



DARTEdge.edge AI

Table of Contents

- **Table of Contents**
- **Instructions for ChatGPT to build a low level language.**
- **DARTEdge.edge Example**
- **ChatGPT Result**
- **DARTEdge IDE**

Instructions for ChatGPT to build a low level language.

Make use of the following rules to create a low level language where the language pattern is defined enclosed within the two apostrophes for the resulting compiler to define the input command pattern and result as an output, Finally title the new Code Language and Compiler "DART Edge" with the extension to my new code ".edge" and after the rules are followed in this inquiry run the example code pattern following the rules listed and build my first DART Edge Application Compatible with Android OS, iOS, Linux and Windows:

Circuit Rules for low level code language patterns:

Rule 1:

Denoting characters for Memory Allocation when embedding language rules in circuit memory uses two apostrophes '

Rule 2:

Accessing memory uses two ':' colons

Rule 3:

Combining DATA uses '|' pipe which is also known as concatenation.

Rule 4:

Peripheral access uses two '|', apostrophe and two ':', colons in the following order: '|::|'

Rule 5:

Function Begin starts with open curly brace '{' where the function name must precede this open curly brace with following format:

'():-:' The alphabet will be used to provide a name within the parentheses by user input.

Rule 6:

The language pattern "Is the same as" uses the following language pattern: '=='

Rule 7:

The language pattern "is not the same as" uses the following language pattern: '!='

Rule 8:

Combine strings with numbers or objects with the following language pattern: '+='

Rule 9:

Function End starts with close curly brace '}'

Rule 10:

Comment out commands for non-compile interference with two forward slash characters '//'

preceding the command or one slash followed by a star and after the command, a star followed by a forward slash.

Rule 11:

define variables by the following pattern: 'var ()'

Where the inputted variable is enclosed in the parentheses and allow the variable to represent any type of math or character.

Rule 12:

Write the code to text file with extension "txt" uses language patterns for compilation denoted by the following: '(file):'

Rule 13:

Compile Code Language Patterns into an Executable Application uses the following at the end of file: '};::' followed by the keyboard combination 'shift+enter'

Rule 14:

variables must be declared immediately after the open curly brace that follows the function name

Rule 15:

A variable can be set to another character or value with following denoted language pattern: 'var ()==()'

Rule 16:

Variables can be set not equal to one another with the following language pattern: 'var ()-=()'

Rule 17:

Explicit Character Patterns use the following language pattern: 'with' will represent the following sequence within enclosed function provided by user input.

Rule 18:

Explicit Character Patterns use the following language pattern: 'place' will tell the function how to perform context of itself at any position with the enclosing curly braces.

Rule 19:

Explicit Character Patterns use the following language pattern: 'andwith' performs and addition to the statement opening, 'with'

Rule 20:

Explicit Character Patterns use the following language pattern: 'thenplace' performs an addition to the preceeding 'then' statement.

Rule 21:

Explicit Character Patterns use the following language pattern: 'when' tells how the function will perform.

DARTEdge.edge Example

Only Compile the following after the current last rule of "21". A work DART Edge language application file will look like "DART Edge.edge":

Example Compile Code from rules:

```
(ExpectedColor):-: {  
  with  
  var (a)  
  Var (b)  
  
  {  
  
    place { var (a) with var (b) } {  
  
      when var (a) = blue and var (b) = yellow  
    }  
    thenplace var (a) with var (b) = green  
  
  }  
  
}::::
```

ChatGPT Result

NOTE:

It is important to note aside from the purpose of this DART Edge Code Project, low level code uses characters in a unique pattern to trigger electron CPU and GPU processes flowing by the the memory and related components on a silicon board to run any code language's own unique pattern as a command; much like magnetizing a screw driver with a car battery. The unique character patterns are fixed in the memory and memory related components to define the new low level language as an operating pattern.

ChatGPT Build v1.0.0 Results:

ChatGPT Utilized csharp to parse the inquiry:

// Example DART Edge code

```
(MyFunction):-: {  
  
    // Declare variables  
    var (a)  
    var (b)  
  
    // Set values for variables  
    var (a) == "Hello, "  
    var (b) == "world!"  
  
    // Concatenate strings  
    place { var (a) + var (b) } {  
  
        // Execute if statement  
        when var (a) == "Hello, " and var (b) == "world!"  
  
    }  
  
    // Set new value for variable  
    thenplace var (b) = "universe."  
  
}:::(MyFunction.edge)
```

DARTEdge IDE

DART Edge AI IDE:

ChatGPT Result:

Sure, here's a comparison of the core language patterns between C++ and DARTEdge, with the missing patterns in DARTEdge filled in:

Common Core Language Patterns:

Variable declaration:

C++: `int num;`

DARTEdge: `var (num)`

Variable assignment:

C++: `num = 42;`

DARTEdge: `var (num) == 42`

Function declaration:

C++: `void myFunction() {}`

DARTEdge: `()--:myFunction {
// Code goes here
}`

Function call:

C++: `myFunction();`

DARTEdge: `myFunction();`

If statement:

C++: `if (num == 42) {}`

DARTEdge: `when var (num) == 42 {}`

If-else statement:

C++: `if (num == 42) {} else {}`

DARTEdge: `when var (num) == 42 {}
elseplace {
// Code for else case goes here
}`

For loop:

C++: `for (int i = 0; i < 10; i++) {}`

DARTEdge: `for var (i) with 0 to 10 step 1 {}`

While loop:

C++: while (num < 100) {}
DARTEdge: while var (num) < 100 {}

Do-while loop:
C++: do {} while (num < 100);
DARTEdge: repeat {
// Code goes here
} when var (num) < 100

Comments:
C++: // This is a comment
DARTEdge: // This is a comment

Additional DARTEdge Language Patterns:

String declaration:
DARTEdge: var (str) == "Hello, world!"

Concatenation:
DARTEdge: var (str) += " I am a string."

Function return value:
DARTEdge: return (value)

File input/output:
DARTEdge: (file):: read (filename);
(file):: write (filename);

With these missing patterns filled in, here's an example of an IDE for the new

DARTEdge language:

<DARTEdge IDE>

[File] [Edit] [Run]

```
{  
  with  
    var (a)  
    var (b)  
  
  {  
    place { var (a) with var (b) } {  
  
      when var (a) == blue and var (b) == yellow  
    }  
  
    thenplace var (a) with var (b) += green  
  }  
}
```



```
}  
  
    return (a)  
};:::
```

This IDE has a simple menu bar with options for opening and saving files, as well as editing and running the code. The code itself is displayed in a code editor, with syntax highlighting for easier reading. When the user is finished writing their code, they can click the "Run" button to compile and execute it. If there are any errors, they will be displayed in a separate panel for easy debugging.