-Republic of the Philippines

Polytechnic University of the Philippines-

Sta. Mesa, Manila

Department of Computer Science

COSC 30063 - Principles of Programming Language

Term Project - Project Proposal

Submitted by

Group 1

Aveno, Eleigh Anne S.

Bacsain, Shan Allen B.

Bruce, Meracle L.

Clavenis, Christine Mae N.

Dasalla, John Mathew G.

Enriquez, Earl Cedric F.

Mutya, Mark Joshua F.

Tejada, Allan Christian D.

Course & Section

BSCS 3-2

Submitted to

Professor Ria Sagum

Table of Contents

I. Introduction

"Klak" is an object-oriented programming language designed to teach beginners programming concepts in Tagalog form. Furthermore, it is intended to be simple, and non-intimidating which encourages people, particularly those who are unfamiliar with IT related activities or tools, to become interested in programming. The name of the Programming Language was inspired by the sound made by the keyboard when users or programmers type. The programming language is designed specifically for Filipinos in order to increase the number of students entering programming fields, thereby advancing the country's IT industry. One of the Developers' motivations for creating this project was the scarcity of Filipino-based Programming Languages; especially, there is only one Filipino-based Programming Language that is known, and it is the "Bato" Programming Language, which is based on Ruby. As a result, the developers want to consider creating a Filipino-based language, which will also boost Filipino programming excellence.

The Programming Language is based on the currently existing Python Programming Language. The developers chose "Python" because the majority of the team is knowledgeable with the programming language. An object-oriented programming language was also chosen for this project because it contains concepts such as encapsulation, abstraction, inheritance, and polymorphism that the developers want others to learn.

For the fact that the developers are already familiar with the said Programming Language, they are also well-informed when it comes to its issues, difficulties, and the benefits and features of the said language. Consequently, the developers have made several changes and adjustments to Python, including the translation of the language from English to Tagalog, in order to make a new language called “Klak”, that is greatly beneficial for newbies in programming. Moreover, the developers have also included some implementations (e.g. switch case) that “python” does not have and that will provide better knowledge to the target programmers. Furthermore, the developers have kept the syntax of the Language less complicated, and visually-pleasant or comfortable to the eyes, to achieve its beginner-friendly features.

II. Syntactic Elements of language

1. Character Set

- Alpha = {Uppercase, Lowercase, Digit, Symbol}
    - Uppercase = {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}
    - Lowercase = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}
    - Digit = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    - Symbol = {+, -, *, /, %, =, _, (,), [,], {, }, <, >, !, ^,”,’,, }

2. Identifiers

In this part, several rules are placed when it comes to declaring and defining identifiers.

- Identifier names can be any combination of letters, digits, and the underscore symbol (_); any other special characters will not be accepted. The first character of the Identifier name, however, must only be a Letter.

  Examples:

  Correct:

  ```
  salita Big_1 = “House”;
  numero age  = 1;
  ```

  Incorrect:

  ```
  salita Name! = “Mark”;
  salita 1D = “Kevin”;
  ```

- Identifier Name should have a minimum length of 1 Character and maximum length of 31 Characters.

Examples:

Correct:

```
salita Hello_2;
klase School [
]
```

Incorrect:

salita ;

klase StudentsfromPolytechnicUniversityofthePhilippines [

]

- Identifier names should not contain any international characters, such as é and ñ.

Examples:

Correct:

salita Nue;

klase School [

]

Incorrect:

salita ñue;

klase Jüd[

]

- Keywords should not be used as identifiers.

Examples:

Correct:

salita New;

klase House [

]

Incorrect:

salita edi;

klase edikung [

]

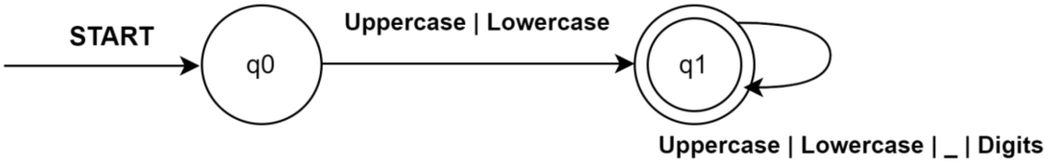


Diagram 1.0. Identifier Deterministic Finite Automaton

3. Operation Symbols

3.a. Arithmetic operations

Operator	Name and Description	Example
+	Addition – for adding numbers.	x + y
-	Subtraction – for subtracting numbers.	x - y
*	Multiplication – for multiplying numbers.	x * y
/	Division – for dividing numbers. Returns the whole quotient including decimal.	x / y
//	Integer Division – for dividing numbers. Returns the floor of division - excluding the remainder/decimal.	x / / y
^	Exponent – for exponent value of a number.	x ^ y
%	Modulo – getting the remainder of a divided number.	x % y

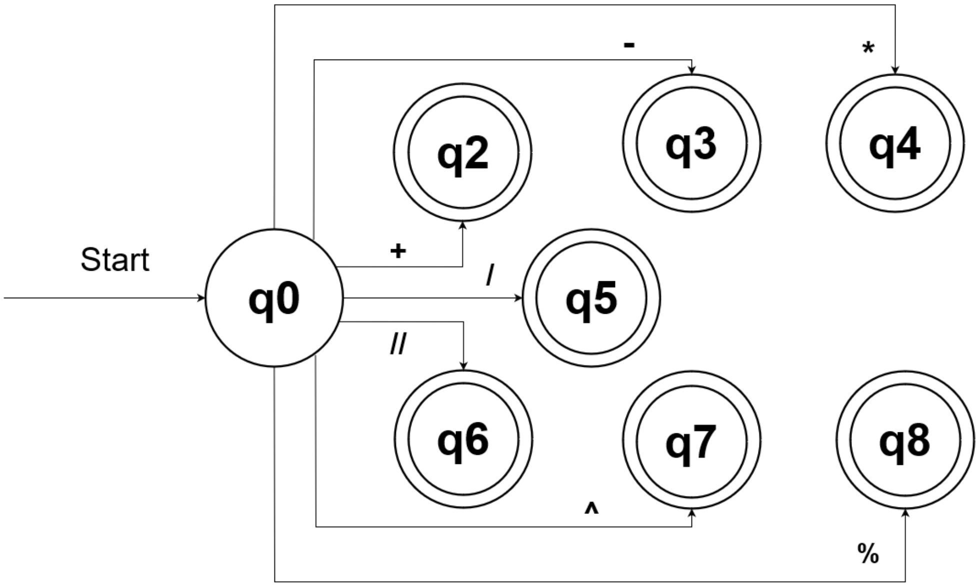


Diagram 2.0. Arithmetic Operations Deterministic Finite Automaton

3.b. Boolean operations

3.b.1. Relational

Operator	Name and Description	Example
<	Less than – compares two numbers with the first number as the lesser value.	x > y
>	Greater than – compares two numbers with the first number as the greater value.	x < y
<=	Less than Equal – compares two numbers with the first number as the less value or equal to the second number.	x <= y
>=	Greater than equal – compares two numbers with the first number as the greater value or equal to the second number.	x >= y
==	Equal to – checks if the first value is equal to the second value.	x == y
!=	Not Equal – checks if the first value is not equal to the second value.	x != y

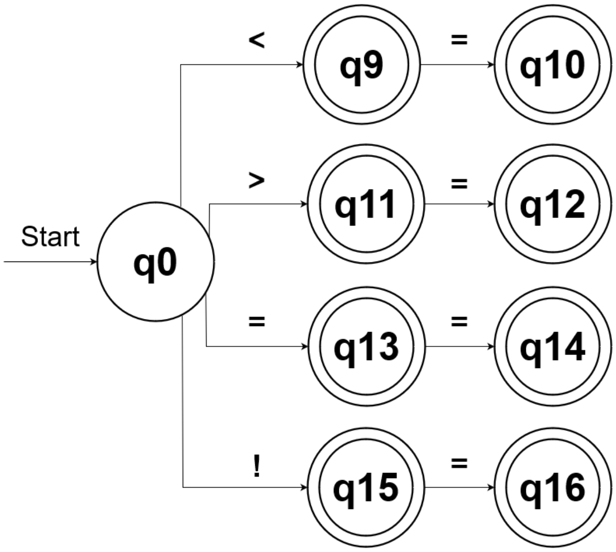


Diagram 2.1. Boolean Operations Deterministic Finite Automaton

3.b.2 Logical

Operator	Description	Example
at	return True if both statements are true.	x<10 at y> 5
oh	return True if one of the statements is true.	x<10 oh y>5
hindi	reverse the output.	hindi(condition)

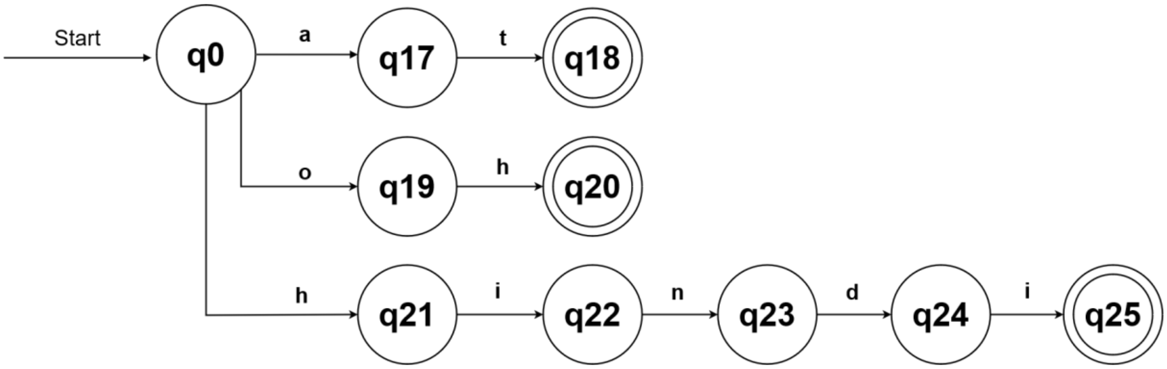


Diagram 2.2. Logical Operations Deterministic Finite Automaton

4. Constants

Constant	Description	Example
numero	A whole number that can be positive or negative.	1
lutang	Represents a floating-point number or whole numbers with decimal.	1.14
karakter	A single unit in the Alpha Character Set.	‘C’
salita	Sequence of characters that would form word or phrases.	“hello”
bul	Represents boolean “true” value.	totoo
bul	Represents boolean “false” value.	mali

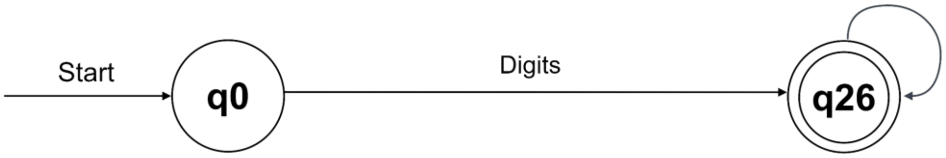


Diagram 3.0. numero Constant Deterministic Finite Automaton

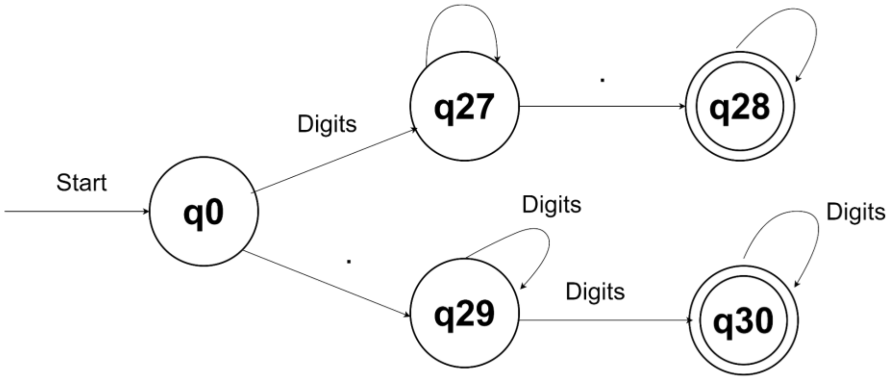


Diagram 3.1. lutang Constant Deterministic Finite Automaton

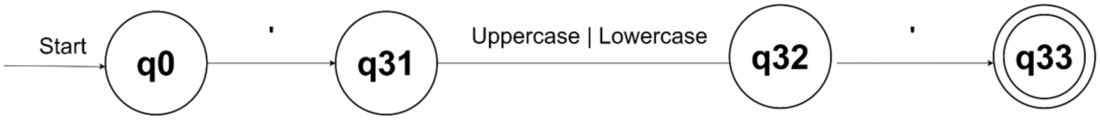


Diagram 3.2. karakter Constant Deterministic Finite Automaton

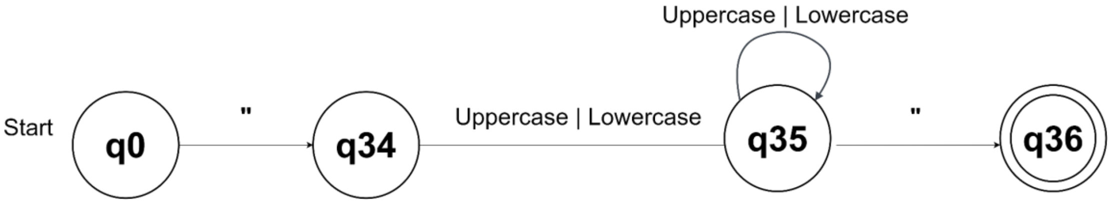


Diagram 3.3. salita Constant Deterministic Finite Automaton

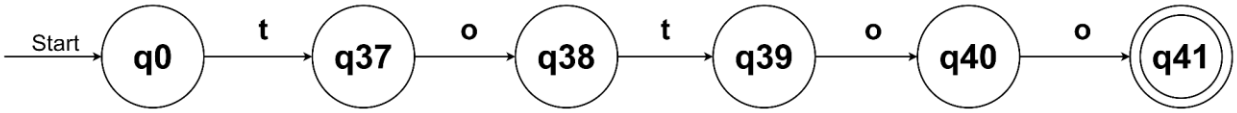


Diagram 3.4. bul ’totoo’ Constant Deterministic Finite Automaton

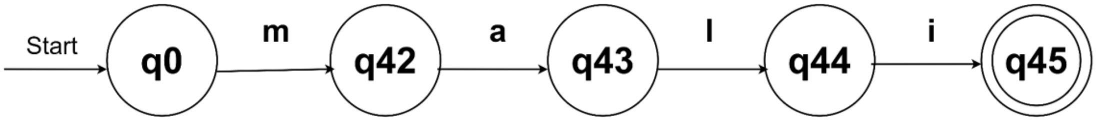


Diagram 3.5. bul 'mali' Constant Deterministic Finite Automaton

5. Keywords and Reserved Words

5.a. Keywords

Keywords	Description
edi	Executed if its "kung" conditional statement partner is false or if all conditional statements (kung and edi kung) are all false, represents an "else" statement of Python Language.
edikung	It is used to make additional conditional statements. It represents the "elif" statement of Python Language
ilimbag	It is used to output any string or object on screen.
lagyan	It is executed to receive input value from user
habang	It is used to make a condition for its corresponding code block to be executed. It represents the "while" statement of Python Language.

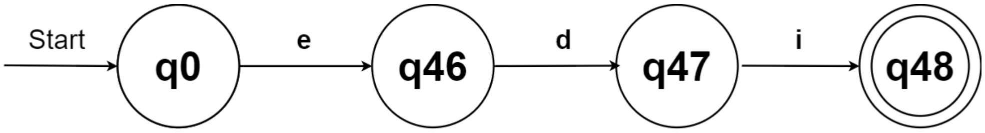


Diagram 4.0. Keyword 'edi' Deterministic Finite Automaton

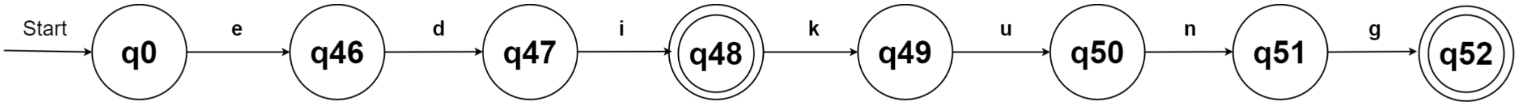


Diagram 4.1. Keyword 'edikung' Deterministic Finite Automaton

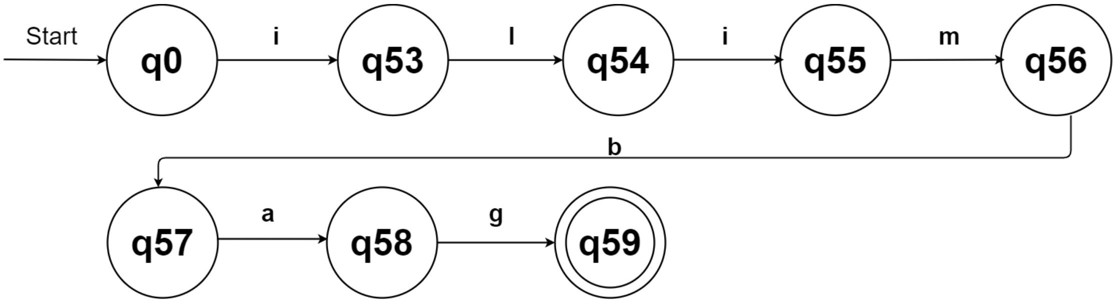


Diagram 4.2. Keyword 'ilimbag' Deterministic Finite Automaton

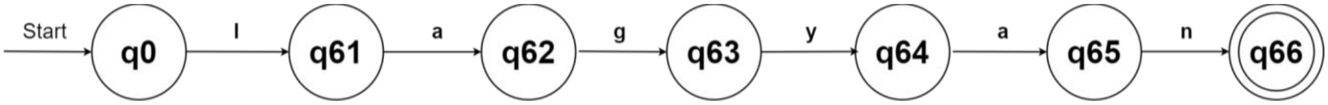


Diagram 4.3. Keyword 'lagyan' Deterministic Finite Automaton

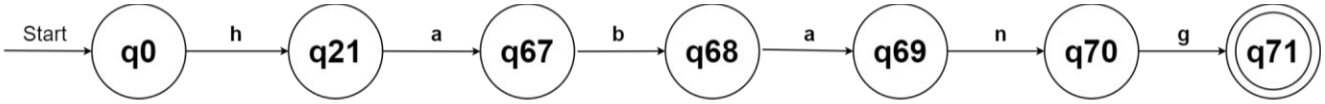


Diagram 4.4. Keyword 'habang' Deterministic Finite Automaton

5.b. Reserved Words

Reserved Words	Description
numero	It represents integers which are digits and may include a dot.
lutang	It represents floating point digits that have 7 digit precision.
salita	It represents collection of characters
karakter	It represents characters
bul	It represents boolean values
sira	It used to break out of a loop
tuloy	It used to continue next iteration of loop
kabtol	It represents a switch case, which executes a statement from multiple cases
pindutan	It represents the “case” of the switch case, which is a statement that can be chosen to be executed depending on the value of the argument passed to the “kabtol”
ilabas	It used to exit a function and return a value
labasmuna	It used to suspend a function to return a generator - represents the “yield statement”
wala	It represents a “void” data type, which is a data type that has no value.
kung	It used to make a conditional statement. It represents the “if” statement of Python Language.
subok	It executes statements inside of it for exception handling
puwera	It executes statements inside of it if the “subok” statements created an error/errors
pal	It used to define a function
klase	It used to define a class - represents the “class statement” of python

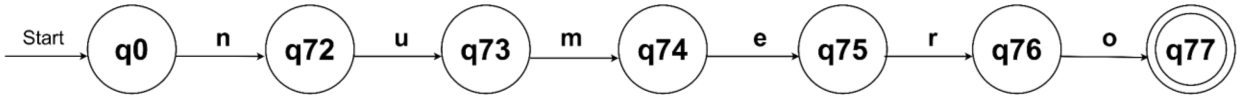


Diagram 5.0. Reserved Words 'numero' Deterministic Finite Automaton

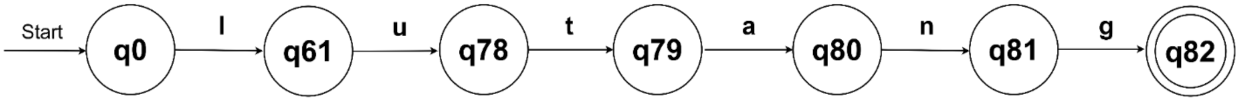


Diagram 5.1. Reserved Words 'lutang' Deterministic Finite Automaton

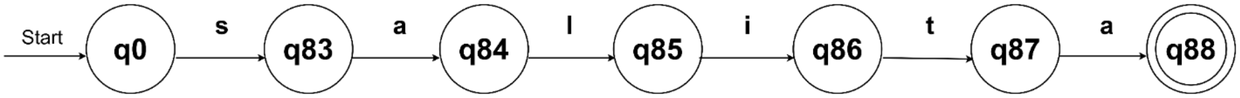


Diagram 5.2. Reserved Words 'salita' Deterministic Finite Automaton

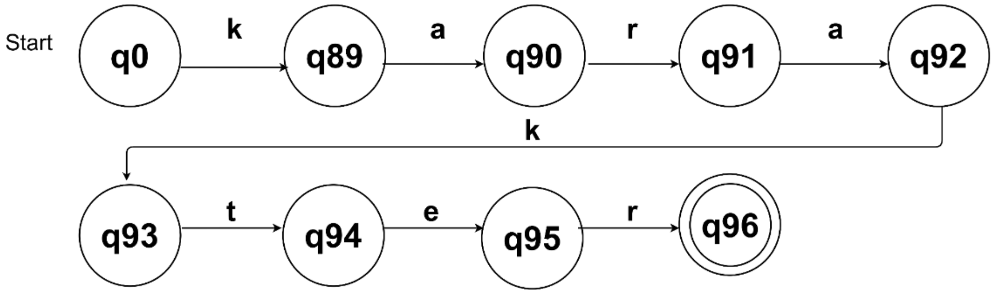


Diagram 5.3. Reserved Words 'karakter' Deterministic Finite Automaton

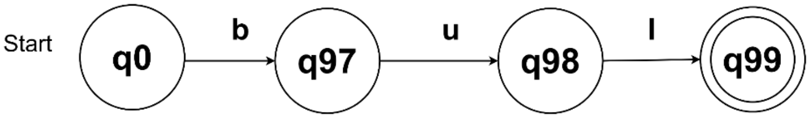


Diagram 5.4. Reserved Words 'bul' Deterministic Finite Automaton

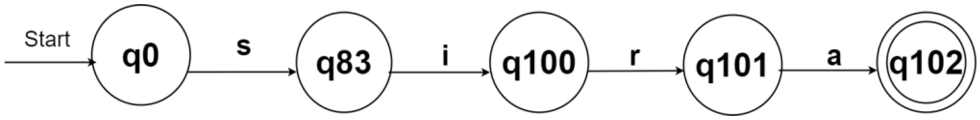


Diagram 5.5. Reserved Words 'sira' Deterministic Finite Automaton

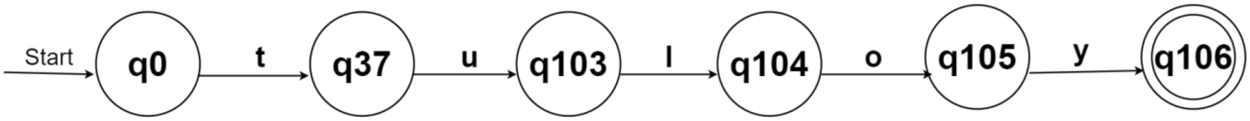


Diagram 5.6. Reserved Words 'tuloy' Deterministic Finite Automaton

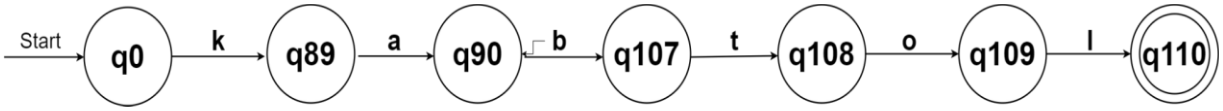


Diagram 5.7. Reserved Words 'kabtol' Deterministic Finite Automaton

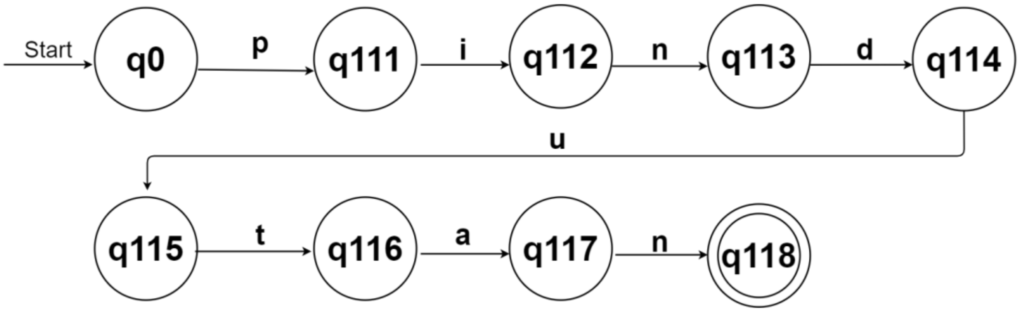


Diagram 5.8. Reserved Words 'pindutan' Deterministic Finite Automaton

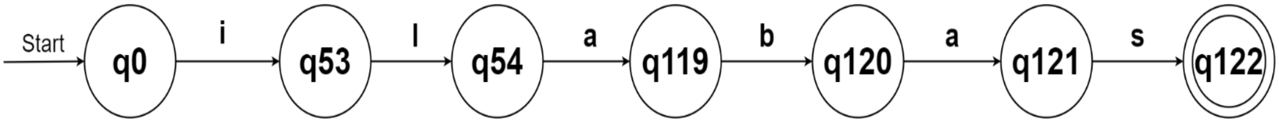


Diagram 5.9. Reserved Words 'ilabas' Deterministic Finite Automaton

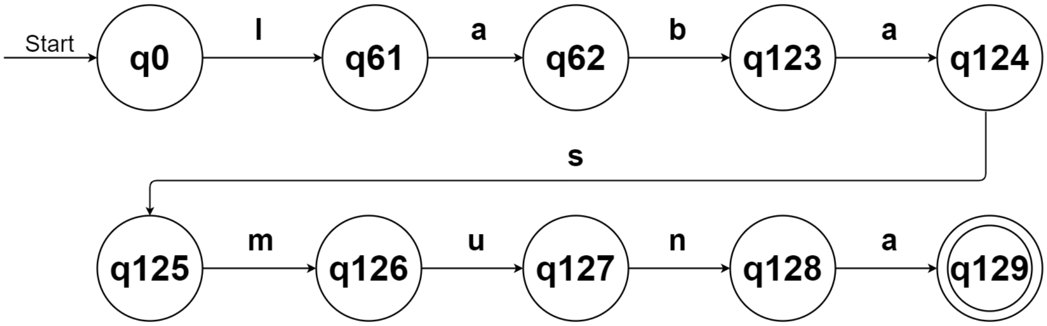


Diagram 5.10. Reserved Words 'labasmuna' Deterministic Finite Automaton

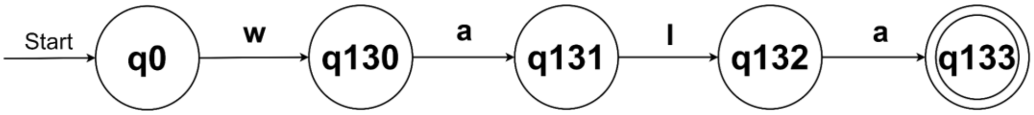


Diagram 5.11. Reserved Words 'wala' Deterministic Finite Automaton

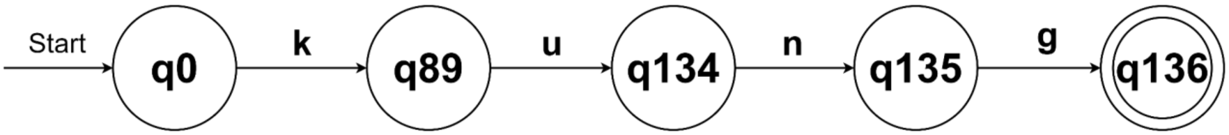


Diagram 5.12. Reserved Words 'kung' Deterministic Finite Automaton

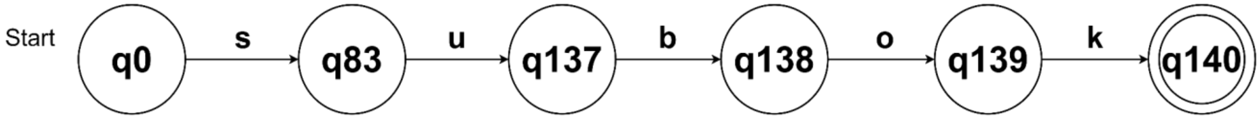


Diagram 5.13. Reserved Words 'subok' Deterministic Finite Automaton

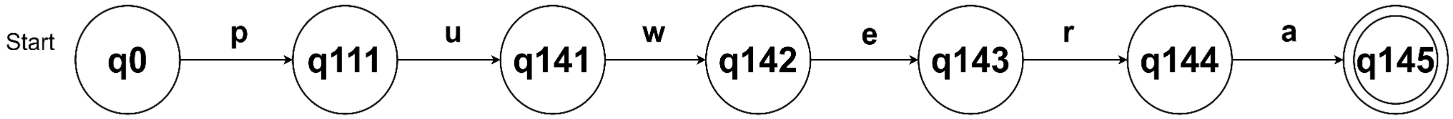


Diagram 5.14. Reserved Words 'puwera' Deterministic Finite Automaton

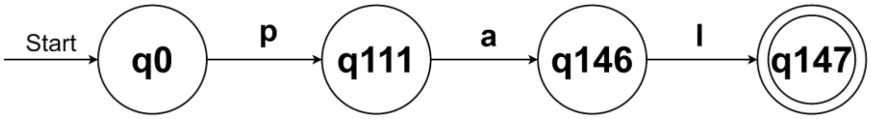


Diagram 5.15. Reserved Words 'pal' Deterministic Finite Automaton

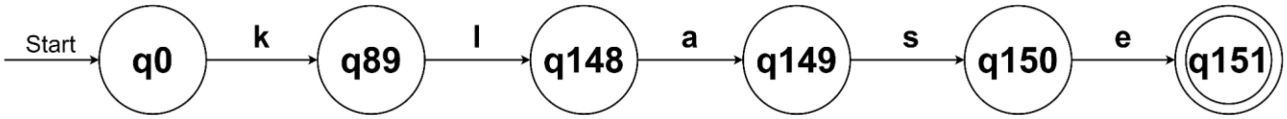


Diagram 5.16. Reserved Words 'klase' Deterministic Finite Automaton

6. Noise words

Noise Words	Description	Example
ay	It is used after the conditional statement in order to emphasize the statement or code block that will be executed if the conditional statement is true.	kung a>b[ay ilimbag(a) ;]
simula	It is used to indicate the start of a code block.	numero pal Function(numero a) [simula a = 3 ilimbag(a) ;]
wakas	It is used to indicate the start of a code block.	numero pal Function(numero a) [simula a = 3 wakas ilimbag(a) ;]
kaunaunahan	It is used to indicate the start of the whole source code.	kaunaunahan klase New_Class[numero pal Square(numero a)[simula a = a ^ 2 ilimbag(a) ;] Square(4)]
kaduluduluhan	It is used to indicate the end of the whole source code.	kaunaunahan klase New_Class[numero pal Square(numero a) [simula a = a ^ 2; ilimbag(a) ;]

Noise Words	Description	Example
		Square(4); kaduluduluhan]
puna	It is used to indicate that there is a comment statement next to it.	simula a = a ^ 2 ; ilimbag(a) ; puna .. Resulta ng “a”

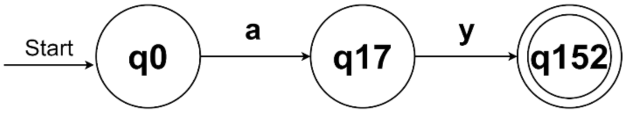


Diagram 6.0. Noise Words ‘ay’ Deterministic Finite Automaton

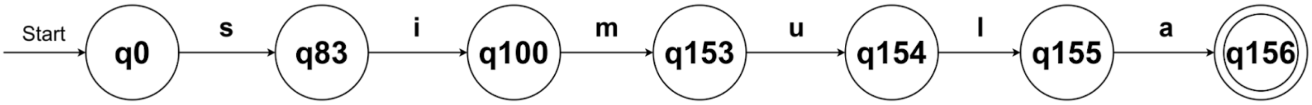


Diagram 6.1. Noise Words ‘simula’ Deterministic Finite Automaton

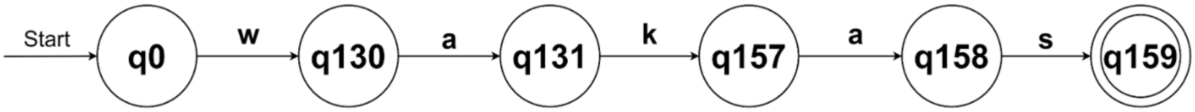


Diagram 6.2. Noise Words ‘wakas’ Deterministic Finite Automaton

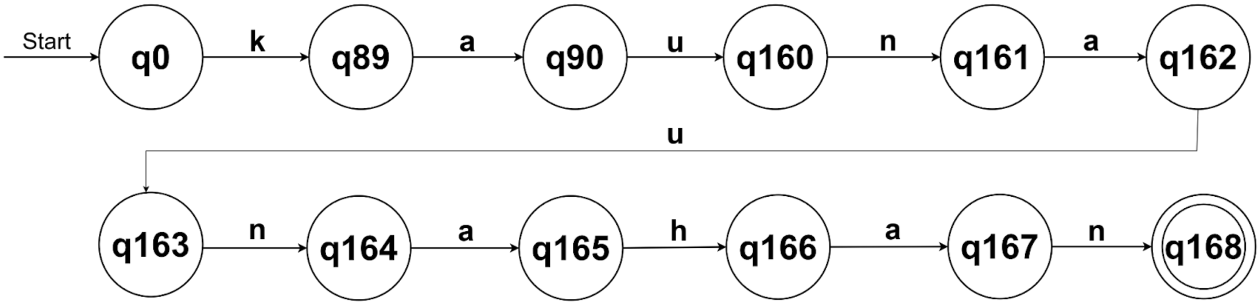


Diagram 6.3. Noise Words ‘kaunaunahan’ Deterministic Finite Automaton

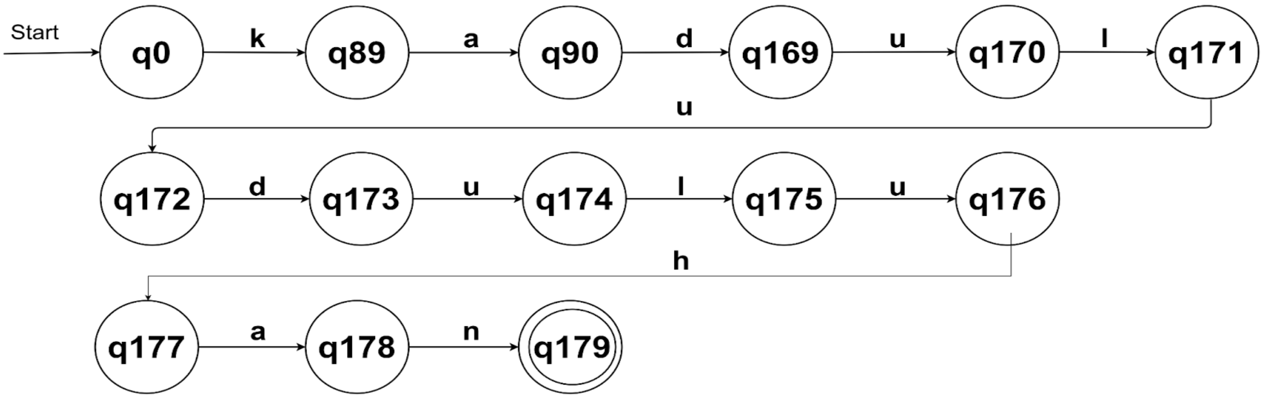


Diagram 6.4. Noise Words 'kaduluduluhan' Deterministic Finite Automaton

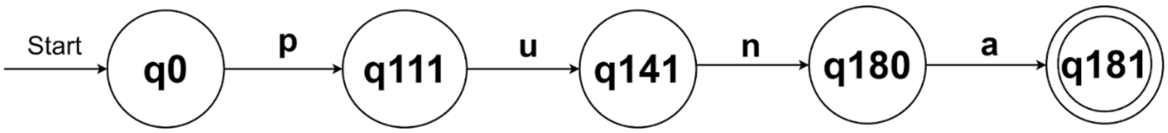


Diagram 6.5. Noise Words 'puna' Deterministic Finite Automaton

7.Comments

Comments	Description	Syntax
..	Single-Line Comment - it is used to document source code using a single line.	.. Comment
…	Multi-Line Comment - it is used to document source code using a single line.	…Comment Comment Comment Comment…

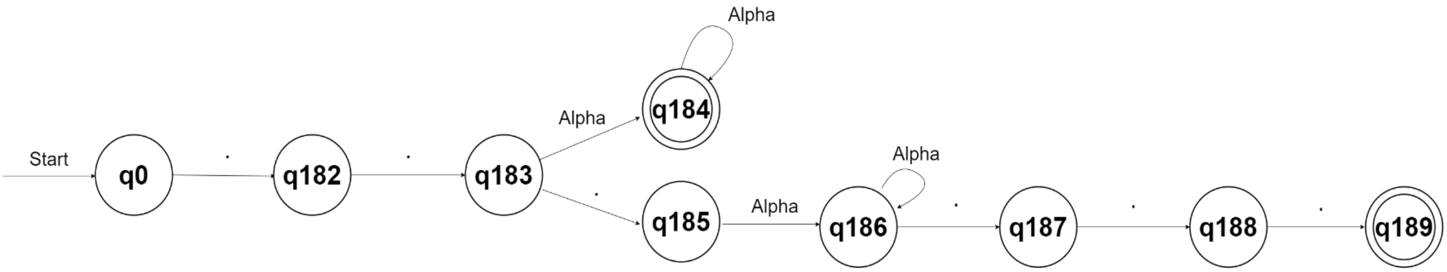


Diagram 7.0. Single and Multi-line Comment Deterministic Finite Automaton

8. Blanks

Blanks in programs use single-character instructions contained within braces and numbers contained within brackets. Each number is an 'instruction' in the sense that 'executing' a number means to push it onto the stack.

“Klak ”will use blanks (spaces) syntactically. “Klak seeks to be explicit and highly prioritizes readability. This will lead to some very concise and user-friendly syntax. But It will be also sensitive when being used for declaring syntax for division. Other spaces will no longer have purpose.

9. Delimiters and brackets

9.a Delimiter

Delimiter	Description
,	It divides more than one parameter in a function declaration and also divides more than one variable in a multiple variable declaration.

9.b Brackets

Bracket	Description
[ ]	It defines the start and end of a code block. It is also used to declare elements of an array, and to declare list literals of a list.
( )	It is used for inputting or outputting variables.

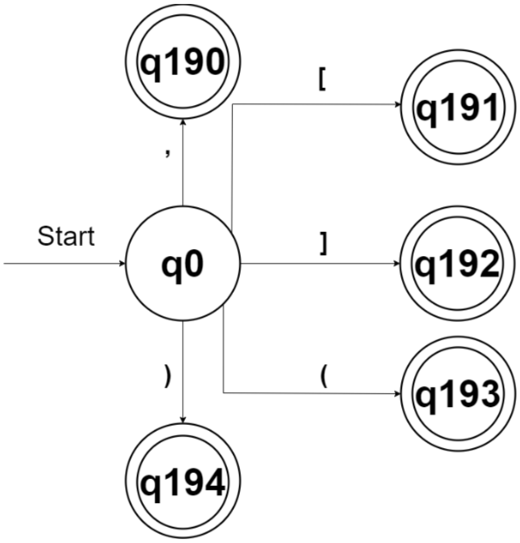


Diagram 8.0. Delimiter and Brackets Deterministic Finite Automaton

10.Free-and-fixed-field formats

The Free-Field format was chosen to be implemented in the Klak Programming Language to let the users have their own spacing style for their source code. Moreover, if the Language does not have limitations when it comes to statement positioning, the users, which are beginners, will have less errors received when experimenting with the Programming Language.

11. Expression

Rules:

- Expressions are made up of numeric values, operators, and, in some cases, parenthesis.
- Expressions can be mathematical/arithmetic or boolean.
- These expressions use arithmetic, relational, and logical operators.
- Expressions are not space sensitive.
- It follows the PEMDAS (Parentheses, Exponent, Multiplication, Division, Addition, Subtraction) rule to determine which operators should be evaluated first.

Order of Precedence from Highest to Lowest:

1. Parentheses ( )
2. Exponent (^)
3. Multiplication, Division, Integer Division and Modulus (*, /, //, %)
4. Addition and Subtraction (+,-)
5. Relational Operators (<, <=, >, >=, ==, !=)
6. Not Operator (hindi)
7. And Operator (at)
8. Or Operator (oh)

11.a. Mathematical/Arithmetic expressions

Mathematical/ Arithmetic Expressions	**Syntax**
+	Variable_name = num1 + num2 ;
-	Variable_name = num1 + num2 ;
*	Variable_name = num1 * num2 ;
/	Variable_name = num1 / num2 ;
//	Variable_name = num1 // num2 ;
^	Variable_name = 2^2 ;
%	Variable_name = num1 % num2 ;

11.b. Boolean expression

11.b.1. Relational

Relational Expression	Syntax
<	resulta = operand1 < operand2 ;
	ilimbag(operand1 < operand2) ;
	kung operand1 < operand2 ;
>	resulta = operand1 > operand2 ;
	ilimbag(operand1 > operand2) ;
	kung operand1 > operand2 ;
<=	resulta = operand1 <= operand2 ;
	ilimbag(operand1<= operand2) ;
	kung operand1 <= operand2 ;
>=	resulta = operand1 >= operand2 ;
	ilimbag(operand1 >= operand2) ;
	kung operand1 >= operand2 ;
==	resulta = operand1 == operand2 ;
	ilimbag(operand1 == operand2) ;
	kung operand1 == operand2 ;
!=	resulta = operand1 != operand2 ;
	ilimbag(operand1 != operand2) ;
	kung operand1 != operand2 ;

11.b.2 Logical

Logical Expression	Syntax
at	resulta = a at b ;
	ilimbag((a > b) at (c < d)) ;
oh	resulta = a oh b ;
	ilimbag((a > b) oh (c < d)) ;
hindi	resulta = hindi operand1 ;
	ilimbag(hindi resulta) ;

12. Statements

12.a. Declaration statement

Declaration Statement	Syntax	Example
Declaration of variable	numero ID ;	numero age ;
Declaration of variable with assigned value.	numero ID = INTEGER;	numero Even = 2 ;
Declaration of Class	klase ID [<statements> ;]	klase Student [numero age = 9 ; ilimbag(age) ;]
Declaration of Function	pal ID (<parameters>) [<statements> ;]	pal Grade (numero id) [numero age = id + 9 ; ilimbag(age) ;]

12.b. Assignment Statement

Assignment Statement	Syntax	Example
Assigning the value of a variable to a constant	ID = FLOAT ;	money = 500.5;
Assigning the value of a variable to an identifier	ID = ID ;	a = b;
Assigning the value of a variable to an expression	ID = <expr> ;	a = 5+15;

12.c. Conditional Statement

Conditional Statement	Syntax	Example
kung	kung <condition > [<statements>]	kung age>=18[aprubado = 1 ;]
edikung	kung <condition > [<statements>] edikung <condition>[<statements>]	kung age>=18[aprubado = 1 ;] edikung age<=17[aprubado = 0 ;]
edi	kung <condition > [<statements>] edikung <condition>[<statements>] edi [<statements>]	kung age>=18[aprubado = 1 ;] edikung age<=17[aprubado = 0 ;] edi [aprubado = 2 ;]

12.d. Iterative Statements

Iterative Statement	Syntax	Example	Output

ikot	ikot (<assigment_statement>, <condition>)[
	<statements>
	]
		ikot (i=0, i<3) [
		ilimbag(i);
		i = i + 1;
		]
			0 1 2

habang	habang <condition>[
	<statements>
	]
		numero i = 0;
		habang i <= 2[
		i = i + 1;
		ilimbag(“Hello”);
		]
			Hello Hello

12.e. Input/Output Statements

Input/Output Function	Description	Syntax
lagyan	Inputs variables or constants.	ID = lagyan({<io_options>});
ilimbag	Outputs variables or constants.	ilimbag ((<io_options>){, <io_options>});

III. Syntax

Language Context-Free Grammar: G = (T, NT, S, P)

Symbol	Name	Instances
T	Terminal Symbols	klase, ID, pal, numero, lutang, karakter, salita, bul, INTEGER, FLOAT, STRING, BOOLEAN, CHARACTER, kung, edikung, edi, ikot, habang, lagyan, ilimbag, hindi, totoo, mali, at, oh, hindi, +, -, /,//, *, %, ^,>,<,>=,<=,==,!=, (,),[,], ,
NT	Non-Terminal Symbols	program, statements, statement, declaration_statement, assignment_statement, conditional_statement, iterative_statement, io_statement, function_passing, variable_declaration, class_declaration, function_declaration, int_declaration, float_declaration, char_declaration, str_declaration, bool_declaration, parameters, parameter, arguments, argument, if_statement, elif_statements, elif_statement, else_statement, condition, input_statement, output_statement, loop_statement, while_statement, io_options, relational_expression, logical_expressions, logical_expression, expression, constant, expr, term, factor, exp, number, bool, data_type
S	Start Symbol	program
P	Production	Refer to Table - A. Table of Syntax Rule (Production Rules)

A. Table of Syntax Rule (Production Rules)

Non-Terminals	Production Rules
<program>	<program> ::= <statements>
<statements>	<statements> :: = <statement> | <statement> <statements>
<statement>	<statement> ::= <declaration_statement> | <assignment_statement>; | <conditional_statement> | <iterative_statement> | <io_statement> | <function_passing>
<declaration_statement>	<declaration_statement> ::= <variable_declaration> | <class_declaration> | <function_declaration>
<variable_declaration>	<variable_declaration> ::= <int_declaration> | <float_decleration> |

Non-Terminals	**Production Rules**
	<char_declaration> | <str_declaration> | <bool_declaration>
<class_declaration>	<class_declaration> ::= klase ID [(<statements>)]
<function_declaration>	<function_declaration> ::= pal ID (<parameters>) [(<statements>)]
<int_declaration>	<int_declaration> ::= numero ID [= <expr>] ;
<float_decleration>	<float_decleration> ::= lutang ID [= <expr>] ;
<char_declaration>	<char_declaration> ::= karakter ID [= (ID | CHARACTER)];
<str_declaration>	<str_declaration> ::= salita ID [= (ID | STRING];
<bool_declaration>	<bool_declaration> ::= bul ID [= (ID | BOOLEAN)];
<parameters>	<parameters> ::= <parameter> | <parameter>,<parameters>
<parameter>	<parameter> ::= <data_type> ID
<assignment_statement>	<assignment_statement> ::= ID = (<expr> | <constant> | ID)
<function_passing>	<function_passing> ::= ID (<arguments>);
<arguments>	<arguments> ::= <argument> | <argument>, <arguments>
<argument>	<argument> ::= ID
<conditional_statement>	<conditional_statement> ::= <if_statement> | <if_statement> <else_statements> | <if_statement> <elif_statements> | <if_statement> <elif_statements> <else_statement>
<if_statement>	<if_statement> ::= kung <condition>[(<statements>)]
<elif_statements>	<elif_statements> ::= <elif_statement> | <elif_statement> <elif_statements>
<elif_statement>	<elif_statement> ::= edikung <condition>[(<statements>)]
<else_statement>	<else_statement> ::= edi [(<statements>)]
<iterative_statement>	<iterative_statement> ::= <loop_statement> | <while_statement>

Non-Terminals	**Production Rules**
<loop_statement>	<loop_statement> ::= ikot (<assigment_statement>, <condition>) [(<statements>)]
<while_statement>	<while_statement> ::= habang <condition> [(<statements>)]
<io_statement>	<io_statement> ::= <input_statement> | <output_statement>
<input_statement>	<input_statement> ::= ID = lagyan([<io_options>]);
<output_statement>	<output_statement> ::= ilimbag (<io_options>[, <io_options>]) ;
<io_options>	<io_options> ::= ID | <constant> | <expression>
<condition>	<condition> ::= <relational_expression> | <logical_expression> | <logical_expressions>
<expr>	<expr> ::= <term> { (+ | -) <term>}
<term>	<term> ::= <factor> { (* | / | // | %) <factor> }
<factor>	<factor> ::= <exp > | [^ <exp>]
<exp>	<exp> ::= (<expr>) | ID | <number>
<expression>	<expression> ::= <expr> | <relational_expression> | <logical_expressions>
<relational_expression>	<relational_expression> ::= (<number> | ID | <bool>) <relational_operator>(<number> | ID | <bool>)
<logical_expressions>	<logical_expressions> ::= <logical_expression> | <logical_expression> <logical_expressions>
<logical_expression>	<logical_expression> ::= hindi (<bool> | <relational_expression>) | (<bool> | <relational_expression>) <logical_operators> (<bool> | <relational_expression>)
<number>	<number> ::= INTEGER | FLOAT
<data_type>	<data_type> ::= numero | lutang | salita | bul | karakter
<constant>	<constant> ::= INTEGER | FLOAT | STRING | BOOLEAN |

Non-Terminals	**Production Rules**
	CHARACTER
<bool>	<bool> ::= totoo | mali
<operators>	<operators> ::= <arithmetic_operator> | <relational_operator> | <logical_operator>
<arithmetic_operator>	<arithmetic_operator> ::= + | - | * | / | // | ^ | %
<relational_operator>	<relational_operator> ::= < | > | >= | <= | == | !=
<logical_operator>	<logical_operator> ::= at | oh | hindi

B. Derivation and Parse-Tree

B.1 Declaration Statement

B.1.a Variable Declaration

Declaration Statement	**Syntax**	**Example**
Declaration of variable	numero ID;	numero age ;

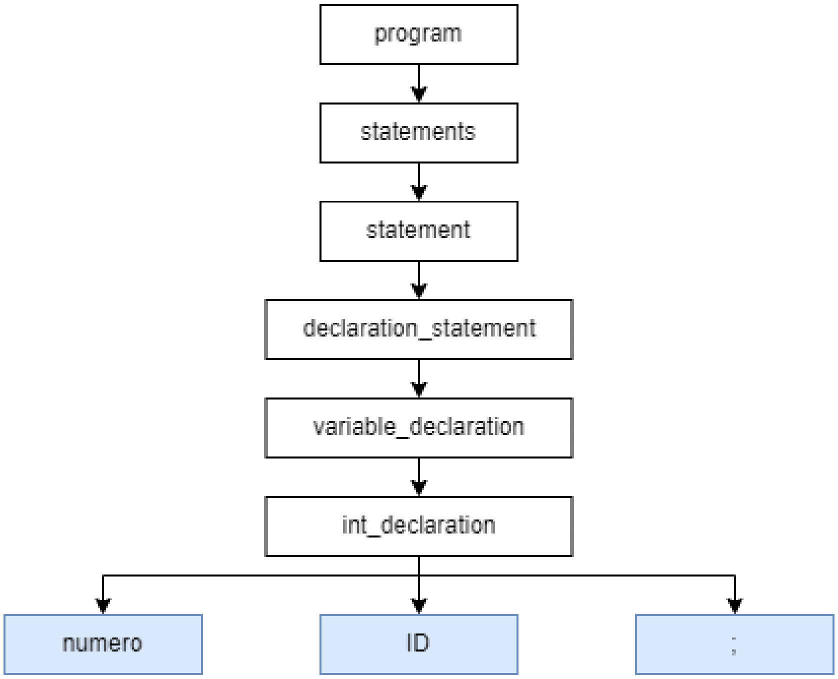
B.1.a.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <declaration_statement>
<declaration_statement> ::= <variable_declaration>
<variable_declaration> ::= <int_declaration>
::= numero ID [= <expr>];
::= numero ID;

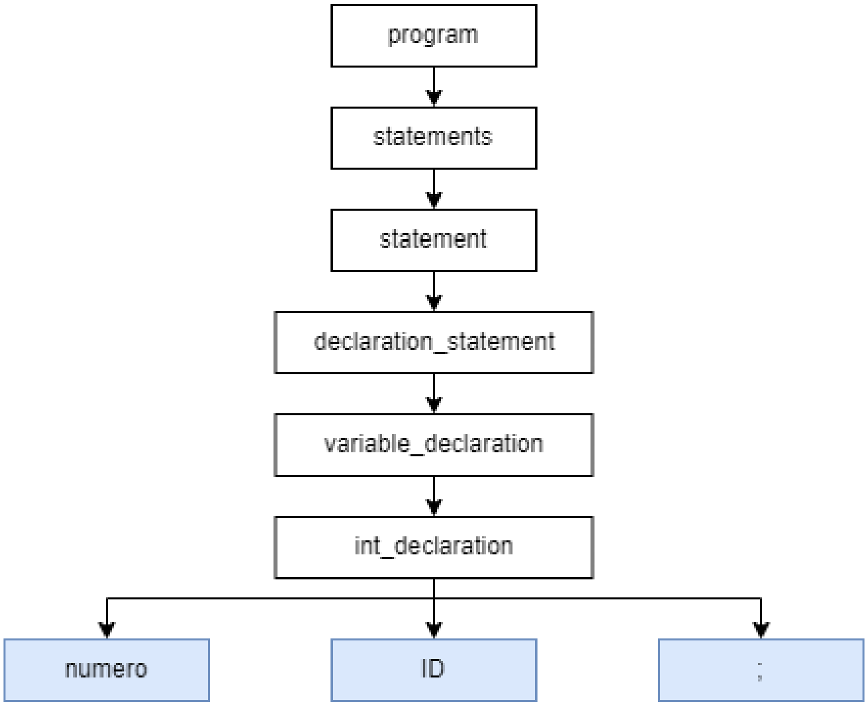
B.1.a 2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <declaration_statement>
<declaration_statement> ::= <variable_declaration>;
<variable_declaration> ::= <int_declaration>;
::= numero ID [= <expr>];
::= numero ID;

B.1.a 3. Top-down Leftmost Parse-Tree



B.1.a 4. Top-down Rightmost Parse-Tree



B.1.b Variable Declaration with Assignment

Declaration Statement	Syntax	Example
Declaration of variable with assigned value.	numero ID = <expr> ;	numero Even = 2;

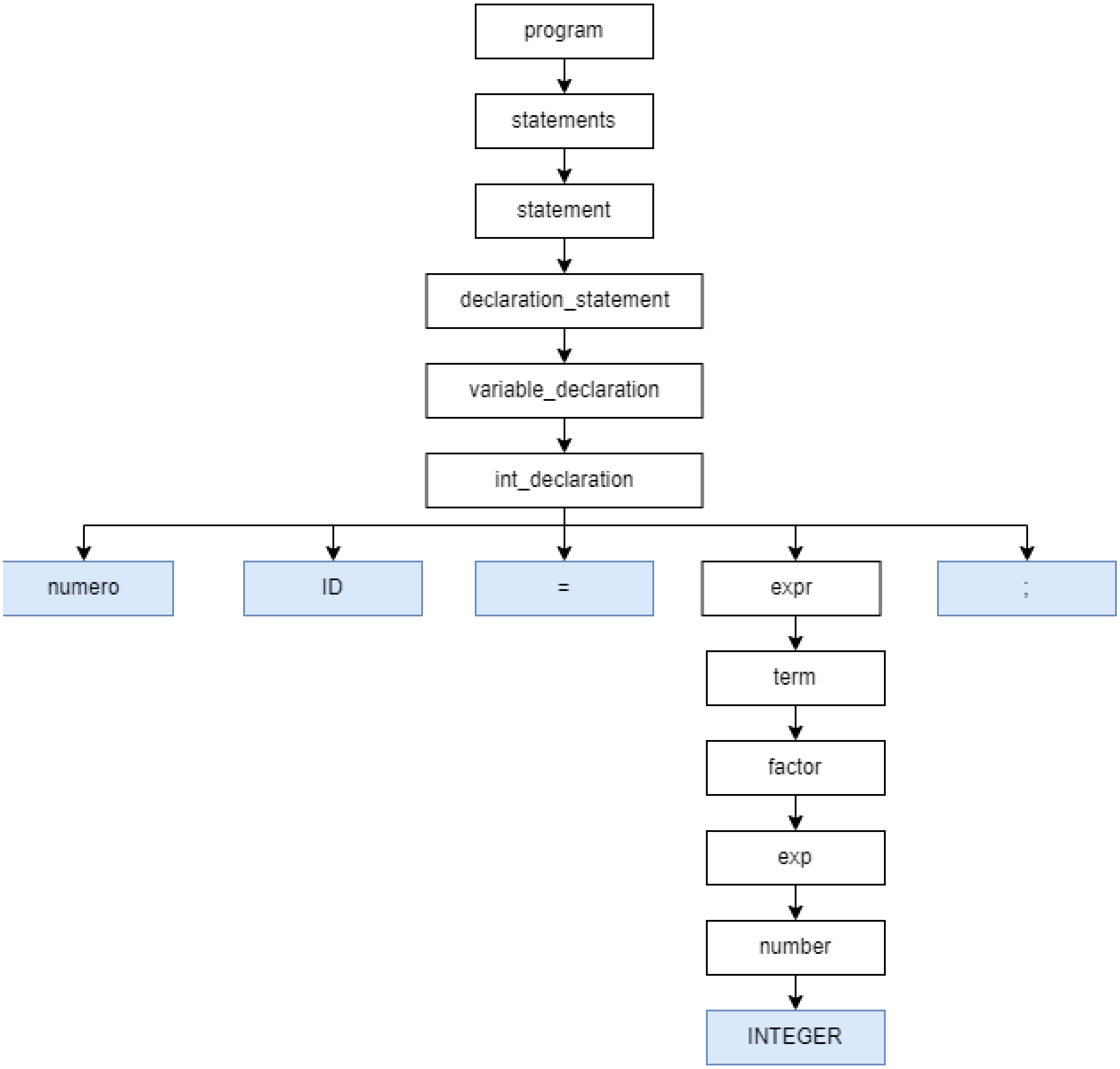
B.1.b.1 Top-down Leftmost Derivation

```
           <program> ::= <statements>
        <statements> ::= <statement>
         <statement> ::= <declaration_statement>
<declaration_statement> ::= <variable_declaration>
 <variable_declaration>  ::= <int_declaration>
                         ::= numero ID[ = <expr>];
                         ::= numero ID = <expr>;
                         ::= numero ID = <term>;
                         ::= numero ID = <factor>;
                         ::= numero ID = <exp>;
                         ::= numero ID = <number>;
                         ::= numero ID =  INTEGER;
```

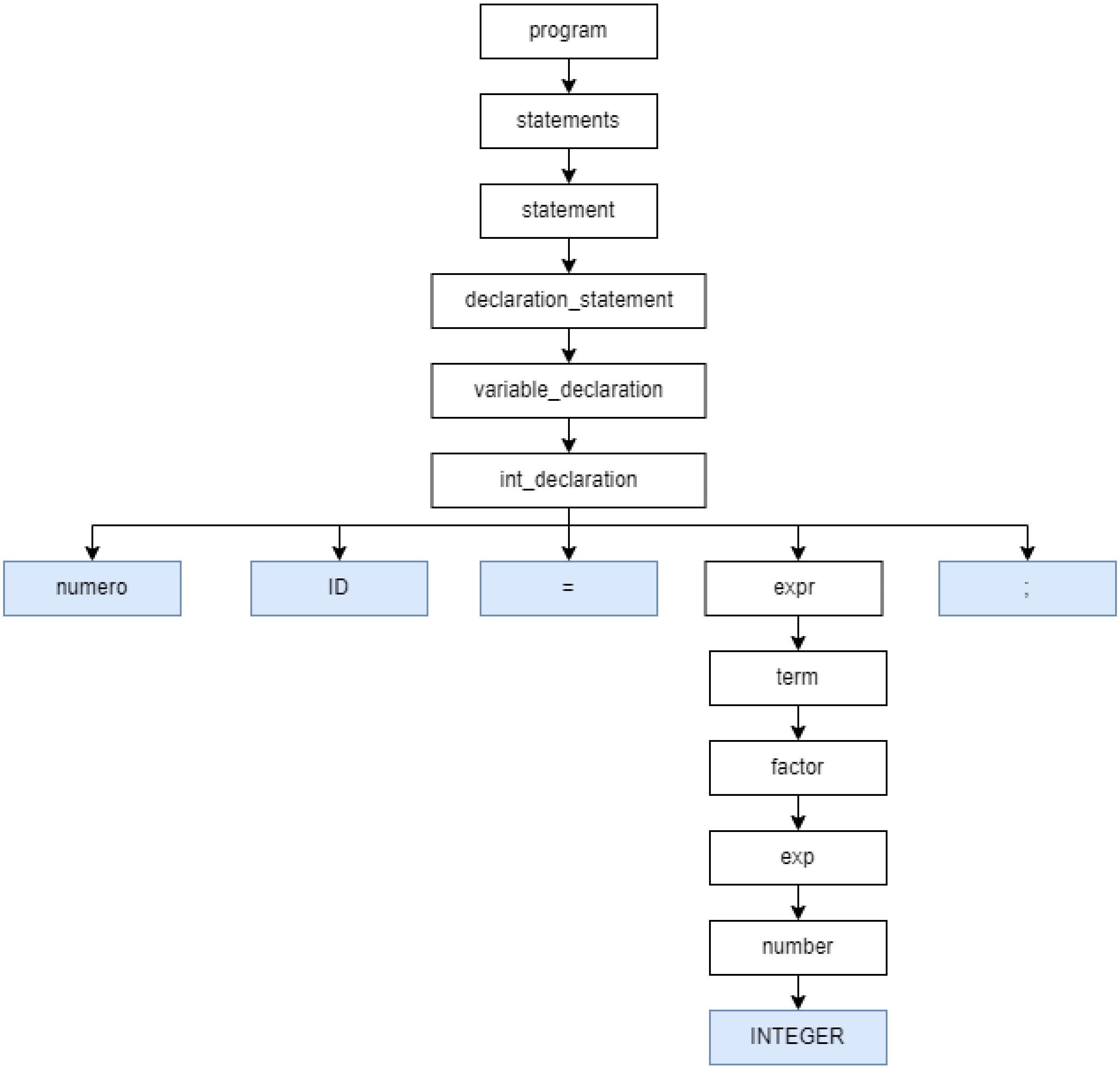
B.1.b.2 Top-down Rightmost Derivation

```
           <program> ::= <statements>
        <statements> ::= <statement>
         <statement> ::= <declaration_statement>
<declaration_statement> ::= <variable_declaration>
  <variable_declaration> ::= <int_declaration>
                         ::= numero ID [ = <expr>];
                         ::= numero ID = <expr>;
                         ::= numero ID = <term>;
                         ::= numero ID = <factor>;
                         ::= numero ID = <exp>;
                         ::= numero ID = <number>;
                         ::= numero ID = INTEGER;
```

B.1.b.3 Top-down Leftmost Parse Tree



B.1.b.4 Top-down Rightmost Parse Tree



B.1.c Function Declaration

Declaration Statement	Syntax	Example
Declaration of function	pal ID (<parameters>)[<statements>]	pal Pulis (numero b)[ilimbag(b+2);]

B.1.c.1 Top-down Leftmost Derivation

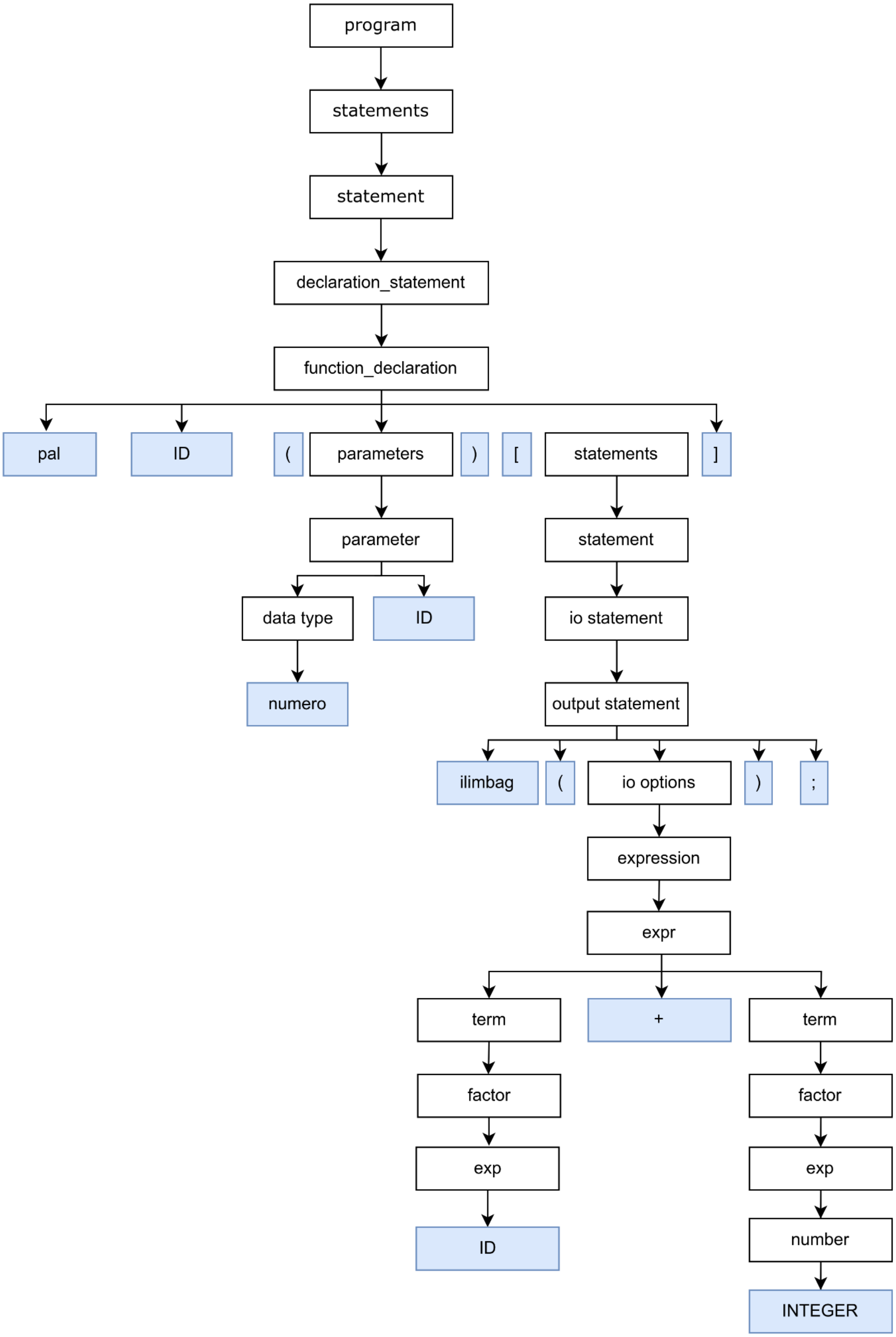
```
            <program> ::= <statements>
           <statements> ::= <statement>
           <statements> ::= <declaration_statement>
            <statement> ::= <function_declaration>
 <function_declaration> ::= pal ID(<parameters>) [<statements>]
                        ::= pal ID(<parameter>) [<statements>]
                        ::= pal ID(<data_type> ID) [<statements>]
                        ::= pal ID(numero ID) [<statements>]
                        ::= pal ID(numero ID) [<statement>]
                        ::= pal ID(numero ID) [<io_statement>]
                        ::= pal ID(numero ID) [<output_statement>]
                        ::= pal ID(numero ID) [ilimbag(<io_options>
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(<expression>
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(<expr>
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(<term>{ (+ | -) <term>}
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag((<factor> { (+ | -) <term>}
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(<exp> { (+ | -) <term>}
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(ID { (+ | -) <term>}
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(ID + <term>
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(ID + <number>
                            [ ,<io_options>]);]
                        ::= pal ID(numero ID) [ilimbag(ID + INTEGER
                            [ ,<io_options>]);]
```

```
                                  ::= pal ID(numero ID) [ilimbag (ID +INTEGER);]
```

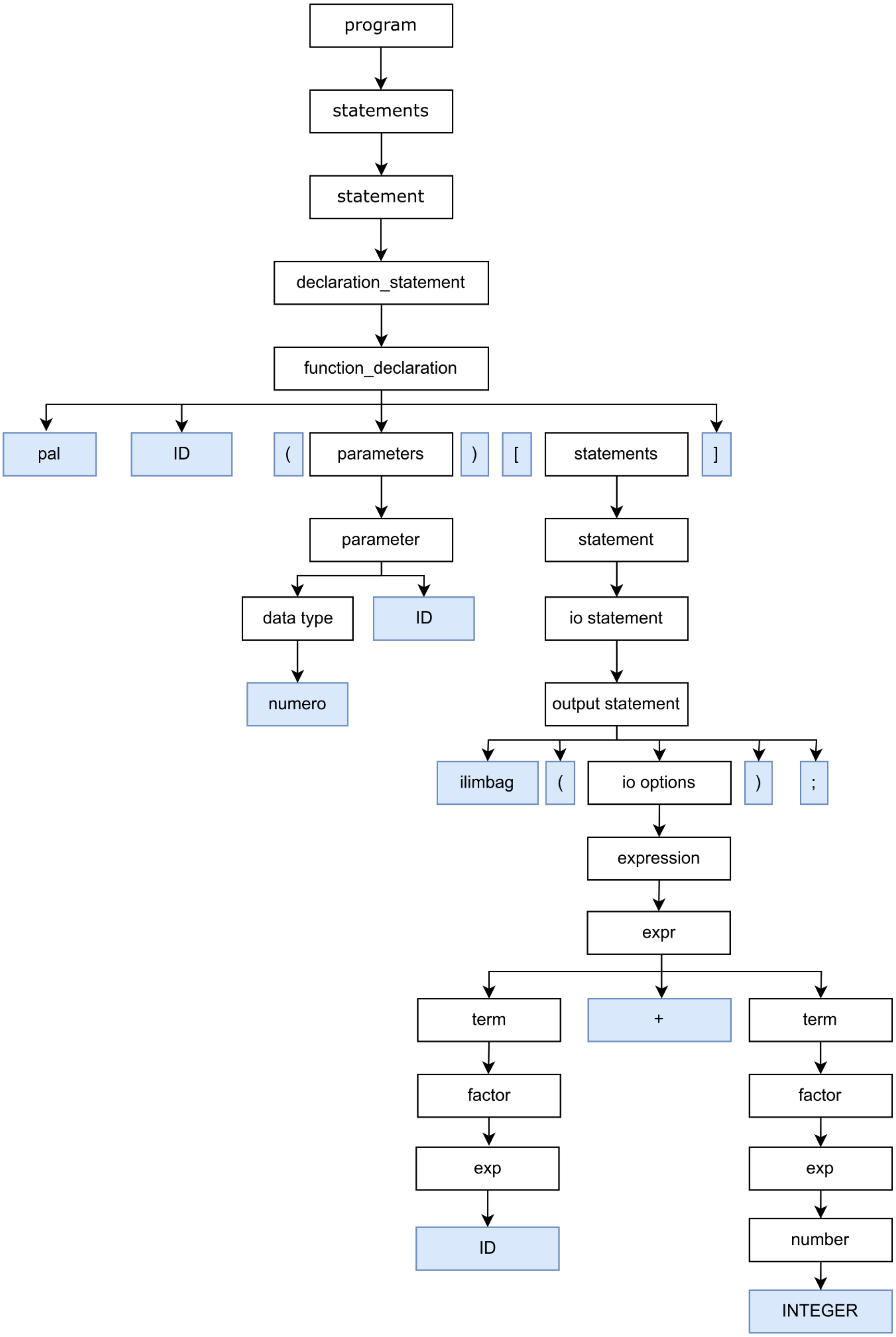
B.1.c.2 Top-down Rightmost Derivation

```
              <program> ::= <statements>
           <statements> ::= <statement>
            <statement> ::= <function_declaration>
<function_declaration> ::= pal ID(<parameters>)[<statements>]
<function_declaration> ::= pal ID(<parameters>)[<statement>]
                        ::= pal ID(<parameters>) [<io_statement>]
                        ::= pal ID(<parameters>) [<output_statement>]
                        ::= pal ID(<parameters>) [ilimbag (<io_options>[ ,
                              <io_options>]);]
                        ::= pal ID(<parameters>) [ilimbag (<io_options>);]
                        ::= pal ID(<parameters>) [ilimbag <expression>);]
                        ::= pal ID(<parameters>) [ilimbag <expr>);]
                        ::= pal ID(<parameters>) [ilimbag(<term>{ (+ | -)
                              <term>});]
                        ::= pal ID(<parameters>) [ilimbag(<term> +  <term>);]
                        ::= pal ID(<parameters>) [ilimbag(<term> +  <factor>);]
                        ::= pal ID(<parameters>) [ilimbag(<term> +  <exp>);]
                        ::= pal ID(<parameters>) [ilimbag(<term> +  <number>);]
                        ::= pal ID(<parameters>) [ilimbag(<term> +  INTEGER);]
                        ::= pal ID(<parameters>) [ilimbag(<factor> +
                             INTEGER);]
                        ::= pal ID(<parameters>) [ilimbag(<exp> +
                             INTEGER);]
                        ::= pal ID(<parameters>) [ilimbag(ID +INTEGER);]
                        ::= pal ID(<parameter>) [ilimbag(ID +INTEGER);]
                        ::= pal ID(<data_type> ID) [ilimbag (ID + INTEGER);]
                        ::= pal ID(numero ID) [ilimbag (ID + INTEGER);]
```

B.1.c.3 Top-down Leftmost Parse Tree



B.1.c.4 Top-down Rightmost Parse Tree



B.1.d Class Declaration

Declaration Statement	Syntax	Example
Declaration of class	klase ID [<statements>]	klase PUP [salita b = “kamusta”; ilimbag(b);]

B.1.d.1 Top-down Leftmost Derivation

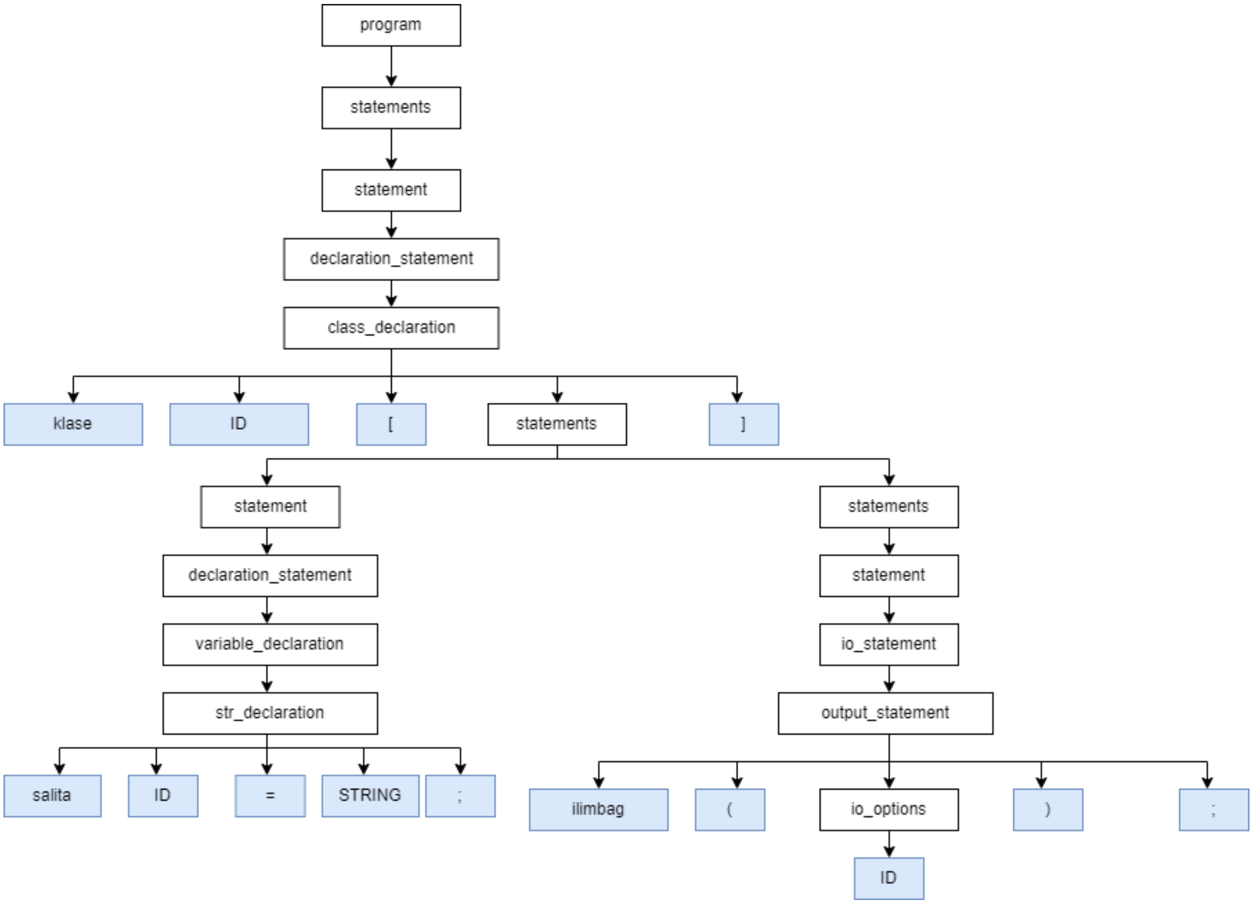
```
             <program> ::= <statements>
            <statements> ::= <statement>
             <statement> ::= <class_declaration>
  <function_declaration> ::= klase ID[[<statements>]]
                         ::= klase ID [<statement><statements>]
                         ::= klase ID [<declaration_statement><statements>]
                         ::= klase ID [<variable_declaration><statements>]]
                         ::= klase ID [<str_declaration> <statements>]
                         ::= klase ID [salita ID [= ( ID | STRING) ]; <statements>]
                         ::= klase ID [salita ID = STRING; <statements>]
                         ::= klase ID [salita ID = STRING; <statement>]
                         ::= klase ID [salita ID = STRING; <io_statement>]
                         ::= klase ID [salita ID = STRING; <output_statement>]
                         ::= klase ID [salita ID = STRING; ilimbag(<io_options>[ ,
                             <io_options>]);]
                         ::= klase ID [salita ID = STRING; ilimbag(ID[ ,
                             <io_options>]); ]
                         ::= klase ID [salita ID = STRING; ilimbag(ID); ]
```

B.1.d.2 Top-down Rightmost Derivation

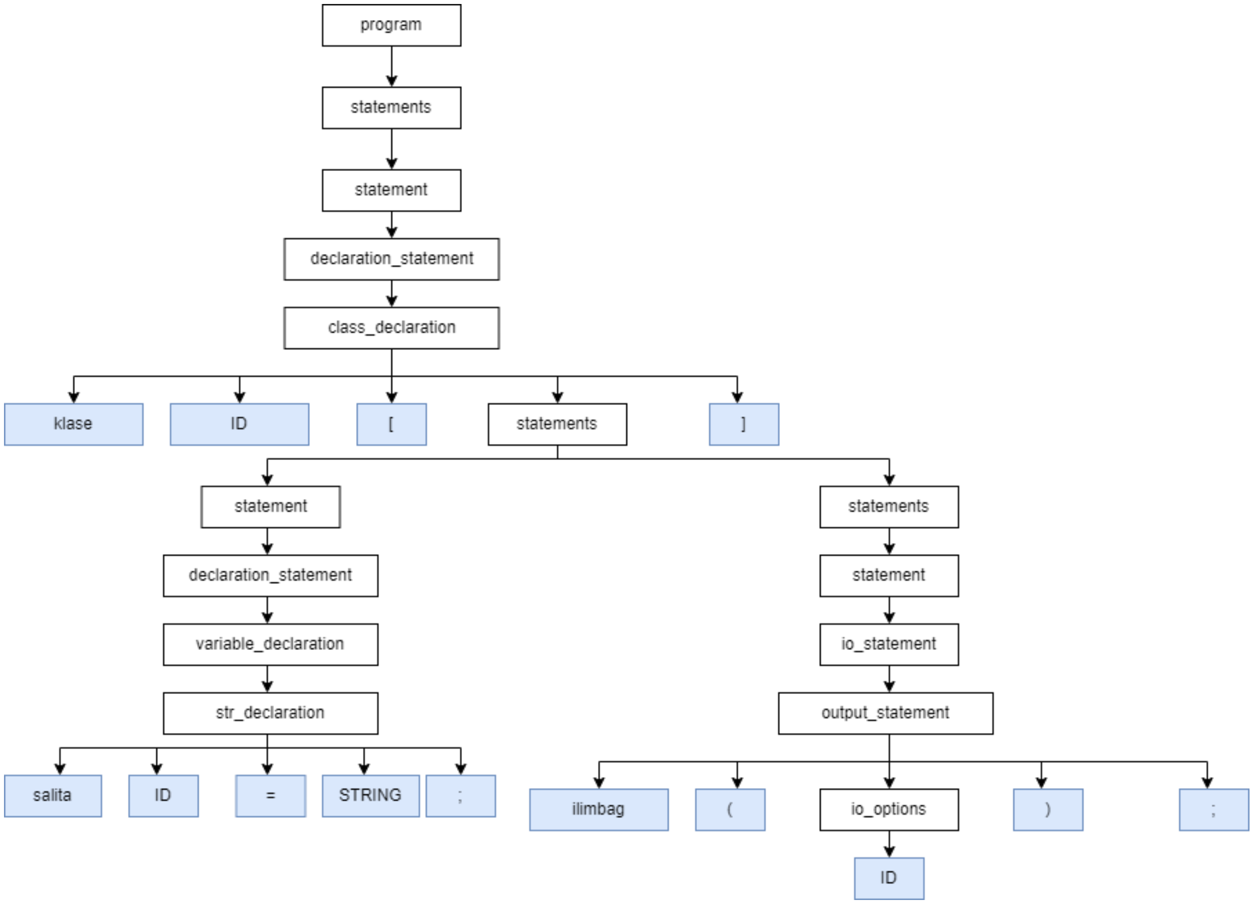
```
             <program> ::= <statements>
            <statements> ::= <statement>
             <statement> ::= <class_declaration>
  <function_declaration> ::= klase ID[<statements>]
                         ::= klase ID [<statement><statements>]
                         ::= klase ID [<statement><statement>]
                         ::= klase ID [<statement><io_statement>]
                         ::= klase ID [<statement> <output_statement>]
```

::= klase ID [<statement> ilimbag(<io_options>[,
<io_options>]);]
::= klase ID [<statement> ilimbag(<io_options>);]
::= klase ID [<statement> ilimbag(ID);]
::= klase ID [<declaration_statement>ilimbag(ID);]
::= klase ID [<variable_declaration>ilimbag(ID);]
::= klase ID [<str_declaration>ilimbag(ID);]
::= klase ID [salita ID [=(ID | STRING)]; ilimbag(ID);]
::= klase ID [salita ID = STRING; ilimbag(ID);]

B.1.d.3 Top-down Leftmost Parse Tree



B.1.d.4 Top-down Rightmost Parse Tree



B.2 Assignment Statement

B.2.a Variable Assignment using Constant

Assignment Statement	Syntax	Example
Assigning the value of a variable to a constant	ID = FLOAT ;	money = 500.5;

B.2.a.1 Top-down Leftmost Derivation

<program> ::= <statements>

<statements> ::= <statement>

<statement> ::= <assignment_statement>;

<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);

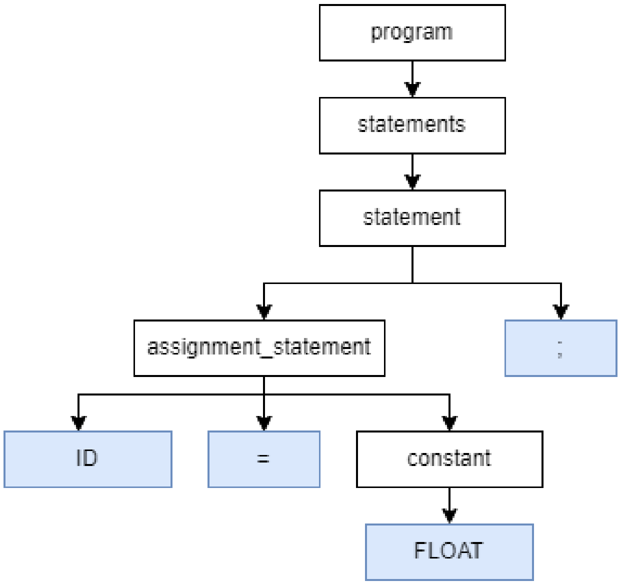
<assignment_statement> ::= ID = <constant>;

::= ID = FLOAT;

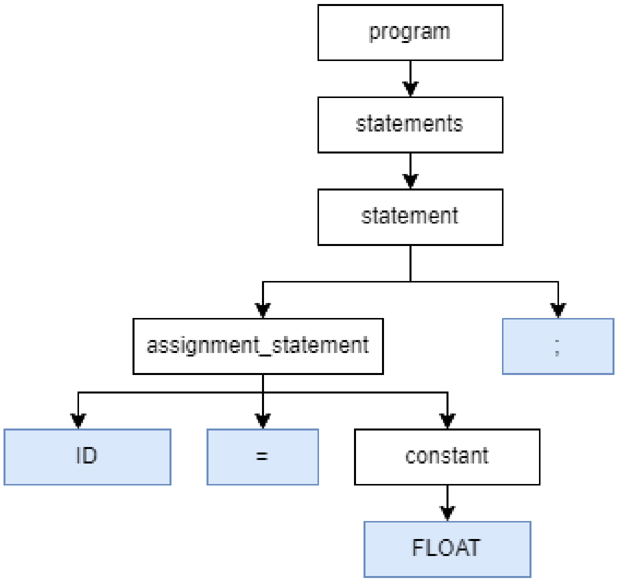
B.2.a.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <assignment_statement>;
<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);
<assignment_statement> ::= ID = <constant>;
::= ID = FLOAT;

B.2.a.3 Top-down Leftmost Parse-Tree



B.2.a.4 Top-down Rightmost Parse-Tree



B.2.b Variable Assignment using Identifier

Assignment Statement	Syntax	Example
Assigning the value of a variable to an identifier	ID = ID	a = b;

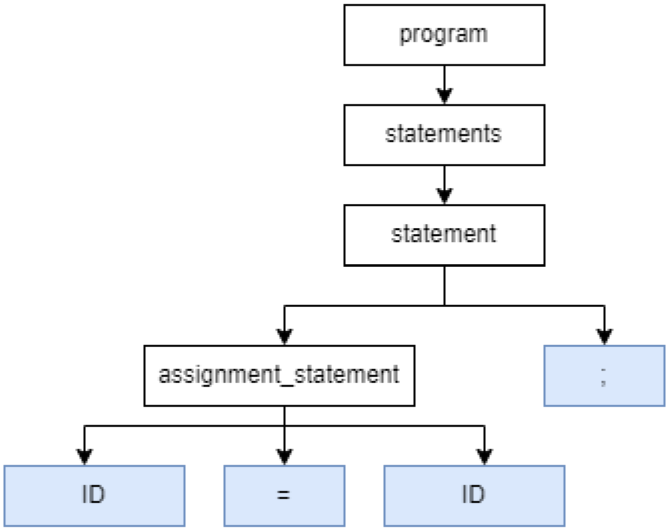
B.2.b.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <assignment_statement>;
<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);
<assignment_statement>::= ID = ID;

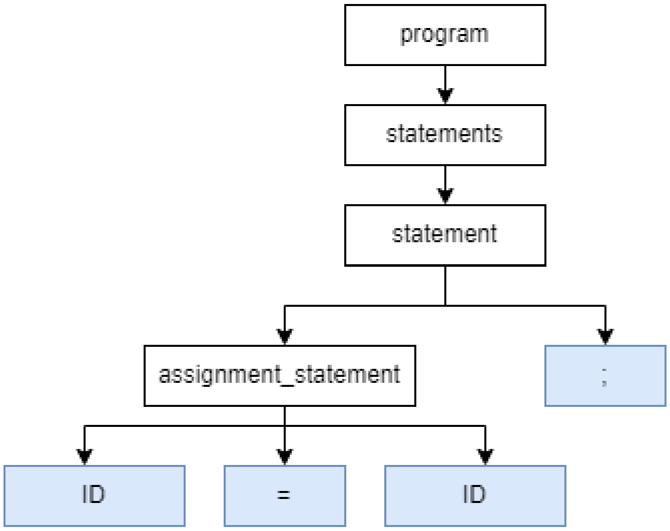
B.2.b.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <assignment_statement>;
<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);
<assignment_statement> ::= ID = ID;

B.2.b.3 Top-down Leftmost Parse Tree



B.2.b.4 Top-down Rightmost Parse Tree



B.2.c Variable Assignment using Expression

Assignment Statement	Syntax	Example
Assigning the value of a variable to an expression	ID = <expr> ;	a = 5+15;

B.2.c.1 Top-down Leftmost Derivation

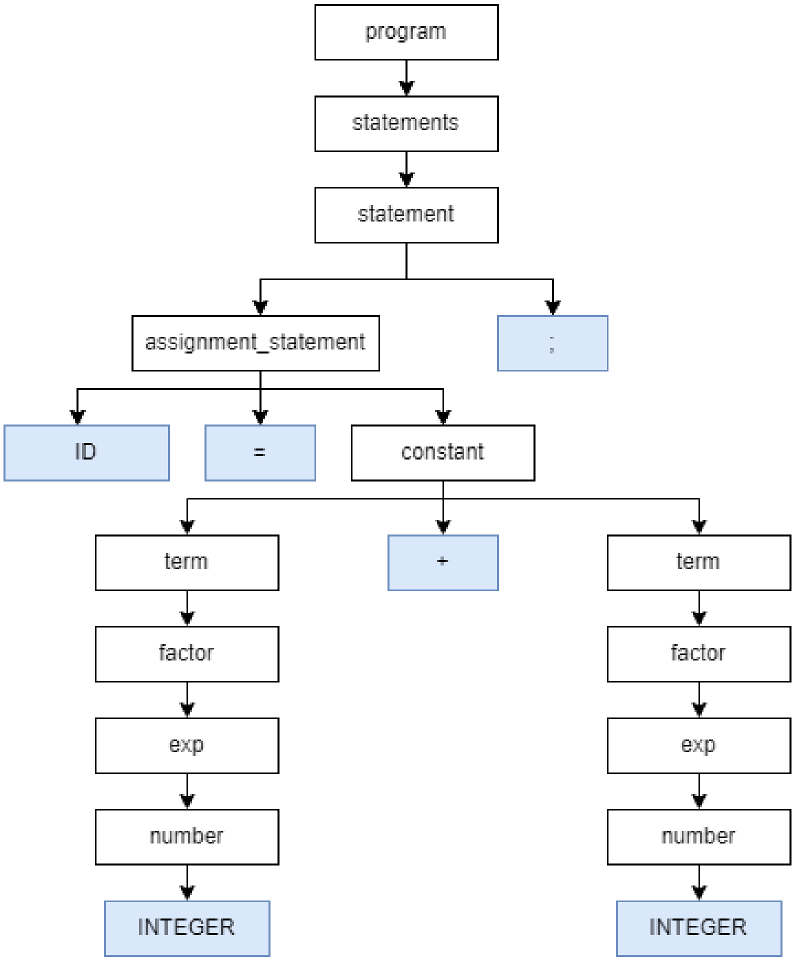
<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <assignment_statement>;
<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);
<assignment_statement> ::= ID = <expr>;
::= ID = <term> { (+|-) <term>};
::= ID = <factor> { (+|-) <term>};
::= ID = <exp> { (+|-) <term>};
::= ID = <number>{ (+|-) <term>};
::= ID = INTEGER { (+|-) <term>};
::= ID = INTEGER + <term>;
::= ID = INTEGER + <factor>;
::= ID = INTEGER + <exp>;
::= ID = INTEGER + <number>;

::= ID = INTEGER + INTEGER;

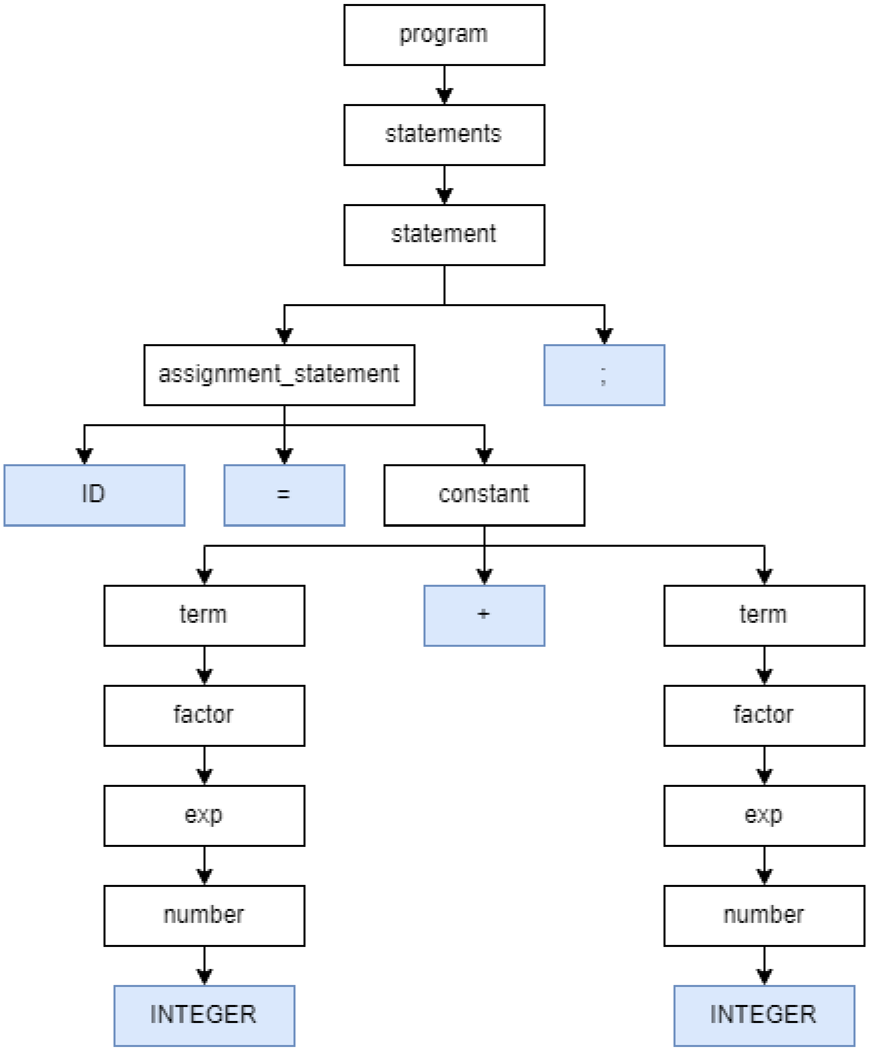
B.2.c.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <assignment_statement>;
<assignment_statement> ::= ID = (<expr> | <constant> | <ID>);
<assignment_statement> ::= ID = <expr>;
::= ID = <term> { (+|-) <term>};
::= ID = <term> + <term>;
::= ID = <term> + <factor>;
::= ID = <term> + <exp>;
::= ID = <term> + <number>;
::= ID = <term> + INTEGER;
::= ID = <factor> + INTEGER;
::= ID = <exp> + INTEGER;
::= ID = <number> + INTEGER;
::= ID = INTEGER + INTEGER;

B.2.c.3 Top-down Leftmost Parse Tree



B.2.c.4 Top-down Rightmost Parse Tree



B.3 Conditional Statement

B.3.a If Statement

Conditional Statement	Syntax	Example
If statement	kung<condition> [(<statements>)]	kung age>=18 [aprubado = 1 ;]

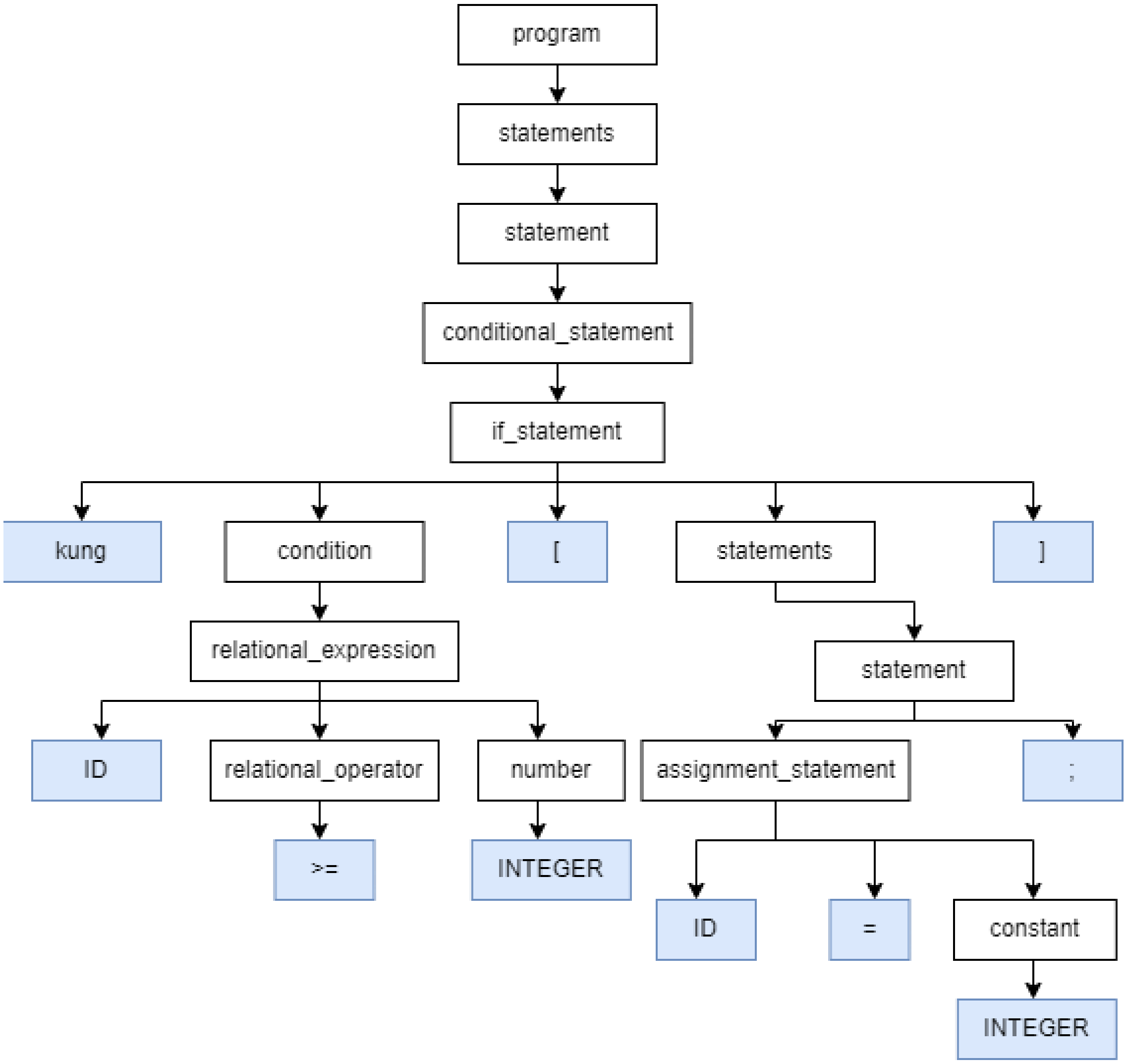
B.3.a.1 Top-down Leftmost Derivation

```
              <program> ::= <statements>
           <statements> ::= <statement>
            <statement> ::= <conditional_statement>
<conditional_statement> ::= <if_statement>
         <if_statement> ::= kung <condition> [<statements>]
                        ::= kung <relational_expression> [<statements>]
                        ::= kung (<number> | ID | <bool>) <relational_operator>
                              (<number> | ID | <bool> )  [<statements>]
                        ::= kung  ID <relational_operator> (<number> | ID | <bool> )
                              [<statements>]
                        ::= kung ID >= (<number> | ID | <bool> ) [<statements>]
                        ::= kung ID >= <number> [<statements>]
                        ::= kung ID >= INTEGER [<statements>]
                        ::= kung ID >= INTEGER [<statement>]
                        ::= kung ID >= INTEGER [<assignment_statement>;]
                        ::= kung ID >= INTEGER [ID = ( <expr> | <constant> | ID);]
                        ::= kung ID >= INTEGER [ID = <constant> ;]
                        ::= kung ID >= INTEGER [ID = INTEGER ;]
```

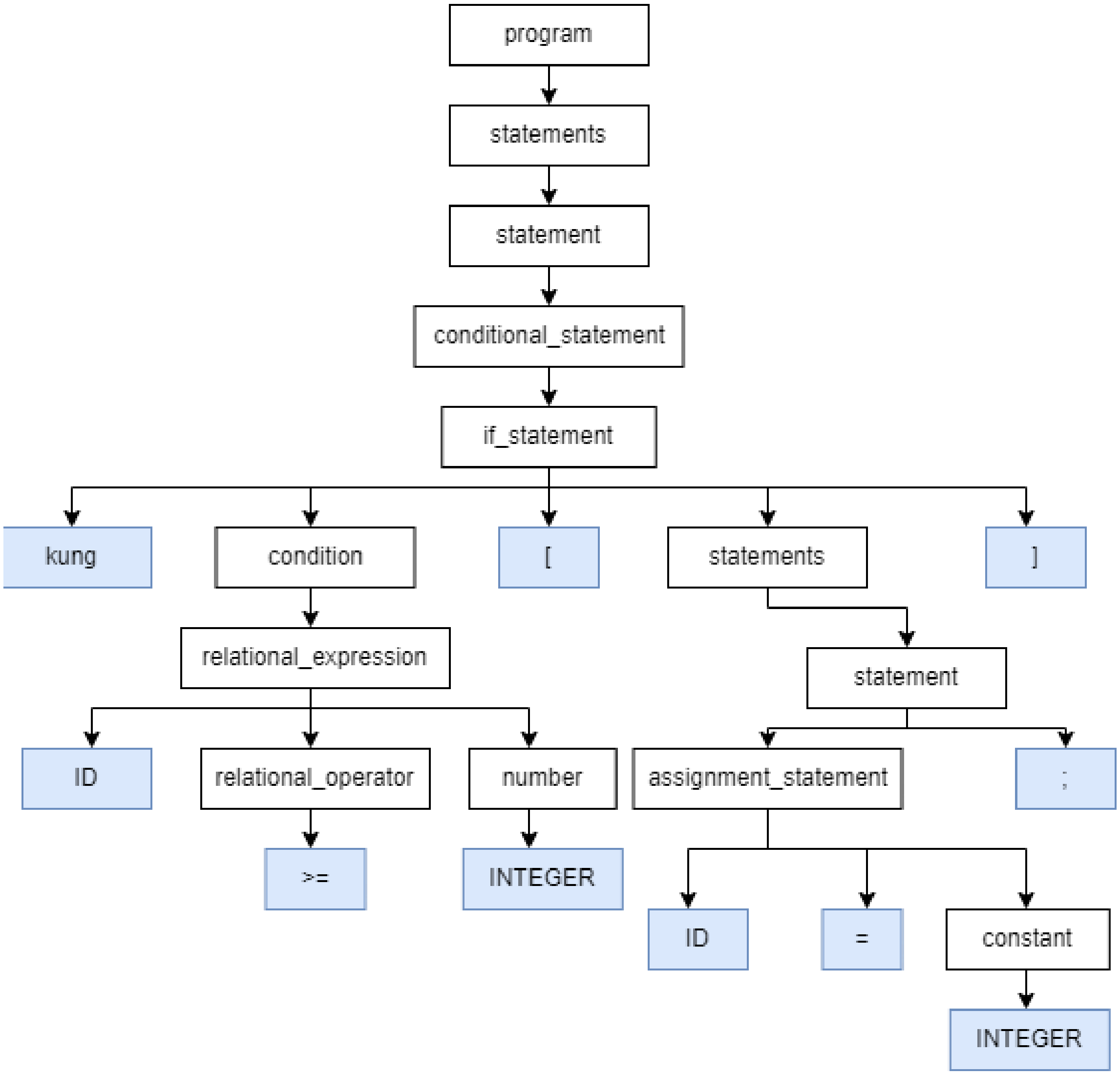
B.3.a.2 Top-down Rightmost Derivation

```
              <program> ::= <statements>
           <statements> ::= <statement>
            <statement> ::= <conditional_statement>
<conditional_statement> ::= <if_statement>
         <if_statement> ::= kung <condition> [<statements>]
                        ::= kung <condition> [<statement>]
                        ::= kung <condition> [<assignment_statement>;]
                        ::= kung <condition> [ID = ( <expr> | <constant> | ID);]
                        ::= kung <condition> [ID = <constant>;]
                        ::= kung <condition>[ID = INTEGER ;]
                        ::= kung <relational_expression> [ID = INTEGER ;]
                        ::= kung (<number> | ID | <bool>) <relational_operator>
                              (<number> | ID | <bool>) [ID = INTEGER ;]
                        ::= kung (<number> | ID | <bool>) <relational_operator> <number>
                            [ID = INTEGER ;]
                        ::= kung (<number> | ID | <bool>) <relational_operator>INTEGER
                            [ID = INTEGER ;]
                        ::= kung (<number> | ID | <bool>) >= INTEGER [ID = INTEGER ;]
                        ::= kung ID >= INTEGER  [ID = INTEGER ;]
```

B.3.a.3 Top-down Leftmost Parse Tree



B.3.a.4 Top-down Rightmost Parse Tree



B.3.b If Elif Statements

Conditional Statement	Syntax	Example
If statement and Elif statement	kung<condition> [(<statements>)] edikung <condition>[(<statements>)]	kung age>=18 [aprubado = 1 ;] edikung age<=17[aprubado = 0 ;]

B.3.b.1 Top-down Leftmost Derivation

```
             <program> ::= <statements>
          <statements> ::= <statement>
           <statement> ::= <conditonal_statement>
<conditonal_statement> ::= <if_statement> <elif_statements>
                       ::= kung <condition> [(<statements>)] <elif_statements>
                       ::= kung <relational_expression> [(<statements>)]<elif_statements>
                       ::= kung (<number> | ID | <bool>)  <relational_operator>
                             (<number> | ID | <bool> ) [(<statements>)] <elif_statements>
                       ::= kung ID <relational_operator>(<number> | ID |  <bool> )
                             [(<statements>)] <elif_statements>
                       ::= kung ID >= (<number> | ID | <bool>)[(<statements>)]
                             <elif_statements>
                       ::= kung ID >= <number> [(<statements>)] <elif_statements>
                       ::= kung ID >= INTEGER [(<statements>)] <elif_statements>
                       ::= kung ID >= INTEGER [(<statement>)] <elif_statements>
                       ::= kung ID >= INTEGER  [(<assignment_statement>;)]
                             <elif_statements>
                       ::= kung ID >= INTEGER [ID = ( <expr> | <constant> | ID);]
                             <elif_statements>
                       ::= kung ID >= INTEGER [ID = <constant>;] <elif_statements>
                       ::= kung ID >= INTEGER [ID = INTEGER;] <elif_statements>
                       ::= kung ID >= INTEGER [ID = INTEGER;] <elif_statement>
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung <condition>
                             [(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung
                             <relational_expression>[(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung
                             (<number> | ID | <bool>) <relational_operator>
                             (<number> | ID | <bool> ) [(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID
                             <relational_operator> (<number> | ID | <bool> ) [(<statements>)]
                       ::= kung ID >= INTEGER [ID =INTEGER;] edikung ID <=
                             (<number> | ID | <bool> ) [(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= <number>
                             [(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                             [(<statements>)]
                       ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                             [(<statement>)]
```

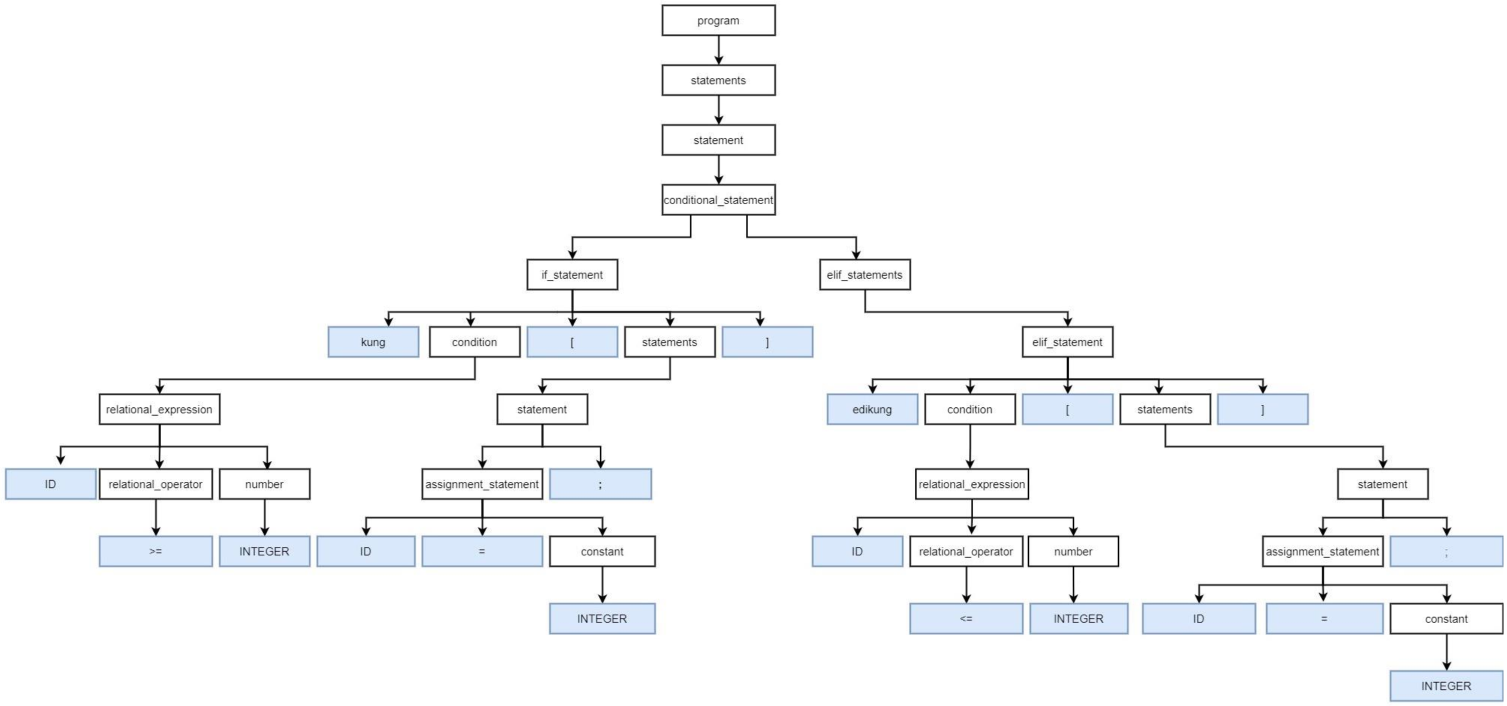
```
                      ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                             [(<assignment_statement>;)]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                            [ID = ( <expr> | <constant> | ID);]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                             [ID = constant;]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edikung ID <= INTEGER
                            [ID = INTEGER;]
```

B.3.b.2 Top-down Rightmost Derivation

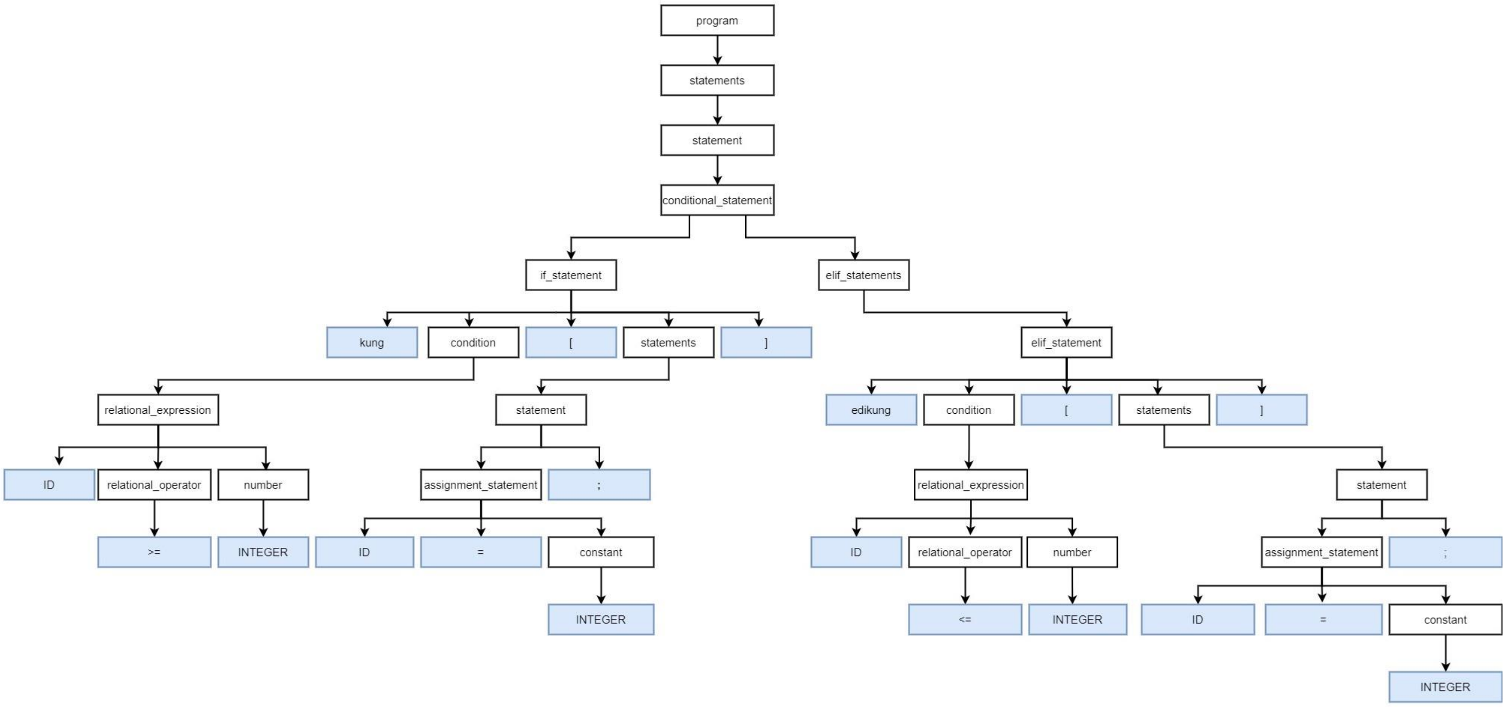
```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <conditonal_statement>
<conditonal_statement> ::= <if_statement> <elif_statements>
                      ::= <if_statement> <elif_statement>
                      ::= <if_statement> edikung <condition>[(<statements>)]
                      ::= <if_statement> edikung <condition>[(<statement>)]
                      ::= <if_statement> edikung <condition>[(<assignment_statement>;)]
                      ::= <if_statement> edikung <condition>
                            [ID = ( <expr> | <constant> | ID);]
                      ::= <if_statement> edikung <condition>[ID = <constant>;]
                      ::= <if_statement> edikung <condition>[ID = INTEGER;]
                      ::= <if_statement> edikung <relational_expression> [ID = INTEGER;]
                      ::= <if_statement> edikung (<number> | ID | <bool>)
                            <relational_operator> (<number> | ID | <bool> )
                            [ID = INTEGER;]
                      ::= <if_statement> edikung (<number> | ID | <bool>)
                            <relational_operator> <number> [ID = INTEGER;]
                      ::= <if_statement> edikung (<number> | ID | <bool>)
                            <relational_operator> INTEGER [ID = INTEGER;]
                      ::= <if_statement> edikung (<number> | ID | <bool>) <= INTEGER
                            [ID = INTEGER;]
                      ::= <if_statement> edikung ID <=  INTEGER [ID = INTEGER;]
                      ::= kung <condition> [(<statements>)] edikung ID <= INTEGER
                            [ID = INTEGER;]
                      ::= kung <condition> [(<statement>)] edikung ID <= INTEGER
                            [ID = INTEGER;]
                      ::= kung <condition> [(<assignment_statement>;)]edikung
                            ID <= INTEGER [ID = INTEGER;]
                      ::= kung <condition> [ID = ( <expr> | <constant> | ID);] edikung
                            ID <= INTEGER [ID = INTEGER;]
```

::= kung <condition> [ID = <constant>;] edikung ID <= INTEGER
[ID = INTEGER;]
::= kung <condition> [ID = INTEGER;] edikung ID <= INTEGER
[ID = INTEGER;]
::= kung <relational_expression> [ID = INTEGER;] edikung
ID <= INTEGER [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator>
(<number> | ID | <bool>) [ID = INTEGER;] edikung
ID <= INTEGER [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator> <number>
[ID = INTEGER;] edikung ID <= INTEGER [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator> INTEGER
[ID = INTEGER;] edikung ID <= INTEGER [ID = INTEGER;]
::= kung (<number> | ID | <bool>) >= INTEGER [ID = INTEGER;]
edikung ID <= INTEGER [ID = INTEGER;]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
ID <= INTEGER [ID = INTEGER;]

B.3.b.3 Top-down Leftmost Parse Tree



B.3.b.4 Top-down Rightmost Parse Tree



B.3.c If else Statement

Conditional Statement	Syntax	Example
If statement and Else statement	kung<condition> [(<statements>)] edi <condition>[(<statements>)]	kung age>=18[aprubado = 1 ;] edi [aprubado = 2 ;]

B.3.c.1 Top-down Leftmost Derivation

<program> ::= <statements>

<statements> ::= <statement>

<statement> ::= <conditonal_statement>

<conditonal_statement> ::= <if_statement> <else_statement>

::= kung <condition>[(<statements>)] <else_statement>

::= kung <relational_expression> [(<statements>)] <else_statement>

::= kung (<number> | ID | <bool>) <relational_operator>
(<number> | ID | <bool>) [(<statements>)] <else_statement>

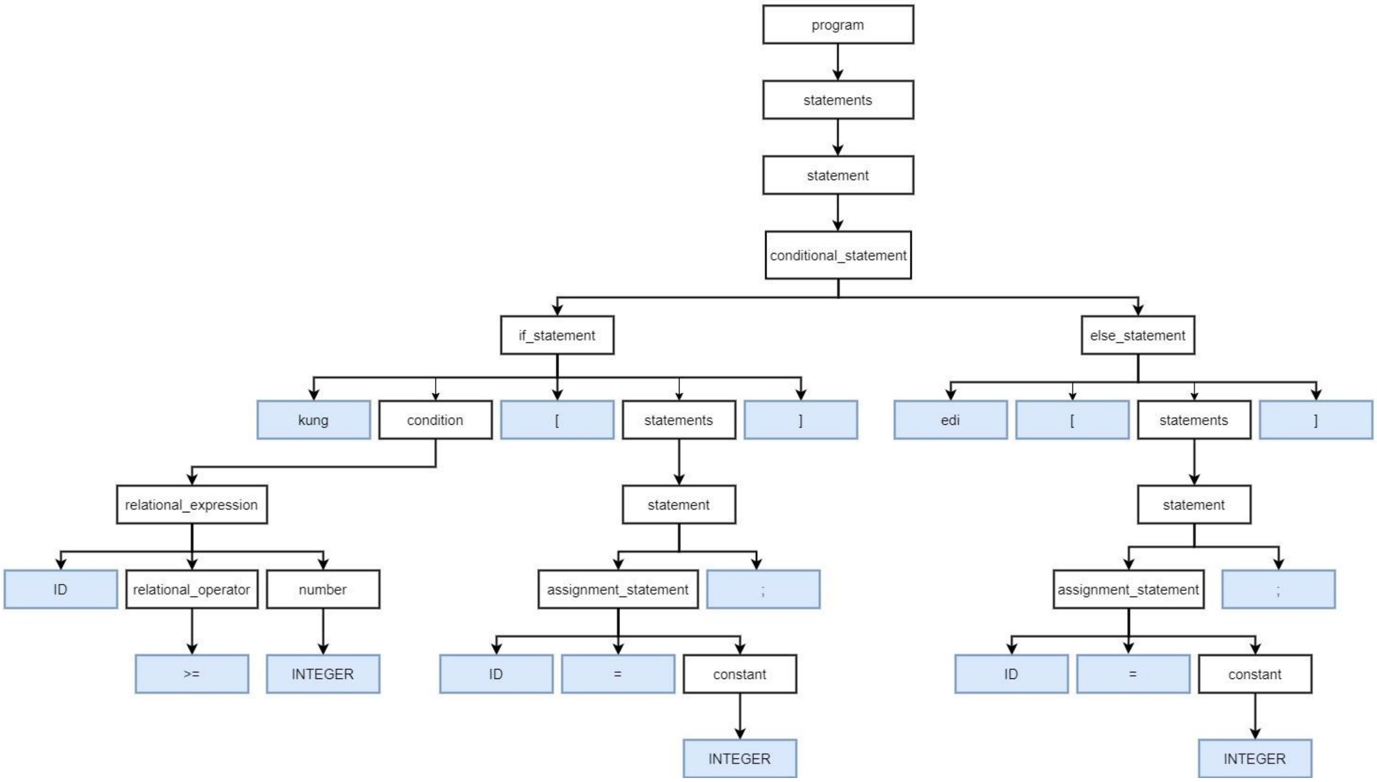
```
                      ::= kung ID <relational_operator> (<number> | ID | <bool>)
                              [(<statements>)] <else_statement>
                      ::= kung ID >= (<number> | ID | <bool>) [(<statements>)]
                              <else_statement>
                      ::= kung ID >= <number> [(<statements>)] <else_statement>
                      ::= kung ID >= INTEGER [(<statements>)] <else_statement>
                      ::= kung ID >= INTEGER [(<statement>)] <else_statement>
                      ::= kung ID >= INTEGER [(<assignment_statement>;)]
                              <else_statement>
                      ::= kung ID >= INTEGER [ID = (<expr> | <constant>| ID);]
                              <else_statement>
                      ::= kung ID >= INTEGER [ID = <constant>;] <else_statement>
                      ::= kung ID >= INTEGER [ID = INTEGER;] <else_statement>
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi [(<statements>)]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi [(<statement>)]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi
                              [(<assignment_statement>;)]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi
                              [ID = ( <expr> | <constant> | ID);]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi [ID = <constant>;]
                      ::= kung ID >= INTEGER [ID = INTEGER;] edi [ID = INTEGER;]
```

B.3.c.2 Top-down Rightmost Derivation

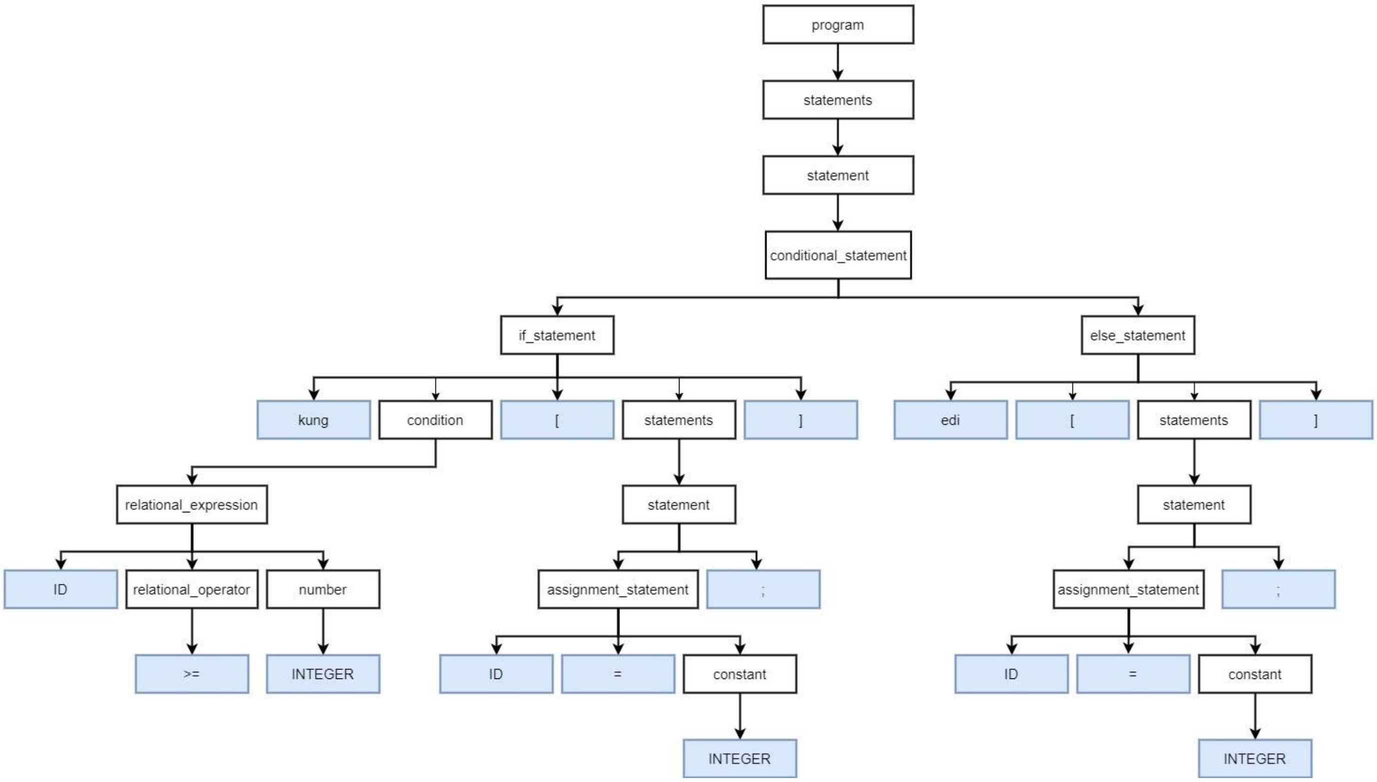
```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <conditonal_statement>
<conditonal_statement> ::= <if_statement> <else_statement>
                      ::= <if_statement> edi [(<statements>)]
                      ::= <if_statement> edi [(<statement>)]
                      ::= <if_statement> edi [(<assignment_statement>; )]
                      ::= <if_statement> edi [ID = (<expr> | <constant> | ID);]
                      ::= <if_statement> edi [ID = <constant>;]
                      ::= <if_statement> edi [ID = INTEGER;]
                      ::= kung <condition> [(<statements>)] edi [ID =INTEGER;]
                      ::= kung <condition> [(<statement>)] edi [ID = INTEGER;]
                      ::= kung <condition> [(<assignment_statement>;)] edi
                              [ID = INTEGER;]
                      ::= kung <condition>[ID = (<expr> | <constant> | ID);] edi
                              [ID = INTEGER;]
                      ::= kung <condition>[ID = <constant>;] edi [ID = INTEGER;]
```

::= kung <condition>[ID = INTEGER;] edi [ID = INTEGER;]
::= kung <relational_expression> [ID = INTEGER;] edi
    [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator>
    (<number> | ID | <bool>) [ID = INTEGER;] edi
    [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator> <number>
    [ID = INTEGER;] edi [ID = INTEGER;]
::= kung (<number> | ID | <bool>) <relational_operator> INTEGER
    [ID = INTEGER;] edi [ID = INTEGER;]
::= kung (<number> | ID | <bool>) >= INTEGER [ID = INTEGER;]
    edi [ID = INTEGER;]
::= kung ID >= INTEGER [ID = INTEGER;] edi [ID = INTEGER;]

B.3.c.3 Top-down Leftmost Parse Tree



B.3.c.4 Top-down Rightmost Parse Tree



B.3.d If elif else Statements

Conditional Statement	Syntax	Example
If statement, Elif statement, and Else statement	kung<condition> [(<statements>)] edikung <condition>[(<statements>)] edi [(<statements>)]	kung age>=18 [aprubado = 1 ;] edikung age<=17[aprubado = 0 ;] edi [aprubado = 2 ;]

B.3.d.1 Top-down Leftmost Derivation

```
                <program> ::= <statements>
             <statements> ::= <statement>
              <statement> ::= <conditional_statement>
  <conditional_statement> ::= <if_statement> <elif_statements> <else_statement>
                          ::= kung <condition> [(<statements>)] <elif_statements>
                                <else_statement>
                          ::= kung <relational_expression> [(<statements>)] <elif_statements>
                                <else_statement>
                          ::= kung (<number> | ID | <bool>) <relational_operator>
                                (<number> | ID | <bool>) [(<statements>) ] <elif_statements>
                                <else_statement>
                          ::= kung ID <relational_operator> (<number> | ID | <bool>)
                                [(<statements>) ] <elif_statements> <else_statement>
                          ::= kung ID >= (<number> | ID | <bool> ) [(<statements>)]
                                <elif_statements> <else_statement>
                          ::= kung ID >= <number> [(<statements>)] <elif_statements>
                                <else_statement>
                          ::= kung ID >= INTEGER [(<statements>)]<elif_statements>
                                <else_statement>
                          ::= kung ID >= INTEGER [<statement>] <elif_statements>
                                <else_statement>
                          ::= kung ID >= INTEGER [<assignment_statement>;]
                                <elif_statements> <else_statement>
                          ::= kung ID >= INTEGER [ID = (<expr> | <constant> | ID);]
                                <elif_statements> <else_statement>
                          ::= kung ID >= INTEGER [ID = <constant>;] <elif_statements>
                                <else_statement>
                          ::= kung ID >= INTEGER [ID = INTEGER;] <elif_statements>
                                <else_statement>
                          ::= kung ID >= INTEGER [ID = INTEGER;] <elif_statement>
                                <else_statement>
                          ::= kung ID >= INTEGER [ID = INTEGER;] edikung
                                <condition> [(<statements>)] <else_statement>
                          ::= kung ID >= INTEGER [ID = INTEGER;] edikung
                                <relational_expression>[(<statements>)] <else_statement>
                          ::= kung ID >= INTEGER [ID = INTEGER;] edikung
                                (<number> | ID | <bool>) <relational_operator>
                                (<number> | ID | <bool>)[(<statements>)] <else_statement>
```

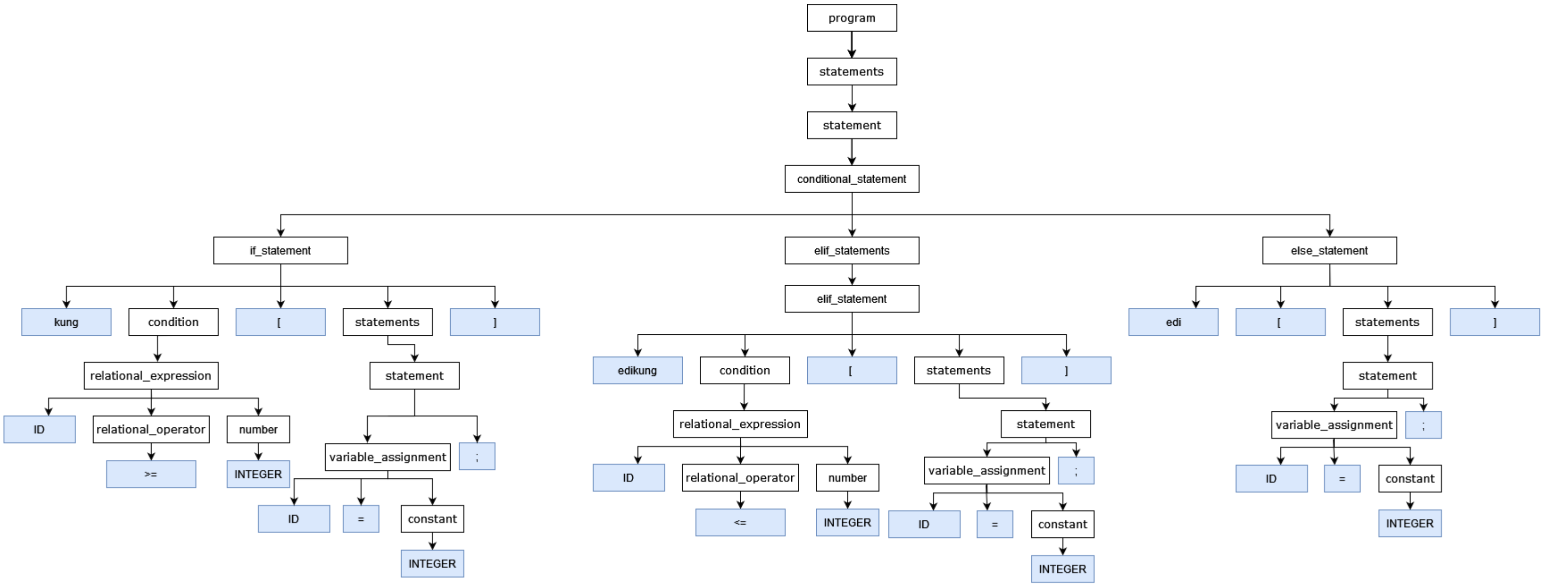
```
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <relational_operator> (<number> | ID | <bool>)
      [(<statements>)] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= (<number> | ID | <bool>) [(<statements>)]
      <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= <number> [(<statements>)] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [(<statements>)] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <=  INTEGER [<statement>] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <=  INTEGER [<asssignment_statement>;] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = (<expr> | <constant> | ID);]
      <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = <constant>;] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] <else_statement>
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi [(<statements>)]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi [<statement>]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi[<assignment_statement>;]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi
      [ID = ( <expr> | <constant> | ID);]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi [ID = <constant>;]
::= kung ID >= INTEGER [ID = INTEGER;] edikung
      ID <= INTEGER [ID = INTEGER;] edi [ID = INTEGER;]
```

B.3.d.2 Top-down Rightmost Derivation

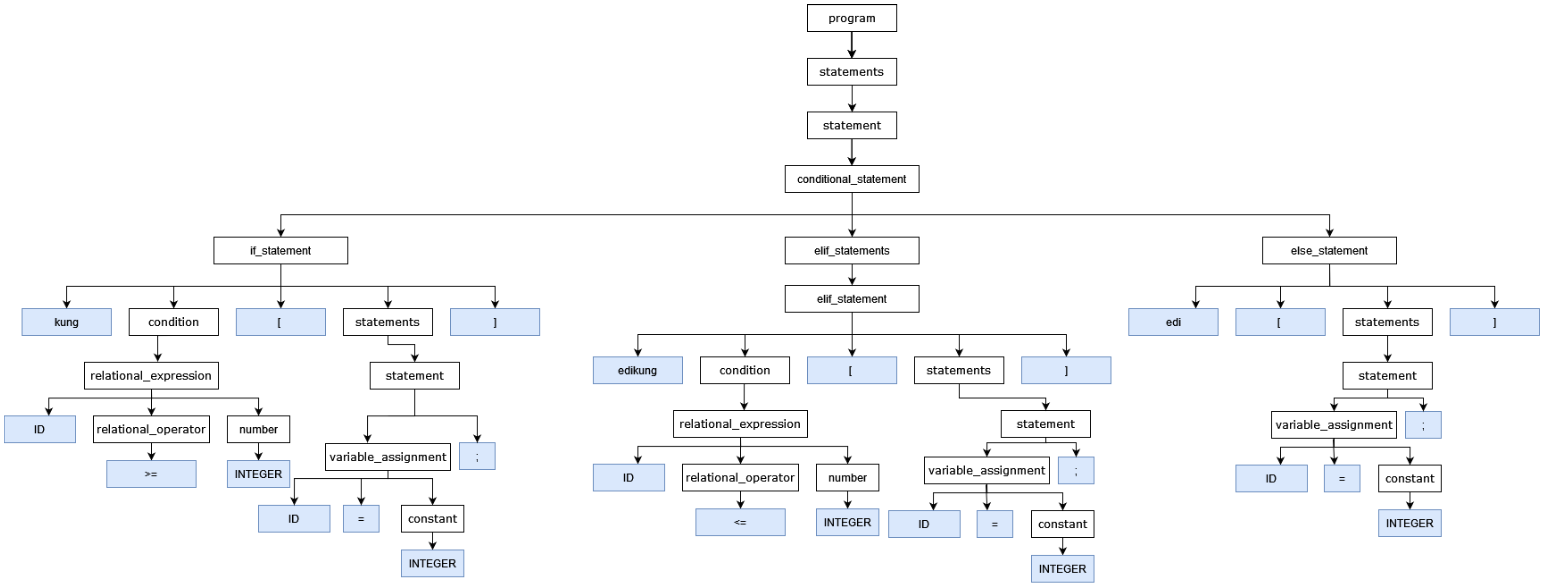
```
              <program> ::= <statements>
           <statements> ::= <statement>
            <statement> ::= <conditional_statement>
<conditional_statement> ::= <if_statement> <elif_statements> <else_statement>
                        ::= <if_statement> <elif_statements> edi [(<statements>)]
                        ::= <if_statement> <elif_statements> edi [<statement>]
                        ::= <if_statement> <elif_statements> edi [<assignment_statement>;]
                        ::= <if_statement> <elif_statements> edi
                              [ID = ( <expr> | <constant> | ID);]
                        ::= <if_statement> <elif_statements> edi [ID = <constant>;]
                        ::= <if_statement> <elif_statements> edi [ID = INTEGER;]
                        ::= <if_statement> <elif_statement> edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition>[(<statements>)]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition>[<statement>]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition> [<assignment_statement>;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition>
                              [ID = ( <expr> | <constant> | ID);] edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition> [ID = <constant>;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung <condition> [ID = INTEGER;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung <relational_expression> [ID = INTEGER;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung (<number> | ID | <bool>)
                              <relational_operator> (<number> | ID | <bool> )
                              [ID = INTEGER;] edi [ID = INTEGER;]

                        ::= <if_statement> edikung (<number> | ID | <bool>)
                              <relational_operator> <number> [ID = INTEGER;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung (<number> | ID | <bool>)
                              <relational_operator> INTEGER [ID = INTEGER;]
                              edi [ID = INTEGER;]
                        ::= <if_statement> edikung (<number> | ID | <bool>) <= INTEGER
                              [ID = INTEGER;] edi [ID = INTEGER;]
```

```
::= <if_statement> edikung ID <= INTEGER [ID = INTEGER;]
        edi [ID = INTEGER;]
::= kung <condition> [(<statements>)] edikung ID <= INTEGER
       [ID = INTEGER;] edi [ID = INTEGER;]
::= kung <condition> [<statement>] edikung ID <= INTEGER
       [ID = INTEGER;] edi [ID = INTEGER;]
::= kung <condition> [<assignment_statement>;] edikung
       ID <= INTEGER [ID = INTEGER;] edi [ID = INTEGER;]
::= kung <condition> [ID = ( <expr> | <constant> | ID);] edikung
       ID  <= INTEGER [ID = INTEGER;] edi [ID = INTEGER;]
::= kung <condition>[ID = <constant>; ] edikung ID <= INTEGER
       [ID = INTEGER;] edi [ID = INTEGER;]
::= kung <condition>[ID = INTEGER; ] edikung ID <= INTEGER
       [ID = INTEGER;]  edi [ID = INTEGER;]
::= kung <relational_expression> [ID = INTEGER ]  edikung
       ID <= INTEGER [ID = INTEGER]  edi [ID = INTEGER]
::= kung (<number> | ID | <bool>) <relational_operator>
       (<number> | ID | <bool> ) [ID = INTEGER ] edikung
       ID <= INTEGER [ID = INTEGER] edi [ID = INTEGER]
::= kung (<number> | ID | <bool>) <relational_operator> <number>
       [ID = INTEGER ] edikung ID <= INTEGER [ID = INTEGER]
       edi [ID = INTEGER]
::= kung (<number> | ID | <bool>) <relational_operator> INTEGER
       [ID = INTEGER ]  edikung ID <= INTEGER [ID = INTEGER]
       edi [ID = INTEGER]
::= kung (<number> | ID | <bool>) >= INTEGER [ID = INTEGER ]
       edikung ID <= INTEGER [ID = INTEGER] edi [ID = INTEGER]
::= kung  ID >= INTEGER [ID = INTEGER ] edikung
       ID <= INTEGER [ID = INTEGER]  edi [ID = INTEGER]
```

B.3.d.3 Top-down Leftmost Parse Tree



B.3.d.4 Top-down Rightmost Parse Tree



B.4 Iterative Statements

B.4.a For loop

Iterative Statement	Syntax	Example

Loop statement	ikot (<assigment_statement>,
	<condition>) [
	(<statements>)]
		ikot (i = 0, i < 3) [
		ilimbag(i);
		i = i + 1 ;
		]

B.4.a.1 Top-down Leftmost Derivation

```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <iterative_statement>
<iterative_statement> ::= <loop_statement>
     <loop_statement> ::= ikot (<assigment_statement>, <condition>) [(<statements>)]
                      ::= ikot (ID = (<expr> | <constant> | ID), <condition>) [(<statements>)]
                      ::= ikot (ID = <constant>, <condition> ) [(<statements>)]
                      ::= ikot (ID = INTEGER, <condition> ) [(<statements>)]
                      ::= ikot (ID = INTEGER, <relational_expression>) [(<statements>)]
                      ::= ikot (ID = INTEGER, (<number> | ID | <bool>) <relational_operator>
                              (<number> | ID | <bool>)) [(<statements>)]
                      ::= ikot (ID = INTEGER, ID <relational_operator>
                              (<number> | ID | <bool> ))[(<statements>)]
                      ::= ikot (ID = INTEGER, ID < (<number> | ID | <bool> ))
                               [(<statements>) ]
                      ::= ikot (ID = INTEGER, ID < <number>) [ (<statements>) ]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ (<statements>) ]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [<statement> <statements>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [<io_statement> <statements>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [<output_statement>
                              <statements> ]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (<io_options>
                              [ ,<io_options>]); <statements>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID [ ,<io_options>]);
                              <statements>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); <statements>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); <statement>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              <assignment_statement>]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = (<expr> | <constant> | ID);]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); ID = <expr>;]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); ID = <term>
                              {(+ | -) <term>};]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); ID = <factor>
                              {(+ | -) <term>};]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); ID = <exp>
                              {(+ | -) <term>};]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID); ID = ID
                              {(+ | -) <term>};]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = ID + <term>;]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = ID + <factor>;]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = ID + <exp>;]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = ID + <number>;]
                      ::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
                              ID = ID + INTEGER;]
```

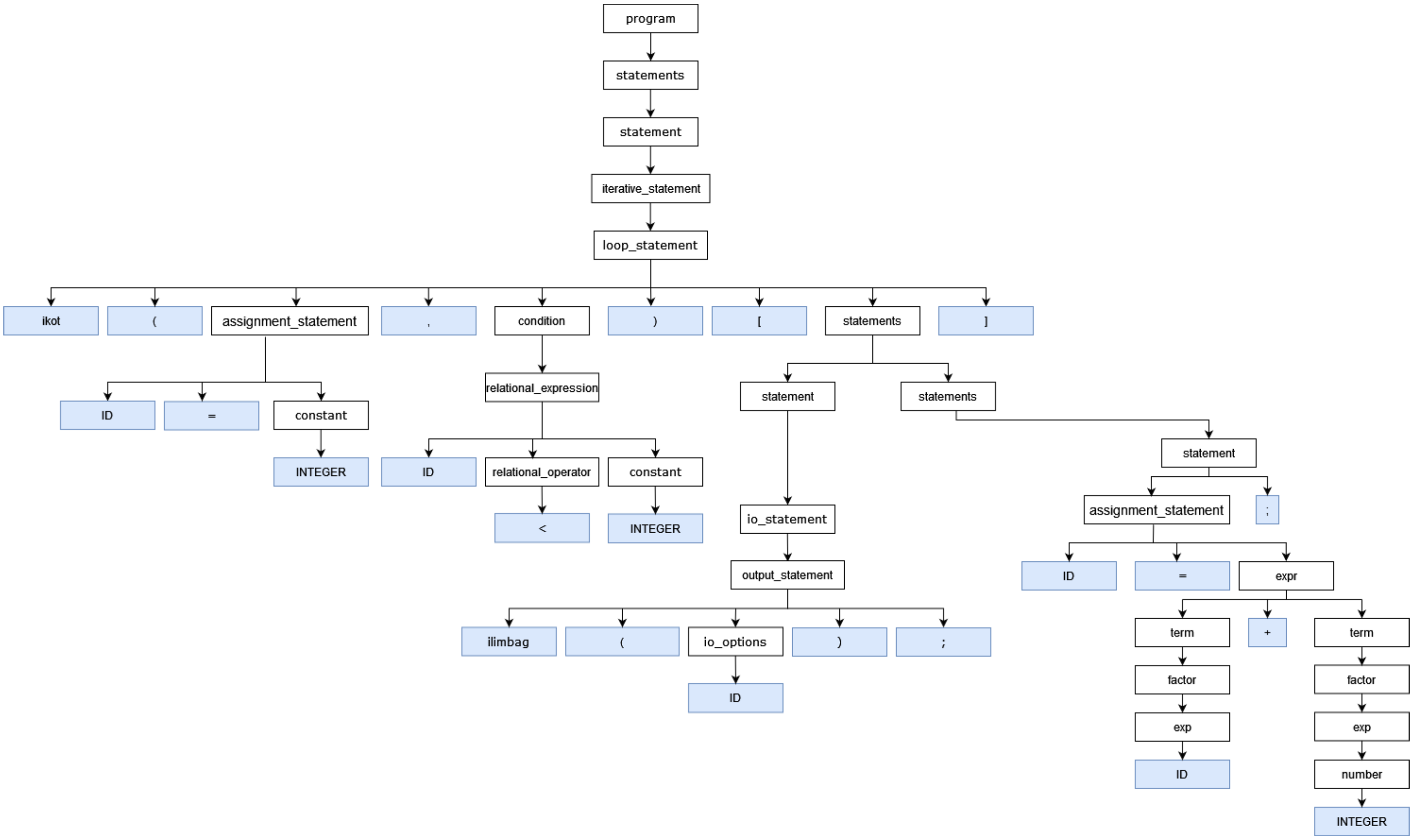
B.4.a.2 Top-down Rightmost Derivation

```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <iterative_statement>
<iterative_statement> ::= <loop_statement>
     <loop_statement> ::= ikot (<assigment_statement>, <condition>) [(<statements>]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              <statements>]
                      ::= ikot (<assigment_statement>, <condition>) [<statement> <statement>]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              <assignment_statement>]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              ID = (<expr> | <constant> | ID);]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              ID = <expr>;]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              ID = <term> {(+ | -) <term>};]
                      ::= ikot (<assigment_statement>, <condition>) [<statement>
                              ID = <term> + <term>;]
```

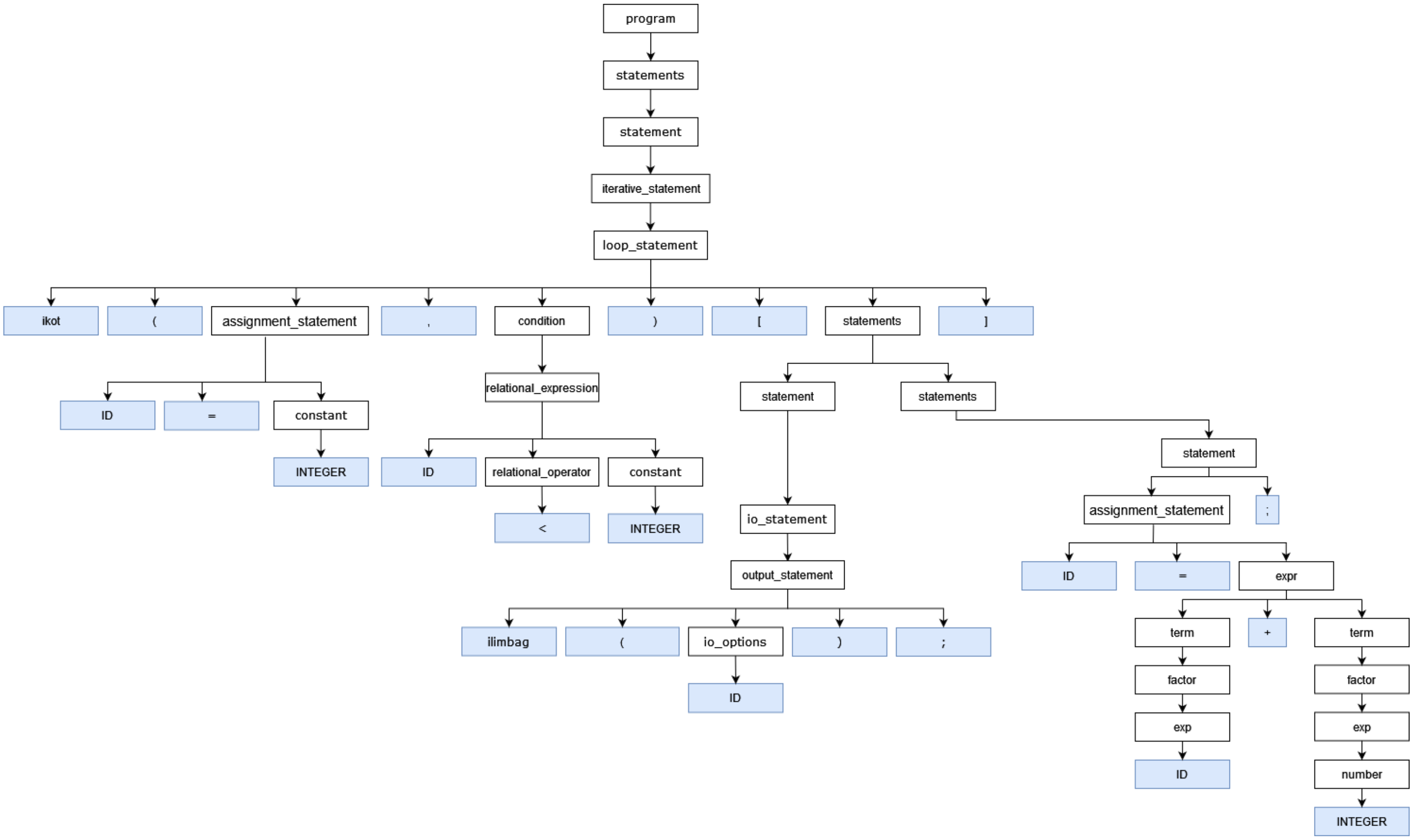
```
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <term> +  <factor>;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <term> +  <exp>;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <term> +  <number>;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <term> +  INTEGER;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <factor> + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID = <exp> + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [<statement>
        ID =  ID + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [<io_statement>
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [<output_statement>
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, <condition>)
        [ilimbag(<io_options>[,<io_options>]); ID = ID + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [ilimbag(<io_options>);
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, <condition>) [ilimbag (ID);
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, <relational_expression>) [ilimbag (ID);
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, (<number> | ID | <bool>)
        <relational_operator> (<number> | ID | <bool>))
        [ilimbag (ID); ID = ID + INTEGER;]
::= ikot (<assigment_statement>, (<number> | ID | <bool>)
        <relational_operator> <number>) [ilimbag (ID);
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, (<number> | ID | <bool>)
        <relational_operator> INTEGER) [ilimbag (ID);
        ID = ID + INTEGER;]
::= ikot (<assigment_statement>, (<number> | ID | <bool>) < INTEGER)
        [ilimbag (ID); ID = ID + INTEGER;]
::= ikot (<assigment_statement>, ID < INTEGER)[ilimbag (ID);
        ID = ID + INTEGER;]
```

::= ikot (ID [= <constant>], ID < INTEGER) [ilimbag (ID);
ID = ID + INTEGER;]
::= ikot (ID = <constant>, ID < INTEGER) [ilimbag (ID);
ID = ID + INTEGER;]
::= ikot (ID = INTEGER, ID < INTEGER) [ilimbag (ID);
ID = ID + INTEGER;]

B.4.a.3 Top-down Leftmost Parse-Tree



B.4.a.4 Top-down Rightmost Parse-Tree



B.4.b While loop

Iterative Statement	Syntax	Example

While statement	habang <condition> [
	<statements>]
		numero i = 0;
		habang i <= 2 [
		i = i + 1;
		ilimbag(“Hello”);
		]

B.4.b.1 Top-down Leftmost Derivation

```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <iterative_statement>
<iterative_statement> ::= <while_statement>
    <while_statement> ::= habang <condition> [<statements>]
                      ::= habang <relational_expression> [<statements>]
                      ::= habang (<number> | ID | <bool>) <relational_operator>
                          (<number> | ID | <bool> )[<statements>]
                      ::= habang ID <relational_operator>
                          (<number> | ID | <bool> )[<statements>]
                      ::= habang ID <= (<number> | ID | <bool> )[<statements>]
                      ::= habang ID <= <number> [<statements>]
                      ::= habang ID <= INTEGER [<statements>]
                      ::= habang ID <= INTEGER [ <statement> <statements> ]
                      ::= habang ID <= INTEGER [ <assignment_statement>; <statements> ]
                      ::= habang ID <= INTEGER [ID = (<expr> | <constant> | ID);
                              <statements> ]
                      ::= habang ID <= INTEGER [ID = <expr>; <statements> ]
                      ::= habang ID <= INTEGER [ID = <term> { (+ | -) <term>};
                              <statements> ]
                      ::= habang ID <= INTEGER [ID = <factor> { (+ | -) <term>};
                              <statements> ]
                      ::= habang ID <= INTEGER [ID = <exp> { (+ | -) <term>};
                              <statements> ]
                      ::= habang ID <= INTEGER [ID = ID { (+ | -) <term>}; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + <term>; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + <factor>; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + <exp>; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + <number>; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER; <statements> ]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER; <statement> ]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER; <io_statement> ]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER; <output_statement>]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER;
                              ilimbag(<io_options>[ , <io_options>]);]
                      ::= habang ID <= INTEGER [ ID = ID + INTEGER;
                              ilimbag(<constant>[ , <io_options>]);]
                      ::= habang ID <= INTEGER [ID = ID + INTEGER; ilimbag(STRING[ ,
                            <io_options>]);]
```

```
                                    := habang ID <= INTEGER [ID = ID + INTEGER; ilimbag(STRING);]
```

B.4.b.2 Top-down Rightmost Derivation

```
            <program> ::= <statements>
         <statements> ::= <statement>
          <statement> ::= <iterative_statement>
<iterative_statement> ::= <while_statement>
    <while_statement> ::= habang <condition> [<statements>]
                      ::= habang <condition> [<statement>]
                      ::= habang <condition> [<statement>]
                      ::= habang <condition> [<statement> <statements>]
                      ::= habang <condition> [<statement> <statement>]
                      ::= habang <condition> [<statement> <io_statement>]
                      ::= habang <condition> [<statement>  ilimbag(<io_options>[ ,
                          <io_options>]);]
                      ::= habang <condition> [<statement>  ilimbag(<io_options>);]
                      ::= habang <condition> [<statement>  ilimbag(<constant>);]
                      ::= habang <condition> [<statement>  ilimbag(STRING);]
                      ::= habang <condition> [<assignment_statement>; ilimbag(STRING);]
                      ::= habang <condition> [ ID = (<expr> | <constant> |ID);
                              ilimbag(STRING);]
                      ::= habang <condition> [ID = (<expr>); ilimbag(STRING );]
                      ::= habang <condition> [ID = (<term> { (+ | -) <term>});
                           ilimbag(STRING);]
                      ::= habang <condition> [ID = (<term> +  <term>);  ilimbag(STRING);]
                      ::= habang <condition> [ID = (<term> + <factor>); ilimbag(STRING);]
                      ::= habang <condition> [ID = (<term> + <exp> ); ilimbag(STRING);]
                      ::= habang <condition> [ID = (<term> + <number>); ilimbag(STRING);]
                      ::= habang <condition> [ID = (<term> +  INTEGER);
                              ilimbag(STRING);]
                      ::= habang <condition> [ID = (<factor> + INTEGER);
                              ilimbag(STRING);]
                      ::= habang <condition> [ID = (<exp> + INTEGER); ilimbag(STRING);]
                      ::= habang <condition> [ID = (ID + INTEGER); ilimbag(STRING);]
                      ::= habang <relational_expression> [ID = ID + INTEGER;
                              ilimbag(STRING);]
                      ::= habang  (<number> | ID | <bool>) <relational_operator>
                              (<number> | ID | <bool> ) [ID = ID + INTEGER;
                              ilimbag(STRING);]
```

::= habang (<number> | ID | <bool>) <relational_operator> <number>

[ID = ID + INTEGER; ilimbag(STRING);]

::= habang (<number> | ID | <bool>) <relational_operator> INTEGER

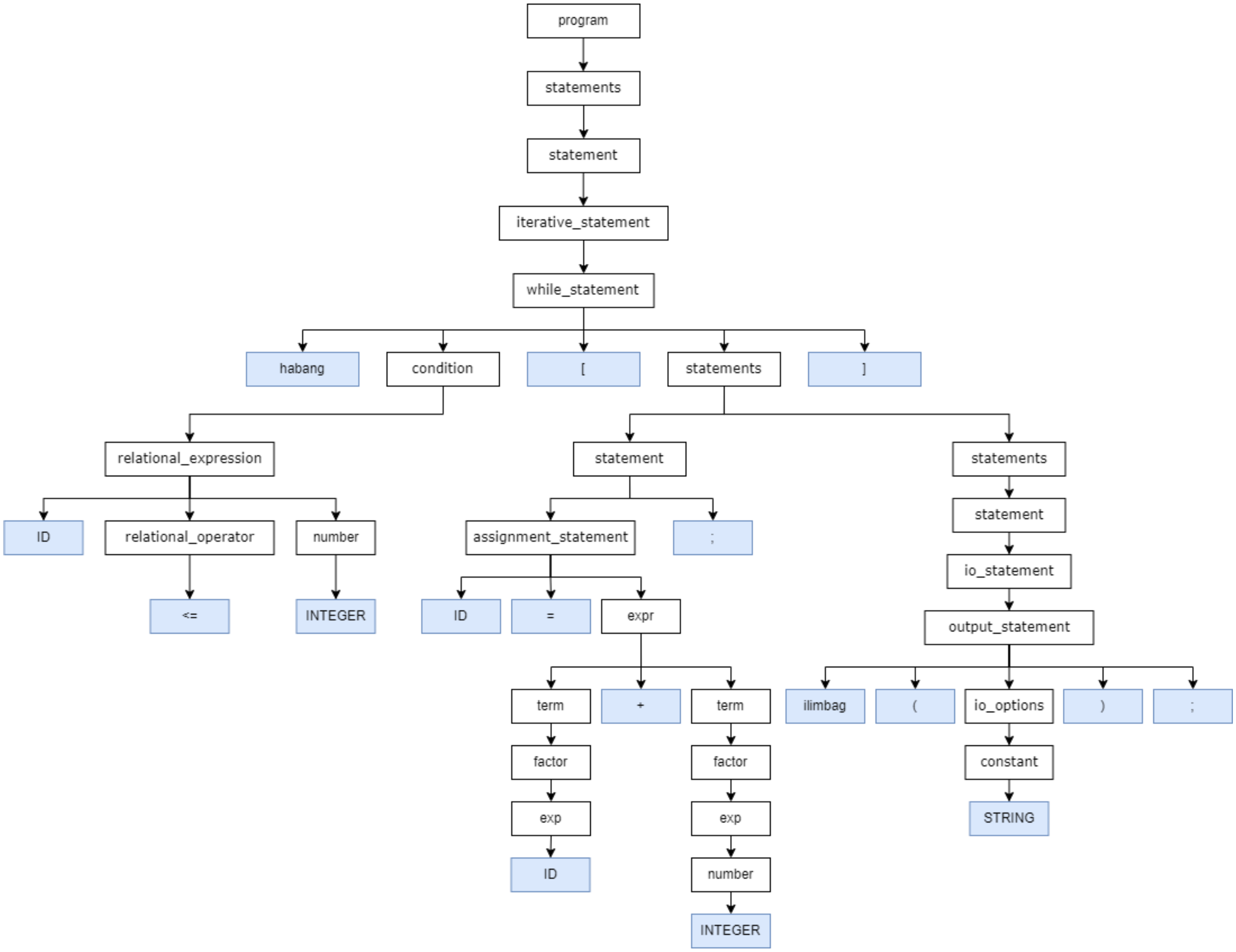
[ID = ID + INTEGER; ilimbag(STRING);]

::= habang (<number> | ID | <bool>) <= INTEGER

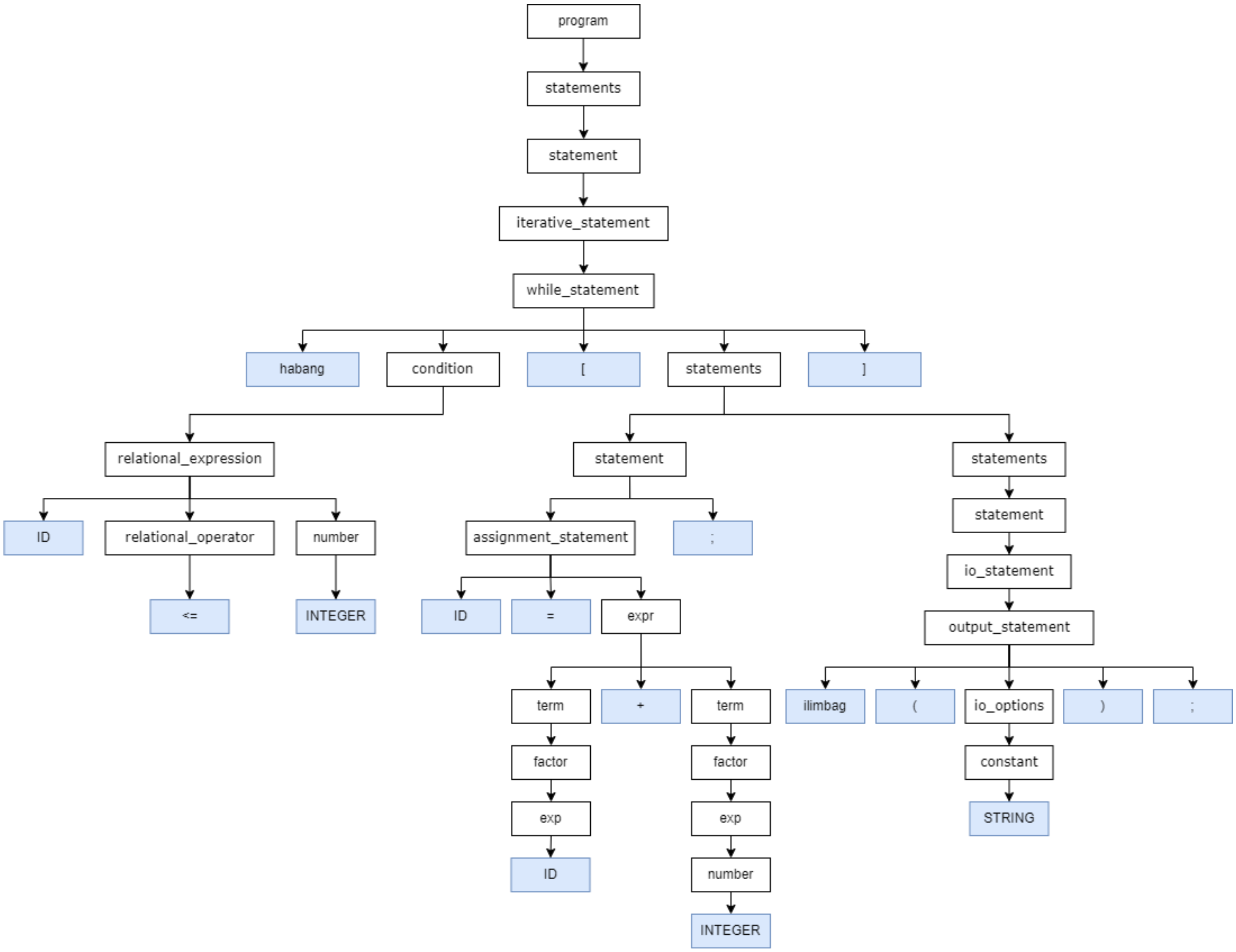
[ID = ID + INTEGER; ilimbag(STRING);]

::= habang ID <= INTEGER [ID = ID + INTEGER; ilimbag(STRING);]

B.4.b.3 Top-down Leftmost Parse-Tree



B.4.b.4 Top-down Rightmost Parse-Tree



B.5 Input Statements

B.5.a Input Statement

Input Function	Syntax	Example
Input statement	ID = lagyan( );	a = lagyan();

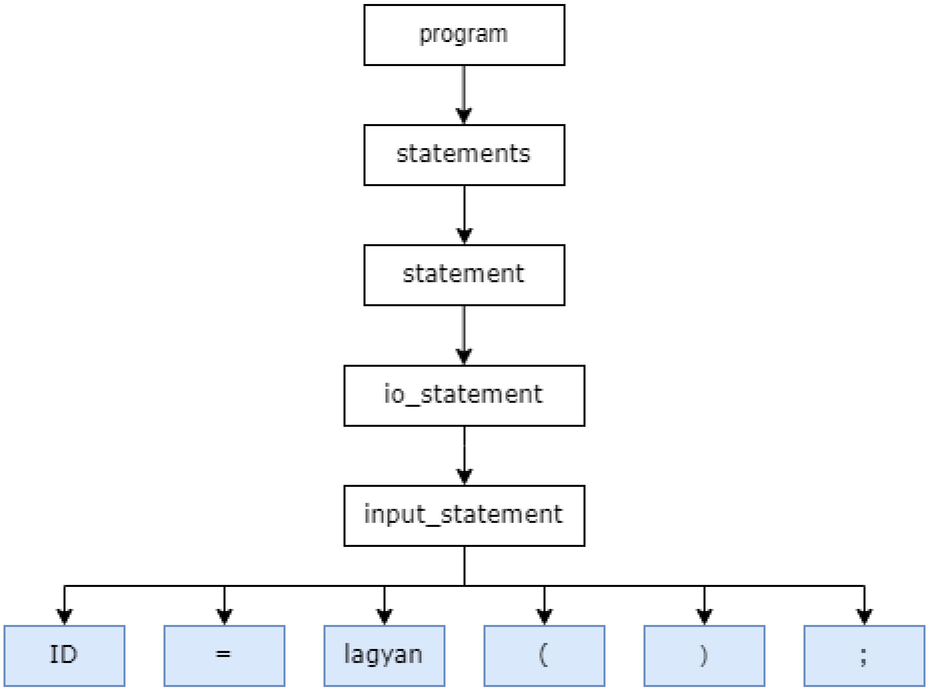
B.5.a.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan([<io_options>]);
::= ID = lagyan( );

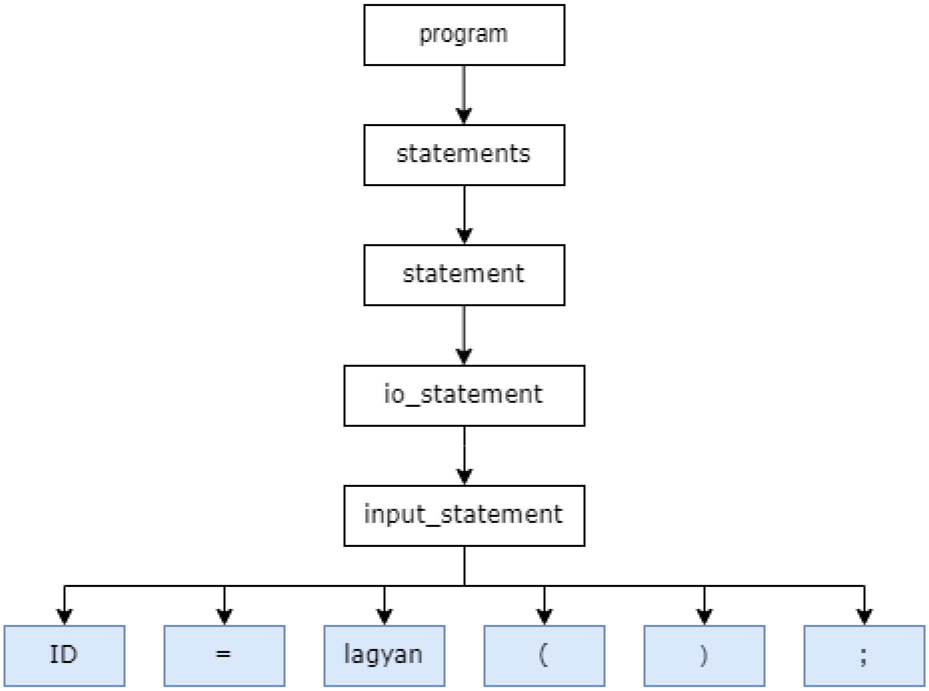
B.5.a.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan([<io_options>]);
::= ID = lagyan( );

B.5.a.3 Top-down Leftmost Parse-Tree



B.5.a.4 Top-down Rightmost Parse-Tree



B.5.b Input Statement with identifier

Input Function	Syntax	Example
Input statement with identifier	ID = lagyan(ID);	a = lagyan(b);

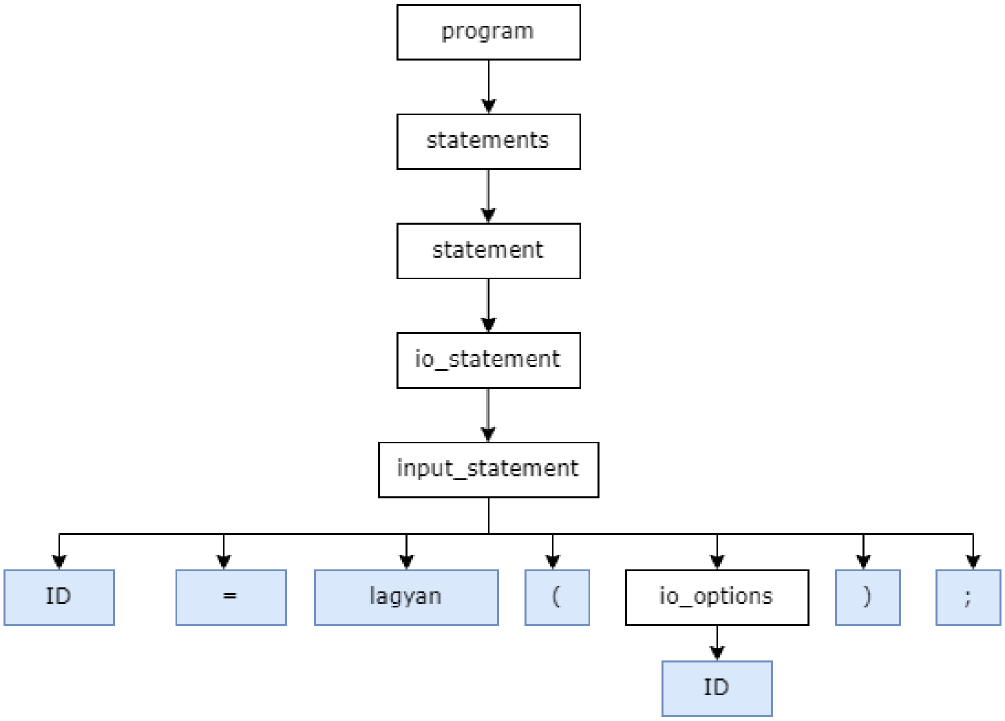
B.5.b.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan([<io_options>]);
<input_statement> ::= ID = lagyan(<io_options>);
::= ID = lagyan(ID);

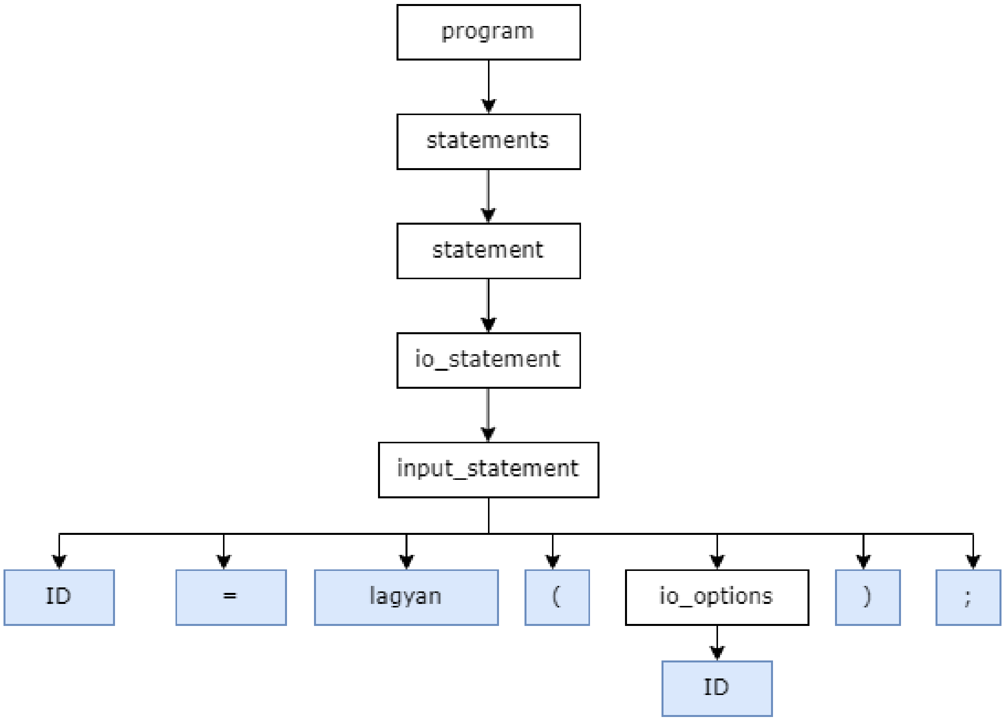
B.5.b.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan([<io_options>]);
<input_statement> ::= ID = lagyan(<io_options>);
::= ID = lagyan(ID);

B.5.b.3 Top-down leftmost Parse tree



B.5.b.4 Top-down Rightmost Parse tree



B.5.c Input Statement with Constant

Input Function	Syntax	Example
Input statement with constant	ID = lagyan(<constant>);	a = lagyan(“hello”);

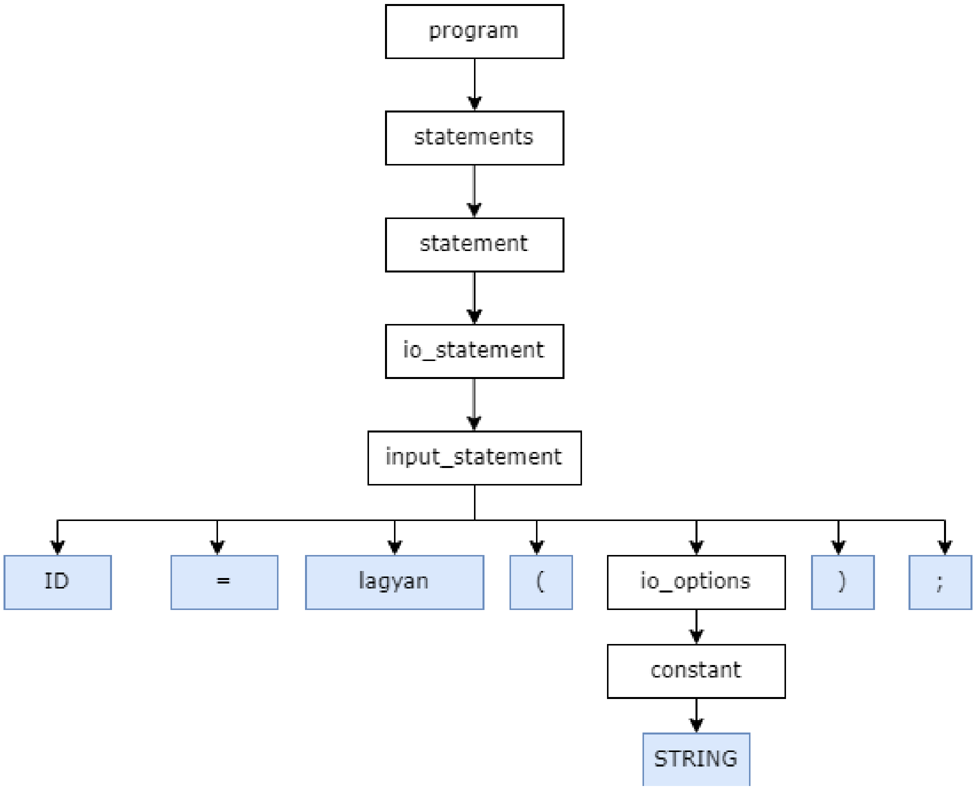
B.5.c.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan([<io_options>]);
<input_statement> ::= ID = lagyan(<io_options>);
::= ID = lagyan(<constant>);
: := ID = lagyan(STRING);

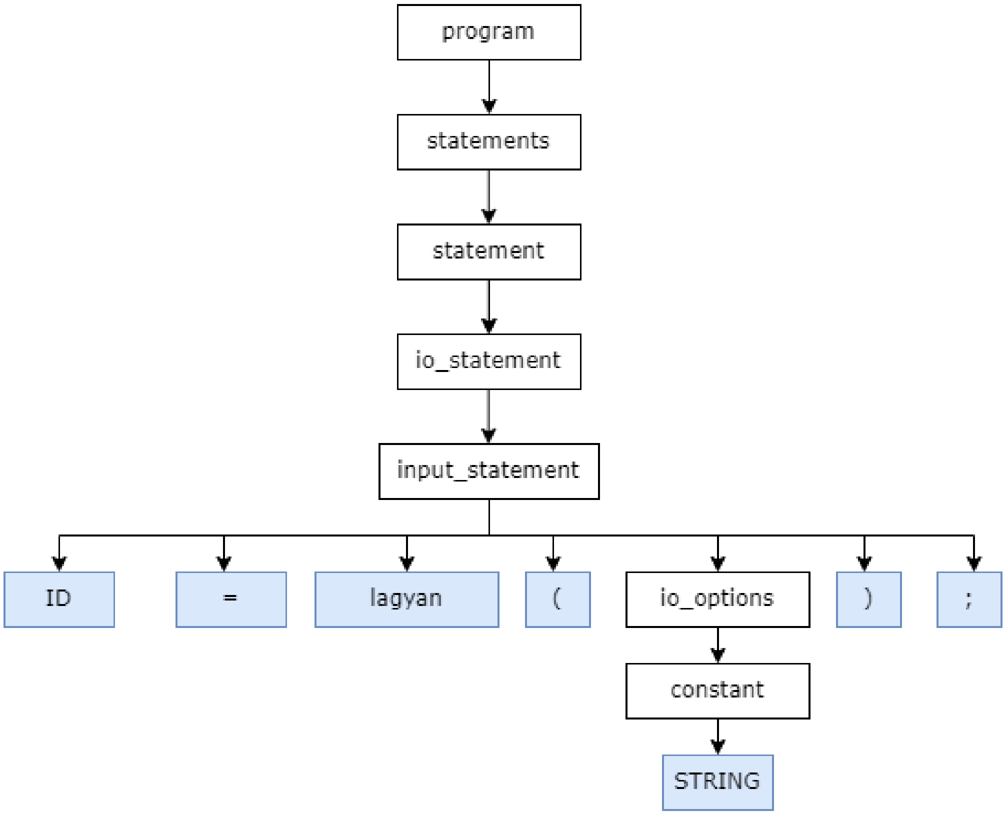
B.5.c.2 Top-down Rightmost Derivation

```
          <program> ::= <statements>
       <statements> ::= <statement>
        <statement> ::= <io_statement>
     <io_statement> ::= <input_statement>
  <input_statement> ::= ID = lagyan([<io_options>]);
  <input_statement> ::= ID = lagyan(<io_options>);
                    ::= ID = lagyan(<constant>);
                    ::= ID = lagyan(STRING);
```

B.5.c.3 Top-down Leftmost Parse tree



B.5.c.4 Top-down Rightmost Parse tree



B.5.d Input Statement with Expression

Input Function	Syntax	Example
Input statement with expression	ID = lagyan(<expr>);	a = lagyan(3-2);

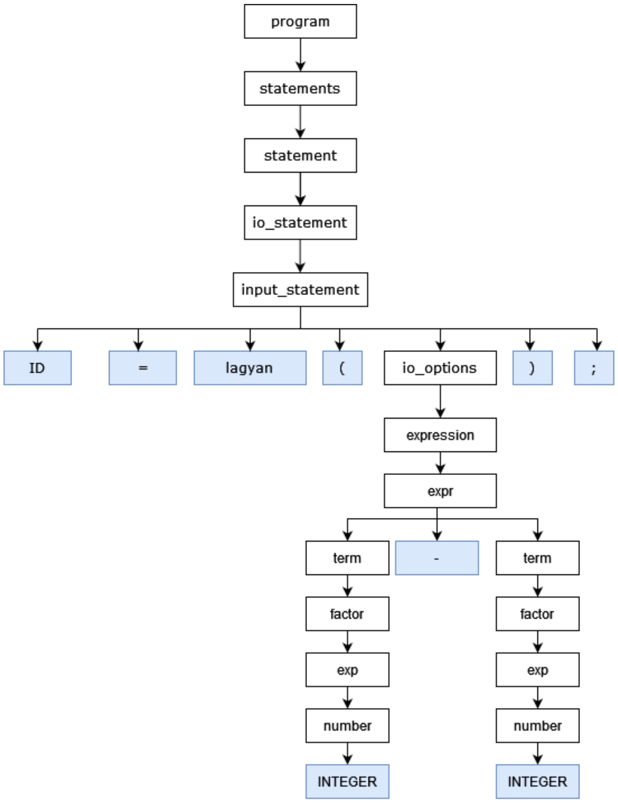
B.5.d.1 Top-down Leftmost Derivation

```
        <program> ::= <statements>
     <statements> ::= <statement>
      <statement> ::= <io_statement>
   <io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan ([<io_options>]);
                  ::= ID = lagyan (<expression>);
                  ::= ID = lagyan (<expr>);
                  ::= ID = lagyan (<term> {(+|-) <term>});
                  ::= ID = lagyan (<factor> {(+|-) <term>});
                  ::= ID = lagyan (<exp> {(+|-) <term>});
                  ::= ID = lagyan (<number> {(+|-) <term>});
                  ::= ID = lagyan (INTEGER {(+|-) <term>});
                  ::= ID = lagyan (INTEGER - <term>);
                  ::= ID = lagyan (INTEGER - <factor>);
                  ::= ID = lagyan (INTEGER - <exp>);
                  ::= ID = lagyan (INTEGER - <number>);
                  ::= ID = lagyan (INTEGER - INTEGER);
```

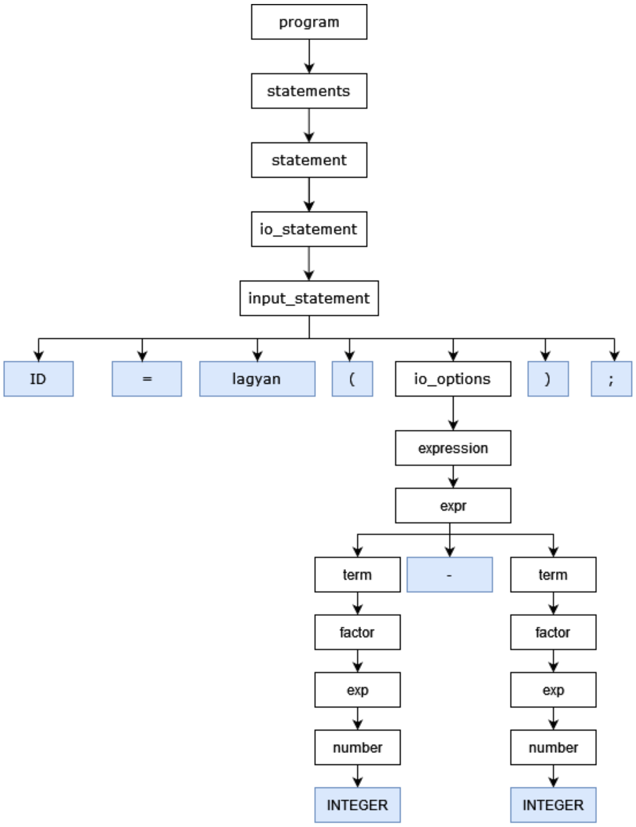
B.5.d.2 Top-down Rightmost Derivation

```
        <program> ::= <statements>
     <statements> ::= <statement>
      <statement> ::= <io_statement>
   <io_statement> ::= <input_statement>
<input_statement> ::= ID = lagyan ([<io_options>]);
                  ::= ID = lagyan (<expression>);
                  ::= ID = lagyan (<expr>);
                  ::= ID = lagyan (<term> {(+|-) <term>});
                  ::= ID = lagyan (<term> - <term>);
                  ::= ID = lagyan (<term> - <factor>);
                  ::= ID = lagyan (<term> - <exp>);
                  ::= ID = lagyan (<term> - <number>);
                  ::= ID = lagyan (<term> - INTEGER);
                  ::= ID = lagyan (<factor> - INTEGER);
                  ::= ID = lagyan (<exp> - INTEGER);
                  ::= ID = lagyan (<number> - INTEGER);
                  ::= ID = lagyan (INTEGER - INTEGER);
```

B.5.d.3 Top-down Leftmost Parse-Tree



B.5.d.4 Top-down Rightmost Parse-Tree



B.6 Output Statements

B.6.a Outputs Variable

Output Function	Syntax	Example
Outputs variable	ilimbag(ID) ;	ilimbag(a);

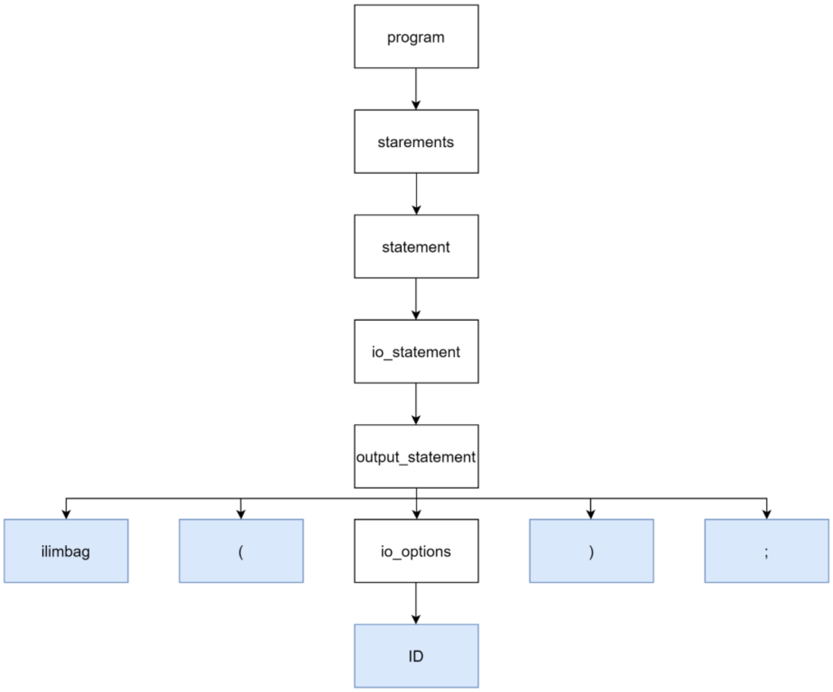
B.6.a.1 Top-down Leftmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <output_statement>
<output_statement> ::= ilimbag(<io_options>[,<io_options>]);
<output_statement> ::= ilimbag(ID[,<io_options>]);
::= ilimbag(ID);

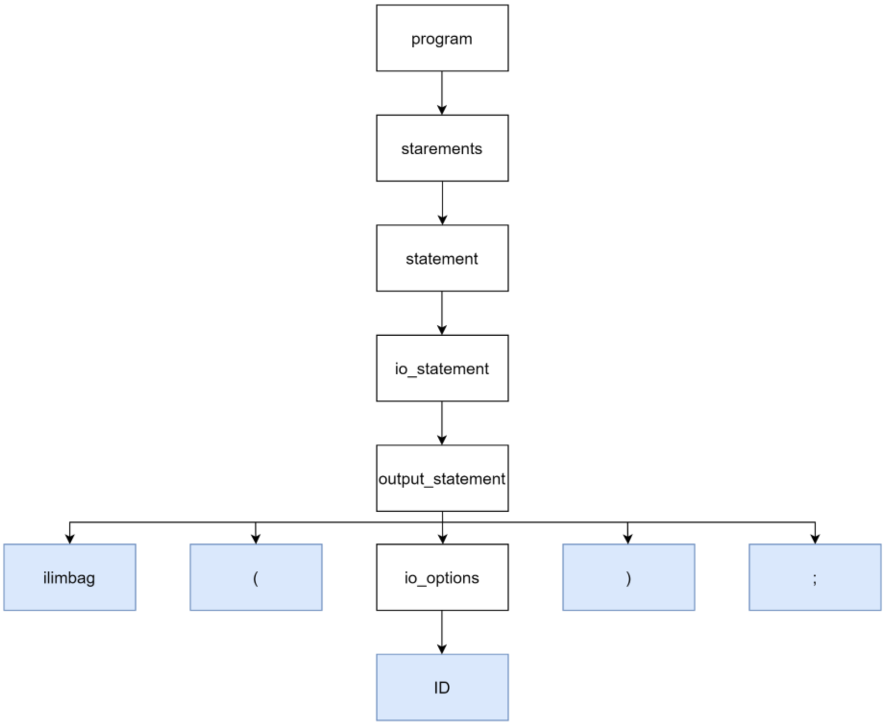
B.6.a.2 Top-down Rightmost Derivation

<program> ::= <statements>
<statements> ::= <statement>
<statement> ::= <io_statement>
<io_statement> ::= <output_statement>
<output_statement> ::= ilimbag(<io_options>[,<io_options>]);
<output_statement> ::= ilimbag(<io_options>);
::= ilimbag(ID);

B.6.a.3 Top-down Leftmost Parse-Tree



B.6.a.4 Top-down Rightmost Parse-Tree



B.6.b Outputs Variable with Constant

Output Function	Syntax	Example
Outputs variable with constant	ilimbag(<constant>,ID) ;	ilimbag(“Hello”,a);

B.6.b.1 Top-down Leftmost Derivation

```
        <program> ::= <statements>
     <statements> ::= <statement>
      <statement> ::= <io_statement>
   <io_statement> ::= <output_statement>
<output_statement> ::= ilimbag(<io_options>[,<io_options>]);
                   ::= ilimbag(<constant>[,<io_options>]);
                   ::= ilimbag(STRING[,<io_options>]);
                   ::= ilimbag(STRING,<io_options>);
                   ::= ilimbag(STRING,ID);
```

B.6.b.2 Top-down Rightmost Derivation

<program> ::= <statements>

<statements> ::= <statement>

<statement> ::= <io_statement>

<io_statement> ::= <output_statement>

<output_statement> ::= ilimbag(<io_options>[,<io_options>]);

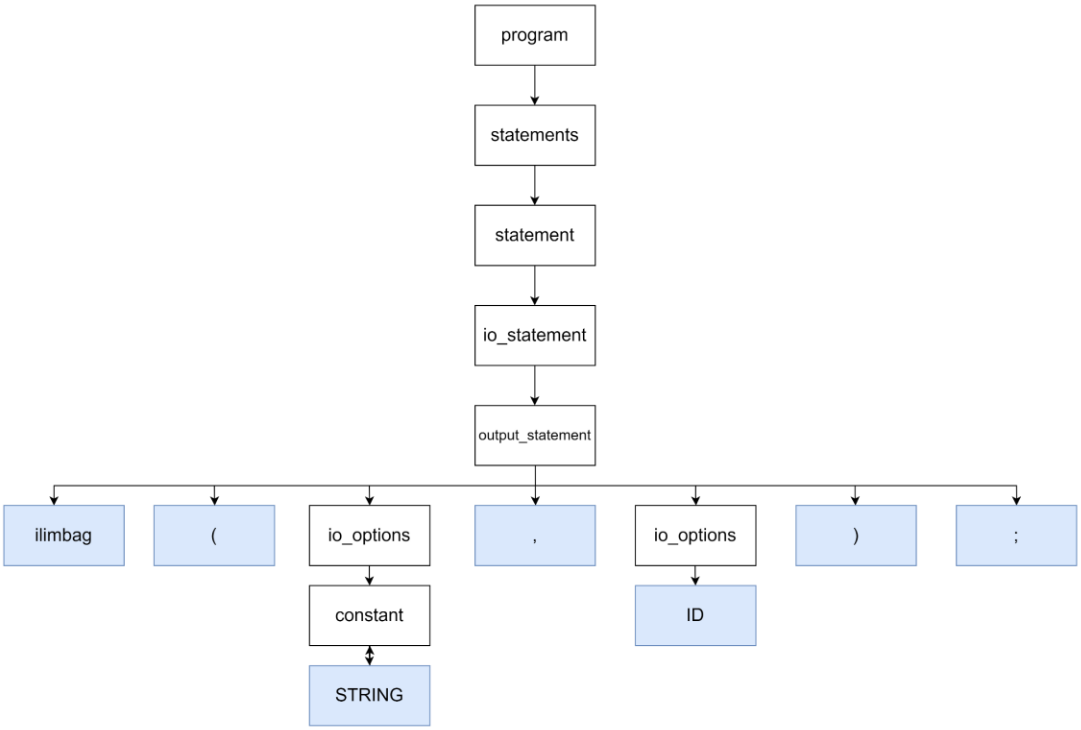
<output_statement> ::= ilimbag(<io_options>,<io_options>);

::= ilimbag(<io_options>, ID);

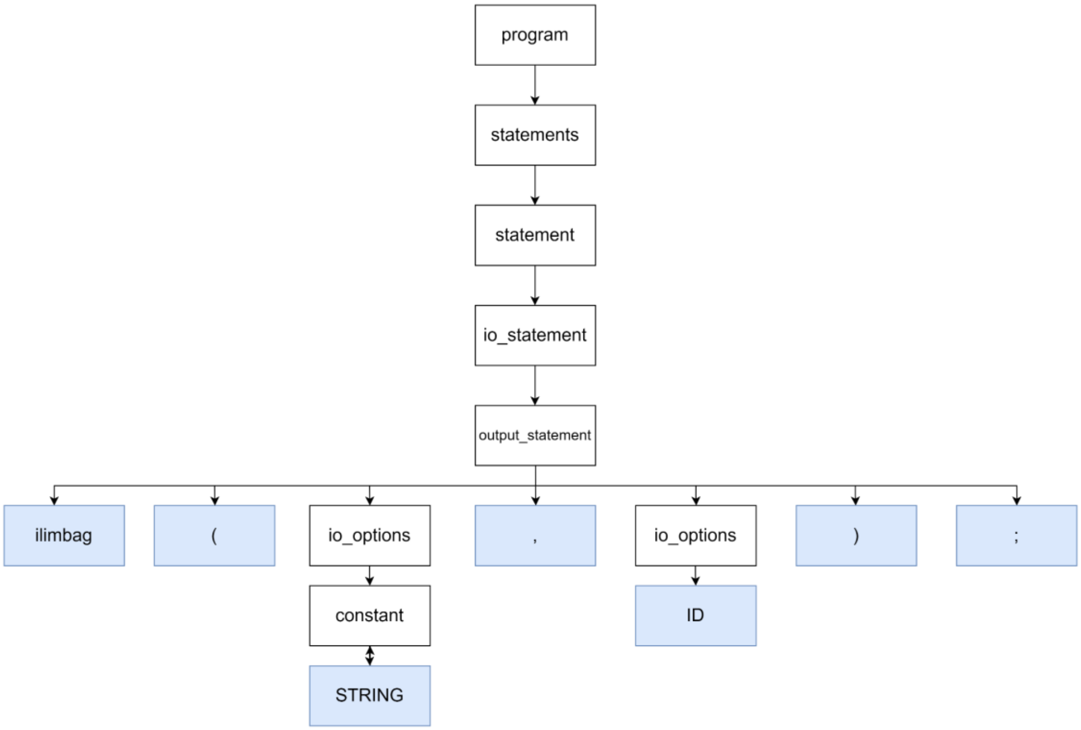
::= ilimbag(<constant>, ID);

::= ilimbag(STRING,ID);

B.6.b.3 Top-down Leftmost Parse-Tree



B.6.b.4 Top-down Rightmost Parse-Tree



B.6.c Outputs Variable modified in an Expression

Output Function	Syntax	Example
Outputs variable which is modified in an expression	ilimbag(<expression>);	ilimbag(a+2);

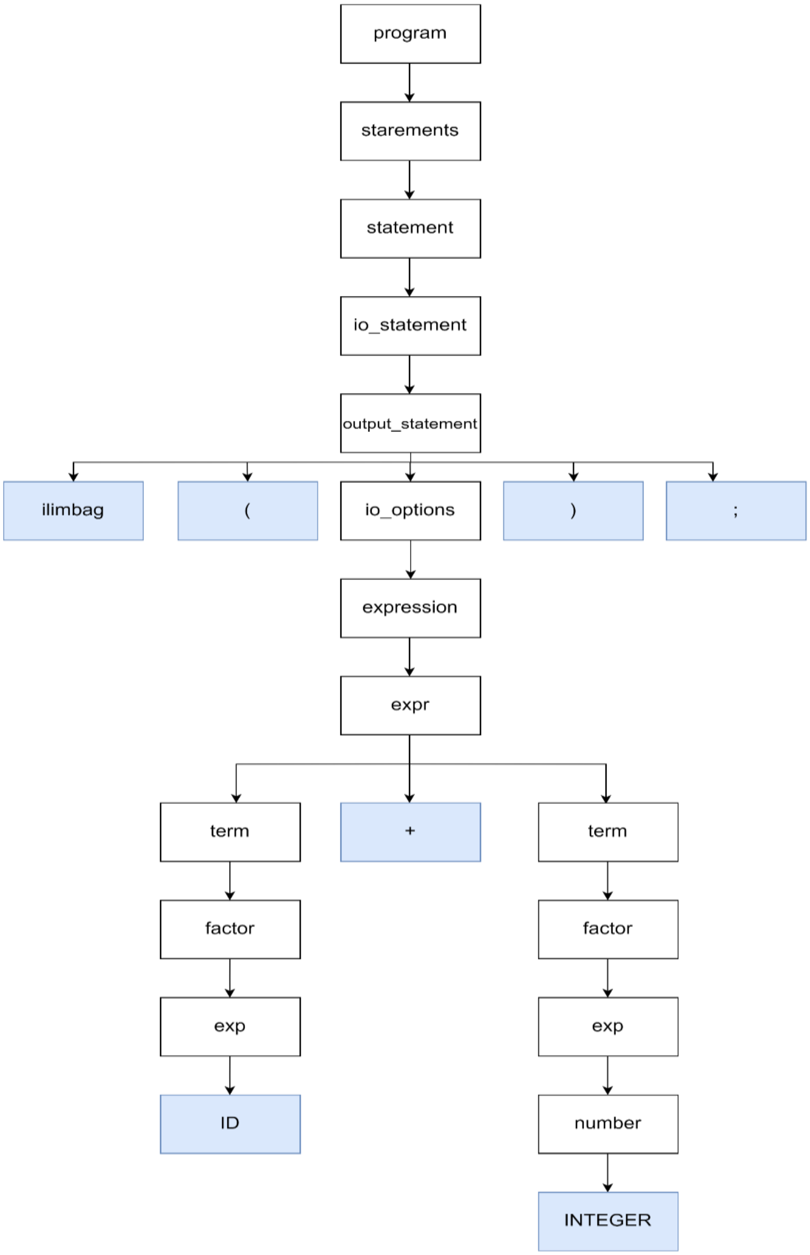
B.6.c.1 Top-down Leftmost Derivation

```
         <program> ::= <statements>
      <statements> ::= <statement>
       <statement> ::= <io_statement>
    <io_statement> ::= <output_statement>
<output_statement> ::= ilimbag(<io_options>[,<io_options> ]);
<output_statement> ::= ilimbag(<io_options>[,<io_options> ]);
                   ::= ilimbag(<expression>[,<io_options> ]);
                   ::= ilimbag(<expr>[,<io_options> ]);
                   ::= ilimbag(<term> {(+|-) <term>}[,<io_options> ]);
                   ::= ilimbag(<factor> {(+|-) <term>}[,<io_options> ]);
                   ::= ilimbag(<exp> {(+|-) <term>}[,<io_options> ]);
                   ::= ilimbag(ID{(+|-) <term>}[,<io_options> ]);
                   ::= ilimbag(ID + <term>[,<io_options> ]);
                   ::= ilimbag(ID + <factor>[,<io_options> ]);
                   ::= ilimbag(ID + <exp>[,<io_options> ]);
                   ::= ilimbag(ID + <number>[,<io_options> ]);
                   ::= ilimbag(ID + INTEGER [,<io_options> ]);
                   ::= ilimbag(ID + INTEGER);
```

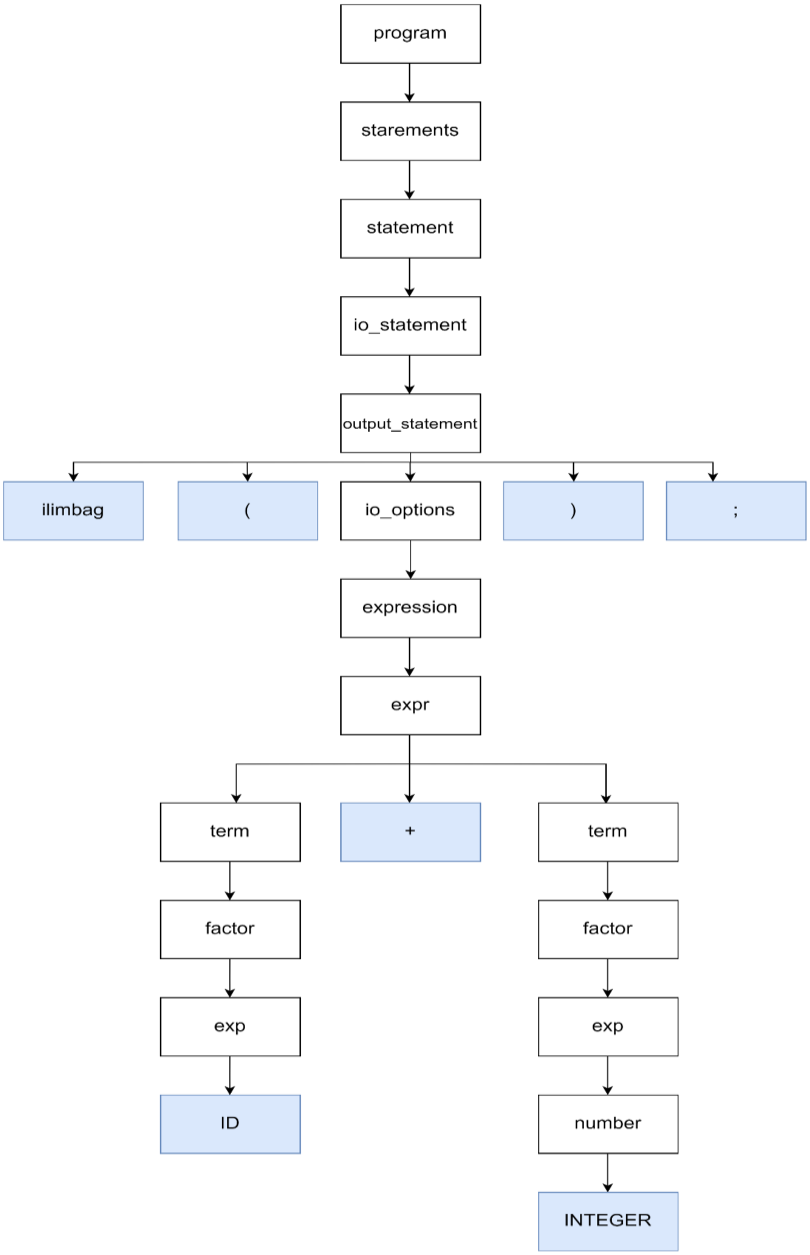
B.6.c.2 Top-down Rightmost Derivation

```
         <program> ::= <statements>
      <statements> ::= <statement>
       <statement> ::= <io_statement>
    <io_statement> ::= <output_statement>
<output_statement> ::= ilimbag(<io_options>[,<io_options> ]);
<output_statement> ::= ilimbag(<io_options>);
                   ::= ilimbag(<expression>);
                   ::= ilimbag(<expr>);
                   ::= ilimbag(<term> {(+|-) <term>});
                   ::= ilimbag(<term> + <term>);
                   ::= ilimbag(<term> + <factor>);
                   ::= ilimbag(<term> + <exp>);
                   ::= ilimbag(<term> + <number>);
                   ::= ilimbag(<term> + INTEGER);
                   ::= ilimbag(<factor> + INTEGER);
                   ::= ilimbag(<exp> + INTEGER);
                   ::= ilimbag( ID + INTEGER);
```

B.6.c.3 Top-down Leftmost Parse-Tree



B.6.c.4 Top-down Rightmost Parse-Tree


program
starements
statement
io_statement
output_statement
ilimbag
(
io_options
)
;
expression
expr
term
+
term
factor
factor
exp
exp
ID
number
INTEGER