**2022-2023学年第2学期**

**编译原理I**

**实验报告**

工作量认定矩阵

说明：

1. 小组每一名成员对自己和其他成员的工作量（百分比）进行评价，并亲笔签字（或电子签字）。
2. 每小组参加验收检查时，上交报告封面和成绩评定表。报告全文仅提交电子版、不需要提交打印版。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 被评价人姓名  评价人 | | 王怡宁 | 周子渔 | 向鑫睿 | 评价人签字 |
| 学号 | 姓名 |
| 20074213 | 王怡宁 | 90 | 88 | 91 |  |
| 20074206 | 向鑫睿 | 89 | 90 | 88 |  |
| 20074214 | 周子渔 | 90 | 88 | 90 |  |

本组的Gitee/GitHub项目地址： https://github.com/zzy020306/compiler-exp

|  |
| --- |
|  |

详细分工表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | **实验一** | **实验二** | **实验三** |
| 王怡宁 | 编写定义规则部分代码。对规则部分要引用的文件和变量进行说明，通常可包含头文件、常数定义、全局变量定义、外部变量定义以及正规式定义等。与组员设计的其他部分进行联调，确保词法分析能够正常进行。设计测试用例，修改规则定义中不合理或有歧义的部分。 | 编写了一个语法分析的递归子程序算法。该算法通过设置一个特殊变量findTerminal，来搜索输入字符串str中的终结符。还设置了一个打印树的函数，它会根据树的宽度输出每一行的节点，以完成分析树的构造。 | 统筹规划三地址代码生成器的代码结构，对代码进行调试和优化。着重编写结构体定义部分，与组员商议定义具体格式。编写测试用例，并手动翻译三地址代码以检测程序设计是否正确。与周子渔同学共同修改scan()函数中出现的错误。 |
| 周子渔 | 编写辅助函数部分代码。定义了yylex()函数和main()函数。能够识别出一个标记并返回其种别值。编写main函数，创造程序的入口，针对不同的输入值返回种别值并打印。设计测试用例，与组员的编写的代码进行联调。 | 成功处理了给定的文法，并进行了消除左递归和提取左因子的操作。编写测试用例，检查语法分析是否正确，检查语法生成树是否设计正确，检查语法树缩进是否合理。与向鑫睿同学合作完成部分递归子程序的逻辑构建，代码优化等工作。 | 编写了一个词法分析程序来实现以下功能：跳过空白字符、识别移进的数字类型、识别关键字。识别运算符，程序与已定义的运算符进行比对，并返回相应的token。与组员共同编写测试用例，对三地址代码生成器的代码进行调整，确保其能进行正确的分析。 |
| 向鑫睿 | 编写规则识别部分代码。为每种标记编写正则表达式，并返回匹配成功的对应标记。与组员的编写的代码进行连接。设计测试用例，验证正则表达式的正确性。 | 为每条改写后文法的产生式绘制语法图，提供递归子程序算法设计思路。为每条改写后文法的产生式编写递归子程序。与周子渔同学合作完成部分递归子程序的逻辑构建，代码优化等工作。 | 设计编译器整体架构，对每条产生式进行语法制导翻译。对每个条件控制的布尔表达式绘制逻辑框图。为各条产生式编写翻译代码。与组员共同编写测试用例，手动绘制测试用例语法树，验证代码逻辑性。 |

目录

[实验一 1](#_Toc105105398)

[实验二 6](#_Toc105105399)

[实验三 16](#_Toc105105400)

## 实验一 词法分析程序的设计与实现

* + 1. **实验目的：**

基本掌握计算机高级程序设计语言的词法分析程序的开发方法。

* + 1. **实验内容：**

选择适当的方法（本实验所使用的是Flex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。

* + 1. **实验要求：**
    2. 根据以下的正规式：

标识符 <字母>(<字母>|<数字字符>)\*

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)\*

八进制整数 0(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)\*

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)\*

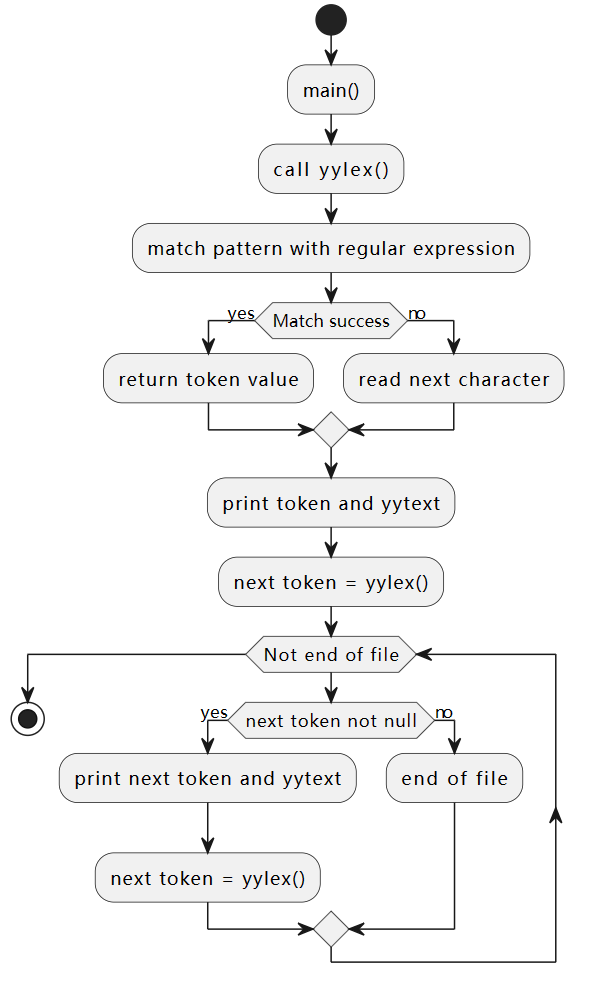
运算符和分隔符 + - \* / > < = ( ) ；

关键字 if then else while do

请给出非法八进制数和非法十六进制数的正规式。非法八进制数的描述为：以“0”打头，后跟由“0”到“9”组成的长度大于等于 1 且含有“8”或“9”的串。非法十六进制数的描述为：以“0x”或“0X”打头，后跟由“0”到“9”、“a”到“z”、“A”到“Z”组成的长度大于等于 1 且含有“g”到“z”、“G”到“Z”的串。

1. 根据状态图，设计词法分析函数（本实验利用Flex自动生成器生成），完成以下功能：
2. 从键盘读入数据，分析出单词。
3. 返回单词种别（用整数表示），
4. 返回单词属性（不同的属性可以放在不同的全局变量中）。
5. 编写测试程序，反复调用函数，输出单词种别和属性。
   * 1. **实验环境：**

Windows操作系统。

* + 1. **实现方法:**
       1. **整体设计**

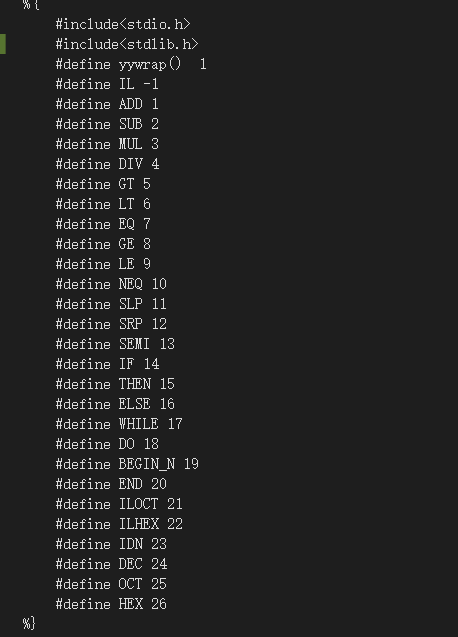
首先"main" 函数开始执行，调用 "yylex" 函数进行词法分析。"yylex" 函数尝试匹配正则表达式规则，如果匹配成功，就返回相应的标记值。然后打印出这个标记值和对应的文本（yytext）。最后重复以上步骤，直到所有的输入都被处理完。

* + - 1. **定义部分**

在定义部分中，对规则部分要引用的文件和变量进行说明，通常可包含头文件、常数定义、全局变量定义、外部变量定义以及正规式定义等。本代码中定义了C语言的的两个头文件stdio.h和stdlib.h，自定义的yywrap()函数，返回值为1，以及本实验所设计的所有标识符，不同进制整数，运算符和关键字的种别值的全局变量定义，方便于以后进行判定。正规式定义用来定义在规则部分引用的正规式，类似于C语言中的宏定义，所以也称宏定义。

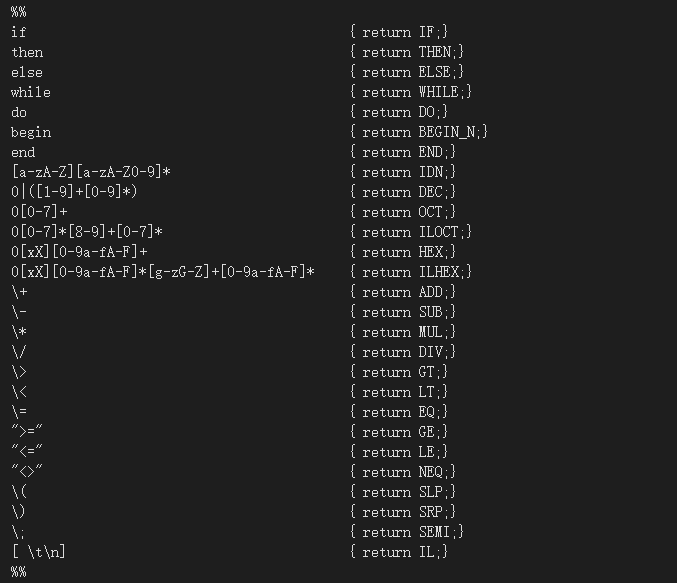
这部分代码主要是定义了一些标记常量（tokens），每一个常量代表一种特定的标记，比如ADD代表加号，SUB代表减号，MUL代表乘号，DIV代表除号等。此外还定义了各种关键字，如IF，THEN，ELSE等。这些标记常量都被赋予了一个唯一的整数值，用于在后续的词法分析过程中表示识别出的单词的种别。例如，"ADD" 代表加号，其种别值为1，"SUB" 代表减号，其种别值为2，等等。

除了宏定义外，定义部分的其他代码需要用“%{”和“%}”括起来，LEX会将这部分内容直接复制到C文件生成的lex.yy.c文件中。在本实验中具体实现如下：



* + - 1. **识别规则部分：**

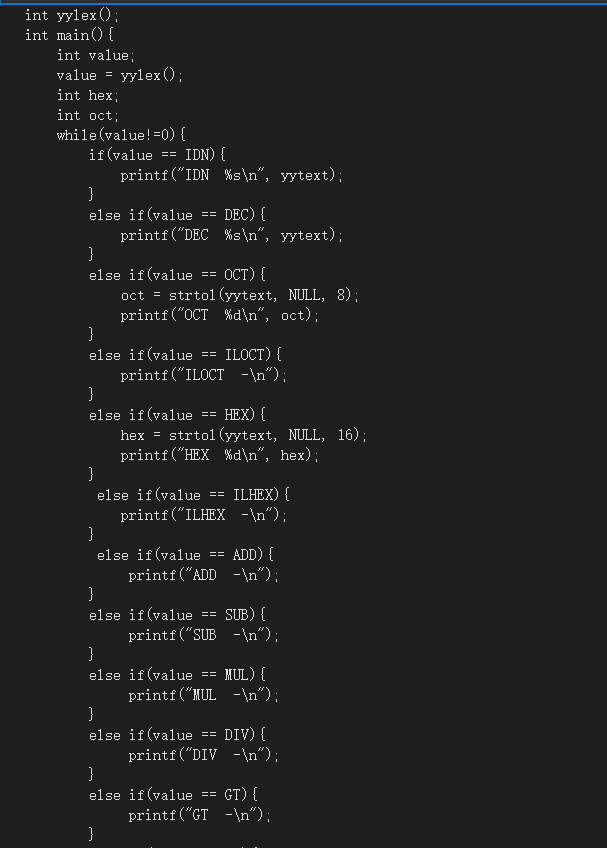
这部分代码是Flex规则部分，每一行代表一个词法规则。规则的左边是一个正则表达式，用于匹配输入文本中的字符序列，右边是当匹配成功时要执行的C代码。例如，[a-zA-Z][a-zA-Z0-9]\* 匹配任何以字母开头的字母数字序列，当匹配成功时，返回标记 IDN。这些规则告诉词法分析器在遇到什么样的字符序列时，应该识别为什么样的标记，并执行何种操作。



其具体实现方法是，遇到定义的正规式，返回之前定义的全局变量的值，其中不同进制的整形表示用正则表达式表示出来。

* + - 1. **辅助函数部分**

在识别规则部分中如果用到的函数不是库函数，则需要在这部分给出函数定义，由用户用C语言编写，它们会由LEX系统直接复制到输出的C程序文件中。本代码所编写的函数部分截图如下：



这部分定义了yylex()函数和main()函数。yylex()函数是Flex自动生成的，用于执行词法分析过程，识别出一个标记并返回其种别值。

yylex()函数，表示从该函数开始分析。在main函数内，定义value表示之前宏观定义过的种别值，value的值由yylex函数给出，之后又定义了hex和oct存放对应的十六进制和八进制数据。

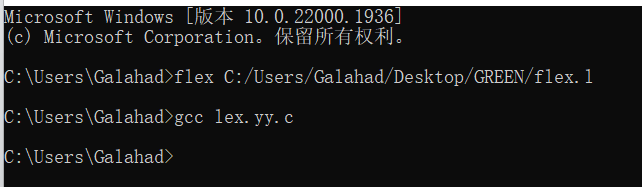
定义循环，只要value的值不是0（0定义的是没有字符进来），进行if判定，若是IDN标识符，则调用yytext指向识别规则中的一个正规式匹配的单词的首字符，打印出来。若是十六进制和八进制，则调用strtol把参数str所指向的字符串根据给定的base转换为一个长整数，字符串中数字的提取，然后打印出来。若是其他情况则根据接受的宏观定义的值，打印出相应的数字。其他的情况都是用类似if语句判断来进行判断。

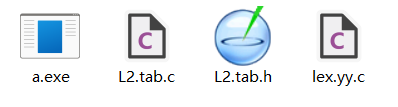
值得注意的是，yytext 是一个全局变量，它包含了最后一个被词法分析器匹配的单词的文本。例如，如果输入var123，并且这个词被识别为一个标识符，那么 yytext 就会包含 var123 这个字符串。这样就可以对源代码进行词法分析，并打印出每一个识别出的单词的种别值和属性值了。

* + 1. **效果展示：**

1. **运行FLEX**

打开命令提示符；输入：flex C:/Users/Galahad/Desktop/GREEN/flex.l生成lex.yy.c文件，gcc该文件生成a.exe可执行程序。



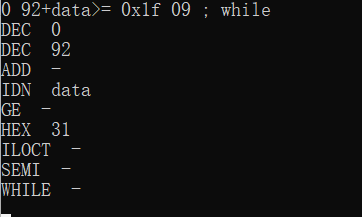


运行a.exe便可以运行语法分析程序。

1. **词法分析程序运行展示：**

实例：0 92+data>= 0x1f 09 ; while

效果展示：



**实验二 语法分析与语法制导的三地址代码生成程序**

* 1. **实验目的**

掌握计算机语言的语法分析程序设计与属性文法应用的实现方法。

* 1. **实验内容**

编制一个能够进行语法分析并生成三地址代码的微型编译程序。

* 1. **实验要求**
  2. 考虑下述语法制导定义中文法，采用递归子程序法，改写文法，构造语法分析程序；

|  |  |
| --- | --- |
| 产生式 | 语义规则 |
| P → L | P.code = L.code |
| P → L P1 | P.code = L.code || P1.code |
| L → S ; | L.code = S.code |
| S → id = E | S.code = E.code || gen(id.place ‘:=’ E.place) |
| S → if C then S1 | C.true = newlabel;  C.false = S.next;  S1.next = S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code |
| S → if C then S1 else S2 | C.true = newlabel;  C.false = newlabel;  S1.next = S2.next =S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code  || gen(‘goto’ S.next)|| gen(C.false  ‘:’)  || S2.code; |
| S → while C do S1 | S.begin = newlabel;  C.true = newlabel;  C.false = S.next;  S1.next = S.begin;  S.code = gen(S.begin ‘:’) || C.code  || gen(C.true ‘:’) || S1.code  || gen(‘goto’ S.begin); |
| C → E1 > E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘>’ E2.place ‘goto’  C.true) || gen(‘goto’C.false) |
| C → E1 < E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 = E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 >= E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘>=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 <= E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 <> E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<>’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| E → E1 + T | E.place = newtemp;  E.code = E1.code || T.code||  gen(E.place ‘:=’ E1.place ‘+’ T.place) |
| E → E1 - T | E.place = newtemp;  E.code = E1.code || T.code ||  gen(E.place ‘:=’ E1.place ‘-’ T.place) |
| E → T | E.place = T.place;  E.code = T.code |
| T → F | T.place = F.place;  T.code = F.code |
| T → T1 \* F | T.place = newtemp;  T.code = T1.code || F.code ||  gen(T.place ‘:=’ T1.place ‘\*’ F.place) |
| T → T1 / F | T.place = newtemp;  T.code = T1.code || F.code ||  gen(T.place ‘:=’ T1.place ‘/’ F.place) |
| F → ( E ) | F.place = E.place;  F.code = E.code |
| F → id | F.place = id.name;  F.code = ‘’ |
| F → OCT | F.place = OCT.name;  F.code = ‘’ |
| F → DEC | F.place = DEC.name;  F.code = ‘’ |
| F → HEX | F.place = HEX.name;  F.code = ‘’ |

* 1. **实验环境**

PC微机

Windows 操作系统

Visual Studio C++ 程序集成环境

* 1. **实验主要步骤**

1. 对给定的文法，消除左递归，提取左因子。

改写前的文法：

P → L |L P

L → S ;

S → id = E|if C then S |if C then S1 else S2|while C do S|

C → E > E | E < E | E = E | E <= E | E >= E | E <> E

E → E + T | E - T | T

T → T \* F | T / F | F

F → ( E ）| id | OCT | DEC | HEX

* 1. 消除左递归：

P → L P1

P1 → P1 | ε

T → F T’

T’→ \* FT’ | / FT’| ε

E → T E’

E’→ + TE’ | - TE’ | ε

* 1. 提取左因子：

C → EC1

C1 → > E | < E | = E | <= E | >= E | <> E

S → id = E|if C then S S1 |while C do S|

S1 → else S

改写后的文法：

P → L P1

P1 → P1 | ε

L → S ;

S → id = E|if C then S S1 |while C do S|

S1 → else S

C → EC1

C1 → > E | < E | = E | <= E | >= E | <> E

T → F T’

T’→ \* FT’ | / FT’| ε

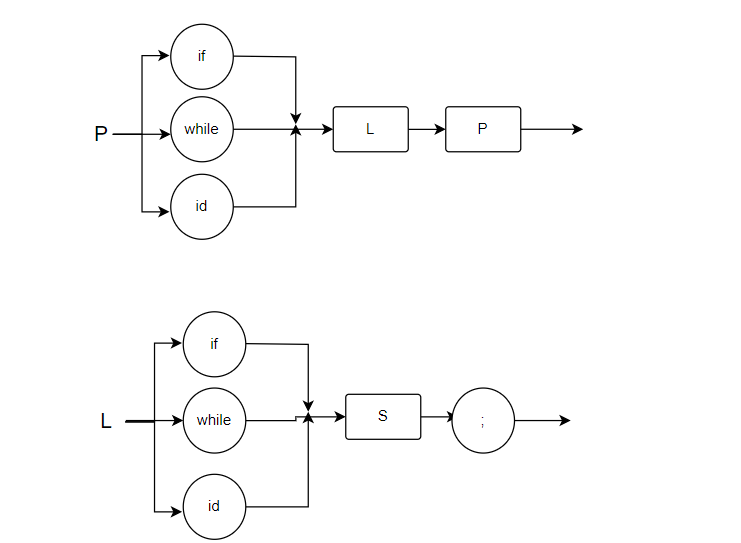
E → T E’

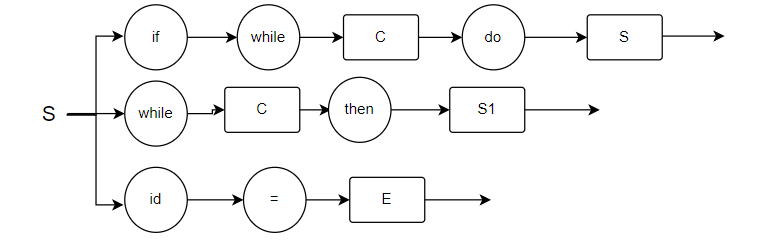
E’→ + TE’ | - TE’ | ε

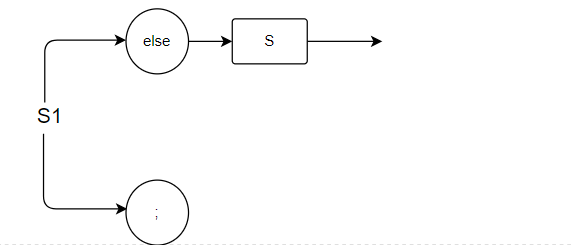
F → ( E ）| id | OCT | DEC | HEX

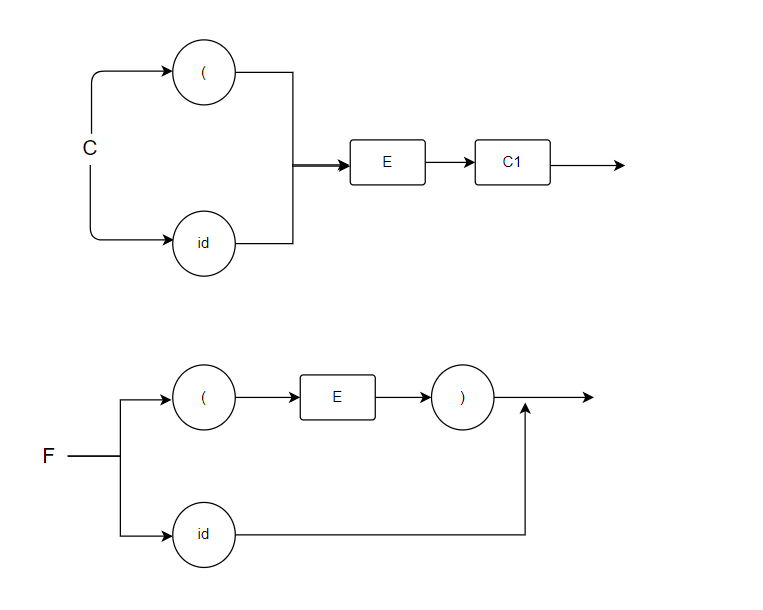
1. 编制并化简语法图

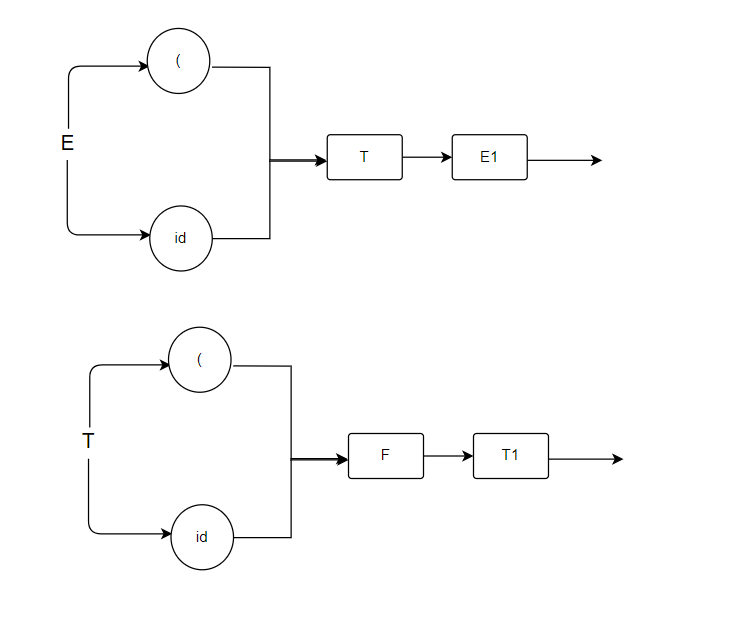
化简后的语法图：

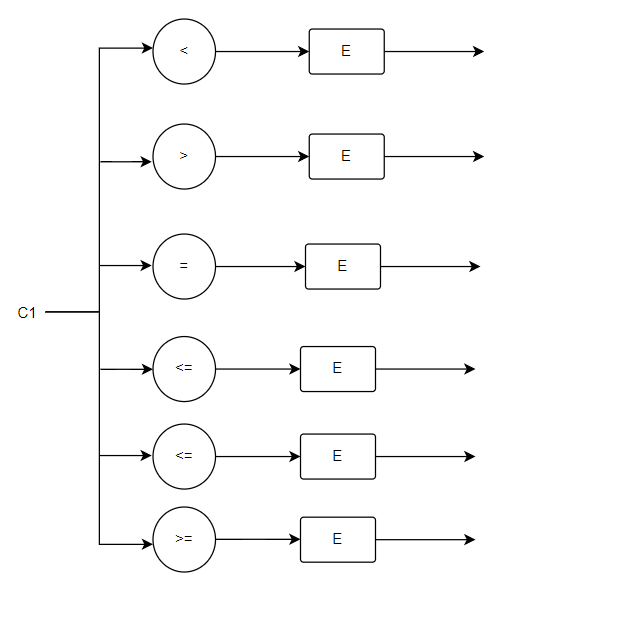


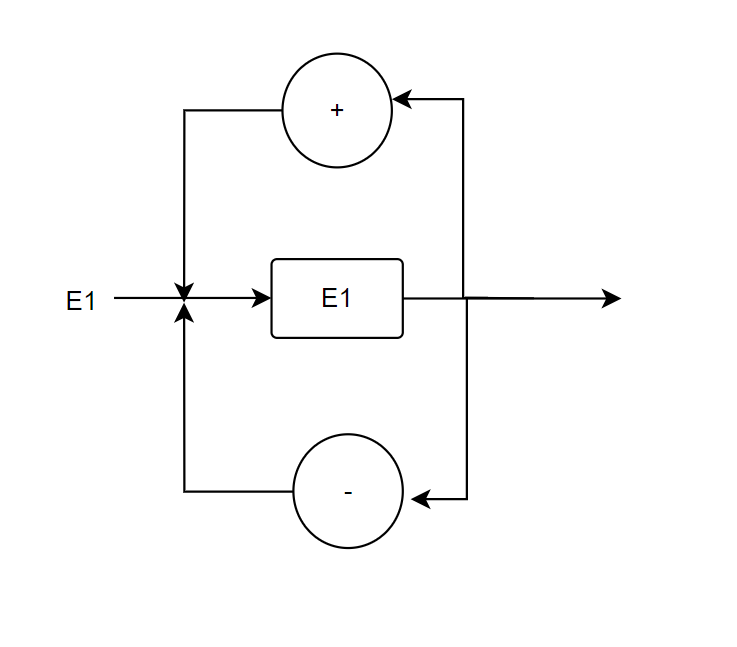


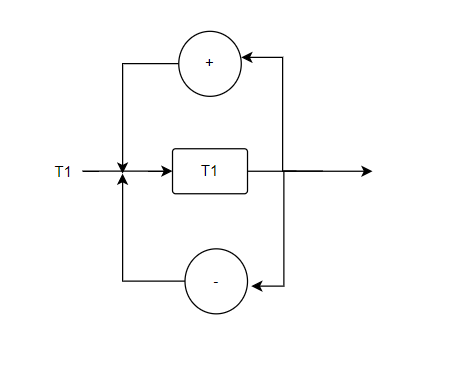












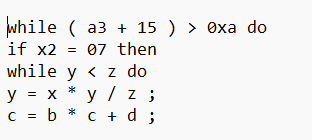
1. 编制递归子程序的算法

自顶向下的递归调用子程序

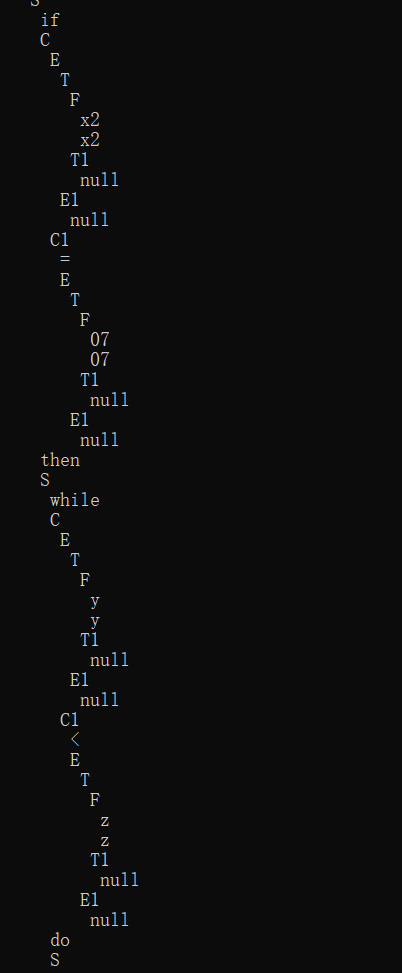
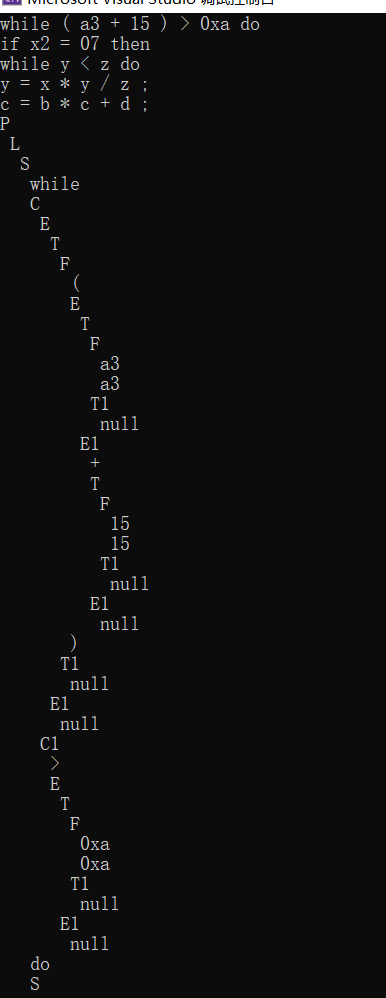
产生式A → X1 X2...Xk...Xn

1. 算法中设置当前寻找终结符变量findTerminal，输入字符串str，树的宽度treeWidth，定义各个状态函数，输入参数为treeWidth。
2. 设置打印树函数，根据树的宽度，输出每一行树的的节点，完成分析树的的构造。
3. 按照候选式的FIRST集识别终结符。如针对文法P → L则计算L的First集{id,while,if}。如果进入终结符匹配这其集合里终结符相匹配则进入P的算法中。
4. 与A对应的处理子程序遇到Xk是终结符时直接进行匹配，findTerminal+1，构造分析树打印出来。
5. 遇到Xk是语法变量时就调用与Xk对应的处理子程序，输入参数为当前treeWidth+1。
6. 当递归函数遇到终结符输出的时候且向下在没有可以递归调用的非终结符时，向上回溯，直到每个终结符都被输出完毕。
7. 当遇到非法输出的时候，报错输出error。
8. 当所有终结符被输出完毕的时候，语法分析树构造成功，输出correct
   1. **效果展示**

输入语法：



输出语法分析树：



# 实验三 语法制导的三地址代码生成

### 实验目的

掌握计算机语言的语法分析程序设计与属性文法应用的实现方法。

### 实验内容

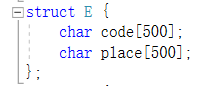
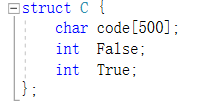
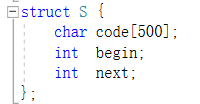
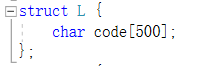
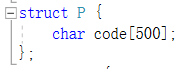
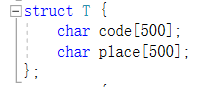
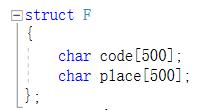
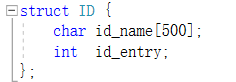
我们组采用LL分析法，设计、实现了一个能够进行三地址代码生成的微型编译程序。

程序主要包含两大部分：1.词法分析程序。2.语法制导编译。

产生式如下：

|  |  |
| --- | --- |
| 产生式 | 语义规则 |
| P → L | P.code = L.code |
| P → L P1 | P.code = L.code || P1.code |
| L → S ; | L.code = S.code |
| S → id = E | S.code = E.code || gen(id.place ‘:=’ E.place) |
| S → if C then S1 | C.true = newlabel;  C.false = S.next;  S1.next = S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code |
| S → if C then S1 else S2 | C.true = newlabel;  C.false = newlabel;  S1.next = S2.next =S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code  || gen(‘goto’ S.next)|| gen(C.false  ‘:’)  || S2.code; |
| S → while C do S1 | S.begin = newlabel;  C.true = newlabel;  C.false = S.next;  S1.next = S.begin;  S.code = gen(S.begin ‘:’) || C.code  || gen(C.true ‘:’) || S1.code  || gen(‘goto’ S.begin); |
| C → E1 > E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘>’ E2.place ‘goto’  C.true) || gen(‘goto’C.false) |
| C → E1 < E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 = E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 >= E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘>=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 <= E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<=’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| C → E1 <> E2 | C.code = E1.code || E2.code ||  gen(‘if’ E1.place ‘<>’ E2.place ‘goto’  C.true) || gen(‘goto’ C.false) |
| E → E1 + T | E.place = newtemp;  E.code = E1.code || T.code||  gen(E.place ‘:=’ E1.place ‘+’ T.place) |
| E → E1 - T | E.place = newtemp;  E.code = E1.code || T.code ||  gen(E.place ‘:=’ E1.place ‘-’ T.place) |
| E → T | E.place = T.place;  E.code = T.code |
| T → F | T.place = F.place;  T.code = F.code |
| T → T1 \* F | T.place = newtemp;  T.code = T1.code || F.code ||  gen(T.place ‘:=’ T1.place ‘\*’ F.place) |
| T → T1 / F | T.place = newtemp;  T.code = T1.code || F.code ||  gen(T.place ‘:=’ T1.place ‘/’ F.place) |
| F → ( E ) | F.place = E.place;  F.code = E.code |
| F → id | F.place = id.name;  F.code = ‘’ |
| F → OCT | F.place = OCT.name;  F.code = ‘’ |
| F → DEC | F.place = DEC.name;  F.code = ‘’ |
| F → HEX | F.place = HEX.name;  F.code = ‘’ |

### 结构体划分

### 关键函数解释

1. int scan(char ss[]);——词法分析程序
   1. 若移进的字符为空白字符，则跳过。
   2. 若移进的字符为数字，则确定进制类型。还会判断是否为小数或负数。
   3. 若移进的字符为字母，那么它可能是一个标识符，或者是一个关键字。对关键字IF、THEN、ELSE、WHILE、DO、BEGIN、END进行比对，如果不是上述关键词之一，那就有可能是标识符，进而查找标识符表确定该标识符是否被定义。如果没有定义，则将其加入标识符表。
   4. 若移进的字符为运算符，则依次对定义的运算符进行比对，并返回相应的token。
2. void findNext(int s);

扫描下一位字符，并覆盖。

1. int judgeReservedWords(char ss[]);

检查是否是保留词：IF THEN ELSE WHILE DO BEGIN END

1. int searchIDN(char ss[]);

在变量表中查找变量。

1. double toDec\_Oct\_Hex(char s[], int n);

将进制转换到十/八/十六进制。

1. void read\_file();

打开输入文件，逐个字符读取文件内容，并在最后加上字符串终止符。

1. char\* gen();

产生新的中间变量t。

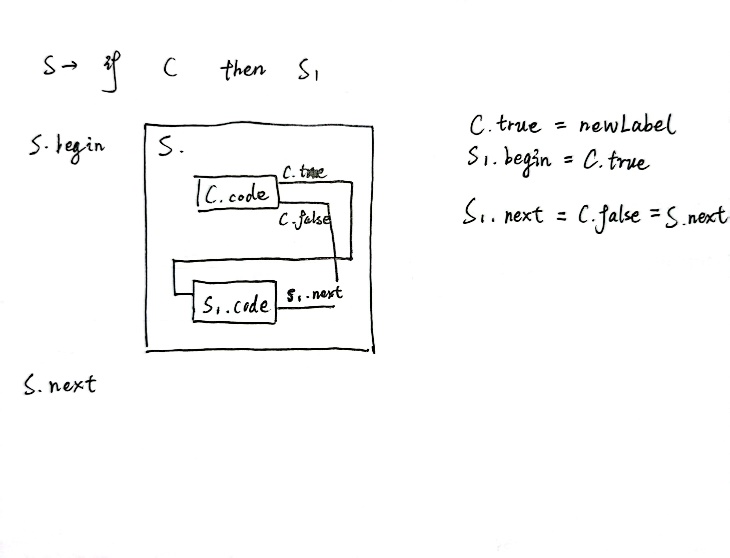
1. int newLabel();

生成新的三地址代码语句标号L。

### 条件控制的布尔表达式翻译思路

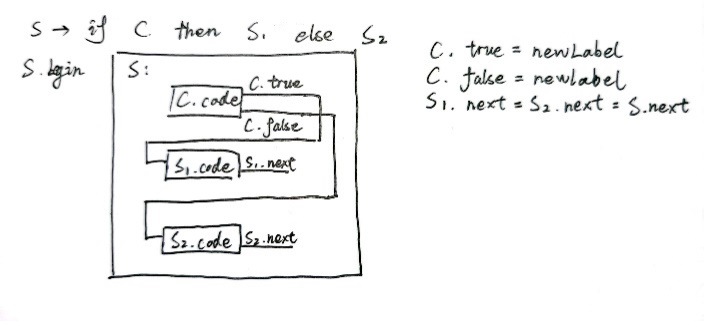
1. **S → if C then S1**

|  |  |
| --- | --- |
| S → if C then S1 | C.true = newlabel;  C.false = S.next;  S1.next = S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code |



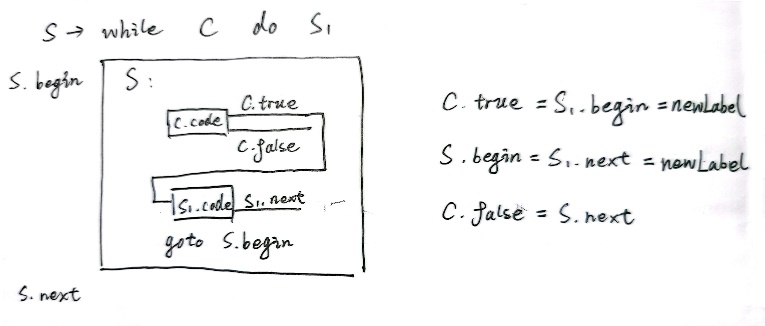
1. **S → if C then S1 else S2**

|  |  |
| --- | --- |
| S → if C then S1 else S2 | C.true = newlabel;  C.false = newlabel;  S1.next = S2.next =S.next;  S.code = C.code || gen(C.true ‘:’) || S1.code || gen(‘goto’ S.next)|| gen(C.false ‘:’) || S2.code; |



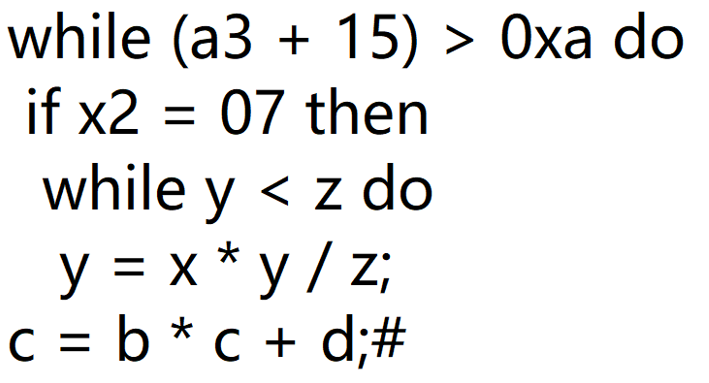
1. **S → while C do S1**

|  |  |
| --- | --- |
| S → while C do S1 | S.begin = newlabel;  C.true = newlabel;  C.false = S.next;  S1.next = S.begin;  S.code = gen(S.begin ‘:’) || C.code || gen(C.true ‘:’) || S1.code || gen(‘goto’ S.begin); |

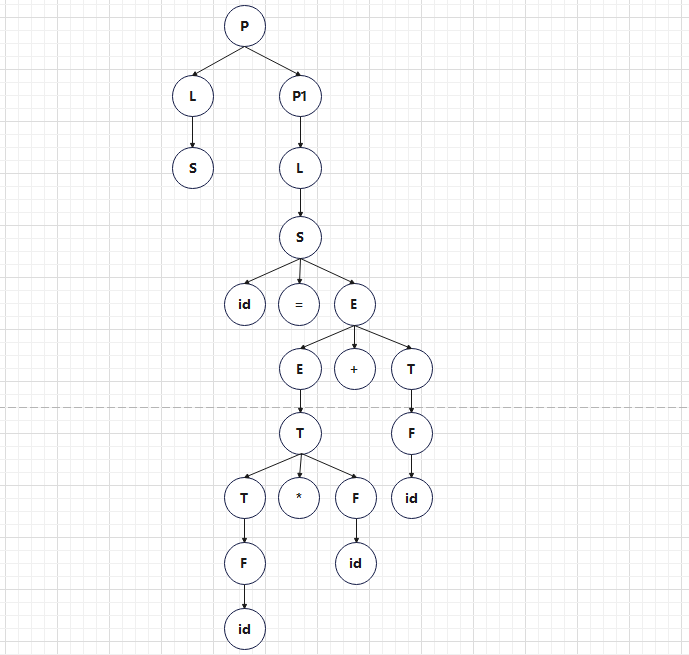
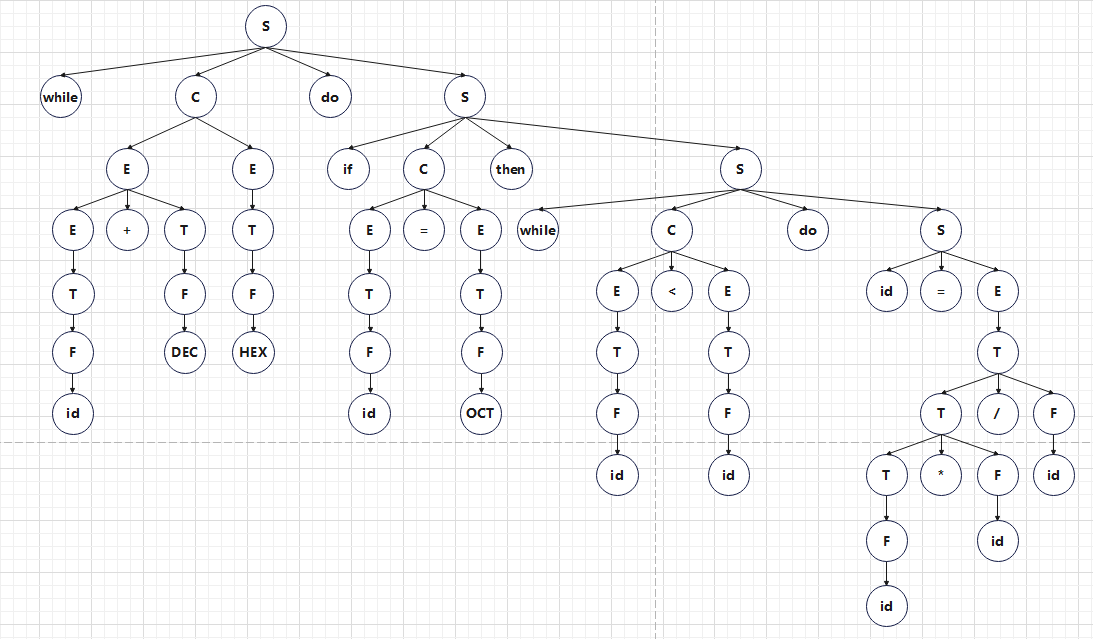


### 测试用例

1. 测试用例



1. 测试用例语法树

1. 测试结果

