Braxton Gunter *(beg2119)* - Tester
Clyde Bazile (*cb3150*) - Language Guru
John Anukem (*jea2161)* - Systems Architect
Sebastien Siclait (*srs2232*) - Tester
Terence Jacobs (*tj2316*) - Project Manager

Newbie: A Language for Those New to Programming

INTRODUCTION

Traditional high-level programming languages are often too cryptic and difficult for new users to understand. The goal with Newbie is to create a pseudo-code like programming language aimed to simplify the programming experience for beginner developers. This will allow new coders the ability to design, implement and better understand common algorithms without the frustration of learning specific programming syntax. Our standard library will specifically allow for easy implementation of basic algorithms involving linked lists, graphs, and trees.

FEATURES

Type inference - The compiler will automatically infer variable types at compile time. This will make it easier for users as they no longer need to declare parameter or variable types. As such, if we declare coms = 4115 and class = 'plt', the compiler will interpret these variables as an integer and string, respectively. This will be done using hindley milner type inference with a standardized notation for common data types.

Automatic variable declaration - During compile-time, we will be identify all variables and their corresponding types. These variables will be automatically initialized to predictable default values. This means that variables do not need to be explicitly declared or initialized.

Dynamic Instance Attribution - The developer will have the ability to programmatically add and edit attributes on an object instance.

DEFAULT DATA TYPES AND PRIMITIVES

num	*Int or Float*
char	*Int or UTF-8 Encoded Symbol*
bool	*Boolean Values true or false*
list	*Ordered Set of AnyType*
string	*Wrapper for Character Array*

KEYWORDS & OPERATORS

+	*Addition*
-	*Subtraction*
*	*Multiplication*
/	*Division*
^	*Exponent*
=	*Assignment*
%	*Modulo*
! | not	*Negation*
== | equals	*Equivalence*
!= (not equals)	*Difference*
<	*Less Than*
>	*Greater Than*
<= | ≤	*Less Than or Equal To*
>= | ≥	*Greater Than or Equal To*
//	*Inline Comment*
/* … */	*Multiline Comment*
and	*Union*
or	*Intersection*
class	*Class Definition*
def	*Function Definition*
return	*End*
null	*No value*
true	*Boolean*
false	*Boolean*

CONTROL STATEMENTS

if	*Conditional*
else	*Catch-all Conditional*
else if	*Additional Conditional*
for each	*Loop for Items in Set*
while	*Loop within Conditional*
break	*Exiting Loops*
continue	*Progress to Next Iteration of Loop*

SYNTAX

The syntax of our language will resemble pseudo-code.
Tabs and newlines are indication of scope.
The *string* datatype is essentially a wrapped character array.
Specifying types is not necessary, but can be done in whole or in part.
.noob file type extension

SAMPLE CODE

```
BREADTH-FIRST SEARCH // uses queue

1. def BFS(G, s)
2.    for each vertex 'u' in G
3.          u.color = “WHITE” // undiscovered
4.          u.d = ∞
5.          u.π = null // predecessor
6.    s.color = “GRAY” // has white vertices connected to it
7.    s.d = 0
8.    s.π = null
9.    Q = null // queue is null
10.   Enqueue(Q, s)
11.   while Q ≠ null
12.         u = Dequeue(Q)
13.         for each 'v' in G.Adj[u]
14.               if v.color == “WHITE”
15.                     v.color = “GRAY”
16.                     v.d = u.d + 1
17.                     v.π = u
18.                     Enqueue(Q, v)
19.         u.color = “BLACK” //discovered along with everything connected
20.
21. class NODE
22.   string color
23.   num d
24.   NODE π
25.
```