**RESERVED WORDS**

|  |  |  |
| --- | --- | --- |
| **C Language** | **Proposed Language** | **Description** |
| main | Lead | Function called at program startup |
| **Data Types**  A data type is a set of [data](http://searchdatamanagement.techtarget.com/definition/data) with values having predefined characteristics. | | |
| Int | Int | A data type that contains whole numbers. |
| double | Double | Data types that contains whole numbers and a decimal part. (64-bit) |
| char | Char | A data type that deals with characters. |
| String | String | A data type that represent a series of characters. |
| Bool | Boolean | A data type can only handle two values: TRUE or FALSE. |
| Void | Null | A return type that will not return a value. |
| Const | Let | has the value and type of the constant value it represents |
| - | Array | Used to declare an array |
| - | of | Use to specify the size of an array |
| - | as | Used to assign a data type |
| - | Task | To declare a function |
| **Data Structure**  a data structure is a particular way of organizing [data](https://en.wikipedia.org/wiki/Data_(computing)) in a computer so that it can be used[efficiently](https://en.wikipedia.org/wiki/Algorithmic_efficiency). | | |
| Struct | Object | A collection of variables placed under a single name. Variables can have different data types |
| **I/O Statements**  I/O Statements are statements used in order to | | |
| Printf | Say | Formats and prints a series of characters and values |
| Scanf | Read | Reads data and stores it in a location. |
| **Conditional Statement** | | |
| If | If | Executes a sequence of statements if the evaluation of conditional expression is true. |
| Else | Otherwise | Extends an IF statement to execute a differentstatement in case the condition if the IF evaluated to be false. |
| Switch | Option | It allows to test the equality of the value of a variable against a list of values stated in each OPTION statements. |
| Case | State | serves as the cases |
| Default | Default | Default statement is used when there is no true condition among the options |
| Break | Stop | A keyword used to terminate the execution of the iteration |
| Continue | Skip | A keyword that is used to jump to the increment and decrement of the iteration. |
| **Looping Statements** | | |
| For | For | Causes its body of statements to be executed provided ‘termination-condition’ is true. |
| While | Until | Executes the given statements in the body as long as the expression is equal to TRUE. |
| Do | Do | Executes the statements in the body once |
| **Supporting Statements** | | |
| Clrscr | Clear | A keyword RESET used to clear the screen |
| Return | Response | Terminates the current function and returns specific value to the caller function |
| True | Yes | Boolean True value |
| False | No | Boolean False value |
| = | is | Used to initialize/equalize a value |
| - | Start | Serves as a start of a task |
| - | End | Indicates the end of a task |
| - | PS | Start of a comment |
| - | Var | Used to declare a variable |
| - | Let | Used to declare a constant |
| - | by | Used in declaring a multi-dimensional array |

**RESERVED SYMBOLS**

1. ***Arithmetic Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | adds two operands |
| - | subtract second operands from first |
| \* | multiply two operand |
| / | divide numerator by denominator |
| % | remainder of division |
| ++ | Increment operator increases integer value by one |
| -- | Decrement operator decreases integer value by one |

1. ***Relational Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Check if two operands are equal |
| =! | Check if two operands are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| => | check left operand is greater than or equal to right operand |
| =< | Check if operand on left is smaller than or equal to right operand |

1. ***Logical Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | Logical AND |
| || | Logical OR |
| ! | Logical NOT |

1. ***Assignment Operators***

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| += | adds right operand to the left operand and assign the result to left | A=+B is same as A=A+B |
| -= | subtracts right operand from the left operand and assign the result to left operand | A=-B is same as A=A-B |
| \*= | multiply left operand with the right operand and assign the result to left operand | A=\*B is same as A=A\*B |
| /= | divides left operand with the right operand and assign the result to left operand | A=/B is same as A=A/B |
| %= | calculate modulus using two operands and assign the result to left operand | A=%B is same as A=A%B |

***V. Other Symbols***

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | | | **Description** |
| . | | | Terminator (Period) | |
| , | | | String Concatenation (Comma) | |
| ; | | | Separator (Semi-colon) | |
| ‘ ’ | | Used in defining the string value (Single quotes) | | |
| ” ” | | Used in defining the char value (Double quotes) | | |
| : | | Used in declaring a function (Colon) | | |
| ( ) | Used in arithmetic, conditions and parameter list. (Parenthesis) | | | |
| [ ] | Used in declaring an array (Brackets) | | | |
| { } | Start and end of a  comment (Braces) | | | |
| @ | Used in accessing the element of an object | | | |

**IDENTIFIERS**

***Rules in naming Identifiers***

1. It must start with an upper case letter.
2. An Identifier can only have alphanumeric characters and underscores.
3. Maximum of 8 characters per identifier.
4. Identifiers are case sensitive.
5. Keywords are not allowed to be used as Identifiers.
6. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

**Example**

|  |  |
| --- | --- |
| **Valid Identifier** | **Invalid Identifier** |
| Num1 | num1 |
| Product\_ | product 7 |
| Stud\_no | 123Let |
| Sum\_3 | Student; |

**CONSTANT, VARIABLE, AND STRUCTURES**

***I. Rules in Constant and Variable Declaration***

1. Declaration of variables can be declared globally or locally.
2. Declaration must begin with the keywords ***Var*** for variable or ***Let*** for constant, followed by the identifier, then a keyword “as” and its data type.
3. Initialization is done by adding an equal sign after the identifier followed by its initial value.
4. Constant must have an initial value.
5. All declarations and initializations must be terminated by a period.
6. Variables with the same data type can be declared or initialized in a single line as long as the identifiers with or without values are separated with a semicolon.
7. Variables with different data types can be declared or initialized in a single line as long as the identifiers with or without values are separated with a period.
8. The value of an initialized variable must be in respect to its data type.

***I.1 Specific Examples***

1. To declare or initialize a constant variable, use the ***Let*** keyword followingthe definition:

|  |
| --- |
| ***Syntax:***  ***Let <variable name> as <Data Type> [is<value>]*.** |

|  |
| --- |
| ***Example:***   * 1. ***Declaration:* Let Num as Int.**   2. ***Initialization:* Let Pi as Double is 3.14.** |

1. To declare or initialize a variable, use the ***Var*** keyword following the definition:

|  |
| --- |
| ***Syntax:***  ***Var<Variable name> as <Data Type> [is<value>].*** |

|  |
| --- |
| ***Example:***   * 1. ***Declaration:* VarNum as Int.**   2. ***Initialization:* VarNum as Int is 0.**   3. ***Declaration and Initialization:* Var Num1; Num2 is 0 as Int.** |

***II. Rules in Object Declaration***

1. Declaration of object must be declared globally.
2. In declaration, the keyword ***Object*** must be followed by an identifier and enclosed with ***Start*** and ***End*** then ended with a period.
3. Declaration of an object inside an object is allowed as long as the object inside an object is already declared globally.
4. Each element of an object must contain a data type, its identifier and terminated with period.
5. Initialization inside an object is not allowed.
6. In initializing an object variable, add an identifier after the keyword End and before the period. To initialize multiple object variables use a semicolon as a separator.
7. In initializing an object variable locally, use the keyword ***Object*** followed by an existing object then followed by an identifier ended with a period. To initialize multiple object variables use a semicolon as a separator.
8. Elements in an object can be access through its object variable name followed by an***@*** symbol and the element name.
9. In assigning a value for each element of an object, object variable name is followed by an***@*** symbol then name of the element followed by an is and its value ended with a period.

***II.1 Specific Examples***

1. To declare a structure of data, use the ***Object*** keyword following the definition:

|  |
| --- |
| ***Syntax:***  ***Object <Object name> Start***  ***[Var<Variable name> as <Data type>].***  ***…***  ***[Object <Object name><Variable name>].***  ***…***  ***End [<Variable name>[;<Variable name 2>; … <Variable name 3>]].*** |

|  |
| --- |
| ***Example:***   * 1. **Object Date Start Var Day as Int. Var Month as String. Var Year as Int. End.** |

|  |
| --- |
| * 1. **Object Personal Start Var Name as String. Var Age as Int. Object Date Birthday. End.** |

|  |
| --- |
| * 1. **Object Profile Start Var Address as String. Var ContactNo as String.  Object Personal Info. End Male; Female.** |

To declare a variable of type object use the following definition:

|  |
| --- |
| ***Syntax:***  **Object <Object name><Variable name>.** |

*[Given that the Object name used was already declared.]*

|  |
| --- |
| ***Example:***   * 1. **Object Profile Employed.** |

To access the element of an object use the following definition:

|  |
| --- |
| ***Syntax:***  **<Object Variable name>@<Element Variable name> [@<Element Variable name>… ][ is <Value>.]** |

|  |
| --- |
| ***Example:***   1. **Employed@ContactNo is ‘12345’.** 2. **Employed@Info@Name is ‘John Doe’.** |

**TASKS**

***I. Rules in Task Declaration***

1. Declaration of functions can only be declared globally.
2. Every program must contain at least one task, the ***Lead*** task. It is the first task executed when the program starts.
3. To declare the task Lead, use the keyword ***Task*** followed by the name Lead and ended by a period.
4. Other Tasks must be declared before the ***Lead*** task.
5. In declaring task, use the keyword ***Task*** followed by an identifier followed by its response-type then use period as terminator.
6. In declaring task, response-type must be specified. Use the keyword ***Null*** if there is no need to response anything.
7. In declaring task with a parameter, declare a variable enclosed by parenthesis after the task name. Use semicolon as separator if multiple parameter is needed.
8. Tasks are called through its function name followed by a pair of parenthesis. If there is/are parameter/s needed, it must be indicated and separated by a semicolon inside the parenthesis.
9. A task can call itself (Recursion).
10. Defining the statements for task is done after the Lead task.
11. In defining the statements for task, same rules apply as to declaring one except for the terminator. Instead of a period, use colon then followed by the keyword ***Start*** and **End***,* between the Start and End is/are the statement/s defined. Lastly, use period to terminate the task.
12. Use the keyword ***Response*** followed by a response value if response is necessary. Response value must be in respect to the response type of the task.
13. A task must have at least one statement.

***I.1 Specific Examples***

1. To declare a task, use the ***Task*** keyword following the definition:

|  |
| --- |
| ***Syntax:***  ***Task Lead.*** *(For Lead task)* ***Task<task name> [( Var<variable name> as <Data type>[; Var<variable name> as <Data type>…] )] as <Data type>.*** *(*For other Tasks*)* |

|  |
| --- |
| ***Example:***   * 1. **Task Lead.**   2. **Task Clean as Null.**   3. **Task Sum (Var X as Int; Var Y as Int) as Int.** |

To define the statement of a task, use the following definition:

|  |
| --- |
| ***Syntax:***  ***Task <task name> [( Var<variable name> as <Data type>[; Var<variable name> as <Data type>…] )] as <Data type>: Start Statement\_1. [Statement\_2].  …  [Response <response value>]. End.*** |

|  |
| --- |
| ***Example:***   1. **Task Sum (Var X as Int; Var Y as Int): Start Var S as Int.  S = X + Y.  Response S. End.** 2. **Task Disp as Null: Start  Say(‘Hello Human’). End.** |

**RULES FOR RESERVED WORDS**

***Rules for Input and Output Statement***

***Read*** statement is used for taking user input or reading data one by one.

1. The keyword Read must only be followed by a space, followed by the identifier that will act as placeholder for the input. The statement should always be terminated by period “.”.
2. Identifiers, element of an array, and element of an object are the only valid placeholders for the values to be input by the user. Other than that like, literals, will not be accepted.
3. There should be only one collect statement for each user’s input and putting multiple placeholders inside the collect statement would produce an error.

|  |
| --- |
| ***Specific example:***   1. **Read Num.** 2. **Read Nums[0].** 3. **Read Group@People.** |

On the other hand, ***Say*** statement is used in displaying a series of characters or values of an identifier.

1. Say statement should only be written as uppercase ‘S’ and lowercase ‘a’ and ‘y’, otherwise it will lead to an error.
2. The keyword Say must only be followed by a space, followed by the value to be printed out, enclosed by double quote. The statement should always be terminated by period “.”.
3. In concatenating values inside the Say statement, the symbol ‘,’should be used.
4. To include double quote, use the ampersand symbol ‘&’, to be followed by a double quote. Same rule applies if you want to add the ampersand symbol.
5. Newline is automatically added to the last character to be printed out.

|  |
| --- |
| ***Specific example:***   * 1. Say “Hello World!”. **OUTPUT: Hello World!**   2. Say “Hello”,”World!”. **OUTPUT: Hello World!**   3. Say “&”Hello World!&””. **OUTPUT: “Hello World!”** |

***Rules for Array Declaration***

1. Declaration of arrays can be declared globally or locally.
2. Declaration must begin with the keyword ***Array***, followed by the keyword ***of*** then the size of the array, then the identifier, then a keyword “as”, and its data type.
3. Initialization is done by adding an is after the data type and each entry must be separated by a semicolon.
4. Initialization can be less than the size of the array but cannot be greater than the size of the array.
5. All declarations and initializations must be terminated by a period.
6. Arrays with the same data type can be declared or initialized in a single line as long as the identifiers with or without values are separated with a semicolon.
7. Arrays with different data types can be declared or initialized in a single line as long as the identifiers with or without values are separated with a period.
8. The value of an initialized variable must be in respect to its data type.
9. Array indices are fixed numeric values as declared in its declaration.
10. Array contents must strictly contain a data of its own data type.
11. Array indices are positive whole numbers only.
12. The array size must be declared.
13. The maximum dimension of an array is two. The minimum size of an array is one and the maximum size of an array is 50.
14. To declare a multidimensional array, add ***by*** right after the first size of the array.
15. Initialization to a multidimensional array is not possible.

|  |
| --- |
| ***Specific example:***   1. **Array of 3 Nums as Int.** 2. **Array of 3 Nums as Int is 1;2;3.** 3. **Array of 3 Names as Stringis“John”;”Doe”.** 4. **Array of 3 by 2 Nums as Int.** |

***Rules for Conditional Statements***

1. There must be always a boolean expression enclosed in parentheses “( )” after the word ***If*** or ***Otherwise***. Boolean expression can be either a boolean value or a conditional expression.The arguments in the conditional expression could be an identifier, an element of an array, an element of an object, a task or a literal, with a conditional/relational operator.
2. Next to it, any number of statements must be enclosed in the keyword ***EndIf*** and ended with.
3. Nested conditional statements are allowed.

|  |
| --- |
| ***Specific example:***   1. **If (Num == 1)  Num is 2. EndIf.** 2. **If (Num==2)  Say(“True”). Otherwise  Say(“False”). EndIf.** |

**STRUCTURE OF THE PROGRAM**

|  |
| --- |
| ***[<Task declaration>]***  ***[<Object declaration>] [<Global variable declaration>]***  ***Task Lead: Start  Statement\_1. [Statement\_2].  … End. [<Task definition>]*** |