Blox

September 28, 2016
Programming Languages and Translators
Stephen A. Edwards

Team Members

Name	UNI	Role
Paul Czopowik	pc2550	Manager
Naeem Bhatti	bnb2115	Language Guru
Tyrone Wilkinson	trw2119	System Architect
Jonathan Voss	jcv2130	Tester

Describe the language that you plan to implement

The Blox language is a language designed to create and connect Lego pieces. It will allow a programmer to build structures out of Lego blocks based on predefined primitive “brick” types that represent basic Lego building blocks. The programmer will be able to define new blocks that are derived from these simple elements and provide instructions on how the blocks are assembled together. Part of the language will also check if certain assemblies are possible. For example, if a block is already joined to another block on one side, another block will not be allowed to be attached to that same side. To facilitate this, the language will be object oriented and will allow the programmer to create “is-a” and “has-a” relationships to objects (inheritance and composition). Besides primitive data types such as int and bool, Blox will provide primitive lego blocks which can then be assembled into more complex structures. For example: a set of 4x4 cubes stacked on top of one another can be made into a pillar. “Construction” of new blocks can be accomplished from pre-existing blocks, which in turn would create a new piece. This is different from joining pieces, which creates a new structure, but are still made of separate bricks. The syntax and design of the language is based on Java. The goal of Blox is to be a simple streamlined language with easy to implement functions for creating and building with the basic brick.

Explain what sorts of programs are meant to be written in your language

Programs written in Blox will create Lego models. For example, the programmer can instantiate a set of bricks, then define functions on how these bricks are put together. New objects can then be defined, such as a house. The program will output a positional data file, which describes where each block belongs in space and which direction it is facing (XML or similar).The intent is to be able to interface this file as input to another utility that can draw the model, producing a 2D or 3D graphical model.

Explain the parts of your language and what they do

The most basic step in Blox is using the “create” keyword to instantiate a brick object. The “join” keyword will allow two pieces to be connected by their complementing sides, or pegs-to-holes. This will also specify how the pieces are attached: top to bottom (stack), offset (staircase), side by side (adjoining), etc. An Edge/Join Validation feature of the language will detect whether the proposed join is possible and reject it otherwise, providing a compile time or runtime error with a description of the problem. The “detach” keyword will allow for the removal of a piece. For example, although a wall may be defined, bricks can still be removed after its construction. This will allow for a simpler method of constructing a wall with a hole, rather than accounting for this during the building of the wall. Easy creation of new pieces will be allowed by copying

parameters from existing pieces (primitives and derived). “Construct” would be used to take separate pieces and join them permanently. For example, if 4 1x1 bricks were assembled side by side, “construct” could then be used to transform them into a single, inseparable 4x1 brick.

Attributes of a Brick

- Size or dimensions
- Position (x,y,z)
- Direction - which way the brick is facing (all bricks are assumed to be right side up)
- Connections - the other pieces that are connected and where they are connected
- Color

Syntax

Primitive Types:

- Int
- Bool
- Char
- Boolean
- Long
- Short

Other Data Types:

- Brick - the fundamental data type of the language
- Array

Operators:

+ - * / = % ++ -- == != > < >=
<= && || !

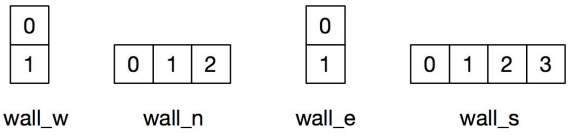
Keywords:

- Create - instantiates an object
- Construct - creates a new type of brick from preexisting ones
- Build - automates a construction process or a series of construction processes
- If, Else
- While, For, Do
- Switch, Case, Default
- Join, Detach

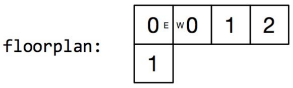
- Void
- Destroy
- Abstract
- Catch
- Try, Throw, Throws, Finally
- True, False
- Null
- Return
- Static
- Package
- Class, Interface, Abstract, Super, Extends, This
- Byte, Boolean, Int, Long, Short, Char

Sample Program:

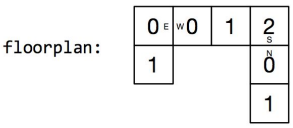
```
Brick make_floorplan()
{
	Create Brick(0,0) floorplan;
	Create Brick(2,1) wall_w;
	Create Brick(3,1) wall_n;
	Create Brick(2,1) wall_e;
	Create Brick(4,1) wall_s;

	2DJoin(floorplan, wall_w, 0E, wall_n, 0W);
	2DJoin(floorplan, wall_n, 2S, wall_e, 0N);
	2DJoin(floorplan, wall_e, 1S, wall_s, 3N);

	return floorplan;
}

void main()
{
	Brick floorplan = make_floorplan();
	Brick frame = Build.layer(floorplan, 2);
	Display(frame);
}
```



2DJoin(floorplan, wall_w, 0E, wall_n, 0W):



2DJoin(floorplan, wall_n, 2S, wall_e, 0N):



2DJoin(floorplan, wall_e, 1S, wall_s, 3N);