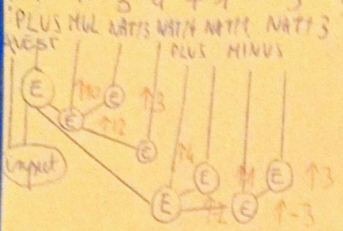
Analyse grammaticale :
① Analyse lexicale → lexème
② Analyse syntaxique → arbre d'analyse

? + * 3 4 + 1 - 3 ? × # 1 / # 1 2 eof



Expressions préfixées AVEC MÉMOIRE

E ↓l ↑R → NAT ↑n R := n
entrée sortie | (#) CALC ↑m R := l[m]
| PLUS E↓l↑v1 E↓l↑v2 R := v1+v2
| MULT E↓l↑v1 E↓l↑v2 R := v1×v2
| DIV E↓l↑v1 E↓l↑v2 R := v1/v2
| MINUS E↓l↑v1

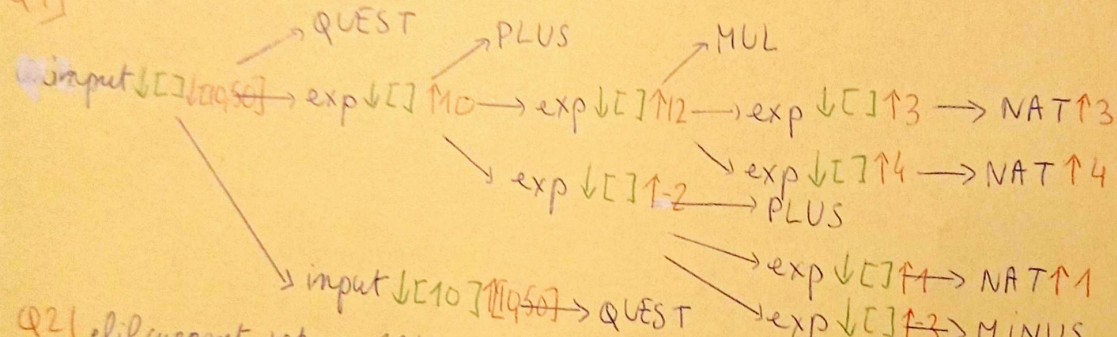
input ↓lin ↑lout → (?) QUEST E↓lin↑v input ↓lin↑lout ↑lres
| (eof) END lout := lin lout := lres

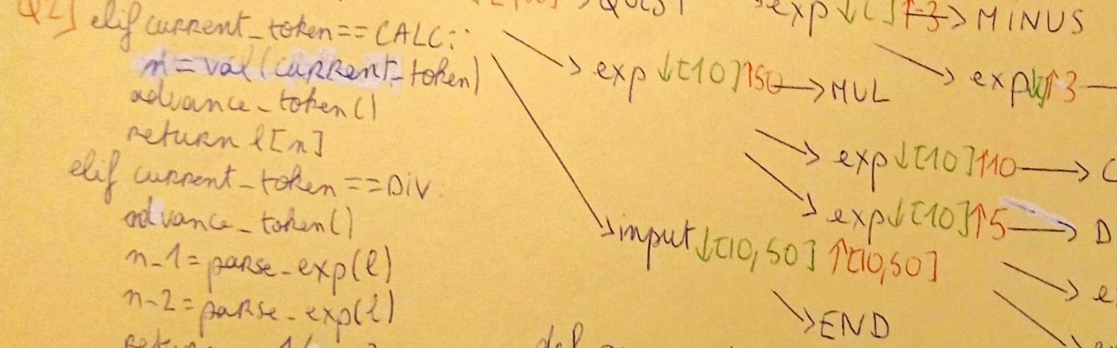
MAIN → input ↓[] ↑l

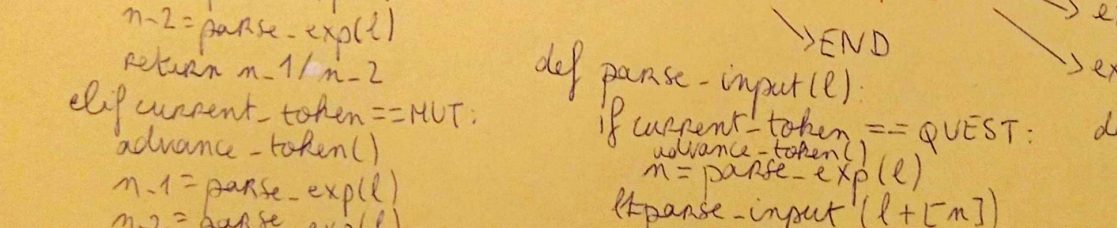
- Attributs synthétisés = sortie de chaque analyseur
- Attributs hérités = entrée de chaque non terminal (NT)

TD1 TL2 : Extension / réduction du projet TL1

Q1]







Q2]
```
elif current_token == CALC:
    n = val(current_token)
    advance_token()
    return l[n]
elif current_token == DIV:
    advance_token()
    n_1 = parse_exp(l)
    n_2 = parse_exp(l)
    return n_1 / n_2
elif current_token == MUL:
    advance_token()
    n_1 = parse_exp(l)
    n_2 = parse_exp(l)
    return n_1 * n_2
else:
    raise SyntaxError:
        print("Expected '/', '*', '+', NAT, # or '-'")
```

```
def parse_input(l):
    if current_token == QUEST:
        advance_token()
        n = parse_exp(l)
        return parse_input(l + [n])
    elif current_token == END:
        return l
    else:
        raise SyntaxError:
            print("Expected '?' or 'eof'")
```

```
def parse_main():
    advance_token
    parse_input([])
```

Q5] Au lieu de reconnaître (QUEST exp)* eof, on reconnaît (exp QUEST)* eof. En fait, QUEST n'est ici pas utile. Il suffit de modifier parse_input pour reconnaître d'abord l'expression.

Q6] MULT NAT↑2 NAT↑5 QUEST END
=> × 2 10 ? eof => [10]

Q7] ça boucle à l'infini (input appelé à l'os)

Q3] QUEST MULT NAT↑2 NAT↑5 END
=> [10]