**2nd Milestone**

1. Name:
2. Design:
3. Grammar of the language driving the implementation of the compiler and
4. byte-code interpreter/runtime environment.
5. Include information about the interpreter.
6. Discuss the parsing technique you will employ.
7. Also discuss any data structures used by the parser and interpreter.

Notes:

One step at a time.  
  
No seriously, start with expressions and operators, work upwards to statements, then to functions/classes etc. Keep a list of what punctuation is used for what.  
  
In parallel define syntax for referring to variables, arrays, hashes, number literals, string literals, other builtin literal. Also in parallel define your data naming model and scoping rules.  
  
To check whether your grammar makes sense focus on a level (literal/variable, operator, expression, statement, function etc) and make sure that punctuation and tokens from other levels interspersed or appended/prepended is not gonna cause an ambiguity.  
  
Finally write it all out in EBNF and run it through ANTLR or similar.  
  
Also best not to reinvent the wheel. I normally start off by choosing sequences to start and end statement blocks and functions, and mathematical operators, that are usually fundamentally C-like, ECMAScript-like, Basic-like, command-list based or XML-based. This helps a lot cos this is what people are used to working with.

**Designing the Grammar**

**Operators: Description:**

|  |  |
| --- | --- |
| **“+”** | **Addition** |
| **“-”** | **Subtraction** |
| **“\*”** | **Multiplication** |
| **“ / “** | **Division** |
| **“ % “** | **Modulus** |
| **“&&”** | **AND** |
| **“ || ”** | **OR** |
| **“< > “** | **NOT** |
| **“ <- “** | **Assignment** |
| **“ $ “** | **Delimiter** |
| **“ <- -> “** | **Comparison** |
| **“ ! <- “** | **Not Equal To** |
| **“ > <- “** | **Greater Than Equal To** |
| **“ < <- “** | **Lesser Than Equal To** |
| **“ > “** | **Greater** |
| **“ < “** | **Lesser** |
| **T** | **True** |
| **F** | **False** |
| **“ ( “** | **Left Parenthesis** |
| **“ ) “** | **Right Parenthesis** |

|  |  |
| --- | --- |
| **“ </ “** | **Begin Block** |
| **“ /> “** | **End Block** |

**Keywords:**

|  |  |
| --- | --- |
| **num** | **Integer/Float** |
| **boole** | **Boolean** |
| **takein** | **Input** |
| **giveout** | **Output** |
| **rl** | **New Line** |
| **agar** | **if** |
| **phir** | **else** |
|  | **Stack** |
|  | **Push** |
|  | **Pop** |
| **gol gol** | **While Loop** |
|  |  |