第二次作业

练习2.1.1

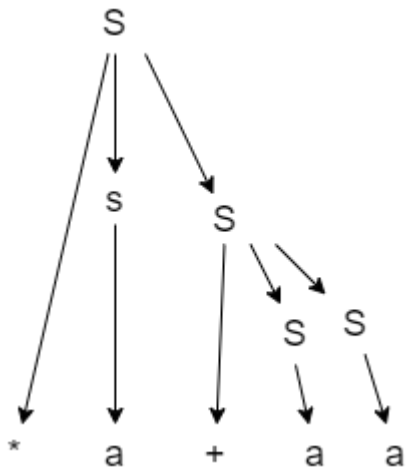
考虑下面的上下文无关文法: $S->+SS|*SS|a$

(1) **试说明如何使用文法生成串**$*a+aa$

```
S => *SS => *S+SS => *a+SS => *a+aS => *a+aa
```

2）**试为这个串构造一棵语法分析树**

如图所示：



3）**该文法生成的语言是什么？为什么?**

该文法生成的语言是所有以{a,*,+,}的前缀表达式，且a为奇数。 将其转化为一般形式：

```
S -> +aS | +Sa | *aS | *Sa |a
```

每一个生成式都是一个前缀表达式，生成的句子也是一个前缀表达式。

练习2.1.2

考虑文法：

```
num -> 101 | 1111 |num0 | num num
```

1. **证明：用该文法生成的所有二进制串的值都能被5整除（提示：对语法分析树的结点数目，即推导步数，使用数学归法）**

基础步骤

首先对于产生式num → 101，其十进制值为5，产生式 num → 1111，其十进制值为15，两个数都可以被5整除。

归纳步骤

假设所有语法分析树高为k的二进制串的值（记为num_{k}）都能被5整除，观察高为k+1的二进制串：

对于产生式

```
    num_k

   /      \

num_k+1   0
```

由于原始的num_k可以被5整除，所以在其后面添加一个0（即乘以2）后得到的新的二进制串的值也可以被5整除。 对于产生式

```
    num_k

   /      \

num_k+1   num_k+1
```

由于两个num_{k+1}都可以被5整除，两个串的组合也能被5整除，（相当于前者乘以2的幂方再加上一个5的倍数）。因此用该文法生成的所有二进制串的值都能被5整除。

2. **上面的文法是否能够生成所有能被5整除的二进制串？**

不能，一个反例式该文法不能生成0，而0可以被5整除。

练习2.1.3

构建一个语法制导翻译方案，该方案把算术表达式从中缀表示方式翻译为运算符在运算分量之后的后缀表示方式。例如，xy-是表达式x-y的后缀表示。给出输入9-5+2和9-5*2的注释分析树

让$expr$表示表达式，$term$表示项，$factor$表示因子，得到的文法如下：

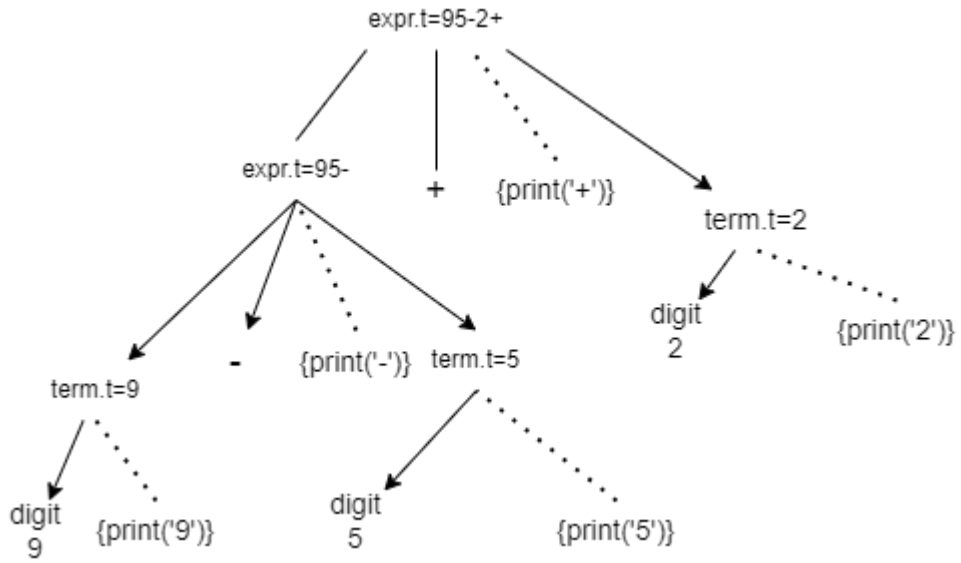
```
expr   → expr + term  | expr - term  | term
term   → term * factor  | term / factor | factor
factor → ( expr ) | digit
```

对应的翻译方案为

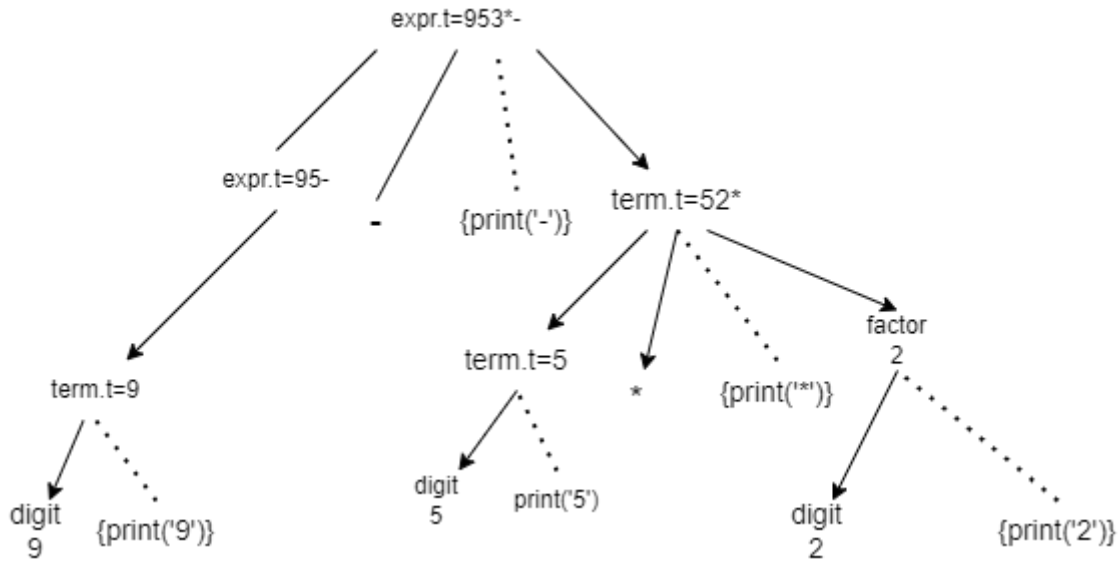
```
(1) expr → expr + term { print('+') }
(2) expr → expr - term { print('-') }
(3) expr → term
(4) term → term * factor { print('*') }
(5) term → term / factor { print('/') }
(6) term → factor
(7) factor → (expr)
(8) factor → digit { print('digit') }
```

注释分析树：

9-5+2



9−5*2



练习2.1.4

从 C99 标准开始，C 语言的 for 语句中初始化子句可以是声明，例如：

```
for ( int i = 0; i < n; i++ )
```

第一在 C++ 中允许类似的语法，但区别是：C++ 中 **初始化语句** 的作用域与**循环语句** 的作用域一致，而在 C 中 **循环语句** 的作用域嵌套于**初始化语句**的作用域中。例如

```
for (int i = 0; ; )
    {
        long i = 1; //在 C 中合法，在 C++ 中非法
    }
```

参考龙书图2-37中的实现，试讨论在两种 for 语句实现中分别应该如何实现符号表来正确区分变量和避免重复声明。

若利用树形结构的链接符号表来实现嵌套，那么为了正确区分变量且避免重复声明，在向符号表添加表项时需要考虑嵌套符号表中的表项。

在C中，for语句的初始化子句和循环语句的作用域需要嵌套，两个符号表是父节点和子节点的关系，故可以直接在对应符号表中添加表项。

在C++中，初始化子句和循环子句的作用域是同一个，那么在符号表中添加表项时需要考虑该符号表中是否已经进行过声明，如果是就不添加表项，否则就添加表项。

```
#include <vector>
public class Env
{
    private Hashtable table;
    protected Env* parent;
    std::vector<Env*> children;

    public Env(Env* p){
        table =new Hashtable; parent = p ; p.children.push_back(this);
    }

    public void put(String s,symbol sym){
         /* 在C++中的操作
        if(get(s))
            {do nothing}
        */
        table.put(s,sym)


    }

    public Symbol get(String s){
            for(Env e =this;e!=null;e=e.parent)
            {
                bool found =(bool)(e.table.get(s));
                if(found) return found;
```

```
                }
                return null;
            }
        }
```

2.1.5

给出一个文法，生成所有被3整除的非负整数的二进制串并给出证明。

文法如下：

S -> DD | ε | D

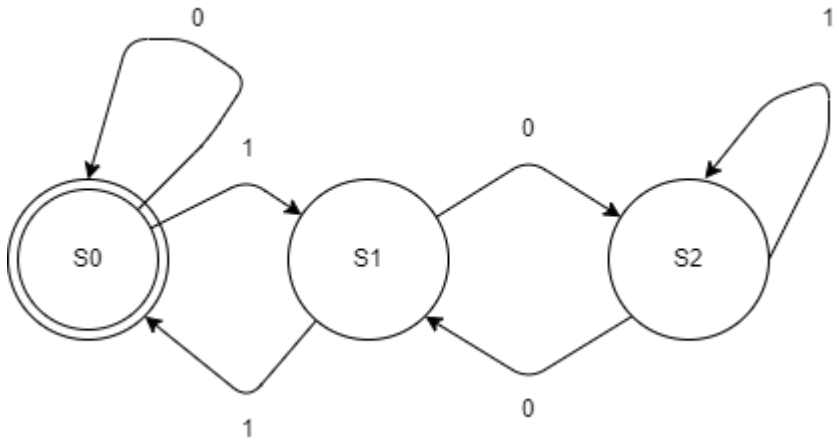
A -> AA | 1 | ε

B -> 0A0

C -> 1B1

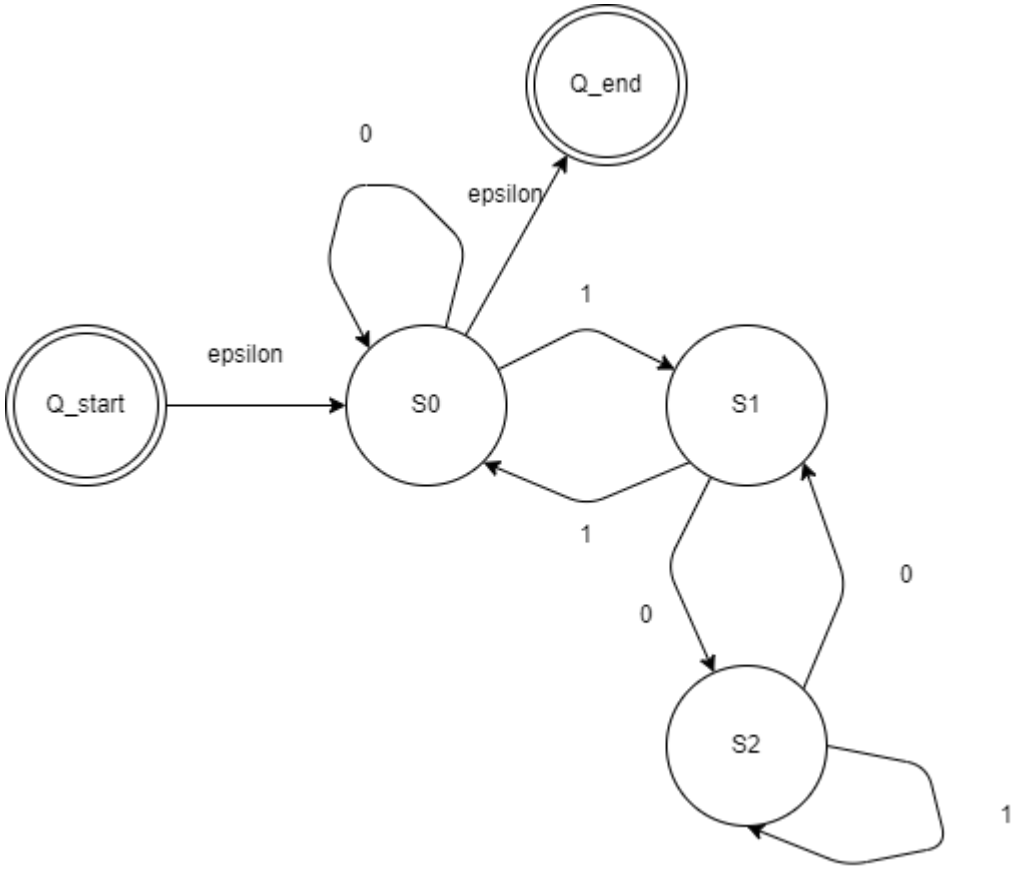
D -> C|0

证明： 给出符合题设的DFA，S0为初始状态，S1为模3余1的状态，S2为模3余2的状态，则可以画得DFA如下图



所示：

将DFA转换为GNFA可以得到



将该NFA转换为RE表达式即为：

```
[(1(01*0)1)|0]*
```

该式转换为文法即可。