HLL – Rough Draft

Tokenizer -> Generates tokens. We need to check valid and invalid tokens. Example 2.2. is a valid token in our language. But 2.2.2. is not. ( Question: Do we use | instead of . to end our sentences???)

If illegal tokens are generated by the Tokenizer -> Throw errors do not parse.

Parser -> Valid Production rules. Clueless .. Does not compute..

**Data Types:**

integer, floating, boolean

**Operators:**

Numerical: +, -, \*, /

Comparison operators: <= , < , > , >=, ==, !=

Logical: AND, OR, **NOT**

**\*\*\*\*\***

**Unary( concern -> needed)?**

**Declaration/ Assignment:**

**Concern\*\*\* Do we keep the assignment and declaration in the same sentence???**

**This could significantly increase our complexity in terms of production rules**

**Example: INTEGER Identifier := Expr**

**INTEGER IDENTIFIER { expr | [] | , IDENTIFIER} –Current assessment – not good ☹.**

integer a, b, c.

integer a := 1, b := 2, c := 3.

integer a := 0.

boolean d.

boolean d := true.

boolean d := false.

floating e.

floating e:= 2.1.

**Collections:**

Array Integer [10] a.

Array floating [20] b.

Array boolean [5] c.

Array integer [10] d := (1, 2, 3, 4).

d [1] := 4.

**Functions:**

function foo ; integer(integer param1, floating param2){

}

**Conditionals**:

when(a > 10){

}else when( a < 10 && a> 5){

}

Else { }

**Loop:**

Loop(a := 0 , a:= 10, a := a+ 1){

}

Or

Loop( a <= 10 ){

a := a+ 1 .

}

**Example Factorial Program**

function factorial ; integer ( integer fact ) {

when ( param == 1 ) {

gimme 1

}

integer factVal .

factVal := fact \* factorial ( fact -1 ).

gimme factVal.

}