ANGLE

Language Specification

Group 7
Tejal Kulkarni - CS21BTECH11058
Deepshikha - CS21BTECH11016
Nitya Bhamidipaty - CS21BTECH11041
Muskan Jaiswal - CS21BTECH11037

Contents

1 Motivation

The motivation behind designing this language is to combine computation and geometry. Though there are other software for creating diagrams, they do not natively support programming. The idea is to make abstract concepts more intuitive by easing the process of making geometrical figures. We feel that this language will help automate the process of visualizing geometrical figures. An user can integrate external data with code to create diagrams. Our language is mainly intended for high school geometry. We feel that integrating our language with school curriculum can help nurture programming mindset at a young age. We hope that our language can be extended further to become the latex of geometry.

2 Lexical Convention

2.1 Comments

Both single and multi-line comments are supported

```
$ single-line comment
$$
    multi-line comment
$$
```

Nested comments are not allowed.

```
$$ not a $$valid comment$$ $$
```

2.2 Whitespace

All whitespaces are ignored except in places where white space are required for separation of tokens. In 'Angle', each statement is terminate by a new line or EOF.

2.3 Identifiers

The rules for identifiers is the same as in C/C++. It starts with letter or underscore, followed by letters, digits or underscores. Our language is case-sensitive.

2.4 Reserved Keywords

if	elif	else	repeat	until	int	real	label	point
bool	line	circle	tri	para	regPoly	and	or	not
func	stop	advance	return	true	false	fig	void	angle

3 Data Types

3.1 Primitive Data Types

Data Types	Description
int	32-bit signed integer value
real	64-bit signed floating point number
label	sequence of characters (string)
bool	stores true/false value
point	coordinates of a point
angle	angle between three points in degrees

3.2 Non-primitive Data Types

Data Types	Description
array	sequence of primitive/non-primitive data types
line	line joining two points
circle	circle with a center and radius
tri	triangle
para	parallelogram
regPoly	regular polygon

4 Declarations

Program consists of various entities such as variables, function and figures. Each of these entities must be declared before use.

4.1 Variable declaration

4.1.1 Primitive

```
int a = 0
real a = 4.5
label s = "Angle"            $ Should use ""
bool x = true
point p = (2,3,"A")          $ (x-coord, y-coord, label)
angle a = < p q r >          $ by default , display is true
angle b = < p q r , false >  $ false implies not to be displayed
```

4.1.2 Non-Primitive

```
$$ ARRAY $$
   int a[4] := {1,2,3,4}     $ array declaration
```

```
 $$ LINE $$
     line l := p-q             $ line declaration where p,q are points
     line l[] := p<->q-r->s    $ line array declaration
     $$
        <-> : denotes line
        -   : denotes segment
         -> : denotes vector
     $$


$$ CIRCLE $$
    circle c := Circle(p,r)    $ (centre,radius)

$$ TRIANGLE $$
    tri t := Tri(p,q,r)        $ three points declaration
    tri t := Tri(4,8,5)        $ SSS declaration
    tri t := Tri(5,4)
    $$
        RHS declaration :
            first argument is hypotenuse
            second is one of the other side
    $$

$$ PARALLELOGRAM $$
    para p := Para(3,60,8)     $ SAS declaration
    $ 2 adjacent sides and the angle contained by them

$$ REGULAR POLYGON $$
    regPoly r := RegPoly(3,8)  $ (no of sides , length of sides)
```

4.2 Declaration Scope

In our language, an identifier in global scope can be redefined in any other scope including functions and figures. An identifier defined in a scope cannot be redefined within the same scope.

4.3 Function declaration

default values

A function like in other programming languages can be used to modularize your code and can be called anywhere in the global scope or figures (described in the next section).

```
func return_type function_name(argument_list) {
    $ statements
}
```

If function return type is not void then you must include return statement in the function.

4.4 Figure declaration

Figure consists of a set of geometrical objects that could be displayed with a given scale and center. Scale is the size of the figure and center is the origin for the objects part of that figure. Functions can be called inside figures as well.

```
fig figure_name(scale,center) {
   $ statements
}
```

Example use of figures:

```
$ Creates a series of kites

fig kite (scale := 1, center := (0, 0)){

    para k := Para(5, 60, 5)
    k.DIAGONAL()

}

int N := 4 $ No of kites

repeat (int i := 1|i <= N|i++)
{
    kite(1, (0, i))
}
```

5 Operators

5.1 Compatible Types +

- POINT
- LABEL
- REAL, BOOL, INT, ANGLE are arithCompatible

We have two new operators , parallel (||) and perpendicular (|−) .

```
p || q
$ check if lines p and q are parallel, returns boolean value

p |- q
$ check if lines p and q are perpendicular, returns boolean value
```

Example:

```
line l1, l2, l3

$ Can have multiple lines aswell
if (l1 || l2 || l3)
    return 0;
```

5.2 Precedence

Operator	Symbol	Associativity
Parenthesis	( )	left to right
Member access	.	left to right
Unary Negation	not	right to left
Increment	++	right to left
Decrement	–	right to left
Exponent	^	left to right
Multiplication,Division,Modulo	*,/,%	left to right
Addition,Subtraction	+,-	left to right
Relational	<=,>=,<,>,==,!=	left to right
Logical	and,or	left to right
Parallel	‖	left to right
Perpendicular	|–	left to right
Assignment	:=,+:=,-:=,^:=,/:=,%:=	right to left

6 Statements

6.1 Compound Statements

A compound statement is a sequence of statements enclosed by braces, for example:

```
{
   int a := 9
   line l := p-q
}
```

6.2 Control Flow

6.2.1 Conditionals

```
$ if - elif - else
    if (i <= 3){
        $ stmt
    }
    elif (i > 3){
        $ stmt
    }
    else{
        $ stmt
```

```
}
```

```
$ if - else
    if (i <= 3){
        $ stmt
    }
    else{
        $ stmt
    }
$ single statement if
    if (i <= 3)
        $ stmt
```

6.2.2 Loop Statements

```
    $$ SIMILAR TO FOR LOOP $$
    repeat(int i := 0 | i < n | i++ )
    {
       $ statemtents
        advance        $ continue statement
    }

    $$ SIMILAR TO WHILE LOOP $$
    until(i < 1)
    {
       $ statements
        stop           $ break statement
    }
```

7 Built-in and Standard Library functions

We plan to implement some standard libraries to provide support for the following domains. Some examples are shown below.

- Line

```
point p := INTERSECTION(a,b)           $ a and b are lines
point p := MIDPOINT(l)                 $ l is a line segment
point p := MIDPOINT(a,b)               $ a, b are points
real dist := SHORTEST_DISTANCE(a,b)    $ a, b are points
line p := ANGLE_BISECTOR(a,b)          $ a, b are lines
line p := LINE_AT_ANGLE(40,a,q)        $ (angle,line,point)
```

- Circle

```
$ o:center, r:radius
circle c := Circle(o,r)
$ q is point of contact of line p to circle c
line p := c.TANGENT(q)
```

```
$ c1 and c2 are circles
point p[] := INTERSECTION(c1,c2)
line l[] := COMMON_TANGENT(c1,c2)
```

- Triangle

```
tri t := Tri(a,b,c)
point p := t.CIRCUMCENTRE()
$ excentre opposite to point q of the triangle t
point p := t.EXCENTRE(q)
point p := t.INCENTRE()
point p := t.ORTHOCENTRE()
$ q is point of triangle from which altitude is drawn
line l := t.ALTITUDE(q)
$ q is vertex of triangle from which median is drawn
line l := t.MEDIAN(q)
point p := t.CENTROID()
```

- Parallelogram

```
para p := Para(a,40,b)
line l[] := p.DIAGONAL()
```

- Regular Polygon

```
regPoly h := regPoly(3, 6)
h.AREA()
h.PERIMETER()
```

- We have two member functions, area and perimeter for all 2D geometric objects.

```
|| p ||          $ distance between p and origin
|| p - q ||      $ distance between p and q
```

8 Sample program

```
func fact(int n){
    if (n == 1 or n == 0){
        return 1
    }
    else{
       return (fact(n-1)*n)
    }
}
```

```
$ Figure to display the circumcircle of a triangle
fig figure_1 (1,(0,0)){
    point a := (1,2)
    point b := (2,7)
    point c := (7,1)
    tri t := Tri(a,b,c)
    point o := t.CIRCUMCENTRE()
    real r := || a-o ||
    circle c := (o,r)
}

$ Figure to display a parallelogram and its diagonals and print its area
fig figure_2 (1,(2,0)){
    para p := Para(3,60,8)
    real area := p.AREA()
    print(area)
}

$ Figure to display adjacent circles with radius equal to sequence of factorials
fig figure_3(scale := 2, centre := (0,0)){
    point p := (0,0)
    int n := 3
    repeat(int i := 1 | i <= n | i++ ){
        Circle(p,fact(i))
        p.x := p.x + 1
    }
}

$ Syntax to display figures$
figure_1()
figure_2()
figure_3()
```