**[SLogo](http://www.cs.duke.edu/courses/compsci308/current/assign/03_slogo/index.php) Part 2 : Basic Implementation**

This part of the project is intended to get a basic version of your API working. Each sub-team should implement their side of the project without regard to the internal workings of the other sub-team's code, using the agreed upon API to communicate back and forth.

**Specification**

For the front end, create an environment for the user to develop SLogo programs. At the very least, it should allow the user to:

* enter commands to the turtle interactively
* display results of the turtle executing commands  
  The turtle should be driven via commands entered by the user and optionally by buttons or other user controls. You should provide feedback so that the user is always aware of the turtle's position, heading, or other parameters affected by the environment. Note, the turtle starts in the center of the display which is considered (0, 0).
* display errors that may result from the user's commands
* choose an image to use for the turtle
* specify a color to use for the pen
* show commands previously run in the environment (even better, make these commands directly clickable to execute)
* access an HTML formatted help page  
  it could just be a "language reference", i.e., the [**command descriptions given with the assignment**](http://www.cs.duke.edu/courses/compsci308/current/assign/03_slogo/commands.php), or you could extend it to include features of the GUI
  + We should just be able to use the swing guidelines that we had in class – however, instead of using buttons on a panel, you should be able to look at the page just by typing in a command instead of doing something else.

For the back end, create an initial version of the SLogo language:

* [**interpret these basic commands**](http://www.cs.duke.edu/courses/compsci308/current/assign/03_slogo/commands.php)
* report errors that may result from incorrectly entered commands

[**SLogo**](http://www.cs.duke.edu/courses/compsci308/current/assign/03_slogo/index.php)**Basic Commands**

* While the command names below are given in all capital letters, SLogo is not case sensitive and should accept any correct spelling of the command names. Note that any values used as parameters to a command may be result of calling any of the commands below or a value stored in a variable. Thus all commands return a value as described below. Also, commands can be formatted over any number of lines, including multiple commands on the same line, but they must be separated by one or more spaces.
* **Basic Syntax**

|  |  |  |  |
| --- | --- | --- | --- |
| **Token** | **Syntax** | **Semantics** | **Examples** |
| **Constant** | -?[0-9]+\.?[0-9]\* | any real valued number note, to avoid potential ambiguity in parsing there should not be a space between the negative sign and the value | 50 -1.3 |
| **Command** | [a-zA-z\_]+(\?)? | any word can be a command-name note, all commands return a numeric value note, built-in commands are given below and user-defined commands cannot reuse those names | forward fd |
| **List** | [  ] | these brackets enclose a list of zero or more commands or variables note, to make parsing easier, these will always be separated from other tokens by spaces | [ fd 50 rt 90 ] |

* **Turtle Commands**

|  |  |
| --- | --- |
| **Name(s)** | **Description** |
| FORWARD *pixels* FD *pixels* | moves turtle forward in its current heading by *pixels* distance returns the value of *pixels* |
| BACK *pixels* BK *pixels* | moves turtle backward in its current heading by *pixels* distance returns the value of *pixels* |
| LEFT *degrees* LT *degrees* | turns turtle counterclockwise by *degrees* angle returns the value of *degrees* |
| RIGHT *degrees* RT *degrees* | turns turtle clockwise by *degrees* angle returns the value of *degrees* |
| SETHEADING *degrees* SETH *degrees* | turns turtle to an absolute heading returns number of degrees moved |
| SETXY *x* *y* GOTO *x* *y* | moves turtle to an absolute screen position, where (0, 0) is the center of the screen returns the distance turtle moved |
| PENDOWN PD | puts pen down such that when the turtle moves, it leaves a trail returns 1 |
| PENUP PU | puts pen up such that when the turtle moves, it does not leave a trail returns 0 |
| SHOWTURTLE ST | makes turtle visible returns 1 |
| HIDETURTLE HT | makes turtle invisible returns 0 |
| HOME | moves turtle to the center of the screen (0 0) returns the distance turtle moved |
| CLEARSCREEN CS | erases turtle's trails and sends it to the home position returns the distance turtle moved |

* **Turtle Queries**

|  |  |
| --- | --- |
| **Name** | **Description** |
| XCOR | returns the turtle's X coordinate from the center of the screen |
| YCOR | returns the turtle's Y coordinate from the center of the screen |
| HEADING | returns the turtle's heading in degrees |
| PENDOWN? PENDOWNP | returns 1 if turtle's pen is down, 0 if it is up |
| SHOWING? SHOWINGP | returns 1 if turtle is showing, 0 if it is hiding |

* **Math Operations**

|  |  |
| --- | --- |
| **Name** | **Description** |
| SUM *expr1* *expr2* + *expr1* *expr2* | returns sum of the values of *expr1* and *expr2* |
| DIFFERENCE *expr1* *expr2* - *expr1* *expr2* | returns difference of the values of *expr1* and *expr2* |
| PRODUCT *expr1* *expr2* \* *expr1* *expr2* | returns product of the values of *expr1* and *expr2* |
| QUOTIENT *expr1* *expr2* / *expr1* *expr2* | returns integer quotient of the values of *expr1* and *expr2* |
| REMAINDER *expr1* *expr2* % *expr1* *expr2* | returns integer remainder on dividing the values of *expr1* by *expr2* |
| MINUS *expr* ~ *expr* | returns negative of the values of *expr* |
| RANDOM *max* | returns random non-negative number strictly less than *max* |
| SIN *degrees* | returns sine of *degrees* |
|  |  |
| COS *degrees* | return cosine of *degrees* |
|  |  |
| TAN *degrees* | returns tangent of *degrees* note, since this function is defined continuously, take appropriate action (i.e., divide-by-zero might silently return zero) |
|  |  |
| ATAN *degrees* | returns arctangent of *degrees* note, since this function is defined continuously, take appropriate action (i.e., divide-by-zero might silently return zero) |
| LOG *expr* | returns natural log of *expr* |
| POW *base exponent* | returns *base* raised to the power of the *exponent* |

* **Boolean Operations**

|  |  |
| --- | --- |
| **Name** | **Description** |
| LESS? *expr1* *expr2* LESSP *expr1* *expr2* | returns 1 if the value of *expr1* is strictly less than the value of *expr2*, otherwise 0 |
| GREATER? *expr1* *expr2* GREATERP *expr1* *expr2* | returns 1 if the value of *expr1* is strictly greater than the value of *expr2*, otherwise 0 |
| EQUAL? *expr1* *expr2* EQUALP *expr1* *expr2* | returns 1 if the value of *expr1*and the value of *expr2* are equal, otherwise 0 |
| NOTEQUAL? *expr1* *expr2* NOTEQUALP *expr1* *expr2* | returns 1 if the value of *expr1* and the value of *expr2* are not equal, otherwise 0 |
| AND *test1* *test2* | returns 1 if *test1* and *test2* are non-zero, otherwise 0 |
| OR *test1* *test2* | returns 1 if *test1* or *test2* are non-zero, otherwise 0 |
| NOT *test* | returns 1 if *test* is 0 and 0 if *test* is non-zero |

* **Control Structures**

|  |  |
| --- | --- |
| **Name** | **Description** |
| REPEAT *expr* [ *command(s)* ] | runs the *commands* given in the list the value of *expr* number of times returns the value of the final command executed note, the value of the current iteration, starting at 1, is automatically assigned to the variable *:repcount* so that it can be accessed by the *commands* |
| IF *expr* [ *command(s)* ] | if *expr* is not 0, runs the *commands* given in the list returns the value of the final command executed |
| IFELSE *expr*  [ *trueCommand(s)* ]  [ *falseCommand(s)* ] | if *expr* is not 0, runs the *trueCommands* given in the first list, otherwise runs the *falseCommands* given in the second list returns the value of the final command executed |