Monday, September 13, 2021 10:48 AM

Predictive Recursive descent parsing

1. One parse function per non-terminal
2. When parse_X() is called
   - success :- part of the input that corresponds to X is "consumed"
     - nothing else is consumed
     - to consume X, we need to consume the RHS of one of the rules of X
   - failure : syntax_error() and exit
     no recovery

S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

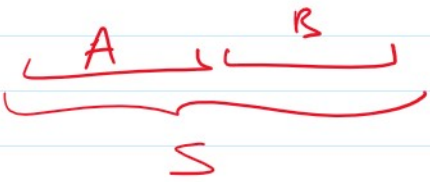
```
parse_input()
{
    parse_S();
    expect(END_OF_FILE);
}
```

Project
input -> tokens_section INPUT_TXT
parse_input()

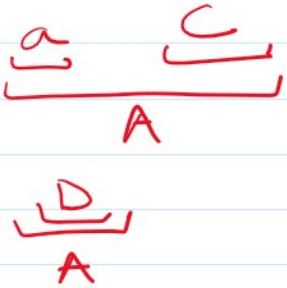
```
parse_input()
{
    parse_tokens_section();
    expect(INPUT_TXT);
    expect(END_OF_FILE);
}
```

S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

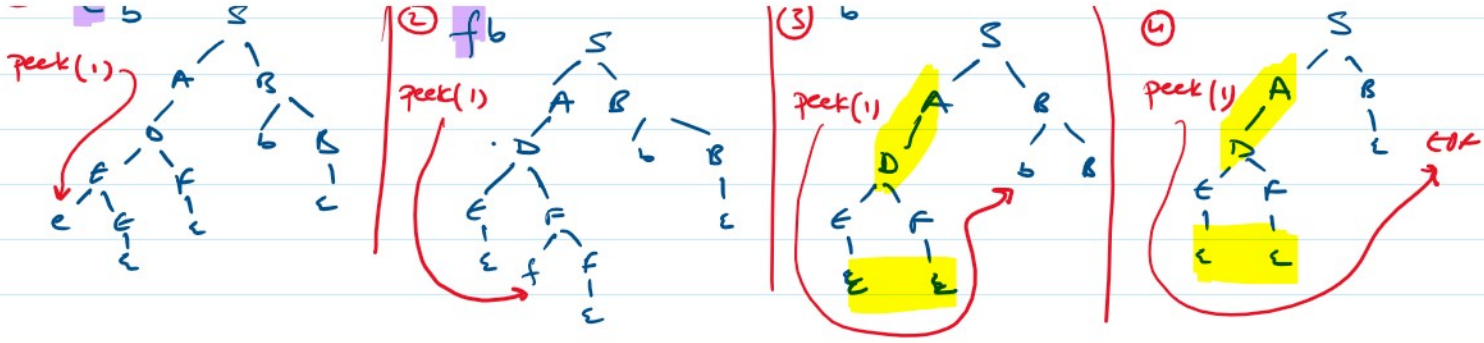
```
parse_S()
{
    parse_A();
    parse_B();

}
```



S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

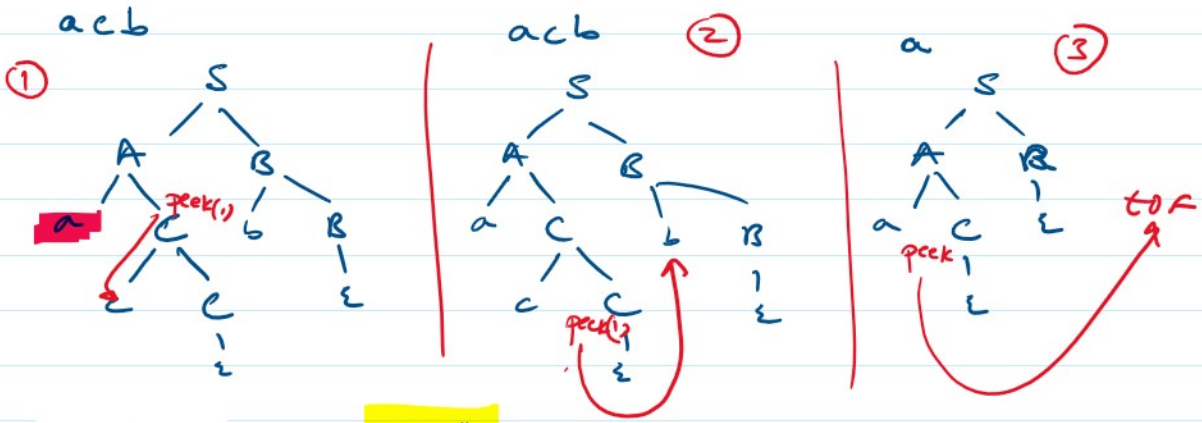



```
parse_A()
{
    t = peek(1);
    if (t.token_type == a_type)
    {
        expect(a_type);
        parse_C();
    } else if ((t.token_type == e_type) (1)
               (t.token_type == f_type) (2)
    {
        parse_D();
    } else if ((t.token_type == b_type) (3)
               (t.token_type == EOF)) (4)
    {
        parse_D()
    } else
        syntax_error();
}
```



S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

```
parse_C()
{
    t = peek(1);
    if (t.token_type == c_type) ①
    {
        expect(c_type);
        parse_C();
    }
    else if ((t.token_type == b_type) ||
             (t.token_type == EOF))
        return;  // C -> ε
    else
        syntax_error();
}
```



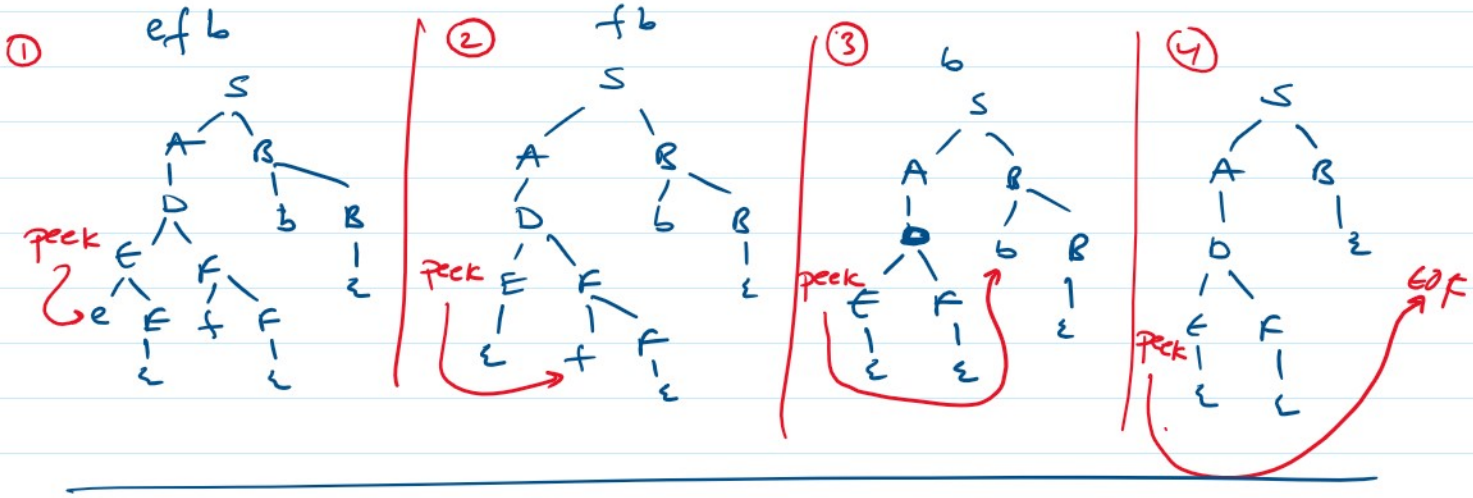
S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

```
parse_D()
{
    parse_E();
    parse_F();
```

}

S -> A B
A -> a C | D
C -> c C | ε
D -> E F
E -> e E | ε
F -> f F | ε
B -> b B | ε

```
parse_E()
{
    t = peek(1);
    if (t.token_type == e_type)  ①
    {
        expect(e_type);
        parse_E();
    } if ((t.token_type == f_type)  ②
          (t.token_type == b_type)  ③
          (t.token_type == EOF))  ④
    else
        return;
    syntax_error();
}
```



Given a grammer, we are going to calculate FIRST sets for all symbols including ε and FOLLOW sets for non-terminals

FIRST sets can contain terminals and/or ε

1. $a \in FIRST(A)$ where A is a terminal or non-terminal and a is a terminal

if $A \overset{*}{\Longrightarrow} a\,w$ ← sequence of terminals and non-terminals

A generates a sequence of symbols starting with a

2. $\varepsilon \in FIRST(A)$ if and only if

$A \overset{*}{\Longrightarrow} \varepsilon$

In this class, we calculate FOLLOW sets for non-terminals

FOLLOW sets can contain terminal and $

$ is a new notation for EOF.

ε cannot be in a FOLLOW set.

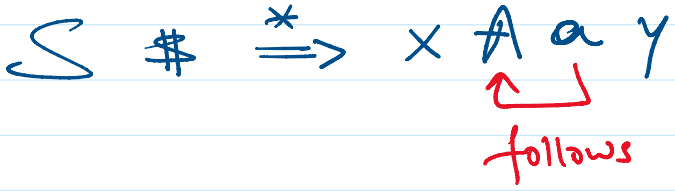
the mighty $ sign

the mighty

In this class $ = EOF

$a \in FOLLOW(A)$ if and only if

$S \$ \overset{*}{\Longrightarrow} x\,A\,a\,y$


follows