Simple recursive descent parser for

S $\rightarrow$ aSa | bSb | c

Just recursively check which rule alternative to apply. It's easy because you can always tell by the symbol you're looking at:
'a': S $\rightarrow$ aSa
'b': S $\rightarrow$ bSb
'c': S $\rightarrow$ c

The bad news is that that doesn't work of the last rule is replaced by S $\rightarrow \varepsilon$, because you see an 'a' or a 'b' this case and you don't know how to continue

In [1]:
```
class RecursiveParser :
    def __init__(self) :
        self.position = 0

    def expect(self,c):
        if self.source[self.position] == c:
            self.position += 1
        else:
            print(c,'expected at position',self.position)

    def S(self):
        if self.source[self.position] == 'a':
            print('S -> aSa')
            self.position += 1
            self.S()
            self.expect('a')
            return
        if self.source[self.position] == 'b':
            print('S -> bSb')
            self.position += 1
            self.S()
            self.expect('b')
            return
        if self.source[self.position] == 'c': # And this is why you can't have an epsi
            print('S -> c')
            self.position += 1
            return

    def parse(self,source):
        self.source = source + '#' # Poor man's eof
        self.position = 0
        self.S()
        self.expect('#')
```

In [2]:
```
rp = RecursiveParser()
rp.parse('abaacaaba')
```

```
S -> aSa
S -> bSb
S -> aSa
S -> aSa
S -> c
```