****

**《编译原理》实验报告**

**组长学号 姓名 韩柳彤**

**组员学号 姓名 张彦博**

**组员学号 姓名 王时予**

**组员学号 姓名 丁天乐**

**组员学号 姓名 钱子牛**

**完成日期 2019年 月 第 周**

目 录

[1. 组内分工与贡献介绍 2](#_Toc27782)

[2. 系统功能概述（包括系统的总体结构） 2](#_Toc25744)

[2.1. 词法分析器 2](#_Toc24045)

[2.2. 语法分析器 2](#_Toc2969)

[2.3. 语义分析与三地址码生成 2](#_Toc15038)

[3. 包含实验指导书要求的内容分系统报告 2](#_Toc13446)

[3.1. 词法分析子系统 2](#_Toc25207)

[3.1.1. 词法的正规式描述 2](#_Toc13304)

[3.1.2. 变换后的正规文法 3](#_Toc26090)

[3.1.3. 状态图 4](#_Toc29539)

[3.1.4. 词法分析程序的主要数据结构与算法 5](#_Toc23805)

[3.1.4.1. 实验内容 5](#_Toc13904)

[3.1.4.2. 实现功能 5](#_Toc17253)

[3.1.4.3. 符号表设计 5](#_Toc32425)

[3.1.4.4. 流程图 6](#_Toc23626)

[3.1.4.5. 实现方法 6](#_Toc12367)

[3.1.4.6. 实现思路 7](#_Toc23540)

[3.1.4.7. 主要数据结构与算法 7](#_Toc22167)

[3.1.5. 实验结果 9](#_Toc20463)

[3.2. 语法分析子系统 10](#_Toc16980)

[3.2.1. 语法分析方法的描述 10](#_Toc16716)

[3.2.2. 语法分析子系统结构 10](#_Toc11679)

[3.2.3. 语法分析子系统的主要数据结构与算法 11](#_Toc10725)

[3.2.3.1. 主要数据结构：抽象语法树 11](#_Toc14976)

[3.2.3.2. 主要算法：建立语法树节点算法 11](#_Toc23764)

[3.2.4. 采用的自动生成技术为Bison自动生成工具 13](#_Toc3098)

[3.2.5. 错误处理函数 15](#_Toc17269)

[3.3. 三地址代码生成器 15](#_Toc30224)

[3.3.1. 语法制导定义 15](#_Toc7085)

[3.3.2. 算法的基本思想 15](#_Toc23611)

[3.3.3. 实验结果 15](#_Toc2389)

[3.3.3.1. 第一次测试 16](#_Toc27882)

[3.3.3.2. 第二次测试 16](#_Toc7152)

[3.3.3.3. 符号表 17](#_Toc29249)

[4. 实验体会 18](#_Toc13076)

# 组内分工与贡献介绍

本小组在实验过程中，由组长韩柳彤同学进行总体实验设计。并在组长的带领下，各组员按实验内容进行分工，各自完成自己的实验内容并实现功能。具体分工如下：

1. 韩柳彤：词法分析器标识符、关键词、各进制数字部分分析与实现；语法分析器逻辑部分
2. 张彦博: 语法分析器运算部分
3. 王时予: 词法分析器算符部分 各进制数转换
4. 丁天乐: 使用LEX实现词法分析器
5. 钱子牛: 重写词法分析器接口

# 系统功能概述（包括系统的总体结构）

系统总体由词法分析、语法分析、三地址码生成和错误识别与处理四部分组成。实现了对输入的代码字符串经过词法分析、语法分析与语义分析的过程生成代码段所对应的三地址码，并对错误代码具有一定的识别与纠错能力。

## 词法分析器

词法分析器实现了识别十进制、八进制、十六进制整数、标识符、主要运算符和主要关键字等功能。词法分析器可以从文件中读入代码，并将解析的结果输出到对应的结果文件中（支持多文件输入输出），并根据解析的结果建立相应的符号表，结果存入符号表文件中。

## 语法分析器

语法分析器实现了根据词法分析器分析的结果对代码段的语法进行识别，并根据产生式进行归约并建立抽象语法树。最后输出代码段最左派生的产生式序列。实现了多文件输入输出。同时在语法分析器中实现了一定的错误识别与处理功能，能够判断出代码段中的错误种类，并实现续编译功能。

## 语义分析与三地址码生成

语义分析及三地址码生成器在实现前两部分功能的同时，针对读入的代码段生成相应的三地址码，并将结果输出至文件当中。

用户在linux系统下，通过命令行启动程序，在启动程序时不输入参数则默认从控制台读入代码；可输入多个参数分别为存有代码的文件名称。

# 包含实验指导书要求的内容分系统报告

## 词法分析子系统

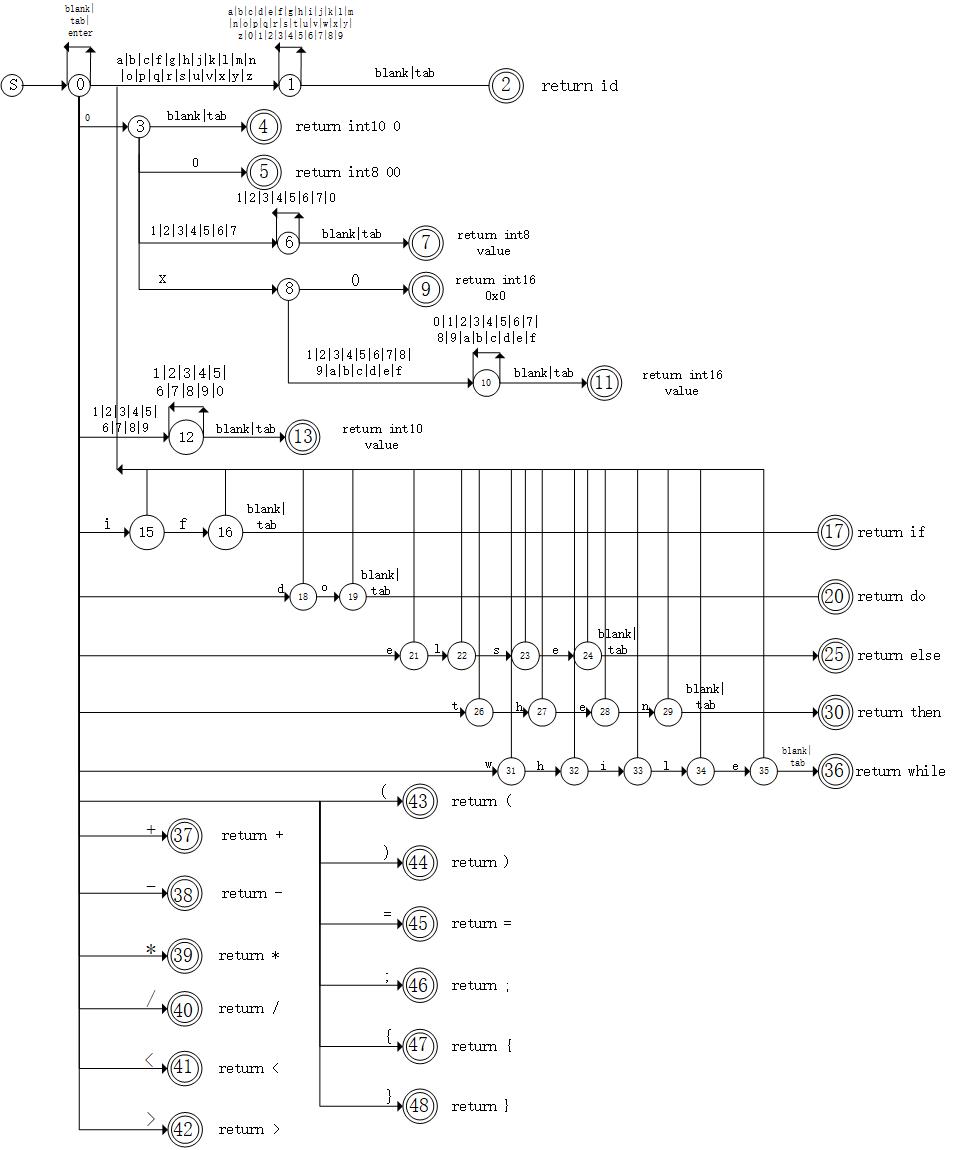
### 词法的正规式描述

1. 标识符 <字母>(<字母>|<数字字符>)\*
2. 十进制整数 0 | (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)\*
3. 八进制整数 0(1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)\*
4. 十六进制整数 0x(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)\*
5. 运算符和分隔符 + - \* / > < = ( ) ；
6. 关键字 if then else while do

### 变换后的正规文法



### 状态图



### 词法分析程序的主要数据结构与算法

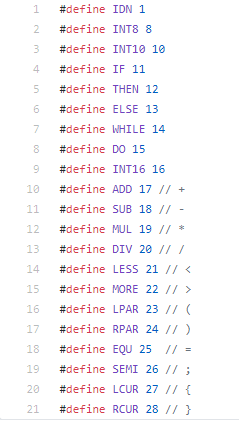
#### 实验内容

选择适当的方法（自行设计、使用Lex等自动生成工具），设计实现一个能够分析三种整数、标识符、主要运算符和主要关键字的词法分析程序。

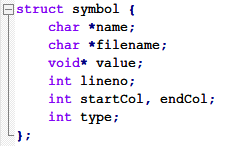
#### 实现功能

1. 识别八进制、十进制、十六进制整数。
2. 将识别的八进制、十六进制数转化为十进制数输出。
3. 识别关键字、运算符和分隔符。
4. 识别标识符。
5. 构造简单的符号表并打印。
6. 简单的错误处理功能。

#### 接口设计



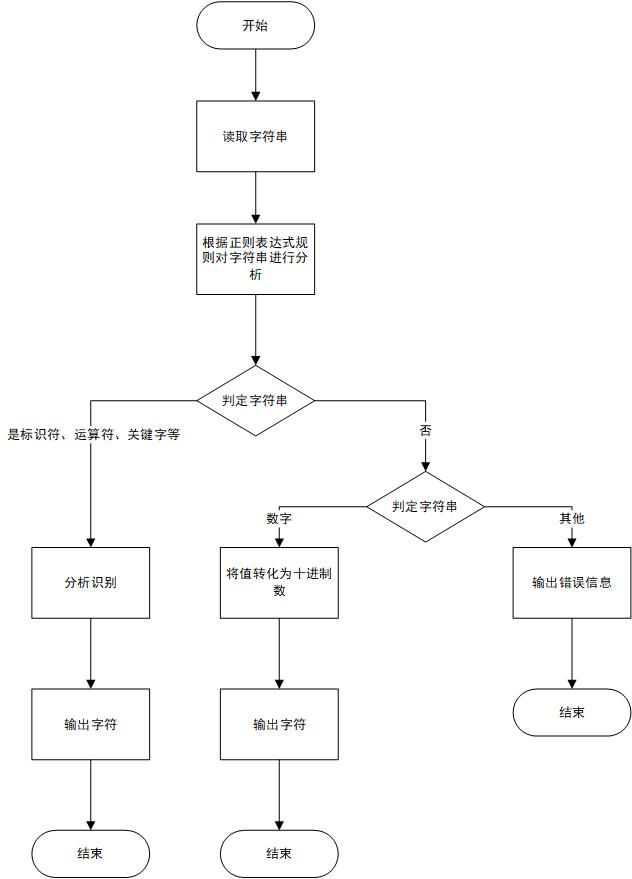
1. 定义结构体存储单个符号相关信息：



1. 符号表采用哈希表实现：



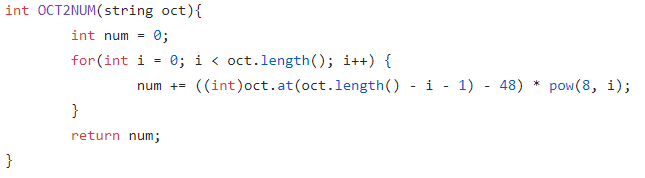
#### 流程图



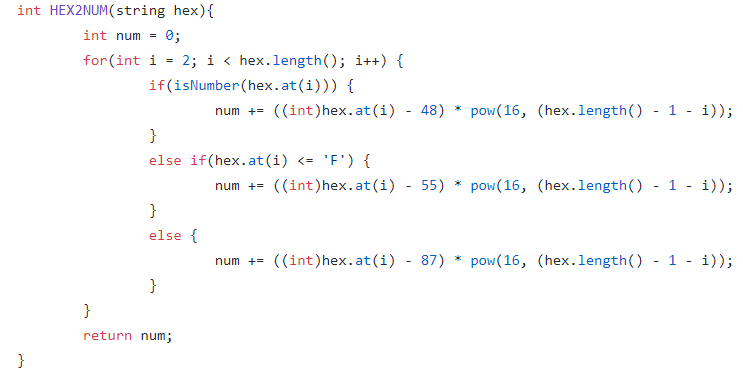
#### 实现方法

定义NUM,IDN\_NAME为分别表示数字和标识符属性的全局变量：

C:\Users\qzn\AppData\Local\Temp\WeChat Files\b01070894e8b9e551f123dcea446823.png

完成八进制转十进制的主函数：

完成十六进制转十进制的主函数：



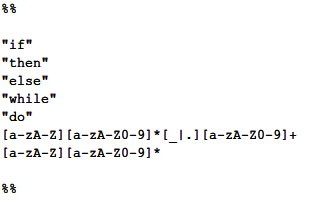
//待添加LEX的实现方式

采用Flex自动生成工具：flex会根据用户定义的正则表达式规则识别token，每识别到一个token后会执行用户事先写好的对应的一系列动作。

1. Flex采用的技术：根据输入的正则表达式构造DFA
2. 解决二义性问题：如：对于if，既可以匹配IF关键字，又可以匹配IDN标识符

Flex解决方式：

1. 词法分析器匹配输入是匹配尽可能多的字符串
2. 如果两个模式都可以匹配的话，匹配在程序中更早出现的模式



#### 实现思路

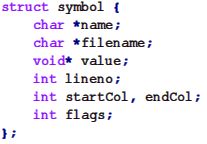
1. 读取运行参数，区别从终端读入数据或从文件读入数据，分别进行相应准备操作。
2. 读取字符串，根据输入正则表达式规则对输入字符串进行词法分析，并执行下列动作：
3. 若是关键字、运算符、括号等，识别后直接向目标文件中输出识别信息。
4. 若是数字，则识别后将其值转化为十进制数后输出至目标文件中。
5. 若是标识符，识别后向符号表中添加该标识符的信息，而后将识别信息输出到文件中。
6. 若是其他未定义情况，输出错误信息。

#### 主要数据结构与算法

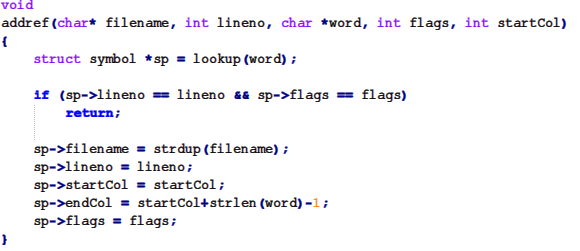
1. 识别词法并输出



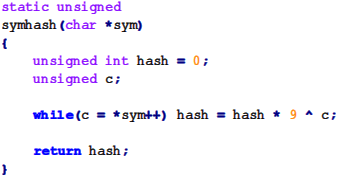
1. 符号表



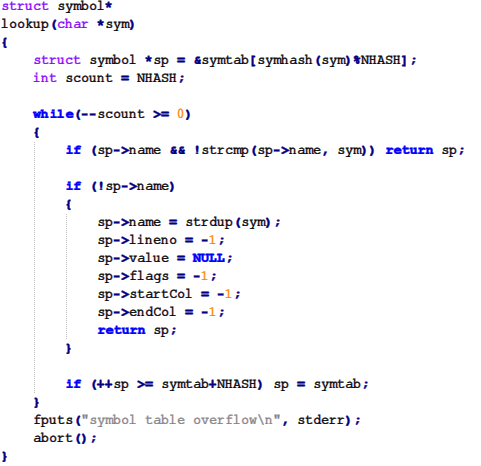
1. 向符号表中添加符号



1. 哈希值的计算



1. 查找符号在符号表中的位置，如果不在则想符号表中申请空间给这个符号

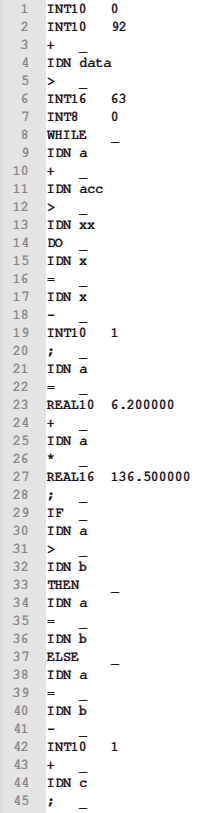
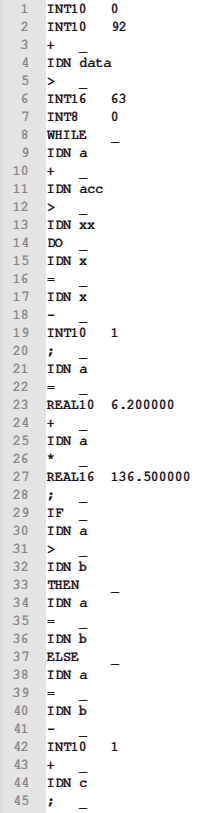


### 实验结果

1. 实验数据

实验数据

1. 实验结果

由实验结果截图可见，本组使用Lex等自动生成工具，设计完成了一个词法分析子系统，且该系统能够分析三种整数、标识符、主要运算符和主要关键字，即可完成实验要求。

## 语法分析子系统

### 语法分析方法的描述

采用Bison自带的语法分析方法LALR(1)，然后构建语法树，根据抽象语法树按照最左派生的顺序输出产生式序列。

### 语法分析子系统结构

前序遍历语法树并输出

生成语法树

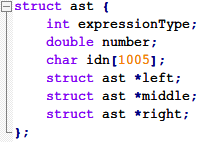
对错误进行定位、输出、续编译等操作

Bison最左规约匹配产生式

词法分析结果

### 语法分析子系统的主要数据结构与算法

#### 主要数据结构：抽象语法树



#### 主要算法：建立语法树节点算法

//建立节点的函数，特用于产生式S->id = E

struct ast \*newastwithidn(int expressionType, struct ast \*idn, struct ast \*l, struct ast \*m, struct ast \*r)

{

struct ast \*a;

a = (struct ast\*)malloc(sizeof(struct ast));

a->expressionType = expressionType;

a->number = -1;

strcpy(a->idn, idn->idn);

a->left = l;

a->middle = m;

a->right = r;

return a;

}

//一般建立节点的函数

struct ast \*newast(int expressionType, struct ast \*l, struct ast \*m, struct ast \*r)

{

struct ast \*a;

a = (struct ast\*)malloc(sizeof(struct ast));

a->expressionType = expressionType;

a->number = -1;

memset(a->idn, 0, sizeof(a->idn));

a->left = l;

a->middle = m;

a->right = r;

return a;

}

struct ast \*newnum(int expressionType, double number)//建立数据型节点的函数

{

struct ast \*a;

a = (struct ast\*)malloc(sizeof(struct ast));

a->expressionType = expressionType;

a->number = number;

memset(a->idn, 0, sizeof(a->idn));

a->left = NULL;

a->middle = NULL;

a->right = NULL;

return a;

}

struct ast \*newidn(int expressionType, char \*idn)//建立idn节点函数

{

struct ast \*a;

a = (struct ast\*)malloc(sizeof(struct ast));

a->expressionType = expressionType;

a->number = -1;

strcpy(a->idn, idn);

a->left = NULL;

a->middle = NULL;

a->right = NULL;

return a;

}

void displayAST(struct ast\* root) //以最左派生顺序输出结果

{

if (root == NULL)

{

printf("root is NULL.\n");

return;

}

fprintf(f, "%s", checkProcedure(root->expressionType));

if (root->number != -1)

{

if (root->expressionType >= 20 && root->expressionType <= 22)

fprintf(f, "\t%d", (int)root->number);

else if (root->expressionType >= 23 && root->expressionType <= 25)

fprintf(f, "\t%lf", root->number);

}

if (strlen(root->idn) > 0)

fprintf(f, "\t%s", root->idn);

fprintf(f, "\n");

if (root->left != NULL)

displayAST(root->left);

if (root->middle != NULL)

displayAST(root->middle);

if (root->right != NULL)

displayAST(root->right);

}

### 采用的自动生成技术为Bison自动生成工具

Bison通过获得flex分析出的token以及用户定义的产生式进行归约，每当根据一条产生式进行归约时，执行用户预先定义好的一系列动作。

其中，Bison核心代码如下：

%%

P: L { $$ = newast(1, NULL, $1, NULL); root = $$; }

| L P { $$ = newast(2, $1, NULL, $2); root = $$; }

;

L: S SEMICOLON { $$ = newast(3, NULL, $1, NULL); }

| S error SEMICOLON { yyerrok; }

| ERROR { yyerror("syntax error, undefined identifier."); yyerrok; }

;

S: I EQUAL E { $$ = newastwithidn(4, $1, NULL, $3, NULL); }

| IF C THEN S ELSE S { $$ = newast(6, $2, $4, $6); }

| IF C THEN S %prec LOWER\_THAN\_ELSE { $$ = newast(5, $2, $4, NULL); }

| WHILE C DO S { $$ = newast(7, $2, NULL, $4); }

| LBR P RBR { $$ = newast(8, NULL, $2, NULL); }

| error EQUAL E { yyerrok; }

| I EQUAL error E { yyerrok; }

| I error { yyerrok; }

| I error EQUAL E { yyerrok; }

| error EQUAL error { yyerrok; }

| IF error C THEN S { yyerrok; }

| IF C error THEN S { yyerrok; }

| IF C THEN error S { yyerrok; }

| IF error THEN S { yyerrok; }

| IF C THEN S ELSE error S { yyerrok; }

| WHILE error C DO S { yyerrok; }

| WHILE C error DO S { yyerrok; }

| WHILE C DO error S { yyerrok; }

;

C: E GREATER E { $$ = newast(9, $1, NULL, $3); }

| E LESS E { $$ = newast(10, $1, NULL, $3); }

| E EQUAL E { $$ = newast(11, $1, NULL, $3); }

| E error GREATER E { yyerrok; }

| E GREATER error E { yyerrok; }

| E error LESS E { yyerrok; }

| E LESS error E { yyerrok; }

| E error EQUAL E { yyerrok; }

| E EQUAL error E { yyerrok; }

| error GREATER E { yyerrok; }

| E GREATER error { yyerrok; }

| error LESS E { yyerrok; }

| E LESS error { yyerrok; }

| error EQUAL E { yyerrok; }

| E EQUAL error { yyerrok; }

| error GREATER error { yyerrok; }

| error LESS error { yyerrok; }

| error EQUAL error { yyerrok; }

;

E: E ADD T { $$ = newast(12, $1, NULL, $3); }

| E SUB T { $$ = newast(13, $1, NULL, $3); }

| T { $$ = newast(14, NULL, $1, NULL); }

| E ADD error T { yyerrok; }

| E SUB error T { yyerrok; }

;

T: T MUL F { $$ = newast(16, $1, NULL, $3); }

| T DIV F { $$ = newast(17, $1, NULL, $3); }

| T MUL I { $$ = newast(16, $1, NULL, $3); }

| T DIV I { $$ = newast(17, $1, NULL, $3); }

| F { $$ = newast(15, NULL, $1, NULL); }

| I { $$ = newast(15, NULL, $1, NULL); }

| T MUL error F { yyerrok; }

| T DIV error F { yyerrok; }

| T MUL error I { yyerrok; }

| T DIV error I { yyerrok; }

;

F: LB E RB { $$ = newast(18, NULL, $2, NULL); }

| OREAL { $$ = newnum(23, yylval.value); }

| HREAL { $$ = newnum(25, yylval.value); }

| DREAL { $$ = newnum(24, yylval.value); }

| OINT { $$ = newnum(20, yylval.value); }

| HINT { $$ = newnum(22, yylval.value); }

| DINT { $$ = newnum(21, yylval.value); }

| LB E error { yyerrok; }

;

I: IDN { $$ = newidn(19, yylval.string); }

;

%%

### 错误处理函数

extern

void yyerror(const char \*s) {

errflag = 1;

int start = yylloc.first\_column;

int end = yylloc.last\_column;

int i;

fprintf(errorFile, "Error: %s on Line: %d:c%d to %d:c%d\n", s, yylineno, start, yylineno, end);

}

## 三地址代码生成器

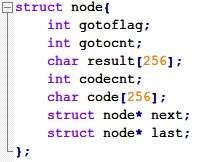
### 语法制导定义

通过将属性与文法符号关联、将语义规则与产生式关联来描述语言结构的翻译方案。对应每一个产生式编写一个语义子程序，当一个产生式获得匹配时，就调用相应的语义子程序来实现语义检查与翻译。

### 算法的基本思想

采用深度优先搜索的方法索搜在语义分析中构建的抽象语法树，自底向上递归生成代码，并根据产生式规则进行链接。实现的思想是用链表存储代码序列，每一个节点只存储一条语句。

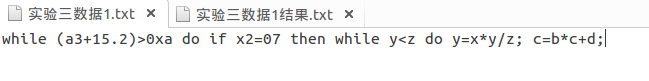
链表的定义：



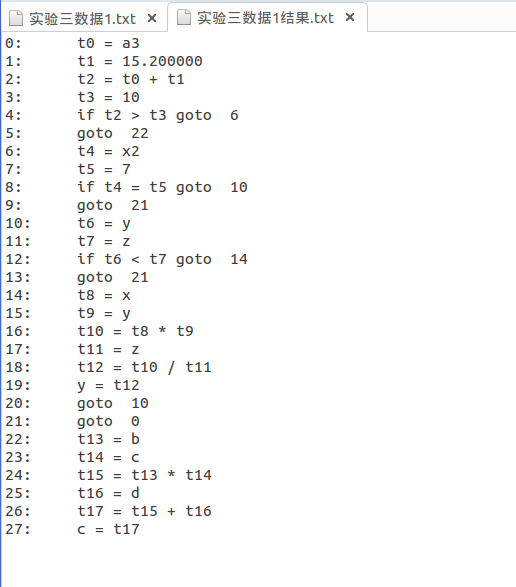
### 实验结果

#### 第一次测试

1. 输入数据

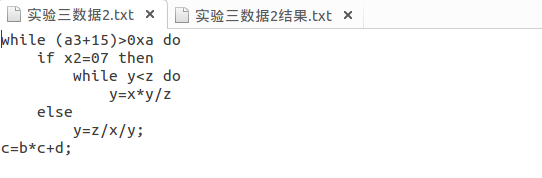


1. 输出数据

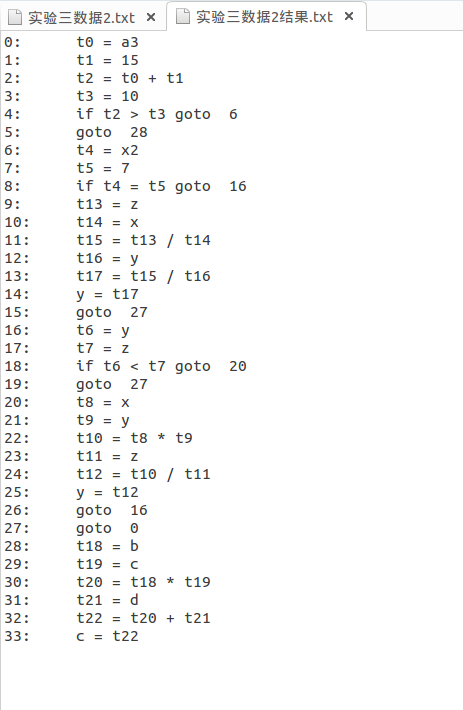


#### 第二次测试

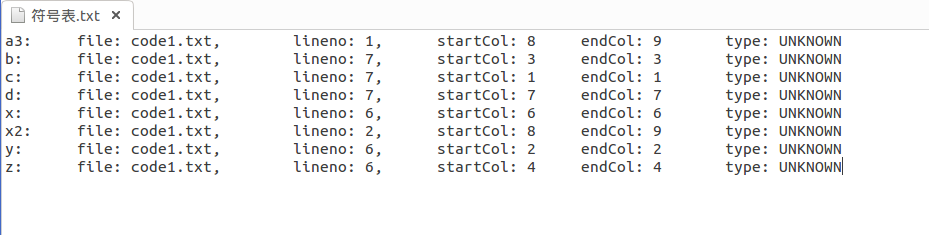
1. 输入数据



1. 输出数据



#### 符号表



# 实验体会

通过这次实验，使组内成员对编译程序、编译器的实现原理有了更深刻的认识。

在实验初期，虽然对课内知识有了一定的了解，但是在设计编译程序的时候显得有些不知所措，经过一次又一次激烈的讨论，才对最后的实验方案有了一个总体的设计。在具体的实现过程中也遇到了很多的问题，也曾一度陷入迷茫，但最后通过查阅资料，组内讨论等方法将问题解决。

编译原理实验不仅是加深了我们对编译课程上学到的知识的理解，同时还锻炼了我们c语言编程、数据机构与算法等方面的能力。除此之外，编译原理实验还加强了我们的团队合作能力，在团队讨论的过程中难免会有分歧，通过组员们之间的密切交流、阐述自己的思路与想法最终也消除了分歧。

最重要的是，通过这次实验让我们对词法分析、LALR(1)分析法、抽象语法树等知识有了切实地深入地认识。

在词法分析的具体实现过程中关于如何切分单词，我们一致认为不能通过空格来切分，因为在程序中分割单词的不仅有空格，还有+, -, \*, /等运算符，所以不能单单通过空格来判断。通常实现词法分析都是通过递归程序实现状态转换图来完成的，但是递归程序非常影响程序的效率，所以我们组认为使用栈来非递归的实现状态转换图。

在实现语法分析的过程中，产生式的定义非常重要，因为在四则运算中涉及到算符优先性的问题，所以可以通过强行指定算符优先级的办法或者通过增加语法范畴增加产生式的方法定义算符的优先级。

在生成三地址码的过程中，可以通过采用递归子程序的方法直接输出代码，而不需要采用数据结构来存储。通过对按照顺序输出的每一行代码标记行号，并在跳转指令中保存相对跳转行数的办法可以实现代替三地址码序列中的标号的功能。

总而言之，通过这次实验使我们都收获良多。

最后，感谢蒋宗礼老师本学期的耐心讲解与指导！老师辛苦了！

2016年6月 第十五周