*USER’S GUIDE*

*VERSION 1.0*

**Directives:**

1. **#import**

* To include a header file, we need to use the keyword “**import**”

For example:

#import(stdio.h) // This syntax would include the library stdio.h into the code.

1. **#porichoy\_dao**

* This language allow you to define something. To define use keyword “**porichoy\_dao**”

For Example:

#porichoy\_dao pi 3.1416 //This defines the word pi as 3.1416

**Data Types:**

* Available data types in this language are following

|  |  |  |
| --- | --- | --- |
| Data Type | Meaning in C | Allocated Size |
| number | int | 4 bytes |
| numdes | Float | 4 bytes |
| longnumber | long long int | 8 bytes |
| doublenumdes | double | 8 bytes |
| borno | char | 1 byte |
| nil | void | No fixed size |

**Variable Declaration:**

The Rules for declaring variables are,

1. The variable must begin with ‘**var\_**’.
2. Immediate After the word ‘var\_’ there must be at least one character.
3. Numbers maybe included while naming the variables.
4. No special character is allowed while naming the variables.

For Example,

number abc //wrong

number var\_a\_bc //wrong

number var\_a\_bc //wrong

number var\_1bc //wrong

number var\_abc //Correct

number var\_abc1 //Correct

number var\_Abc1 //Correct

**Input And Output:**

To take input from user “**obtain**” keyword is used.

For Example,

obtain<var\_a> //This takes an input from user and put it in var\_a

For giving output “**speak**” keyword is used.

For Example,

speak<var\_a> //This outputs the content of var\_a

**Loop Instruction:**

1. **loopfor:**

Loopfor works like the for loop in C. The basic syntax for this loop is

**loopfor<initialization section | condition checking section | increment/decrement section>**

Here each section is mandatory.

For example:

loopfor<var\_i=0 | var\_i$$10 | var\_i inc>

[

speak<var\_i>|

]

1. **loop:**

It works like while loop in C. The basic syntax for this loop is

**loop< Condition >**

For Example:

loop<var\_i$$10>

[

speak<var\_i>|

var\_i inc|

]

1. **start while< >:**

It works like do while loop in C. At first it executes atleast one time then it checks the condition. The basic syntax is:

start

[

]

while<condition>

**Process Control Operators:**

1. **go:**

It forces next iteration to take place skipping all the code after this. Works like **continue** in C.

1. **ber\_ho:**

It terminates or exit the process or Loop. Works like **break** in C.

**Comment:**

To for single line comment “**!**” is used and for multiline comment “**!> <!**” is used.

For Example:

! This is single line comment

!> this is

multiline comment<!

**Relational Operator:**

1. **$$ :**

This Operator is used as greater than sign (>). Its return type is Boolean.

For Example:

9$$6 //Returns True

6$$9 //Returns false

1. **%% :**

This Operator is used as less than sign (<). Its return type is Boolean.

For Example:

9%%6 //Returns false

6%%9 //Returns true

1. **=**

This operator checks if the two inputs are equal or not. If equal it returns true, else returns false.

For Example:

9=6 //Returns True

6=6 //Returns False

**Logical Operator:**

1. **(+)**

This Operator works as logical and (&&) in the language.

1. **or**

This operator works as logical or (||) in the language.

1. **~**

This operator works as logical not in the language.

**Assignment Operator:**

In this language assignment operator is represented by “**@**”

For Example:

a@10 //This means 10 should be assigned to a

**Arithmetic Operator:**

1. **add:**

It works as addition operator.

For Example:

var\_result@10add12 //Result holds 22

1. **sub:**

It works as subtraction operator.

For Example:

var\_result@12sub12 //Result holds 0

1. **mul:**

It works as multiplication operator.

For Example:

var\_result@12mul12 //Result holds 144

1. **div:**

It works as division operator.

For Example:

var\_result@12div12 //Result holds 1

**Increment Decrement:**

1. **inc:**

This is used as increment operator.

For Example:

var\_a@10|

var\_a inc| // here a is 11

1. **dec:**

This is used as decrement operator.

For Example:

var\_a@10|

var\_a dec| // here a is 9

**Conditional Statements:**

1. **jodi:**

It checks the condition given after this. If the condition is true then the syntax behind this is executed. Basic syntax is:

**jodi<Condition>**

For Example:

var\_a@10

jodi<a=10>

[

speak< “HI”>|

]

1. **nahole:**

If the “jodi” condition fails then the compiler checks if the nahole’s condion is true or not. If it is true then it is executed. Basic Syntax is:

**jodi<Condition>**

**[**

**]**

**nahole<condition>**

**[**

**]**

1. **lastly:**

This works as the default statement for “jodi”. If no condition is true then this lastly would definitely be executed. Basic syntax is :

**jodi<Condition>**

**[**

**]**

**nahole<condition>**

**[**

**]**

**lastly**

**[**

**]**

It can work without nahole too. But jodi is mandatory for using lastly.

**jodi<Condition>**

**[**

**]**

**lastly**

**[**

**]**

**Brackets:**

1. **( ):**

This is used for importing the library.

For example:

#import(stdio.h)

1. **< >**

This is used as parenthesis in the language for replacement of “(“ and “)”.

For example:

speak< “abrar”>

loop< 1 >

var\_a = 10+<11-1> // this means 10+(11-1) would be updated to var\_a

1. [ ]

This is used as curly braces in the language. It is used for declaring the scope of some function.

For example:

if< 1 >

[

var\_a@10|

var\_b@12|

]

1. { }

This works as the square bracket in the language. May used for array operations.

For Example:

var\_a{10}|

**User Defined Variable:**

The keyword “udclass” is used for user defined variable.

For Example:

udclass user[

number var\_a|

borno var\_b|

]

**Pointer Operator:**

The keyword “**ptr**” is used to denote the pointer.

For Example:

number ptr var\_a| //declaration of a pointer type variable

**Reference Operator:**

Here “$” is used as a reference operator.

For example:

var\_a@$var\_b|

**File Control Operator:**

1. **Openthis:**

This help to open a file. Returns a file pointer.

1. **Close this:**

This helps to close a file.

**Compile Time Unary Operator:**

1. **fullsize:**

returns the size of a variable or a constant in compile time.

**Switch Case:**

1. **sw:**

The Keyword “sw” is used as a replacement of switch which holds the variable whose value would be matched in the cases.

1. **ca:**

These are case statements, if the condition in switch matches with any case, it executes.

1. **def:**

It is used as a default statement of switch case.

**Some Reserved Keyword:**

1. **ret:**

used to return some value from some function.

1. **|**

Used to declare end of a statement (Replacement of “**;**” in C)

1. **fixed:**

Used to make a variable static.

|  |
| --- |
| Table of Contents |

|  |  |
| --- | --- |
| **Index** | **Page Number** |
| **A. Directives:** 2  **B. Data Types:** 2  **C. Variable Declaration:** 2  **D. Input And Output:** 3  **E. Loop Instruction:** 3  **F. Process Control Operators:** 4  **G. Comment:** 4  **H. Relational Operator:** 4  **I. Logical Operator:** 5  **J. Assignment Operator:** 5  **K. Arithmetic Operator:** 5  **L. Increment Decrement:** 6  **M. Conditional Operator:** 6  **N. Brackets:** 7  **O. User Defined Variable:** 8  **P. Pointer Operator:** 8  **Q. Reference Operator:** 8  **R. File Control Operator:** 8  **S. Compile Time Unary Operator:** 9  **T.** **Switch Case:** 9  **U.Some Reserved Keyword:** 9 | |