Geo

A geometric solution language

Proposal

Group Members

PM (make sure conductible, write-ups):

Qi Wang (qw2197)

Language Guru (design language syntax):

Yuechen Zhao (yz2877)

System Architect (based on syntax, build compiler):

Zichen Chao (zc2321)

Tester (test test test test):

Ziyi Luo (zl2471)

(COMS W4115) Programming Languages and Translators

Fall 2015

Sept. 27, 2015

1. Description

Geo is an objective-orient geometric solution language that enables students, physicists, mathematicians, as well as other geometry-interested professionals to solve spatial problems efficiently. It involves functionalities that computes relationships among geometric figures, including lines, circles, and rectangles, etc. Users are able to define figures, set their moving patterns, and perform various analysis (including static and dynamic analyses) of the interacting figures. If desired, a user can also change the moving pattern or shape of objects.

2. Syntax

Syntax of Geo is simple and friendly. In this section, the basic syntax of Geo will be introduced.

2.1. Comments

Geo has same comment syntax as C and Java. It allows single-line comment and multiple-line comment like the following example:

```
// This is a single-line comment
/* This is a multiple-line
   comment */
```

2.2. *Panel* declaration

All works of Geo must be assigned in *Panel*s. The declaration of *Panel* as well as basic configurations is shown as bellow:

```
/* Define panel */
@Panel panelDemo
 /* Mode choices: Console & Figure */
@Mode Console
/* Default: Cartesian coordinate system */
//@Co Cart
…...
/* End panel indicates the boundary of a panel*/
@End Panel
```

Note that, several panels are allowed to exist in one Geo program. All *Panel*s in one Geo are allowed to work either serially or parallelly during the run time. Console mode presents all result of analyses in a command line window. Figure mode presents a much more intuitive view of the geometric objects in the program.

2.3 Primitive data types and Variable declaration

Primitive data types of Geo includes C-like data types (i.e., *Bool*, *Int*, *Float* and *String*) and geometric types. Note that the names of all basic data types start with an uppercase letter. Declarations of C-like data types are shown as follows:

```
stat = true; //Declare a Bool named stat
int1 = 3; // Declare an Int named int1
float1 = 3.;// Declare a Float named float1
string1 = ”Hello world”; //Declare a String named string1
```

Geometric types include *Dot*, *Line*, *Rect*, and *Circle*. A *Dot* is defined by its coordinate (i.e., x and y in Cartesian coordinate system), a *Line* by its slope and intersect or two coordinates (named a and b), a *Rect* by a coordinate and its width and height, and a *Circle* by its center and radius.

```
/* Dot definition */
dot1 = [2,3];
dot2 = [3,4];

/* Line definition */
line1 =(3,4); //Line(a:Float,b:Float) denotes the function y=ax+b
line2 = Line([0,0],[5,10]); // Two points determine a line

/* Rectangle definition */
rect1 = Rect([2,5],3,5); //Rect(Coordinate:Dot, Width:Float, Height:Float)

/* Circle definition */
circle1 = Circle([3,5],5); //Circle(Center:Dot,Radius:Float)
```

2.4 Function

Function of Geo allows one return value, but note that all input parameters of a function are reference parameters in default, which means that if the input parameters are modified in the function, the parameter will keep that change after the execution of the function. ”Const” is strongly recommended to add in the declaration of a function in order to avoid the modification of input parameters.

```
//Function with a Circle as the input and a Dot as the output
Function Get_circle_center(c:Circle):Dot:
return c.Center;
End
```

```
//A Function without return value.
Function Print_circle_center(c:Circle Const):
Println(c.Center);
End
```

2.5 Control Flow

Geo includes If-Elif-Else statements. The following example uses a build-in function Line.Intersect(Circle):int to examine the intersect point number(s) of a line and a circle.

```
If(line1.Intersect(circle1) > 0):
Println("Intersection");
Elif(line1.Intersect(circle1) == 0):
Println("Tangent");
Else:
Println("Neither intersection nor tangent");
End
```

Besides, we have loop control flow like While and For-in statements:

```
While(true):
…
End

//Declare a List of Circles
list_of_circle = {Circle([3.4,5.1],5.2),Circle([1,9],6.5),Circle([2.3,4.1],7.3)};
For c:Circle in list_of_circle:
Println(c.Center);
End
```

2.6 Dynamic analysis statement

Geo includes a special Run statement for all kinds of dynamic geometric analysis. For all geometric-type objects with *Timer*(s), the properties of these geometric objects may be shown and analysis in a Run statement. An example of Run is shown as bellow:

```
t1 = Timer(-0.5); //Timer(Timestep:float)
t2 = Timer(1);

/* As for a line, the standard expression is ax+b=y */
line1.AddTimer(t1,a); //For each time in Run, a+(Timestep of t1) -> a
circle.AddTimer(t2,r);//(x-a)^2+(y-b)^2=r^2

/* Run line1 & circle1 20 times. Note that both line1 and circle1 must have a timer for
dynamic analysis*/
/* Run(Times:int,Geo_obj1:Geo_type,Timer_of_Geo_obj1:Timer, ...) */
Run(20,line1,t1,circle1,t2):
```

```
Println("Intersection length: ",line1.IntersectLength(circle1));
/* Line.Expression:string */
Println(line1.Expression);
/* Circle.Expression:string */
Println(circle1.Expression);
End //End of Run
```

Run is a powerful statement that allows diverse dynamic analyses. For example, one can set the distance between a dot and the center of a circle to be fixed, and set the length between a segment and a circle to be dynamic changing.

3. Build-in functions support

Geo includes a large number of built-in functions for basic geometric analysis. For instance, Geo has Intersect function for *Line* and *Circle*, *Line* and *Triangle* .etc, Distance function for *Dot* and *Dot*, *Dot* and *Line* .etc. Besides, Geo includes many basic system input & output commands.

```
/* Line.Intersect(Circle):Int */
line1.Intersect(circle1);

/* Line.Intersect(Circle):Float */
line1.IntersectLength(circle1)

/* Dot.Distance(Dot):Float or Dot.Distance(Line):Float */
dot1.Distance(dot2);

/* Console mode ONLY */
/* Println(String/Float/Int/Dot)*/
Println(line1.Expression);
Println(circle1.Expression);
/* Println function allows different data type in one function, divided by comma. The following example prints a string and float*/
Println(dot1.Distance(dot2),” ”,dot1.Distance(line1));
```

4. Samples of Geo

4.1 A static analysis sample

```
@Panel paneldemo
@Mode Console
dot1 = [5,5];
circle_list = {Circle([2,4],6),Circle([0,0],5)};
For c:Circle in circle_list:
```

```
Println(”Distance between dot1 and the center of ”,c.Expression, ” is ”,
dot1.Distance(c.center) );
End
@End Panel
```

4.2 A dynamic analysis sample

```
@Panel paneldemo
@Mode Console
t1 = Timer(0.1);
line1 = Line(2,3);
line1.AddTimer(t1);
circle1 = Circle([3,5],4);

Run(100,line1,t1):
If(line1.Intersect(circle1)>=0):
Println(line1.IntersectLength(circle1));
Else:
Println(”Not intersect.”);
End
End

@End Panel
```