GA$$P Programming Language

A breath of fresh air

Team Members

Adam Fowler (ajf2177) - Language Guru

Patrycja Przewoznik (pap2154) - Tester

Sam Weissmann (spw2136) - Manager

Swan Htet (sh3969) - System Architect

Yuanxin Yang (yy3036) - System Architect

1. Language Overview
GA$$P is an statically typed object-oriented general purpose programming language with its roots in the C++ and Java programming languages offering inheritance with an everything is an object worldview.

The syntax of GA$$P will be familiar to anyone who has previously programmed in Java or C/C++ as GA$$P uses braces to delineate scope and includes all of the usual types and operations (e.g. int, float, boolean, char, and string) and variables with static type declarations. The inclusion of strong static typing will allow GA$$P to protect the user from a host of errors using type checking on functions, methods, and operators. GA$$P further offers the ability for users to define their own types with inherent behavior.

The goal of the GA$$P language is to enhance the familiar base of Java and C++ by replacing primitive types with objects and removing null expressions which Tony Hoare famously referred to as his “Billion dollar mistake”.

- Strong static typing allowing type checking for safer code
- Everything is an object (MVP)
- No nulls (Optional instead)
- Inheritance / virtual method dispatch (stretch goal)
- Exceptions

2. Standard Library
GA$$P will feature a core set of built in operators, data types as objects, functions, and I/O functionality.

2.1. Operators

Arithmetic	+, -, *, /, %, **, ++, --
Comparison	==, !=, <, <=, >, >= ,
Logical	&& (and), || (or), ! (not)
Bitwise	&, |, ^, <<, >>, ~,

2.2. Built-in Data Types

Name	Description	Operators	Syntax
int	32 bit integer	All	`int x = 3;`
float	64 bit float	All	`float x = 3.14;`
char	8 bit character	All	`char x = “a”;`

boolean	Binary true/false value	Logical, Comparison	`bool x = true;`
array	Fixed length collection	Comparison	`int[3] x = [1,2,3];`
string	Immutable string	+ (concat)	`str x = "Hello World";`

2.3. Built-in functions and I/O

GA$$P's standard library will include a small set of core functions; will support IO with the stdin, stdout and stderr channels; and will also have a larger set of mathematical functions available via a linked C library.

Built in functions	
`min()`	Returns smallest number
`max()`	Returns largest number
`sqrt()`	Computes square root of a number
`random()`	Generates a random number
`exp()`	Calculates a number raised to some power
`quicksort()`	Performs quicksort on an array

2.4. Control Flow and Scope

GA$$P supports if/else statements, switch statements, for loops, and while loops. Scope is defined using curly braces.

2.5. Reserved Words

Control	`if, else, switch, case, for, while, break, return`
Types	`int, float, bool, char, array, string,`
Other	`true, false, class, const, void, private, public, extends, print`

3. Syntax

The syntax of GA$$P is derived primarily from C++ and Java.

3.1. Function Syntax

GA$$P borrows much of its function syntax from C++. Functions pass arguments by value and does not provide support for default arguments, variable arguments, or overloading.

Function declaration

```
return_type fn_name(data_type arg1, data_type arg2, ...){
   ;
}
```

Function with a return type

```
int foo(int a, int b){
      return a*b;
}
```

Without a return type

```
void foo(){
      ;
}
```

Functions may be invoked by name within other functions in the following manner:

```
void foo(){
      printf("Hello World!");
}

void foo1(){
      foo();
      print("Wassup")
}
```

3.2 Comments

Multiline comments are delimited with /* comment here */ syntax. GA$$P does not support single line comments.

4. Examples

Quicksort implemented in GA$$P

```
/* initially low and high refers to index 0 and index n-1, where n
is the size of the array
*/

int partition (int a[], int low, int high)
{
      int pivot = a[high];
      int i = (low-1);

      for (int j = low; j <= high-1; j++)
      {
            if (a[j] <= pivot)
            {
                  i++;
                  int temp = a[i];
                  a[i] = a[j];
                  a[j] = temp;
            }
      }
      int temp = a[i+1];
      a[i+1] = a[high];
      a[high] = temp;
      return (i+1);
}

void quickSort(int a[], int low, int high)
{
      if (low < high)
      {
            int split = partition(a, low, high);  /* split is the
partition index */
            quickSort(a, low, split-1);
            quickSort(a, split+1, high);
      }
}
```