# 上 海 大 学

# 2021-2022 春季学期

# 《编译原理 A（08305144）》课程考核报告

学 号：19120158

姓 名：唐新喆

教 师：许东

课程考核评分表

| 序号 | 题号 | 分值 | 成绩 |
|---|---|---|---|
| 1 | 题目 1 | 20 | |
| 2 | 题目 2 | 20 | |
| 3 | 题目 3 | 30 | |
| 4 | 题目 4 | 20 | |
| 5 | 题目 5 | 10 | |
| 考核成绩 | | 100 | |
| 评阅人签名 | | | |

计算机工程与科学学院

2022 年 6 月

## 一、注意事项：

1、课程考核必须由考生独立完成。考核报告提交结束后将进行查重处理，对有抄袭现象的材料，考核成绩作 0 分处理！！！

2、考核报告在 6 月 7 日中午 12：00 之前提交到超星平台上任课老师课程作业的相关目录，逾期没有提交的同学作缺考处理。

3、考核报告用 PDF 文件格式，文件名格式为：学号-姓名.PDF,例如：19120000-张三.PDF。

## 二、考核题目

C 语言的弟弟 C--语言诞生了，它乖巧可爱（功能简单，C 语言的子集），有点小脾气（部分语法表达故意有所不同），有点傻（存在一些缺陷），目前的编译器都嫌弃它，请您帮助它设计一套编译器。

C--语言的文法如下：

<程序>→<函数定义>|<程序><函数定义>

<函数定义>→<类型说明>标识符()<复合语句>

<类型说明>→char|int|float

<复合语句>→{ }|{<语句列表>}|{<声明列表><语句列表>}

<声明列表>→<类型说明><标识符列表>；|<声明列表><类型说明><标识符列表>；

<标识符列表>→标识符|<标识符列表>,标识符

<语句列表>→<语句>；|<语句列表><语句>；

<语句>→<复合语句>|<赋值语句>|<选择语句>|<循环语句>|<输入语句>|<输出语句>|<返回语句>

<赋值语句>→标识符=<表达式>

<表达式>→常量|标识符|标识符()|(<表达式>)|<表达式><算符><表达式>

<算符>→+|-|*|/|>|<|>=|<=|!=|==

<选择语句>→if(<表达式>)<复合语句>|if(<表达式>)<复合语句>else<复合语句>

<循环语句>→while(<表达式>)<复合语句>

<返回语句>→return<表达式>

<输入语句>→cin >>标识符

<输出语句>→cout << <表达式>

# 用< >括起来的是非终结符，未用<>括起来的中英文符号都是终结符

# 标识符:以下划线开头的小写字母和数字串，如_main, _a, _b1,_max2

# 常量:无符号数必须带个后缀，如 123i，12.5f；字符常量，如‘C’；字符串常量，如“Hello C--”

# 除 C--文法明确定义的和以上特别说明之外，按照 C 语言对 C—文法未尽之处进行理解（不能扩充文法），例如：算符优先级和结合律、else 最近匹配等按照 C 语言文法规定理解。

## 题目 1：程序设计　（20 分）

请用 C—语言编写程序，特别**注意严格按照 C—文法定义书写**。

要求：输入 100 个正整数，求最大值和平均值，并输出。

## 题目 2：词法分析　（20 分）

1）根据 C—语言的文法，总结 C—语言的词汇表（保留字、运算符、界符）。（5 分）

2）设计标识符、常量两类词汇的正规式，并构造有穷自动机，要求写出过程。（15 分）

## 题目 3：语法分析　（30 分）

构造 C—语言“赋值语句”的 SLR(1) 分析表（如果有冲突，请按照 C—语义修改文法，保证修改前后文法等价，并写出修改后的文法），写出详细的构造过程。

## 题目 4：中间代码生成（20 分）

构造 C—语言的翻译模式，以生成中间代码（三地址码）

## 题目 5：课程总结（10 分）

不少于 200 字的课程总结，包括课程体会，以及对课程教学的建议。

〈题目部分结束，总计 5 道题目〉

# 三、《编译原理（08305013）》课程考核报告

**注意：**

1）题目论述要详细，但不要冗长，以下报告篇幅可以自行增加；

2）需要画图的部分可以用画图工具，也可以使用纸张手绘拍照插入进来；

3）报告正文使用宋体、五号字、1.25 倍行距排版。

## 题目 1：

代码以图片的方式给出，目的是保证格式与高亮。

注：作为测试，代码的算法思路在 C 语言中可以得到正确的运行结果。

```
int _main()
{
    int _x, _max, _sum, _avg, _count;
    cin >> _x;
    _max = _x;
    _sum = _x;
    _count = 1i;
    while(_count < 100i)
    {
        cin >> _x;
        _count = _count + 1i;
        _sum = _sum + _x;
        if(_x > _max)
        {
            _max = _x;
        }
    }
    _avg = _sum / 100i;
    cout << _max;
    cout << _avg;
    return 0i;
}
```

## 题目 2：

（1）词汇表

保留字：char、int、float、if、else、while、return、cin、cout

运算符：+、-、*、/、>、<、>=、<=、!=、==、=

界符：(、 )、 {、 }、 ,、 ;

（2）正规式和自动机

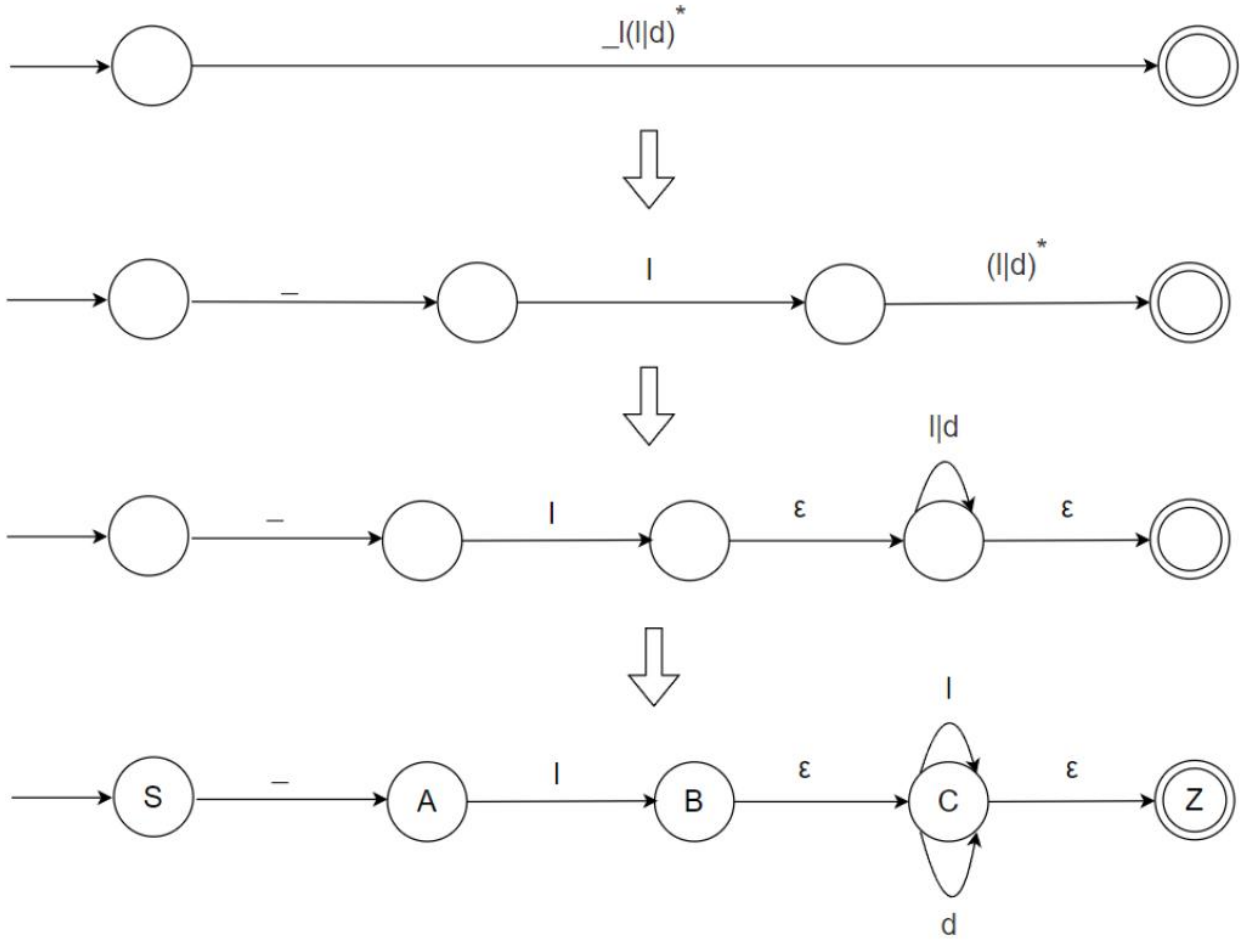
标识符的正规式：_l(l|d)*

令 Σ = {l, d, _}，其中 l 为 a~z 中的小写字母，d 为 0~9 中的数字，则标识符可以用Σ上的正规式 _l(l|d)* 表示。

注：word 文档中说明标识符为以下划线开头的小写字母和数字串，如_main, _a, _b1,_max2。这里没有出现下划线_后为数字的案例，因此，我按照 C 语言的标识符命名规范，将标识符视作必须以下划线_和一个小写字母开头，而例如_1 则不视为标识符。

构造有穷自动机

1、由正规式构造出 NFA



2、根据 NFA 图得到状态转换表（子集构造法）

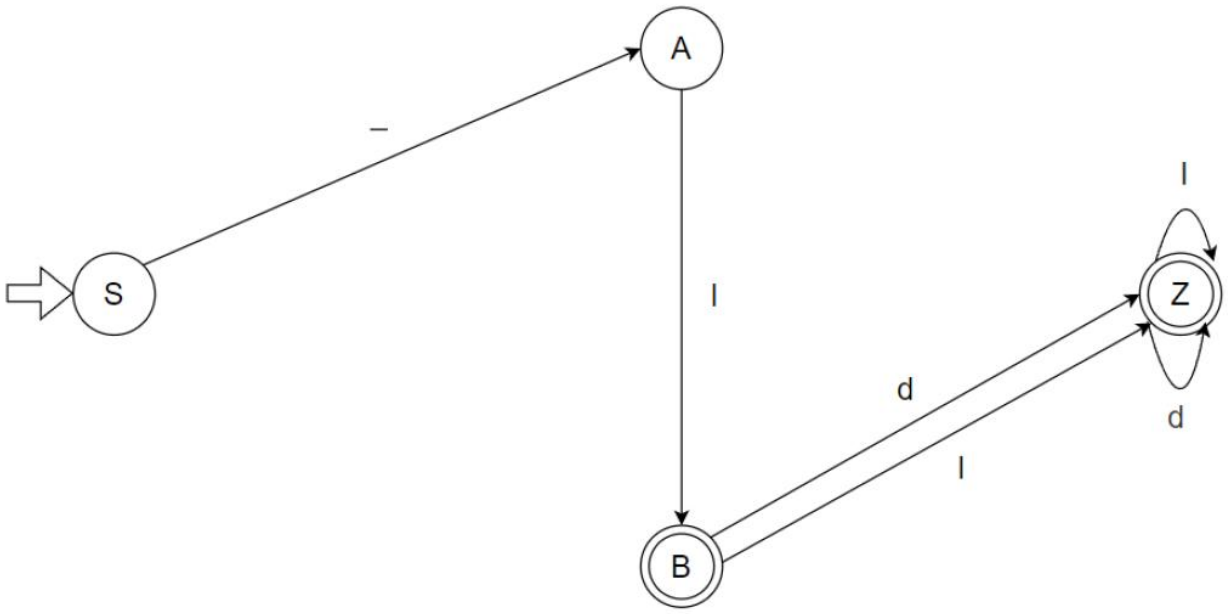
| | _ | l | d |
|---|---|---|---|
| {S} | {A} | ∅ | ∅ |
| {A} | ∅ | {B,C,Z} | ∅ |
| {B,C,Z} | ∅ | {C,Z} | {C,Z} |
| {C,Z} | ∅ | {C,Z} | {C,Z} |

3、根据状态转换表得到 DFA 图

对不同的状态集进行编号，得到：

| | _ | l | d |
|---|---|---|---|
| S {S} | {A} A | ∅ | ∅ |
| A {A} | ∅ | {B,C,Z} B | ∅ |
| B {B,C,Z} | ∅ | {C,Z} Z | {C,Z} Z |
| Z {C,Z} | ∅ | {C,Z} Z | {C,Z} Z |

初步绘制 DFA 图



4、DFA 图的最小化

π0 = （{S, A}, {B, Z}）

S 只能代入_，A 只能代入 l，因此把 A 提取出来。

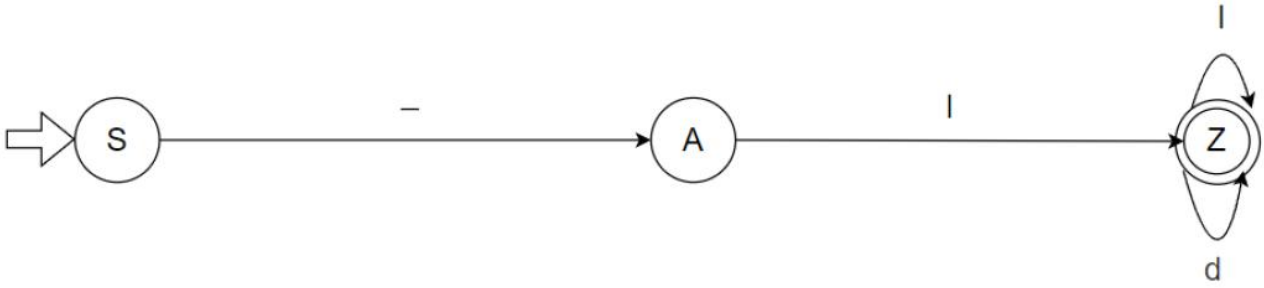
π1 = （{S}, {A}, {B, Z}）

{B, Z}l = {Z} ⊆ π1

{B, Z}d = {Z} ⊆ π1

所以得到了最小化的集合 π1，且 B 和 Z 是等价的状态，可以从两者中任意去掉一个。

最后得到最小化 DFA：




常量的正规式：(dd*i) | (dd*(.dd*| ε)f) | ( ‘(l|L|d|s)’ ) | (“(l|L|d|s)*”)

令字母表Σ = {l, L, d, s, i, f, -}，

辅助字母表Σ’ = { ε， ‘, ’, “, ”, ., *, (, ), |,}，

其中 l 为 a~z 中的小写字母，L 为 A~Z 中的大写字母，d 为 0~9 中的数字（这里不考虑 e）， s 为单个的符号。

我将常量分为 4 种，分别是无符号整数（以 i 结尾），无符号浮点数（以 f 结尾），字符常量（’’之间必须有一个字符）和字符串常量，分别表示为：

注：因为文法中的类型说明里只有 int，没有 unsigned int，且从其他老师处得知只需考虑无符号数，因此这里不考虑有符号数。如果考虑有符号数，其表示为-dd*(.dd*| ε)

无符号整数：dd*i

无符号浮点数：dd*(.dd*| ε)f

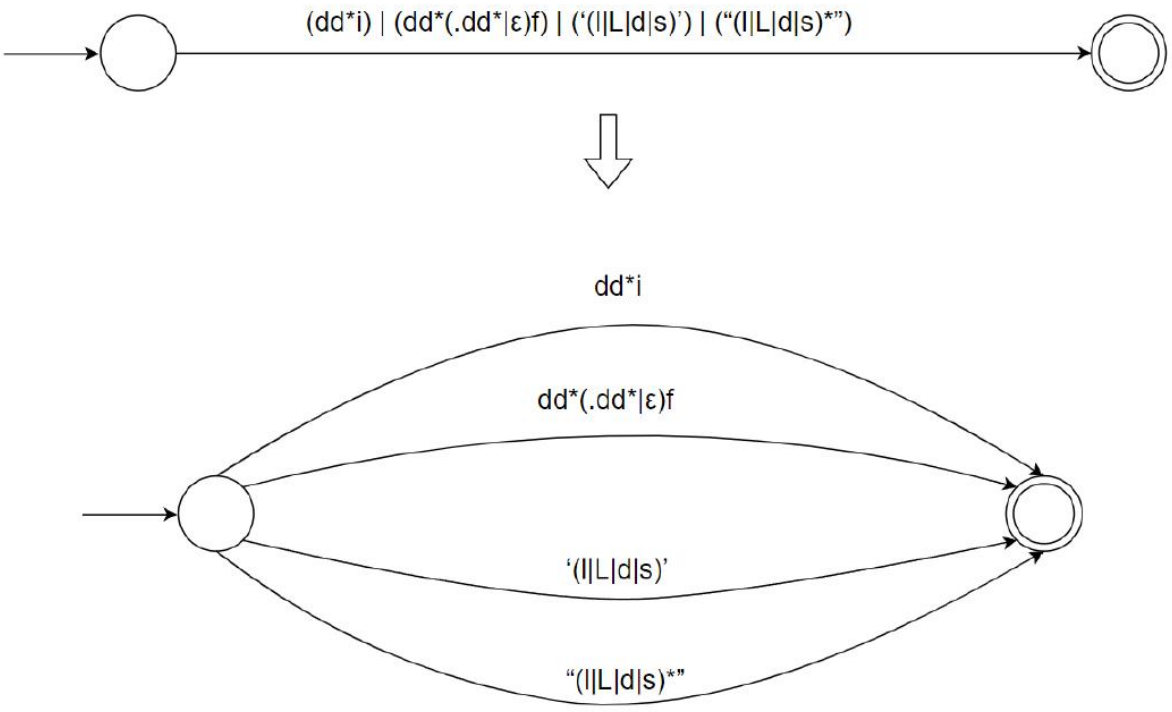
字符常量： ‘(l|L|d|s)’

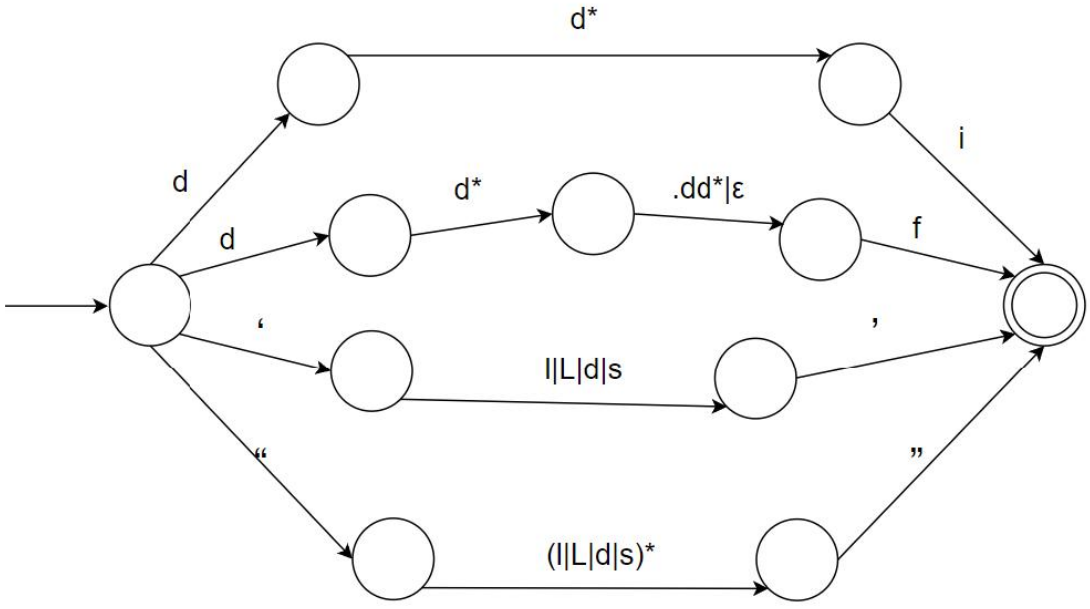
字符串常量：“(l|L|d|s)*”

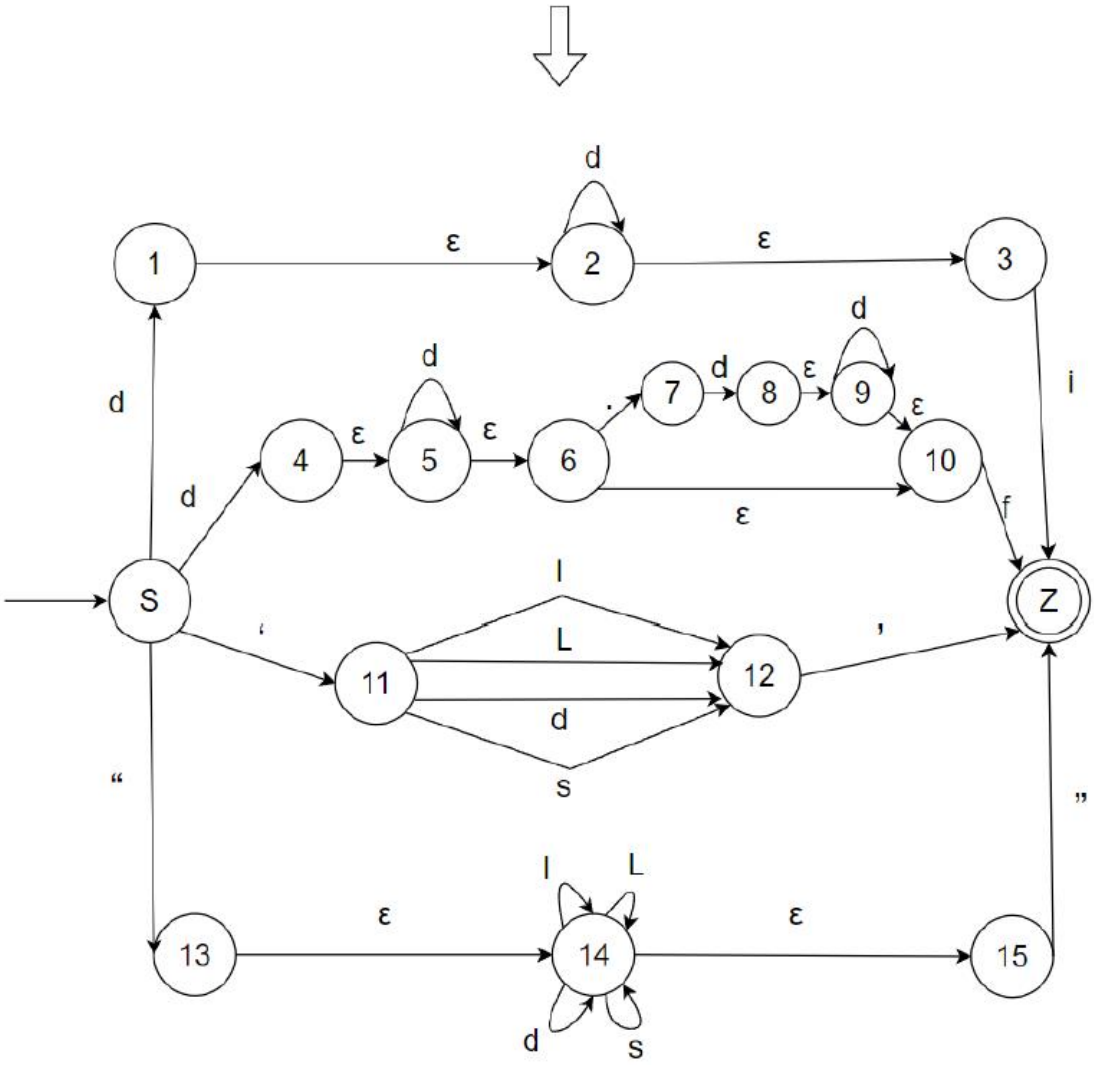
则常量可以用Σ上的正规式 (dd*i) | (dd*(.dd*| ε)f) | ( ‘(l|L|d|s)’ ) | (“(l|L|d|s)*”) 表示。

构造有穷自动机

1、由正规式构造出 NFA


(dd*i) | (dd*(.dd*|ε)f) | (‘(l|L|d|s)’) | (“(l|L|d|s)*”)
dd*i
dd*(.dd*|ε)f
‘(l|L|d|s)’
“(l|L|d|s)*”


d*
i
d
d*
.dd*|ε
f
d
‘
’
l|L|d|s
“
”
(l|L|d|s)*




2、根据 NFA 图得到状态转换表（子集构造法）

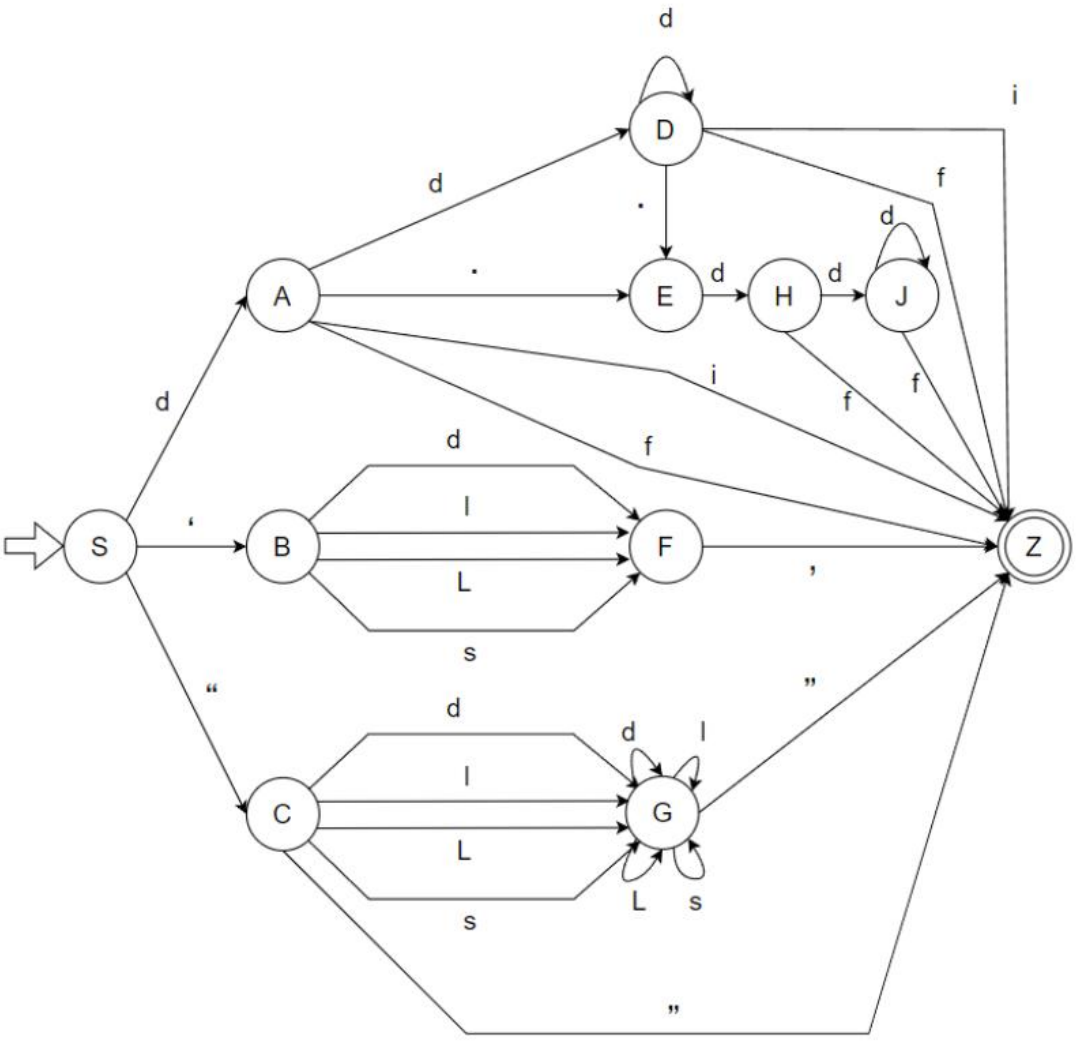
| | d | i | . | f | ‘(左) | l | L | s | ‘(右) | “(左) | ”(右) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| {S} | {1,2,3,4,5,6,10} | ∅ | ∅ | ∅ | {11} | ∅ | ∅ | ∅ | ∅ | {13,14,15} | ∅ |
| {1,2,3,4,5,6,10} | {2,3,5,6,10} | {Z} | {7} | {Z} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| {11} | {12} | ∅ | ∅ | ∅ | ∅ | {12} | {12} | {12} | ∅ | ∅ | ∅ |
| {13,14,15} | {14,15} | ∅ | ∅ | ∅ | ∅ | {14,15} | {14,15} | {14,15} | ∅ | ∅ | {Z} |
| {2,3,5,6,10} | {2,3,5,6,10} | {Z} | {7} | {Z} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| {Z} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| {7} | {8,9,10} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| {12} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | {Z} | ∅ | ∅ |
| {14,15} | {14,15} | ∅ | ∅ | ∅ | ∅ | {14,15} | {14,15} | {14,15} | ∅ | ∅ | {Z} |
| {8,9,10} | {9,10} | ∅ | ∅ | {Z} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| {9,10} | {9,10} | ∅ | ∅ | {Z} | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |

3、根据状态转换表得到 DFA 图

对不同的状态集进行编号，得到：

| | d | i | . | f | '(左) | l | L | s | '(右) | "(左) | "(右) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S {S} | {1,2,3,4,5,6,10}A | ø | ø | ø | {11}B | ø | ø | ø | ø | {13,14,15}C | ø |
| A {1,2,3,4,5,6,10} | {2,3,5,6,10}D | {Z}Z | {7}E | {Z}Z | ø | ø | ø | ø | ø | ø | ø |
| B {11} | {12}F | ø | ø | ø | ø | {12}F | {12}F | {12}F | ø | ø | ø |
| C {13,14,15} | {14,15}G | ø | ø | ø | ø | {14,15}G | {14,15}G | {14,15}G | ø | ø | {Z}Z |
| D {2,3,5,6,10} | {2,3,5,6,10}D | {Z}Z | {7}E | {Z}Z | ø | ø | ø | ø | ø | ø | ø |
| Z {Z} | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø |
| E {7} | {8,9,10}H | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø |
| F {12} | ø | ø | ø | ø | ø | ø | ø | ø | {Z}Z | ø | ø |
| G {14,15} | {14,15}G | ø | ø | ø | ø | {14,15}G | {14,15}G | {14,15}G | ø | ø | {Z}Z |
| H {8,9,10} | {9,10} J | ø | ø | {Z}Z | ø | ø | ø | ø | ø | ø | ø |
| J {9,10} | {9,10} J | ø | ø | {Z}Z | ø | ø | ø | ø | ø | ø | ø |

初步绘制 DFA 图



4、DFA 图的最小化

π 0 = （{S, A,B,C,D,E,F,G,H,J}, {Z}）

根据各个集合里各个状态能代入的符号，将集合重新划分为：

π 1 = （{S}, {A, D}, {B}, {C, G}, {E}, {F}, {H, J}, {Z}）

{A, D}d = {D} ⊆ π 1

{A, D}i = {Z} ⊆ π 1

{A, D}. = {E} ⊆ π 1

{A, D}f = {Z} ⊆ π 1

所以 A 和 D 是等价的状态，可以从两者中任意去掉一个。

{C, G}d = {G} ⊆ π 1

{C, G}l = {G} ⊆ π 1

{C, G}L = {G} ⊆ π 1

{C, G}s = {G} ⊆ π 1

{C, G}” = {Z} ⊆ π 1

所以 C 和 G 是等价的状态，可以从两者中任意去掉一个。

{H, J}d = {J} ⊆ π 1

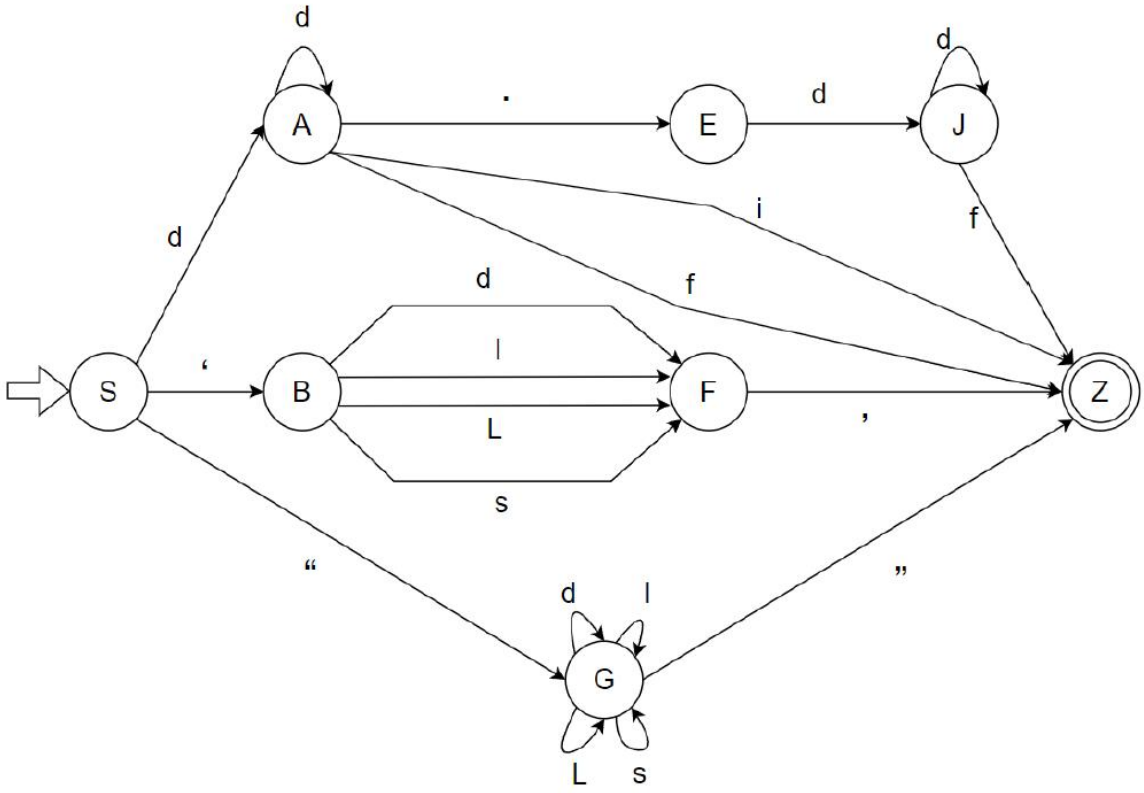
{H, J}f = {Z} ⊆ π 1

所以 H 和 J 是等价的状态，可以从两者中任意去掉一个。

此时 π = （{S}, {A}, {B}, {G}, {E}, {F}, {J}, {Z}）

| | d | i | . | f | ‘(左) | l | L | s | ‘(右) | “(左) | ” (右) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S {S} | A | ø | ø | ø | B | ø | ø | ø | ø | G | ø |
| A {1,2,3,4,5,6,10} | A | Z | E | Z | ø | ø | ø | ø | ø | ø | ø |
| B {11} | F | ø | ø | ø | ø | F | F | F | ø | ø | ø |
| Z {Z} | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø |
| E {7} | J | ø | ø | ø | ø | ø | ø | ø | ø | ø | ø |
| F {12} | ø | ø | ø | ø | ø | ø | ø | ø | Z | ø | ø |
| G {14,15} | G | ø | ø | ø | ø | G | G | G | ø | ø | Z |
| J {9,10} | J | ø | ø | Z | ø | ø | ø | ø | ø | ø | ø |

最后得到最小化 DFA：



## 题目 3：

赋值语句相关文法：

<语句列表>→<语句>；|<语句列表><语句>；

<语句>→<赋值语句>

<赋值语句>→标识符=<表达式>

S　　a　　A

<表达式>→常量|标识符|标识符()|(<表达式>)|<表达式><算符><表达式>

A　　b　　c　　B

<算符>→+|-|*|/|>|<|>=|<=|!=|==

B　　d

给定文法 G[S]：

S→a=A

A→b|c|c()|(A)|ABA

B→d

S 表示赋值语句，a 表示标识符，A 表示表达式，b 是常量，c 是标识符，B 是算符，d 表示具体的算符。

文法的扩展并编号：

（1）S’ → S

（2）S → a = A

（3）A → b

（4）A → c

（5）A → c ( )

（6）A → ( A )

（7）A → A B A

（8）B → d

写出扩展文法 G’的所有项目：

S’ → .S

S’ → S.

S → .a = A

S → a. = A

S → a =. A

S → a = A.

A → .b

A → b.

A → .c

A → c.

A → .c ( )

A → c. ( )

A → c (. )

A → c ( ).

A → .( A )

A → (. A )

A → ( A. )

A → ( A ).

A →. A B A

A → A. B A

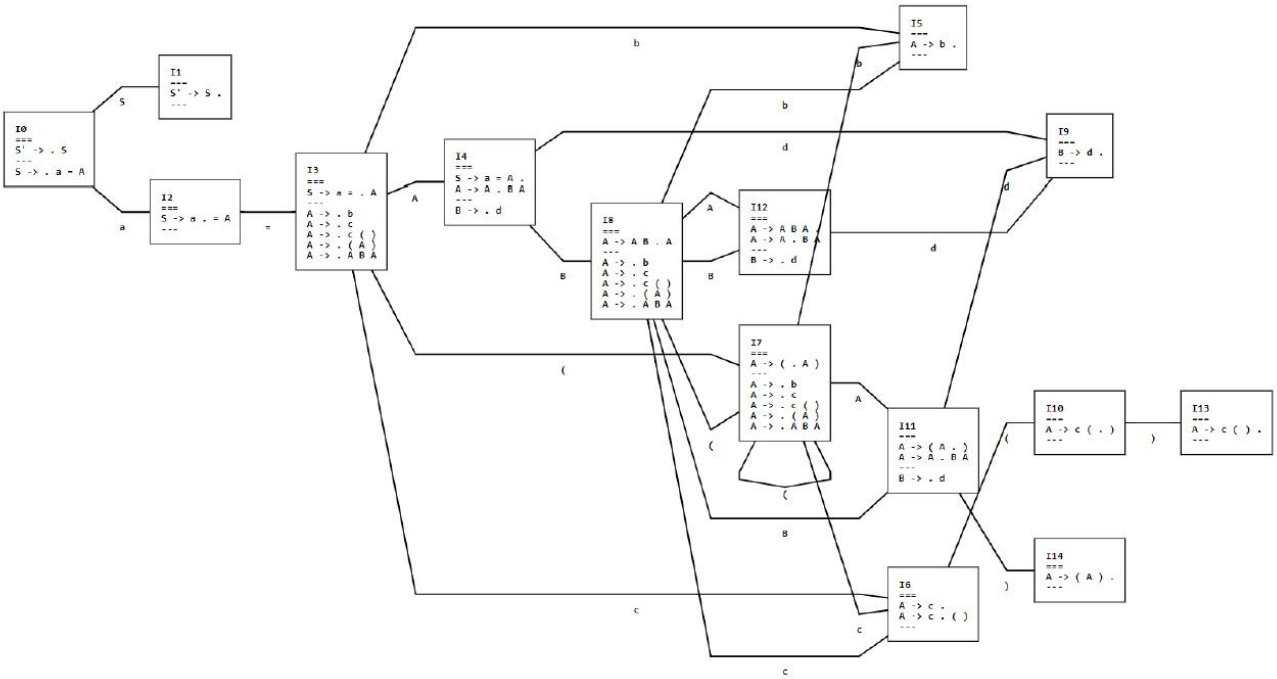
A → A B. A

A → A B A.

B → .d

B → d.

构建识别 G’活前缀的 DFA




求出 FIRST 集和 FOLLOW 集

| | FIRST | FOLLOW |
|---|---|---|
| S' | {a} | {#} |
| S | {a} | {#} |
| A | {b,c,(} | {#,),d} |
| B | {d} | {b,c,(} |

构建 SLR(1)分析表

| [状态 STATE] | [ | | | | 动作 ACTION | | | ] | [ | 转换 GOTO | ] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | = | b | c | ( | ) | d | # | S | A | B |
| 0 | s2 | | | | | | | | 1 | | |
| 1 | | | | | | | | acc | | | |
| 2 | | s3 | | | | | | | | | |
| 3 | | | s5 | s6 | s7 | | | | | 4 | |
| 4 | | | | | | | s9 | r1 | | | 8 |
| 5 | | | | | | r2 | r2 | r2 | | | |
| 6 | | | | | s10 | r3 | r3 | r3 | | | |
| 7 | | | s5 | s6 | s7 | | | | | 11 | |
| 8 | | | s5 | s6 | s7 | | | | | 12 | |
| 9 | | | r7 | r7 | r7 | | | | | | |
| 10 | | | | | | s13 | | | | | |
| 11 | | | | | | s14 | s9 | | | | 8 |
| 12 | | | | | | r6 | s9 / r6 | r6 | | | 8 |
| 13 | | | | | | r4 | r4 | r4 | | | |
| 14 | | | | | | r5 | r5 | r5 | | | |

由表可知出现了冲突，要消除左递归，需要修改赋值语句的文法：

赋值语句

<赋值语句>→标识符=<表达式 A>

S　　a　=　A

<表达式 A>→常量<表达式 B>|标识符<表达式 B>|标识符()<表达式 B>|(<表达式 A>)<表达式 B>

A　　b　B　　c　B　　c()　B　　(A)　B

<表达式 B>→<算符><表达式 A><表达式 B>| ε

B　　D　A　　B

<算符>→+|-|*|/|>|<|>=|<=|!=|==

D　　d

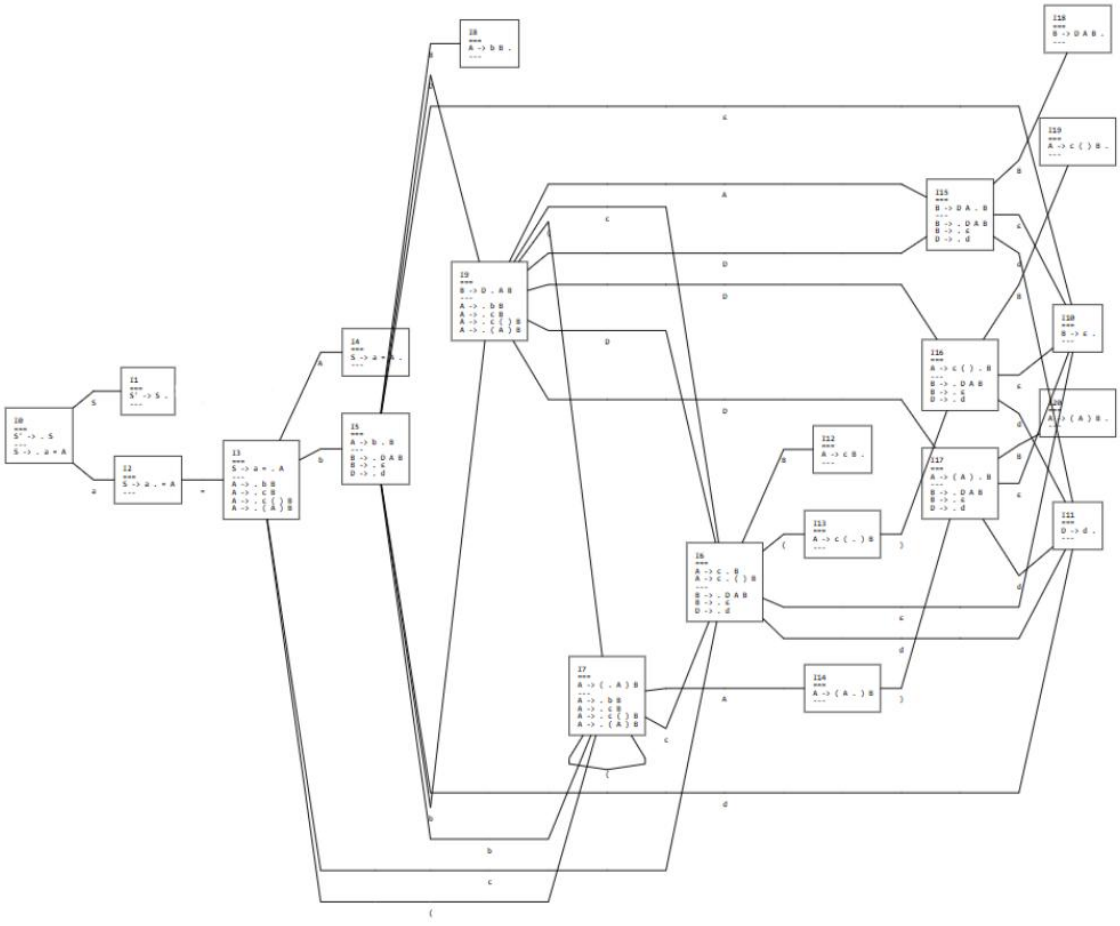
S→a=A

A→bB|cB|c()B|(A)B

B→DB| ε

D→d

继续之前的流程，构建识别 G'活前缀的 DFA：



求出 FIRST 集和 FOLLOW 集

| | FIRST | FOLLOW |
|---|---|---|
| S' | {a} | {#} |
| S | {a} | {#} |
| A | {b,c,(} | {#,),d} |
| B | {d} | {#,),\|,d} |
| D | {d} | {b,c,(} |

构建 SLR(1)分析表

| [状态 STATE] | [ | | | | 动作 | ACTION | | | | ] | [ | 转换 | GOTO | ] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | = | b | c | ( | ) | \| | ε | d | # | S | A | B | D |
| 0 | s2 | | | | | | | | | | 1 | | | |
| 1 | | | | | | | | | | acc | | | | |
| 2 | | s3 | | | | | | | | | | | | |
| 3 | | | s5 | s6 | s7 | | | | | | | 4 | | |
| 4 | | | | | | | | | | r1 | | | | |
| 5 | | | | | | | | | s10 | | | | 8 | 8 |
| 6 | | | | | s12 | | | | s10 | | | | 11 | 9 |
| 7 | | | s5 | s6 | s7 | | | | | | | 13 | | |
| 8 | | | | | | r2 | | | r2 | r2 | | | | |
| 9 | | | s5 | s6 | s7 | | | | | | | 14 | | |
| 10 | | | r7 | r7 | r7 | | | | | | | | | |
| 11 | | | | | | r3 | | | r3 | r3 | | | | |
| 12 | | | | | | s15 | | | | | | | | |
| 13 | | | | | | s16 | | | | | | | | |
| 14 | | | | | | | | | s10 | | | | 17 | 9 |
| 15 | | | | | | | | | s10 | | | | 18 | 9 |
| 16 | | | | | | | | | s10 | | | | 19 | 9 |
| 17 | | | | | | | s20 | | | | | | | |
| 18 | | | | | | r4 | | | r4 | r4 | | | | |
| 19 | | | | | | r5 | | | r5 | r5 | | | | |
| 20 | | | | | | | | s21 | | | | | | |
| 21 | | | | | | r6 | r6 | | r6 | r6 | | | | |

由表可知没有出现冲突，C—语言“赋值语句”的 SLR(1) 分析表构造完成。

## 题目 4：

注：本题将 C--语言主要分为声明语句、赋值语句、控制语句。本题主要参照课本 8.3.3 生成三地址码 / 211 编写，在查重上可能存在相似的部分。

声明语句

S → T id; S | ε

T → char | int | float

符号 S 表示声明序列；

符号 id 表示变量名；

符号 T 表示基本类型。

在声明语句中，需要处理变量的类型并且分配相对地址，一个变量的相对地址由其起始地址和类型宽度共同决定，类型宽度受到变量类型的影响而不同。C--语言的类型只有 3 个，即 char、int 和 float，这 3 个基本类型的宽度应该是语言本身定义好的，按照 C 语言的定义，我将 C--语言的变量类型宽度设定为：char 的类型宽度为 1，int 的类型宽度为 4，float 的类型宽度为 4。

增加一条产生式 P->S {Stack.push(top); Stack.push(move); top=new env(); move=0}；即在第一个声明之前,对当前的上下文环境进行保存,该动作把指向当前符号表的指针 top 和记录字段偏移量的变量 move 保存到 Stack 中，并把 top 指向一个新符号表，最后 move 被设置为 0。在处理产生式 S → T id; S | ε 时，每创建一个变量 id 时就将其加入符号表，并把它的相对地址设置为当前 move 的值，move 再添加 id 对应的的类型宽度的值。

P → S　　{Stack.push(top); Stack.push(move); top=new env(); move=0}

S → T id;　{top.push(id.val, T.type, move); move = move + T.width;} S

S → ε　　{T.type = record(top); T.width = move; move = Stack.pop(); top = Stack.pop()}

T → char　{T.type = char; T.width = 1}

T → int　　{T.type = int; T.width = 4}

T → float　{T.type = float; T.width = 4}

产生式 S → T id; S 先通过 top.push(id.val, T.type, move) 为 id.val 创建一个符号表条目。top 指向当前符号表，条目的数据区存放了类型 T.type 以及相对地址。

最后，在产生式 S → ε 中，对之前的上下文环境进行还原，把记录的类型设为 record(top)，把记录的宽度设为 move，并把 top 指向之前保存在 Stack 中的符号表，把 move 设为之前保存在 Stack 中的值。

赋值语句

对表达式 E 求值并储存在变量 id 中

S→id:=E　{ S.code := E.code || generate( id.place ':=' A.place)}

E→id　{ E. place := id.place; E.code := "" }

E→E1+E2　{ E.place := newname;　generate(E.place　':='　E1.place　'+'　E2.place)}

```
E→E1-E2  { E.place := newname;  generate(E.place  ':='  E1.place  '-'  E2.place)}
E→E1*E2  { E.place := newname;  generate(E.place  ':='  E1.place  '*'  E2.place)}
E→E1/E2  { E.place := newname;  generate(E.place  ':='  E1.place  '/'  E2.place)}
E→(E1)   { E.place := E1.place; E.code := E1.code }
```

表达式

```
E→(E1)      { E.place:=E1.place }
E→id        { E.place:=id.place }
```

控制语句

```
P→S;        {S.next := newlabel} S {generate(S. next ':') }
S→if        {E.true := newlabel; E.false := S.next} E
            {S1.next := S.next} S1 { S.code := E.code || generate(E.true':') || S1. code}
S→if        {E.true := newlabel; E.false := newlabel} E
            {S1.next := S.next} S1 else {S2.next := S.next} S2
            {S.code := E.code || generate(E.true':') || S1.code ||
                    generate('goto' S.next) || generate(E.false':') || S2. code}
S→while     {E.true := newlabel; E.false := S.next} E
            {S1.next := newlabel} S1
            {S.code := generate(S1.next ':') || E.code || generate(E.true ':') ||
                    S1.code || generate('goto'S1.next) }
S→          {S1.next := newlabel} S1; {S2.next := S.next} S2
            {S.code := S1.code || generate(S1.next ':') || S2. code}
```

## 题目 5：

从刚开始编写和运行 C 语言时就会用到编译器，但那时对编译器的了解，只是将其作为底层的一种工具。不论是在命令行中手动编译代码再运行，还是直接在 IDE 中一键编译并运行代码，都只是把编译当做是一个必备步骤而已。但是，在学习了编译原理课程之后，才明白编译器在编程中的重要作用，虽然大多数人习惯于只写源代码，剩下的交给编译器，但是作为计算机专业的学生，必须要对编译器做了什么有所了解。

编译原理这门课程，主要就是在介绍将高级语言翻译成机器代码的过程中用到的各种原理和算法。我的学习感受是要学的内容很多，而且比较难，因为编译原理偏向底层，而且实验课程中要设计到非常多的数据结构、算法，是对过往所学的一种融会贯通。编译原理涉及的知识点会伴随许多图表，比较难理解和记忆，想真正的掌握需要下一番功夫。编译原理不简单，但有趣且有用，它让我能够了解语言设计者的设计思想，例如理解了语言中关键字存在的原因和作用，甚至自己也可以成为一个设计者。

对于课程教学的建议，我认为老师可以对实验内容做出更多的指导，因为理论课中的知识含量已经很大了，实验的要求也不算很低。但是这也可能是我处于线上教学的原因，不能切身在实验室中请教老师，得到老师的指点和教导，因此这仅仅是我个人的一点点小建议。

〈报告结束〉