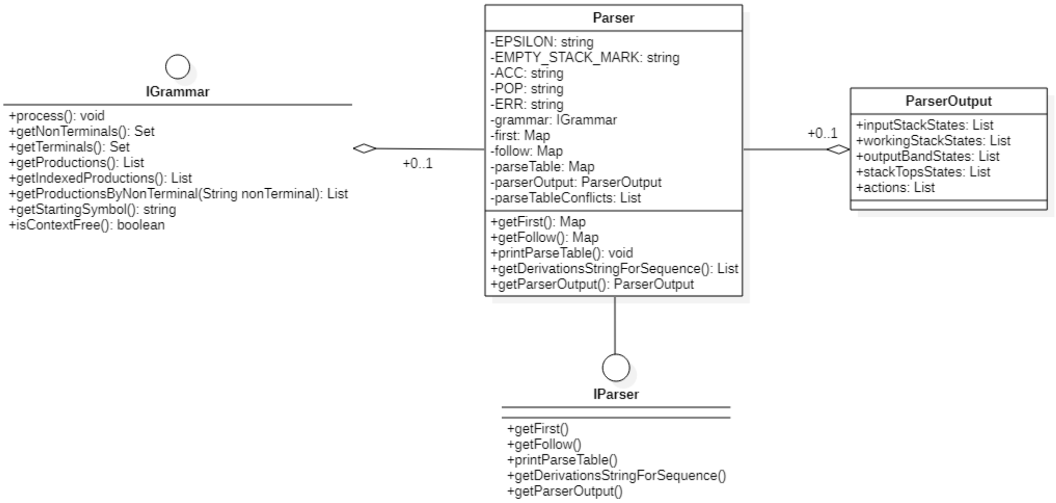
https://github.com/alex-nedelcu/formal-languages-compiler-design



Parser(IGrammar grammar):

Parser initialization overviews the following:
- sets Parser grammar to received argument
- asserts grammar is context free
- initializes First
- computes First
- initializes Follow
- computes Follow
- initializes Parse Table
- computes Parse Table

initializeFirst():

This method initializes First for both *terminals* and *non-terminals* as:

- For each *terminal*, First is the terminal itself.
- For each *non-terminal*, First is an empty list.

For terminals, initialization of First is now complete.
For non-terminals, initialization continues.

For each production (e.g. S -> aB), the source's First (e.g. First(S)) extends with the first symbol from target (e.g. a), if the first symbol from target:

- Is Epsilon
- Is terminal and does not already exist in source's First

computeFirst():

This method computes First for both terminals and non-terminals.

For each production (e.g. S -> aB), it takes the target symbols one by one (e.g. a, then B).
It extends the First of source (e.g. First(S)) with First of current symbol minus Epsilon (e.g. First(a) – Epsilon).
If First of current target symbol does NOT contain Epsilon, method skips to next production.

The method loops over all productions again and again until no changes happen to any First.

initializeFollow():

This method initializes Follow for *non-terminals* as:

- If non-terminal is starting symbol, Follow = Epsilon
- Else, Follow = empty list

computeFollow():

This method computes Follow for non-terminals.

For each non-terminal, the method takes all productions with the non-terminal in its RHS.
For each such production, the method identifies the position of the non-terminal in the RHSs.
For all such positions, next symbol (position + 1) becomes current symbol:

- Follow(non-terminal) extends with First(current symbol) minus Epsilon.
- If First(current symbol) has Epsilon, move to next symbol if exists and repeat above step.
- If method reached end of RHS, Follow(non-terminal) extends with Follow(source)
- Else, stop.

The method loops over all non-terminals again and again until no changes happen to any Follow.

extractRowKeysFromGrammar(IGrammar grammar):

This method returns a Set of:

- Non-terminals of Grammar
- Terminals of grammar
- Empty stack mark

extractColumnKeysFromGrammar(IGrammar grammar):

This method returns a Set of:

- Terminals of grammar
- Empty stack mark

computeParseTable():

This method computes the Parse Table for the Parser's grammar.

To compute the parse table, the method first obtains:

- Terminals from grammar
- Non-terminals from grammar
- Row Keys from grammar
- Column Keys from grammar

Then it indexes each production in the grammar.

For each row key, the method takes column keys one by one.

```
-   IF row key = non-terminal
    AND column key = terminal
    AND column key != epsilon,
    for each production:
        o   alpha = unique target of production
        o   compute First(alpha)
        o   if First(alpha) contains column key, add production index to Parse Table
-   ELSE IF row key = non-terminal
    AND Follow(row key) contains column key
    for each production:
        o   alpha = unique target of production
        o   compute First(alpha)
        o   if First(alpha) contains Epsilon, add production index to Parse Table
-   ELSE IF row key = column key
    AND row key = terminal
        o   Add “POP” to Parse Table
-   ELSE IF row key = Empty stack mark
    AND column key = Empty stack mark
        o   Add “ACC” to Parse Table
-   ELSE
        o   Add “ERR” to Parse Table
```

getDerivationsStringForSequence(String sequence):

To compute derivations string, the method first obtains:

- Indexed productions
- Input Stack (alpha)
- Working Stack (beta)
- Output band (pi)

The method initializes input stack:

- Push Empty stack mark
- In reverse order, push each character in the sequence into input stack

The method initializes working stack:

- Push Empty stack mark
- Push grammar starting symbol

This method generates the derivation string for the given sequence.
While not STOPPED, the method:

- Takes from Parse Table the next action:
    - From Parse Table matrix, x = workingStackTop, y = inputStackTop
    - i.e. Action = ParseTable[workingStackTop][inputStackTop]
- IF action is production index:
    - Get production with given index
    - Pop from working stack
    - In reverse order, Push RHS of production to working stack
    - Add production index to output band
- ELSE IF action is “POP”
    - Pop top of input stack
    - Pop top of working stack
- ELSE IF action is “ACC”
    - Sequence is ACCEPTED
- ELSE
    - STOP

IF outputBand is empty AND sequence is accepted, add Epsilon to outputBand.
IF sequence is NOT accepted, output band = List(“ERR”)
Return outputBand