COMS W4115:
Programming Languages and Translators

GPA: White Paper
09/22/2006

mbp2103 : Michael Pierorazio, Project Manager
dg2267 : Dmitry Gimzelberg, Systems Architect
vaz2001 : Vincenzo Zarrillo, Systems Integrator
jmg2105 : Juan Gutierrez, Quality Assurance

GPA:
General Purpose to Assembly Language

Introduction

GPA is a light weight, general purpose language whose design incorporates the low-level understanding of assembly language while allowing the use of high level constructs such as functions and loops. The premise behind GPA comes from programmers who are used to working close to hardware but require some high level constructs, and as a result, GPA falls somewhere between assembly code and the C language in terms of abstraction. GPA is simple to learn for those who know another language, but due to its high level syntax, it is also easy for beginners to understand. The goal of GPA is to be simple, powerful, and fast.

Features

Too often programmers become frustrated when their code does not run because they forgot a semicolon or some other form of punctuation; this leads to unclear error-messages from the compiler which confuses the user. Stream-lined syntax is fundamental to GPA. As such, there are no curly braces or semi-colons of any kind; instead, scope is determined by whitespace.

Since GPA is designed to be simple for beginners, it has no explicit types. Those with no experience in programming languages are often confused by data types so the simple construct for variable declaration is '**variable:** x'. However, not all users are beginners, so there is a shorter way to declare variables, '**v:** x'. This construct allows more advanced users to write code faster. In fact, almost all keywords in GPA have long and short forms, corresponding to beginner and advanced modes. For instance, functions are declared as '**procedure:** function1([arguments])' or '**p:** function1([arguments])'. Both types of keywords can be interchanged within the same program.

Goals

Since this language is more low-level than C, it can translate almost one-to-one with assembly code blocks. This will enable GPA to be compiled easily to x86 assembly code. This may lead to GPA running faster than other higher level languages. If there is time, basic library support will be added to GPA.

Sample Code

```
import:math.gpa //:a simple import scheme would include copying
i:string.gpa    //:the code found in the target file directly into
                //:the file being compiled.

//: Here is a one-line comment: "import:" and "i:" are interchangeable.
/*: This is a comment block in gpa
   notice how ':' will delineate
   all special words.
*/:

go!() //:"go!" is the main function in gpa
  variable:p=0  //:notice how "variable" and "v" are interchangeable
  v:a,b,c       //:multiple variables can be declared in one line
  v:d=1,e=2,f=3 //:vars can be declared and instantiated in one line

  v:array[10]               //:an array declaration
  v:string='hello, world!' //:a string declaration and instantiation

 /*:the same abilities to declare variables transfer to variable
    declaration in loops. loop syntax below
    loop:[variable declaration(s), comma separated]:[condition(s) to
    break loop, separated by logical operators]
 */:

  //:start of loop, shift away two spaces from the go! block
  loop:i=0,j=0:i >= 99 || p > 10
    if: i < 9
      foo(i)
    else:
      bar(j++)
    i++
  //:end of loop, shift back two spaces to return to go!

  r='1'~'1' //:concatenation
  s=1+1     //:addition
  print: 'The result: ' r ' ' s  //: this prints "The result: 11 2"
  println: 'We are done flexing GPA's muscles.'

  a=10
  b=47
  c=gcd(a,b)
  println: 'The gcd of ' a 'and ' b 'is ' c '.'

procedure:gcd(a,b)
//: could also have been writen as "p:gcd(a,b)"
  if:(a == 0 && b == 0) //: also could be if: (a == 0 AND b == 0)
    b = 1
  elsif: (b == 0)
    b = a
  elsif: (a != 0)
    loop:: (a != b)
      if (a <b)
        b -= a
      else
        a -= b
 return:b

p:foo()
 return:1
p:bar()
 return:1
```