

**编译原理实践报告**

|  |  |
| --- | --- |
| 设计题目： | C--语言设计 |
| 姓 名： | 王旭东 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1504 班 |
| 学 号： | U201514588 |
| 指导教师： | 祝建华 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2018 年 6月25日

**目 录**

[1 选题背景 1](#_Toc517889359)

[1.1 任务 1](#_Toc517889360)

[1.2 目标 1](#_Toc517889361)

[1.3 源语言定义 1](#_Toc517889362)

[2 实验一 词法分析和语法分析 3](#_Toc517889363)

[2.1 单词文法描述 3](#_Toc517889364)

[2.2 语言文法描述 3](#_Toc517889365)

[2.3词法分析器设计 6](#_Toc517889366)

[2.4语法分析器设计 7](#_Toc517889367)

[2.5结果展示 9](#_Toc517889368)

[3课程总结 13](#_Toc517889369)

[参考文献 14](#_Toc517889370)

# 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。

通过实践，熟悉编译程序的总体结构，熟悉编译程序各组成部分及其任务，学习编译过程各阶段所要解决的问题及其采用的方法和技术，掌握关键算法的工作原理。

## 1.3 源语言定义

在本次编译原理实践中，选择C--作为源语言，其风格与C语言相似，该语言支持的基本类型分别如下：

**基本数据类型：**char类型，float类型，int类型；

**关键字：**int，char，float，else，if，return，while；

**常量：**十进制整数和float浮点数，常量字符（如’a’）；

**算术运算：**+，-，\*，/；

**逻辑运算：**&&,||；

**移位运算：**>>,<<；

**比较运算：**<，>，<=，>=，==，!=；

**一元运算：**前置++,前置--,后置++,后置--,+(正号)，-(负号)；

**控制语句：**if，if-else，while，for；

**跳转语句：**break,return；

**注释：**//行级注释，/\*块级注释\*/。

除了上述基本的运算操作外，还支持函数的声明、定义和调用（允许递归调用）。允许的函数规则如下：

函数名不能重复。函数可以事先声明，也可以直接定义，但不允许函数的重复定义。

函数定义时会先判断函数是否声明过，如果函数事先声明过，则检测函数的参数列表与之前函数声明时的参数列表是否匹配，不匹配则报错。

在调用函数时会检测函数有无定义或者声明，没有则报错；如果函数已经定义或者声明过，则检测函数调用时所传的参数个数和类型是否符合函数定义或者声明的标准，如果不符合则报错。

# 2 实验一 词法分析和语法分析

## 2.1 单词文法描述

单词是 C-- 语言中具有独立意义的最小单位， 可分为 5 大类： 关键字（保留字）、 运算符、 界符、 常量和标识符。

C-- 中的关键字（它们都是保留字）包括： 基本类型关键字 int、 float和char 的， 定义函数时用到的 int、 float 和 char， 分支与循环语句涉及的 if、 else、 while， 返回语句 return。

C-- 中的运算符包括：“+ - \* / < <= > >= = == != && || ! [ ] （）”， 涉及到了算术运算、比较运算和逻辑运算，其中（）在函数调用时有特殊意义，不能被当作单纯的分隔符看待。

C--中界符包括：“; ， . { }”。

C-- 中的常量包括：float浮点数，整数，字符。float浮点型常量可用一般的小数形式表示，例如“6.666666、888.888”等，注意，在本词法设计中，小数“.001”和“1.”都算合法小数。

C—中的标识符是以字母或下划线开头的字母、数字和下划线的序列。C—是大小写敏感的，例如if是一个关键字，但IF却是一个标识符，Fun和fun是两个不同的标识符。

## 2.2 语言文法描述

**终结符集合：**

*//% token 定义终结符的语义值类型*

%token <type\_int> INT */\*指定INT的语义值是type\_int，有词法分析得到的数值\*/*

%token <type\_id> ID RELOP TYPE */\*指定ID,RELOP 的语义值是type\_id，有词法分析得到的标识符字符串\*/*

%token <type\_float> FLOAT */\*指定ID的语义值是type\_id，有词法分析得到的标识符字符串\*/*

%token <type\_char> CHAR

%token LP RP LC RC SEMI COMMA */\*用bison对该文件编译时，带参数-d，生成的exp.tab.h中给这些单词进行编码，可在lex.l中包含parser.tab.h使用这些单词种类码\*/*

%token PLUS MINUS ADDA SUBS STAR DIV ASSIGNOP AND OR NOT IF ELSE WHILE RETURN

**非终结符集合：**

*// %type 定义非终结符的语义值类型*

%type <ptr> program ExtDefList ExtDef Specifier ExtDecList FuncDec CompSt VarList VarDec ParamDec Stmt StmList DefList Def DecList Dec Exp Args

**代码规范如下：**

1 program: ExtDefList

2 ExtDefList: /\* empty \*/

3 | ExtDef ExtDefList

4 ExtDef: Specifier ExtDecList SEMI

5 | Specifier FuncDec CompSt

6 | error SEMI

7 Specifier: TYPE

8 ExtDecList: VarDec

9 | VarDec COMMA ExtDecList

10 VarDec: ID

11 FuncDec: ID LP VarList RP

12 | ID LP RP

13 VarList: ParamDec

14 | ParamDec COMMA VarList

15 ParamDec: Specifier VarDec

16 CompSt: LC DefList StmList RC

17 StmList: /\* empty \*/

18 | Stmt StmList

19 Stmt: Exp SEMI

20 | CompSt

21 | RETURN Exp SEMI

22 | IF LP Exp RP Stmt

23 | IF LP Exp RP Stmt ELSE Stmt

24 | WHILE LP Exp RP Stmt

25 | error

26 DefList: /\* empty \*/

27 | Def DefList

28 Def: Specifier DecList SEMI

29 DecList: Dec

30 | Dec COMMA DecList

31 Dec: VarDec

32 | VarDec ASSIGNOP Exp

33 Exp: Exp ASSIGNOP Exp

34 | Exp AND Exp

35 | Exp OR Exp

36 | Exp RELOP Exp

37 | Exp PLUS Exp

38 | Exp MINUS Exp

39 | Exp ADDA

40 | Exp SUBS

41 | ADDA Exp

42 | SUBS Exp

43 | Exp STAR Exp

44 | Exp DIV Exp

45 | LP Exp RP

46 | MINUS Exp

47 | NOT Exp

48 | ID LP Args RP

49 | ID LP RP

50 | ID

51 | INT

52 | FLOAT

53 | CHAR

54 | error SEMI

55 Args: Exp COMMA Args

56 | Exp

接下来具体地介绍 C--中有关程序结构、 作用域、 类型、 变量、 函数、 类和对象的内容。

**程序结构：**

一个 C-- 程序是一个函数定义的序列， 其中每个函数定义包含该函数的完整描述。

一个C—程序应当包含一个int型的main函数。

**作用域：**

Decaf 支持多种层次的作用域。最高层次是全局作用域，其中可以包含函数定义和外部变量定义，每个函数都有一个用于声明参数列表的参数作用域和存放函数体的局部作用域。函数局部作用域中一对大括号建立了一个嵌套的局部作用域。需要注意以下几点：

1. 函数和外部变量定义可以在声明之前使用。
2. 局部作用域中的变量必须先声明后使用。
3. 同一个作用域中的标识符是唯一的。
4. 在嵌套的作用域中重新声明的标识符屏蔽外层的同名标识符，但不允许在局部作用域中声明与外层的局部作用域或参数作用域中的变量同名的变量。此外，函数名不会被任何非全局标识符屏蔽。
5. 不可访问在一个已经关闭的作用域中声明的标识符。

**变量：**

变量的类型可以是int、float、char三种类型之一。

**函数：**

函数的类型可以是int、float、char三种类型之一。

## 2.3词法分析器设计

词法分析器采用的工具是自动化生成工具GUN Flex，该工具要求词法规则以正则表达式（正规式）给出，并根据给定的此法规则生成相应的词法分析程序。Flex的原理是有穷自动机，即Flex会将用正则表达式表示的词法规则等价转化为相应的有穷自动机FA，生成对应的词法分析程序。所以，设计词法分析器的关键是设计能准确识别各类单词的正则表达式。

根据2.1的分析，合法单词包括关键字、运算符、界符、常量和标识符，以及其他一些辅助单词。

关键字的正则表达式比较简单，例如对于关键字while而言，其正则表达式就是“while”（包括引号），以此类推可以得到所有关键字的正则表达式。

运算符与界符的正则表达式与关键字的正则表达式类同，不再赘述。

产量的表达式相对复杂。对于int、float、char整型常量，正则表达式分别为：

**int** [0-9]+

**float** [+-]?([0-9]\*\.[0-9]+)|([0-9]+\.)

**char** \'[^\'\n']\'

字符的正则表达式为：

**id** [A-Za-z][A-Za-z0-9]\*

行级注释和块级注释的正则表达式为：

(\/\/.\*)|(\/\\*((\\*[^\/]?)|[^\\*]\*)\*\\*\/)

为了能在词法分析和语法分析报时提供错误的详细位置信息，运用了Flex的部分高级特性，例如：开启yylineno选项，从而全局变量yylineno会记录当前正在分析的词法单元在源程序中的行号，并由Flex自动维护（初值设为1）。

在用户自定义部分，可选择的打印词法分析的二元组，并返回词法单元属性：

"int" {

    i**f**(select) **printf**("(%s, TYPEINT)\n", yytext);

**strcpy**(yylval.type\_id, yytext);

    return TYPE;

    }

"return" {

    i**f**(select) **printf**("(%s, RETURN)\n", yytext);

    return RETURN;

    }

"if" {

    i**f**(select) **printf**("(%s, IF)\n", yytext);

    return IF;

    }

其中“select”为自定义全局变量，select置1时打印词法分析结果，否则不打印。

## 2.4语法分析器设计

语法分析器的实现采用的是自动化生成工具GUN Bison，Bison可以根据给定的语法规则，自动化生成对应的语法分析程序。语法分析不仅仅是判断源程序的语句是否符合语法规则，还应该构造源程序对应的语法分析树，用于编译的后续阶段。Bison和Flex可以无缝对接，即将Flex进行词法分析后得到的单词序列作为Bison的输入，从而来进行语法分析。

**设计语法分析器的第一步**，便是设计相应的语法规则。语法规则在 2.2 中的 C--语法规范中已经详细给出，这里需要做的便是将语法规则按照 Bison 的标 准写成相应的生成式。由 C-- 语法规范直接转化来的生成式存在大量的移进-规约冲突或规约规约冲突，需要通过一定的方法来消除冲突。对二义性与冲突处理，则是通过显示规定优先级和结合性 来解决。经过排查移进-规约冲突和规约-规约冲突的来源，不难发现大部分的冲 突来自于运算符，造成的原因便是分析器不知道运算符的优先级和结合性。例如， 对于算数表达式“1+2-3”，分析器并不知道是先算“1+2”还是“2-3”。当然， 如果告诉分析器“+”和“-”都是左结合，那么分析器自然知道是要先计算“1+2” 而不是是“2-3”。除了算数表达式外，IF-ELSE 语句的语法也会带来移进-规约冲 突。为了解决二义性与冲突，在 Bison 中定义的结合性和优先级如下所示。

%left ASSIGNOP

%left OR

%left AND

%left RELOP

%left PLUS MINUS

%left STAR DIV

%right UMINUS NOT

%left ADDA SUBS

%nonassoc LOWER\_THEN\_ELSE

%nonassoc ELSE

**设计语法分析器的第二步**（应首先完成第一步的语法分析部分，即语法的生成式已经无移进-规约冲突且符合语法规则）便是构造语法分析树。根据编译原理课程上所学的知识，为了实现在语法分析的同时构造语法树，应该为语法的每条产生式添加一定的语义动作来完成叶节点的生成和添加到已有语法树。这就要求终结符和非终结符应该都有相关的“属性”作为语义动作的对象。再结合最 终目的是构造语法分析树，不难想到这里的“属性”应该选择语法分析树的节点 指针。考虑到语法分析树是一棵各节点度都不相等的树，所以决定采用“孩子兄 弟法”表示语法分析树，这样可以通过二叉树的相关操作（自己比较熟悉）来实现语法分析树的相关操作。

为此，设计并定义语法分析树的节点结构体如下：

struct **node** {

*//以下对结点属性定义没有考虑存储效率，只是简单地列出要用到的一些属性*

        enum node\_kind kind;

        union {

                 char type\_id[33]; *//由标识符生成的叶结点*

                 int type\_int; *//由int生成的叶结点*

char type\_char; *//由char类型生成的叶节点*

                 float type\_float; *//由float类型生成的叶结点*

         };

struct **node** \*ptr[3]; *//由kind确定有多少棵子树*

int level; *//层号*

int place; *//存放表达式入口的位置，这里可以用符号表的位置序号，由此使用一个变量*

char Etrue[15],Efalse[15]; *//对布尔表达式的翻译时，真假转移目标的标号*

char Snext[15]; *//语句S执行后的下一条语句位置标号*

struct **codenode** \*code; *//该结点中间代码单链表头指针*

char type[10],op[10];

int pos; *//语法单位所在位置行号*

int num; *//计数器，可以用来统计形参个数*

};

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

完成了语法分析树相关结构体和函数的定义和实现后，需要将其将其与Bison生成代码相结合，需要将终结符和非终结符的属性值类型声明为语法树的结构体类型，同时为每条生成式添加语义动作，终结符和非终结符具体代码详见2.2，这里不再赘述。语义动作产生式源程序如下：

program: ExtDefList {**printf**("语法树根节点\n"); **display**($1,0);} /\*显示语法树\*/

;

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

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

|error SEMI {$$=NULL;**printf**("Error at line %d\n",yylineno);}

;

Specifier: TYPE {$$=**mknode**(TYPE,NULL,NULL,NULL,yylineno);**strcpy**($$->type,$1);}

;

………………

## 2.5结果展示

测试源程序为test.c

int a,b,c;

float a,g;

float m;

float **d**(int n){

    int x,y;

    x++;

}

int **f**(int a, float b,int c,int d) {

int d,n;

    char z = 'm';

float x,y=20.0;

    a=(10+b)\*c ; */\*CS1504班\*/*

    b=1

    b++; *//U201514588 王旭东*

    b--;

    ++b;

**abc**(1+2,a\*c);

    while (a>1 && b<1)

        a=12;

    if (! a<b) {

     int xx;

        f=2+c;

        {

         int yyy;

        }

        a=1;

        }

else a+b;

    return -(a+b)\*c;

}

源代码中在第13行设置了语法错误。

测试结果如下图

词法打印结果如图2-1和2-2所示

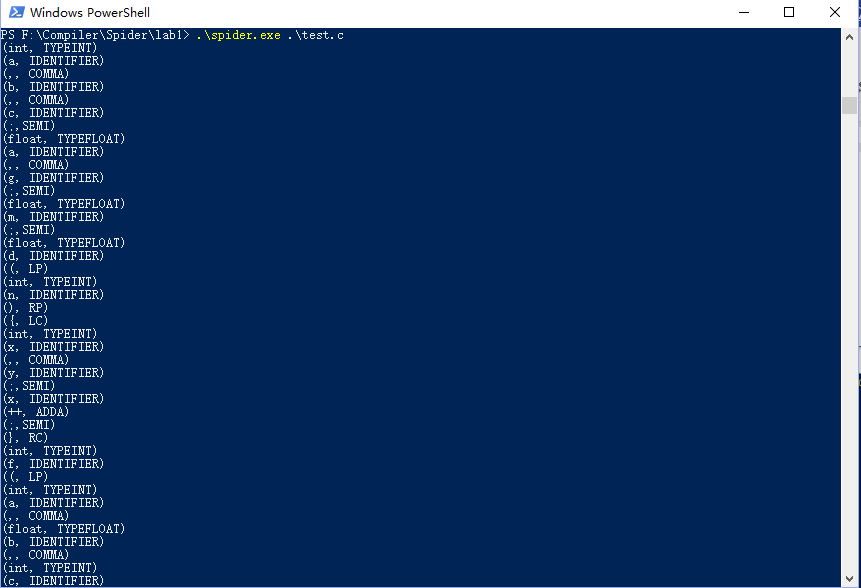


图2-1 词法分析打印结果

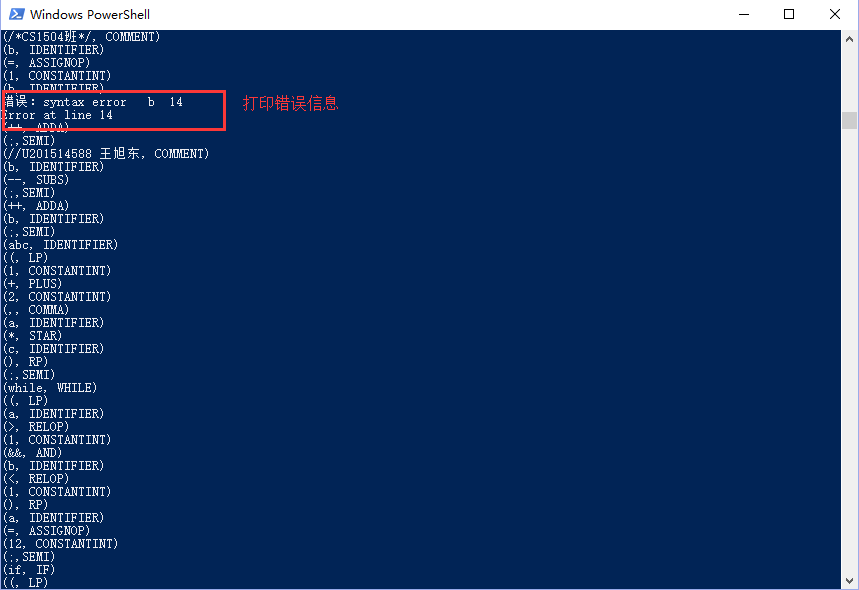


图2-2 词法分析打印结果

语法树打印结果如图2-3和2-4所示

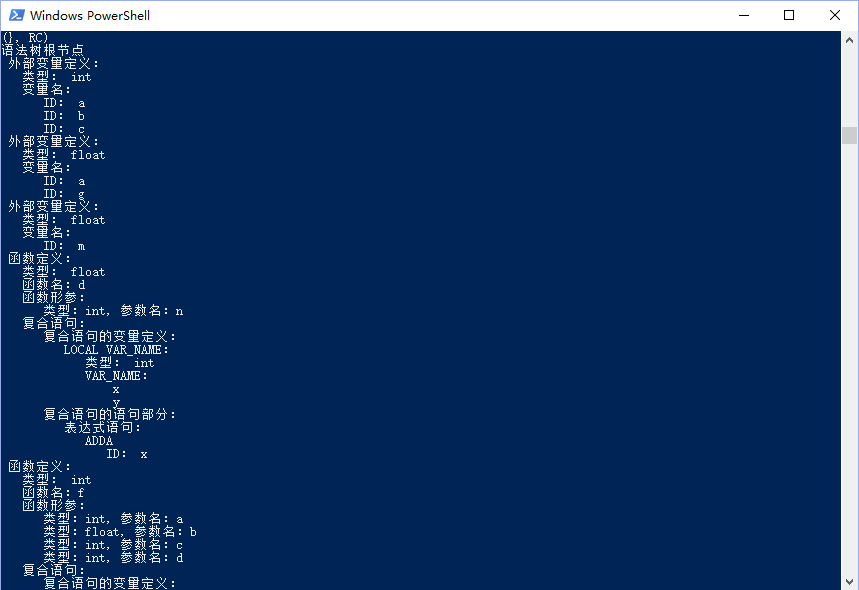


图2-3 语法分析打印结果

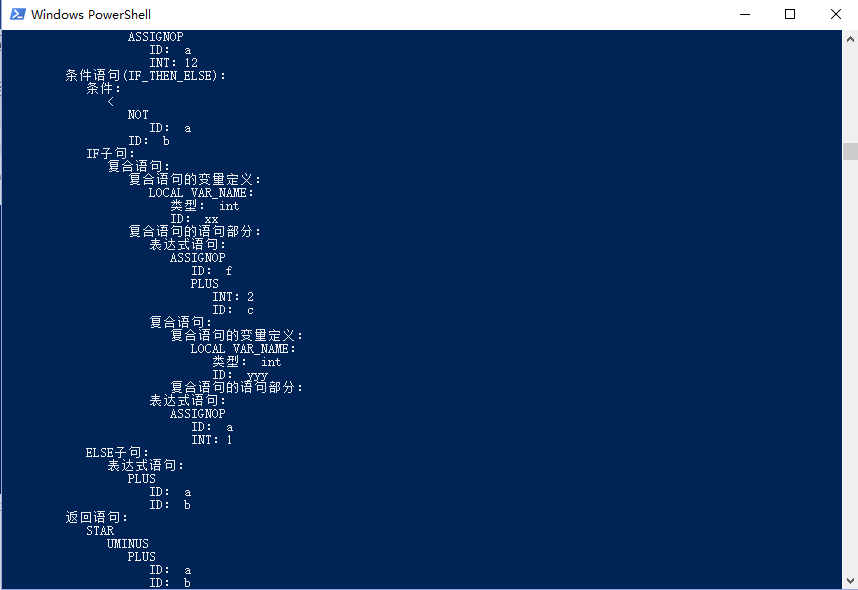


图2-4 语法分析打印结果

# 3课程总结

总体来说，这学期的编译原理实验取得了很多成果和心得，但是很遗憾的是只完成了实验一，实验二动手做了但是并没有完成。

在这个耗费时间较长精力较多的实验中，我学会了如何编写词法分析器的正规表达式和使用自动化生成工具Flex得到词法分析程序，学会了如何让使用自动化生成工具Bison得到语法分析程序，同时也对利用语法分析树进行语义分析和得到其符号表有一定的了解。

刚开始接触实验其实是比较懵逼的，实验和理论课上讲的东西有很大差别，在课堂上只是单纯的听老师讲这个地方应该如何设计，那个阶段又该如何规划，但从来没有考虑过这样设计和规划的原因，自由当自己亲身去实践体会时，才能理解编译器这样设计的原因和初衷，才感受到了历史上前人的成就多么的伟大。刚开始对Flex和Bison非常陌生，因为以前从来没有接触过，通过查阅资料和运行老师给的一些例子后才知道Flex和Bison的大致使用方法。Flex主要是将输入的文本与常规表达式进行匹配，一旦匹配到已经定义的模式，就执行相应的动作，或者显示相应的错误信息。Bison是将Flex分析的单词序列做语法分析，同时还要求生成语法树，这也帮助我回顾了数据结构和C语言等知识。

纵观这学期所学的编译原理，其实编译是一个软件连接硬件的过程，从顶层的源程序，逐步分析深入，直到汇编代码级，编译原理是实现各种软件和硬件开发的基础，是计算机专业的必备知识。

虽然实验花费了较长时间，而且因为自己太菜有没有全部完成，但还是有很多收获和思考的，不仅仅学到了很多新的知识，更改变了自己对整个计算机系统的认知，开拓了视野和眼界。

# 参考文献

[1] 王生原等. 编译原理（第三版） . 北京： 清华大学出版社， 2015

[2] 许畅等. 编译原理实践与指导教程. 北京：机械工业出版社，2015

[3] 曹计昌等. C 语言程序设计. 北京：科学出版社，2008