**FORTE**

**Motivation:**

**FORTE** is a procedural and a general-purpose object-oriented **programming language** going to be developed for adding better concepts like type checking, data abstraction, and object-oriented programming with more clear syntax, structure and keywords which makes the programming more efficient, elegant and straightforward for each software engineer or programmer.

**Targeted Audience:**

Programmers and developer.

**Language Paradigm:**

Multi-paradigm (procedural, object oriented).

**Case sensitivity:**

It is case sensitive.

**Keywords:**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Keyword** | **Keyword** |
| Exclude | Head | write |
| Dup | Whilst | Lett |
| Cip | State | Legal |
| Otherwise | Stag | Devoid |
| Execute | fract | Elseward |
| this | cont | Secret |
| Str | NULL | Status |
| Stop | read | Secured |
| Deliver | Global | fixed |

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Keyword** | **Keyword** |
| Exclude as delete | head as main | Write as cout |
| Dup as double | Whilst as while | Lett as char |
| Cip as int | State as case | Legal as true |
| Otherwise as else | Stagnant as static | Devoid as void |
| Perform as do | fract as float | Else if as elseward |
| this | cont | Secret as private |
| Str as string | NULL | Status as boolean |
| Stop as break | Read as cin | Secured as protected |
| Deliver as public | Global as public | Fixed as static |

**Data Types:**

* Cipher ( cip ) as int
* Duple ( dup ) as double
* Letter ( lett ) as char
* Status ( status ) as boolean
* Fraction ( fract ) as float
* Strand ( str ) as string

**Iterative statements:**

* Whilst loop //Pre-incremental
* Whilst loop //Post-incremental
* Do whilst loop

**Conditional Statements:**

* If
* otherwise

**Comments (Multi line + Single line):**

* ##
* #< >#

**Line terminator:**

* :

**Operators (Mention Each Operator with their class part):**

* +
* -
* /
* %
* \*
* &&
* ||
* !=
* ==
* =
* <
* >
* >=
* <=

**Punctuators (Mention each punctuator with their class part):**

* [ ]
* { } as <: :>
* ,
* .
* :
* ()
* “ ” as ~ ~
* ‘ ‘ as ` `

**Identifiers:**

* They must begin with a letter or underscore (\_).
* They must consist of only letters, digits, or underscore. No other special character is allowed.
* It should not be a keyword.
* It must not contain white space.
* It should be up to 31 characters long as only first 31 characters are significant.

**Syntax Specification:**

**Start\_End:**

global fixed head( )<:

#code here

:>

**Declaration :**

fract a=3.2:

lett ch = ` a ` :

str string1 = ~ hello world ~ :

**INPUT OUTPUT:**

Cip value:

Read(value):

Write(~ Value = ~ + value ):

**Functions:**

r\_devoid name()

<: :>

r\_devoid name(fract a)

<: :>

r\_dup name(dup a)

<:

return a:

:>

**Array 1D/2D:**

cip array[size]=[1,2,3]:

cip array[row , column]=[1,2,3

1,3,4]:

**Classes/structures:**

**class name**

<:

    # Access specifier

    secret:

   # Data Members

    str name:

global:

   # Member Functions()

   devoid setname(str n)  <:

Name = n:

:>

:>

**Struct name**<:

cip id:

Dup name:

:> :

**Loops :**

//***Pre incremental whilst loop***

cip i=0:

Whilst(i<=number: i++)

<:

#statement

:>

//***Post incremental whilst loop***

cip i=0:

Whilst(i<=number)

<:

#statement

i++:

:>

//***Post condtion checker loop***

Execute <:

#statement

:> whilst (Condition)

**Conditional statements:**

if(num>0)

<:

#statement

:>

Otherwise

<:

if(num<0)

<:

#statement

:>

:>

Otherwise

<:

#statement

:>