Code Structure

Program Entry & Exit

```
Start {
.
}end
```

Variable Declaration -

Syntax- Type Id; / type Id, Id, Id;
Type-Integer, float, string, void
Identifier- [a-zA-Z]

Variable assign -

```
integer x = 2;
Integer x, integer f=5, integer z = 10;
Int x,y, z
float x = 2;
float x=2, float f=5, float z =10;
String ms="Hello, EasyScript!";
String m = "Hello, EasyScript!", l="Hello, EasyScript!";
```

Print Statement

```
show(message)
show(“message”+variable+“message”)
```

Exp= the summation is 5 and continue..

Input Statement

```
x = getInput()  # Define Method later
```

Conditional Statement

```
if (x > 5) {
  print("x is greater than 5")
}
```

```
Else if(condition)
{
  print("x is not greater than 5")}


 else {
  print("x is not greater than 5")}
```

Loop

```
while(condition){
}
```

Function

```
type function id(a, b) {
  return a + b;
}
```

Function Call

```
result = add(5, 7)
```

Assignment

```
Id= a + b;
Id= 1;
```

Return

```
return 0;
return a +b;
```

Language description

It’s structure is divided into three parts.

1. Algo name / Program Name

2. Declaration part / Initialization block
3. Main program block / Main Program

It's more c like also has some syntax like python for example it uses the function show() and the arguments are concatenated with +
Declaration both (variable and function will be at the beginning)
Only while loop is used
Identifier can not start with digit, it will start with only non digit

Language Structure

Algorithm name- ID
Declaration Part- (all the declaration will be here including variable and function declaration_definition)

Commented [1]:

Start {
//statement
}end

Algorithm Implementation with our syntax

```
Algorithm name-fibonacci
Declaration Part- integer t1 = 0, t2 = 1, nextTerm = 0, n=5;
 start{

  show("Fibonacci Series:"+ t1 +","+t2);
  nextTerm = t1 + t2;

  while (nextTerm <= n) {
    show(nextTerm);
    t1 = t2;
    t2 = nextTerm;
    nextTerm = t1 + t2;
  }

}end
```

Updated grammar after professor’s feedback

Commented [2]: @muyiwaojo10@gmail.com Check out this one, I have adjusted some of the rules. The modified rules are changed to red. You can look.

program ::= algorithm_name variable_function_declarations main_block

```
algorithm_name::=‘Algorithm name-’ id
variable_function_declarations::= ‘Declaration block-’ declarations  | ε

declarations ::= declaration declarations_tail
declarations_tail ::= declarations declaration_tail| ε
declaration ::= variable_declaration | function_declaration
Edited: declaration ::= type var_func_declaration_tail
var_func_declaration_tail::= variable_declaration | function_declaration
variable_declaration ::=  variables';'
function_declaration ::= 'function' Id '(' parameters? ')' block


variable_declaration ::= type variables ';'
type ::= 'integer' | 'float' | 'string' | 'void'

variables::= variable variable_tail
variable_tail::= ‘,’ variable variable_tail |  ε
variable ::= id variable2
variable2::= “=” expression | ε

Id ::= string IdTail
IdTail ::= string IdTail | digits IdTail | ε
https://www.javatpoint.com/javacc
value ::=  number | string
string ::= alphanumeric string'
string' ::= alphanumeric string' | ε
alphanumeric ::= [a-zA-Z0-9!@#$%^&*()_+-=[]{};:'",.<>?/\\|]
(String becomes string literal
Number becomes number literal in the code)

number::= digits ('.' digits')?
digits ::= digit digits'
```

digits' ::= digit digits' | ε
digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

function_declaration ::= type 'function' Id '(' parameters? ')' block
parameters ::= parameter parameters_tail
parameters_tail ::= ',' parameter parameters_tail | ε
parameter::= type variables

main_block ::=   'start' block 'end'
block ::= '{' content '}'
content ::= statements | ε

statements ::= statement statements_tail
statements_tail ::= statement statements_tail | ε
statement ::= if_statement | for_statement | while_statement | assignment_statement |
print_statement |comment_statement | break | return

If_statement: if '(' bool_expression ')' block (else_if '(' bool_expression ')' block)* (else block)?
while_statement: WHILE '(' bool_expression ')' block

assignment_statement ::= variable_assignment | function_call_assignment
variable_assignment ::= variables '=' arithmetic_expression ';'

Edited for the error:
Assignment_statement ::= Id '=' Assignment_statement_tail
Assignment_statement_tail ::= variable_assignment | function_call_assignment
variable_assignment ::= arithmetic_expression ';'
function_call_assignment ::= function_call ';'

function_call_assignment ::= variable '=' function_call ';'
function_call ::= 'function' Id '(' parameters? ')'
comment_statement::='#'value'#'
return_statement ::= 'return' arithmatic_expression ';'
break ::= break';'

```
print_statement ::= "show" '(' print_arguments ')' ';'
print_arguments ::= print_argument print_arguments_tail
print_arguments_tail ::= '+'  print_argument  print_arguments_tail | ε
print_argument ::=  ‘ “ ’string ‘ ” ’ | id


expression ::= bool_expression | arithmatic_expression

bool_expression ::= b_term bool_expression_tail
bool_expression_tail ::= OR b_term bool_expression_tail | ε
b_term ::= b_factor b_term_tail
b_term_tail ::= AND  b_term_tail| ε
b_factor ::= NOT bool_value | bool_value
bool_value ::= id | true | false | '(' bool_expression ')' | function_call | compare_expression
compare_expression ::= arithmatic_expression compare_operator arithmatic_expression
compare_operator ::= '<' | '>' | '<=' | '>=' | '==' | '!='

arithmatic_expression ::= a_term arithmatic_expression_tail
arithmatic_expression_tail ::= '+' arithmatic_expression_tail | '-' arithmatic_expression_tail | ε

a_term ::= a_factor a_term_tail
a_term_tail ::= '*' a_term_tail | '/'  a_term_tail | ε

a_factor ::=  number  | id   | '(' arithmatic_expression ')'  | function_call
```

Necessary study materials

https://youtu.be/lVq3kFTkeWg?si=zyAkzTOs88w1T0zu

https://www.geeksforgeeks.org/construction-of-ll1-parsing-table/

https://www.youtube.com/watch?v=oOCromcWnfc

Algorithm name- fibonacci
Declaration- int a,b=10;
   Int function add() { //function body statement statement }

start {
//statement
}end
print_statement ::= "show" '(' print_arguments ')' ';'
print_arguments ::= print_argument print_arguments_tail
print_arguments_tail ::= '+' print_arguments| ε
print_argument ::= ‘ “ ’string ‘ ” ’ | variable

```
nextTerm = t1 + t2;
```

 https://www.javatpoint.com/javacc
https://inscription.uni.lu/Inscriptions/Public/Admission

Slide
https://www.canva.com/design/DAF6Ktc_uHs/7orz4D1-MLfEGfh5N0KB6g/edit
Algorithm

Algorithm name-fibonacci

Declaration block-

    integer t1 = 0,

    t2 = 1,

    nextTerm = 0,

```
    n=5;



start{

    show("Fibonacci Series:" + t1 + "," + t2);

    nextTerm = t1 + t2;



    while (nextTerm <= n) {

        show(nextTerm);

        t1 = t2;

        t2 = nextTerm;

        nextTerm = t1 + t2;
    }


}end
```