Compiler Construction/ Complier Design
Assignment Stage:1

Group 1	**Group 2**
Member1: Shubham Sharma ,2012C6PS493P	Member3:Vivek Sahu, 2012A7PS058P
Member 2:Suyash Ahuja,2012C6PS558P	Member4:Piyush Dosi, 2012C6PS317P

1. Language Features

Name	**Description**	**Example**

Keywords	The various keywords present in the language include ’if’, ’return’, ’for’, ’building’, ’floor’, ’.start’, ’.end’	floor f{
		}
		here floor is a keywprd

Identifiers	Identifiers in the language are of the form [_a-zA-Z][_a-zA-Z0-9]* ie. They begin with an alphabetical character followed by an alphanumeric character.	An example of identifier is ‘w1’.For instance: int w1;
Data types	The data types supported in our language are int, float, distance, point, wall, door, window and ratio.	ratio r = 3:4;

Operators and operations defined on each data type	**‘=’** is the Assignment operator defined on all data types.
	‘–‘operator performs subtraction operation on int and float
	‘*’ operator computes simple multiplication of two int or float specified around it.
	‘+’ operator computes simple addition of int or float data types.
	‘||’thickness operator works on walls.
	‘&’operator works on points
		Point p1=(w1,w1.start,d);
		Wall W1=t||(p1&p2);

Function	A procedure can be called with any number of parameters of available data types , and only one return data type. In the example shown Wall is return type of procedure.	Wall function(Wall w1,Point p2) {
		
		}

Scope Rules	Static scoping , nested scoping.	

Conditional Statements	Our language supports if construct, with conditions being checked via composition of relational operators (<,<=,>,>=,==) and identifiers.	if(a==b) {
		}

Iterative Statements	The language supports a ‘for’ loop design as described in the example on the right.	for(count=1;count<12;count=count+1)

	In the example shown count is an integer type variable.	
Name	**Description**	**Example**
I/O Operations	Since our language is a domain specific language that writes programs only for the purpose of drawing, therefore the programs written in our language do not take inputs, hence no Input operations. Also the output has to be given by default , as such there is no output operator as well.	None
Expression	Expressions are of the forms as shown in the examples to the right.	i=i+1 or i=i-1 or i=i*I or i=i/I or W1=t||(p1&p2);

Assignment Statements	Different data types have different types of assignment statements. However the assignment operator is always '='.Examples of various assignment statements have been shown on the right.	ratio r = 3:4;
		point p1=(3,4);
		point P1=w1.start;
		point P1=w1./end;
		Point p1=(w1,w1.start,d);
		Wall W1=t||(p1&p2);
		Wall W1=t||(w2,p1,theta1)
		Wall w1=(w2,w2.end,90,14);
		Door d1= (p1,p2);

2. Lexical Units: The following are the lexical units in our programming language:

Pattern	Token	Purpose
EPS	TK_EPSILON	Epsilon
(	TK_ROUND_OPEN	Delimiter
)	TK_ROUND_CLOSE	Delimiter
{	TK_CURLY_OPEN	Delimiter
}	TK_CURLY_CLOSE	Delimiter
,	TK_COMMA	Delimiter
int	TK_INT	Datatype
float	TK_FLOAT	Datatype
distance	TK_DISTANCE	Datatype

point	TK_POINT	Datatype
wall	TK_WALL	Datatype
door	TK_DOOR	Datatype
window	TK_WINDOW	Datatype
ratio	TK_RATIO	Datatype
for	TK_FOR	Keyword “for”
;	TK_SEMICOLON	Delimiter
if	TK_IF	Keyword “if”
return	TK_RETURN	Keyword “return”
=	TK_EQUALTO	Assignment Operator
[	TK_SQUARE_OPEN	Delimiter
]	TK_SQUARE_CLOSE	Delimiter
:	TK_COLON	Ratio Operator
||	TK_THICKNESS	Thickness Operator
.start	TK_DOTSTART	Keyword “.start”
.end	TK_DOTEND	Keyword “.end”
&	TK_AND	Point Operator
*	TK_MUL	Multiply Operator
-	TK_MINUS	Subtraction Operator
+	TK_PLUS	Addition Operator
/	TK_DIVIDE	Division Operator
building	TK_BUILDING	Keyword “building”
floor	TK_FLOOR	Keyword “floor”
<	TK_LESS_THAN	Conditional Operator
>	TK_GREATER_THAN	Conditional Operator
!=	TK_NOT_EQUAL_TO	Conditional Opeator
[_a-zA-Z][_a-zA-Z0-9]*	TK_IDENTIFIER	Alphabetical Characters
[-+]?[0-9]*\.?[0-9]+	TK_LITERAL	Rational Number

3. LL(1) Grammar:

```
<Prog> ===> <functions><building>
<functions> ===> <function><functions>/ EPS
<function> ===><type>
<id>TK_ROUND_OPEN<params>TK_ROUND_CLOSETK_CURLY_OPEN<stats>TK_CURLY_C
LOSE
<type> ===>
TK_INT/TK_FLOAT/TK_DISTANCE/TK_POINT/TK_WALL/TK_DOOR/TK_WINDOW/TK_RATI
O
<params>===> <type><id>TK_COMMA<params>/EPS
<stats>===> <stat> <stats>/ EPS
<stat>===> <funcstats>/<returnstats>/<forfunction>/<iffunction>
<forfunction>===>TK_FOR
TK_ROUND_OPEN<exp>TK_SEMICOLON<exp>TK_SEMICOLON<exp>TK_ROUND_CLOSET
K_CURLY_OPEN<stats>TK_CURLY_CLOSE
<iffunction>===>TK_IF
TK_ROUND_OPEN<exp>TK_ROUND_CLOSETK_CURLY_OPEN<stats>TK_CURLY_CLOSE
<returnstats>===>TK_RETURN<Kim>TK_SEMICOLON
<funcstats>===><normalstats>/<id><LF3>TK_SEMICOLON
<normalstats>===><type><id><Zip>
<Zip>===>TK_SEMICOLON/TK_SQUARE_OPEN<Kim>TK_SQUARE_CLOSETK_SEMICOLON
/=<E>TK_SEMICOLON
<LF3>===> =<E>/TK_SQUARE_OPEN<Kim>TK_SQUARE_CLOSE=<E>
<Kim>===><id>/<literal>
<E>===><Kim><K>/TK_ROUND_OPEN<LF1>
<LF1>===><id><LF4>/<literal>TK_COMMA<Kim>TK_ROUND_CLOSE
<LF4>===>TK_COMMA<LF5>
<LF5>===><id><LF6>/<literal>TK_ROUND_CLOSE
<LF6>===>TK_ROUND_CLOSE/TK_COMMA<Kim>TK_ROUND_CLOSE
<K>===>TK_COLON<Kim>/TK_THICKNESS
TK_ROUND_OPEN<id><LF9>/TK_ROUND_OPEN<LF7>/<arithmeticoperators><Kim>/EPS
/TK_DOTSTART/TK_DOTEND
<LF7>===><id>TK_COMMA<LF8>/<literal>TK_COMMA<buildparams>TK_ROUND_CLOSE
/TK_ROUND_CLOSE
<LF8>===><id><LF0>/TK_ROUND_CLOSE/<literal>TK_COMMA<buildparams>TK_ROUND
_CLOSE
<LF0>===>TK_ROUND_CLOSE/TK_COMMA<buildparams>TK_ROUND_CLOSE
<buildparams>===> <Kim>TK_COMMA<buildparams>/EPS
<LF9>===>TK_AND
<id>TK_ROUND_CLOSE/TK_COMMA<id>TK_COMMA<Kim>TK_COMMA<Kim>TK_ROUN
D_CLOSE
<arithmeticoperators>===>TK_MUL/TK_MINUS/TK_PLUS/TK_DIVIDE
<building> ===> TK_BUILDING<id> TK_CURLY_OPEN<newstats>TK_CURLY_CLOSE
<newstats>===><body><newstats>/EPS
<body>===><forrelatedstuff>/<floor>/<funcstats>/<ifrelatedstuff>
```

```
<ifrelatedstuff>===>TK_IF TK_ROUND_OPEN<exp>TK_ROUND_CLOSETK_CURLY_OPEN<funcstats><newstats>TK_CURLY_CLOSE
<forelatedstuff>===>TK_FOR TK_ROUND_OPEN<exp>TK_SEMICOLON<exp>TK_SEMICOLON<exp>TK_ROUND_CLOSETK_CURLY_OPEN<funcstats><newstats>TK_CURLY_CLOSE
<floor>===>TK_FLOOR<id><arr>TK_CURLY_OPEN<funcstats><newstats>TK_CURLY_CLOSE/<id><arr>TK_CURLY_OPEN<funcstats><newstats>TK_CURLY_CLOSE
<arr> ===> TK_SQUARE_OPEN <literal> TK_SQUARE_CLOSE/ EPS
<exp>===>id<conditionaloperator><E>
<conditionaloperator>===>TK_LESS_THAN<LF10>/TK_GREATER_THAN<LF10>/TK_EQUALTO<LF10>/TK_NOT_EQUAL_TO
<LF10>===>TK_EQUALTO/EPS
<id>===> TK_IDENTIFIER
<literal>===>TK_LITERAL
```

4. Test Cases:
   a. Test Case 1:

```
building b{
        floor f1{
                point p1 = (0,0);
                point p2 = (100,0);
                wall w1 = 2||(p1 & p2);
                wall w2 = 3||(w1,p2,90,40);
                point p4 = w2.end;
                point p3 = (p4,w2,5);
                window win1 = (p1,p2);
                door d1 = (p3,p4);
        }
}
```

   b. Test Case 2:

```
wall func(wall w, ratio r, float d,){
        point p3 = w.start;
        point p=(w,p3,r);
        wall w1 = 2||(w,p,90,d);
        return w1
}

building b{
        floor f {
                point p1 = (0,0);
                point p2 = (25,0);
                point p3 = (25,25);
                point p4 = (0,25)
                wall bound = 2||(p1 & p2);
                wall w;
                ratio r = 1:2;
                w = func(bound,r,10,);

        }
}
```

c. Test Case 3:

```
building b1{
        floor f1{
                point p1 = (0,0);
                Point p2 = (100,0);
                wall w1 = 2||(p1 & p2);
                wall w2 = 3||(w1,p2,90,40);

        }

        floor f2{
                point p3 = (30,0);
                point p4 = (70,0);
                wall w3 = 2||(p3 & p4);
        }
}
```

d. Test Case 4:

```
building b{
        floor f{


                int i = 1;
                point p1 = (0,0);
                point p2 = (25,0);
                point p3 = (5,0);
                wall bound= 2||(p1 & p2);
                if(i==1)
                {
                        door d = (p1,p3);
                }
        }
}
```

e. Test Case 5:

```
Building b{
        Floor f{

                wall W[4];
                int i =0;

                for(i = 0 ;i<4;i = i+1){
                        int var_a = 2*i;

                        point pt_d = (0,var_a);
                        point pt_e = (25,var_a);

                         w[i] = 2||(pt_d & pt_e);

                }
        }
}
```