```
$Id: asg3-smalltalk-mbst.mm,v 1.10 2021-05-11 13:40:17-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cse112-wm/Assignments/asg3-smalltalk-mbst
URL: https://www2.ucsc.edu/courses/cse112-wm/:/Assignments/asg3-smalltalk-mbst/
```

1. Overview

Smalltalk is a pure object-oriented language, where everything is an object, and all actions are accomplished by sending messages, even for what in other languages would be control structures. The syntax of Smalltalk, like that of Scheme, is exceedingly simple. It, along with Simula 67, propagated object-oriented features into many other languages, although languages like C++ and Java are hybrid, not purely object-oriented.

In this project we will be using Gnu Smalltalk, the Smalltalk for those who can type. References:

https://ftp.gnu.org/gnu/smalltalk/
https://www.gnu.org/software/smalltalk/
https://www.gnu.org/software/smalltalk/manual/
https://www.gnu.org/software/smalltalk/manual-base/
https://learnxinyminutes.com/docs/smalltalk/
http://www.angelfire.com/tx4/cus/notes/smalltalk.html

2. Running Gnu Smalltalk interactively

Smalltalk is an interpretibve language and can be run from the command line or by a scriptl To run **gst**, add the following to your **$HOME/.bash_profile**:

export PATH=$PATH:/afs/cats.ucsc.edu/courses/cse112-wm/usr/smalltalk/bin

When running **gst** interactively, use the command **rlwrap** to gain access to the readline arrow keys so that you can recover earlier typed lines. Notice the unexpected(?) operator precedence. Example:

```
-bash-$ rlwrap gst
GNU Smalltalk ready
st> 2 + 3 * 6.
30
st> 2 raisedTo: 128.
340282366920938463463374607431768211456
st> (-1 arcCos * -1 sqrt) exp + 1.
(0.0+0.00000000000000012246467991473532i)
st> 5 sqrt + 1 / 2.
1.618033988749895
st> ^D
```

Instead of explicitly typing **rlwrap** every time you start **gst** interactively, create the command **wgst** by putting the following in in your **$HOME/.bash_profile**:

alias wgst='rlwrap gst'

3. The program `mbint.st`

Complete the implementation of **mbint.st**, a Smalltalk implementation of the Basic interpreter from previous projects. The following classes are of interest:

`Object subclass: Debug`
A singleton class with a debug flag which can be turned on from the command line. Multiple **`-d`** options increase the level of debugging for more information.

`Object subclass: MiniBasic`
The root class to hold methods that are usefully inherited by other classes. **`unimplemented:`** is used to print error messages for classes not fully implemented. **`prefix`** is used by the multiple **`printOn:`** methods to label the output when **`self`** is printed.

`MiniBasic subclass: Expr`
`Expr` is the abstract base class from which all other expression classes inherit.

`Expr subclass: NumExpr`
Holds numbers of class **`FloaatD`** or **`Complex`**. Complex numbers are not specifically handled, but may be produced by expressions such as **`(-1 sqrt)`**.

`Expr subclass: VarExpr`
Each instance represents a simple variable. The class variable **`varDict`** holds the symbol table.

`Expr subclass: UnopExpr`
Implements unary operators (functions). The **`value`** method is to be implemented.

`Expr subclass: BinopExpr`
Implements binary operators. The **`value`** method is to be implemented.

`Expr subclass: ArrayExpr`
Holds the array table as a class varible and array names as instances.

`Expr extend`
Is an extension of **`Expr`** placed later in the file so that it can refer to its subclasses. **`Expr>>parse:`** accepts the array prefix notation of the intermediate format and produces the proper class instances.

`MiniBasic subclass: Stmt`
Is the root of the statment interpreters. The class variable **`stmtNr`** contains the number of the next statement to be implemented. **`labelDict`** is to contain the labels from the original program as keys and the corresponding statement numbers as values.

`Stmt subclass: DimStmt`
Interprets the **`dim`** statement.

`Stmt subclass: LetStmt`
Evaluates an expression and assigns it to the appriate variable.

`Stmt subclass: GotoStmt`
The **`interp`** method assign a new value to **`stmtNr`**.

`Stmt subclass: IfStmt`
Evaluates the expression. If true, behaves like the **`goto`** statement.

`Stmt subclass: InputStmt`
Reads input for each variable mentioned on the input line and stores its value

in the appropriate symbol table. If the token read **`isNumber`** it is returned. otherwise **`NaN`** (`0.0/0.0`) is returned.

`Stmt subclass: PrintStmt`
Prints the list of arguments in readable format.

`Stmt subclass: NullStmt`
A dummy slot used when the intermediate language has an empty statement. **`interp`** does nothing.

`Stmt extend`
Is an extension of **`Stmt`** that can refer to the subclasses. **`Stmt>>parse`** converts the intermediate langage into an object structure.

`MiniBasic subclass: Interpreter`
Initializes the progra structure and controls the **`Stmt`** interpreters. **`Interpreter>>print`** is for debugging only.

`Object subclass: Main`
Takes care of scanning the command line option and operand, produces suitable error messages if appropriate, and calls the interpreter.

4. What to submit

Submit **`mbint.st`** and **`README`**. Also **`PARTNER`** if doing pair programming.