



## **1. Lexical Conventions**

### **1.1 Tokens**

SGML has following types of tokens. identifiers, keywords, strings, geometric operators, and other separators. Blanks, tabs, newline and comments are ignored except as they are used to separate tokens. Some white spaces are required to separate otherwise adjacent identifiers, keywords, constants, strings, and operators. If the input stream is parsed into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

### **1.2 Comments**

Comments in SGML either a single line comment or multi-line comments, begin with “/\*” and end with “\*/”.

### **1.3 Identifiers**

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore ‘\_’ counts as a letter. SGML is case insensitive; the upper and lower case letters are treated same, so the identifier ‘ab’ and ‘AB’ are considered to be same identifiers. There is no limit on the length of identifiers.

### **1.4 Keywords**

The following identifiers are reserved as keywords within the SGML language, and may not be used as any variable names:

*x, y, z, start\_point, end\_point, center, radius, height, side, vertices, area, volume, length, perimeter, angle, midpoint, point, line, triangle, square, rectangle, parallelogram, circle, cylinder, cone, Sphere, cube, distance, perpendicular, intersection, between, Inscribed, within, find, given;*

### **1.5 Constants**

Constants in the SGML language are integer and float.

#### **1.5.1 Integer Constants**

An integer consists of a sequence of one or more consecutive digits.

#### **1.5.2 Floating Constants**

A floating constant consists of an integer part, a decimal point, a fraction part, an e, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing.

### **1.6 Other Kinds of Tokens**

Each token will be described in detail in the following sections.

( ) ; = + - ,

## **2. Types**

SGML supports following

Object type.

*point, line, triangle, square, rectangle, parallelogram, circle, cylinder, cone, Sphere, cube*

Attribute type

*x, y, z, start\_point, end\_point, center, radius, height, side, vertices, area, volume, length, perimeter, angle, midpoint*

Relationship type

*distance, perpendicular, intersection, between, Inscribed, within*

### 3. Operators

Language has four basic operator types.

Given: Provides the object declaration and assigns the value of attributes

Find: Computes derived attribute or question from template

Draw: Draw the object

=: Assigns the value of object attribute

### 4. Expressions

Below is general layout of expression in SGML. The details are given in ANLR grammar.

Expression = Given <fact> find <question>

Fact= <object\_type> object\_name

; <attribute\_type> of object\_name = <attribute\_value>

Question= <derived attribute type> of object\_name;

Example:

Give Point A; coordinate of A=(1,2); find x of A;

### 5. Scoping Rule

Based on object name type, the meaning of attribute type will differ. Example

Radius of AA;

AA could be circle, sphere, cylinder or cone. Actual attribute value will be determined by scoping rule.

## ANLR Grammar

```
class SGMLParser extends Parser;
options {
    buildAST = true; // Enable AST building
    k = 2;          // Need to distinguish between ID by itself and ID
ASSIGN
}

//tokens {
// STATEMENTS;
//expr;
//}

//file
// : (expr !)+ EOF!
// { #file = #([STATEMENTS],file); }
// ;
//Question expression
question : give_fact ask_question draw_object;
give_fact : ((GIVEN) ( expr1 )+ (subexp1)+);
expr1 : object_type (object_name)+ COMMA;
subexp1 : attribute_name ASSIGN attr_value (unit)? COMMA ;
attribute_name : attr_name OF object_name;

ask_question: ((FIND) ( expr2)+);
expr2 : (derived_attr_name OF object_name ASSIGN QQ)|expr3;
expr3 : question_bank;
question_bank : DISTANCE BETWEEN POINT object_name AND object_name
|ANGLE BETWEEN LINE object_name AND object_name
| AREA OF INTERSECTION BETWEEN CIRCLE object_name AND object_name
| AREA OF CIRCLE INSCRIBED WITHIN TRIANGLE object_name;

draw_object: (DRAW (object_name)+)*;
object_type : OBJECT_TYPE;
object_name : IDENT;
attr_name : ATTR_NAME;
derived_attr_name : DERIVED_ATTR;
attr_value : LBRAC NUMBER COMMA NUMBER RBRAC |NUMBER|object_name;
unit: UNIT;
```

```

class SGMLLexer extends Lexer;
options {
    testLiterals = false; // By default, don't check tokens against keywords
    k = 2;                // Need to decide when strings literals end
    charVocabulary = '\3..\377'; // Accept all eight-bit ASCII characters
}
// other token
PLUS : '+' ;
MINUS : '-' ;
LBRAC : '(' ;
RBRAC : ')';
ASSIGN : '=' ;
SEMI : ';' ;
DOT : '.' ;
COMMA : ',' ;
QQ : '?';
//operator
GIVEN : "Given";
FIND : "find";
OF : "of";
DRAW : "draw";

//attribute type
X : "x";
Y : "y";
Z : "z";
COORDINATE : "coordinate";
START_POINT : "start_point";
END_POINT : "end_point";
CENTER : "center";
RADIUS : "radius";
HEIGHT : "height";
SIDE : "side";
VERTICES : "vertices";
ATTR_NAME : X|Y|Z|START_POINT|END_POINT|CENTER|
RADIUS|HEIGHT|SIDE|VERTICES;
// derived attribute type
AREA : "area";
VOLUME : "volume";
LENGTH : "length";
PERIMETER : "perimeter";
ANGLE : "angle";
MIDPOINT : "midpoint";
DERIVED_ATTR: "area"|"volume"|"length"|"perimeter"|"angle"|"midpoint";
//object type
POINT : "point";
LINE : "line";
TRIANGLE : "triangle";
SQUARE : "square";
RECTANGLE : "rectangle";
PARALLELOGRAM : "parallelogram";
CIRCLE : "circle";
CYLINDER : "cylinder";
CONE : "cone";

```

```

SPHERE      :           "sphere";
CUBE        :           "cube";
OBJECT_TYPE : "point"|"line"|"triangle"|"square"|"rectangle"|"parallelogram"|
"circle"|"cylinder"|"cone"|"sphere"|"cube";
//object relationship
RELATION : "distance"|"perpendicular"|"intersection"|"between"|"inscribed"|"within";
DISTANCE  : "distance";
PERPENDICULAR : "perpendicular";
INTERSECTION : "intersection";
BETWEEN    : "between";
INSCRIBED  : "inscribed";
WITHIN     : "within";

//unit
UNIT : "MM"|"CM"|"M"|"KM"|"MILE";
IDENT
  options {testLiterals=true;}
  : ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')*
  ;

// A little unorthodox: most punctuation characters get their own rule,
// but since we're using "(" and ")" in the parser, we need parenthesis
// to match as keywords, Thus, we set testLiterals true for this rule.
PARENS
options {
  testLiterals = true;
}
: '(' | ')';

protected LETTER : ( 'a'..'z' | 'A'..'Z' );
protected DIGIT  : '0'..'9';

ID
options {
  testLiterals = true;
}
: LETTER (LETTER | DIGIT | '_' ) * ;

NUMBER : (DIGIT)+;

// Strings are "like this ""double quotes"" doubled to include them"
// Note that testLiterals are false so we don't have to worry about
// strings such as "if"
STRING : ""! ( "" ""! | ~( "" ) ) * ""!;

WS : ( ' '
      | '\t'
      | '\n' { newline(); }
      | '\r'
      ) { $setType(Token.SKIP); }
;
QUESTION : 'Q' NUMBER '.'
{ $setType(Token.SKIP); };

```

### Examples:

**Q1.** Given point A,B;  
Coordinate of A= (1,2);  
Coordinate of B= (2,3);  
Given Line AB;  
Starting\_point of AB=A;  
End\_point of AB=B;  
Find length of AB=?  
Find midpoint of AB=?  
Draw line AB

**Q2.** Given circle C1;  
Radius of c1=5 cm;  
Find perimeter of C1=?  
Find area of C1=?  
Draw circle C1;

**Q3.** Given triangle ABC;  
Vertices A of ABC=(0,0);  
Vertices B of ABC= (2,2);  
Vertices C of ABC=(3,3)  
Find area of ABC=?  
Find angle of ABC=?

### 9. References

- [1] *C Reference Manual*, Dennis M. Ritchie.
- [2] Mirage LRM