**1.4 Definitions of SFP**

We modify FP into SFP. The definitions of SFP are as follows.

<program> ::= <statements>

<statements> ::= <statement>+

<statement> ::= { = <identifier> <parameter> } |

{ loop <identifier> <statements> }

{ print <parameter> }

<parameter> ::= <function-call> | <identifier> | <number>

<function-call> ::= { <function-name> <parameter> <parameter> }

<function-name> ::= + | \*

The **identifier and number definitions for SFP are the same as those in FP**. Note that the negative

sign “–” in SFP is important. There are no character strings or Boolean in SFP.

P -> T

T -> S T | S

S -> { = x R } | { l x T } |{ print R }

R -> C | x | n //n as non-terminal or terminal

C -> { f R R}

f -> + | \*

**3 Project PT -- Syntax Analysis of SFP using Parsing Table**

In this project, you need to build a parser from scratch, based on the concept you learnt in class.

Consider the SFP language. First, build the action table and goto table of the parsing table

manually for the SFP language.

In the **action table**, you have **tokens**: {, }, l (loop), p (print), =, f (+, \*), n (integer and float), x

(identifier). You also have $ in the action table.

In the **goto table**, you have **non-terminal symbols** P (program), T (statements), S (statement), R

(parameter), C (function call). If you need additional symbols, just use additional upper case

letters to represent the symbol.

|  |  |
| --- | --- |
| Non-terminal | Represents |
| P | Program |
| T | Statements |
| S | Statement |
| R | Parameter |
| C | Function Call |