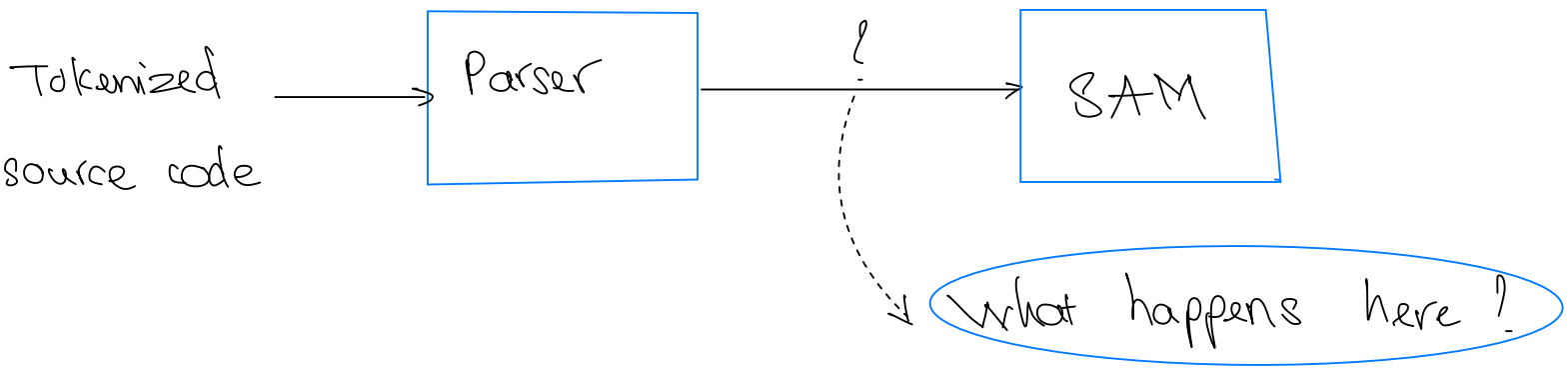
CODE GENERATION FOR SAM

Motivation question:

How do the (Recursive-descent) Parser generates code for SAM?



Short-answer: We can generate SAM code from the Parser

How actually?

Recalls in "Recursive-Descent Parser: Implementation", we generates code from the Parsing Table. However, the method generated return nothing (void), like:

```
void parse_S () :
  switch (token) :
    case num :
    case "(" :
        parse_E()
        parse_S'()
        return
```

Now we fix this to return String of SAM code. Then, after recursively run the Parse, we concat the strings of SAM code to get the final SAM program.

```
String parse_S () :
  switch (token) :
    case num :
    case "(" :
      str_E = parse_E()
      str_S' = parse_S'()
      return str_E + str_S'
```

concat

Genenerate SAM code : Implementation

To actually spits out SAM code, we create a getExpr method that maps the expression to corresponding SAM code.

```
String getExpr () :
    switch (token) :
        case num :
            return " PUSHIMM {num} "
        case " + " :
            return " ADD "
        case " $ " :
            return " STOP "
```

input token

output SAM code

Complete example:

With this grammar :

$$
\begin{aligned}
S &\rightarrow E S' \\
S' &\rightarrow \varepsilon \\
S' &\rightarrow + S \\
E &\rightarrow num \\
E &\rightarrow (S)
\end{aligned}
$$

And this predictive table:

	num	+	(	)	$
S	$\rightarrow ES'$		$\rightarrow ES'$		
S'		$\rightarrow +S$		$\rightarrow \varepsilon$	$\rightarrow \varepsilon$
E	$\rightarrow num$		$\rightarrow (S)$		

We can have this compiler :

next page

SAM code generator

```
String getExpr:
    swith (token):
        case "num":
            return "PUSHIMM {num}"
        case "+":
            return "ADD"
        case ")":
        case "$":
            return "STOP"
```

```
String parse_S ():
    switch (token):
        case num:
        case "(":
            str_E = parse_E()
            str_S' = parse_S'()
            return str_E + str_S'
```

```
String parse_S' ():
    swith (token)
        case "+":
            sam_str = getExpr()
            token = input.read()
            str_S = parse_S()
            return str_S + sam_str
        case ")"
        case "$":
            return getExpr()
```

```
String parse_E():
    swith (token)
        case num:
            sam_str = getExpr()
            token = input.read()
            return sam_str
        case "(":
            token = input.read()
            str_S = parse_S()
            if token != ")": throw Error
            token = input.read()
            return str_S
```

Example run using (2 + 3)

parse S |(2 + 3)

parse E |(2 + 3)

parse S (|2 + 3) "PUSHIMM 2 ; PUSHIMM 3 ; STOP"

E S'

parse E (|2+3) returns PUSHIMM 2

parse S' (2|+ 3) "PUSHIMM 3 ; STOP"

parse S (2 +|3)

- parse E (2 +|3) return PUSHIMM 3
- parse S' (2 + 3|) return STOP