BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Floor Plan Language

A domain specific programming language

Shubham Sharma and Suyash Ahuja

3/10/2014

ODU
TOILET 5'-4"x8'-0"
6'-0" WIDE BALCONY
ODU
BEDROOM 9'-8.5"x17'-5"
KITCHEN 8'-0"x11'-3"
BEDROOM 10'-0"x11'-3"
MASTER BEDROOM 10'-6"x16'-10"
5'-6"x3'-6"
LIVING ROOM 11'-6"x11'-7.5"
TOILET 8'-0"x6'-0"
DINING 14'-0"x11'-10"
TOILET 5'-6"x8'-0"

A special purpose programming language designed for drawing floor plans in an easy and efficient manner.

Programming Language for designing floor Plans

Description of Application Domain: The idea is to simplify the job of civil engineers by providing a programming language to code the process of developing building plans. The language will prove useful to plan rooms, spaces, doors, mazes and other features in an entire building. Hostels can be easily constructed.

Wait there's more! If you are a pac-man fan! You can even design mazes. Difficult to navigate labyrinths/puzzles can be designed. The approach is kept to be top down i.e from building to floors to walls.

The plan is not just a bird eye view but an accurate measured drawing to scale floors in a building. Along with planning, it can also help in cost estimation by providing facility to calculate wall areas, perimeters etc.

Description of Problem/Tasks that can be solved:

1. Specification of building design by a top down approach. Building is divided into floors. Floors are made of walls, doors and windows.
2. Calculating cost of construction/painting of a building /floor to substantial degrees of accuracy. Calculating perimeter for fencing etc.
3. Patterns like in a puzzle maze can be very difficult to design (Can be done easily using iterative approach in this language).
4. Floors can be compared to know whether they are same or not. This can be useful for navigation purposes.

Programming features that the language provides:

1. **Strongly Typed -** There is a need to specify data types explicitly.
2. **Readability** – The language has high level specifications and hence is easily readable.
3. **Writability** – The language has very well defined patterns for large number of operations, hence it is easy to write.
4. **Modularity** – Language supports modular programming via procedural calls and definitions. Parameters can only be passed by values.
5. **Static Scoping** – Variables follow static scoping.

Tokens

1. **Ratio** – Ratio is a relationship between two numbers.
    - Ratio r = 3:4;

2. **Point** – A point in floor plan . Can be specified in 3 ways
    - Point p1 is (x,y)
      Giving x and y coordinates
      Eg Point p1=(3,4);
    - Point p1 divides wall w1 in ratio r1 from its start point
      Point is dividing wall w1 in ratio r1 as seen from w1's starting point.
      Eg Point p1=(w,2:3) // divides wall w in ratio 2:3.
    - Point P1=w1.start/end;
      P1 is start or end point of wall w.
      Eg.P1=w1.start;
    - Point p1=(w1,w1.start/end,d);
      Point is at a distance of d from start/end point of wall w1.
      Eg Point p1=(w1,w1.start/end,d)

3. **Wall** – A Wall in floor plan. Each wall has a notion of start point and end point.
    - Wall W1=t||(p1&p2);
      W1 is a wall which has start point as p1 and end point as p2 with thickness t.
      Eg Wall w1=(0,0) & (0,10);
    - Wall W1=t||(w2,p1,theta1)
      A wall of length d is drawn from another wall w2 at point p1 on w , with an angle of theta1 in anticlockwise direction from w2. Here p1 is start point of w1 and a thickness of t.
      Eg Wall w1=(w2,w2.end,90,14);

4. **Doors/Windows** – a Door or a Window can be specified in a floor plan as
    - Door d1= (p1,p2) //p1 and p2 are points on the wall.
    - Window w1= (p1,p2) // p1 and p2 are points on a single wall.

5. **Floor** – A floor defines a plan for a particular floor for a building.
   A Floor plan is given by
   Floor F1;
   F1{

```
// contains all walls, doors and windows which constitute the particular floor
};
```

6. **Building** – This is the topmost construct which specifies a building. This is also the construct from where program execution starts and is required in program.

```
Building b{
//floor/array of floors specifying the building strucuture.
Floor F[4]; // building of 4 floors including ground floor
F[0]
{
 // floor plan of ground or 0th floor
}
F[1]
{
// plan of 1st floor
}
.
.
.
}// ends building instance
```

7. **Loops** – A ‘for' loop design is allowed
    - for (count=1; count<12;count=count+1) // here count is an integer type variable.
    - for (each <subtype> <variable name>in <subtype array>)

      ```
      {
      }
      ```

      eg . for (each wall w in w1[5])// here w1 is an array of walls.
8. **If construct** – Language supports if else construct , with conditions being checked via composition of relational operators(<,<=,>,>=,==) and identifiers.

   ```
   Eg If(a==b)
   { // Statements
   }
   ```

9. **Integer** - Integer data type is present for variables eg Int c1;
10. **Float** – Float data type is present for variables eg Float f1;

11. **Distance** - Distance is a variable type which is used to specify distance between a point and a point or a point and a line.
   Eg Distance d = p2-p1; // '-' operator computes distance between points p1 and p2.
   Distance d=w1-p1 // d is distance between point and wall.

12. **Function/Procedure** – A procedure can be called with any number of parameters of available data types , and only one return data type
   Eg. Wall func(Wall w1,Point p2 ,…..)
   {
   // Statements
   }// Wall is return type of procedure.

13. **Arrays** – A building may consist of an array of floors. Therfore arrays are allowed for each data types
   Eg. Floor a[6].

14. **Construct(wall/floor/door/all/all-r1,r2….)** - Construct s a wall / floor . Additionally if all walls / floors are to be constructed , construct(all) comes handy.
   Eg. Construct(w1) // Construct walls w1
   Construct(all) // Construct all walls,doors windows if inside floor definition present in floor , else if present in building definition , constructs all floors.
   Construct(all – r,s,…) constructs all objects except r,s

15. **{ }** – Used for a collection of statements.

16. **'='** is the Assignment operator.

17. ';' is statement/collection of statements termination operator.

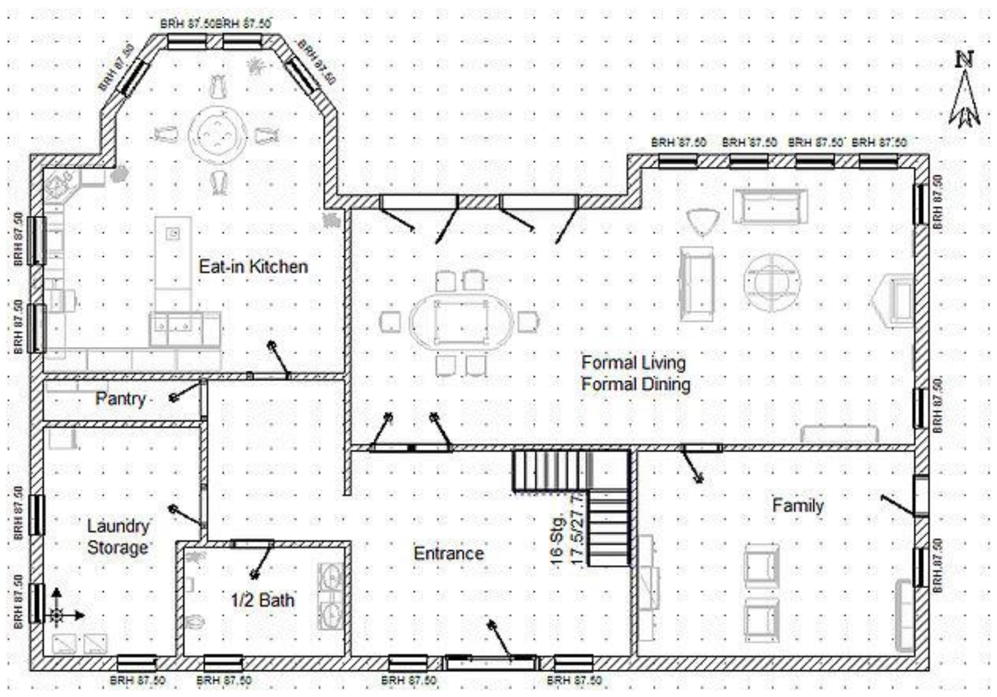
18. '–' operator computes distance when used for distance type variable , else it performs subtraction operation.

19. '*' operator computes simple multiplication of two numbers specified around it.

20. '+' operator computes simple addition of two numbers specified around it. Numbers may be integer or float.

Codes in the language:

1. A code for a building like the one shown below can be easily written using the language. A sample code to draw the building that gives a general idea has been shown below. Points have been initialized in both (x,y) form as well as (point, wall, distance) form. The building block contains the floor block. This building has only one floor, as such only one floor block is present. Most of the walls of the diagram have been drawn, a door and a window have also been drawn using the code to illustrate how the language would work.



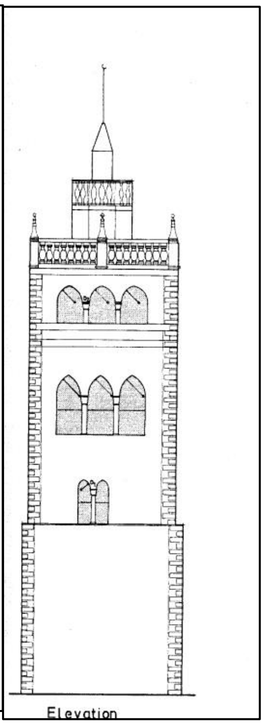
Code for a floor plan, like the one shown above

```
Building b{
	floor f1{
		 Point p1 = (0,0);
		Point p2 = (100,0);
		Wall w1 = 2||(p1 & p2);
		Wall w2 = 2||(w1,p2,90,40);
		 Wall w3 = 2||(w2,w2.end,90,30);
		Wall w4 = 2||(w3,w3.end,90,5);
		Wall w5 = 2||(w4,w4.end,90,30);
		Wall w6 = 2||(w5,w5.end,90,15);
		Wall w7 = 2||(w6,w6.end,45,15);
		Wall w8 = 2||(w7,w7.end,45,15);
		Wall w9 = 2||(w8,w8.end,45,15);
		Wall w10 = 2||(w9,w9.end,45,10);
		Wall w11 = 2||(w10,w10.end,90,4);
		Wall w12 =2||(w11,w11.end,90,40);
		point p3 = (w1.start,w1,17);
		Wall w13 = 1||(w1,p3,90,15);
		Wall w14= 1||(w13,w13.end,90,15);
		Wall w15= 1||(w14,w14.end,90,15);
		point p4 = (w2,w2.start,20);
		Wall w16 = 1||(w2,p4,90,40);
		point p9 = (w3,w3.start,20);
		Wall w17 =1|| (w3,p9,90,25);
		point p10 = (w11,w11.start,20);
		Wall w18 = 1|| (w3,p10,90,20);
		point p11 = (w11,w11.start,25);
		Wall w19 =1|| (w3,p10,90,10);
		point p5 = (w1.start,w1,20);
		point p6 = (w1.start,w1,22);`
		window win1 = (p5,p6);
		point p7 = (w1.start,w1,30);
		point p8 = (w1.start,w1,34);
		door d1 = (p7,p8);
		Construct(all)
	}
	Construct(f1)
}
```

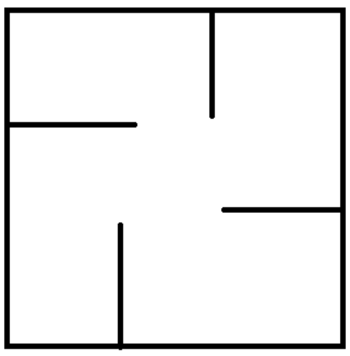
2. A code has been shown below to draw a minar like figure with 5 floors and a repetitive pattern of four Walls. A for loop can be written to make drawing easier. As is apparent from the code how easy it is to draw such buildings:

```
Building b{
point p1 = (0,0);
point p2 = (10,0);
floor f[5];
for(i=0; i<5;i = i + 1){
            f[i]{
                        wall w[4];
                        wall bound[4];
                         w[0] = 1||(p1 & p2);
                        for(j=1; j < 4; j = j+1){
                           w[j] = 1||(w[j-1],w[j-1].end,90,10);
                        }

                        }
                        construct(f[i]);
            }
}
```



3. This code shows how the programmer can write functions to implement some patterns to be drawn in his building. This can be extended tp write much more complicated patterns like the one shown on the right.



```
Wall func(Wall w,Ratio r,float d){
        Point p=(w,w.start,r);
        return 2||(w,p,90,d);
}

Building b{
        Floor f;
        f {
                Point p1 = (0,0);
                Point p2 = (25,0);
                Wall bound[4];
                Wall bound[0]= 2||(p1 & p2);
                Wall bound[1]= 2||(bound[0],bound[0].end,90,25);
                Wall bound[2]= 2||(bound[1],bound[1].end,90,25);
                Wall bound[3]= 2||(bound[2],bound[2].end,90,25);
                Wall w[4];
                for (each wall t in w[4]){
                        w[i] = func(bound[i],1:2,10);
                Construct(all);
                }
        Construct(f);
        }
}
```

4. Another aspect of the programming language could involve calculating area to be painted, or total area of windows or doors, this could prove useful for cost estimates, where rate is per unit area. Similar loops can be written for all doors or windows in the building.

```
//basically for loop through all Walls in the building
float area,h;
for each Wall r in w[4]{
  area = h*length(r.end,r.start);
}
```

5. To solve the problem of floor comparison. We have ? operator

```
Int flag =0;
If(f1 == f2){
        Flag = 1;
}
```

PS: Thank you for your cooperation, patience and all the help given.