%s : LB\n", yytext); return LB; }

"]" {printf("%s : RB\n", yytext); return RB; }

"{" {printf("%s : LC\n", yytext); return LC; }

"}" {printf("%s : RC\n", yytext); return RC; }

"," {printf("%s : COMMA\n", yytext); return COMMA; }

";" {printf("%s : SEMICOLON\n", yytext); return SEMICOLON; }

"?" {printf("%s : QUESTION\n", yytext); return QUESTION; }

":" {printf("%s : COLON\n", yytext); return COLON; }

"!" {printf("%s : NOT\n", yytext); return NOT; }

"~" {printf("%s : TILDE\n", yytext); return TILDE; }

"=" {printf("%s : ASSIGN\n", yytext); return ASSIGN; }

"-" {printf("%s : MINUS\n", yytext); return MINUS; }

"+" {printf("%s : ADD\n", yytext); return ADD; }

"\*" {printf("%s : MUL\n", yytext); return MUL; }

"/" {printf("%s : DIV\n", yytext); return DIV; }

"%" {printf("%s : MOD\n", yytext); return MOD; }

"&&" {printf("%s : ADD\n", yytext); return ADD; }

"||" {printf("%s : OR\n", yytext); return OR; }

"==" {printf("%s : EQ\n", yytext); return EQ; }

"!=" {printf("%s : NE\n", yytext); return NE; }

"<" {printf("%s : LT\n", yytext); return LT; }

"<=" {printf("%s : LE\n", yytext); return LE; }

">" {printf("%s : GT\n", yytext); return GT; }

">=" {printf("%s : GE\n", yytext); return GE; }

{ID} {printf("%s : ID\n", yytext); return ID; }

{INT\_LIT} {printf("%s : INT\_LIT\n", yytext); return INT\_LIT; }

{FLOAT\_LIT} {printf("%s : FLOAT\_LIT\n", yytext); return FLOAT\_LIT; }

{MultilineComment} {}

{SingleLineComment} {}

{Invalid} {printf("Lexical error - line %d : %s\n", yylineno,yytext);return LEX\_ERR;}

[ \n] {}

[ \r\t] {}

[ \n\t]+ {}

. {printf("Lexical error - line %d : %s\n", yylineno,yytext);return LEX\_ERR;}

0([0-7][8-9]|[8-9][0-7]|[8-9][8-9]) {printf("Lexical error - line %d : %s\n", yylineno,yytext);return LEX\_ERR;}

1. 用户子程序部分，本部分为 C 代码，一般这里定义一些辅助函数等，如动作代码中使用到的辅助函数。

int main(int argc, char\*\* argv) {

    if(argc > 1) {

        if(!(yyin = fopen(argv[1], "r"))) {

           perror(argv[1]);

           return  1;

        }

        int symbol;

        while (symbol = yylex()) {};

        fclose(yyin);

    } else printf("No source file given!\n");

    return 0;

}

# 语法分析器的设计与实现

## SysY语言语法分析器的设计

bison工具采用了LALR（1）的自底向上的分析技术，完成语法分析。通常语义分析是采用语法制导的语义分析，所以在语法分析的同时还可以完成部分语义分析的工作，在bison文件中还会包含一些语义分析的工作。bison程序的扩展名为.y，其文件由声明部分、辅助定义部分、规则部分和用户函数部分组成。

使用bison采用的是LR分析法，需要在每条规则后给出相应的语义动作,例如对规则：Exp →Exp =Exp，在parser.y中为：

Exp: Exp ASSIGNOP Exp {$$=mknode(ASSIGNOP,$1,$3,NULL,yylineno); }

规则后面{}中的是当完成归约时要执行的语义动作。规则左部的Exp的属性值用$$表示，右部有2个Exp，位置序号分别是1和3，其属性值分别用$1和$3表示。定义的mknode函数，完成建立一个树结点，这里的语义动作是将建立的结点的指针返回赋值给规则左部Exp的属性值，表示完成此次归约后，生成了一棵子树，子树的根结点指针为$$，根结点类型是ASSIGNOP，表示是一个赋值表达式。该子树有2棵子树，第一棵是$1表示的左值表达式的子树，在mini-c中简单化为只要求ID表示的变量作为左值，第二棵对应是$3的表示的右值表达式的子树，另外yylineno表示行号。通过上述给出的所有规则的语义动作，当一个程序使用LR分析法完成语法分析后，如果正确则可生成一棵抽象语法树。

对语句分析时，一旦有错，跳过分号（SEMI），继续进行语法分析。

构造抽象语法树是语法分析阶段很重要的一个任务。AST不同于推导树，去掉了一些修饰性的单词部分，简明地把程序的语法结构表示出来，后续的语义分析、中间代码生成都可以通过遍历抽象语法树来完成。

构造 AST 需要有 mknode() 函数来创建并构造节点，具体代码如下：

struct node \* mknode(int kind,struct node \*first,struct node \*second, struct node \*third,int pos ) {

  struct node \*T=(struct node \*)malloc(sizeof(struct node));

  T->kind=kind;

  T->ptr[0]=first;

  T->ptr[1]=second;

  T->ptr[2]=third;

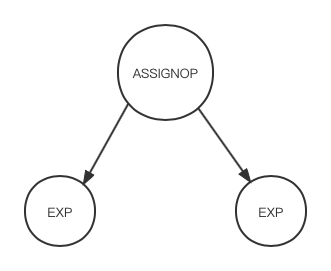
  T->pos=pos;

  return T;

}

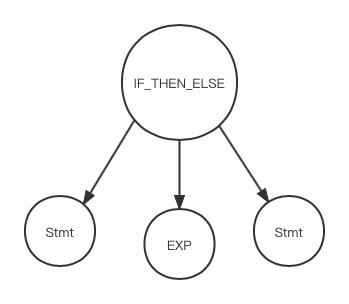
阅读代码可以发现，首先通过 malloc 函数分配空间创建一个新的node，然后用传入的参数进行初始化kind、pos和指向的节点ptr数组，并且返回这个新的节点给外部表示，进行后续使用。此外还需要有一个display()函数实现对抽象语法树的先序遍历。

上述提到的赋值语句即可构造语法树如下：



If语句Stmt：IF LP Exp RP Stmt ELSE Stmt

其生成对应AST的规则为$$=mknode(IF\_THEN\_ELSE,$3,$5,$7,yylineno); 也即生成如下图所示的语法树：



本章节实验要求利用flex+bison生成SysY2022的语法分析程序。要求任给一个SysY2022语言的源程序，能识别并定位源程序中的语法错误。

## SysY语言语法分析器的实现

1. 修改def.h定义，在node\_kind枚举类型中增加一些变量。

enum node\_kind  { EXT\_DEF\_LIST,EXT\_VAR\_DEF,FUNC\_DEF,FUNC\_DEC,EXT\_DEC\_LIST,PARAM\_LIST,PARAM\_DEC,

VAR\_DEF,DEC\_LIST,DEF\_LIST,COMP\_STM,STM\_LIST,EXP\_STMT,IF\_THEN,IF\_THEN\_ELSE, FUNC\_CALL,ARGS,

FUNCTION,PARAM,ARG,CALL,LABEL,GOTO,JLT,JLE,JGT,JGE,EQ,NEQ,

EXT\_CONST\_DEF, VAR\_DEC,FOR\_ARGS, FOR\_STMT, BREAK\_STMT, CONTINUE\_STMT,VAR\_ARR\_DEC, ARRAYB,PLUS\_PLUS\_EXP, MINUS\_MINUS\_EXP};

最后一行从EXT\_CONST\_DEF开始的枚举变量为新增的。

同时增加一个全局变量char类型数组filename用来保存文件名。

char filename[100];

1. 编写 make 文件，生成 parser 程序用来测试一系列测试用例：

parser: parser.tab.c ast.c lex.yy.c

 gcc parser.tab.c ast.c lex.yy.c -o parser

parser.tab.c: sysy.y

 bison -d sysy.y -o parser.tab.c

lex.yy.c: sysy.l

 flex sysy.l

1. 在flex词法描述文件sysy.l文件中使用flex定义所需的符号。其中规则部分和3.2小节中描述的规则部分一致，列出部分代码作为示例：

{int}        {yylval.type\_int=atoi(yytext); return INT;}

{float}      {yylval.type\_float=atof(yytext); return FLOAT;}

"int"        {strcpy(yylval.type\_id,  yytext);return TYPE;}

"float"      {strcpy(yylval.type\_id,  yytext);return TYPE;}

1. 在bison语法描述文件sysy.y文件，构造上下文无关文法，着重处理归约-归约冲突。完成这部分是需要实现上一小节所描述的显示语法树，即在规则部分后增加语义动作。语法树的根结点的program给出产生式推导出非终结符ExtDefList，同时给出对应的语义动作，输出一个语法树并完成语义分析。在这里也列出部分代码作为示例：

program: ExtDefList    {printf("CompUnit:\n"); display($1,3); */\*semantic\_Analysis0($1);\*/*};    */\*显示语法树,语义分析\*/*

ExtDefList: {$$=NULL;}

          | ExtDef ExtDefList {$$=mknode(EXT\_DEF\_LIST,$1,$2,NULL,yylineno);};   *//每一个EXTDEFLIST的结点，其第1棵子树对应一个外部变量声明或函数*

ExtDef:   Specifier ExtDecList SEMI   {$$=mknode(EXT\_VAR\_DEF,$1,$2,NULL,yylineno);}   *//该结点对应一个外部变量声明*

         | CONST Specifier ExtDecList SEMI   {$$=mknode(EXT\_CONST\_DEF,$2,$3,NULL,yylineno);}   *//该结点对应一个外部常量声明*

         |Specifier FuncDec CompSt    {$$=mknode(FUNC\_DEF,$1,$2,$3,yylineno);}         *//该结点对应一个函数定义*

         | VoidType FuncDec CompSt      {$$=mknode(FUNC\_DEF,$1,$2,$3,yylineno);}

         | error SEMI   {$$=NULL; };

VoidType:       VOID {$$=mknode(TYPE, NULL, NULL, NULL, yylineno); $$->type=VOID;}

Specifier:  TYPE    {$$=mknode(TYPE,NULL,NULL,NULL,yylineno);strcpy($$->type\_id,$1);$$->type=!strcmp($1,"int")?INT:FLOAT;};

ExtDecList:  VarDec      {$$=$1;}       */\*每一个EXT\_DECLIST的结点，其第一棵子树对应一个变量名(ID类型的结点),第二棵子树对应剩下的外部变量名\*/*

           | VarDec COMMA ExtDecList {$$=mknode(EXT\_DEC\_LIST,$1,$3,NULL,yylineno);} ;

VarArrDec: ID ArrayB {$$=mknode(VAR\_ARR\_DEC, $2, NULL, NULL, yylineno);strcpy($$->type\_id, $1);};

ArrayB: {$$=NULL;}

        | LB Exp RB ArrayB {$$=mknode(ARRAYB, $2, $4, NULL, yylineno);};

VarDec: VarArrDec {$$=$1;}

        |  ID ASSIGNOP Exp      {$$=mknode(VAR\_DEC,$3,NULL,NULL,yylineno);strcpy($$->type\_id,$1);};   *//ID结点，标识符符号串存放结点的type\_id*

# 一个L-翻译模式的递归下降翻译程序的实现

## L-翻译模式自顶向下语义计算

L-属性文法是既有综合属性又有继承属性的文法，同时需要满足以下两个条件：（1）产生式右端某个符号继承属性的计算必须位于该符号之前，其语义动作不访问位于它右边符号的属性，只依赖于该语义动作左边符号的属性；（2）产生式左部非终结符的综合属性的计算只能在所用到的属性都已计算出来之后进行。

L-翻译模式自顶向下的语义计算方法可以融入到LL(1)预测分析过程之中的方法，改造递归下降分析程序来完成L-翻译模式的语义计算。改造后的分析子程序称为语义计算子函数，改造后的递归下降分析程序称为递归下降语义计算程序或递归下降翻译程序。

本实验要求实现递归的自顶向下语义计算。其具体实现步骤如下：

1. 检查基础文法是否为LL(1)文法
2. 根据文法构造LL(1)分析子程序
3. 启动分析子程序，完成语义计算

其中第二步是整个方法的关键部分。

## 一个L-翻译模式递归下降翻译程序的实现

1. L-翻译模式

实验中给出了可以将一个二进制小数转换成十进制小数的L-翻译模式，具体如下：

1. N → S1 '.' S2 { N.val = S1.val + S2.val \* 2^(-S2.len)}
2. S → B {R.ival = B.val; R.ilen = 1} R { S.val = R.val; S.len := R.len }
3. R → B {R1.ival=2 \* R.ival + B.val; R1.ilen = R.ilen + 1} R1 {R.val = R1.val; R.len = R1.len }
4. R → ε {R.val = R.ival; R.len:=R.ilen }
5. B → '0' {B.val = 0}
6. B → '1' {B.val = 1}

可以计算出产生式的SELECT集合：

SELECT(N → S '.' S) = { '0', '1'}

SELECT(S → BR) = {'0','1'}

SELECT(R → BR) = {'0','1'}

SELECT(R → ε) = {'.','#'}

SELECT(B → '0') = {'0'}

SELECT(B → '1') = {'1'}

不难看出，该文法的确为LL(1)文法。

1. 程序中所涉及到的数据结构定义与函数声明如下：
2. typedef struct {
3. int val;
4. int len;
5. } b\_info;        *//非终结符的数据结构*
6. char getToken();          *//获取下一字符*
7. void matchToken(char expected);          *//匹配字符*
8. float parseN();          *//文法符号N对应的子程序*
9. b\_info parseS();          *//文法符号S对应的子程序*
10. b\_info parseR(int v, int l);          *//文法符号R对应的子程序*
11. int parseB();        *//文法符号B对应的子程序(*
12. char lookahead;         *//当前字符*

入口main函数通过getToken()函数获得下一个字符保存在全局变量lookahead中，并且直接调用parseN()获取最终的val值。不难分析出，这必然是一个递归函数，parseN()函数内部继续调用了parseS()函数。

1. parseS()的实现

* 分析

根据非终结符S的推导式，且，可知，只有当lookahead为0或1时才能进行推导。且由于该推导式的语义动作如下：

S → B {R.ival = B.val; R.ilen = 1} R { S.val = R.val; S.len := R.len }

可以根据产生式推导出来，计算S的val和len值转化为求产生式中R的val和len值，因为两者的值是一致的。而R的ival值是根据B的val值得到的，可以通过调用parseB()函数来得到，因此parseR()函数的第一个参数为parseB()。同时注意到R的ilen值为1，因此第二个参数直接传入1即可。

* 具体实现

b\_info parseS(){

    return parseR(parseB(), 1);

}

1. parseR()的实现

* 分析

非终结符R的推导式为，且

因此可推知当lookahead等于0或1时，用推导式，当lookahead等于.或\n时，用推导式。推导式对应的语义动作如下：

1. R → B {R1.ival=2 \* R.ival + B.val; R1.ilen = R.ilen + 1} R1 {R.val = R1.val; R.len = R1.len }
2. R → ε {R.val = R.ival; R.len:=R.ilen }

由于R有继承属性R.ival以及R.ilen，因此在调用parseR()时需要将继承属性传入函数，也即将继承属性设置为R的形参。其他实现与parseS()类似：先计算继承属性，在计算综合属性。

* 具体实现

详细代码如下：

b\_info parseR(int v, int l){

    if(lookahead == '0' || lookahead == '1'){

        float f = parseB() + 2 \* v;

        return parseR(f , l + 1);

    }

    else if (lookahead == '.' || lookahead == '\n'){

        b\_info b;

        b.val = v;

        b.len = l;

        return b;

    }

    else {

        printf("syntax error\n");

        exit(0);

    }

}

# PL/0编译系统分析与修改

## PL/0编译系统分析

PASCAL语言是上世纪七八十年代非常流行的语言，其设计者为ACM图灵奖获得者Niklaus Wirth。它语法严谨，层次分明，程序易写，可读性强，是第一个结构化编程语言。PL/0语言是一种类PASCAL语言，是教学用程序设计语言，它比PASCAL语言简单，作了一些限制。PL/0的程序结构比较完全，赋值语句作为基本结构，构造概念有顺序执行、条件执行和重复执行。此外，PL0还具有子程序概念，包括过程说明和过程调用语句。

PL/0语言及其编译器亦为Wirth本人所设计，该编译系统用PASCAL语言编写。本实训所用的编译系统C语言源代码是基于Wirth的PASCAL版本改写。

PL/0编译系统采用的是在虚拟机上运行，即将PL/0源程序翻译成一种中间代码P-CODE，然后在P-CODE虚拟机上运行。

PL/0词法分析主要由函数getsym()完成,getsym会调用getch()。getsym返回下一个token。除标识符和无符号整数外，其他单词符号只包含了单词种别的信息。标识符和无符号整数的单词符号包含单词种别和单词自身的值两个部分。由于标识符是在语法分析阶段登录在符号表里的，所以对于标识符来说，PL/0词法分析程序所返回的单词自身的值不是符号表位置的指针，而是标识符的名字串。PL/0编译程序定义3个全程变量来传递单词种别和单词自身的值：

1. 通过全局变量sym传递单词种别；

enum symbol {

    nul,         ident,     number,     plus,      minus,

    times,       slash,     oddsym,     eql,       neq,

    lss,         leq,       gtr,        geq,       lparen,

    rparen,      comma,     semicolon,  period,    becomes,

    beginsym,    endsym,    ifsym,      thensym,   whilesym,

    writesym,    readsym,   dosym,      callsym,   constsym,

    varsym,      procsym,

};

enum symbol sym;

1. 通过全局变量id传递标识符单词自身的值，即标识符的名字；

char id[al+1];

1. 通过全局变量num传递无符号整数单词自身的值，即它的整数数值；

int num;

此外，值得注意的是在识别字母数字串的单词后如何区分是标识符还是保留字。常采取的方法是预设一个保留字表，通过查表来确定是否保留字。其中保留字名字的保存按照字母顺序排列，便于折半查找。

另外，还需要设置指令名称数组。

*/\* 设置指令名称 \*/*

 strcpy(&(mnemonic[lit][0]), "lit");

 strcpy(&(mnemonic[opr][0]), "opr");

 strcpy(&(mnemonic[lod][0]), "lod");

 strcpy(&(mnemonic[sto][0]), "sto");

 strcpy(&(mnemonic[cal][0]), "cal");

 strcpy(&(mnemonic[inte][0]), "int");

 strcpy(&(mnemonic[jmp][0]), "jmp");

 strcpy(&(mnemonic[jpc][0]), "jpc");

PL/0的文法是LL(1)的，其语法分析采用自顶向下分析法，具体采用递归下降子程序法，即为每个语法单位编写一个子程序。

PL/0的语义分析和中间代码生成差不多都在block()完成，gen()只负责将生成的中间代码记录到中间代码数组。PL/0没有对中间代码进行优化。对中间代码的解释执行由interprete()完成。PL/0运行虚拟机是一个栈式虚拟机，数据存储和算术、逻辑运算都在栈顶进行。

## PL/0编译系统的修改

1. 在入口main函数中，增加从命令行输入参数的两个形参int argc和char\* argv[]，并用下列语句将第二个参数拷贝到fname文件名字符串中：

strcpy(fname,argv[1]);

1. 把main函数中scanf、printf和fprintf等函数注释掉，把main函数中 listswitch 和 tableswitch 变量设置为false，不输出虚拟机代码和名字表等内容。
2. 把getch函数中printf和fprintf函数注释掉，不需输出从源程序读入的字符
3. 实现listallcode()函数。在虚拟机代码执行中间代码前调用listallcode()函数一次性输出全部代码，同时将中间代码写入fa.tmp文件。其实现的代码和源程序中void listcode(int cx0)函数中代码基本一致，除了不需要用listswitch变量进行判断之外。
4. void listallcode()
5. {
6. fa = fopen("fa.tmp","w");
7. for(int i=0;i<cx;i++){
8. printf("%d %s %d %d\n", i, mnemonic[code[i].f], code[i].l, code[i].a);
9. fprintf(fa,"%d %s %d %d\n", i, mnemonic[code[i].f], code[i].l, code[i].a);
10. }
11. fclose(fa);
12. }

# 总结与展望

## 实验总结

在这次编译原理实验加深了我对词法分析、语法分析、L-翻译模式等理论知识的理解。但是由于本次实验难度比较大，我只完成了接下来的几部分：

1. 完成了编译工具链的使用。
2. 完成了词法分析的设计与实现，学会了如何使用flex工具构造词法分析器。
3. 完成了语法分析器的设计与实现。我在更深程度地理解了抽象语法树，对构造 AST 的实现有了一个比较直观的认识，并且在阅读和修改bison的源文件时对语法规则和语义动作有了一定程度的了解。
4. 完成了L-翻译模式的递归子下降翻译程序的实现。这部分实验在理解了L-翻译模式中计算继承属性和综合属性的值之后比较简单。
5. 完成了PL/0编译系统分析与修改。

## 实验感想

编译原理实验无疑是我在本科阶段课程中遇到最难的一次实验。其中最困难的部分就是语法分析器的构造。尽管我认为我在理论知识上已经把课程要求的语法分析知识点都掌握了，但看到所给的代码还是无从下手。

在词法分析器的构造时，确实让我学到了很多。通过实践一步一步完成各个分析的部分，特别是构建规则部分的时候，我能够全面地去考虑一个编译器到底需要哪些符号词汇，以及如何把它们完善地表示出来。

我认为本次实验的问题在于，简单的部分大家都可以很轻易地完成，比如说编译工具链的使用、PL/0编译系统分析与修改这两个部分，起不到很大的思考和实践意义。但是在语法分析器和其他符号表管理、静态语义分析、中间代码生成等等部分难度又过高。如果花费远超于课时的时间在实验上，很显然这是不合理而且起不到实践意义的。我个人觉得实验的意义在于我们能更深层次去思考课本上教给我们的知识，把它转化成代码，而不是重新花费大量时间学习新知识。我认为在词法分析和L-翻译模式的递归子下降翻译程序这两块的内容是设计得比较合理的，结合课本上的知识让我们能很好地去思考。

## 实验展望

编译原理实验我完成得情况并不理想，属于是“打卡式”实验。一方面大三这个阶段各种事情比较多，另一方面对编译器所需要用到的各种工具flex、bison等等理解也不深刻，实践知识不牢固。

接下去的计算机专业学习中，如果再涉及到编译的知识，我会再对这些知识进行研究并且动手完成实验。

# 参考文献

[1] 王生元 等. 编译原理(第三版). 北京：清华大学出版社，20016

[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