****

Pamantasan ng Lungsod ng Maynila

(University of the City of Manila)

Intramuros, Manila

College of Engineering and Technology

Computer Science Department

Compiler Design – CSC 0324.1-1

**J4CK Compiler**

A Proposed Compiler

Submitted to the Computer Science Department

Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements

For Compiler Design

ABENALES, John Carlo H.

LIMPIN, James Patrick T.

MAGSINO, Jun Allen R.

SALAS, Luke Joshua B.

Ms. Leisyl Ocampo-Mahusay

Adviser

February 2021

Table of Contents

[**LANGUAGE OVERVIEW** 4](#_Toc75356968)

[**GENERAL RULES** 5](#_Toc75356969)

[**STRUCTURE OF THE PROGRAM** 6](#_Toc75356970)

[**Printing of Hello World** 7](#_Toc75356971)

[**RESERVED WORDS** 8](#_Toc75356972)

[Main 8](#_Toc75356973)

[Data Types 8](#_Toc75356974)

[Input and Output Statements 8](#_Toc75356975)

[Conditional Statements 9](#_Toc75356976)

[Looping Statements 9](#_Toc75356977)

[Others 10](#_Toc75356978)

[**RESERVED SYMBOLS** 11](#_Toc75356979)

[Arithmetic Operator 11](#_Toc75356980)

[Relational Operator 11](#_Toc75356981)

[Logical Operator 11](#_Toc75356982)

[Assignment Operator 12](#_Toc75356983)

[Unary Operator 12](#_Toc75356984)

[Others 13](#_Toc75356985)

[**SPECIFIC RULES** 14](#_Toc75356986)

[**DATA TYPES** 14](#_Toc75356987)

[**LITERALS** 15](#_Toc75356988)

[**IDENTIFIERS** 19](#_Toc75356989)

[**CONSTANTS** 20](#_Toc75356990)

[**VARIABLES** 21](#_Toc75356991)

[**ARITHMETIC EXPRESSION** 22](#_Toc75356992)

[**ASSIGNMENT STATEMENT** 24](#_Toc75356993)

[**RELATIONAL EXPRESSION** 25](#_Toc75356994)

[**LOGICAL EXPRESSION** 27](#_Toc75356995)

[**STRING CONCATENATION** 28](#_Toc75356996)

[**UNARY OPERATORS** 29](#_Toc75356997)

[**OPERATORS PRECEDENCE** 31](#_Toc75356998)

[**COMMENTS** 32](#_Toc75356999)

[**INPUT AND OUTPUT STATEMENTS** 33](#_Toc75357000)

[**CONDITIONAL STATEMENT** 37](#_Toc75357001)

[**LOOPING STATEMENT** 40](#_Toc75357002)

[**CONTROL STATEMENTS** 44](#_Toc75357003)

[**Break** 44](#_Toc75357004)

[**Continue** 45](#_Toc75357005)

[**ARRAYS** 46](#_Toc75357006)

[**deck (struct)** 52](#_Toc75357007)

[**FUNCTIONS** 56](#_Toc75357008)

[**BLANK STATEMENT** 59](#_Toc75357009)

[**REGULAR EXPRESSION** 60](#_Toc75357010)

[Reserved words 60](#_Toc75357011)

[Reserved symbols 61](#_Toc75357012)

[Literals and Comment 62](#_Toc75357013)

[**REGULAR DEFINITION** 64](#_Toc75357014)

[Boundary 64](#_Toc75357015)

[Reserved Words 64](#_Toc75357016)

[Reserved Symbols 65](#_Toc75357017)

[Identifiers 65](#_Toc75357018)

[Literals 65](#_Toc75357019)

[**TRANSITION DIAGRAM** 66](#_Toc75357020)

[**Reserved Words** 66](#_Toc75357021)

[**Reserved Symbols** 68](#_Toc75357022)

[**PIECELIT** 70](#_Toc75357023)

[**SETLIT** 70](#_Toc75357024)

[**NUMLIT** 70](#_Toc75357025)

[**DECIMALLIT** 71](#_Toc75357026)

[**IDENTIFIERS** 73](#_Toc75357027)

[**CONTEXT FREE GRAMMAR** 74](#_Toc75357028)

[**FIRST SET** 81](#_Toc75357029)

[**FOLLOW SET** 85](#_Toc75357030)

[**PREDICT SET** 0](#_Toc75357031)

[**PREDICT TABLE** 13](#_Toc75357032)

[**Appendix A - Lexical Test Scripts** 0](#_Toc75357033)

[**Reserved Words** 0](#_Toc75357034)

[**Reserved Symbols** 10](#_Toc75357035)

[**piecelit** 27](#_Toc75357036)

[**setlit** 28](#_Toc75357037)

[**numlit** 28](#_Toc75357038)

[**decimallit** 29](#_Toc75357039)

[**Identifiers** 30](#_Toc75357040)

[**Appendix B – Syntax Test Scripts** 31](#_Toc75357041)

# **LANGUAGE OVERVIEW**

***J4CK*** is a software development platform which mostly adapts the English Language and the structure of C Language. This compiler caters people who are starting to understand the field of programming and computing. Also, it targets those people especially developers who wants to migrate from English-like syntax terms.

Logo:



***J4CK*** came from the idea of Jack in playing cards. This “Jack” from its modern meaning “man” or “common man” means it targets all people that can be an inspiring or professional developer. In addition, the ‘4’ stands for those people who created this idea which have letter ‘J’ in their names. Four suits in every corners represent division in society that means each and every individual can use this compiler as a tool for software development.

# **GENERAL RULES**

1. A program must have a main function wherein it equivalents to TABLE() function that is needed to be declared. [ main() = TABLE( ) ]
2. Declaration of table function must be declared *once only*.
3. The TABLE function must be in **ALL CAPS**.
4. The curly braces { } in C are also being used as the start-end of the function.
5. Block of code under looping statement and conditional statement must be enclosed with curly braces even if it has only one statement.
6. Statement/s ends with a semicolon except for loop, while loop,

conditional statements. ( ; )

1. The global declaration must be declared before the TABLE function. (Constant Declarations, Variable Declaration, Data Structure, Functions). Also, there is no specific order that must be followed.
2. Declaration of local variables are allowed anywhere within the TABLE() and user – defined functions. Hence, before using the variables it must be declared first.
3. J4CKis a *case sensitive*programming language.
4. Single line comments must begin with a vertical bar symbol “ | ”.
5. Comments must be before and after a statement/s.
6. Single quotation marks ( ‘ ‘ ) must be used in enclosing a character data type values.
7. Double quotation marks ( “ “ ) must be used in enclosing a string data type values.
8. The reserved word **tell** pertains to printing a statements and **check** is for the input statements. Hence, all reserved words are in **small case letters** expect for the TABLE function.
9. All reserved words cannot be used as identifiers.

# **STRUCTURE OF THE PROGRAM**

<global declaration>

TABLE ()

{

<statement/s>

<comment/s>

}

where <global declaration/s> 🡪 <constant declaration/s>

<global variable declaration/s>

<data structure declaration/s>

<function declaration/s>

<statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

# **Printing of Hello World**

TABLE()

{

tell (“Hello World!”);

}

# **RESERVED WORDS**

## Main

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved word** | **Description** |
| main | TABLE | It is the function that serves as the entry point of any C program. |

## Data Types

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved Words** | **Description** |
| int | num | A data type that holds whole numbers value. |
| float | decimal | A data type that holds decimal numbers value. |
| char | piece | A data type that holds a singular character value. |
| string | set | A data type that holds characters value. |
| boolean | option | A data type that holds ‘true’ or ‘false’ value. |
| void | void | A word that represents the absence of value. |

## Input and Output Statements

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved Words** | **Description** |
| printf | tell | Displaying output |
| scanf | check | Scanning input |

## Conditional Statements

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved Words** | **Description** |
| if | if | If a certain condition is true, it executes the block of code. |
| else | else | If the same condition is false, it executes the block of code. |
| else if | elsif | If the prior condition is false, it executes the block of code. |

## Looping Statements

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved Words** | **Description** |
| for | for | It is used to repeat a specific block of code in a known number of times. |
| do | do | It executes the block of code once before the condition is tested. |
| while | while | As long as the given condition is true, it repeatedly executes the target statement. |

## Others

|  |  |  |
| --- | --- | --- |
| **C Language** | **Reserved Words** | **Description** |
| struct | deck | Used for Record Declaration |
| const | locked | Used for Constant Declaration |
| continue | continue | Used to jump to the next iteration |
| break | break | Exits the loop |
| true | true | Boolean value |
| false | false | Boolean value |
| return | return | Returns a certain value or function |
| clrscr | blank | Used to clear the console screen. |

# **RESERVED SYMBOLS**

## Arithmetic Operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| + | Addition | Two values are added |
| - | Subtraction | Two values are subtracted |
| \* | Multiplication | Two values are multiplied |
| / | Division | Two values are divided |
| % | Modulus | Acquired the remainder of the values |

## Relational Operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| == | Equal to | Evaluates values if they are both equal |
| != | Not Equal to | Evaluates values if they are both not equal |
| < | Less than | Evaluates if the left-side value is lower than the right-side value |
| > | Greater than | Evaluates if the left-side value is greater than the right-side value |
| <= | Less than or equals to | Evaluates if the left-side value is less than or equals to the right-side value |
| >= | Greater than or  equals to | Evaluates if the left-side value is greater than or equals to the right-side value |

## Logical Operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| ! | Not | Negates the Boolean Value |
| && | And | Evaluates if both conditions are satisfied then it returns value of 1 else return value of 0 |
| ## | Or | Evaluates at least one of the conditions is satisfied then it returns value of 1 else return value of 0 |

## Assignment Operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| = | Simple assignment operator | Assigns a certain value to a variable |
| += | Add and assignment operator | Adds right operand to the left operand and assign the result to the left operand |
| -= | Subtract and assignment operator | Subtracts right operand from the left operand and assign the result to left operand |
| \*= | Multiply and assignment operator | Multiplies right operand with the left operand and assigns the result to the left operand |
| /= | Divide and assignment operator | Divides right operator with the left operand and assigns the result to the left operand |
| %= | Modulus and assignment operator | Gets the remainder of division of right operand with the left operand and assigns the result to the left operand |

## Unary Operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| ++ | Increment | Increases the value by one |
| - - | Decrement | Decreases the value by one |

## Others

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Description** |
| ; | Semicolon | Terminates the line or expression |
| , | Comma | Separates value |
| . | Period | Accessing struct member |
| ‘ ‘ | Single quote | Defining single character value |
| “ ” | Double quote | Defining string value |
| [ ] | Brackets | Encloses array size |
| { } | Curly Braces | Used to start and end the program; Used to enclose block of codes |
| | | Vertical Bar | Single line comment |
| ( ) | Open and Close Parentheses | To group arithmetic expression |

# **SPECIFIC RULES**

## **DATA TYPES**

A data type is a classification that set outs what a variable or object can contain in computer programming. It identifies what type of mathematical, logical and relational operations that can be done. Variable must be appointed to a specific data type when storing a value.

|  |  |  |
| --- | --- | --- |
| **Data type** | **Counterpart** | **Description** |
| num | int | A type of data that accepts whole numbers. |
| decimal | float | A type of data that accepts a floating-point number. |
| piece | char | A type of data that accepts character. |
| set | string | A type of data that accepts combination of characters. |
| option | boolean | A type of data that returns either true or false. |

## **LITERALS**

Literals are notations for representing a fixed value in source code. Almost all programming languages have notations for atomic values such as integers, floating-point numbers, and strings, and usually for booleans and characters; some also have notations for elements of enumerated types and compound values such as arrays, records, and objects. An anonymous function is a literal for the function type.

**Example:**

num sample = 200

wherein num = data type

sample = identifier

200 = literal

|  |  |  |
| --- | --- | --- |
| **Data type** | **Literal Example** | **Description** |
| num | no = 69 | 69 is the literal of “no” which is a num(int) data type |
| decimal | pi = 3.14 | 3.14 is the literal of “pi” which is a decimal(float) data type |
| piece | cletter = ‘J’ | J is the literal of “cletter” which is a piece(char) data type |
| set | csuit = “JACK” | JACK is the literal of “csuit” which is a set(string) data type |
| option | check = true | true is the literal of “check” which is a option(boolean) data type |

1. **num** – this data type holds whole numbers.

Rules in num Literal:

1. A num literal can have one (*1)* up to ten (*10)* places of value in the source code from -9999999999 to 9999999999, if it exceeds it will be rejected. It can process from -2,147,483,648 to 2,147,483,647 on run time.
2. The tilde symbol (~) is used to indicate a negative value.
3. If the value is containing leading zeroes, the zero/es will be eliminated until it approaches a non-zero digit.

Ex. 005 it returns a value of 5.

|  |  |
| --- | --- |
| **Accepted Example** | **Rejected Example** |
| 123456 | ‘123456’ |
| 9999911111 | 231#@123 |
| ~246 | 2.46 |

2. **decimal** – this data type holds decimal numbers.

Rules in decimal Literal:

1. A decimal literal can have one (*1)* up to ten (*10)* places of value and one *(1)* up to six *(6)* and in the source code from -9999999999.999999 to 9999999999.999999, if it exceeds it will be rejected. It can process from *±5.0 × 10−324 up to ±1.7 × 10308* on run time.
2. The tilde symbol (~) is used to indicate a negative value.
3. The stored value must be a floating-point number.
4. If the value is containing leading zeroes or tailing zeroes, the zero/es will be eliminated until it approaches a non-zero digit.

Ex. 5.1000 it returns a value of 5.1.

0001.2 it returns a value of 1.2.

|  |  |
| --- | --- |
| **Accepted Example** | **Rejected Example** |
| 123.456 | ‘123456’ |
| 9999.91111 | 231#@.123 |
| 232.45 | “14.3” |

1. **piece** – this data type holds a single character.

Rules in piece Literal:

1. A piece literal must be enclosed by a single quotation mark (‘’).
2. A piece literal value can be any of ascii characters except newline, horizontal tab, single quotation. To use these characters, the caret characters must be used.

|  |  |
| --- | --- |
| **Caret Characters** | **Characters Represented** |
| ^n | newline |
| ^t | horizontal tab |
| ^’ | single quote |
| ^” | double quote |
| ^^ | caret |

|  |  |
| --- | --- |
| **Accepted Example** | **Rejected Example** |
| ‘A’ | “A” |
| ‘a’ | ‘ab’ |
| ‘^’’ | ‘’’ |

1. **set** – this data type holds a string of characters.

Rules in set Literal:

1. A string literal must be enclosed by a double quotation mark (“”).
2. A string literal can be a single ASCII character.
3. A string literal value can be any of the ASCII character except newline, horizontal tab, and double quotation. To use these characters, caret characters must be used.

|  |  |
| --- | --- |
| **Accepted Example** | **Rejected Example** |
| “Compiler” | ‘Compiler’ |
| “ANALYSIS” | ANALYSIS |
| “1234” | 123.5 |

1. **Option** – this data type holds “true” and “false” value.

Rules in option Literal:

1. An option literal must be in lower case letters.
2. It only accepts *true or false* value.

|  |  |
| --- | --- |
| **Accepted Example** | **Rejected Example** |
| true | ‘true’ |
| false | “false” |
| true | tuer |

## **IDENTIFIERS**

Identifiers are set of alpha-numeric characters that are mainly used in identifying variables, structures, functions, packages and etc. They must be unique and does not overlaps with the reserved words. Identifiers should be *placed after* the data type.

RULES IN IDENTIFIERS:

1. Identifiers must be in **ALL** lowercase.
2. Identifiers should start with lowercase letter.
3. Identifiers are limited to *only 25 characters*.
4. Identifiers should be an *alpha-numeric character*s and an underscore ( \_ ) is used to *separate* the words.
5. Identifiers should not use reserved words.

|  |  |
| --- | --- |
| **Accepted Examples** | **Rejected Examples** |
| average | AveragE |
| user\_add | user-add |
| res7 | \res |

## **CONSTANTS**

Constants defines a value that cannot be change throughout the execution of the program.

RULES IN DECLARING CONSTANTS:

1. Constants *must be declared globally.*
2. Reserved word *locked* is used in declaring in constants. It should be in small caps.
3. To declare a constant, it must start with the reserved word *locked* followed by the data type then identifier and the assignment operator ( = ).

Example: locked <data type> <identifier> = <value> ;

1. Initialization is *required upon declaring* a constant.
2. Multiple declaration of constant is allowed.
3. Single line of declaration of constants with the same data type can be done by the use of comma ( , ) as a separator.

|  |  |
| --- | --- |
| **Accepted Examples** | **Rejected Examples** |
| locked decimal pi = 3.14; | locked pi = 3.14; |
| locked num a = 1, b =5; | locked num a = 1, b = 2.5; |
| locked piece a = ‘A’; | LOCKED piece b = ‘B’; |

## **VARIABLES**

Variables are used to store values or information in a descriptive name for readability and clarity of the program.

RULES IN DECLARING VARIABLES:

1. Global or local declarations of variables is allowed.
2. Variables must be declared after the data type.
3. In declaring locally, variables can be declared anywhere within the function.
4. In declaring locally, variables must be declared first before implementing it.
5. In declaring globally, variables must be declared before the TABLE () function.
6. Semi-colon ( ; ) is required at the end of declaration of variables.
7. Single line of declaration of variables with the same data type can be done by the use of comma ( , ) as a separator.

**Syntax:**

<data type> <variable> = <value>;

<data type> <variable> = <value>, <variable> = <value>;

**Example:**

num a = 1, b = 2;

float c = 3.5;

piece let = ‘A’;

set sample = “Good day”;

## **ARITHMETIC EXPRESSION**

This expression contains mathematical symbols, numbers, variables that will be used for computation inside the program.

RULES IN USING ARITHMETIC EXPRESSION:

1. The compiler will follow the PEMDAS Rule (Parenthesis, Exponent, Multiplication, Division, Addition, Subtraction). If the operator has the same precedence it will be executed from left to right.
2. Operator should be placed in the middle of two operands.
3. num and decimal data type values are only allowed to get through the arithmetic expression.
4. Operands with different data types is allowed in executing arithmetic expression. If one of the operand is float, the outcome would be float.

**Syntax for Arithmetic Expression:**

<operand> <mathematical operator> <operand>

wherein operand can be = num literal

decimal literal

identifier

arithmetic expression

function calling

|  |  |
| --- | --- |
| **Operator** | **Meaning/Use** |
| + | Addition or unary plus |
| - | Subtraction or unary minus |
| \* | Multiplication |
| / | Division |
| % | Modulo or remainder after division |

**Arithmetic Expression for Different Data Types:**

<num literal> <mathematical operator> <num literal> = num literal

<num literal> <mathematical operator> <decimal literal> = decimal literal

< decimal literal > <mathematical operator> <decimal literal> = decimal literal

wherein mathematical operator can be = + , - , \* , / , %

## **ASSIGNMENT STATEMENT**

Used to assigning or injecting values to variables for the use of computation throughout the program.

Rules in Using Assignment Statement:

1. The value must match with the designated data type.
2. The equal sign symbol (=) acts as an assignment operator to store values.

**Syntax of Assignment Statement:**

<identifier> = <expression>;

wherein expression = literals

identifier

arithmetic expression

function calling

unary statement

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same as** |
| + | a += b | a = a + b |
| - | a -= b | a = a - b |
| \* | a \*= b | a = a \* b |
| / | a /= b | a = a / b |
| % | a %= b | a = a % b |

## **RELATIONAL EXPRESSION**

This expression contains relational operators, numbers, variables that will be used for comparison of two operands to determine which relational relationship they fall.

Rules in Using Relational Expression:

1. Variables that will be used in this expression must be declared first prior to this expression.
2. Relational Expression must be done between two values or identifiers.
3. Identifiers must be written first before the literals
4. The relational operators ‘==’ and ‘!=’ is applicable for all data types.
5. The relational operators ‘>’,‘<’,’>=’,<= is only applicable for num literals and decimal literals.

**Syntax for Relation Expression:**

<expression> <relational operator> <expression>

where expression can be = literals , identifier, function calling

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Return value** |
| **==** | Variable a is equal to variable b | 6 == 6 | 1 |
| 6 == 9 | 0 |
| > | Variable a is greater than variable b | 5 > 2 | 1 |
| 2 > 5 | 0 |
| < | Variable a is less than variable b | 16 < 32 | 1 |
| 64 < 1 | 0 |
| != | Variable a is not equal to variable b | 45 != 54 | 1 |
| 88 != 88 | 0 |
| >= | Variable a is greater than or equal to variable b | 15 >= 15 | 1 |
| 78 >= 146 | 0 |
| <= | Variable a is less than or equal to variable b | 66 <= 245 | 1 |
| 14 <= 98 | 0 |

## **LOGICAL EXPRESSION**

This expression contains two or more expressions that will be evaluated in which logical relationship they will fall.

Rules in Using Logical Expression:

1. This expression will be applied to two or more values or expression
2. Variables that will be used in this expression must be declared first prior to this expression.
3. When using the logical not ‘!’, it must be followed by an open parenthesis.

**Syntax for Logical Expression:**

<expression> <logical expression><expression>

where expression = relational expression

identifier

literals

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Return Value** |
| && | Is True only if all operands are true | ((2 > 1) && (5 != 6)) | 1 |
| ((78 == 1) && (21 < 2)) | 0 |
| ## | Is true only if either one operand is true | (15 <= 15) ## (37 > 66) | 1 |
| ((7 > 43) ## (16 >= 12)) | 1 |
| ! | Logical ‘NOT’  Is true only if the operand is 0 | If a = 3  !(a == 2) | 1 |
| !(a == 3 ) | 0 |

## **STRING CONCATENATION**

Used to compound literals, variables, deck items (struct items), constants and value of functions that has set data type (string).

Rules in Using string concatenation:

1. The plus sign (+) is used to concatenate variables, literals, and constants that stores set literals.

Ex. set a = “Hello”;

set b = “World”;

a = a + b; | “HelloWorld”

1. String concatenation may only find in declarations and assignment statements.
2. Appending piece literal(char) to another piece literal or string literal and storing it in set data type storage is not allowed.

|  |  |
| --- | --- |
| Accepted Examples | Rejected Examples |
| a = “Hello” + “World” | a = “Hello” + ‘a’ |
| b = b + “World” | b = ‘a’ + ‘b’ |

## **UNARY OPERATORS**

Unary Operators are used to produce a new value by acting upon a single operand.

Types of Unary Operators:

1. **Increment -** It is used to increment the value of the variable by 1.

The increment can be done in two ways:

1. **Prefix Increment** - The operator precedes the operand (ex. ++a). The value of operand will be altered before it is used.
2. **Postfix Increment -** The operator follows the operand (ex. a++). The value operand will be altered after it is used.
3. **Decrement -** It is used to decrement the value of the variable by 1.

The decrement can be done in two ways:

1. **Prefix Decrement** - The operator precedes the operand (ex. --a). The value of operand will be altered before it is used.
2. **Postfix Decrement -** The operator follows the operand (ex. a--). The value operand will be altered after it is used.

Rules in INCREMENT/DECREMENT:

1. If the increment/decrement ends with a semicolon, it will be considered as a statement. Moreover, increment/decrement statements can only be found in declarations, conditional, looping, and assignment statements.
2. num (int) literal that is stored in variables are the only allowed operands to use in unary operation.
3. In using unary operation in mathematical expressions, a space must be between the assignment, arithmetic and the unary operation, regardless if the unary operator is either prefix or postfix.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Accepted Example** | **Rejected Example** |
| ++ | a++; | a(++); |
| b = x++ +3; | b = x++\*3; |
| sum += ++s; | sum += ++s/2; |
| -- | --z; | (--)z; |
| y= --a -1; | y= a---1; |
| x -=y-- +z; | x -=y--+z; |

## **OPERATORS PRECEDENCE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator** | **Description** | **Associativity** |
| 1 | ++  --  ( )  [ ] | Suffix Increment  Suffix Decrement  Function  Array subscripting | Left to Right |
| 2 | ++  --  ! | Prefix Increment  Prefix Decrement  Logical NOT | Right to Left |
| 3 | \*  /  % | Multiplication  Division  Modulus | Left to Right |
| 4 | +  - | Addition  Subtraction | Left to Right |
| 5 | >  <  >=  <= | Greater than  Less than  Greater than or equal  Less than or equal | Left to Right |
| 6 | ==  != | Equal  Not Equal | Left to Right |
| 7 | && | Logical AND | Left to Right |
| 8 | ## | Logical OR | Left to Right |
| 9 | =  +=  -=  \*=  /= | Simple assignment  Add & assignment  Subtract & assignment  Multiply & assignment  Divide & assignment | Right to Left |

## **COMMENTS**

It helps to understand the purpose of a certain function not only by the who made the program but also to those people who wants to take a look on how the system works.

Rules in Making comments:

1. In a one liner comment, the reserved symbol “ | ” must be used.
2. In one liner comment, once vertical bar symbol is used, all succeeding characters will be treated as comment.
3. Putting comment within expressions will be rejected.

|  |  |
| --- | --- |
| **Accepted Examples** | **Rejected Examples** |
| |Example of comment | Example of comment^^ |

## **INPUT AND OUTPUT STATEMENTS**

1. **Input Statement -** Reserved word **check** is for accepting values. It indicates input statements.

Rules in Using Input Statement:

1. The reserved word **check** must be placed first before the identifier then followed by enclosed parenthesis.
2. One **check** per input statements; multiple inputs mean multiple declaration of **check.**
3. Values can be stored in variables, array element, member of the deck.

**Syntax for input statement:**

check(<variable>);

check (<array element>);

check(<member of the deck>);

**Example:**

check(b);

check(sample[0]);

check (stud.name);

1. **Output Statement**

Reserved word **tell** is for displaying values or strings. It indicates output statements.

Rules in Using Output Statement:

1. The reserved word *tell* used enclosed parenthesis to display something.
2. When displaying strings, after the open parentheses it must be enclosed with (“ ”).
3. Identifiers can be also displayed in which variables, constants, member of the structure, and array element can only be displayed.
4. If you want to display identifiers together with strings a comma ( , ) is used to separate them.
5. When using identifier, it must be declared first before the reserved word *tell*.
6. In using operators within the output statement, the expression must be enclosed with a parenthesis.
7. Printing directly of literals aside from string is not allowed.

**Rejected Example**:

tell(<literal>)

wherein literal = num

decimal

piece

option

**Syntax 1:**

tell(“<string>”);

**Valid example:**

tell(“Hello world!”)

**Syntax 2:**

tell(identifiers);

**Valid Example:**

tell(sample); wherein *sample* is a variable

**Syntax 3:**

tell(“<string>”, <identifier>);

**Valid Example:**

tell("hello world ", sample); wherein *sample* is a variable

**Syntax 4:**

tell(“<string>”, (identifier + identifier));

**Valid Example:**

tell ("hello ", (a+b)); wherein *a* and *b* is a variable

**Syntax 5:**

tell("<string>", <structure member>);

**Valid Example:**

tell("Book 1 author: ", Book1.author);

**Syntax 6:**

tell(<variable[array element]>);

**Valid Example:**

tell(a[1]); = where in a[5]= {1, 2, 3, 4, 5};

## **CONDITIONAL STATEMENT**

1. **if Statement -** Used to execute a statement if the condition is achieved.

Rules in Using if Statement:

1. Statements under if statement must be enclosed with curly braces.
2. Relational expressions are a must in this statement: also, Logical Expression can be used in this statement.
3. Variables that are being used in condition of the if statement must be declared first.
4. If the condition did not meet, the statements under the if statement will be ignored.

**Syntax of if statement:**

if (<condition>)

{

<statement/s>;

}

condition = relational expressions

logical expressions

literals

identifier

1. **else Statement -** if the condition in the if statement did not achieved, the statements under else statement will be executed.

Rules in Using else Statement:

1. Statements under if and else statement must be enclosed with curly braces.
2. The else statement cannot be declared if there is no if statement prior to it.
3. If the condition in the **if statement** is not satisfied, statements under **else statement** will be executed. Otherwise, if the condition in the **else statement** is not achieved, statements under that will be ignored.

**Syntax of else statement:**

if (<condition>)

{

<statement/s>;

}

else

{

<statement/s>;

}

condition = relational expressions, logical expressions, literals, identifiers

1. **elsif Statement -** it is used when the one or more conditions are needed to be tested.

Rules in Using elsif Statement:

1. Statements under **if, elsif** and **else statement** must be enclosed with curly braces.
2. Variables that are being used in the condition of if and elsif statement must be declared first.
3. If the condition in the **if statement** is not satisfied, **elsif statement** is the next conditional statement to be evaluated if it didn’t meet it will proceed to other **elsif statements.** This will continue until the condition is satisfied. If none of the conditional statements are satisfied no statements will be executed.

**Syntax of elsif Statement:**

if (<condition>)

{

<statement/s>;

}

elsif (<condition2>)

{

<statement/s>;

}

else

{

<statement/s>;

}

wherein condition2 = relational expression, logical expression, literals,

identifier

## **LOOPING STATEMENT**

1. **for Statement -** used to efficiently repeat a block of statement in a specific number of times.

Rules in Using for Statement:

1. **for statement** can only be used in TABLE function and user-defined function.
2. **for statement** consist of initialization, condition, and count expression. It is enclosed with parentheses.
3. Statements under for statement should be enclose with curly braces.
4. In **initialization,** it is an optional to include howeverthe variable must be declared first before the initialization part of the for statement.

**Example:**

num a;

for(a = 0; a < 5; a++){

<statement/s>

}

Where <statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

1. In **initialization**, the syntax is: *<variable> = <value>;* wherein the value is a num literal or a num data type variable.
2. In **condition**, the syntax is: *<variable> <relational operator> <value>*
3. Arithmetic expression can happen after the equal symbols ( = ) for initialization and condition.
4. In **count expression,** it can be either increment or decrement.
5. The **condition**, and **count expression** must be present in using the for statement. Otherwise, it will result to an error.
6. The statements under for loop statement will repeat until the condition becomes false.

**Syntax of for Statement:**

for (<initialization>; <condition>; <count expression>)

{

<statement/s>;

}

Where <statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

1. **while Statement -** statements under this loop will be repeated as long as the condition is satisfied.

Rules in Using while Statement:

1. The condition in while statement must be enclosed with parentheses.
2. Variables that are being used in the while statement must be declared first before the condition.
3. Statement under this loop statement must be enclose with curly braces.
4. The condition in while statement will be evaluated first, if not satisfied statement under this loop statement will not be executed.

**Syntax of for while Statement:**

while (<condition>)

{

<statement/s>;

}

Where <statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

1. **do – while Statement -** in this statement, it will be executed once then it evaluates the condition at the end of execution of statements. It will repeat the blocks of statements under this until the condition will not meet.

Rules in Using do-while Statement:

1. The condition at the end must be enclosed with parentheses.
2. Variables that are being used in the do-while statement must be declared first before the looping statement.
3. Statement/s under this loop statement must be enclosed with curly braces.
4. Before it reaches to the condition, **the statements will be executed once** then evaluation of condition statement. It will continue looping as long as it satisfies the condition.

**Syntax of for while Statement:**

do{

<statement/s>;

} while (<condition>);

Where <statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

## **CONTROL STATEMENTS**

### **Break**

It is a loop control statement which is used to terminate the loop. When it is encountered, the loop iterations immediately stop and the program control resumes at the next statement following the loop.

RULES IN USING THE break STATEMENT:

1. Thebreak statementcan be found and used within the conditional statement as long as it is inside the looping statement.

**Syntax:**

break;

**Example:**

for(i = 0; i<n; i+=) {

if(a > 10) {

break;

}

tell(“hello world”);

}

### **Continue**

This statement is found inside looping statements, wherein it is used for jumping into the next iteration and skipping the execution of succeeding statements inside the body loop in the current iteration.

RULES IN USING THE continue STATEMENT:

1. The continue statementcan be found and used within the conditional statement as long as it is inside the looping statement.

**Syntax:**

continue;

**Example:**

while (a < 10) {

tell(“Hello”);

if(a == 5) {

continue;

}

tell(“World”);

## **ARRAYS**

An array is a data structure which can store a fixed-size collection of the same elements with the same data type. It refers to as variable that can store multiple values.

Rules in Array Declaration:

1. Arrays can be declared locally or globally.
2. Declaring an array is the same as the rules in declaring variables except that the variable has a size included enclosed with a square bracket. ( [ ] ).
3. Declaring multiple array with the same data type in a single line is not allowed.
4. Two is the maximum range of array dimension.
5. There are ways in declaring an array size:
   1. The size of the array could be a positive integer value.

**Example:**

num sample[5];

* 1. The size of the array could be a variable that consists of num data type value.

**Example:**

num x = 5;

num sample[x];

**Syntax:**

One-dimensional array:

*<data type> <variable> <[array size]>;*

Two-dimensional array:

*<data type> <variable> <[array size 1]> <[array size 2]>;*

Rules in Array INITIALIZATIOn:

1. If the size of the array is not fully used, then the following cases may happen:
2. If the data type is num and decimal, it will be considered 0 as the element value.

**Example:**

num sample[2] = (1);

Array Elements: sample[0] = 1

sample[1] = 0

1. If the data type is piece, it will be considered empty character (‘ ‘).

**Example:**

piece sample[3] = (‘A’, ‘B’);

Array Elements: sample[0] = A

sample[1] = B

sample[2] = ‘ ‘

1. If the data type is set, it will be considered empty string (“ “).

**Example:**

set sample[3] = (“Apple”, “Banana”);

Array Elements: sample[0] = Apple

sample[1] = Banana

sample[2] = “ “

1. The Equal symbol ( = ) is used to assign the element values to the array.
2. For one-dimensional array, the elements should be enclosed with a parenthesis () and each array element should be separated by a comma ( , ).

Ex. num sample[5] = (1, 2, 3);

1. In two-dimensional array, each group of array elements should be enclose by a parenthesis () and must be separated by a comma ( , ).

Ex. num sample[2][3] = ((1,2), (3,4), (5,6))

**Syntax:**

One-dimensional array:

*<data type> <variable> <[array size]> = (<elements>);*

Two-dimensional array:

*<data type> <variable> <[array size 1]> <[array size 2]> = ((<elements>),( <elements>));*

HOW TO ACCESS ARRAY:

1. To access the array element, it must start with the array variable followed by square brackets containing the index element of the array.
2. The array index could be a num literal.
3. The index element can be a variable but must be num data type only.
4. Accessing the array can be done in printing, scanning, and looping.

PRINTING CERTAIN ELEMENT OF AN ARRAY USING INDEX:

**Syntax:**

tell(<variable>[array index]);

**Example:**

TABLE();

{

decimal grades[5] = (1.5 , 2.0 , 3.0 , 5.0 , 1.0);

tell(grades[0]);

tell(grades[1]);

PRINTING CERTAIN ELEMENT OF AN ARRAY USING VARIABLE:

**Syntax:**

tell(<variable> [<variable>])

1. In for loop:

TABLE();

{

decimal grades[5] = (1.5 , 2.0 , 3.0, 5.0 , 1.0);

num i;

for (i=0; i<5;i++)

{

tell(grades[i]);

}

}

1. In while loop:

TABLE();

{

decimal grades[5] = (1.5 , 2.0 , 3.0 , 5.0 , 1.0);

num i;

while(i < 5)

{

tell(grades[i]);

}

}

1. In do while loop:

TABLE();

{

decimal grades[5] = (1.5 , 2.0 , 3.0, 5.0 , 1.0);

num i;

do

{

tell(grades[i]);

i++;

}while(i < 5);

## **deck (struct)**

*deck* holds a collection of data elements under a single name. These data elements, known as deck items can have different data types. decks have deck objects, which are the occurrences of the *deck*.

Rules in Struct Declaration:

1. deck must be declared using the reserved word deck, followed by the user-defined deck tag.
2. All decks must have at least one deck item.
3. deck item/s of the deck must be enclosed in open and close curly braces ( { } ).
4. The closing curly brace must be followed by a deck object. All structures must have at least one deck object.
5. Naming a deck tag, deck item, and deck object follow the rules in naming an identifier.
6. deck items which are variables within a deck, must be declared writing the data type first, followed by the identifier.
7. Multiple declarations of deck items in a single statement is not allowed, even if these deck items have the same data type.
8. deck item/s initialization is not allowed. However, one a deck item is declared, it will have the following value depending on its data type:

|  |  |
| --- | --- |
| **DATA TYPE** | **VALUE** |
| num | 0 |
| decimal | 0.00 |
| piece | ‘ ’(empty character) |
| set | “ ”(empty string) |
| option | false |

1. Arrays of deck and arrays within a deck is possible in J4CK. A deck object maybe a one dimensional or two dimensional. Additionally, each deck item in a deck may also be a one-dimensional array o two-dimensional array.
2. For struct items that contains set literals, you can access each character by writing square brackets after the struct item identifier, and inside is the index of the particular character.

**Syntax:**

deck <deck tag> {

<data type> <deck item1>;

<data type> <deck item2>;

**.**

**.**

**.**

<data type> <deck itemn>;

} <deck object1><deck object2> … <deck objectn>;

**Example:**

deck students {

num age;

dec gwa;

set sec[10];

} student1,student2;

num TABLE()

{

return 0;

}

Rules in deck Definition and deck item accessing:

1. A deck item must be accessed only inside the TABLE function or any other user-defined functions.
2. The deck object, followed by a period, then the deck item must be written to access a struct item. There must be no spaces between the period, and the deck item.
3. In case of array of deck objects, the index enclosed with square brackets must be immediately followed by the identifier. The same rule applies for array of deck items.

**Syntax:**

<deck object>.<deck item>

<deck object>[<size>].<deck item>[<size>]

**Example:**

student1.age = 21;

student2.gwa = 1.0;

## **FUNCTIONS**

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Rules in function declaration:

1. Functions should always be declared either with a data type or void.
2. Rules for creating identifiers are also aligned with the naming of functions.
3. The body of non void functions should not be empty or contain a statement or a block of statements. However, in void functions, it could have an empty statement.
4. Other functions should be placed before the TABLE () function.
5. A function can be classified as a *void function* wherein it does not return a value when the function is executed.
6. A function can be classified as a *non-void function* wherein it returns a value when the function is executed.
7. *Non-void function* can contain any of the data type such as num, decimal, piece, set and option.
8. To put a parameter in functions, the data type of the variable must be indicated. Use comma ( , ) to separate multiple variables.
9. Statement in the functions can contain printing, scanning, declaration, initializations, conditional, looping, loop control, any expressions, return statement (non – void) and function calling and blank statement.
10. Non void functions have a return statement. The reserved word return is used to return values, variable, or expression that has the same return data type of the function.

**Syntax: (VOID FUNCTION)**

void <function name> ()

{

<statement/s>;

}

Where <statement/s> 🡪 <variable declaration/s>

<input or output statement/s>

<conditional statement/s>

<looping statement/s>

<blank statement/s>

<function calling/s>

<assignment statement/s>

**Example:**

Ex.

void greet()

num a = 1;

{

tell(“The answer is: “, a);

a += 1;

}

**Syntax: (NON- VOID FUNCTION)**

<data type> <function name> (<parameters>, <parameters>)

{

<statement/s>;

return <variable>;

}

**Example:**

num add(num a, num b)

{

return a+b;

}

Rules in calling a function:

1. Calling the function itself is allowed.
2. In calling *void function,* the identifier of the function is followed by enclosed parenthesis.
3. In calling *non-void function,* the order, data type and number of the parameters of the function must be equal to the order, data type and number of the arguments passed. The argument can be variable, array element, or struct member.
4. The arguments can be literal, array element, variable, constant, struct member, math expression or function that will return a value.
5. The comma symbol (,) is used to separate multiple arguments to be passed.
6. The function can be called within the TABLE function.
7. The function can call itself, also known as recursive function

**Syntax:**

**Non-void:**

<identifier>(<parameter>, <parameter>);

**Void:**

<identifier> ();

## **BLANK STATEMENT**

A blank statement clears the output screen or console. It moves the cursor to the upper left-hand corner of the screen.

RULES IN USING THE BLANK STATEMENT:

1. The blank statement can be used within user-defined function or within the TABLE function. Also, it can be used within statements such as conditional and looping.

**Syntax:**

blank;

**Example:**

TABLE()

{

tell(“Hello”);

blank;

}

# **REGULAR EXPRESSION**

## Reserved words

|  |  |  |  |
| --- | --- | --- | --- |
| **C Language** | **Proposed Language** | **Regular Expression** | **Token** |
| main | TABLE | (T)(A)(B)(L)(E) | TABLE |
| int | num | (n)(u)(m) | num |
| float | decimal | (d)(e)(c)(i)(m)(a)(l) | decimal |
| char | piece | (p)(i)(e)(c)(e) | piece |
| string | set | (s)(e)(t) | set |
| boolean | option | (o)(p)(t)(i)(o)(n) | option |
| void | void | (v)(o)(i)(d) | void |
| printf | tell | (t)(e)(l)(l) | tell |
| scanf | check | (c)(h)(e)(c)(k) | check |
| if | if | (i)(f) | if |
| else | else | (e)(l)(s)(e) | else |
| else if | elsif | (e)(l)(s)(i)(f) | elsif |
| for | for | (f)(o)(r) | for |
| do | do | (d)(o) | do |
| while | while | (w)(h)(i)(l)(e) | while |
| struct | deck | (d)(e)(c)(k) | deck |
| const | locked | (l)(o)(c)(k)(e)(d) | locked |
| continue | continue | (c)(o)(n)(t)(i)(n)(u)(e) | continue |
| break | break | (b)(r)(e)(a)(k) | break |
| true | true | (t)(r)(u)(e) | true |
| false | false | (f)(a)(l)(s)(e) | false |
| return | return | (r)(e)(t)(u)(r)(n) | return |
| clrscr | blank | (b)(l)(a)(n)(k) | blank |

## Reserved symbols

|  |  |  |
| --- | --- | --- |
| **Reserved Symbol** | **Regular Expression** | **Token** |
| + | (+) | + |
| - | (-) | - |
| \* | (\*) | \* |
| / | (/) | / |
| % | (%) | % |
| == | (=)(=) | == |
| != | (!)(=) | != |
| < | (<) | < |
| > | (>) | > |
| <= | (<)(=) | <= |
| >= | (>)(=) | >= |
| ! | (!) | ! |
| && | (&)(&) | && |
| ## | (#)(#) | ## |
| = | (=) | = |
| += | (+)(=) | += |
| -= | (-)(=) | -= |
| \*= | (\*)(=) | \*= |
| /= | (/)(=) | /= |
| %= | (%)(=) | %= |
| ++ | (+)(+) | ++ |
| - - | (-)(-) | - - |
| ; | (;) | ; |
| , | (,) | , |
| . | (.) | . |
| ‘ ’ | (‘)(‘) | ‘ ‘ |
| “ ” | (“)(”) | “ ” |
| [ ] | ([)(]) | [ ] |
| { } | ({)(}) | { } |
| ( ) | (( ) ( )) | ( ) |

## Literals and Comment

|  |  |  |
| --- | --- | --- |
| **Literals and Comment** | **Regular Expression** | **Token** |
| num literal | (~/λ)(num)(num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) | num literal |
| decimal literal | (~/λ)(num)(num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) (.)(num) (num/λ) (num/λ) (num/λ) (num/λ) (num/λ) | decimal literal |
| piece literal | (‘)(ascii1)(‘) | piece literal |
| set literal | (“)(ascii2)(“) | set literal |
| option literal | (t)(r)(u)(e) | option literal |
| (f)(a)(l)(s)(e) |
| Single line comment | (|) | | |
| identifier  identifier | (lowercase)(lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ)  (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) (lowercase / numbers /\_ / λ) | identifier  identifier |

# **REGULAR DEFINITION**

## Boundary

|  |  |
| --- | --- |
| 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} |
| 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} |
| nonzero | {1,2,3,4,5,6,7,8,9} |
| numbers | {0,nonzero} |
| mathoptr | {+,-,\*,/,%} |
| reloptr | {<, >, =} |
| logoptr | {&, #} |
| ascii | {any printable characters} |
| ascii\_p | {any printable characters (except ‘, ^, newline, horizontal tab)} |
| ascii\_s | {any printable characters (except “, ^, newline, horizontal tab )} |

## Reserved Words

|  |  |
| --- | --- |
| whitespace | {space, newline} |
| bound1 | {whitespace, semi-colon} |
| bound2 | {whitespace, (} |
| bound3 | {whitespace, { } |
| bound4 | {bound1, comma, ) , logoptr} |

## Reserved Symbols

|  |  |
| --- | --- |
| bound5 | {bound2, lowercase, numbers, ~ } |
| bound6 | {bound1, lowercase, ), numbers} |
| bound7 | {bound5, “, ‘ } |
| bound8 | {bound4, mathoptr, reloptr } |
| bound9 | {bound7, !, ) } |
| bound10 | {bound4, mathoptr, reloptr, !, [, .} |
| bound11 | {whitespace, lowercase, numbers, ~} |
| bound12 | {whitespace, lowercase,} } |
| bound13 | {bound2, lowercase, } } |
| bound14 | {bound4, !, = } |
| bound15 | {bound4, reloptr, mathoptr } |
| bound16 | {bound7, !, +, - } |
| bound17 | {bound5, “} |

## Identifiers

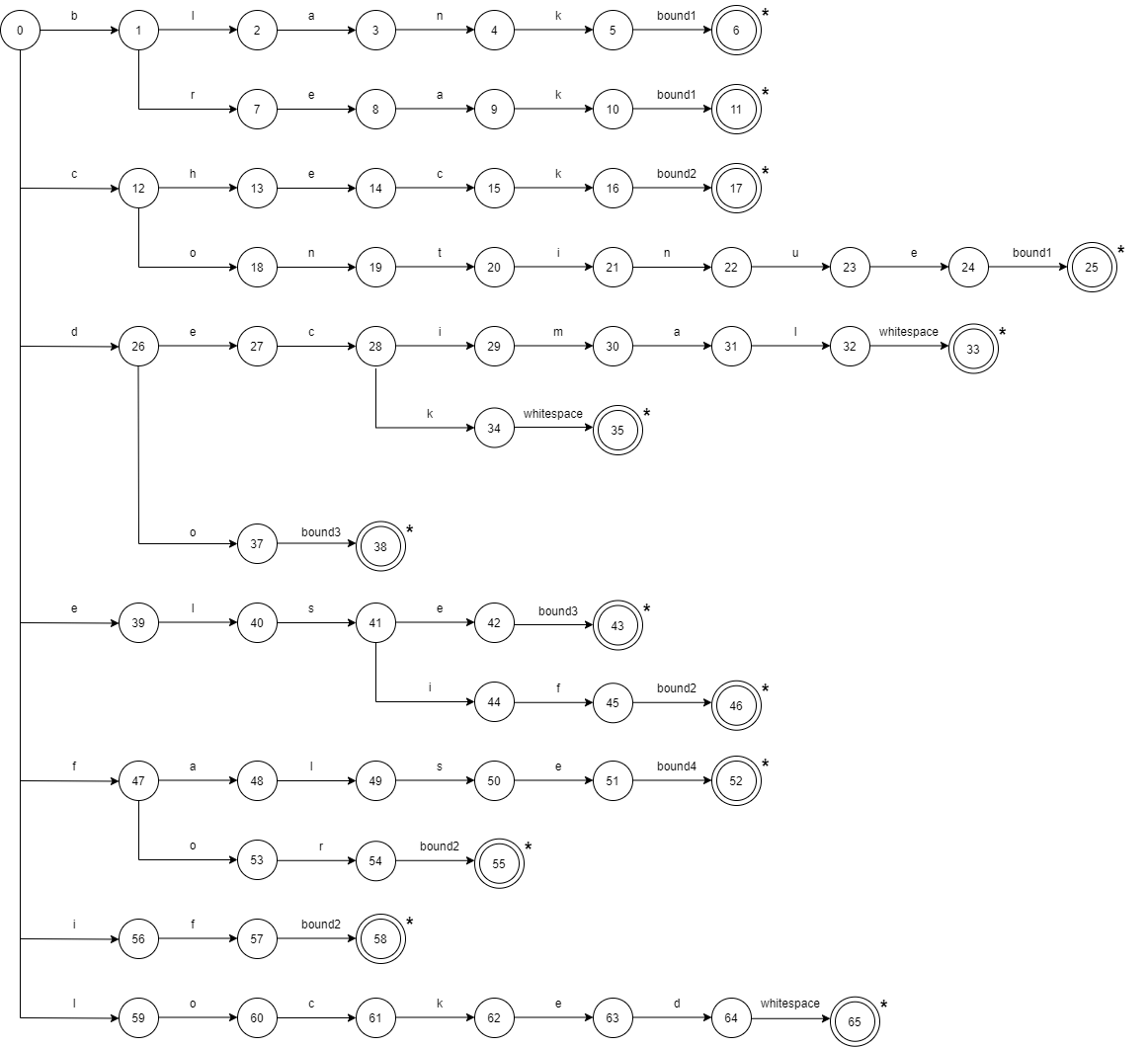
|  |  |
| --- | --- |
| bound\_id | {bound4, mathoptr, reloptr, [, ], (, . , { , !} |
| def | {lowercase, numbers, \_, λ } |

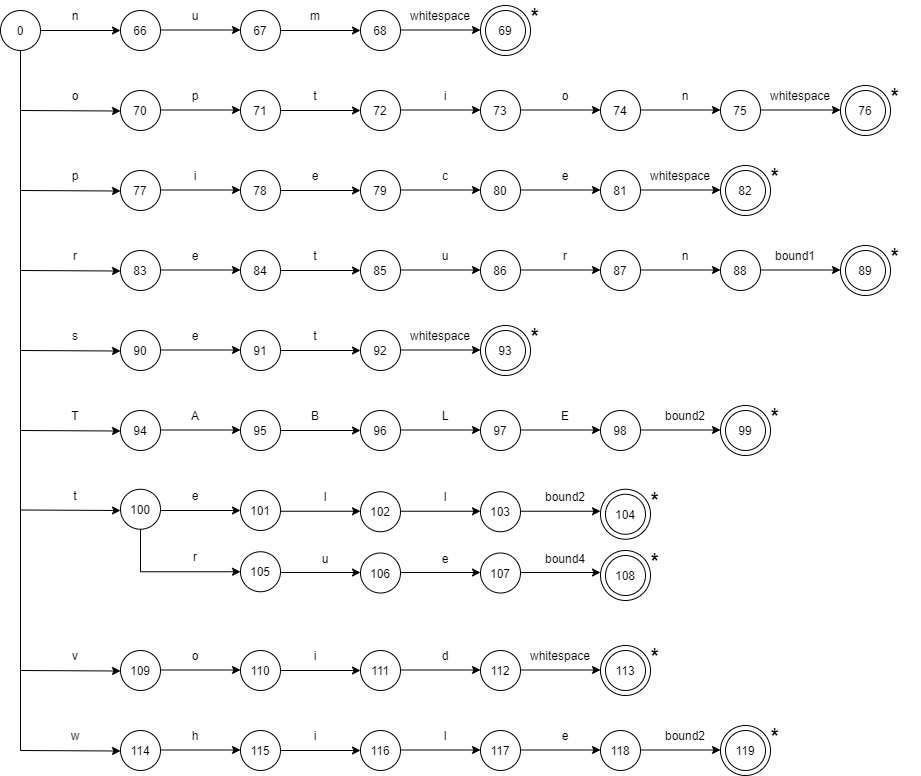
## Literals

|  |  |
| --- | --- |
| bound\_dec | {bound15 } |
| bound\_num | {bound15, ] } |
| bound\_set | {bound 14, +} |

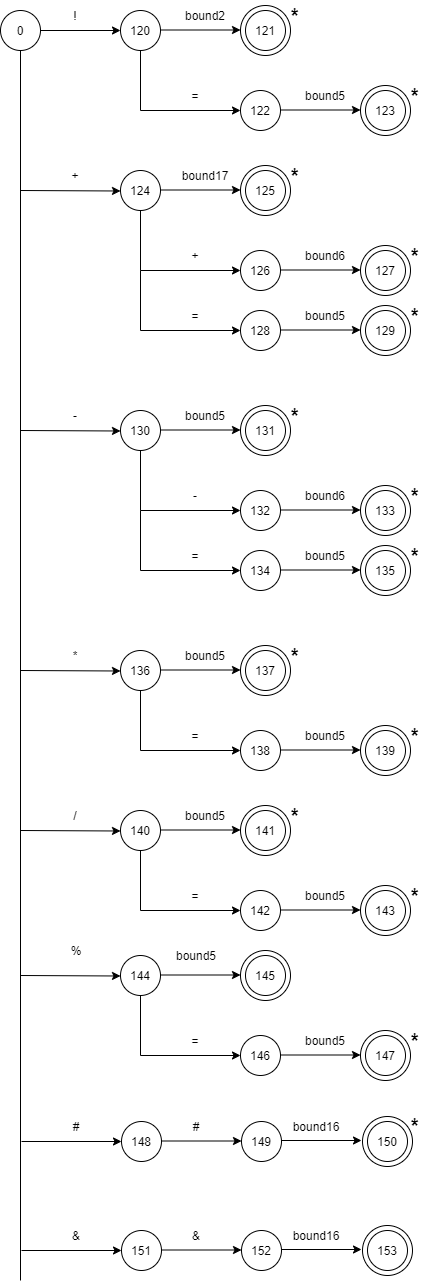
# **TRANSITION DIAGRAM**

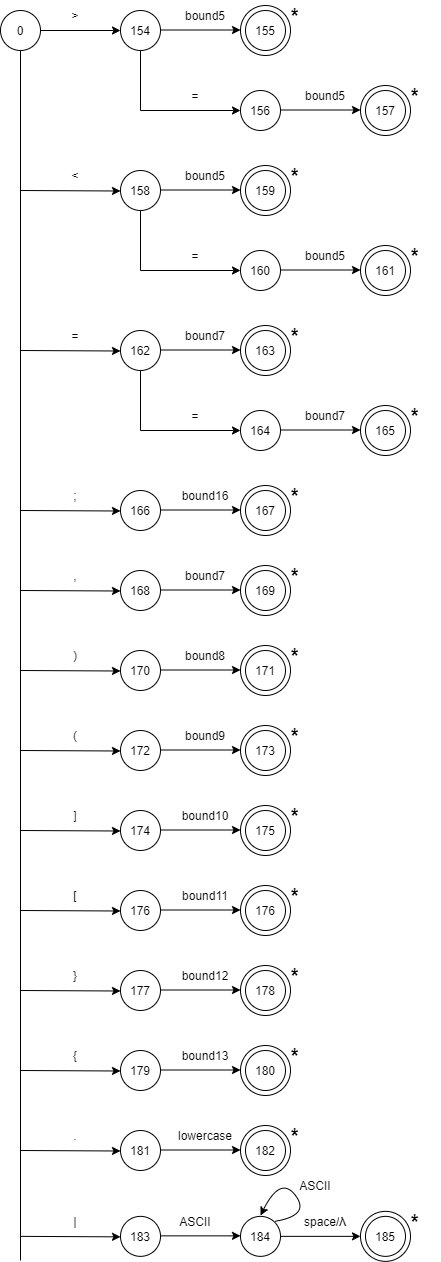
## **Reserved Words**

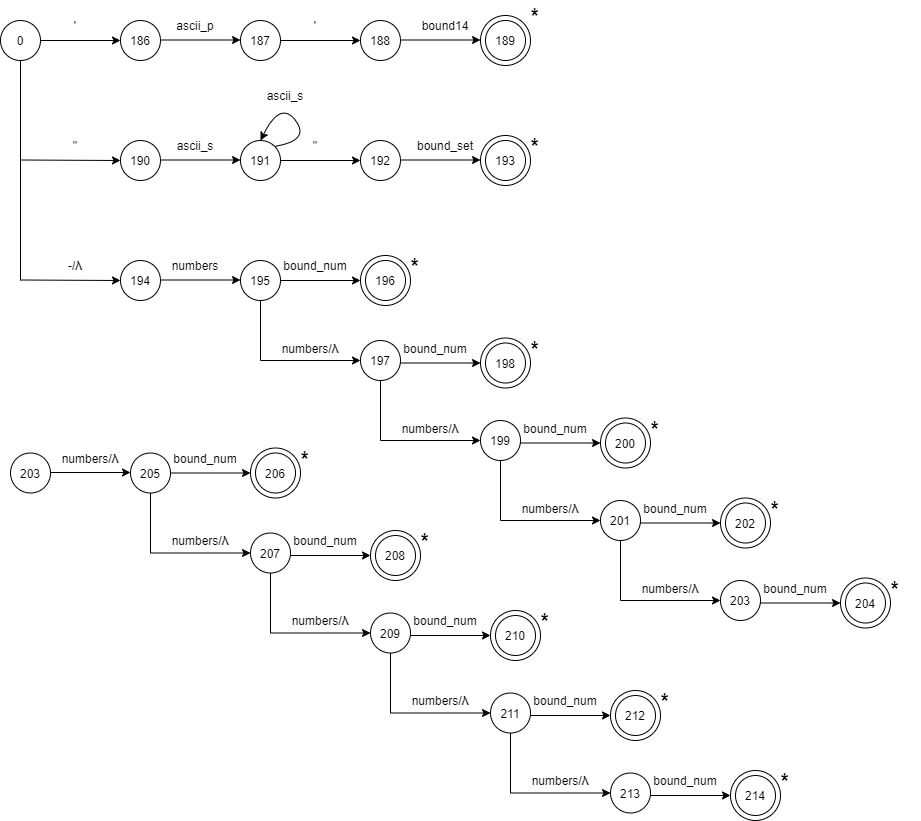




## **Reserved Symbols**



** Comments**

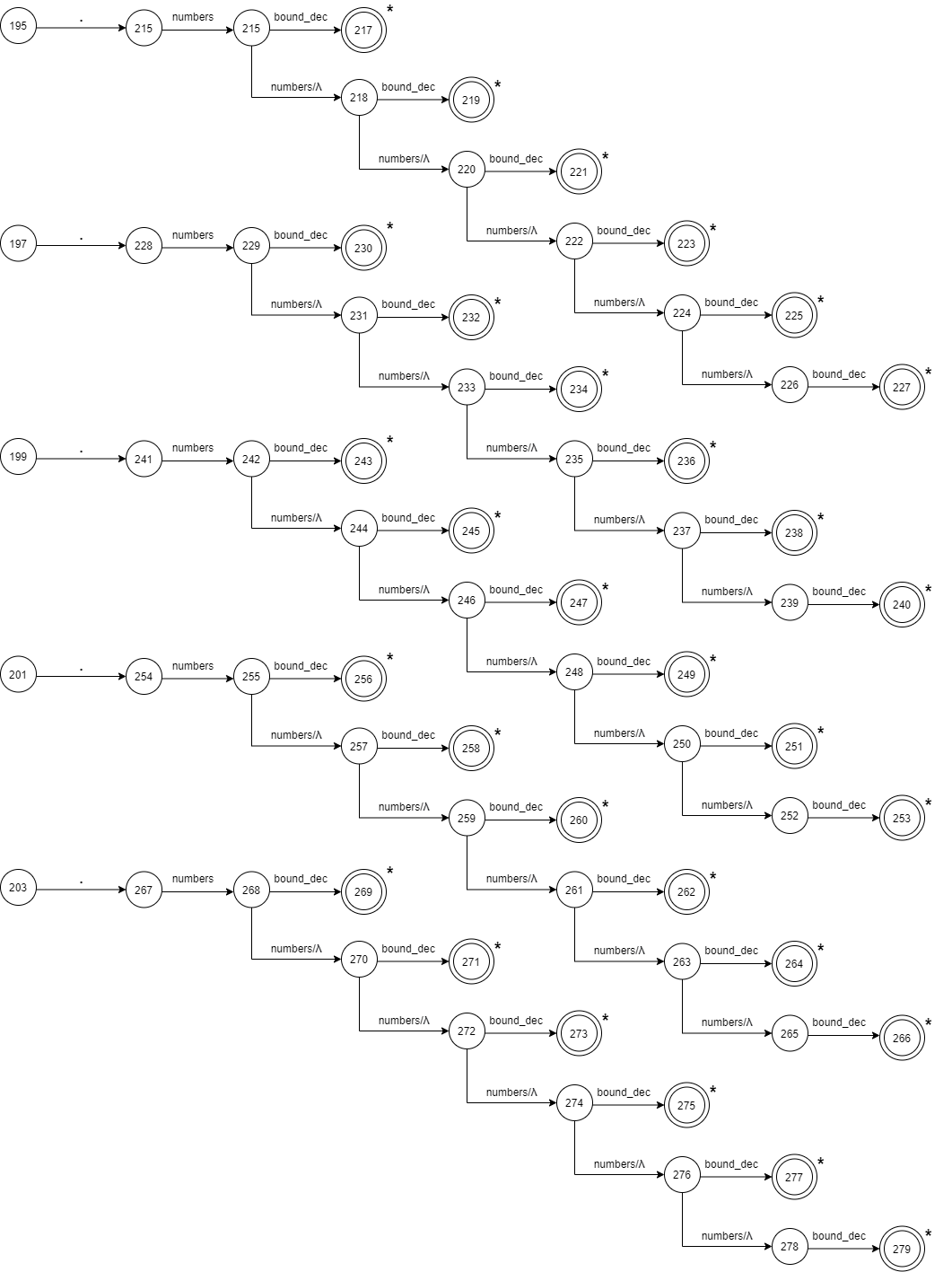


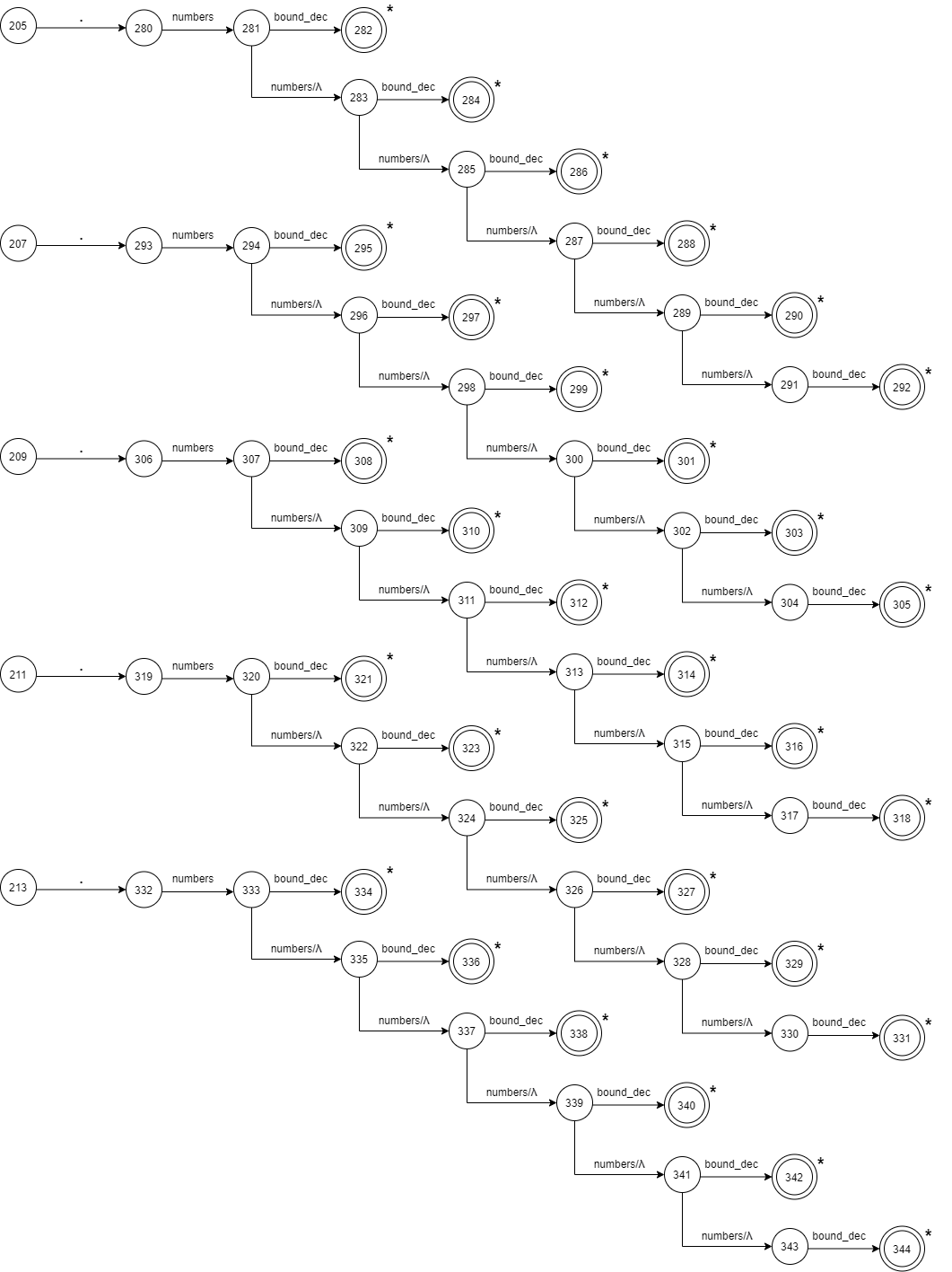
## **PIECELIT**

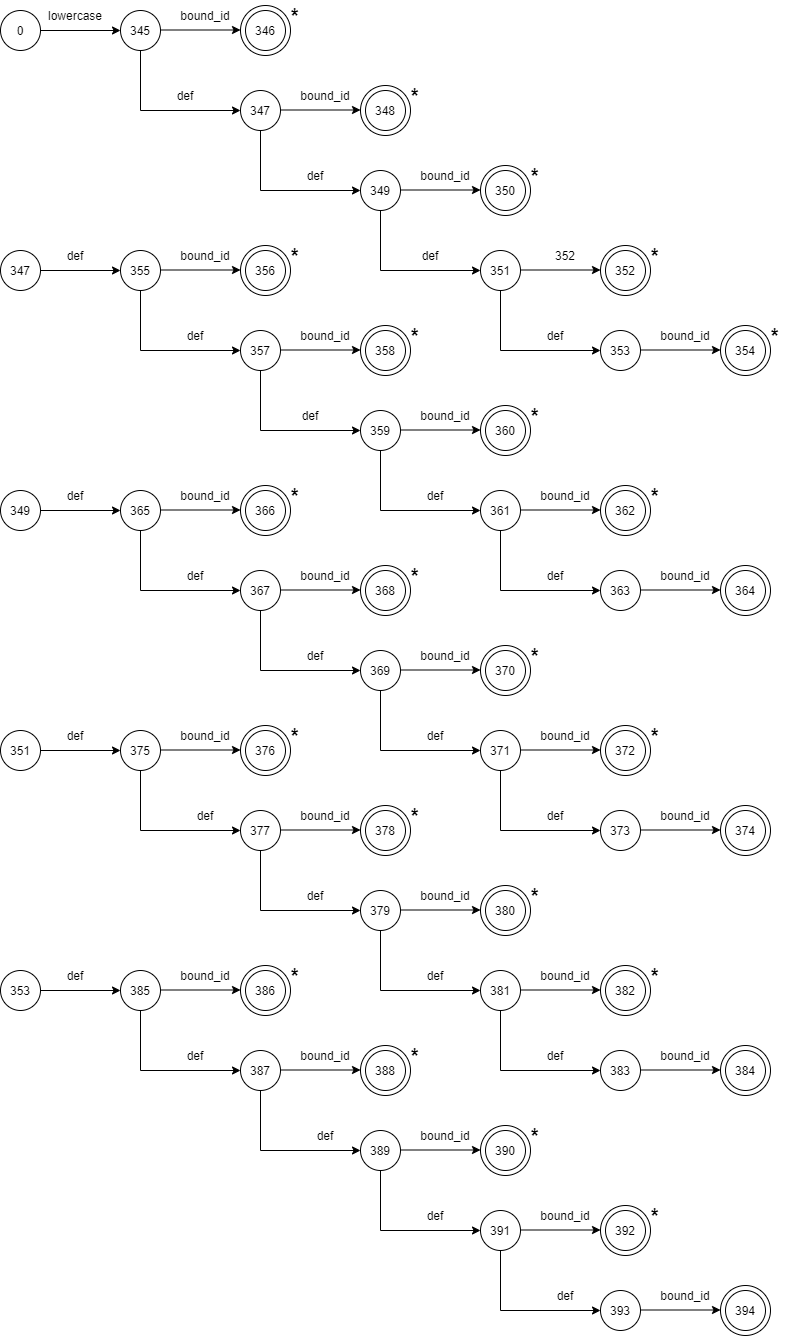
## **SETLIT**

## **NUMLIT**

## **DECIMALLIT**







## **IDENTIFIERS**

# **CONTEXT FREE GRAMMAR**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROGRAM** | | **→** | **PRODUCTION SET** |
| 1 | <program> | → | <global> TABLE() { <statements> } |
| 2 | <global> | → | <const\_dec> <global> |
| 3 | <global> | → | <data\_id> <global> |
| 4 | <global> | → | <deck\_dec> <global> |
| 5 | <global> | → | <void> <global> |
| 6 | <global> | → | λ |
| 7 | <const\_dec> | → | locked <const\_dec1>; |
| 8 | <const\_dec1> | → | <datatype> identifier = <literals> <const\_dec2> |
| 9 | <const\_dec2> | → | ,identifier = <literals> <const\_dec2> |
| 10 | <const\_dec2> | → | λ |
| 11 | <datatype> | → | num |
| 12 | <datatype> | → | decimal |
| 13 | <datatype> | → | piece |
| 14 | <datatype> | → | set |
| 15 | <datatype> | → | option |
| 16 | <data\_id> | → | <datatype> identifier <data\_cont> |
| 17 | <data\_cont> | → | <data\_altr> <id\_add>; |
| 18 | <data\_cont> | → | (<param>) {<func\_stmt><return1>} |
| 19 | <data\_altr> | → | <var\_dec> |
| 20 | <data\_altr> | → | <1d\_arr> |
| 21 | <data\_altr> | → | λ |
| 22 | <id\_add> | → | ,identifier <data\_altr> <id\_add> |
| 23 | <id\_add> | → | λ |
| 24 | <var\_dec> | → | =<literals> |
| 25 | <var\_dec> | → | λ |
| 26 | <literals> | → | numlit |
| 27 | <literals> | → | decimallit |
| 28 | <literals> | → | piecelit |
| 29 | <literals> | → | setlit |
| 30 | <literals> | → | optionlit |
| 31 | <1d\_arr> | → | [ <array\_altr> ] <2d\_arr> |
| 32 | <1d\_arr> | → | λ |
| 33 | <array\_altr> | → | numlit <array\_altr\_expr> |
| 34 | <array\_altr> | → | identifier <array\_altr\_expr> |
| 35 | <2d\_arr> | → | <1d\_arr\_init> |
| 36 | <2d\_arr> | → | [<array\_altr>] <2d\_arr\_init> |
| 37 | <2d\_arr> | → | λ |
| 38 | <1d\_arr\_init> | → | =(<arr\_elem>) |
| 39 | <arr\_elem> | → | <arr\_lit> <arr\_elem1> |
| 40 | <arr\_elem1> | → | , <arr\_elem> |
| 41 | <arr\_elem1> | → | λ |
| 42 | <arr\_lit> | → | numlit |
| 43 | <arr\_lit> | → | decimallit |
| 44 | <arr\_lit> | → | piecelit |
| 45 | <arr\_lit> | → | setlit |
| 46 | <2d\_arr\_init> | → | =(<2d\_arr\_init1>) |
| 47 | <2d\_arr\_init> | → | λ |
| 48 | <2d\_arr\_init1> | → | (<arr\_elem>) <2d\_arr\_init2> |
| 49 | <2d\_arr\_init2> | → | , ( <arr\_elem> ) <2d\_arr\_init2> |
| 50 | <2d\_arr\_init2> | → | λ |
| 51 | <math\_optr> | → | + |
| 52 | <math\_optr> | → | - |
| 53 | <math\_optr> | → | \* |
| 54 | <math\_optr> | → | / |
| 55 | <math\_optr> | → | % |
| 56 | <size\_arr> | → | [<array\_altr>] <size\_arr1> |
| 57 | <size\_arr> | → | λ |
| 58 | <size\_arr1> | → | [<array\_altr>] |
| 59 | <size\_arr1> | → | λ |
| 60 | <deck\_dec> | → | deck identifier { <deck\_cont> } <deck\_obj> ; |
| 61 | <deck\_cont> | → | <datatype> identifier <size\_arr> ; <deck\_elem> |
| 62 | <deck\_elem> | → | <deck\_cont> |
| 63 | <deck\_elem> | → | λ |
| 64 | <deck\_obj> | → | identifier <size\_arr><deck\_obj1> |
| 65 | <deck\_obj1> | → | ,<deck\_obj> |
| 66 | <deck\_obj1> | → | λ |
| 67 | <void> | → | void identifier ( <param> ) { <statements> } |
| 68 | <param> | → | <datatype> identifier <param1> |
| 69 | <param> | → | λ |
| 70 | <param1> | → | ,<datatype> identifier <param1> |
| 71 | <param1> | → | λ |
| 72 | <func\_stmt> | → | <func2> <func\_stmt> |
| 73 | <func\_stmt> | → | λ |
| 74 | <func2> | → | <datatype> identifier <data\_altr> <id\_add>; |
| 75 | <func2> | → | identifier <id\_next\_func>; |
| 76 | <id\_next\_func> | → | <id\_next> |
| 77 | <id\_next\_func> | → | (<funct\_param>) |
| 78 | <func2> | → | <incdec\_optr>identifier; |
| 79 | <func2> | → | <io\_stmt> |
| 80 | <func2> | → | <func\_if\_stmt> |
| 81 | <func2> | → | <looping> |
| 82 | <func2> | → | blank; |
| 83 | <func\_if\_stmt> | → | if(<condition>){<func\_states>}<func\_elsif\_stmt>  <func\_else\_stmt> |
| 84 | <func\_states> | → | <func3><func\_states\_rear> |
| 85 | <func\_states\_rear> | → | <func\_states> |
| 86 | <func\_states\_rear> | → | λ |
| 87 | <func3> | → | <datatype> identifier <data\_altr> <id\_add>; |
| 88 | <func3> | → | identifier <id\_next\_func>; |
| 89 | <func3> | → | <incdec\_optr>identifier; |
| 90 | <func3> | → | <io\_stmt> |
| 91 | <func3> | → | <func\_if\_stmt> |
| 92 | <func3> | → | <return1> |
| 93 | <func3> | → | <looping> |
| 94 | <func3> | → | blank; |
| 95 | <func\_elsif\_stmt> | → | elsif (<condition>){<func\_states>}<func\_elsif\_stmt> |
| 96 | <func\_elsif\_stmt> | → | λ |
| 97 | <func\_else\_stmt> | → | else{< func\_states >} |
| 98 | <func\_else\_stmt> | → | λ |
| 99 | <return1> | → | return <return\_choices><return\_add>; |
| 100 | <return\_add> | → | , <return\_choices><return\_add> |
| 101 | <return\_add> | → | λ |
| 102 | <return\_choices> | → | numlit<digit\_rear\_rtn> |
| 103 | <return\_choices> | → | decimallit<digit\_rear\_rtn> |
| 104 | <return\_choices> | → | setlit |
| 105 | <return\_choices> | → | optionlit |
| 106 | <return\_choices> | → | identifier<return\_rear><digit\_rear\_rtn> |
| 107 | <return\_rear> | → | (<funct\_param\_rtn>) |
| 108 | <return\_rear> | → | λ |
| 109 | <digit\_rear\_rtn> | → | <math\_optr><digit\_opMath\_rtn> |
| 110 | <digit\_rear\_rtn> | → | λ |
| 111 | <funct\_param\_rtn> | → | <value\_rtn> <digit\_rear\_rtn><func\_param\_rear\_rtn> |
| 112 | <funct\_param\_rtn> | → | λ |
| 113 | <func\_param\_rear\_rtn> | → | , <funct\_param\_rtn> |
| 114 | <func\_param\_rear\_rtn> | → | λ |
| 115 | <value\_rtn> | → | <literals> |
| 116 | <value\_rtn> | → | identifier |
| 117 | <digit\_opMath\_rtn> | → | <digit1\_rtn><digit\_rear\_rtn> |
| 118 | <digit1\_rtn> | → | (<digit\_rtn><digit\_rear\_rtn>) |
| 119 | <digit1\_rtn> | → | <digit\_rtn> |
| 120 | <digit\_rtn> | → | numlit <incdec\_rear> |
| 121 | <digit\_rtn> | → | decimallit <incdec\_rear> |
| 122 | <digit\_rtn> | → | identifier |
| 123 | <statements> | → | <func> <statements> |
| 124 | <statements> | → | λ |
| 125 | <func> | → | <datatype> identifier <data\_altr> <id\_add>; |
| 126 | <func> | → | identifier <id\_next\_func>; |
| 127 | <func> | → | <incdec\_optr>identifier; |
| 128 | <func> | → | <io\_stmt> |
| 129 | <func> | → | <if\_stmt> |
| 130 | <func> | → | <looping> |
| 131 | <func> | → | blank; |
| 132 | <id\_next> | → | <incdec\_optr> |
| 133 | <id\_next> | → | <id\_rear> <id\_funcrear> |
| 134 | <id\_next> | → | <digit\_rear> |
| 135 | <id\_next> | → | λ |
| 136 | <incdec\_optr> | → | ++ |
| 137 | <incdec\_optr> | → | -- |
| 138 | <id\_rear> | → | <id\_choices> |
| 139 | <id\_rear> | → | λ |
| 140 | <id\_choices> | → | <elem> |
| 141 | <id\_choices> | → | <ins\_arr> |
| 142 | <elem> | → | .identifier <size\_arr> |
| 143 | <elem> | → | λ |
| 144 | <ins\_arr> | → | [ <array\_altr> ] <ins\_arr2> |
| 145 | <ins\_arr2> | → | [ <array\_altr> ] |
| 146 | <ins\_arr2> | → | <elem> |
| 147 | <ins\_arr2> | → | λ |
| 148 | <id\_funcrear> | → | <ass\_optr> <ass\_opValues> |
| 149 | <ass\_optr> | → | = |
| 150 | <ass\_optr> | → | += |
| 151 | <ass\_optr> | → | -= |
| 152 | <ass\_optr> | → | \*= |
| 153 | <ass\_optr> | → | /= |
| 154 | <ass\_optr> | → | %= |
| 155 | <ass\_opValues> | → | <digit\_opMath> |
| 156 | <ass\_opValues> | → | setlit <str\_con> |
| 157 | <ass\_opValues> | → | optionlit |
| 158 | <ass\_opValues> | → | piecelit |
| 159 | <digit\_opMath> | → | <digit><digit\_rear> |
| 160 | <digit\_opMath> | → | (<digit1><digit\_rear>)<digit\_rear> |
| 161 | <digit1> | → | (<digit><digit\_rear>) |
| 162 | <digit1> | → | <digit> |
| 163 | <digit> | → | numlit <incdec\_rear> |
| 164 | <digit> | → | decimallit <incdec\_rear> |
| 165 | <digit> | → | identifier <choice\_rear> |
| 166 | <digit> | → | <incdec\_optr> <digit\_cont> |
| 167 | <digit\_cont> | → | numlit |
| 168 | <digit\_cont> | → | decimallit |
| 169 | <digit\_cont> | → | identifier |
| 170 | <choice\_rear> | → | <incdec\_optr> |
| 171 | <choice\_rear> | → | <id\_cmpr> |
| 172 | <choice\_rear> | → | λ |
| 173 | <digit\_rear> | → | <math\_optr><digit\_opMath> |
| 174 | <digit\_rear> | → | λ |
| 175 | <id\_cmpr> | → | <id\_rear> |
| 176 | <id\_cmpr> | → | (<funct\_param>) |
| 177 | <id\_cmpr> | → | λ |
| 178 | <funct\_param> | → | <value> <digit\_rear><func\_param\_rear> |
| 179 | <func\_param\_rear> | → | , <funct\_param> |
| 180 | <func\_param\_rear> | → | λ |
| 181 | <value> | → | <literals> |
| 182 | <value> | → | <elem\_rear> |
| 183 | <elem\_rear> | → | identifier<elem> |
| 184 | <io\_stmt> | → | tell( <output>); |
| 185 | <io\_stmt> | → | check(identifier<id\_rear>); |
| 186 | <output> | → | <digit\_opMath1> <output\_rear> |
| 187 | <output> | → | <setlit> <output\_rear> |
| 188 | <output\_rear> | → | ,<output> |
| 189 | <output\_rear> | → | λ |
| 190 | <digit\_opMath1> | → | <digit2><digit\_rear1> |
| 191 | <digit\_opMath1> | → | (<digit3><digit\_rear1>) <digit\_rear1> |
| 192 | <digit3> | → | (<digit2><digit\_rear1>) |
| 193 | <digit3> | → | <digit2> |
| 194 | <digit2> | → | identifier <id\_cmpr> |
| 195 | <digit\_rear1> | → | <mathoptr><digit\_opMath1> |
| 196 | <digit\_rear1> | → | λ |
| 197 | <if\_stmt> | → | if(<condition>){<statements>}<elsif\_stmt>  <else\_stmt> |
| 198 | <condition> | → | <conditional\_optr> |
| 199 | <condition> | → | <not1>(<conditional\_optr>)<rear> |
| 200 | <rear> | → | <logoptr\_rear> |
| 201 | <rear> | → | <conditional\_rear> |
| 202 | <rear> | → | λ |
| 203 | <conditional\_rear> | → | <reloptr><cmpr> |
| 204 | <conditional\_rear> | → | λ |
| 205 | <compare> | → | identifier<id\_cmpr> <digit\_optrCmpr> |
| 206 | <compare> | → | numlit <digit\_optrCmpr> |
| 207 | <compare> | → | decimallit <digit\_optrCmpr> |
| 208 | <compare> | → | setlit |
| 209 | <compare> | → | optionlit |
| 210 | <compare> | → | piecelit |
| 211 | <logoptr\_rear> | → | <logoptr><logoptr\_add><logoptr\_rear> |
| 212 | <logoptr\_rear> | → | λ |
| 213 | <logoptr> | → | && |
| 214 | <logoptr> | → | ## |
| 215 | <logoptr\_add> | → | <condition> |
| 216 | <not1> | → | ! |
| 217 | <not1> | → | λ |
| 218 | <conditional\_optr> | → | <compare\_optr><conditional\_optrear> |
| 219 | <conditional\_optrear> | → | <reloptr><cmpr><logoptr\_rear> |
| 220 | <conditional\_optrear> | → | <logoptr\_rear> |
| 221 | <cmpr> | → | <compare> |
| 222 | <cmpr> | → | <not1>(<compare\_optr>) |
| 223 | <compare\_optr> | → | <compare> |
| 224 | <digit\_optrCmpr> | → | <math\_optr><digit\_opMath> |
| 225 | <digit\_optrCmpr> | → | λ |
| 226 | <reloptr> | → | == |
| 227 | <reloptr> | → | != |
| 228 | <reloptr> | → | > |
| 229 | <reloptr> | → | < |
| 230 | <reloptr> | → | >= |
| 231 | <reloptr> | → | <= |
| 232 | <elsif\_stmt> | → | elsif (<condition>){<statements>}<elsif\_stmt> |
| 233 | <elsif\_stmt> | → | λ |
| 234 | <else\_stmt> | → | else{<statements>} |
| 235 | <else\_stmt> | → | λ |
| 236 | <looping> | → | for(<init>;<condition>;<for\_unary>){<states>} |
| 237 | <looping> | → | while(<condition>){<states>} |
| 238 | <looping> | → | do{<states>}while(<condition>); |
| 239 | <states> | → | <func1><states\_rear> |
| 240 | <states\_rear> | → | <states> |
| 241 | <states\_rear> | → | λ |
| 242 | <func1> | → | <datatype> identifier <data\_altr> <id\_add>; |
| 243 | <func1> | → | identifier <id\_next\_func>; |
| 244 | <func1> | → | <incdec\_optr>identifier; |
| 245 | <func1> | → | <io\_stmt> |
| 246 | <func1> | → | <if\_stmt1> |
| 247 | <func1> | → | <looping> |
| 248 | <func1> | → | <ctrl> |
| 249 | <func1> | → | blank; |
| 250 | <if\_stmt1> | → | if(<condition>){<states>}<elsif\_stmt1><else\_stmt1> |
| 251 | <elsif\_stmt1> | → | elsif (<condition>){<states>}<elsif\_stmt> |
| 252 | <elsif\_stmt1> | → | λ |
| 253 | <else\_stmt1> | → | else{<states>} |
| 254 | <else\_stmt1> | → | λ |
| 255 | <ctrl> | → | continue; |
| 256 | <ctrl> | → | break; |
| 257 | <init> | → | identifier = <digit\_opMath> |
| 258 | <init> | → | λ |
| 259 | <for\_unary> | → | identifier<for\_rear> |
| 260 | <for\_unary> | → | <incdec\_optr>identifier <for\_digitopMath\_rear> |
| 261 | <for\_rear> | → | <incdec\_optr> |
| 262 | <for\_rear> | → | <ass\_optr> <ass\_opValues1> |
| 263 | <for\_rear> | → | <math\_optr> <ass\_opValues1> |
| 264 | <ass\_opValues1> | → | <for\_digitopMath> |
| 265 | <for\_digitopMath> | → | numlit <for\_digitopMath\_rear> |
| 266 | <for\_digitopMath> | → | identifier <for\_digitopMath\_rear> |
| 267 | <for\_digitopMath\_rear> | → | <math\_optr><for\_digitopMath> |
| 268 | <for\_digitopMath\_rear> | → | λ |
| 269 | <str\_con> | → | + <setlit> <str\_con> |
| 270 | <str\_con> | → | λ |
| 271 | <array\_altr\_expr> | → | <math\_optr> <array\_altr> |
| 272 | <array\_altr\_expr> | → | λ |
| 273 | <incdec\_rear> | → | <incdec\_optr> |
| 274 | <incdec\_rear> | → | λ |

# **FIRST SET**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROGRAM** | | **→** | **FIRST SET** |
| 1 | <program> | → | TABLE, λ, locked, num, decimal, piece, set, option, deck, void |
| 2 | <global> | → | λ, locked, num, decimal, piece, set, option, deck, void |
| 3 | <const\_dec> | → | locked |
| 4 | <const\_dec1> | → | num, decimal, piece, set, option |
| 5 | <const\_dec2> | → | ,, λ |
| 6 | <datatype> | → | num, decimal, piece, set, option |
| 7 | <data\_id> | → | num, decimal, piece, set, option |
| 8 | <data\_cont> | → | =, λ, [ |
| 9 | <data\_altr> | → | =, λ, [ |
| 10 | <id\_add> | → | ,, λ |
| 11 | <var\_dec> | → | =, λ |
| 12 | <literals> | → | numlit, decimallit, piecelit, setlit, optionlit |
| 13 | <1d\_arr> | → | [, λ |
| 14 | <array\_altr> | → | numlit, identifier |
| 15 | <2d\_arr> | → | =, [, λ |
| 16 | <1d\_arr\_init> | → | = |
| 17 | <arr\_elem> | → | numlit, decimallit, piecelit, setlit |
| 18 | <arr\_elem1> | → | ,, λ |
| 19 | <arr\_lit> | → | numlit, decimallit, piecelit, setlit |
| 20 | <2d\_arr\_init> | → | =, λ |
| 21 | <2d\_arr\_init1> | → | ( |
| 22 | <2d\_arr\_init2> | → | ,, λ |
| 23 | <math\_optr> | → | +, -, \*, /, % |
| 24 | <size\_arr> | → | [, λ |
| 25 | <size\_arr1> | → | [, λ |
| 26 | <deck\_dec> | → | deck |
| 27 | <deck\_cont> | → | num, decimal, piece, set, option |
| 28 | <deck\_elem> | → | num, decimal, piece, set, option, λ |
| 29 | <deck\_obj> | → | identifier |
| 30 | <deck\_obj1> | → | ,, λ |
| 31 | <void> | → | void |
| 32 | <param> | → | num, decimal, piece, set, option, λ |
| 33 | <param1> | → | ,, λ |
| 34 | <func\_stmt> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, blank, λ |
| 35 | <func2> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, blank |
| 36 | <id\_next\_func> | → | ++, --, ., λ, [,+, -, \*, /, % |
| 37 | <func\_if\_stmt> | → | if |
| 38 | <func\_states> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank |
| 39 | <func\_states\_rear> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank, λ |
| 40 | <func3> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank |
| 41 | <func\_elsif\_stmt> | → | elsif, λ |
| 42 | <func\_else\_stmt> | → | else, λ |
| 43 | <return1> | → | return |
| 44 | <return\_add> | → | ,, λ |
| 45 | <return\_choices> | → | numlit, decimallit, setlit, optionlit, identifier |
| 46 | <return\_rear> | → | (,λ |
| 47 | <digit\_rear\_rtn> | → | +, -, \*, /, %, λ |
| 48 | <funct\_param\_rtn> | → | numlit, decimallit, piecelit, setlit, optionlit, identifier, λ |
| 49 | <func\_param\_rear\_rtn> | → | , λ |
| 50 | <value\_rtn> | → | numlit, decimallit, piecelit, setlit, optionlit, identifier |
| 51 | <digit\_opMath\_rtn> | → | (,numlit, decimallit, identifier |
| 52 | <digit1\_rtn> | → | (,numlit, decimallit, identifier |
| 53 | <digit\_rtn> | → | numlit, decimallit, identifier |
| 54 | <statements> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank, λ |
| 55 | <func> | → | num, decimal, option, set, piece, identifier, ++, --, tell, check, if, for, do, while, blank |
| 56 | <id\_next> | → | ++, --, ., λ, [,+, -, \*, /, % |
| 57 | <incdec\_optr> | → | ++, -- |
| 58 | <id\_rear> | → | ., [, λ |
| 59 | <id\_choices> | → | ., [, λ |
| 60 | <elem> | → | ., λ |
| 61 | <ins\_arr> | → | [ |
| 62 | <ins\_arr2> | → | [,. , λ |
| 63 | <id\_funcrear> | → | =, +=, -=, \*=, /=, %= |
| 64 | <ass\_optr> | → | =, +=, -=, \*=, /=, %= |
| 65 | <ass\_opValues> | → | numlit, decimallit, identifier, (, setlit, optionlit, piecelit, ++, -- |
| 66 | <digit\_opMath> | → | numlit, decimallit, identifier, (, ++, -- |
| 67 | <digit1> | → | (, numlit, decimallit, identifier, ++, -- |
| 68 | <digit> | → | numlit, decimallit, identifier, ++, -- |
| 69 | <digit\_cont> | → | numlit, decimallit, identifier |
| 70 | <choice\_rear> | → | ++, --, ., [, λ, ( |
| 71 | <digit\_rear> | → | +, -, \*, /, %, λ |
| 72 | <id\_cmpr> | → | ., [, (,λ |
| 73 | <funct\_param> | → | numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 74 | <func\_param\_rear> | → | ,, λ |
| 75 | <value> | → | numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 76 | <elem\_rear> | → | identifier |
| 77 | <io\_stmt> | → | tell, check |
| 78 | <output> | → | identifier, (, setlit |
| 79 | <output\_rear> | → | ,, λ |
| 80 | <digit\_opMath1> | → | identifier, ( |
| 81 | <digit3> | → | (, identifier |
| 82 | <digit2> | → | identifier |
| 83 | <digit\_rear1> | → | +, -, \*, /, %, λ |
| 84 | <if\_stmt> | → | if |
| 85 | <condition> | → | identifier, numlit, decimalit, piecelit, optionlit, setlit, !, λ |
| 86 | <rear> | → | &&, ##, ==, !=, >, <, >=, <=, λ |
| 87 | <conditional\_rear> | → | ==, !=, >, <, >=, <=, λ |
| 88 | <compare> | → | identifier, numlit, decimallit, piecelit, optionlit, setlit |
| 89 | <logoptr\_rear> | → | &&, ##, λ |
| 90 | <logoptr> | → | &&, ## |
| 91 | <logoptr\_add> | → | identifier, numlit, decimalit, piecelit, optionlit, setlit, ! |
| 92 | <not1> | → | !, λ |
| 93 | <conditional\_optr> | → | identifier, numlit, decimallit, setlit, piecelit, optionlit |
| 94 | <conditional\_optrear> | → | ==, !=, >, <, >=, <=, &&, ## |
| 95 | <cmpr> | → | identifier, numlit, decimallit, piecelit, setlit, optionlit, !, λ |
| 96 | <compare\_optr> | → | identifier, numlit, decimallit, piecelit, setlit, optionlit |
| 97 | <digit\_optrCmpr> | → | +, -, \*, /, %, λ |
| 98 | <reloptr> | → | ==, !=, >, <, >=, <= |
| 99 | <elsif\_stmt> | → | elsif, λ |
| 100 | <else\_stmt> | → | else, λ |
| 101 | <looping> | → | for, while, do |
| 102 | <states> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, continue, break, blank |
| 103 | <states\_rear> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, continue, break, blank, λ |
| 104 | <func1> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, continue, break, blank |
| 105 | <if\_stmt1> | → | if |
| 106 | <elsif\_stmt1> | → | elsif, λ |
| 107 | <else\_stmt1> | → | else, λ |
| 108 | <ctrl> | → | continue, break |
| 109 | <init> | → | identifier, λ |
| 110 | <for\_unary> | → | identifier, ++, -- |
| 111 | <for\_rear> | → | ++, --, =, +=, -=, \*=, /=, %=, +, -, \*, /, % |
| 112 | <ass\_opValues1> | → | numlit, identifier |
| 113 | <for\_digitopMath> | → | numlit, identifier |
| 114 | <for\_digitopMath\_rear> | → | +, -, \*, /, %, λ |
| 115 | <str\_con> | → | +, λ |
| 116 | <array\_altr\_expr> | → | +, -, \*, /, %, λ |
| 117 | <incdec\_rear> | → | ++, --, λ |

# **FOLLOW SET**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROGRAM** | | **→** | **FOLLOW SET** |
| 1 | <global> | → | TABLE |
| 2 | <const\_dec> | → | locked, num, decimal, piece, set, option, deck, void, TABLE |
| 3 | <const\_dec1> | → | ; |
| 4 | <const\_dec2> | → | ; |
| 5 | <datatype> | → | identifier |
| 6 | <data\_id> | → | locked, num, decimal, piece, set, option, deck, void, TABLE |
| 7 | <data\_cont> |  | locked, num, decimal, piece, set, option, deck, void, TABLE |
| 8 | <data\_altr> | → | ,, ; |
| 9 | <id\_add> | → | ; |
| 10 | <var\_dec> | → | ,, ; |
| 11 | <literals> | → | ,, ;, +, -, \*, /, %, ) |
| 12 | <1d\_arr> | → | ,, ; |
| 13 | <array\_altr> | → | ] |
| 14 | <2d\_arr> | → | ,, ; |
| 15 | <1d\_arr\_init> | → | ,, ; |
| 16 | <arr\_elem> | → | ) |
| 17 | <arr\_elem1> | → | ) |
| 18 | <arr\_lit> | → | ,, ) |
| 19 | <2d\_arr\_init> | → | ,, ; |
| 20 | <2d\_arr\_init1> | → | ) |
| 21 | <2d\_arr\_init2> | → | ) |
| 22 | <math\_optr> | → | numlit, decimallit, identifier, (, ++, -- |
| 23 | <size\_arr> | → | ,, ; }, =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ), numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 24 | <size\_arr1> | → | ; |
| 25 | <deck\_dec> | → | locked, num, decimal, piece, set, option, deck, void, TABLE |
| 26 | <deck\_cont> | → | } |
| 27 | <deck\_elem> | → | } |
| 28 | <deck\_obj> | → | ; |
| 29 | <deck\_obj1> | → | ; |
| 30 | <void> | → | locked, num, decimal, piece, set, option, deck, void, TABLE |
| 31 | <param> | → | ) |
| 32 | <param1> | → | ) |
| 33 | <func\_stmt> | → | return |
| 34 | <func2> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, blank, return |
| 35 | <id\_next\_func> | → | ; |
| 36 | <func\_if\_stmt> | → | num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, return |
| 37 | <func\_states> | → | } |
| 38 | <func\_states\_rear> | → | } |
| 39 | <func3> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank |
| 40 | <func\_elsif\_stmt> | → | else, num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, return |
| 41 | <func\_else\_stmt> | → | num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, return |
| 42 | <return1> | → | }, num, decimal, piece, option, set, identifier, ++, --, tell, check, if, return, for, while, do, blank |
| 43 | <return\_add> | → | ; |
| 44 | <return\_choices> | → | , |
| 45 | <return\_rear> | → | +, -, \*, /, %, ; |
| 46 | <digit\_rear\_rtn> | → | ;, ,, ) |
| 47 | <funct\_param\_rtn> | → | ) |
| 48 | <funct\_param\_rear\_rtn> | → | ) |
| 49 | <value\_rtn> | → | +, -, \*, /, %, ,, ) |
| 50 | <digit\_opMath\_rtn> | → | ;, ,, ) |
| 51 | <digit1\_rtn> | → | +, -, \*, /, %, ;, ,, ) |
| 52 | <digit\_rtn> | → | +, -, \*, /, %, ) |
| 53 | <statements> | → | } |
| 54 | <func> | → | num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, } |
| 55 | <id\_next> | → | ; |
| 56 | <incdec\_optr> | → | identifier, ;, numlit, decimallit, +, -, \*, /, %, ) |
| 57 | <id\_rear> | → | =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, ) |
| 58 | <id\_choices> | → | =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, ) |
| 59 | <elem> | → | =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, ), numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 60 | <ins\_arr> | → | =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, ) |
| 61 | <ins\_arr2> | → | =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, ) |
| 62 | <id\_funcrear> | → | ; |
| 63 | <ass\_optr> | → | numlit, decimallit, identifier, (, setlit, optionlit, piecelit, ++, -- |
| 64 | <ass\_opValues> | → | ; |
| 65 | <digit\_opMath> | → | ;, ), ==, !=, >, <, >=, <=, &&, ##, , |
| 66 | <digit1> | → | +, -, \*, /, %, ) |
| 67 | <digit> | → | +, -, \*, /, %, ) |
| 68 | <digit\_cont> |  | +, -, \*, /, %, ) |
| 69 | <choice\_rear> | → | +, -, \*, /, %, ) |
| 70 | <digit\_rear> | → | ;, ), ==, !=, >, <, >=, <=, &&, ##, ,, |
| 71 | <id\_cmpr> | → | +, -, \*, /, %, ,, ), ;, ==, !=, >, <, >=, <=, &&, ## |
| 72 | <funct\_param> | → | ) |
| 73 | <func\_param\_rear> | → | ) |
| 74 | <value> | → | +, -, \*, /, %, ,,numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 75 | <elem\_rear> | → | +, -, \*, /, %, ,,numlit, decimallit, piecelit, optionlit, setlit, identifier |
| 76 | <io\_stmt> | → | num, decimal, piece, set, option, identifier, ++, --, if, continue, break, for, do, while, blank, return, } |
| 77 | <output> | → | ) |
| 78 | <output\_rear> | → | ) |
| 79 | <digit\_opMath1> | → | ,, ) |
| 80 | <digit3> | → | +, -, \*, /, %, ) |
| 81 | <digit2> | → | +, -, \*, /, %, ,, ) |
| 82 | <digit\_rear1> | → | ,, ) |
| 83 | <if\_stmt> | → | num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, } |
| 84 | <condition> | → | ),; |
| 85 | <rear> | → | ),; |
| 86 | <conditional\_rear> | → | ),; |
| 87 | <compare> | → | ), ;, ==, !=, >, <, >=, <=, &&, ## |
| 88 | <logoptr\_rear> | → | ),; |
| 89 | <logoptr> | → | identifier, numlit, decimalit, piecelit, optionlit, setlit, ! |
| 90 | <logoptr\_add> | → | ),;, &&, ##, |
| 91 | <not1> | → | ( |
| 92 | <conditional\_optr> | → | ),; |
| 93 | <conditional\_optrear> | → | ),; |
| 94 | <cmpr> | → | ),;, &&, ## |
| 95 | <compare\_optr> | → | ), ==, !=, >, <, >=, <=, &&, ## |
| 96 | <digit\_optrCmpr> | → | ), ;, ==, !=, >, <, >=, <=, &&, ## |
| 97 | <reloptr> | → | identifier, numlit, decimallit, piecelit, setlit, optionlit, !, &&, ##, ==, !=, >, <, >=, <=, ), ; |
| 98 | <elsif\_stmt> | → | else, num, decimal, piece, set, option, identifier, ++, --, tell, check, for, do, while, blank, } |
| 99 | <else\_stmt> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, for, do, while, blank, } |
| 100 | <looping> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, blank, return |
| 101 | <states> | → | } |
| 102 | <states\_rear> | → | } |
| 103 | <func1> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, continue, break, for, do, while, blank, } |
| 104 | <if\_stmt1> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, continue, break, for, do, while, blank, } |
| 105 | <elsif\_stmt1> | → | else, num, decimal, piece, set, option, identifier, ++, --, tell, if, check, continue, break, for, do, while, blank, } |
| 106 | <else\_stmt1> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, continue, break, for, do, while, blank, } |
| 107 | <ctrl> | → | num, decimal, piece, set, option, identifier, ++, --, tell, check, if, continue, break, for, do, while, blank, } |
| 108 | <init> | → | ; |
| 109 | <for\_rear> | → | ) |
| 110 | <ass\_opValues1> | → | ) |
| 111 | <for\_digitopMath> | → | ) |
| 112 | <for\_digitopMath\_rear> | → | ) |
| 113 | <for\_unary> | → | ) |
| 114 | <str\_con> | → | ; |
| 115 | <array\_altr\_expr> | → | ] |
| 116 | <incdec\_rear> | → | +, -, \*, /, %, ) |

# **PREDICT SET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | <program> | FIRST(<global> TABLE() { <statements> }) | FIRST(global) | {TABLE, locked, num, decimal, piece, set, option, deck, void} |
| 2 | <global> | FIRST(<const\_dec> <global>) | FIRST(const\_dec) | {locked} |
| 3 | <global> | FIRST(<data\_id> <global>) | FIRST(data\_id) | {num, decimal, piece, set, option} |
| 4 | <global> | FIRST(<deck\_dec> <global>) | FIRST(deck\_dec) | {deck} |
| 5 | <global> | FIRST(<void> <global>) | FIRST(void) | {void} |
| 6 | <global> | FIRST(λ) | FIRST(λ) **U** FOLLOW(global) | {TABLE} |
| 7 | <const\_dec> | FIRST (locked <const\_dec1>;) | FIRST (locked) | {locked} |
| 8 | <const\_dec1> | FIRST(<datatype> identifier = <literals> <const\_dec2>;) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 9 | <const\_dec2> | FIRST(,identifier = <literals> <const\_dec2>) | FIRST(,) | {,} |
| 10 | <const\_dec2> | FIRST(λ) | FIRST(λ) **U** FOLLOW(const\_dec2) | {;} |
| 11 | <datatype> | FIRST(num) | FIRST(num) | {num} |
| 12 | <datatype> | FIRST(decimal) | FIRST(decimal) | {decimal} |
| 13 | <datatype> | FIRST(piece) | FIRST(piece) | {piece} |
| 14 | <datatype> | FIRST(set) | FIRST(set) | {set} |
| 15 | <datatype> | FIRST(option) | FIRST(option) | {option} |
| 16 | <data\_id> | FIRST(<datatype> identifier <data\_cont>) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 17 | <data\_cont> | FIRST(<data\_altr> <id\_add>;) | FIRST(data\_altr) | {=, [} |
| 18 | <data\_cont> | FIRST((<param>) { <func\_stmt> <return1> }) | FIRST(() | {(} |
| 19 | <data\_altr> | FIRST(<var\_dec>) | FIRST(var\_dec) | {=} |
| 20 | <data\_altr> | FIRST(<1d\_arr>) | FIRST(1d\_arr) | {[} |
| 21 | <data\_altr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(data\_altr) | {,, ;} |
| 22 | <id\_add> | FIRST(,identifier <data\_altr> <id\_add>) | FIRST(,) | {,} |
| 23 | <id\_add> | FIRST(λ) | FIRST(λ) **U** FOLLOW(id\_add) | {;} |
| 24 | <var\_dec> | FIRST(=<literals> <var\_dec2>) | FIRST(=) | {=} |
| 25 | <var\_dec> | FIRST(λ) | FIRST(λ) **U** FOLLOW(var\_dec) | {,, ;} |
| 26 | <literals> | FIRST(numlit) | FIRST(numlit) | {numlit} |
| 27 | <literals> | FIRST(decimallit) | FIRST(decimallit) | {decimallit} |
| 28 | <literals> | FIRST(piecelit) | FIRST(piecelit) | {piecelit} |
| 29 | <literals> | FIRST(setlit) | FIRST(setlit) | {setlit} |
| 30 | <literals> | FIRST(optionlit) | FIRST(optionlit) | {optionlit} |
| 31 | <1d\_arr> | FIRST([ <array\_altr> ] <2d\_arr>) | FIRST([) | {[} |
| 32 | <1d\_arr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(1d\_arr) | {,, ;} |
| 33 | <array\_altr> | FIRST(numlit <array\_altr\_expr>) | FIRST(numlit) | {numlit} |
| 34 | <array\_altr> | FIRST(identifier <array\_altr\_expr>) | FIRST(identifier) | {identifier} |
| 35 | <2d\_arr> | FIRST(<1d\_arr\_init>) | FIRST(1d\_arr\_init) | {=} |
| 36 | <2d\_arr> | FIRST([ <array\_altr> ] <2d\_arr\_init>) | FIRST([) | {[} |
| 37 | <2d\_arr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(2d\_arr) | {,, ;} |
| 38 | <1d\_arr\_init> | FIRST (=( <arr\_elem> )) | FIRST (=) | {=} |
| 39 | <arr\_elem> | FIRST (<arr\_lit> <arr\_elem1>) | FIRST (arr\_lit) | {numlit, decimallit, piecelit, setlit} |
| 40 | <arr\_elem1> | FIRST (, <arr\_elem>) | FIRST (,) | {,} |
| 41 | <arr\_elem1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(arr\_elem1) | {)} |
| 42 | <arr\_lit> | FIRST(numlit) | FIRST(numlit) | {numlit} |
| 43 | <arr\_lit> | FIRST(decimallit) | FIRST(decimallit) | {decimallit} |
| 44 | <arr\_lit> | FIRST(piecelit) | FIRST(piecelit) | {piecelit} |
| 45 | <arr\_lit> | FIRST(setlit) | FIRST(setlit) | {setlit} |
| 46 | <2d\_arr\_init> | FIRST(=( <2d\_arr\_init1> )) | FIRST (=) | {=} |
| 47 | <2d\_arr\_init> | FIRST(λ) | FIRST(λ) **U** FOLLOW(2d\_arr\_init) | {,, ;} |
| 48 | <2d\_arr\_init1> | FIRST(( <arr\_elem> ) <2d\_arr\_init2>) | FIRST(() | {(} |
| 49 | <2d\_arr\_init2> | FIRST(, ( <arr\_elem> ) <2d\_arr\_init2> | FIRST(,) | {,} |
| 50 | <2d\_arr\_init2> | FIRST(λ) | FIRST(λ) **U** FOLLOW(2d\_arr\_init2) | {)} |
| 51 | <math\_optr> | FIRST(+) | FIRST(+) | {+} |
| 52 | <math\_optr> | FIRST(-) | FIRST(-) | {-} |
| 53 | <math\_optr> | FIRST(\*) | FIRST(\*) | {\*} |
| 54 | <math\_optr> | FIRST(/) | FIRST(/) | {/} |
| 55 | <math\_optr> | FIRST(%) | FIRST(%) | {%} |
| 56 | <size\_arr> | FIRST([ <array\_altr> ] <size\_arr1>) | FIRST([) | {[} |
| 57 | <size\_arr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(size\_arr) | {,, ; }, =, +=, -=, \*=, /=, %= , +, -, \*, /, %, ), numlit, decimallit, piecelit, optionlit, setlit, identifier } |
| 58 | <size\_arr1> | FIRST([ <array\_altr> ]) | FIRST([) | {[} |
| 59 | <size\_arr1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(size\_arr1) | {;} |
| 60 | <deck\_dec> | FIRST(deck identifier { <deck\_cont> } <deck\_obj> ;) | FIRST(deck) | {deck} |
| 61 | <deck\_cont> | FIRST(<datatype> identifier <size\_arr> ; <deck\_elem>) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 62 | <deck\_elem> | FIRST(<deck\_cont>) | FIRST(<deck\_cont>) | {num, decimal, piece, set, option} |
| 63 | <deck\_elem> | FIRST(λ) | FIRST(λ) **U** FOLLOW(deck\_elem) | {}} |
| 64 | <deck\_obj> | FIRST(identifier <deck\_obj1>) | FIRST(identifier) | {identifier} |
| 65 | <deck\_obj1> | FIRST(,<deck\_obj>) | FIRST(,) | {,} |
| 66 | <deck\_obj1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(deck\_obj1) | {;} |
| 67 | <void> | FIRST(void identifier ( <param> ) { <statements> } ) | FIRST(void) | {void} |
| 68 | <param> | FIRST(<datatype> identifier <param1>) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 69 | <param> | FIRST(λ) | FIRST(λ) **U** FOLLOW(param) | {)} |
| 70 | <param1> | FIRST(,<datatype> identifier <param1>) | FIRST(,) | {,} |
| 71 | <param1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(param1) | {)} |
| 72 | <func\_stmt> | FIRST(<func2> <func\_stmt>) | FIRST(func2) | {num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, blank} |
| 73 | <func\_stmt> | FIRST(λ) | FIRST(λ) **U** FOLLOW(func\_stmt) | {return} |
| 74 | <func2> | FIRST(<datatype> identifier <data\_altr> <id\_add>;) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 75 | <func2> | FIRST(identifier <id\_next\_func>;) | FIRST(identifier) | {identifier} |
| 76 | <id\_next\_func> | FIRST(<id\_next>) | FIRST(id\_next) | {++, --, ., [,+, -, \*, /, %} |
| 77 | <id\_next\_func> | FIRST((<funct\_param>);) | FIRST(() | {(} |
| 78 | <func2> | FIRST(<incdec\_optr>identifier;) | FIRST(incdec\_optr) | {++, --} |
| 79 | <func2> | FIRST(<io\_stmt>) | FIRST(io\_stmt) | {tell, check} |
| 80 | <func2> | FIRST(<func\_if\_stmt>) | FIRST(func\_if\_stmt) | { if} |
| 81 | <func2> | FIRST(<looping>) | FIRST(looping) | { for, while, do} |
| 82 | <func2> | FIRST(blank;) | FIRST(blank) | {blank} |
| 83 | <func\_if\_stmt> | FIRST(if(<condition>){<func\_states>}<func\_elsif\_stmt>  <func\_else\_stmt>) | FIRST(if) | {if} |
| 84 | <func\_states> | FIRST(<func3><func\_states\_rear>) | FIRST(func3) | {num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank} |
| 85 | <func\_states\_rear> | FIRST(<func\_states>) | FIRST(func\_states) | { num, decimal, piece, set, option, identifier, ++, --, tell, check, if, return, for, while, do, blank} |
| 86 | <func\_states\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(func\_states\_rear) | {}} |
| 87 | <func3> | FIRST(<datatype> identifier <data\_altr> <id\_add>;) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 88 | <func3> | FIRST (identifier <id\_next\_func>;) | FIRST(identifier) | {identifier} |
| 89 | <func3> | FIRST(<incdec\_optr>identifier;) | FIRST(incdec\_optr) | {++, --} |
| 90 | <func3> | FIRST(<io\_stmt>) | FIRST(io\_stmt) | {tell, check} |
| 91 | <func3> | FIRST(<func\_if\_stmt>) | FIRST(func\_if\_stmt) | { if} |
| 92 | <func3> | FIRST(<return1>) | FIRST(return1) | { return} |
| 93 | <func3> | FIRST(<looping>) | FIRST(looping) | { for, while, do} |
| 94 | <func3> | FIRST(blank;) | FIRST(blank) | {blank} |
| 95 | <func\_elsif\_stmt> | FIRST(elsif (<condition>){<func\_states>}<func\_elsif\_stmt>) | FIRST(elsif) | {elsif} |
| 96 | <func\_elsif\_stmt> | FIRST(λ) | FIRST(λ) **U** FOLLOW(func\_elsif\_stmt) | {else, num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, return} |
| 97 | <func\_else\_stmt> | FIRST(else{< func\_states >}) | FIRST(else) | {else} |
| 98 | <func\_else\_stmt> | FIRST(λ) | FIRST(λ) **U** FOLLOW(func\_else\_stmt) | { num, decimal, piece, option, set, identifier, ++, --, tell, check, if, for, while, do, blank, return} |
| 99 | <return1> | FIRST(return <return\_choices> <return\_add>;) | FIRST(return) | {return} |
| 100 | <return\_add> | FIRST(, <return\_choices><return\_add>) | FIRST(,) | {,} |
| 101 | <return\_add> | FIRST(λ) | FIRST(λ) **U** FOLLOW(return\_add) | {;} |
| 102 | <return\_choices> | FIRST(numlit<digit\_rear\_rtn>) | FIRST(numlit) | {numlit} |
| 103 | <return\_choices> | FIRST(decimallit<digit\_rear\_rtn>) | FIRST(decimallit) | {decimallit} |
| 104 | <return\_choices> | FIRST(setlit) | FIRST(setlit) | {setlit} |
| 105 | <return\_choices> | FIRST(optionlit) | FIRST(optionlit) | {optionlit} |
| 106 | <return\_choices> | FIRST(identifier<return\_rear><digit\_rear\_rtn>) | FIRST(identifier) | {identifier} |
| 107 | <return\_rear> | FIRST((<funct\_param\_rtn>)) | FIRST(() | {(} |
| 108 | <return\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(return\_rear) | {+, -, \*, /, %, ;} |
| 109 | <digit\_rear\_rtn> | FIRST(<math\_optr><digit\_opMath\_rtn>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 110 | <digit\_rear\_rtn> | FIRST(λ) | FIRST(λ) U FOLLOW(digit\_rear\_rtn) | {;, , )} |
| 111 | <funct\_param\_rtn> | FIRST(<value\_rtn> <digit\_rear\_rtn>< func\_param\_rear\_rtn >) | FIRST(value\_rtn) | { numlit, decimallit, piecelit, setlit, optionlit, identifier} |
| 112 | <funct\_param\_rtn> | FIRST(λ) | FIRST(λ) **U** FOLLOW(funct\_param\_rtn) | {)} |
| 113 | <func\_param\_rear\_rtn> | FIRST(,<funct\_param\_rtn>) | FIRST(,) | {,} |
| 114 | <func\_param\_rear\_rtn> | FIRST(λ) | FIRST(λ) **U** FOLLOW(func\_param\_rear\_rtn) | {)} |
| 115 | <value\_rtn> | FIRST(<literals>) | FIRST(literals) | {numlit, decimallit, piecelit, setlit, optionlit} |
| 116 | <value\_rtn> | FIRST(identifier) | FIRST(identifier) | {identifier} |
| 117 | <digit\_opMath\_rtn> | FIRST(<digit1\_rtn><digit\_rear\_rtn>) | FIRST(digit1\_rtn) | {(,numlit, decimallit, identifier} |
| 118 | <digit1\_rtn> | FIRST((<digit\_rtn><digit\_rear\_rtn>)) | FIRST(() | {(} |
| 119 | <digit1\_rtn> | FIRST(<digit\_rtn>) | FIRST(digit\_rtn) | {numlit, decimallit, identifier} |
| 120 | <digit\_rtn> | FIRST(numlit <incdec\_rear>) | FIRST(numlit) | {numlit} |
| 121 | <digit\_rtn> | FIRST(decimallit <incdec\_rear>) | FIRST(decimallit) | {decimallit} |
| 122 | <digit\_rtn> | FIRST(identifier) | FIRST(identifier) | {identifier} |
| 123 | <statements> | FIRST(<func> <statements>) | FIRST(func) | {num, decimal, option, set, piece, identifier, ++, --, tell, check, if, for, do, while, blank} |
| 124 | <statements> | FIRST(λ) | FIRST(λ) **U** FOLLOW(statements) | {}} |
| 125 | <func> | FIRST(<datatype> identifier <data\_altr> <id\_add>;) | FIRST(datatype) | {num, decimal, piece, set, option} |
| 126 | <func> | FIRST(identifier<id\_next\_func>;) | FIRST(identifier) | {identifier} |
| 127 | <func> | FIRST(<incdec\_optr>identifier;) | FIRST(incdec\_optr) | {++, --} |
| 128 | <func> | FIRST(<io\_stmt>) | FIRST(io\_stmt) | {tell, check} |
| 129 | <func> | FIRST(<if\_stmt>) | FIRST(if\_stmt) | {if} |
| 130 | <func> | FIRST(<looping>) | FIRST(looping) | {for, do, while} |
| 131 | <func> | FIRST(blank;) | FIRST(blank) | {blank} |
| 132 | <id\_next> | FIRST(<incdec\_optr>) | FIRST(incdec\_optr) | {++, --} |
| 133 | <id\_next> | FIRST(<id\_rear> <id\_funcrear>) | FIRST(id\_rear) | {., [} |
| 134 | <id\_next> | FIRST(<digit\_rear>) | FIRST(digit\_rear) | {+, -, \*, /, %} |
| 135 | <id\_next> | FIRST(λ) | FIRST(λ) **U** FOLLOW(id\_next) | { ; } |
| 136 | <incdec\_optr> | FIRST(++) | FIRST(++) | {++} |
| 137 | <incdec\_optr> | FIRST(--) | FIRST(--) | {--} |
| 138 | <id\_rear> | FIRST(<id\_choices>) | FIRST(id\_choices) | {., [} |
| 139 | <id\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(id\_rear) | {=, +=, -=, \*=, /=, %= , +, -, \*, /, %, ,, )} |
| 140 | <id\_choices> | FIRST(<elem>) | FIRST(elem) | {.} |
| 141 | <id\_choices> | FIRST(<ins\_arr>) | FIRST(ins\_arr) | {[} |
| 142 | <elem> | FIRST(.identifier) | FIRST(.) | {.} |
| 143 | <elem> | FIRST(λ) | FIRST(λ) **U** FOLLOW(elem) | {), =, +=, -=, \*=, /=, %=, +, -, \*, /, %, ,, numlit, decimallit, piecelit, optionlit, setlit, identifier } |
| 144 | <ins\_arr> | FIRST ([ <array\_altr> ] <ins\_arr2>) | FIRST ([) | {[} |
| 145 | <ins\_arr2> | FIRST ([ <array\_altr> ]) | FIRST ([) | {[} |
| 146 | <ins\_arr2> | FIRST(<elem>) | FIRST(elem) | {.} |
| 147 | <ins\_arr2> | FIRST(λ) | FIRST(λ) **U** FOLLOW(ins\_arr2) | {), =, +=, -=, \*=, /=, %=, +, -, \*, /, %, , } |
| 148 | <id\_funcrear> | FIRST(<ass\_optr> <ass\_opValues>) | FIRST(ass\_optr) | {=, +=, -=, \*=, /=, %=} |
| 149 | <ass\_optr> | FIRST(=) | FIRST(=) | {=} |
| 150 | <ass\_optr> | FIRST(+=) | FIRST(+=) | {+=} |
| 151 | <ass\_optr> | FIRST(-=) | FIRST(-=) | {-=} |
| 152 | <ass\_optr> | FIRST(\*=) | FIRST(\*=) | {\*=} |
| 153 | <ass\_optr> | FIRST(/=) | FIRST(/=) | {/=} |
| 154 | <ass\_optr> | FIRST(%=) | FIRST(%=) | {%=} |
| 155 | <ass\_opValues> | FIRST(<digit\_opMath>) | FIRST(digit\_opMath) | {numlit, decimallit, identifier, (, ++, --} |
| 156 | <ass\_opValues> | FIRST(setlit <str\_con>) | FIRST(setlit) | {setlit} |
| 157 | <ass\_opValues> | FIRST(optionlit) | FIRST(optionlit) | {optionlit} |
| 158 | <ass\_opValues> | FIRST(piecelit) | FIRST(piecelit) | {piecelit} |
| 159 | <digit\_opMath> | FIRST(<digit><digit\_rear>) | FIRST(digit) | { numlit, decimallit, identifier, ++, --} |
| 160 | <digit\_opMath> | FIRST((<digit1><digit\_rear>)<digit\_rear>) | FIRST(() | {(} |
| 161 | <digit1> | FIRST((<digit><digit\_rear>) | FIRST(() | {(} |
| 162 | <digit1> | FIRST(<digit>) | FIRST(digit) | {numlit, decimallit, identifier, ++, --} |
| 163 | <digit> | FIRST(numlit) | FIRST(numlit) | {numlit} |
| 164 | <digit> | FIRST(decimallit) | FIRST(decimallit) | {decimallit} |
| 165 | <digit> | FIRST(identifier <choice\_rear>) | FIRST(identifier) | {identifier} |
| 166 | <digit> | FIRST(<incdec\_optr> <digit\_cont>) | FIRST(incdec\_optr) | {++, --} |
| 167 | <digit\_cont> | FIRST(numlit) | FIRST(numlit) | {numlit} |
| 168 | <digit\_cont> | FIRST(decimallit) | FIRST(decimallit) | {decimallit} |
| 169 | <digit\_cont> | FIRST(identifier) | FIRST(identifier) | {identifier} |
| 170 | <choice\_rear> | FIRST(<incdec\_optr>) | FIRST(incdec\_optr) | {++, --} |
| 171 | <choice\_rear> | FIRST(<id\_cmpr>) | FIRST(id\_cmpr) | {., [, (} |
| 172 | <choice\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(choice\_rear) | {+, -, \*, /, %, )} |
| 173 | <digit\_rear> | FIRST(<math\_optr><digit\_opMath>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 174 | <digit\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(digit\_rear) | {,, ;, ),&&, ##, ==, !=, >, <, >=, <=} |
| 175 | <id\_cmpr> | FIRST(<id\_rear>) | FIRST(id\_rear) | {., [} |
| 176 | <id\_cmpr> | FIRST( (<funct\_param>)) | FIRST(() | {(} |
| 177 | <id\_cmpr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(id\_cmpr) | {+, -, \*, /, %, ,, ), ;, ==, !=, >, <, >=, <=, &&, ##} |
| 178 | <funct\_param> | FIRST(<value> <digit\_rear><func\_param\_rear>) | FIRST(value) | {numlit, decimallit, piecelit, optionlit, setlit, identifier} |
| 179 | <func\_param\_rear> | FIRST(, <funct\_param>) | FIRST(,) | {,} |
| 180 | <func\_param\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(funct\_param\_rear) | {)} |
| 181 | <value> | FIRST(<literals>) | FIRST(literals) | {numlit, decimallit, optionlit, setlit, piecelit} |
| 182 | <value> | FIRST(<elem\_rear>) | FIRST(elem\_rear) | { identifier} |
| 183 | <elem\_rear> | FIRST(identifier<elem>) | FIRST(identifier) | { identifier} |
| 184 | <io\_stmt> | FIRST(tell( <output>);) | FIRST(tell) | {tell} |
| 185 | <io\_stmt> | FIRST(check(identifier<id\_rear>);) | FIRST(check) | {check} |
| 186 | <output> | FIRST(<digit\_opMath1> <output\_rear>) | FIRST(digit\_opMath1) | {identifier, (} |
| 187 | <output> | FIRST(setlit <output\_rear>) | FIRST(setlit) | {setlit} |
| 188 | <output\_rear> | FIRST(,<output>) | FIRST(,) | {,} |
| 189 | <output\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(output\_rear) | {)} |
| 190 | <digit\_opMath1> | FIRST(<digit2><digit\_rear1>) | FIRST (digit2) | {identifier} |
| 191 | <digit\_opMath1> | FIRST((<digit3><digit\_rear1>) <digit\_rear1>) | FIRST(() | {(} |
| 192 | <digit3> | FIRST((<digit2><digit\_rear1>)) | FIRST(() | {(} |
| 193 | <digit3> | FIRST(<digit2>) | FIRST(digit2) | {identifier} |
| 194 | <digit2> | FIRST(identifier <id\_cmpr>) | FIRST(identifier) | {identifier} |
| 195 | <digit\_rear1> | FIRST(<mathoptr><digit\_opMath1>) | FIRST(mathoptr) | {+, -, \*, /, %} |
| 196 | <digit\_rear1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(digit\_rear1) | {), ,} |
| 197 | <if\_stmt> | FIRST(if(<condition>){<statements>}<elsif\_stmt><else\_stmt>) | FIRST(if) | {if} |
| 198 | <condition> | FIRST(<conditional\_optr>) | FIRST(conditional\_optr) | {numlit, decimallit, identifier, setlit, piecelit, optionlit} |
| 199 | <condition> | FIRST(<not1>(<conditional\_optr><rear>)<rear>) | FIRST(not1) | {!} |
| 200 | <rear> | FIRST(<logoptr\_rear>) | FIRST(logoptr\_rear) | {&&, ##} |
| 201 | <rear> | FIRST(<conditional\_rear>) | FIRST(conditional\_rear) | {==, !=, >, <, >=, <=} |
| 202 | <rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(rear) | {), ;} |
| 203 | <conditional\_rear> | FIRST(<reloptr><cmpr>) | FIRST(reloptr) | {==, !=, >, <, >=, <=} |
| 204 | <conditional\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(conditional\_rear) | {), ;} |
| 205 | <compare> | FIRST(identifier<id\_cmpr> <digit\_optrCmpr>) | FIRST(identifier) | {identifier} |
| 206 | <compare> | FIRST(numlit <digit\_optrCmpr>) | FIRST(numlit) | {numlit} |
| 207 | <compare> | FIRST(decimallit <digit\_optrCmpr>) | FIRST(decimallit | {decimallit} |
| 208 | <compare> | FIRST(setlit) | FIRST(setlit) | {setlit} |
| 209 | <compare> | FIRST(optionlit) | FIRST(optionlit) | {optionlit} |
| 210 | <compare> | FIRST(piecelit) | FIRST(piecelit) | {piecelit} |
| 211 | <logoptr\_rear> | FIRST(<logoptr><logoptr\_add><logoptr\_rear>) | FIRST(logoptr) | {&&, ##} |
| 212 | <logoptr\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(logoptr\_rear) | { ), ; } |
| 213 | <logoptr> | FIRST(&&) | FIRST(&&) | {&&} |
| 214 | <logoptr> | FIRST(##) | FIRST(##) | {##} |
| 215 | <logoptr\_add> | FIRST(<condition>) | FIRST(condition) | {numlit, decimalit, identifier, piecelit, optionlit, setlit, !} |
| 216 | <not1> | FIRST(!) | FIRST(!) | {!} |
| 217 | <not1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(not1) | {(} |
| 218 | <conditional\_optr> | FIRST (<compare\_optr><conditional\_optrear>) | FIRST(compare\_optr) | {numlit, decimallit, identifier, piecelit, setlit, optionlit} |
| 219 | <conditional\_optrear> | FIRST(<reloptr><cmpr><logoptr\_rear>) | FIRST(reloptr) | {==, !=, >, <, >=, <=} |
| 220 | <conditional\_optrear> | FIRST(<logoptr\_rear>) | FIRST(logoptr\_rear) | {&&, ##} |
| 221 | <cmpr> | FIRST(<compare>) | FIRST(compare) | {identifier, numlit, decimallit, piecelit, optionlit, setlit} |
| 222 | <cmpr> | FIRST(<not1>(<compare\_optr>)) | FIRST(not1) | {!} |
| 223 | <compare\_optr> | FIRST(<compare>) | FIRST(compare) | { identifier, numlit, decimallit, piecelit, optionlit, setlit} |
| 224 | <digit\_optrCmpr> | FIRST (<math\_optr><digit\_opMath>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 225 | <digit\_optrCmpr> | FIRST (λ) | FIRST(λ) **U** FOLLOW(digit\_optrCmpr) | {), ==, !=, >, <, >=, <=, &&, ##} |
| 226 | <reloptr> | FIRST(==) | FIRST(==) | {==} |
| 227 | <reloptr> | FIRST(!=) | FIRST(!=) | {!=} |
| 228 | <reloptr> | FIRST(>) | FIRST(>) | {>} |
| 229 | <reloptr> | FIRST(<) | FIRST(<) | {<} |
| 230 | <reloptr> | FIRST(>=) | FIRST(>=) | {>=} |
| 231 | <reloptr> | FIRST(<=) | FIRST(<=) | {<=} |
| 232 | <elsif\_stmt> | FIRST(elsif (<condition>){<statements>}<elsif\_stmt>) | FIRST(elsif) | {elsif} |
| 233 | <elsif\_stmt> | FIRST(λ) | FIRST(λ) **U** FOLLOW(elsif\_stmt) | {else, num, decimal, piece, set, option, identifier, ++, --, tell, check, for, do, while, blank, }} |
| 234 | <else\_stmt> | FIRST(else{<statements>}) | FIRST(else) | {else} |
| 235 | <else\_stmt> | FIRST(λ) | FIRST(λ) **U** FOLLOW(else\_stmt) | {num, decimal, piece, set, option, identifier, ++, --, tell, check, for, do, while, blank, } } |
| 236 | <looping> | FIRST(for(<init>;<condition>;<for\_unary>){<states>}) | FIRST(for) | {for} |
| 237 | <looping> | FIRST(while(<condition>){<states>}) | FIRST(while) | {while} |
| 238 | <looping> | FIRST(do{<states>}while(<condition>);) | FIRST(do) | {do} |
| 239 | <states> | FIRST(<func1><states\_rear>) | FIRST(func1) | { num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, continue, break, blank} |
| 240 | <states\_rear> | FIRST(<states>) | FIRST(states) | { num, decimal, piece, set, option, identifier, ++, --, tell, check, if, for, while, do, continue, break, blank } |
| 241 | <states\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(states\_rear) | { } } |
| 242 | <func1> | FIRST(<datatype> identifier <data\_altr> <id\_add>;) | FIRST(datatype) | { num, decimal, piece, set, option} |
| 243 | <func1> | FIRST(identifier <id\_next\_func>;) | FIRST(identifier) | {identifier} |
| 244 | <func1> | FIRST(<incdec\_optr>identifier;) | FIRST(incdec\_optr) | {++, --} |
| 245 | <func1> | FIRST(<io\_stmt>) | FIRST(io\_stmt) | {tell, check} |
| 246 | <func1> | FIRST(<if\_stmt1>) | FIRST(if\_stmt1) | {if} |
| 247 | <func1> | FIRST(<looping>) | FIRST(looping) | {for, do, while} |
| 248 | <func1> | FIRST(<ctrl>) | FIRST(ctrl) | {break, continue} |
| 249 | <func1> | FIRST(blank;) | FIRST(blank) | {blank} |
| 250 | <if\_stmt1> | FIRST(if(<condition>){<states>}<elsif\_stmt1><else\_stmt1>) | FIRST(if) | {if} |
| 251 | <elsif\_stmt1> | FIRST(elsif (<condition>){<states>}<elsif\_stmt>) | FIRST(elsif) | {elsif} |
| 252 | <elsif\_stmt1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(elsif\_stmt1) | {else, num, decimal, piece, set, option, identifier, ++, --, tell, check, continue, break, for, do, while, blank, }} |
| 253 | <else\_stmt1> | FIRST(else{<states>}) | FIRST(else) | {else} |
| 254 | <else\_stmt1> | FIRST(λ) | FIRST(λ) **U** FOLLOW(else\_stmt1) | {num, decimal, piece, set, option, identifier, ++, --, tell, check, continue, break, for, do, while, blank, }} |
| 255 | <ctrl> | FIRST(continue;) | FIRST(continue) | {continue} |
| 256 | <ctrl> | FIRST(break;) | FIRST(break) | {break} |
| 257 | <init> | FIRST(identifier = <for\_init>) | FIRST(identifier) | {identifier} |
| 258 | <init> | FIRST(λ) | FIRST(λ) **U** FOLLOW(init) | {;} |
| 259 | <for\_unary> | FIRST(identifier<for\_rear>) | FIRST(identifier) | {identifier} |
| 260 | <for\_unary> | FIRST(<incdec\_optr>identifier <for\_digitopMath\_rear>) | FIRST(incdec\_optr) | {++, --} |
| 261 | <for\_rear> | FIRST(<incdec\_optr>) | FIRST(incdec\_optr) | {++, --} |
| 262 | <for\_rear> | FIRST(<ass\_optr> <ass\_opValues1>) | FIRST(ass\_optr) | {=, +=, -=, \*=, /=, %=} |
| 263 | <for\_rear> | FIRST(<math\_optr> <ass\_opValues1>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 264 | <ass\_opValues1> | FIRST(<for\_digitopMath>) | FIRST(for\_digitopMath) | {numlit, identifier} |
| 265 | <for\_digitopMath> | FIRST(numlit <for\_digitopMath\_rear>) | FIRST(numlit ) | {numlit} |
| 266 | <for\_digitopMath> | FIRST(identifier <for\_digitopMath\_rear>) | FIRST(identifier) | {identifier} |
| 267 | <for\_digitopMath\_rear> | FIRST(<math\_optr><for\_digitopMath>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 268 | <for\_digitopMath\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(for\_digitopMath\_rear) | {)} |
| 269 | <str\_con> | FIRST(+ <setlit> <str\_con>) | FIRST(+) | {+} |
| 270 | <str\_con> | FIRST(λ) | FIRST(λ) **U** FOLLOW(str\_con) | { ; } |
| 271 | <array\_altr\_expr> | FIRST(<math\_optr><array\_altr>) | FIRST(math\_optr) | {+, -, \*, /, %} |
| 272 | <array\_altr\_expr> | FIRST(λ) | FIRST(λ) **U** FOLLOW(array\_altr\_expr) | {]} |
| 273 | <incdec\_rear> | FIRST(<incdec\_optr>) | FIRST(incdec\_optr) | {++, --} |
| 274 | <incdec\_rear> | FIRST(λ) | FIRST(λ) **U** FOLLOW(indec\_rear) | {+, -, \*, /, %, )} |

# **PREDICT TABLE**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | blank | break | check | continue | decimal | deck | do | else |
| <program> |  |  |  |  | 1 | 1 |  |  |
| <global> |  |  |  |  | 3 | 4 |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  | 8 |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  | 12 |  |  |  |
| <data\_id> |  |  |  |  | 16 |  |  |  |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> |  |  |  |  |  |  |  |  |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  | 60 |  |  |
| <deck\_cont> |  |  |  |  | 61 |  |  |  |
| <deck\_elem> |  |  |  |  | 62 |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  | 68 |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> | 72 |  | 72 |  | 72 |  | 72 |  |
| <func2> | 82 |  | 79 |  | 74 |  | 81 |  |
| <id\_next\_func> |  |  |  |  |  |  |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> | 84 |  | 84 |  | 84 |  | 84 |  |
| <func\_states\_rear> | 85 |  | 85 |  | 85 |  | 85 |  |
| <func3> | 94 |  | 90 |  | 87 |  | 93 |  |
| <func\_elsif\_stmt> | 96 |  | 96 |  | 96 |  | 96 | 96 |
| <func\_else\_stmt> | 98 |  | 98 |  | 98 |  | 98 | 97 |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <funct\_param\_rtn> |  |  |  |  |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> | 123 |  | 123 |  | 123 |  | 123 |  |
| <func> | 131 |  | 128 |  | 125 |  | 130 |  |
| <id\_next> |  |  |  |  |  |  |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  |  |  |  |  |  |  |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> |  |  |  |  |  |  |  |  |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  |  |  |  |  |  |  |  |
| <id\_funcrear> |  |  |  |  |  |  |  |  |
| <ass\_optr> |  |  |  |  |  |  |  |  |
| <ass\_opValues> |  |  |  |  |  |  |  |  |
| <digit\_opMath> |  |  |  |  |  |  |  |  |
| <digit1> |  |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear> |  |  |  |  |  |  |  |  |
| <id\_cmpr> |  |  |  |  |  |  |  |  |
| <funct\_param> |  |  |  |  |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  | 185 |  |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  |  |  |
| <if\_stmt> |  |  |  |  |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> |  |  |  |  |  |  |  |  |
| <conditional\_rear> |  |  |  |  |  |  |  |  |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> |  |  |  |  |  |  |  |  |
| <cmpr> |  |  |  |  |  |  |  |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  |  |  |  |  |  |  |  |
| <reloptr> |  |  |  |  |  |  |  |  |
| <elsif\_stmt> | 233 |  | 233 |  | 233 |  | 233 | 233 |
| <else\_stmt> | 235 |  | 235 |  | 235 |  | 235 | 234 |
| <looping> |  |  |  |  |  |  | 238 |  |
| <states> | 239 | 239 | 239 | 239 | 239 |  | 239 |  |
| <states\_rear> | 240 | 240 | 240 | 240 | 240 |  | 240 |  |
| <func1> | 249 | 248 | 245 | 248 | 242 |  | 247 |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> | 252 | 252 | 252 | 252 | 252 |  | 252 | 252 |
| <else\_stmt1> | 254 | 254 | 254 | 254 | 254 |  | 254 | 253 |
| <ctrl> |  | 256 |  | 255 |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  |  |  |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  |  |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | elsif | false | for | if | locked | num | option | piece |
| <program> |  |  |  |  | 1 | 1 | 1 | 1 |
| <global> |  |  |  |  | 2 | 3 | 3 | 3 |
| <const\_dec> |  |  |  |  | 7 |  |  |  |
| <const\_dec1> |  |  |  |  |  | 8 | 8 | 8 |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  |  | 11 | 15 | 13 |
| <data\_id> |  |  |  |  |  | 16 | 16 | 16 |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> |  |  |  |  |  |  |  |  |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  | 61 | 61 | 61 |
| <deck\_elem> |  |  |  |  |  | 62 | 62 | 62 |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  | 68 | 68 |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> |  |  | 72 | 72 |  | 72 | 72 | 72 |
| <func2> |  |  | 81 | 80 |  | 74 | 74 | 74 |
| <id\_next\_func> |  |  |  |  |  |  |  |  |
| <func\_if\_stmt> |  |  |  | 83 |  |  |  |  |
| <func\_states> |  |  | 84 | 84 |  | 84 | 84 | 84 |
| <func\_states\_rear> |  |  | 85 | 85 |  | 85 | 85 | 85 |
| <func3> |  |  | 93 | 91 |  | 87 | 87 | 87 |
| <func\_elsif\_stmt> | 95 |  | 96 | 96 |  | 96 | 96 | 96 |
| <func\_else\_stmt> |  |  | 98 | 98 |  | 98 | 98 | 98 |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <funct\_param\_rtn> |  |  |  |  |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  |  | 123 | 123 |  | 123 | 123 | 123 |
| <func> |  |  | 130 | 129 |  | 125 | 125 | 125 |
| <id\_next> |  |  |  |  |  |  |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  |  |  |  |  |  |  |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> |  |  |  |  |  |  |  |  |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  |  |  |  |  |  |  |  |
| <id\_funcrear> |  |  |  |  |  |  |  |  |
| <ass\_optr> |  |  |  |  |  |  |  |  |
| <ass\_opValues> |  |  |  |  |  |  |  |  |
| <digit\_opMath> |  |  |  |  |  |  |  |  |
| <digit1> |  |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear> |  |  |  |  |  |  |  |  |
| <id\_cmpr> |  |  |  |  |  |  |  |  |
| <funct\_param> |  |  |  |  |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  |  |  |
| <if\_stmt> |  |  |  | 197 |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> |  |  |  |  |  |  |  |  |
| <conditional\_rear> |  |  |  |  |  |  |  |  |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> |  |  |  |  |  |  |  |  |
| <cmpr> |  |  |  |  |  |  |  |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  |  |  |  |  |  |  |  |
| <reloptr> |  |  |  |  |  |  |  |  |
| <elsif\_stmt> | 232 |  | 233 |  |  | 233 | 233 | 233 |
| <else\_stmt> |  |  | 235 |  |  | 235 | 235 | 235 |
| <looping> |  |  | 236 |  |  |  |  |  |
| <states> |  |  | 239 | 239 |  | 239 | 239 | 239 |
| <states\_rear> |  |  | 240 | 240 |  | 240 | 240 | 240 |
| <func1> |  |  | 247 | 246 |  | 242 | 242 | 242 |
| <if\_stmt1> |  |  |  | 250 |  |  |  |  |
| <elsif\_stmt1> | 251 |  | 252 |  |  | 252 | 252 | 252 |
| <else\_stmt1> |  |  | 254 |  |  | 254 | 254 | 254 |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  |  |  |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  |  |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | return | set | TABLE | tell | true | void | while | ! |
| <program> |  | 1 | 1 | 1 |  | 1 |  |  |
| <global> |  | 3 | 6 |  |  | 5 |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  | 8 |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  | 14 |  |  |  |  |  |  |
| <data\_id> |  | 16 |  |  |  |  |  |  |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> |  |  |  |  |  |  |  |  |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  | 61 |  |  |  |  |  |  |
| <deck\_elem> |  | 62 |  |  |  |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  | 67 |  |  |
| <param> |  | 68 |  |  |  |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> | 73 | 72 |  | 72 |  |  | 72 |  |
| <func2> |  | 74 |  | 79 |  |  | 81 |  |
| <id\_next\_func> |  |  |  |  |  |  |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> | 84 | 84 |  | 84 |  |  | 84 |  |
| <func\_states\_rear> | 85 | 85 |  | 85 |  |  | 85 |  |
| <func3> | 92 | 87 |  | 90 |  |  | 93 |  |
| <func\_elsif\_stmt> | 96 | 96 |  | 96 |  |  | 96 |  |
| <func\_else\_stmt> | 98 | 98 |  | 98 |  |  | 98 |  |
| <return1> | 99 |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <funct\_param\_rtn> |  |  |  |  |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  | 123 |  | 123 |  |  | 123 |  |
| <func> |  | 125 |  | 128 |  |  | 130 |  |
| <id\_next> |  |  |  |  |  |  |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  |  |  |  |  |  |  |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> |  |  |  |  |  |  |  |  |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  |  |  |  |  |  |  |  |
| <id\_funcrear> |  |  |  |  |  |  |  |  |
| <ass\_optr> |  |  |  |  |  |  |  |  |
| <ass\_opValues> |  |  |  |  |  |  |  |  |
| <digit\_opMath> |  |  |  |  |  |  |  |  |
| <digit1> |  |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear> |  |  |  |  |  |  |  |  |
| <id\_cmpr> |  |  |  |  |  |  |  |  |
| <funct\_param> |  |  |  |  |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  | 184 |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  |  |  |
| <if\_stmt> |  |  |  |  |  |  |  |  |
| <condition> |  |  |  |  |  |  |  | 199 |
| <rear> |  |  |  |  |  |  |  |  |
| <conditional\_rear> |  |  |  |  |  |  |  |  |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  |  | 215 |
| <not1> |  |  |  |  |  |  |  | 216 |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> |  |  |  |  |  |  |  |  |
| <cmpr> |  |  |  |  |  |  |  | 222 |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  |  |  |  |  |  |  |  |
| <reloptr> |  |  |  |  |  |  |  |  |
| <elsif\_stmt> |  | 233 |  | 233 |  |  | 233 |  |
| <else\_stmt> |  | 235 |  | 235 |  |  | 235 |  |
| <looping> |  |  |  |  |  |  | 237 |  |
| <states> |  | 239 |  | 239 |  |  | 239 |  |
| <states\_rear> |  | 240 |  | 240 |  |  | 240 |  |
| <func1> |  | 242 |  | 245 |  |  | 247 |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  | 252 |  | 252 |  |  | 252 |  |
| <else\_stmt1> |  | 254 |  | 254 |  |  | 254 |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  |  |  |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  |  |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | != | + | ++ | += | - | -- | -= | \* |
| <program> |  |  |  |  |  |  |  |  |
| <global> |  |  |  |  |  |  |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  |  |  |  |  |
| <data\_id> |  |  |  |  |  |  |  |  |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  | 51 |  |  | 52 |  |  | 53 |
| <size\_arr> |  | 57 |  | 57 | 57 |  | 57 | 57 |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  |  |  |  |
| <deck\_elem> |  |  |  |  |  |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> |  |  | 72 |  |  | 72 |  |  |
| <func2> |  |  | 78 |  |  | 78 |  |  |
| <id\_next\_func> |  | 76 | 76 |  | 76 | 76 |  | 76 |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> |  |  | 84 |  |  | 84 |  |  |
| <func\_states\_rear> |  |  | 85 |  |  | 85 |  |  |
| <func3> |  |  | 89 |  |  | 89 |  |  |
| <func\_elsif\_stmt> |  |  | 96 |  |  | 96 |  |  |
| <func\_else\_stmt> |  |  | 98 |  |  | 98 |  |  |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  | 108 |  |  | 108 |  |  | 108 |
| <digit\_rear\_rtn> |  | 109 |  |  | 109 |  |  | 109 |
| <funct\_param\_rtn> |  |  |  |  |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  |  | 123 |  |  | 123 |  |  |
| <func> |  |  | 127 |  |  | 127 |  |  |
| <id\_next> |  | 134 | 132 |  | 134 | 132 |  | 134 |
| <incdec\_optr> |  |  | 136 |  |  | 137 |  |  |
| <id\_rear> |  | 139 |  | 139 | 139 |  | 139 | 139 |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> |  | 143 |  | 143 | 143 |  | 143 | 143 |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  | 147 |  | 147 | 147 |  | 147 | 147 |
| <id\_funcrear> |  |  |  | 148 |  |  | 148 |  |
| <ass\_optr> |  |  |  | 150 |  |  | 151 |  |
| <ass\_opValues> |  |  | 155 |  |  | 155 |  |  |
| <digit\_opMath> |  |  | 159 |  |  | 159 |  |  |
| <digit1> |  |  | 162 |  |  | 162 |  |  |
| <digit> |  |  | 166 |  |  | 166 |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  | 172 | 170 |  | 172 | 170 |  | 172 |
| <digit\_rear> | 174 | 173 |  |  | 173 |  |  | 173 |
| <id\_cmpr> | 177 | 177 |  |  | 177 |  |  | 177 |
| <funct\_param> |  |  |  |  |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  | 195 |  |  | 195 |  |  | 195 |
| <if\_stmt> |  |  |  |  |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> | 201 |  |  |  |  |  |  |  |
| <conditional\_rear> | 203 |  |  |  |  |  |  |  |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> | 219 |  |  |  |  |  |  |  |
| <cmpr> |  |  |  |  |  |  |  |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> | 225 | 224 |  |  | 224 |  |  | 224 |
| <reloptr> | 227 |  |  |  |  |  |  |  |
| <elsif\_stmt> |  |  | 233 |  |  | 233 |  |  |
| <else\_stmt> |  |  | 235 |  |  | 235 |  |  |
| <looping> |  |  |  |  |  |  |  |  |
| <states> |  |  | 239 |  |  | 239 |  |  |
| <states\_rear> |  |  | 240 |  |  | 240 |  |  |
| <func1> |  |  | 244 |  |  | 244 |  |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  |  | 252 |  |  | 252 |  |  |
| <else\_stmt1> |  |  | 254 |  |  | 254 |  |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  | 260 |  |  | 260 |  |  |
| <for\_rear> |  | 263 | 261 | 262 | 263 | 261 | 262 | 263 |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  | 267 |  |  | 267 |  |  | 267 |
| <str\_con> |  | 269 |  |  |  |  |  |  |
| <array\_altr\_expr> |  | 271 |  |  | 271 |  |  | 271 |
| <incdec\_rear> |  | 274 | 273 |  | 274 | 273 |  | 274 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | \*= | / | /= | % | %= | && | ## | > |
| <program> |  |  |  |  |  |  |  |  |
| <global> |  |  |  |  |  |  |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  |  |  |  |  |
| <data\_id> |  |  |  |  |  |  |  |  |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  | 54 |  | 55 |  |  |  |  |
| <size\_arr> | 57 | 57 | 57 | 57 | 57 |  |  |  |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  |  |  |  |
| <deck\_elem> |  |  |  |  |  |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> |  |  |  |  |  |  |  |  |
| <func2> |  |  |  |  |  |  |  |  |
| <id\_next\_func> |  | 76 |  | 76 |  |  |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> |  |  |  |  |  |  |  |  |
| <func\_states\_rear> |  |  |  |  |  |  |  |  |
| <func3> |  |  |  |  |  |  |  |  |
| <func\_elsif\_stmt> |  |  |  |  |  |  |  |  |
| <func\_else\_stmt> |  |  |  |  |  |  |  |  |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  | 108 |  | 108 |  |  |  |  |
| <digit\_rear\_rtn> |  | 109 |  | 109 |  |  |  |  |
| <funct\_param\_rtn> |  |  |  |  |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  |  |  |  |  |  |  |  |
| <func> |  |  |  |  |  |  |  |  |
| <id\_next> |  | 134 |  | 134 |  |  |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> | 139 | 139 | 139 | 139 | 139 |  |  |  |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> | 143 | 143 | 143 | 143 | 143 |  |  |  |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> | 147 | 147 | 147 | 147 | 147 |  |  |  |
| <id\_funcrear> | 148 |  | 148 |  | 148 |  |  |  |
| <ass\_optr> | 152 |  | 153 |  | 154 |  |  |  |
| <ass\_opValues> |  |  |  |  |  |  |  |  |
| <digit\_opMath> |  |  |  |  |  |  |  |  |
| <digit1> |  |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  | 172 |  | 172 |  |  |  |  |
| <digit\_rear> |  | 173 |  | 173 |  | 174 | 174 | 174 |
| <id\_cmpr> |  | 177 |  | 177 |  | 177 | 177 | 177 |
| <funct\_param> |  |  |  |  |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  | 195 |  | 195 |  |  |  |  |
| <if\_stmt> |  |  |  |  |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> |  |  |  |  |  | 200 | 200 | 201 |
| <conditional\_rear> |  |  |  |  |  |  |  | 203 |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  | 211 | 211 |  |
| <logoptr> |  |  |  |  |  | 213 | 214 |  |
| <logoptr\_add> |  |  |  |  |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> |  |  |  |  |  |  |  | 219 |
| <cmpr> |  |  |  |  |  | 220 | 220 |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  | 224 |  | 224 |  | 225 | 225 | 225 |
| <reloptr> |  |  |  |  |  |  |  | 228 |
| <elsif\_stmt> |  |  |  |  |  |  |  |  |
| <else\_stmt> |  |  |  |  |  |  |  |  |
| <looping> |  |  |  |  |  |  |  |  |
| <states> |  |  |  |  |  |  |  |  |
| <states\_rear> |  |  |  |  |  |  |  |  |
| <func1> |  |  |  |  |  |  |  |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  |  |  |  |  |  |  |  |
| <else\_stmt1> |  |  |  |  |  |  |  |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  |  |  |  |  |  |  |
| <for\_rear> | 262 | 263 | 262 | 263 | 262 |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  | 267 |  | 267 |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  | 271 |  | 271 |  |  |  |  |
| <incdec\_rear> |  | 274 |  | 274 |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | >= | < | <= | = | == | ; | , | ) |
| <program> |  |  |  |  |  |  |  |  |
| <global> |  |  |  |  |  |  |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  | 10 | 9 |  |
| <datatype> |  |  |  |  |  |  |  |  |
| <data\_id> |  |  |  |  |  |  |  |  |
| <data\_cont> |  |  |  | 17 |  |  |  |  |
| <data\_altr> |  |  |  | 19 |  | 21 | 21 |  |
| <id\_add> |  |  |  |  |  | 23 | 22 |  |
| <var\_dec> |  |  |  | 24 |  | 25 | 25 |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  | 32 | 32 |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  |  | 35 |  | 37 | 37 |  |
| <1d\_arr\_init> |  |  |  | 38 |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  | 40 | 41 |
| <arr\_lit> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  | 46 |  | 47 | 47 |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  | 49 | 50 |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> |  |  |  | 57 |  | 57 | 57 | 57 |
| <size\_arr1> |  |  |  |  |  | 59 |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  |  |  |  |
| <deck\_elem> |  |  |  |  |  |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  | 66 | 65 |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  |  |  | 69 |
| <param1> |  |  |  |  |  |  | 70 | 71 |
| <func\_stmt> |  |  |  |  |  |  |  |  |
| <func2> |  |  |  |  |  |  |  |  |
| <id\_next\_func> |  |  |  |  |  |  |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> |  |  |  |  |  |  |  |  |
| <func\_states\_rear> |  |  |  |  |  |  |  |  |
| <func3> |  |  |  |  |  |  |  |  |
| <func\_elsif\_stmt> |  |  |  |  |  |  |  |  |
| <func\_else\_stmt> |  |  |  |  |  |  |  |  |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  | 101 | 100 |  |
| <return\_choices> |  |  |  |  |  |  |  |  |
| <return\_rear> |  |  |  |  |  | 108 |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  | 110 | 110 | 110 |
| <funct\_param\_rtn> |  |  |  |  |  |  |  | 112 |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  | 113 | 114 |
| <value\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_opMath\_rtn> |  |  |  |  |  |  |  |  |
| <digit1\_rtn> |  |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  |  |  |  |  |  |  |  |
| <func> |  |  |  |  |  |  |  |  |
| <id\_next> |  |  |  |  |  | 135 |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  |  | 139 |  |  | 139 | 139 |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> |  |  |  | 143 |  |  |  | 143 |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  |  |  | 147 |  |  | 147 | 147 |
| <id\_funcrear> |  |  |  | 148 |  |  |  |  |
| <ass\_optr> |  |  |  | 149 |  |  |  |  |
| <ass\_opValues> |  |  |  |  |  |  |  |  |
| <digit\_opMath> |  |  |  |  |  |  |  |  |
| <digit1> |  |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> |  |  |  |  |  |  |  | 172 |
| <digit\_rear> | 174 | 174 | 174 |  | 174 | 174 | 174 | 174 |
| <id\_cmpr> | 177 | 177 | 177 |  | 177 | 177 | 177 | 177 |
| <funct\_param> |  |  |  |  |  |  | 179 |  |
| <func\_param\_rear> |  |  |  |  |  |  |  | 180 |
| <value> |  |  |  |  |  |  |  |  |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> |  |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  | 188 | 189 |
| <digit\_opMath1> |  |  |  |  |  |  |  |  |
| <digit3> |  |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  | 196 | 196 |
| <if\_stmt> |  |  |  |  |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> | 201 | 201 | 201 |  | 201 | 202 |  | 202 |
| <conditional\_rear> | 203 | 203 | 203 |  | 203 | 204 |  | 204 |
| <compare> |  |  |  |  |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  | 212 |  | 212 |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  |  |  |
| <conditional\_optrear> | 219 | 219 | 219 |  | 219 |  |  |  |
| <cmpr> |  |  |  |  |  |  |  |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> | 225 | 225 | 225 |  | 225 |  |  | 225 |
| <reloptr> | 230 | 229 | 231 |  | 226 |  |  |  |
| <elsif\_stmt> |  |  |  |  |  |  |  |  |
| <else\_stmt> |  |  |  |  |  |  |  |  |
| <looping> |  |  |  |  |  |  |  |  |
| <states> |  |  |  |  |  |  |  |  |
| <states\_rear> |  |  |  |  |  |  |  |  |
| <func1> |  |  |  |  |  |  |  |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  |  |  |  |  |  |  |  |
| <else\_stmt1> |  |  |  |  |  |  |  |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  | 258 |  |  |
| <for\_unary> |  |  |  | 262 |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  | 268 |
| <str\_con> |  |  |  |  |  | 270 |  |  |
| <array\_altr\_expr> |  |  |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  | 274 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | ( | ] | [ | } | { | . | piecelit | setlit |
| <program> |  |  |  |  |  |  |  |  |
| <global> |  |  |  |  |  |  |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  |  |  |  |  |
| <data\_id> |  |  |  |  |  |  |  |  |
| <data\_cont> | 18 |  | 17 |  |  |  |  |  |
| <data\_altr> |  |  | 20 |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> |  |  |  |  |  |  | 28 | 29 |
| <1d\_arr> |  |  | 31 |  |  |  |  |  |
| <array\_altr> |  |  |  |  |  |  |  |  |
| <2d\_arr> |  |  | 36 |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> |  |  |  |  |  |  | 39 | 39 |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> |  |  |  |  |  |  | 44 | 45 |
| <2d\_arr\_init> | 48 |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> |  |  | 56 | 57 |  |  | 57 | 57 |
| <size\_arr1> |  |  | 58 |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  |  |  |  |
| <deck\_elem> |  |  |  | 63 |  |  |  |  |
| <deck\_obj> |  |  |  |  |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> |  |  |  |  |  |  |  |  |
| <func2> |  |  |  |  |  |  |  |  |
| <id\_next\_func> | 77 |  | 76 |  |  | 76 |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> |  |  |  |  |  |  |  |  |
| <func\_states\_rear> |  |  |  | 86 |  |  |  |  |
| <func3> |  |  |  |  |  |  |  |  |
| <func\_elsif\_stmt> |  |  |  |  |  |  |  |  |
| <func\_else\_stmt> |  |  |  |  |  |  |  |  |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> |  |  |  |  |  |  |  | 104 |
| <return\_rear> | 107 |  |  |  |  |  |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <funct\_param\_rtn> |  |  |  |  |  |  | 111 | 111 |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> |  |  |  |  |  |  | 115 | 115 |
| <digit\_opMath\_rtn> | 117 |  |  |  |  |  |  |  |
| <digit1\_rtn> | 118 |  |  |  |  |  |  |  |
| <digit\_rtn> |  |  |  |  |  |  |  |  |
| <statements> |  |  |  | 124 |  |  |  |  |
| <func> |  |  |  |  |  |  |  |  |
| <id\_next> |  |  | 133 |  |  | 133 |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  | 138 |  |  | 138 |  |  |
| <id\_choices> |  |  | 141 |  |  | 140 |  |  |
| <elem> |  |  |  |  |  | 142 | 143 | 143 |
| <ins\_arr> |  |  | 144 |  |  |  |  |  |
| <ins\_arr2> |  |  | 145 |  |  | 146 |  |  |
| <id\_funcrear> |  |  |  |  |  |  |  |  |
| <ass\_optr> |  |  |  |  |  |  |  |  |
| <ass\_opValues> | 155 |  |  |  |  |  | 158 | 156 |
| <digit\_opMath> | 160 |  |  |  |  |  |  |  |
| <digit1> | 161 |  |  |  |  |  |  |  |
| <digit> |  |  |  |  |  |  |  |  |
| <digit\_cont> |  |  |  |  |  |  |  |  |
| <choice\_rear> | 171 |  | 171 |  |  | 171 |  |  |
| <digit\_rear> |  |  |  |  |  |  |  |  |
| <id\_cmpr> | 176 |  | 175 |  |  | 175 |  |  |
| <funct\_param> |  |  |  |  |  |  | 178 | 178 |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> |  |  |  |  |  |  | 181 | 181 |
| <elem\_rear> |  |  |  |  |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> | 186 |  |  |  |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  | 187 |
| <digit\_opMath1> | 191 |  |  |  |  |  |  |  |
| <digit3> | 192 |  |  |  |  |  |  |  |
| <digit2> |  |  |  |  |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  |  |  |
| <if\_stmt> |  |  |  |  |  |  | 198 | 198 |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> |  |  |  |  |  |  |  |  |
| <conditional\_rear> |  |  |  |  |  |  |  |  |
| <compare> |  |  |  |  |  |  | 210 | 208 |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> |  |  |  |  |  |  | 215 | 215 |
| <not1> | 217 |  |  |  |  |  |  |  |
| <conditional\_optr> |  |  |  |  |  |  | 218 | 218 |
| <conditional\_optrear> |  |  |  |  |  |  |  |  |
| <cmpr> |  |  |  |  |  |  | 221 | 221 |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> |  |  |  |  |  |  | 223 | 223 |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  |  |  |  |  |  |  |  |
| <reloptr> |  |  |  |  |  |  |  |  |
| <elsif\_stmt> |  |  |  | 233 |  |  |  |  |
| <else\_stmt> |  |  |  | 235 |  |  |  |  |
| <looping> |  |  |  |  |  |  |  |  |
| <states> |  |  |  |  |  |  |  |  |
| <states\_rear> |  |  |  | 241 |  |  |  |  |
| <func1> |  |  |  |  |  |  |  |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  |  |  | 252 |  |  |  |  |
| <else\_stmt1> |  |  |  | 254 |  |  |  |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  |  |  |  |  |  |
| <for\_unary> |  |  |  |  |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> |  |  |  |  |  |  |  |  |
| <for\_digitopMath> |  |  |  |  |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  | 272 |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PRODUCTION | numlit | decimallit | optionlit | identifier |  |  |  |  |
| <program> |  |  |  |  |  |  |  |  |
| <global> |  |  |  |  |  |  |  |  |
| <const\_dec> |  |  |  |  |  |  |  |  |
| <const\_dec1> |  |  |  |  |  |  |  |  |
| <const\_dec2> |  |  |  |  |  |  |  |  |
| <datatype> |  |  |  |  |  |  |  |  |
| <data\_id> |  |  |  |  |  |  |  |  |
| <data\_cont> |  |  |  |  |  |  |  |  |
| <data\_altr> |  |  |  |  |  |  |  |  |
| <id\_add> |  |  |  |  |  |  |  |  |
| <var\_dec> |  |  |  |  |  |  |  |  |
| <var\_dec2> |  |  |  |  |  |  |  |  |
| <literals> | 26 | 27 | 30 |  |  |  |  |  |
| <1d\_arr> |  |  |  |  |  |  |  |  |
| <array\_altr> | 33 |  |  | 34 |  |  |  |  |
| <2d\_arr> |  |  |  |  |  |  |  |  |
| <1d\_arr\_init> |  |  |  |  |  |  |  |  |
| <arr\_elem> | 39 | 39 |  |  |  |  |  |  |
| <arr\_elem1> |  |  |  |  |  |  |  |  |
| <arr\_lit> | 42 | 43 |  |  |  |  |  |  |
| <2d\_arr\_init> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init1> |  |  |  |  |  |  |  |  |
| <2d\_arr\_init2> |  |  |  |  |  |  |  |  |
| <math\_optr> |  |  |  |  |  |  |  |  |
| <size\_arr> | 57 | 57 | 57 | 57 |  |  |  |  |
| <size\_arr1> |  |  |  |  |  |  |  |  |
| <deck\_dec> |  |  |  |  |  |  |  |  |
| <deck\_cont> |  |  |  |  |  |  |  |  |
| <deck\_elem> |  |  |  |  |  |  |  |  |
| <deck\_obj> |  |  |  | 64 |  |  |  |  |
| <deck\_obj1> |  |  |  |  |  |  |  |  |
| <void> |  |  |  |  |  |  |  |  |
| <param> |  |  |  |  |  |  |  |  |
| <param1> |  |  |  |  |  |  |  |  |
| <func\_stmt> |  |  |  | 72 |  |  |  |  |
| <func2> |  |  |  | 75 |  |  |  |  |
| <id\_next\_func> |  |  |  |  |  |  |  |  |
| <func\_if\_stmt> |  |  |  |  |  |  |  |  |
| <func\_states> |  |  |  | 84 |  |  |  |  |
| <func\_states\_rear> |  |  |  | 85 |  |  |  |  |
| <func3> |  |  |  | 88 |  |  |  |  |
| <func\_elsif\_stmt> |  |  |  | 96 |  |  |  |  |
| <func\_else\_stmt> |  |  |  | 98 |  |  |  |  |
| <return1> |  |  |  |  |  |  |  |  |
| <return\_add> |  |  |  |  |  |  |  |  |
| <return\_choices> | 102 | 103 | 105 | 106 |  |  |  |  |
| <return\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <funct\_param\_rtn> | 111 | 111 | 111 | 111 |  |  |  |  |
| <func\_param\_rear\_rtn> |  |  |  |  |  |  |  |  |
| <value\_rtn> | 115 | 115 | 115 | 116 |  |  |  |  |
| <digit\_opMath\_rtn> | 117 | 117 |  | 117 |  |  |  |  |
| <digit1\_rtn> | 119 | 119 |  | 119 |  |  |  |  |
| <digit\_rtn> | 120 | 121 |  | 122 |  |  |  |  |
| <statements> |  |  |  | 123 |  |  |  |  |
| <func> |  |  |  | 126 |  |  |  |  |
| <id\_next> |  |  |  |  |  |  |  |  |
| <incdec\_optr> |  |  |  |  |  |  |  |  |
| <id\_rear> |  |  |  |  |  |  |  |  |
| <id\_choices> |  |  |  |  |  |  |  |  |
| <elem> | 143 | 143 | 143 | 143 |  |  |  |  |
| <ins\_arr> |  |  |  |  |  |  |  |  |
| <ins\_arr2> |  |  |  |  |  |  |  |  |
| <id\_funcrear> |  |  |  |  |  |  |  |  |
| <ass\_optr> |  |  |  |  |  |  |  |  |
| <ass\_opValues> | 155 | 155 | 158 | 155 |  |  |  |  |
| <digit\_opMath> | 159 | 159 |  | 159 |  |  |  |  |
| <digit1> | 162 | 162 |  | 162 |  |  |  |  |
| <digit> | 163 | 164 |  | 165 |  |  |  |  |
| <digit\_cont> | 167 | 168 |  | 169 |  |  |  |  |
| <choice\_rear> |  |  |  |  |  |  |  |  |
| <digit\_rear> |  |  |  |  |  |  |  |  |
| <id\_cmpr> |  |  |  |  |  |  |  |  |
| <funct\_param> | 178 | 178 | 178 | 178 |  |  |  |  |
| <func\_param\_rear> |  |  |  |  |  |  |  |  |
| <value> | 181 | 181 | 181 | 182 |  |  |  |  |
| <elem\_rear> |  |  |  | 183 |  |  |  |  |
| <io\_stmt> |  |  |  |  |  |  |  |  |
| <output> |  |  |  | 186 |  |  |  |  |
| <output\_rear> |  |  |  |  |  |  |  |  |
| <digit\_opMath1> |  |  |  | 190 |  |  |  |  |
| <digit3> |  |  |  | 193 |  |  |  |  |
| <digit2> |  |  |  | 194 |  |  |  |  |
| <digit\_rear1> |  |  |  |  |  |  |  |  |
| <if\_stmt> | 198 | 198 | 198 | 198 |  |  |  |  |
| <condition> |  |  |  |  |  |  |  |  |
| <rear> |  |  |  |  |  |  |  |  |
| <conditional\_rear> |  |  |  |  |  |  |  |  |
| <compare> | 206 | 207 | 209 | 205 |  |  |  |  |
| <logoptr\_rear> |  |  |  |  |  |  |  |  |
| <logoptr> |  |  |  |  |  |  |  |  |
| <logoptr\_add> | 215 | 215 | 215 | 215 |  |  |  |  |
| <not1> |  |  |  |  |  |  |  |  |
| <conditional\_optr> | 218 | 218 | 218 | 218 |  |  |  |  |
| <conditional\_optrear> |  |  |  |  |  |  |  |  |
| <cmpr> | 221 | 221 | 221 | 221 |  |  |  |  |
| <compare\_rear> |  |  |  |  |  |  |  |  |
| <compare\_optr> | 223 | 223 | 223 | 223 |  |  |  |  |
| <digit\_id> |  |  |  |  |  |  |  |  |
| <digit\_optrCmpr> |  |  |  |  |  |  |  |  |
| <reloptr> |  |  |  |  |  |  |  |  |
| <elsif\_stmt> |  |  |  | 233 |  |  |  |  |
| <else\_stmt> |  |  |  | 235 |  |  |  |  |
| <looping> |  |  |  |  |  |  |  |  |
| <states> |  |  |  | 239 |  |  |  |  |
| <states\_rear> |  |  |  | 240 |  |  |  |  |
| <func1> |  |  |  | 243 |  |  |  |  |
| <if\_stmt1> |  |  |  |  |  |  |  |  |
| <elsif\_stmt1> |  |  |  | 252 |  |  |  |  |
| <else\_stmt1> |  |  |  | 254 |  |  |  |  |
| <ctrl> |  |  |  |  |  |  |  |  |
| <init> |  |  |  | 257 |  |  |  |  |
| <for\_unary> |  |  |  | 259 |  |  |  |  |
| <for\_rear> |  |  |  |  |  |  |  |  |
| <ass\_opValues1> | 264 |  |  | 264 |  |  |  |  |
| <for\_digitopMath> | 265 |  |  | 266 |  |  |  |  |
| <for\_digitopMath\_rear> |  |  |  |  |  |  |  |  |
| <str\_con> |  |  |  |  |  |  |  |  |
| <array\_altr\_expr> |  |  |  |  |  |  |  |  |
| <incdec\_rear> |  |  |  |  |  |  |  |  |

# **Appendix A - Lexical Test Scripts**

## **Reserved Words**

|  |  |  |  |
| --- | --- | --- | --- |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| blank⎵ | No Lexical Error | No Lexical Error | OK |
| blank**↵** | No Lexical Error | No Lexical Error | OK |
| blank; | No Lexical Error | No Lexical Error | OK |
| blank` | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘’’ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘’’ | OK |
| blank@ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘@’ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘@’ | OK |
| blank# | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘#’ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘#’ | OK |
| blank$ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘$’ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘$’ | OK |
| blank^ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘^’ | LEXICAL ERROR: On ‘blank’ Invalid Delimiter ‘^’ | OK |
| break⎵ | No Lexical Error | No Lexical Error | OK |
| break**↵** | No Lexical Error | No Lexical Error | OK |
| break; | No Lexical Error | No Lexical Error | OK |
| break^ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘^’ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘^’ | OK |
| break& | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘&’ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘&’ | OK |
|  |  |  |  |
| break\* | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘\*’ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘\*’ | OK |
| break) | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘)’ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘)’ | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| break( | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘(’ | LEXICAL ERROR: On ‘break’ Invalid Delimiter ‘(’ | OK |
| check⎵ | No Lexical Error | No Lexical Error | OK |
| check**↵** | No Lexical Error | No Lexical Error | OK |
| check( | No Lexical Error | No Lexical Error | OK |
| check- | LEXICAL ERROR: On ‘check’ Invalid Delimiter ‘-‘ | LEXICAL ERROR: On ‘check’ Invalid Delimiter ‘-‘ | OK |
| check# | LEXICAL ERROR: On ‘check’ Invalid Delimiter ‘#‘ | LEXICAL ERROR: On ‘check’ Invalid Delimiter ‘#‘ | OK |
| check+ | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘+‘ | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘+‘ | OK |
| check$ | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘$‘ | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘$‘ | OK |
| check! | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘!‘ | LEXICAL ERROR: On ‘check’  Invalid Delimiter ‘!‘ | OK |
| continue⎵ | No Lexical Error | No Lexical Error | OK |
| continue**↵** | No Lexical Error | No Lexical Error | OK |
| continue; | No Lexical Error | No Lexical Error | OK |
| continue# | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘#‘ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘#‘ | OK |
| continue@ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘@‘ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘@‘ | OK |
| continue\* | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘\*‘ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘\*‘ | OK |
| continue” | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘”‘ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘”‘ | OK |
| continue+ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘+‘ | LEXICAL ERROR: On ‘continue’ Invalid Delimiter ‘+‘ | OK |
| decimal⎵ | No Lexical Error | No Lexical Error | OK |
| decimal**↵** | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| decimal~ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘~‘ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘~‘ | OK |
|  |  |  |  |
| decimal\* | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘\*‘ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘\*‘ | OK |
| decimal= | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘=‘ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘=‘ | OK |
| decimal& | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘&‘ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘&‘ | OK |
| decimal? | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘?‘ | LEXICAL ERROR: On ‘decimal’ Invalid Delimiter ‘?‘ | OK |
| deck⎵ | No Lexical Error | No Lexical Error | OK |
| deck**↵** | No Lexical Error | No Lexical Error | OK |
| deck{ | LEXICAL ERROR: On 'deck', Invalid Delimiter '{' | LEXICAL ERROR: On 'deck', Invalid Delimiter '{' | OK |
| deck+ | LEXICAL ERROR: On 'deck', Invalid Delimiter '+' | LEXICAL ERROR: On 'deck', Invalid Delimiter '+' | OK |
| deck= | LEXICAL ERROR: On 'deck', Invalid Delimiter '=' | LEXICAL ERROR: On 'deck', Invalid Delimiter '=' | OK |
| deck@ | LEXICAL ERROR: On 'deck', Invalid Delimiter '@' | LEXICAL ERROR: On 'deck', Invalid Delimiter '@' | OK |
| deck! | LEXICAL ERROR: On 'deck', Invalid Delimiter '!' | LEXICAL ERROR: On 'deck', Invalid Delimiter '!' | OK |
| deck^ | LEXICAL ERROR: On 'deck', Invalid Delimiter '^' | LEXICAL ERROR: On 'deck', Invalid Delimiter '^' | OK |
| do⎵ | No Lexical Error | No Lexical Error | OK |
| do**↵** | No Lexical Error | No Lexical Error | OK |
| do{ | No Lexical Error | No Lexical Error | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| do+ | LEXICAL ERROR: On 'do', Invalid Delimiter '+' | LEXICAL ERROR: On 'do', Invalid Delimiter '+' | OK |
| do) | LEXICAL ERROR: On 'do', Invalid Delimiter ')' | LEXICAL ERROR: On 'do', Invalid Delimiter ')' | OK |
| do\* | LEXICAL ERROR: On 'do', Invalid Delimiter '\*' | LEXICAL ERROR: On 'do', Invalid Delimiter '\*' | OK |
| do` | LEXICAL ERROR: On 'do', Invalid Delimiter '’' | LEXICAL ERROR: On 'do', Invalid Delimiter '’' | OK |
| do; | LEXICAL ERROR: On 'do', Invalid Delimiter ';' | LEXICAL ERROR: On 'do', Invalid Delimiter ';' | OK |
| else⎵ | No Lexical Error | No Lexical Error | OK |
| else**↵** | No Lexical Error | No Lexical Error | OK |
| else{ | No Lexical Error | No Lexical Error | OK |
| else$ | LEXICAL ERROR: On 'else', Invalid Delimiter '$' | LEXICAL ERROR: On 'else', Invalid Delimiter '$' | OK |
| else& | LEXICAL ERROR: On 'else', Invalid Delimiter '&' | LEXICAL ERROR: On 'else', Invalid Delimiter '&' | OK |
| else( | LEXICAL ERROR: On 'else', Invalid Delimiter '(' | LEXICAL ERROR: On 'else', Invalid Delimiter '(' | OK |
| else# | LEXICAL ERROR: On 'else', Invalid Delimiter '#' | LEXICAL ERROR: On 'else', Invalid Delimiter '#' | OK |
| else} | LEXICAL ERROR: On 'else', Invalid Delimiter '}' | LEXICAL ERROR: On 'else', Invalid Delimiter '}' | OK |
| elsif⎵ | No Lexical Error | No Lexical Error | OK |
| elsif**↵** | No Lexical Error | No Lexical Error | OK |
| elsif( | No Lexical Error | No Lexical Error | OK |
| elsif/ | LEXICAL ERROR: On 'elsif', Invalid Delimiter '/' | LEXICAL ERROR: On 'elsif', Invalid Delimiter '/' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| elsif& | LEXICAL ERROR: On 'elsif', Invalid Delimiter '&' | LEXICAL ERROR: On 'elsif', Invalid Delimiter '&' | OK |
| elsif{ | LEXICAL ERROR: On 'elsif', Invalid Delimiter '{' | LEXICAL ERROR: On 'elsif', Invalid Delimiter '{' | OK |
| elsif^ | LEXICAL ERROR: On 'elsif', Invalid Delimiter '^' | LEXICAL ERROR: On 'elsif', Invalid Delimiter '^' | OK |
| elsif! | LEXICAL ERROR: On 'elsif', Invalid Delimiter '!' | LEXICAL ERROR: On 'elsif', Invalid Delimiter '!' | OK |
| false⎵ | No Lexical Error | No Lexical Error | OK |
| false**↵** | No Lexical Error | No Lexical Error | OK |
| false, | No Lexical Error | No Lexical Error | OK |
| false; | No Lexical Error | No Lexical Error | OK |
| false) | No Lexical Error | No Lexical Error | OK |
| false& | No Lexical Error | No Lexical Error | OK |
| false# | No Lexical Error | No Lexical Error | OK |
| false! | LEXICAL ERROR: On ‘false’, Invalid Delimiter '!' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '!' | OK |
| false( | LEXICAL ERROR: On ‘false’, Invalid Delimiter '(' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '(' | OK |
| false{ | LEXICAL ERROR: On ‘false’, Invalid Delimiter '{' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '{' | OK |
| false@ | LEXICAL ERROR: On ‘false’, Invalid Delimiter '@' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '@' | OK |
| false+ | LEXICAL ERROR: On ‘false’, Invalid Delimiter '+' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '+' | OK |
| for⎵ | No Lexical Error | No Lexical Error | OK |
| for**↵** | No Lexical Error | No Lexical Error | OK |
| for( | No Lexical Error | No Lexical Error | OK |
| for^ | LEXICAL ERROR: On ‘false’, Invalid Delimiter '^' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '^' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| for= | LEXICAL ERROR: On ‘false’, Invalid Delimiter '=' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '=' | OK |
| for) | LEXICAL ERROR: On ‘false’, Invalid Delimiter ')' | LEXICAL ERROR: On ‘false’, Invalid Delimiter ')' | OK |
| for{ | LEXICAL ERROR: On ‘false’, Invalid Delimiter '{' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '{' | OK |
| for% | LEXICAL ERROR: On ‘false’, Invalid Delimiter '%' | LEXICAL ERROR: On ‘false’, Invalid Delimiter '%' | OK |
| if⎵ | No Lexical Error | No Lexical Error | OK |
| if**↵** | No Lexical Error | No Lexical Error | OK |
| if( | No Lexical Error | No Lexical Error | OK |
| if) | LEXICAL ERROR: On 'if', Invalid Delimiter ')' | LEXICAL ERROR: On 'if', Invalid Delimiter ')' | OK |
| if! | LEXICAL ERROR: On 'if', Invalid Delimiter '!' | LEXICAL ERROR: On 'if', Invalid Delimiter '!' | OK |
| if[ | LEXICAL ERROR: On 'if', Invalid Delimiter '[' | LEXICAL ERROR: On 'if', Invalid Delimiter '[' | OK |
| if| | LEXICAL ERROR: On 'if', Invalid Delimiter '|' | LEXICAL ERROR: On 'if', Invalid Delimiter '|' | OK |
| if’ | LEXICAL ERROR: On 'if', Invalid Delimiter '’' | LEXICAL ERROR: On 'if', Invalid Delimiter '’' | OK |
| locked⎵ | No Lexical Error | No Lexical Error | OK |
| locked**↵** | No Lexical Error | No Lexical Error | OK |
| locked? | LEXICAL ERROR: On 'locked', Invalid Delimiter '?' | LEXICAL ERROR: On 'locked', Invalid Delimiter '?' | OK |
| locked= | LEXICAL ERROR: On 'locked', Invalid Delimiter '=' | LEXICAL ERROR: On 'locked', Invalid Delimiter '=' | OK |
| locked# | LEXICAL ERROR: On 'locked', Invalid Delimiter '#' | LEXICAL ERROR: On 'locked', Invalid Delimiter '#' | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| locked, | LEXICAL ERROR: On 'locked', Invalid Delimiter ',' | LEXICAL ERROR: On 'locked', Invalid Delimiter ',' | OK |
| locked/ | LEXICAL ERROR: On 'locked', Invalid Delimiter '/' | LEXICAL ERROR: On 'locked', Invalid Delimiter '/' | OK |
| num⎵ | No Lexical Error | No Lexical Error | OK |
| num**↵** | No Lexical Error | No Lexical Error | OK |
| num- | LEXICAL ERROR: On 'num', Invalid Delimiter '-' | LEXICAL ERROR: On 'num', Invalid Delimiter '-' | OK |
| num\* | LEXICAL ERROR: On 'num', Invalid Delimiter '\*' | LEXICAL ERROR: On 'num', Invalid Delimiter '\*' | OK |
| num$ | LEXICAL ERROR: On 'num', Invalid Delimiter '$' | LEXICAL ERROR: On 'num', Invalid Delimiter '$' | OK |
| num^ | LEXICAL ERROR: On 'num', Invalid Delimiter '^' | LEXICAL ERROR: On 'num', Invalid Delimiter '^' | OK |
| num& | LEXICAL ERROR: On 'num', Invalid Delimiter '&' | LEXICAL ERROR: On 'num', Invalid Delimiter '&' | OK |
| option⎵ | No Lexical Error | No Lexical Error | OK |
| option**↵** | No Lexical Error | No Lexical Error | OK |
| option> | LEXICAL ERROR: On 'option', Invalid Delimiter '>' | LEXICAL ERROR: On 'option', Invalid Delimiter '>' | OK |
| option? | LEXICAL ERROR: On 'option', Invalid Delimiter '?' | LEXICAL ERROR: On 'option', Invalid Delimiter '?' | OK |
| option: | LEXICAL ERROR: On 'option', Invalid Delimiter ':' | LEXICAL ERROR: On 'option', Invalid Delimiter ':' | OK |
| option; | LEXICAL ERROR: On 'option', Invalid Delimiter ';' | LEXICAL ERROR: On 'option', Invalid Delimiter ';' | OK |
| option\ | LEXICAL ERROR: On 'option', Invalid Delimiter '\' | LEXICAL ERROR: On 'option', Invalid Delimiter '\' | OK |
| piece⎵ | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| piece**↵** | No Lexical Error | No Lexical Error | OK |
| piece[ | LEXICAL ERROR: On 'piece', Invalid Delimiter '[' | LEXICAL ERROR: On 'piece', Invalid Delimiter '[' | OK |
| piece# | LEXICAL ERROR: On 'piece', Invalid Delimiter '#' | LEXICAL ERROR: On 'piece', Invalid Delimiter '#' | OK |
| piece^ | LEXICAL ERROR: On 'piece', Invalid Delimiter '^' | LEXICAL ERROR: On 'piece', Invalid Delimiter '^' | OK |
| piece% | LEXICAL ERROR: On 'piece', Invalid Delimiter '%' | LEXICAL ERROR: On 'piece', Invalid Delimiter '%' | OK |
| piece( | LEXICAL ERROR: On 'piece', Invalid Delimiter '(' | LEXICAL ERROR: On 'piece', Invalid Delimiter '(' | OK |
| return⎵ | No Lexical Error | No Lexical Error | OK |
| return**↵** | No Lexical Error | No Lexical Error | OK |
| return; | No Lexical Error | No Lexical Error | OK |
| return= | LEXICAL ERROR: On 'return', Invalid Delimiter '=' | LEXICAL ERROR: On 'return', Invalid Delimiter '=' | OK |
| return# | LEXICAL ERROR: On 'return', Invalid Delimiter '#' | LEXICAL ERROR: On 'return', Invalid Delimiter '#' | OK |
| return, | LEXICAL ERROR: On 'return', Invalid Delimiter ',' | LEXICAL ERROR: On 'return', Invalid Delimiter ',' | OK |
| return| | LEXICAL ERROR: On 'return', Invalid Delimiter '|' | LEXICAL ERROR: On 'return', Invalid Delimiter '|' | OK |
| return! | LEXICAL ERROR: On 'return', Invalid Delimiter '!' | LEXICAL ERROR: On 'return', Invalid Delimiter '!' | OK |
| set⎵ | No Lexical Error | No Lexical Error | OK |
| set**↵** | No Lexical Error | No Lexical Error | OK |
| set/ | LEXICAL ERROR: On 'set', Invalid Delimiter '/' | LEXICAL ERROR: On 'set', Invalid Delimiter '/' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| set] | LEXICAL ERROR: On 'set', Invalid Delimiter ']' | LEXICAL ERROR: On 'set', Invalid Delimiter ']' | OK |
| set} | LEXICAL ERROR: On 'set', Invalid Delimiter '}' | LEXICAL ERROR: On 'set', Invalid Delimiter '}' | OK |
| set( | LEXICAL ERROR: On 'set', Invalid Delimiter '(' | LEXICAL ERROR: On 'set', Invalid Delimiter '(' | OK |
| set@ | LEXICAL ERROR: On 'set', Invalid Delimiter '@' | LEXICAL ERROR: On 'set', Invalid Delimiter '@' | OK |
| TABLE⎵ | No Lexical Error | No Lexical Error | OK |
| TABLE**↵** | No Lexical Error | No Lexical Error | OK |
| TABLE( | No Lexical Error | No Lexical Error | OK |
| TABLE; | LEXICAL ERROR: On 'TABLE', Invalid Delimiter ';' | LEXICAL ERROR: On 'TABLE', Invalid Delimiter ';' | OK |
| TABLE) | LEXICAL ERROR: On 'TABLE', Invalid Delimiter ')' | LEXICAL ERROR: On 'TABLE', Invalid Delimiter ')' | OK |
| TABLE@ | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '@' | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '@' | OK |
| TABLE# | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '#' | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '#' | OK |
| TABLE- | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '-' | LEXICAL ERROR: On 'TABLE', Invalid Delimiter '-' | OK |
| tell⎵ | No Lexical Error | No Lexical Error | OK |
| tell**↵** | No Lexical Error | No Lexical Error | OK |
| tell( | No Lexical Error | No Lexical Error | OK |
| tell$ | LEXICAL ERROR: On 'tell', Invalid Delimiter '$' | LEXICAL ERROR: On 'tell', Invalid Delimiter '$' | OK |
| tell) | LEXICAL ERROR: On 'tell', Invalid Delimiter ')' | LEXICAL ERROR: On 'tell', Invalid Delimiter ')' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| tell{ | LEXICAL ERROR: On 'tell', Invalid Delimiter '{' | LEXICAL ERROR: On 'tell', Invalid Delimiter '{' | OK |
| tell’ | LEXICAL ERROR: On 'tell', Invalid Delimiter '’' | LEXICAL ERROR: On 'tell', Invalid Delimiter '’' | OK |
| tell” | LEXICAL ERROR: On 'tell', Invalid Delimiter '"' | LEXICAL ERROR: On 'tell', Invalid Delimiter '"' | OK |
| true⎵ | No Lexical Error | No Lexical Error | OK |
| true**↵** | No Lexical Error | No Lexical Error | OK |
| true, | No Lexical Error | No Lexical Error | OK |
| true; | No Lexical Error | No Lexical Error | OK |
| true) | No Lexical Error | No Lexical Error | OK |
| true& | No Lexical Error | No Lexical Error | OK |
| true# | No Lexical Error | No Lexical Error | OK |
| true\* | LEXICAL ERROR: On 'true', Invalid Delimiter '\*' | LEXICAL ERROR: On 'true', Invalid Delimiter '\*' | OK |
| true+ | LEXICAL ERROR: On 'true', Invalid Delimiter '+' | LEXICAL ERROR: On 'true', Invalid Delimiter '+' | OK |
| true{ | LEXICAL ERROR: On 'true', Invalid Delimiter '{' | LEXICAL ERROR: On 'true', Invalid Delimiter '{' | OK |
| true| | LEXICAL ERROR: On 'true', Invalid Delimiter '|' | LEXICAL ERROR: On 'true', Invalid Delimiter '|' | OK |
| true~ | LEXICAL ERROR: On 'true', Invalid Delimiter '~' | LEXICAL ERROR: On 'true', Invalid Delimiter '~' | OK |
| void⎵ | No Lexical Error | No Lexical Error | OK |
| void**↵** | No Lexical Error | No Lexical Error | OK |
| void^ | LEXICAL ERROR: On 'void', Invalid Delimiter '^' | LEXICAL ERROR: On 'void', Invalid Delimiter '^' | OK |
| void; | LEXICAL ERROR: On 'void', Invalid Delimiter ';' | LEXICAL ERROR: On 'void', Invalid Delimiter ';' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| void: | LEXICAL ERROR: On 'void', Invalid Delimiter ':' | LEXICAL ERROR: On 'void', Invalid Delimiter ':' | OK |
| void} | LEXICAL ERROR: On 'void', Invalid Delimiter '}' | LEXICAL ERROR: On 'void', Invalid Delimiter '}' | OK |
| void( | LEXICAL ERROR: On 'void', Invalid Delimiter '(' | LEXICAL ERROR: On 'void', Invalid Delimiter '(' | OK |
| while⎵ | No Lexical Error | No Lexical Error | OK |
| while**↵** | No Lexical Error | No Lexical Error | OK |
| while( | No Lexical Error | No Lexical Error | OK |
| while? | LEXICAL ERROR: On 'while', Invalid Delimiter '?' | LEXICAL ERROR: On 'while', Invalid Delimiter '?' | OK |
| while) | LEXICAL ERROR: On 'while', Invalid Delimiter ')' | LEXICAL ERROR: On 'while', Invalid Delimiter ')' | OK |
| while{ | LEXICAL ERROR: On 'while', Invalid Delimiter '{' | LEXICAL ERROR: On 'while', Invalid Delimiter '{' | OK |
| while+ | LEXICAL ERROR: On 'while', Invalid Delimiter '+' | LEXICAL ERROR: On 'while', Invalid Delimiter '+' | OK |
| while^ | LEXICAL ERROR: On 'while', Invalid Delimiter '^' | LEXICAL ERROR: On 'while', Invalid Delimiter '^' | OK |
|  |  |  |  |

## **Reserved Symbols**

*LEGEND: (lowercase = a-z) (numbers = 1,2,3,4,5,6,7,8,9,0)*

|  |  |  |  |
| --- | --- | --- | --- |
| !⎵ | No Lexical Error | No Lexical Error | OK |
| !**↵** | No Lexical Error | No Lexical Error | OK |
| !( | No Lexical Error | No Lexical Error | OK |
| !# | LEXICAL ERROR: On '!', Invalid Delimiter '#' | LEXICAL ERROR: On '!', Invalid Delimiter '#' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| !) | LEXICAL ERROR: On '!', Invalid Delimiter ')' | LEXICAL ERROR: On '!', Invalid Delimiter ')' | OK |
| !% | LEXICAL ERROR: On '!', Invalid Delimiter '%' | LEXICAL ERROR: On '!', Invalid Delimiter '%' | OK |
| !@ | LEXICAL ERROR: On '!', Invalid Delimiter '@' | LEXICAL ERROR: On '!', Invalid Delimiter '@' | OK |
| !- | LEXICAL ERROR: On '!', Invalid Delimiter '-' | LEXICAL ERROR: On '!', Invalid Delimiter '-' | OK |
| !=⎵ | No Lexical Error | No Lexical Error | OK |
| !=**↵** | No Lexical Error | No Lexical Error | OK |
| !=( | No Lexical Error | No Lexical Error | OK |
| !=(lowercase) | No Lexical Error | No Lexical Error | OK |
| !=(numbers) | No Lexical Error | No Lexical Error | OK |
| !=~ | No Lexical Error | No Lexical Error | OK |
| !=” | No Lexical Error | No Lexical Error | OK |
| !=’ | No Lexical Error | No Lexical Error | OK |
| !={ | LEXICAL ERROR: On '!=', Invalid Delimiter '{' | LEXICAL ERROR: On '!=', Invalid Delimiter '{' | OK |
| !=^ | LEXICAL ERROR: On '!=', Invalid Delimiter '^' | LEXICAL ERROR: On '!=', Invalid Delimiter '^' | OK |
| !=+ | LEXICAL ERROR: On '!=', Invalid Delimiter '+' | LEXICAL ERROR: On '!=', Invalid Delimiter '+' | OK |
| !=) | LEXICAL ERROR: On '!=', Invalid Delimiter ')' | LEXICAL ERROR: On '!=', Invalid Delimiter ')' | OK |
| !=[ | LEXICAL ERROR: On '!=', Invalid Delimiter '[' | LEXICAL ERROR: On '!=', Invalid Delimiter '[' | OK |
| +⎵ | No Lexical Error | No Lexical Error | OK |
| +**↵** | No Lexical Error | No Lexical Error | OK |
| +( | No Lexical Error | No Lexical Error | OK |
| +(lowercase) | No Lexical Error | No Lexical Error | OK |
| +(numbers) | No Lexical Error | No Lexical Error | OK |
| +~ | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| +” | No Lexical Error | No Lexical Error | OK |
| +- | LEXICAL ERROR: On '+', Invalid Delimiter '-' | LEXICAL ERROR: On '+', Invalid Delimiter '-' | OK |
| +% | LEXICAL ERROR: On '+', Invalid Delimiter '%' | LEXICAL ERROR: On '+', Invalid Delimiter '%' | OK |
| +\_ | LEXICAL ERROR: On '+', Invalid Delimiter '\_' | LEXICAL ERROR: On '+', Invalid Delimiter '\_' | OK |
| +[ | LEXICAL ERROR: On '+', Invalid Delimiter '[' | LEXICAL ERROR: On '+', Invalid Delimiter '[' | OK |
| +{ | LEXICAL ERROR: On '+', Invalid Delimiter '{' | LEXICAL ERROR: On '+', Invalid Delimiter '{' | OK |
| ++⎵ | No Lexical Error | No Lexical Error | OK |
| ++**↵** | No Lexical Error | No Lexical Error | OK |
| ++; | No Lexical Error | No Lexical Error | OK |
| ++(lowercase) | No Lexical Error | No Lexical Error | OK |
| ++) | No Lexical Error | No Lexical Error | OK |
| ++(numbers) | No Lexical Error | No Lexical Error | OK |
| ++$ | LEXICAL ERROR: On '++', Invalid Delimiter '$' | LEXICAL ERROR: On '++', Invalid Delimiter '$' | OK |
| ++\* | LEXICAL ERROR: On '++', Invalid Delimiter '\*' | LEXICAL ERROR: On '++', Invalid Delimiter '\*' | OK |
| ++^ | LEXICAL ERROR: On '++', Invalid Delimiter '^' | LEXICAL ERROR: On '++', Invalid Delimiter '^' | OK |
| ++` | LEXICAL ERROR: On '++', Invalid Delimiter '’' | LEXICAL ERROR: On '++', Invalid Delimiter '’' | OK |
| ++| | LEXICAL ERROR: On '++', Invalid Delimiter '|' | LEXICAL ERROR: On '++', Invalid Delimiter '|' | OK |
| +=⎵ | No Lexical Error | No Lexical Error | OK |
| +=**↵** | No Lexical Error | No Lexical Error | OK |
| +=( | No Lexical Error | No Lexical Error | OK |
| +=(lowercase) | No Lexical Error | No Lexical Error | OK |
| +=(numbers) | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| +=~ | No Lexical Error | No Lexical Error | OK |
| +== | LEXICAL ERROR: On '+=', Invalid Delimiter '=' | LEXICAL ERROR: On '+=', Invalid Delimiter '=' | OK |
| +=& | LEXICAL ERROR: On '+=', Invalid Delimiter ' &' | LEXICAL ERROR: On '+=', Invalid Delimiter ' &' | OK |
| +=) | LEXICAL ERROR: On '+=', Invalid Delimiter ')' | LEXICAL ERROR: On '+=', Invalid Delimiter ')' | OK |
| +=[ | LEXICAL ERROR: On '+=', Invalid Delimiter '[' | LEXICAL ERROR: On '+=', Invalid Delimiter '[' | OK |
| +=@ | LEXICAL ERROR: On '+=', Invalid Delimiter '@' | LEXICAL ERROR: On '+=', Invalid Delimiter '@' | OK |
| -⎵ | No Lexical Error | No Lexical Error | OK |
| -**↵** | No Lexical Error | No Lexical Error | OK |
| -( | No Lexical Error | No Lexical Error | OK |
| -(lowercase) | No Lexical Error | No Lexical Error | OK |
| -(numbers) | No Lexical Error | No Lexical Error | OK |
| -~ | No Lexical Error | No Lexical Error | OK |
| -\_ | LEXICAL ERROR: On '-', Invalid Delimiter '\_' | LEXICAL ERROR: On '-', Invalid Delimiter '\_' | OK |
| -+ | LEXICAL ERROR: On '-', Invalid Delimiter '+' | LEXICAL ERROR: On '-', Invalid Delimiter '+' | OK |
| -! | LEXICAL ERROR: On '-', Invalid Delimiter '!' | LEXICAL ERROR: On '-', Invalid Delimiter '!' | OK |
| -; | LEXICAL ERROR: On '-', Invalid Delimiter ';' | LEXICAL ERROR: On '-', Invalid Delimiter ';' | OK |
| -{ | LEXICAL ERROR: On '-', Invalid Delimiter '{' | LEXICAL ERROR: On '-', Invalid Delimiter '{' | OK |
| --⎵ | No Lexical Error | No Lexical Error | OK |
| --**↵** | No Lexical Error | No Lexical Error | OK |
| --; | No Lexical Error | No Lexical Error | OK |
| --(lowercase) | No Lexical Error | No Lexical Error | OK |
| --) | No Lexical Error | No Lexical Error | OK |
| --(numbers) | No Lexical Error | No Lexical Error | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
| --\* | LEXICAL ERROR: On '--', Invalid Delimiter '\*' | LEXICAL ERROR: On '--', Invalid Delimiter '\*' | OK |
| --( | LEXICAL ERROR: On '--', Invalid Delimiter '(' | LEXICAL ERROR: On '--', Invalid Delimiter '(' | OK |
| --\ | LEXICAL ERROR: On '--', Invalid Delimiter '\' | LEXICAL ERROR: On '--', Invalid Delimiter '\' | OK |
| --/ | LEXICAL ERROR: On '--', Invalid Delimiter '/' | LEXICAL ERROR: On '--', Invalid Delimiter '/' | OK |
| --{ | LEXICAL ERROR: On '--', Invalid Delimiter '{' | LEXICAL ERROR: On '--', Invalid Delimiter '{' | OK |
| -=⎵ | No Lexical Error | No Lexical Error | OK |
| -=**↵** | No Lexical Error | No Lexical Error | OK |
| -=( | No Lexical Error | No Lexical Error | OK |
| -=(lowercase) | No Lexical Error | No Lexical Error | OK |
| -=(numbers) | No Lexical Error | No Lexical Error | OK |
| -=~ | No Lexical Error | No Lexical Error | OK |
| -=# | LEXICAL ERROR: On '-=', Invalid Delimiter '#' | LEXICAL ERROR: On '-=', Invalid Delimiter '#' | OK |
| -=& | LEXICAL ERROR: On '-=', Invalid Delimiter '&' | LEXICAL ERROR: On '-=', Invalid Delimiter '&' | OK |
| -=” | LEXICAL ERROR: On '-=', Invalid Delimiter '”' | LEXICAL ERROR: On '-=', Invalid Delimiter '”' | OK |
| -=: | LEXICAL ERROR: On '-=', Invalid Delimiter ':' | LEXICAL ERROR: On '-=', Invalid Delimiter ':' | OK |
| -=; | LEXICAL ERROR: On '-=', Invalid Delimiter ';' | LEXICAL ERROR: On '-=', Invalid Delimiter ';' | OK |
| \*⎵ | No Lexical Error | No Lexical Error | OK |
| \***↵** | No Lexical Error | No Lexical Error | OK |
| \*( | No Lexical Error | No Lexical Error | OK |
| \*(lowercase) | No Lexical Error | No Lexical Error | OK |
| \*(numbers) | No Lexical Error | No Lexical Error | OK |
| \*~ | No Lexical Error | No Lexical Error | OK |
| \*% | LEXICAL ERROR: On '\*', Invalid Delimiter '%' | LEXICAL ERROR: On '\*', Invalid Delimiter '%' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
| \*+ | LEXICAL ERROR: On '\*', Invalid Delimiter '+' | LEXICAL ERROR: On '\*', Invalid Delimiter '+' | OK |
| \*! | LEXICAL ERROR: On '\*', Invalid Delimiter '!' | LEXICAL ERROR: On '\*', Invalid Delimiter '!' | OK |
| \*# | LEXICAL ERROR: On '\*', Invalid Delimiter '#' | LEXICAL ERROR: On '\*', Invalid Delimiter '#' | OK |
| \*) | LEXICAL ERROR: On '\*', Invalid Delimiter ')' | LEXICAL ERROR: On '\*', Invalid Delimiter ')' | OK |
| \*=⎵ | No Lexical Error | No Lexical Error | OK |
| \*=**↵** | No Lexical Error | No Lexical Error | OK |
| \*=( | No Lexical Error | No Lexical Error | OK |
| \*=(lowercase) | No Lexical Error | No Lexical Error | OK |
| \*= (numbers) | No Lexical Error | No Lexical Error | OK |
| \*=~ | No Lexical Error | No Lexical Error | OK |
| \*=; | LEXICAL ERROR: On '+=', Invalid Delimiter ';' | LEXICAL ERROR: On '+=', Invalid Delimiter ';' | OK |
| \*=) | LEXICAL ERROR: On '+=', Invalid Delimiter ')' | LEXICAL ERROR: On '+=', Invalid Delimiter ')' | OK |
| \*=] | LEXICAL ERROR: On '+=', Invalid Delimiter ']' | LEXICAL ERROR: On '+=', Invalid Delimiter ']' | OK |
| \*={ | LEXICAL ERROR: On '+=', Invalid Delimiter '{' | LEXICAL ERROR: On '+=', Invalid Delimiter '{' | OK |
| \*=! | LEXICAL ERROR: On '+=', Invalid Delimiter '!' | LEXICAL ERROR: On '+=', Invalid Delimiter '!' | OK |
| /⎵ | No Lexical Error | No Lexical Error | OK |
| /**↵** | No Lexical Error | No Lexical Error | OK |
| /( | No Lexical Error | No Lexical Error | OK |
| /(lowercase) | No Lexical Error | No Lexical Error | OK |
| /(numbers) | No Lexical Error | No Lexical Error | OK |
| /~ | No Lexical Error | No Lexical Error | OK |
| /& | LEXICAL ERROR: On '/', Invalid Delimiter '&' | LEXICAL ERROR: On '/', Invalid Delimiter '&' | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
| /\ | LEXICAL ERROR: On '/', Invalid Delimiter '\' | LEXICAL ERROR: On '/', Invalid Delimiter '\' | OK |
| /{ | LEXICAL ERROR: On '/', Invalid Delimiter '{' | LEXICAL ERROR: On '/', Invalid Delimiter '{' | OK |
| /; | LEXICAL ERROR: On '/', Invalid Delimiter ';' | LEXICAL ERROR: On '/', Invalid Delimiter ';' | OK |
| /” | LEXICAL ERROR: On '/', Invalid Delimiter '”' | LEXICAL ERROR: On '/', Invalid Delimiter '”' | OK |
| /=⎵ | No Lexical Error | No Lexical Error | OK |
| /=**↵** | No Lexical Error | No Lexical Error | OK |
| /=( | No Lexical Error | No Lexical Error | OK |
| /=(lowercase) | No Lexical Error | No Lexical Error | OK |
| /=(numbers) | No Lexical Error | No Lexical Error | OK |
| /=~ | No Lexical Error | No Lexical Error | OK |
| /== | LEXICAL ERROR: On '/=', Invalid Delimiter '=' | LEXICAL ERROR: On '/=', Invalid Delimiter '=' | OK |
| /=# | LEXICAL ERROR: On '/=', Invalid Delimiter '#' | LEXICAL ERROR: On '/=', Invalid Delimiter '#' | OK |
| /=! | LEXICAL ERROR: On '/=', Invalid Delimiter '!' | LEXICAL ERROR: On '/=', Invalid Delimiter '!' | OK |
| /=+ | LEXICAL ERROR: On '/=', Invalid Delimiter '+' | LEXICAL ERROR: On '/=', Invalid Delimiter '+' | OK |
| /=$ | LEXICAL ERROR: On '/=', Invalid Delimiter '$' | LEXICAL ERROR: On '/=', Invalid Delimiter '$' | OK |
| &&⎵ | No Lexical Error | No Lexical Error | OK |
| &&**↵** | No Lexical Error | No Lexical Error | OK |
| &&( | No Lexical Error | No Lexical Error | OK |
| &&(lowercase) | No Lexical Error | No Lexical Error | OK |
| &&(numbers) | No Lexical Error | No Lexical Error | OK |
| &&~ | No Lexical Error | No Lexical Error | OK |
| &&” | No Lexical Error | No Lexical Error | OK |
| &&’ | No Lexical Error | No Lexical Error | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| &&! | No Lexical Error | No Lexical Error | OK |
| &&{ | LEXICAL ERROR: On '&&', Invalid Delimiter '{' | LEXICAL ERROR: On '&&', Invalid Delimiter '{' | OK |
| &&) | LEXICAL ERROR: On '&&', Invalid Delimiter ')' | LEXICAL ERROR: On '&&', Invalid Delimiter ')' | OK |
| &&[ | LEXICAL ERROR: On '&&', Invalid Delimiter '[' | LEXICAL ERROR: On '&&', Invalid Delimiter '[' | OK |
| &&@ | LEXICAL ERROR: On '&&', Invalid Delimiter '@' | LEXICAL ERROR: On '&&', Invalid Delimiter '@' | OK |
| &&| | LEXICAL ERROR: On '&&', Invalid Delimiter '|' | LEXICAL ERROR: On '&&', Invalid Delimiter '|' | OK |
| &&\_ | LEXICAL ERROR: On '&&', Invalid Delimiter '\_' | LEXICAL ERROR: On '&&', Invalid Delimiter '\_' | OK |
| ##⎵ | No Lexical Error | No Lexical Error | OK |
| ##**↵** | No Lexical Error | No Lexical Error | OK |
| ##( | No Lexical Error | No Lexical Error | OK |
| ##(lowercase) | No Lexical Error | No Lexical Error | OK |
| ##(numbers) | No Lexical Error | No Lexical Error |  |
| ##~ | No Lexical Error | No Lexical Error | OK |
| ##” | No Lexical Error | No Lexical Error | OK |
| ##’ | No Lexical Error | No Lexical Error | OK |
| ##! | No Lexical Error | No Lexical Error | OK |
| ##{ | LEXICAL ERROR: On '##', Invalid Delimiter '{' | LEXICAL ERROR: On '##', Invalid Delimiter '{' | OK |
| ##] | LEXICAL ERROR: On '##', Invalid Delimiter ']' | LEXICAL ERROR: On '##', Invalid Delimiter ']' | OK |
| ##} | LEXICAL ERROR: On '##', Invalid Delimiter '}' | LEXICAL ERROR: On '##', Invalid Delimiter '}' | OK |
| ##\ | LEXICAL ERROR: On '##', Invalid Delimiter '\' | LEXICAL ERROR: On '##', Invalid Delimiter '\' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| ##> | LEXICAL ERROR: On '##', Invalid Delimiter '>' | LEXICAL ERROR: On '##', Invalid Delimiter '>' | OK |
| <⎵ | No Lexical Error | No Lexical Error | OK |
| <**↵** | No Lexical Error | No Lexical Error | OK |
| <( | No Lexical Error | No Lexical Error | OK |
| <(lowercase) | No Lexical Error | No Lexical Error | OK |
| <(numbers) | No Lexical Error | No Lexical Error | OK |
| <~ | No Lexical Error | No Lexical Error | OK |
| <” | LEXICAL ERROR: On '<', Invalid Delimiter '"' | LEXICAL ERROR: On '<', Invalid Delimiter '"' | OK |
| <’ | LEXICAL ERROR: On '<', Invalid Delimiter '’' | LEXICAL ERROR: On '<', Invalid Delimiter '’' | OK |
| <{ | LEXICAL ERROR: On '<', Invalid Delimiter '{' | LEXICAL ERROR: On '<', Invalid Delimiter '{' | OK |
| <) | LEXICAL ERROR: On '<', Invalid Delimiter ')' | LEXICAL ERROR: On '<', Invalid Delimiter ')' | OK |
| <[ | LEXICAL ERROR: On '<', Invalid Delimiter '[' | LEXICAL ERROR: On '<', Invalid Delimiter '[' | OK |
| <=⎵ | No Lexical Error | No Lexical Error | OK |
| <=**↵** | No Lexical Error | No Lexical Error | OK |
| <=( | No Lexical Error | No Lexical Error | OK |
| <=(lowercase) | No Lexical Error | No Lexical Error | OK |
| <=(numbers) | No Lexical Error | No Lexical Error | OK |
| <=~ | No Lexical Error | No Lexical Error | OK |
| <=# | LEXICAL ERROR: On '<=', Invalid Delimiter '#' | LEXICAL ERROR: On '<=', Invalid Delimiter '#' | OK |
| <=@ | LEXICAL ERROR: On '<=', Invalid Delimiter '@' | LEXICAL ERROR: On '<=', Invalid Delimiter '@' | OK |
| <=| | LEXICAL ERROR: On '<=', Invalid Delimiter '|' | LEXICAL ERROR: On '<=', Invalid Delimiter '|' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| <=) | LEXICAL ERROR: On '<=', Invalid Delimiter ')' | LEXICAL ERROR: On '<=', Invalid Delimiter ')' | OK |
| <=; | LEXICAL ERROR: On '<=', Invalid Delimiter ';' | LEXICAL ERROR: On '<=', Invalid Delimiter ';' | OK |
| >⎵ | No Lexical Error | No Lexical Error | OK |
| >**↵** | No Lexical Error | No Lexical Error | OK |
| >( | No Lexical Error | No Lexical Error | OK |
| >(lowercase) | No Lexical Error | No Lexical Error | OK |
| >(numbers) | No Lexical Error | No Lexical Error | OK |
| >~ | No Lexical Error | No Lexical Error | OK |
| ># | LEXICAL ERROR: On '>', Invalid Delimiter '#' | LEXICAL ERROR: On '>', Invalid Delimiter '#' | OK |
| >& | LEXICAL ERROR: On '>', Invalid Delimiter '&' | LEXICAL ERROR: On '>', Invalid Delimiter '&' | OK |
| >! | LEXICAL ERROR: On '>', Invalid Delimiter '!' | LEXICAL ERROR: On '>', Invalid Delimiter '!' | OK |
| >; | LEXICAL ERROR: On '>', Invalid Delimiter ';' | LEXICAL ERROR: On '>', Invalid Delimiter ';' | OK |
| >. | LEXICAL ERROR: On '>', Invalid Delimiter '.' | LEXICAL ERROR: On '>', Invalid Delimiter '.' | OK |
| >=⎵ | No Lexical Error | No Lexical Error | OK |
| >=**↵** | No Lexical Error | No Lexical Error | OK |
| >=( | No Lexical Error | No Lexical Error | OK |
| >=(lowercase) | No Lexical Error | No Lexical Error | OK |
| >=(numbers) | No Lexical Error | No Lexical Error | OK |
| >=~ | No Lexical Error | No Lexical Error | OK |
| >=) | LEXICAL ERROR: On '>=', Invalid Delimiter ')' | LEXICAL ERROR: On '>=', Invalid Delimiter ')' | OK |
| >=. | LEXICAL ERROR: On '>=', Invalid Delimiter '.' | LEXICAL ERROR: On '>=', Invalid Delimiter '.' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| >=> | LEXICAL ERROR: On '>=', Invalid Delimiter '>' | LEXICAL ERROR: On '>=', Invalid Delimiter '>' | OK |
| >== | LEXICAL ERROR: On '>=', Invalid Delimiter '=' | LEXICAL ERROR: On '>=', Invalid Delimiter '=' | OK |
| >=@ | LEXICAL ERROR: On '>=', Invalid Delimiter '@' | LEXICAL ERROR: On '>=', Invalid Delimiter '@' | OK |
| =⎵ | No Lexical Error | No Lexical Error | OK |
| =**↵** | No Lexical Error | No Lexical Error | OK |
| =( | No Lexical Error | No Lexical Error | OK |
| =(lowercase) | No Lexical Error | No Lexical Error | OK |
| =(numbers) | No Lexical Error | No Lexical Error | OK |
| =~ | No Lexical Error | No Lexical Error | OK |
| =” | No Lexical Error | No Lexical Error | OK |
| =’ | No Lexical Error | No Lexical Error | OK |
| ={ | LEXICAL ERROR: On '=', Invalid Delimiter '{' | LEXICAL ERROR: On '=', Invalid Delimiter '{' | OK |
| =! | LEXICAL ERROR: On '=', Invalid Delimiter '!' | LEXICAL ERROR: On '=', Invalid Delimiter '!' | OK |
| =< | LEXICAL ERROR: On '=', Invalid Delimiter '<' | LEXICAL ERROR: On '=', Invalid Delimiter '<' | OK |
| =\* | LEXICAL ERROR: On '=', Invalid Delimiter '\*' | LEXICAL ERROR: On '=', Invalid Delimiter '\*' | OK |
| =+ | LEXICAL ERROR: On '=', Invalid Delimiter '+' | LEXICAL ERROR: On '=', Invalid Delimiter '+' | OK |
| ==⎵ | No Lexical Error | No Lexical Error | OK |
| ==**↵** | No Lexical Error | No Lexical Error | OK |
| ==( | No Lexical Error | No Lexical Error | OK |
| ==(lowercase) | No Lexical Error | No Lexical Error | OK |
| ==(numbers) | No Lexical Error | No Lexical Error | OK |
| ==~ | No Lexical Error | No Lexical Error | OK |
| ==” | No Lexical Error | No Lexical Error | OK |
| ==’ | No Lexical Error | No Lexical Error | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| =={ | LEXICAL ERROR: On '==', Invalid Delimiter '{' | LEXICAL ERROR: On '==', Invalid Delimiter '{' | OK |
| ==| | LEXICAL ERROR: On '==', Invalid Delimiter '|' | LEXICAL ERROR: On '==', Invalid Delimiter '|' | OK |
| ==; | LEXICAL ERROR: On '==', Invalid Delimiter ';' | LEXICAL ERROR: On '==', Invalid Delimiter ';' | OK |
| ==. | LEXICAL ERROR: On '==', Invalid Delimiter '.' | LEXICAL ERROR: On '==', Invalid Delimiter '.' | OK |
| ==: | LEXICAL ERROR: On '==', Invalid Delimiter ':' | LEXICAL ERROR: On '==', Invalid Delimiter ':' | OK |
| ;⎵ | No Lexical Error | No Lexical Error | OK |
| ;**↵** | No Lexical Error | No Lexical Error | OK |
| ;( | No Lexical Error | No Lexical Error | OK |
| ;(lowercase) | No Lexical Error | No Lexical Error | OK |
| ;(numbers) | No Lexical Error | No Lexical Error | OK |
| ;~ | No Lexical Error | No Lexical Error | OK |
| ;” | No Lexical Error | No Lexical Error | OK |
| ;’ | No Lexical Error | No Lexical Error | OK |
| ;! | No Lexical Error | No Lexical Error | OK |
| ;+ | No Lexical Error | No Lexical Error | OK |
| ;- | No Lexical Error | No Lexical Error | OK |
| ;{ | LEXICAL ERROR: Invalid Delimiter '{' | LEXICAL ERROR: Invalid Delimiter '{' | OK |
| ;= | LEXICAL ERROR: Invalid Delimiter '=' | LEXICAL ERROR: Invalid Delimiter '=' | OK |
| ;> | LEXICAL ERROR: Invalid Delimiter '>' | LEXICAL ERROR: Invalid Delimiter '>' | OK |
| ;& | LEXICAL ERROR: Invalid Delimiter '&' | LEXICAL ERROR: Invalid Delimiter '&' | OK |
| ;< | LEXICAL ERROR: Invalid Delimiter '<' | LEXICAL ERROR: Invalid Delimiter '<' | OK |
| ,⎵ | No Lexical Error | No Lexical Error | OK |
| , **↵** | No Lexical Error | No Lexical Error | OK |
| ,(lowercase) | No Lexical Error | No Lexical Error | OK |
| ,(numbers) | No Lexical Error | No Lexical Error | OK |
| ,~ | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
| ,” | No Lexical Error | No Lexical Error | OK |
| ,’ | No Lexical Error | No Lexical Error | OK |
| ,+ | LEXICAL ERROR: On ',', Invalid Delimiter '+' | LEXICAL ERROR: On ',', Invalid Delimiter '+' | OK |
| ,& | LEXICAL ERROR: On ',', Invalid Delimiter '&' | LEXICAL ERROR: On ',', Invalid Delimiter '&' | OK |
| ,# | LEXICAL ERROR: On ',', Invalid Delimiter '#' | LEXICAL ERROR: On ',', Invalid Delimiter '#' | OK |
| ,) | LEXICAL ERROR: On ',', Invalid Delimiter ')' | LEXICAL ERROR: On ',', Invalid Delimiter ')' | OK |
| ,@ | LEXICAL ERROR: On ',', Invalid Delimiter '@' | LEXICAL ERROR: On ',', Invalid Delimiter '@' | OK |
| )⎵ | No Lexical Error | No Lexical Error | OK |
| ) **↵** | No Lexical Error | No Lexical Error | OK |
| ), | No Lexical Error | No Lexical Error | OK |
| )) | No Lexical Error | No Lexical Error | OK |
| ); | No Lexical Error | No Lexical Error | OK |
| )# | No Lexical Error | No Lexical Error | OK |
| )& | No Lexical Error | No Lexical Error | OK |
| )+ | No Lexical Error | No Lexical Error | OK |
| )- | No Lexical Error | No Lexical Error | OK |
| )\* | No Lexical Error | No Lexical Error | OK |
| )/ | No Lexical Error | No Lexical Error | OK |
| )% | No Lexical Error | No Lexical Error | OK |
| )> | No Lexical Error | No Lexical Error | OK |
| )< | No Lexical Error | No Lexical Error | OK |
| )= | No Lexical Error | No Lexical Error | OK |
| )$ | LEXICAL ERROR: On ')', Invalid Delimiter '$' | LEXICAL ERROR: On ')', Invalid Delimiter '$' | OK |
| )^ | LEXICAL ERROR: On ')', Invalid Delimiter '^' | LEXICAL ERROR: On ')', Invalid Delimiter '^' | OK |
| )( | LEXICAL ERROR: On ')', Invalid Delimiter '(' | LEXICAL ERROR: On ')', Invalid Delimiter '(' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| )^ | LEXICAL ERROR: On ')', Invalid Delimiter '^' | LEXICAL ERROR: On ')', Invalid Delimiter '^' | OK |
| )0 | LEXICAL ERROR: On ')', Invalid Delimiter '0' | LEXICAL ERROR: On ')', Invalid Delimiter '0' | OK |
| (⎵ | No Lexical Error | No Lexical Error | OK |
| (**↵** | No Lexical Error | No Lexical Error | OK |
| (( | No Lexical Error | No Lexical Error | OK |
| ((lowercase) | No Lexical Error | No Lexical Error | OK |
| ((numbers) | No Lexical Error | No Lexical Error | OK |
| (~ | No Lexical Error | No Lexical Error | OK |
| (“ | No Lexical Error | No Lexical Error | OK |
| (‘ | No Lexical Error | No Lexical Error | OK |
| (! | No Lexical Error | No Lexical Error | OK |
| () | No Lexical Error | No Lexical Error | OK |
| ($ | LEXICAL ERROR: On '(', Invalid Delimiter '$' | LEXICAL ERROR: On '(', Invalid Delimiter '$' | OK |
| (# | LEXICAL ERROR: On '(', Invalid Delimiter '#' | LEXICAL ERROR: On '(', Invalid Delimiter '#' | OK |
| (& | LEXICAL ERROR: On '(', Invalid Delimiter '&' | LEXICAL ERROR: On '(', Invalid Delimiter '&' | OK |
| ({ | LEXICAL ERROR: On '(', Invalid Delimiter '{' | LEXICAL ERROR: On '(', Invalid Delimiter '{' | OK |
| (} | LEXICAL ERROR: On '(', Invalid Delimiter '}' | LEXICAL ERROR: On '(', Invalid Delimiter '}' | OK |
| ]⎵ | No Lexical Error | No Lexical Error | OK |
| ]**↵** | No Lexical Error | No Lexical Error | OK |
| ], | No Lexical Error | No Lexical Error | OK |
| ]; | No Lexical Error | No Lexical Error | OK |
| ]) | No Lexical Error | No Lexical Error | OK |
| ]# | No Lexical Error | No Lexical Error | OK |
| ]& | No Lexical Error | No Lexical Error | OK |
| ]+ | No Lexical Error | No Lexical Error | OK |
| ]- | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
| ]\* | No Lexical Error | No Lexical Error | OK |
| ]/ | No Lexical Error | No Lexical Error | OK |
| ]% | No Lexical Error | No Lexical Error | OK |
| ]> | No Lexical Error | No Lexical Error | OK |
| ]< | No Lexical Error | No Lexical Error | OK |
| ]= | No Lexical Error | No Lexical Error | OK |
| ]! | No Lexical Error | No Lexical Error | OK |
| ][ | No Lexical Error | No Lexical Error | OK |
| ]. | No Lexical Error | No Lexical Error | OK |
| ]! | No Lexical Error | No Lexical Error | OK |
| ](numbers) | LEXICAL ERROR: On ']', Invalid Delimiter '1' | LEXICAL ERROR: On ']', Invalid Delimiter '1' | OK |
| ]] | LEXICAL ERROR: On ']', Invalid Delimiter ']' | LEXICAL ERROR: On ']', Invalid Delimiter ']' | OK |
| ]} | LEXICAL ERROR: On ']', Invalid Delimiter '}' | LEXICAL ERROR: On ']', Invalid Delimiter '}' | OK |
| ]| | LEXICAL ERROR: On ']', Invalid Delimiter '|' | LEXICAL ERROR: On ']', Invalid Delimiter '|' | OK |
| ]{ | LEXICAL ERROR: On ']', Invalid Delimiter '{' | LEXICAL ERROR: On ']', Invalid Delimiter '{' | OK |
| [⎵ | No Lexical Error | No Lexical Error | OK |
| [(lowercase) | No Lexical Error | No Lexical Error | OK |
| [(numbers) | No Lexical Error | No Lexical Error | OK |
| [~ | No Lexical Error | No Lexical Error | OK |
| [[ | LEXICAL ERROR: On '[', Invalid Delimiter '[' | LEXICAL ERROR: On '[', Invalid Delimiter '[' | OK |
| [# | LEXICAL ERROR: On '[', Invalid Delimiter '#' | LEXICAL ERROR: On '[', Invalid Delimiter '#' | OK |
| [& | LEXICAL ERROR: On '[', Invalid Delimiter '&' | LEXICAL ERROR: On '[', Invalid Delimiter '&' | OK |
| [@ | LEXICAL ERROR: On '[', Invalid Delimiter '@' | LEXICAL ERROR: On '[', Invalid Delimiter '@' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| [. | LEXICAL ERROR: On '[', Invalid Delimiter '.' | LEXICAL ERROR: On '[', Invalid Delimiter '.' | OK |
| }⎵ | No Lexical Error | No Lexical Error | OK |
| }**↵** | No Lexical Error | No Lexical Error | OK |
| } (lowercase) | No Lexical Error | No Lexical Error | OK |
| }} | No Lexical Error | No Lexical Error | OK |
| }{ | LEXICAL ERROR: On '}', Invalid Delimiter '{' | LEXICAL ERROR: On '}', Invalid Delimiter '{' | OK |
| }@ | LEXICAL ERROR: On '}', Invalid Delimiter '@' | LEXICAL ERROR: On '}', Invalid Delimiter '@' | OK |
| }+ | LEXICAL ERROR: On '}', Invalid Delimiter '+' | LEXICAL ERROR: On '}', Invalid Delimiter '+' | OK |
| }- | LEXICAL ERROR: On '}', Invalid Delimiter '-' | LEXICAL ERROR: On '}', Invalid Delimiter '-' | OK |
| }(numbers) | LEXICAL ERROR: On '}', Invalid Delimiter '}' | LEXICAL ERROR: On '}', Invalid Delimiter '}' | OK |
| {⎵ | No Lexical Error | No Lexical Error | OK |
| {**↵** | No Lexical Error | No Lexical Error | OK |
| {( | No Lexical Error | No Lexical Error | OK |
| {(lowercase) | No Lexical Error | No Lexical Error | OK |
| {} | No Lexical Error | No Lexical Error | OK |
| {! | LEXICAL ERROR: On '{', Invalid Delimiter '!' | LEXICAL ERROR: On '{', Invalid Delimiter '!' | OK |
| {(numbers) | LEXICAL ERROR: On '{', Invalid Delimiter '1' | LEXICAL ERROR: On '{', Invalid Delimiter '1' | OK |
| {? | LEXICAL ERROR: On '{', Invalid Delimiter '?' | LEXICAL ERROR: On '{', Invalid Delimiter '?' | OK |
| {& | LEXICAL ERROR: On '{', Invalid Delimiter '&' | LEXICAL ERROR: On '{', Invalid Delimiter '&' | OK |
|  |  |  |  |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| %⎵ | No Lexical Error | No Lexical Error | OK |
| %**↵** | No Lexical Error | No Lexical Error | OK |
| % (lowercase) | No Lexical Error | No Lexical Error | OK |
| %(numbers) | No Lexical Error | No Lexical Error | OK |
| %~ | No Lexical Error | No Lexical Error | OK |
| %( | No Lexical Error | No Lexical Error | OK |
| %) | LEXICAL ERROR: On '%', Invalid Delimiter ')' | LEXICAL ERROR: On '%', Invalid Delimiter ')' | OK |
| %& | LEXICAL ERROR: On '%', Invalid Delimiter '&' | LEXICAL ERROR: On '%', Invalid Delimiter '&' | OK |
| %# | LEXICAL ERROR: On '%', Invalid Delimiter '#' | LEXICAL ERROR: On '%', Invalid Delimiter '#' | OK |
| %[ | LEXICAL ERROR: On '%', Invalid Delimiter '[' | LEXICAL ERROR: On '%', Invalid Delimiter '[' | OK |
| %] | LEXICAL ERROR: On '%', Invalid Delimiter ']' | LEXICAL ERROR: On '%', Invalid Delimiter ']' | OK |
| %=⎵ | No Lexical Error | No Lexical Error | OK |
| %=**↵** | No Lexical Error | No Lexical Error | OK |
| %=( | No Lexical Error | No Lexical Error | OK |
| %=(lowercase) | No Lexical Error | No Lexical Error | OK |
| %=(numbers) | No Lexical Error | No Lexical Error | OK |
| %=~ | No Lexical Error | No Lexical Error | OK |
| %== | LEXICAL ERROR: On '/=', Invalid Delimiter '=' | LEXICAL ERROR: On '/=', Invalid Delimiter '=' | OK |
| %=# | LEXICAL ERROR: On '/=', Invalid Delimiter '#' | LEXICAL ERROR: On '/=', Invalid Delimiter '#' | OK |
| %=! | LEXICAL ERROR: On '/=', Invalid Delimiter '!' | LEXICAL ERROR: On '/=', Invalid Delimiter '!' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| %z=$ | LEXICAL ERROR: On '/=', Invalid Delimiter '$' | LEXICAL ERROR: On '/=', Invalid Delimiter '$' | OK |
| .(lowercase) | No Lexical Error | No Lexical Error | OK |
| .(numbers) | LEXICAL ERROR: On '.', Invalid Delimiter '1' | LEXICAL ERROR: On '.', Invalid Delimiter '1' | OK |
| .` | LEXICAL ERROR: On '.', Invalid Delimiter '`' | LEXICAL ERROR: On '.', Invalid Delimiter '`' | OK |
| .# | LEXICAL ERROR: On '.', Invalid Delimiter '#' | LEXICAL ERROR: On '.', Invalid Delimiter '#' | OK |
| .& | LEXICAL ERROR: On '.', Invalid Delimiter '&' | LEXICAL ERROR: On '.', Invalid Delimiter '&' | OK |
|  |  |  |  |

## **piecelit**

*LEGEND: (ascii\_p = any printable characters (except ‘, ^, newline, horizontal tab))*

|  |  |  |  |
| --- | --- | --- | --- |
| ‘ascii\_p’⎵ | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’**↵** | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’, | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’; | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’) | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’# | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’& | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’! | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’= | No Lexical Error | No Lexical Error | OK |
| ‘ascii\_p’+ | LEXICAL ERROR: On ''a'', Invalid Delimiter '+' | LEXICAL ERROR: On ''a'', Invalid Delimiter '+' | OK |
| ‘ascii\_p’- | LEXICAL ERROR: On ''a'', Invalid Delimiter '-' | LEXICAL ERROR: On ''a'', Invalid Delimiter '-' | OK |
| ‘ascii\_p’/ | LEXICAL ERROR: On ''a'', Invalid Delimiter '/' | LEXICAL ERROR: On ''a'', Invalid Delimiter '/' | OK |
| ‘ascii\_p’( | LEXICAL ERROR: On ''a'', Invalid Delimiter '(' | LEXICAL ERROR: On ''a'', Invalid Delimiter '(' | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |

## **setlit**

*LEGEND:* *(ascii\_s = any printable characters (except “, ^, newline, horizontal tab))*

|  |  |  |  |
| --- | --- | --- | --- |
| “ascii\_s”⎵ | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”**↵** | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”, | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”; | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”) | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”# | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”& | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”! | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”= | No Lexical Error | No Lexical Error | OK |
| “ascii\_s”- | LEXICAL ERROR: On '"a"', Invalid Delimiter '-' | LEXICAL ERROR: On '"a"', Invalid Delimiter '-' | OK |
| “ascii\_s”/ | LEXICAL ERROR: On '"a"', Invalid Delimiter '/' | LEXICAL ERROR: On '"a"', Invalid Delimiter '/' | OK |
| “ascii\_s”( | LEXICAL ERROR: On '"a"', Invalid Delimiter '(' | LEXICAL ERROR: On '"a"', Invalid Delimiter '(' | OK |
| “ascii\_s”% | LEXICAL ERROR: On '"a"', Invalid Delimiter '%' | LEXICAL ERROR: On '"a"', Invalid Delimiter '%' | OK |
| “ascii\_s”^ | LEXICAL ERROR: On '"a"', Invalid Delimiter '^' | LEXICAL ERROR: On '"a"', Invalid Delimiter '^' | OK |
|  |  |  |  |
|  |  |  |  |

## **numlit**

*LEGEND:* *(numbers = 1,2,3,4,5,6,7,8,9,0)*

|  |  |  |  |
| --- | --- | --- | --- |
| (numbers) ⎵ | No Lexical Error | No Lexical Error | OK |
| (numbers) **↵** | No Lexical Error | No Lexical Error | OK |
| (numbers), | No Lexical Error | No Lexical Error | OK |
| (numbers); | No Lexical Error | No Lexical Error | OK |
| (numbers)# | No Lexical Error | No Lexical Error | OK |
| (numbers)& | No Lexical Error | No Lexical Error | OK |
| (numbers)> | No Lexical Error | No Lexical Error | OK |
| (numbers)< | No Lexical Error | No Lexical Error | OK |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| (numbers)+ | No Lexical Error | No Lexical Error | OK |
| (numbers)- | No Lexical Error | No Lexical Error | OK |
| (numbers)\* | No Lexical Error | No Lexical Error | OK |
| (numbers)/ | No Lexical Error | No Lexical Error | OK |
| (numbers)% | No Lexical Error | No Lexical Error | OK |
| (numbers)] | No Lexical Error | No Lexical Error | OK |
| (numbers)@ | LEXICAL ERROR: On '123', Invalid Delimiter '@' | LEXICAL ERROR: On '123', Invalid Delimiter '@' | OK |
| (numbers)^ | LEXICAL ERROR: On '123', Invalid Delimiter '^' | LEXICAL ERROR: On '123', Invalid Delimiter '^' | OK |
| (numbers)? | LEXICAL ERROR: On '123', Invalid Delimiter '?' | LEXICAL ERROR: On '123', Invalid Delimiter '?' | OK |
| (numbers)[ | LEXICAL ERROR: On '123', Invalid Delimiter '[' | LEXICAL ERROR: On '123', Invalid Delimiter '[' | OK |

## **decimallit**

*LEGEND: (numbers = 1,2,3,4,5,6,7,8,9,0)*

|  |  |  |  |
| --- | --- | --- | --- |
| (numbers).(numbers) ⎵ | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers) **↵** | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers), | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers); | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)# | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)& | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)> | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)< | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)= | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)+ | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)- | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)\* | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)/ | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)% | No Lexical Error | No Lexical Error | OK |
| (numbers).(numbers)] | LEXICAL ERROR: Invalid Delimiter ']' | LEXICAL ERROR: Invalid Delimiter ']' | OK |
| (numbers).(numbers)@ | LEXICAL ERROR: Invalid Delimiter '@' | LEXICAL ERROR: Invalid Delimiter '@' | OK |
|  |  |  |  |
| **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |
| (numbers).(numbers)! | LEXICAL ERROR: Invalid Delimiter '!' | LEXICAL ERROR: Invalid Delimiter '!' | OK |
| (numbers).(numbers)? | LEXICAL ERROR: Invalid Delimiter '?' | LEXICAL ERROR: Invalid Delimiter '?' | OK |
|  |  |  |  |

## **Identifiers**

|  |  |  |  |
| --- | --- | --- | --- |
| id ⎵ | No Lexical Error | No Lexical Error | OK |
| id **↵** | No Lexical Error | No Lexical Error | OK |
| id, | No Lexical Error | No Lexical Error | OK |
| id; | No Lexical Error | No Lexical Error | OK |
| id# | No Lexical Error | No Lexical Error | OK |
| id& | No Lexical Error | No Lexical Error | OK |
| id+ | No Lexical Error | No Lexical Error | OK |
| id- | No Lexical Error | No Lexical Error | OK |
| id\* | No Lexical Error | No Lexical Error | OK |
| id/ | No Lexical Error | No Lexical Error | OK |
| id% | No Lexical Error | No Lexical Error | OK |
| id< | No Lexical Error | No Lexical Error | OK |
| id> | No Lexical Error | No Lexical Error | OK |
| id= | No Lexical Error | No Lexical Error | OK |
| id[ | No Lexical Error | No Lexical Error | OK |
| id] | No Lexical Error | No Lexical Error | OK |
| id( | No Lexical Error | No Lexical Error | OK |
| id. | No Lexical Error | No Lexical Error | OK |
| id{ | No Lexical Error | No Lexical Error | OK |
| id) | No Lexical Error | No Lexical Error | OK |
| id! | No Lexical Error | No Lexical Error | OK |
| id^ | LEXICAL ERROR: Invalid Delimiter '^' | LEXICAL ERROR: Invalid Delimiter '^' | OK |
| id@ | LEXICAL ERROR: Invalid Delimiter '@' | LEXICAL ERROR: Invalid Delimiter '@' | OK |
| id? | LEXICAL ERROR: Invalid Delimiter '?' | LEXICAL ERROR: Invalid Delimiter '?' | OK |
| id~ | LEXICAL ERROR: Invalid Delimiter '~' | LEXICAL ERROR: Invalid Delimiter '~' | OK |
| id| | LEXICAL ERROR: Invalid Delimiter '|' | LEXICAL ERROR: Invalid Delimiter '|' | OK |

# **Appendix B – Syntax Test Scripts**

**Constant Declaration**

<const\_dec> → locked <const\_dec1>;

<const\_dec1> → <datatype> identifier = <literals> <const\_dec2>

<const\_dec2> → ,identifier = <literals> <const\_dec2>

<const\_dec2> → λ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | locked num a = 2;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 2 | locked num a = 2, b = 2;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 3 | locked num a = 2, b = 2;  locked set c = "hello";  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 4 | locked num ;  TABLE(){} | SYNTAX ERROR: Error on 'num',Unexpected ';',  Expected 'identifier' | SYNTAX ERROR: Error on 'num',Unexpected ';',  Expected 'identifier' | OK |
| 5 | locked num = ;  TABLE(){} | SYNTAX ERROR: Error on 'num',Unexpected '=',  Expected 'identifier' | SYNTAX ERROR: Error on 'num',Unexpected '=',  Expected 'identifier' | OK |
| 6 | locked num a = ;  TABLE(){} | SYNTAX ERROR: Error on '=', Unexpected ';', Expected 'numlit, decimallit, piecelit, setlit, optionlit' | SYNTAX ERROR: Error on '=', Unexpected ';', Expected 'numlit, decimallit, piecelit, setlit, optionlit' | OK |
| 7 | locked num a 2 ;  TABLE(){} | SYNTAX ERROR: Error on 'identifier',Unexpected 'numlit',  Expected '=' | SYNTAX ERROR: Error on 'identifier',Unexpected 'numlit',  Expected '=' | OK |
| 8. | locked num a=2  TABLE(){} | SYNTAX ERROR: Error on 'numlit',Unexpected 'TABLE',  Expected ';' | SYNTAX ERROR: Error on 'numlit',Unexpected 'TABLE',  Expected ';' | OK |
|  |  |  |  |  |

**Function Declaration**

<data\_id> → <datatype> identifier <data\_cont>

<data\_cont> → ( <param> ) { <func\_stmt> <return1> }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | num sample(num var1){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 2 | num sample(num var1,num var2){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 3 | num sample(num var1,num var2, num var3){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 4 | decimal sample(num var1){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 5 | set sample(num var1){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 6 | option sample(num var1){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 7 | piece sample(num var1){  tell("hello");  return var1;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 8. | num sample(num var1){  tell("hello");  }  TABLE(){} | SYNTAX ERROR: Error on ';',Unexpected '}', Expected 'return' | SYNTAX ERROR: Error on ';',Unexpected '}', Expected 'return' | OK |
| 9 | num (num var1){  tell("hello");  return var1;  }  TABLE(){} | SYNTAX ERROR: Error on 'num', Unexpected '(', Expected 'identifier' | SYNTAX ERROR: Error on 'num', Unexpected '(', Expected 'identifier' | OK |
| 10 | num sample(var1){  tell("hello");  return var1;  }  TABLE(){} | SYNTAX ERROR: Error on '(', Unexpected 'identifier', Expected 'num, decimal, piece, set, option,)' | SYNTAX ERROR: Error on '(', Unexpected 'identifier', Expected 'num, decimal, piece, set, option,)' | OK |
| 11 | num sample(num ){  tell("hello");  return var1;  }  TABLE(){} | SYNTAX ERROR: Error on 'num', Unexpected ')', Expected 'identifier' | SYNTAX ERROR: Error on 'num', Unexpected ')', Expected 'identifier' | OK |
| 12 | num sample(num var1)  tell("hello");  return var1;  }  TABLE(){} | SYNTAX ERROR: Error on ')',Unexpected 'tell', Expected '{' | SYNTAX ERROR: Error on ')',Unexpected 'tell', Expected '{' | OK |

**Global Declaration**

<data\_id> → <datatype> identifier <data\_cont>

<data\_cont> → <data\_altr> <id\_add>;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | num a;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 2 | num a = 1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 3 | num a = 1, b;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 4 | num a = 1, b=2;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 5 | set sample = "hello";  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 6 | piece sample = 'a';  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 7 | decimal sample = 2.2;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 8. | option sample = true;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 9 | set sample = "hello";  num sample1 = 123;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 10 | num sample[4];  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 11 | num sample[1][2];  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 12 | num sample[4][4]=((1,2),(2,2),(3,3));  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 13 | num sample[4]=(1,2,3);  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 14 | num = 2;  TABLE(){} | SYNTAX ERROR: Error on 'num', Unexpected '=', Expected 'identifier' | SYNTAX ERROR: Error on 'num', Unexpected '=', Expected 'identifier' | OK |
| 15 | num sample 2;  TABLE(){} | SYNTAX ERROR: Error on 'identifier', Unexpected 'numlit', Expected '(,=,[' | SYNTAX ERROR: Error on 'identifier', Unexpected 'numlit', Expected '(,=,[' | OK |
| 16 | num sample = ;  TABLE(){} | SYNTAX ERROR: Error on '=', Unexpected ';' Expected 'numlit, decimallit, piecelit, setlit, optionlit' | SYNTAX ERROR: Error on '=', Unexpected ';' Expected 'numlit, decimallit, piecelit, setlit, optionlit' | OK |
| 17 | num sample = 2  TABLE(){} | SYNTAX ERROR: Error on 'numlit', Unexpected 'TABLE', Expected ';' | SYNTAX ERROR: Error on 'numlit', Unexpected 'TABLE', Expected ';' | OK |
| 18 | num sample = 2, decimal b;  TABLE(){} | SYNTAX ERROR: Error on ',', Unexpected 'decimal',  Expected 'identifier' | SYNTAX ERROR: Error on ',', Unexpected 'decimal',  Expected 'identifier' | OK |
|  |  |  |  |  |

**Structure Declaration**

<deck\_dec> → deck identifier { <deck\_cont> } <deck\_obj> ;

<deck\_cont> → <datatype> identifier <size\_arr> ; <deck\_elem>

<deck\_elem> → <deck\_cont>

<deck\_elem> → λ

<deck\_obj> → identifier <deck\_obj1>

<deck\_obj1> → ,<deck\_obj>

<deck\_obj1> → λ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | deck sample{  num a;  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 2 | deck sample{  num a;  decimal b;  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 3 | deck sample{  num a;  decimal b;  set c;  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 4 | deck sample{  num a;  decimal b;  set c;  option d;  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 5 | deck sample{  num a;  decimal b;  set c;  option d;  piece e;  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 6 | deck sample{  num a[4];  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 7 | deck sample{  num a[4][4];  }sample1;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 8. | deck sample{  num a;  }sample1,sample2;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 9 | deck sample{  num a;  }sample1,sample2,sample3;  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 10 | deck {  num a;  }sample1;  TABLE(){} | SYNTAX ERROR: Error on 'deck', Unexpected '{',  Expected 'identifier' | SYNTAX ERROR: Error on 'deck', Unexpected '{',  Expected 'identifier' | OK |
| 11 | deck sample  num a;  }sample1;  TABLE(){} | SYNTAX ERROR: Error on 'identifier', Unexpected 'num',  Expected '{' | SYNTAX ERROR: Error on 'identifier', Unexpected 'num',  Expected '{' | OK |
| 12 | deck sample{  }sample1;  TABLE(){} | SYNTAX ERROR: Error on '{', Unexpected '}',  Expected 'num, decimal, piece, set, option' | SYNTAX ERROR: Error on '{', Unexpected '}',  Expected 'num, decimal, piece, set, option' | OK |
| 13 | deck sample{  num a  }sample1;  TABLE(){} | SYNTAX ERROR: Error on 'identifier', Unexpected '}',  Expected ';' | SYNTAX ERROR: Error on 'identifier', Unexpected '}',  Expected ';' | OK |
| 14 | deck sample{  num a;  } ;  TABLE(){} | SYNTAX ERROR: Error on '}', Unexpected ';',  Expected 'identifier' | SYNTAX ERROR: Error on '}', Unexpected ';',  Expected 'identifier' | OK |
|  |  |  |  |  |

**Void Declaration**

<void> → void identifier ( <param> ) { <statements> }

<param> → <datatype> identifier <param1>

<param> → λ

<statements> → <func> <statements>

<statements> → λ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | void sample(num a){  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 2 | void sample(num a, num b){  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 3 | void sample(num a){  tell("hello");  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 4 | void sample(num a){  tell("hello");  check(a);  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 5 | void sample(num a){  a = b + c;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 6 | void sample(num a){  if(a<b){  tell("hello");  }  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 7 | void sample(num a){  if(a<b ## b <= 2){  tell("hello");  }  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 8. | void sample(num a){  num i;  for(i=0;i<a;i++){  blank;  }  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 9 | void sample(num a){  num b, c, d;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 10 | void sample(num a){  a++;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 11 | void sample(num a){  while(a<b){  tell("hello");  }  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 12 | void sample(num a,num b){  do{  tell("hello");  }while(a<b);  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 13 | void sample(num a,num b){  if(a<b){  a++;  }elsif(b>a){  b++;  }else{  blank;  }  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 14 | void (num a){  num a;  }  TABLE(){} | SYNTAX ERROR: Error on 'void', Unexpected '(',  Expected 'identifier' | SYNTAX ERROR: Error on 'void', Unexpected '(',  Expected 'identifier' | OK |
| 15 | void sample num a){  num a;  }  TABLE(){} | SYNTAX ERROR: Error on 'identifier', Unexpected 'num',  Expected '(' | SYNTAX ERROR: Error on 'identifier', Unexpected 'num',  Expected '(' | OK |
| 16 | void sample (num ){  num a;  }  TABLE(){} | SYNTAX ERROR: Error on 'num', Unexpected '{',  Expected 'identifier' | SYNTAX ERROR: Error on 'num', Unexpected '{',  Expected 'identifier' | OK |
| 17 | void sample (num a)  num a;  }  TABLE(){} | SYNTAX ERROR: Error on ')', Unexpected 'num',  Expected '{' | SYNTAX ERROR: Error on ')', Unexpected 'num',  Expected '{' | OK |
| 18 | void sample (num a){  num b;  } | SYNTAX ERROR: Error on '}', Unexpected '',  Expected 'TABLE, locked, datatype, deck, void' | SYNTAX ERROR: Error on '}', Unexpected '',  Expected 'TABLE, locked, datatype, deck, void' | OK |

**TABLE FUNCTION**

<program> → <global> TABLE() { <statements> }

<statements> → <func> <statements>

<statements> → λ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SAMPLE**  **NUMBER** | **SAMPLE INPUT** | **EXPECTED**  **OUTPUT** | **ACTUAL**  **OUTPUT** | **REMARKS** |
|  |  |  |  |  |
| 1 | TABLE(){  num a;  } | No Syntax Error | No Syntax Error | OK |
| 2 | TABLE(){  num a, b;  } | No Syntax Error | No Syntax Error | OK |
| 3 | TABLE(){  num a[4][4];  } | No Syntax Error | No Syntax Error | OK |
| 4 | TABLE(){  num a[4];  } | No Syntax Error | No Syntax Error | OK |
| 5 | TABLE(){  num a=1, b = 1;  } | No Syntax Error | No Syntax Error | OK |
| 6 | TABLE(){  num a=1, b = 1;  decimal c = 2.2;  set d = "hello";  option e = true;  piece f = 'a';  } | No Syntax Error | No Syntax Error | OK |
| 7 | TABLE(){  num a[4]=(1,2,3);  decimal b[4][4]=((1,2),(1,2),(1,2));  } | No Syntax Error | No Syntax Error | OK |
| 8. | TABLE(){  a++;  a--;  } | No Syntax Error | No Syntax Error | OK |
| 9 | TABLE(){  a[1] = b + c;  a[2][2] = b + c;  a.sample = a + b;  } | No Syntax Error | No Syntax Error | OK |
| 10 | TABLE(){  a = b + c;  a+= b;  a-= b;  a\*= b;  a/= b;  a%=b;  } | No Syntax Error | No Syntax Error | OK |
| 11 | TABLE(){  a = b + c;  a = b - c;  a = b \* c;  a = b / c;  a = b % c;  } | No Syntax Error | No Syntax Error | OK |
| 12 | TABLE(){  a+(1/3);  } | No Syntax Error | No Syntax Error | OK |
| 13 | TABLE(){  ++a;  --a;  } | No Syntax Error | No Syntax Error | OK |
| 14 | void (num a){  num a;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 15 | void sample num a){  num a;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 16 | void sample (num ){  num a;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 17 | void sample (num a)  num a;  }  TABLE(){} | No Syntax Error | No Syntax Error | OK |
| 18 | void sample (num a){  num b;  } | No Syntax Error | No Syntax Error | OK |
| 19 | TABLE(){  tell("hello");  } | No Syntax Error | No Syntax Error | OK |
| 20 | TABLE(){  tell(a);  } | No Syntax Error | No Syntax Error | OK |
| 21 | TABLE(){  tell(a, "hello");  } | No Syntax Error | No Syntax Error | OK |
| 22 | TABLE(){  tell(a, "hello", b);  } | No Syntax Error | No Syntax Error | OK |
| 23 | TABLE(){  tell(a+b, "hello", c);  } | No Syntax Error | No Syntax Error | OK |
| 24 | TABLE(){  tell(a+(b+c), "hello", c);  } | No Syntax Error | No Syntax Error | OK |
| 25 | TABLE(){  tell(a+(b+c), "hello", c, "world");  } | No Syntax Error | No Syntax Error | OK |
| 26 | TABLE(){  check(a);  } | No Syntax Error | No Syntax Error | OK |
| 27 | TABLE(){  check(a[1]);  } | No Syntax Error | No Syntax Error | OK |
| 28 | TABLE(){  check(a[1][2]);  } | No Syntax Error | No Syntax Error | OK |
| 29 | TABLE(){  check(a.sample);  } | No Syntax Error | No Syntax Error | OK |
| 30 | TABLE(){  if(a<b){  tell("hello");  }  } | No Syntax Error | No Syntax Error | OK |
| 31 | TABLE(){  if(a<b ## b>= a){  tell("hello");  }  } | No Syntax Error | No Syntax Error | OK |
| 32 | TABLE(){  if((a<b) ## (b>= a)){  tell("hello");  }  } | No Syntax Error | No Syntax Error | OK |
| 33 | TABLE(){  if((a<b) ## (b>= a) && (a==b)){  tell("hello");  }  } | No Syntax Error | No Syntax Error | OK |
| 34 | TABLE(){  if((a<b) ## (b>= a) && (a==b)){  tell("hello");  }elsif((a<b) ## (b>= a) && (a==b)){  tell("world");  }else{  tell("hi");  }  } | No Syntax Error | No Syntax Error | OK |
| 35 | TABLE(){  if(a>b){  if(a<b){  a = b + c;  }  }  } | No Syntax Error | No Syntax Error | OK |
| 36 | TABLE(){  if(a == true){  blank;  a = b + c;  tell("hello");  check (a);  for(i=0;i<a;i++){  break;  continue;  }  }  } | No Syntax Error | No Syntax Error | OK |
| 37 | TABLE(){  if(a == true){  blank;  a = b + c;  tell("hello");  check (a);  for(i=0;i<a;i++){  break;  continue;  }  }else{  while(a<b){  a++;  a+1;  }  do{  tell("hello");  }while(a>b ## b==1);  }  } | No Syntax Error | No Syntax Error | OK |
| 38 | TABLE(){  for(i=0; i<a;i++){  a = b + c;  }  } | No Syntax Error | No Syntax Error | OK |
| 39 | TABLE(){  while(i<a){  a = b + c;  }  } | No Syntax Error | No Syntax Error | OK |
| 40 | TABLE(){  do{  tell("hello");  }while(a<b);  } | No Syntax Error | No Syntax Error | OK |
| 41 | TABLE(){  for(i=1+(a+b); i<=a ## a>b; i++){  tell("hello");  }  } | No Syntax Error | No Syntax Error | OK |
| 42 | TABLE(){  for(i=0; i<a;i++){  a = b + c;  if(a<b){  break;  tell("hello");  }  }  } | No Syntax Error | No Syntax Error | OK |
| 43 | TABLE(){  while(a>=b){  tell("hello");  if(a<b){  continue;  }  a = a + b;  }  } | No Syntax Error | No Syntax Error | OK |
| 44 | TABLE(){  do{  if(a<b){  break;  }  a = b + c;  b++;  check(a);  }while(a<b);  } | No Syntax Error | No Syntax Error | OK |
| 45 | TABLE(){  blank;  } | No Syntax Error | No Syntax Error | OK |
| 46 | TABLE ){  blank;  } | SYNTAX ERROR: Error on 'TABLE', Unexpected ')',  Expected '(' | SYNTAX ERROR: Error on 'TABLE', Unexpected ')',  Expected '(' | OK |
| 47 | TABLE( {  blank;  } | SYNTAX ERROR: Error on '(', Unexpected '{',  Expected ')' | SYNTAX ERROR: Error on '(', Unexpected '{',  Expected ')' | OK |
| 48 | TABLE()  blank;  } | SYNTAX ERROR: Error on ')', Unexpected 'blank',  Expected '{' | SYNTAX ERROR: Error on ')', Unexpected 'blank',  Expected '{' | OK |
| 49 | TABLE(){  blank  } | SYNTAX ERROR: Error on 'blank', Unexpected '}',  Expected ';' | SYNTAX ERROR: Error on 'blank', Unexpected '}',  Expected ';' | OK |
| 50 | TABLE(){  blank; | SYNTAX ERROR: Error on ';', Unexpected '',  Expected '}' | SYNTAX ERROR: Error on ';', Unexpected '',  Expected '}' | OK |