Simple Geographical Analysis Language(SGAL)

Language Reference Manual

Loganathan Vijay Sambandhan(CVN)
lvs2116

Introduction

The Simple geographical language is built to easily create simple Objects in a Geographical space, and perform simple spatial analysis like nearest neighbor, objects within a boundary. Using the Language real world features like Schools, Roads, County Boundary can be created and the spatial queries about their whereabouts in relation to other features can be answered.

The language will support only the Geographic Coordinate system. All features should have their Coordinates in latitude longitude Decimal Degrees.

There are three kind of spatial objects which can be created,

Point: has just a Latitude Longitude.
Poly-line: will be a collection of points.
Polygon: will also be a collection of Points but will form a closed Object.

Lexical Conventions

Comments
Comments begin with the characters /* and continue until */.

Identifiers
Identifiers consist of letters, digits, and underscores. The first character must be a letter and can be followed by Numbers or Digits. The maximum length for an Identifier will be 8 letters. The Language is case sensitive.

Keywords
The following are keywords and may not be used as Identifiers.

Line
Polygon
Point

true
false
iter
if
else

Function
Number
Boolean

Constants
There is a constant number constant.

Number Constants
A Number consists of a decimal point and at least one digit.

Boolean Constants
A Boolean constant can have a value of true or false.

Spatial Types
There are three spatial types provided with the language.

Point
Line
Polygon

A point can be declared like

```
Point p =  [72.0 56.3] ;
```

The numbers withing the [ ] denote the X and Y of the Point. They are to be separated by a blank space. Once a point is declared the X and Y can be accessed by using the **.,** like p.X, p.Y.

The Polygon and Line are an collection of points and can be declared with points being separated with a comma like,

```
Line pLine =  [-79.78 48.32, -79.76 48.35, -79.86 48.35, -79.76 48.85] ;

Polygon pPolygon =  [-79.78 48.32, -79.76 48.35, -79.86 48.35, -79.76 48.85] ;
```

Points in a line or polygon can be accessed using the **IterPoints** loop.

Loops

There is a single iterator loop present in the language called as **IterPoints**. This loop will run the block of code once for each point in the polygon or Line object given to it. For example

Polygon pPolygon -> [-79.78 48.32, -79.76 48.35, -79.86 48.35, -79.76 48.85] ;

```
IterPoints (Point p in pPolygon)
{
        /* Loop Code */
}
```

The statements for the iterations have to be defined within the **{ }.**

Conditional Statements

If the expression between the "**( )"** is true then the First Block of Statements get executed and if not then the block of statements after Else gets executed.

If (*Expression*) { statements } else {statements }

Both the If and the Else Statements blocks have to be defined within **{}.**

Operators and Expressions

***** (multiplication)denotes multiplication between two numbers.

/ (division)denotes division between two numbers.

- (subtraction)denotes subtraction between two numbers. When used with a single number it indicates a negative value.

+ (addition)denotes addition between two numbers.

= (assignment). Assigns the value of the object on the RHS to the LHS object. Provided both sides are of same type.

&& Logical And operator.

|| Logical OR operator

! Logical Not operator

< Less than

> Greater than

<= Less than equal to

>= Greater than equal to

== Equivalence operator

All the comparison operators can be used for only numbers and no other types.

; Statement delimiter.

Built in Functions

There are some utility functions inbuilt to work off the existing spatial types.

Length (Line pLine); The length function takes in a Line Object and returns the Length of the Line.

Area (Polygon pPolygon); The area function takes in a Polygon Object and returns the Area of the Polygon.

MBD (Line OR Point OR Polygon); The MBD function takes in a Shape object and return a Polygon object which represents the Minimum Bounding rectangle of the Shape.

The structure of SGAL Program

A program consists of declaration statements, function definitions and function Calls.

Variable Declaration
A variable declared has to be initialized to a Value where its being declared.

Number i = 2323.4404;

Complex datatypes like Line, Point, and Polygon take a collection of coordinates as an Input.

Line pLine = [-79.78 48.32, -79.76 48.35, -79.86 48.35, -79.76 48.85] ;

Function Declaration
Function has to be declared using the Key word Function and all its contents should be within { }.

```
Function  returnType functionName( type par1, type par 2)
        {

        }
```

Function Calls
Function calls can be made by using the Function name followed by the input parameters.

Scope
SGAL is statically scoped. All variables declared in a function are available in that function. Variables declared in statements outside functions are available to all code declared below that statement. Within its scope there can be only one declaration with a particular name.

Example

The program below declares two objects of type Point and Polygon and checks if the point lies within the Polygon or not.

```
/* Declare Point */
Point p = [-79.78 48.32];

/* Declare polygon */
Polygon  polygon = [-79.78 48.32, -79.76 48.35, -79.86 48.35, -79.76 48.85] ;


/*  Utility function to determine whether a map point is inside of a given polygon*/

Function Boolean IsWithin (Point p1, Polygon polygon)
{
                /* An Impossible Coordinate */
                Point previousPoint = [ - 999,-999];

                        IterPoints ( Point p2 in polygon)
                        {

                                Boolean inPoly = false;

                        /* Algorithm to determine if point is within a Polygon */
                        if ((p2.X < p1.X && previousPoint.X >= p1.X) ||
                                (previousPoint.X < p1.X && p2.X >= p1.X))
                                {
                                if (p2.Y + (p1.X - p2.X) / (previousPoint.X - p1.X) * (previousPoint.Y
                                - p2.Y) < p1.Y)

                                        {
                                                inPoly = true;
                                        }

                                /* Store this Point in polygon as we will need it next iteration */

                                         previousPoint = p2;
                                }

                                if (inPoly)
                                        return true;
                        }

                        return false;
}

/* finally call the function */
        isWithin (p,polygon);
```