FINAL REPORT:
Extended Motion Description Language (eMDL)

Jerin Kurian
Rajesh Banik
Russell Klopfer
Andrew Han

5/13/03
COMS4115
Professor Edwards

Introduction

The Extended Motion Description Language (eMDL) is designed to be an upgrade to the Motion Description Language (MDL). MDL was designed to be a language to write programs that could draw and manipulate shapes. The reason eMDL was developed was to further and expand the functionalities dealing with these programs. The eMDL is simple, object-oriented, robust, and include animation capabilities in order to make the original more powerful and useful.

Simple

In extending MDL, eMDL is clearly more advanced in many ways, but the language is still as simple and easy to understand as the original language. Being more advanced in logical programming, eMDL is similar to popular high level languages: added functionalities will include such things as looping for repeated figures that help in creating frames for animation, and functions to help in creating special shapes. The basic commands and syntax of MDL are currently as follows:

```
Line x1, y1, z1, x2, y2, z2
   -  Draw a line from (x1, y1, z1) to (x1, y2, z2).
Box x, y, z, dx, dy, dz
   -  Draw a box with one corner at the specified point and
      its x, y, and z lengths as specified.
sphere x, y, z, r
   - Draw a sphere with radius r, centered at (x, y, z).
torus x, y, z, R, r
   - Draw a torus with radii r and r, centered at (x, y, z).
move x, y, z [knob]
    -  Make a new transformation matrix, which is the result
       of translating the current transformation matrix as
```

```
        specified.
rotate <axis> n [knob]
    -  Make a new transformation matrix, which is the result
       of rotating the current transformation matrix around
       the specified axis n degrees.
push – Push copy of top matrix onto stack.
Pop  - Throw out top matrix on stack.
```

In compiling the eMDL code, the parser takes care of the hard work in creating these extra features. If one were to use the MDL language to program something capable of the same functions, the programs would be longer and more convoluted than need be. In making these features easier to implement, the language becomes more efficient and powerful in its use.

Another aspect of being simple is the output of the program. After translating the eMDL code into its spatial dimensions, the program outputs the corresponding drawings into a *.gif file for easy viewing.

Object-oriented

Differing from the original MDL, eMDL is an object-oriented programming language. Although not as comprehensive as the big object-oriented languages such as Java, C++, or Smalltalk, eMDL's object orientation makes its programs more powerful than before. This is because object-oriented programming languages enable programmers to create modules that do not need to be changed when a new type of object is introduced. Therefore, object-oriented programs are easier to modify.

The structure of MDL is stack based, so in order to change or modify any shapes, one would have to first pop off the stack, modify the specific object, and then push it back on the stack. On the other hand, because the structure of eMDL is object-oriented based, making modifications to specific objects is much simpler and

more efficient, making eMDL a superior and necessary upgrade especially beneficial in the animation aspect.

Robust

The ability to have a language that comes with a wide range of capabilities and that also does not break down easily is very important. With bigger languages, even experts can make simple mistakes that allow unpredictable problems to arise. In eMDL, the logical syntax with rigid parameters and lack of detailed tidbits helps solve these problems. The compiler will also be able to specifically pinpoint any problems because of the language specifications making debugging a less daunting task. The eMDL code is capable of being parsed by any MDL parser, also adding to the language's robustness.

Animation

MDL capabilities include outputting a single picture image, but eMDL will have animation capabilities. In an animation, the easily modified objects can be altered for a specified number of individual frames in a series that can then be compiled and outputted into an animated gif. This feature definitely sets it apart from simpler animation applications/languages such as Logo or Mathematica, while not as advanced and intensive as Flash.

The eMDL is a useful upgrade because the language is now more powerful and useful, yet easy to learn and easy to use because it is object-oriented and robust. The language also allows simple animation without being too simple or as intensive as other applications. The eMDL will definitely make simple graphics and animations easier for everyone.

1. Language Tutorial

The eMDL language allows for easy creation and manipulation of MDL files and graphics. All eMDL scripts must follow the basic form of the language. This form consists of filename specification, function declarations, variable declarations, and then the main body.

1.1 Filename Specification

Users may want to specify a filename for all MDL files that will be created for their eMDL scripts. To do this, users must include a filename specification as the first line in their script. For example:

```
filename mdlTest;
```

will result in all MDL files being named mdlTest<number>.mdl. The number is enumerated based on how many times the user has called save in the eMDL script. Namely, the first save will be named mdlTest0.mdl, the second save will be mdlTest1.mdl, and so on. By default, the file extension .mdl will be added to each saved file so it is not recommended to put a file extension in the filename specification.

However, filename specification is not necessary. If the filename specification is not included, then a default filename will be used. This will result in all of the save calls in the eMDL script to output tmp<number>.mdl enumerated in the same manner as described above.

1.2 Function Declarations

The next section of and eMDL script are the function declarations. All function declarations must occur at the top a script. It is not possible to create a function after variable declarations have occurred or in the middle of the body.

The eMDL language supports macro style functions with no parameters. Every function must have a unique identifier. If a function is overloaded, the first declaration will be the instance of the function that can be called.

The following is an example of a function that will move a predefined variable sphere1 up the y-axis 50 pixels and a predefined variable box2 down the y-axis 50 pixels.

```
function move50 {
      move sphere1 <0, 50, 0>;
      move box2 <0, -50, 0>;
}
```

The semantics of the predefined function move will be discussed later in this tutorial. The function declaration consists of the keyword function followed by the name of the function. Next, the body of the function is defined encapsulated in curly-braces. The function body can consist of a combination of predefined functions, variable assignments, other user-defined functions, and repeat clauses. Recursive function calls, while not illegal, are not recommended due to the macro nature of eMDL. Most likely, recursive calls will lead to infinite recursion during compilation.

1.3 Variable Declarations

Variable declarations must all occur before the actual body of an eMDL script. Every variable declared in this section will have a global scope. There is only one scope in eMDL, so no variables may be declared with scopes local to another section of code.

There are five variable types in the eMDL language as defined in the Language Reference Manual: integer, box, line, sphere, and torus. Each has its own declaratory statement:

- set *name* as integer <*value*>

  creates an integer with the specified name and value

- set *name* as box <*x, y, z, dx, dy, dz*>

creates a box with the given name starting at point (x, y, z) with lengths of dx, dy, and dz in the x, y, and z directions respectively

- set *name* as line <*x, y, z, x2, y2, z2*>

  creates a line with the given name going from point (x, y, z) to point (x2, y2, z2)

- set *name* as sphere <*x, y, z, r*>

  creates a sphere with the given name at point (x, y, z) with a radius of r

- set *name* as torus <*x, y, z, r, R*>

  creates a torus with the given name centered at point (x, y, z) with an inner radius of r and an outer radius of R.

Each variable must have its own unique identifier. When a variable is declared, it must also be initialized using the parameters explained above. Any value passed to the initializing statement for a variable must be a positive value.

1.4 Main Body

The main body of an eMDL script consists of the actual graphical manipulations. Any variable declared in the previous section can be changed and moved in order to create the picture, effect, or animation that a user wants.

In the body, a user may do any of the following:

- Utilize any of the assignment statements outlined in the LRM
- Call on the predefined functions move, scale, and hide
- Call a user defined function
- Create a repeat statement
- Remove a variable
- Save the current state of the program into an MDL file

1.5 Compilation

To compile an eMDL script, a user must run the eMDL compiler. You can run the compiler with the following command:

```
java eMDL <filename>
```

where <filename> is replaced with the appropriate eMDL file.

The complier will lex, parse, and walk all of the code provided in the file and return any errors and warnings. If there are no warnings or errors, it will create MDL files for every call to the save function in your eMDL script. The MDL files will be named using the conventions discussed above.

Once you have MDL files, you may run them through an MDL parser of your choice to create pictures and animated gifs.

2. Language Reference Manual

2.1 Introduction

The Extended Motion Description Language (eMDL) is very similar to its predecessor MDL. However, there are facets of eMDL not found in MDL. In eMDL the language has been extended to handle the simple integer type as well as permit the program to contain functions and loops. This additional functionality, in tandem with a newly implemented save function, will permit the programmer to create animations, as well as more complicated scenes, much more easily.

2.2 Structure

When writing a program in eMDL, you must structure your program according to these rules: The first line of the program must always have the form:

```
filename name-of-file-containing-program;
```

The next section of the program is the function declaration section. If you wish to define functions you must do it in this section. The rest of the program is the body of the program, which takes on no special form.

2.3 Lexical Conventions

There are five types of tokens: identifiers, keywords, constants, expressions, and separators. White space is ignored unless it serves as a separator.

2.3.1. Comments

Comments are restricted to one line at a time. A commented line begins with the '#' character.

2.3.2. Identifiers

An identifier is a sequence of alphanumeric characters, including the '_' character. Upper and lower case letters are considered different.

2.3.3. Keywords

The following identifiers are reserved as keywords and may not be used as variable names:

```
set         as          save
func        function    repeat
line        box         sphere
torus       int         move
scale       filename
```

2.3.4. Constants

2.3.4.1. Integer Constants

An integer constant is a sequence of digits (0-9).

2.3.4.2. String Constants

A string is a sequence of characters surrounded by double-quotes (“*string*”).

2.4 Syntax notation

In the syntax notation used in this manual, syntactic categories are indicated by *italic* type, and literal words and characters in gothic. An optional terminal or non-terminal symbol is indicated by the subscript “opt.”

2.5 What’s in a Name?

EMDL bases its interpretation of identifiers on the type. There are four types in eMDL:

```
1. line x1, y1, z1, x2, y2, x2 -- Draw a line from (x1, y1, z1) to (x2, y2, z2).
2. box x, y,z, dx, dy, dz -- Draw a box with one corner at the specified point and its x, y, and z lengths.
3. sphere x, y, z, r -- Draw a sphere with radius r, centered at (x,y,z).
```

4. torus x, y, z, outerR, innerR -- Draw a torus with radii outerR and innerR, centered at (x,y,z).

5. int *var-name* – Allots space in memory where an integer can be stored and referenced by its identifier *varname*.

2.6 Objects and lvalues

An object is a data structure consisting of values which will be translated into pixels in specific positions on a drawing board; an lvalue is an expression referring to an object.

2.7 Expressions

The order of execution of operators is found at the end of this document in section 12.

2.7.1. Primary expressions

2.7.1.1. identifier

An identifier is a primary expression when it has been properly declared. An identifier's type is specified in its declaration. For example, set X as box creates the identifier X of type box.

2.7.1.2. constant

A decimal constant is a primary expression.

2.7.1.4. (expression)

A parenthesized expression whose type and value are identical to those of the unadorned expression. Presence of parentheses does not affect whether the expression is an lvalue.

2.7.1.5. func primary-expression

A function call is a primary expression followed by parentheses containing a possibly null, comma-delimited list of expressions, which will serve as the arguments to be sent to the function. Arguments are passed as values.

2.7.1.6. primary-lvalue . member-of-structure

An lvalue expression followed by a dot followed by the name of a member of a structure is an lvalue. The lvalue is assumed to have the same structure as the one containing the referenced member. The result of the expression is an lvalue corresponding to the member of the structure. An example: obj.x would return an lvalue currently containing the x coordinant of the object obj.

2.7.2. Unary operators

Expressions containing unary operators gourp left to right.

2.7.2.1. lvalue-expression ++

The result is the value of the object referred to by the lvalue expression. After the result is noted, the value of the lvalue expression is increased by one.

2.7.2.2. lvalue-expression –

The result is the value of the objected refrenced by the lvalue expression. After the result is noted, the value of the lvalue expression is decreased by one.

2.7.3. Multiplicative operators

The multiplicative operators *, /, and % group left to right. Can be preformed only on decimal constants and identifiers resulting in decimals.

2.7.3.1. expression * expression

The binary * operator indicates multiplication.

2.7.3.2. expression / expression

The binary / operator indicates division.

2.7.3.3. expression % expression

The binary % operator yields the remainder from the division of the first expression by the second.

2.7.4. Additive operators

The additive operators + and – group left to right. Can be preformed only on decimal constants and identifiers resulting in decimals.

2.7.4.1. expression + expression

The result is the sum of the expressions.

2.7.4.2. expression – expression

The results is the difference of the expressions.

2.7.5. Assignment operators

2.7.5.1. lvalue = expression

The value of the expression replaces the value of the lvalue. The operands must both be int.

2.8 Declarations

Declarations have the form:

Set declarator as decl-specifier < parameter-list >

The declarator is the identifier being declared. The decl-specifier is a type-specifier. The parameter list is a list of constant integers; variables are not allowed.

2.8.1. Type specifiers

The type specifiers are:

`box`

`line`

`sphere`

`torus`

`int`

2.8.2. Declarators

A declarator can be any identifier that is not a reserved word.

2.8.3. Meaning of declarators

Each declarator is taken to be an assertion that when a construction of the same from as the declarator appears in an expression, it yields an object of the indicated type. Each declarator contains exactly one identifier, which is declared.

Some sample declarations:

Declaration of an object:

```
set X as box;
```

Declaration of a function:

```
function fata {
      statement
      ...
      ...
}
```

2.9 Statements

Statements are executed in sequence.

2.9.1. Expression statement

Expression statements have the form:

```
expression;
```

2.9.2. Repeat statement

The repeat statement has the following form:

```
repeat < int-expression > { statement(s) };
```

The parenthesized int-expression must result in a value of type int. The int-expression takes on the form:

```
int-expression:
      integer-constant
```

The statements will be executed repeatedly as many times as are indicated by the value of the parenthesized expression. The C equivalent to the repeat statement is as follows:

```
for (int i=0; i<int-expression; i++) { statement(s) }
```

2.9.3 Null statement

The null statement has the form:

```
;
```

2.10 Scope

All identifiers are given global scope. It is an error to redeclare an identifier.

2.11 Constant expressions

Some expressions in eMDL evaluate to constants. These expressions can be connected by the binary operators + - * / %. Parentheses can be used for grouping, but not for function calls.

2.12. Operator priority

eMDL only allows the programmer to execute at most one operation per line, unless the second operator is the assignment operator. This reduces operator priority to a matter of sequence. For example, the operation in C q = (1 + 2) * 3 would be written in eMDL in two lines:

```
q = 1 + 2;
q = q * 3;
```

3. Project Plan

3.1 Timeline and Goals

Our team set out a timeline of projected dates in which to finish aspects of our project.

2-18-03	eMDL whitepaper
3-10-03	eMDL grammar complete, core language features defined
3-27-03	Language reference manual
4-11-03	Lexer and parser complete
4-28-03	Static semantics
5-02-03	Test/Test Suite complete

3.2 Team Roles and Responsibilities

Jerin Kurian	Team leader, architecture, parser/lexer
Rajesh Banik	Architecture, parser/lexer, static semantics
Russell Klopfer	Documentation, grammar, debugging
Andrew Han	Documentation, regression testing, debugging

3.3 Software Development Environment

The project used a C compiler and MDL parser to test and develop the eMDL code. We used ANTLR to create our lexer and parser. Java was used to write the symbol list and compiler for other elements of the project. In viewing the final gif images, we used the gnu image manipulation program (GIMP).

3.4 Project Log

The project log below lists the significant meetings where notable and progressive things were accomplished.

2-14-03	Meeting discussing whitepaper, project specifics
2-17-03	Discuss and write whitepaper, language
2-18-03	Completed eMDL whitepaper
3-07-03	Meeting about language, CFG, LRM
3-10-03	eMDL grammar complete, core language features defined

3-24-03	Meeting, writing LRM
3-25-03	Writing LRM, start lexer/parser
3-27-03	Completed language reference manual
4-01-03	Coding lexer/parser
4-03-03	Coding lexer/parser
4-11-03	Lexer and parser first working version
4-13-03	Testing – syntax, lexer/parser working
4-17-03	Code generation, treewalker
4-24-03	Code generation, static semantics
4-28-03	Static semantics
5-02-03	Regressive testing/debugging
5-05-03	Test/Test Suite complete

4. Architectural Design

The architecture of eMDL is comprised of the traditional elements of a compiler. These are: lexer, parser, tree-walker and error-handler. Each of these components was created using ANTLR, which was used to produce Java files. The interaction of these components is pictured below (fig. 5.1).

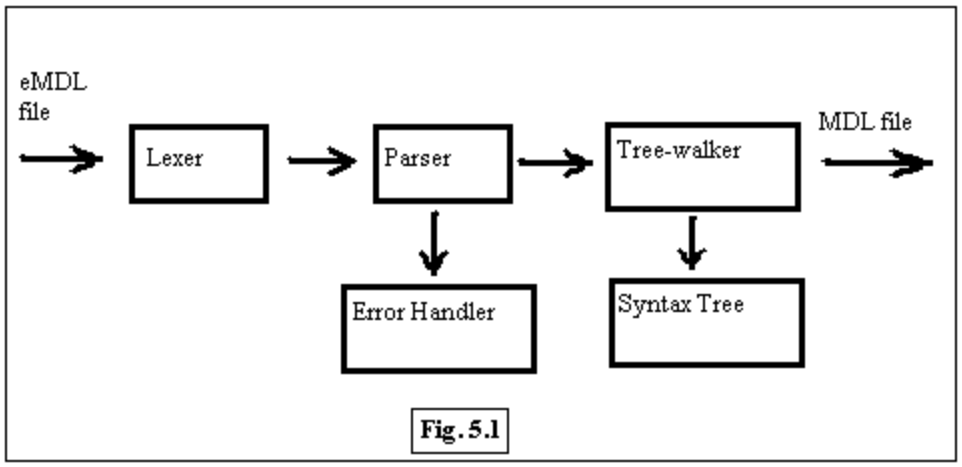

Fig. 5.1

As you can see, the output of this chain is an MDL file. Since eMDL is actually a translator, it must now translate the MDL file into a language that can be translated by a native compiler into machine code. Thus, there are essentially two compilers involved in processing an eMDL file. The second "compiler", used to process the MDL file, is simpler than the eMDL translator as it consists of only a parser written in C that outputs a gif file (fig. 5.2).

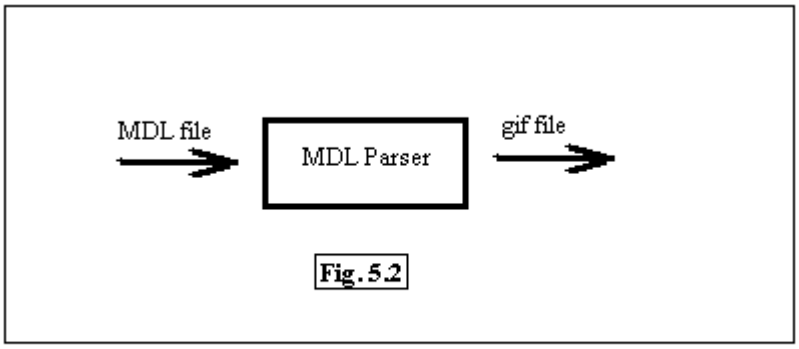

Fig. 5.2

The reason why these are listed as separate segments of the architecture is that each time the programmer calls the save function, a new MDL file is output, which will eventually be converted into a gif file for an animation. Thus, it is necessary to run these programs separately, so that all of the MDL files can be compiled at once.

Errors produced by the parser or tree-walker are sent to the error handler, which will inform the programmer of what error occurred.

5. Testing Plan

5.1 Overview

Our testing consisted of two major test phases. Phase I first tested if the lexer and parser were working properly. Specific variables were inputted into the parser to test if the variables were read correctly. Phase II checked the static semantics by using a test program (show below). This program was a comprehensive test of whether the overall program was functioning: by creating functions, calling functions, calling saves/output, declaring variables, and all the different types of assignment statements.

5.2 Testing Code

5.2.1 Phase I

Much of the testing in Phase I did not implement complete programs; code snippets and random variables were inputted to ensure the proper parsing, tokenization, and static semantic analysis.

5.2.2 Phase II

The program below was part of regressive testing and the comprehensive test of the eMDL syntax:

test.mdl:

```
filename ddd;

function f {

x.y=<2>;

x.y=<2>;

x.y=x.y;
```

```
x.y=2+x.y;

x.y=x.y+2;

x.y++;

x.y--;

move x <1,2,3>;

}

function dd {

x.y=<2>;

}

set x as box <2,2,2,2,2,2>;

set Y as line <2,2,2,2,2,2>;

set Z as sphere <2,2,2,2>;

set A as torus <2,2,2,2,1>;

set B as integer <1>;

x.y=<2>;

x=Y;

x.y=x.y;

x.y=2+x.y;

x.y=x.y+2;

x.y++;

x.y--;

funct f;

save;
```

6. Lessons

Jerin Kurian

I was the one who wrote up most of the TreeWalker and one of the most important lessons I learned was that we had to regressively test our program against what we had tested earlier, because even the smallest change in our walker will create an error or a disambiguity somewhere somehow. We also learned some interesting ways to manipulate the tree structure, and how to save specific parts of the tree structure when we were trying to figure out how to save our function body to be called later on by a function call. What we had to do in this case was to create a copy of our AST at the point that we start defining our function declaration. This way, all we are saving are the function declarations, and when we need to call a function from our main body, we just call a treeWalker function on this new tree, rather than the tree that we had been parsing through, which would be in the body section when a function call is made.

Rajesh Banik

After working on creating eMDL, I have learned a lot about working with teammates, especially about the necessity of communication. When working in a group, it is always important for all of the teammates to understand each other. Without this understanding, it is almost impossible to work on different sections of a project and come out with a cohesive end product. Also, this project has given me a new appreciation for the programming languages that are currently used. After spending

hours upon hours working on this eMDL translator, I can't even begin to imagine how much time and work went into creating a larger language such as Java or C.

Russell Klopfer

By working on eMDL I have learned a lot about what is involved in creating a programming language. First, since I wrote some of the original CFG's for the compiler, I obtained some insights into the abstract theory underlying all languages. Furthermore, being involved in the creation of the compiler itself, I was able to see how much of this theory manifests itself in practice, and how complicated it is to make this transition.

Andrew Han

The project's success is really dependent on organization and preparation. Especially in a class or in the real world with very rigid timelines, it was very important to keep up with our timelines. Luckily, our group was very cohesive and we were able to work well together with each of us fulfilling complementary roles. In terms of documenting, it was very important keep what was going on but at the same time understand how the code and project was always changing, but still driving towards the same destination.

Appendix

emdlTreeWalker.g:

```
{
import java.io.*;
import antlr.CommonAST;
import antlr.DumpASTVisitor;
}
class emdlTreeWalker extends TreeParser;
options
{
    importVocab=emdl;
    buildAST=false;
}
{

/*
@author Jerin Kurian - jk1243@columbia.edu
*/

    //create a globalhashtable to store all variables
    //this is in essence our symbol table
    //can we put objects in our hash table?

    String filename = "tmp";
    int filenamecounter=0;
```

```
//needed for function calls
AST rootHolder = null;

EmdlList list = new EmdlList();

//used for binary operations
int binop(int x, String op, int y){

    if (op.equals("+")){
        return x+y;
    }
    else if (op.equals("-")){
        return x-y;
    }
    else if (op.equals("*")){
        return x*y;
    }
    else if (op.equals("/")){
        return x/y;
    }
    else if (op.equals("%")){
        return x%y;
    }
    else {
        return -1;
    }
```

```
    }

    //used for unary operations
    int unaryop(int x, String op){
        if (op.equals("++")){
            return ++x;
        }
        else if (op.equals("--")){
            return --x;
        }
        else{
            return -1;
        }
    }
 }

startRule: (fileName)? functionBegin variableBegin bodyBegin ;

fileName: (#(FILENAME a:ID){filename=a.getText();});

functionBegin: (#(FUNCTIONDECSTART {rootHolder=astFactory.dupList((AST)_t);}
(functionDeclarations)*));

variableBegin: (#(VARIABLEDECSTART (variableDeclarations)*));

bodyBegin: (#(STARTBODY (body)*));
```

```
type: ("box"|"sphere"|"int"|"torus"|"line");



functionDeclarations: #(FUNCTIONBEGIN a:ID);



functionFind[AST functname, String name]: { AST
tmp=astFactory.dupList(functname); _t = functname;}

#(FUNCTIONBEGIN  a:ID

{

            if (name.equals(a.getText())){

               findfunctionBody(_t);

            }

            else {

                _t=((AST)_t).getNextSibling();

                functionFind(_t, tmp.getNextSibling(), name);

            }

        });



findfunctionBody:

#(FUNCTIONBODY (body)*);



//eatfunctionBody: FUNCTIONBODY;



variableDeclarations:((#(BOX a:ID x1:INT y1:INT z1:INT x2:INT y2:INT z2:INT)

                {

                if (list.ifNotExists(a.getText())){

                 list.insert(new EmdlNode("box",a.getText(),
Integer.parseInt(x1.getText()),

                    Integer.parseInt(y1.getText()),
Integer.parseInt(z1.getText()),
```

```
                    Integer.parseInt(x2.getText()),
Integer.parseInt(y2.getText()),

                    Integer.parseInt(z2.getText())));

                }

                else{

                    System.out.println(a.getText()+" was previously
declared.");

                    System.exit(1);

                }

            })



| (#(LINE b:ID xx1:INT yy1:INT zz1:INT xx2:INT yy2:INT zz2:INT)

                {

                if (list.ifNotExists(b.getText())){

                 list.insert(new EmdlNode("line",b.getText(),
Integer.parseInt(xx1.getText()),

                    Integer.parseInt(yy1.getText()),
Integer.parseInt(zz1.getText()),

                    Integer.parseInt(xx2.getText()),
Integer.parseInt(yy2.getText()),

                    Integer.parseInt(zz2.getText())));

                }

                else{

                    System.out.println(b.getText()+" was previously
declared.");

                    System.exit(1);

                }

            })

| (#(SPHERE c:ID x:INT y:INT z:INT r:INT)

                {
```

```
                if (list.ifNotExists(c.getText())){

                 list.insert(new EmdlNode(c.getText(),
Integer.parseInt(x.getText()),

                    Integer.parseInt(y.getText()),
Integer.parseInt(z.getText()),

                    Integer.parseInt(r.getText())));

                }

                else{

                    System.out.println(c.getText()+" was previously
declared.");

                    System.exit(1);

                }

})

| (#(TORUS d:ID xx:INT yy:INT zz:INT rr:INT bigR:INT)

               {

                if (list.ifNotExists(d.getText())){

                list.insert(new EmdlNode(d.getText(),
Integer.parseInt(xx.getText()),

                        Integer.parseInt(yy.getText()),
Integer.parseInt(zz.getText()),

                    Integer.parseInt(rr.getText()),
Integer.parseInt(bigR.getText())));

                }

                else{

                    System.out.println(d.getText()+" was previously
declared.");

                    System.exit(1);

                }

})

| (#(INTEGER e:ID i:INT)

                {
```

```
                if (list.ifNotExists(e.getText())){

                 list.insert(new EmdlNode(e.getText(),
Integer.parseInt(i.getText())));

                }

                else{

                    System.out.println(e.getText()+" was previously
declared.");

                    System.exit(1);

                }

})

);



body:

//{list.print();}

(assignment|functionCall|unaryop|repeat|move|scale|save|remove|hide)

//{list.print();}

;



remove: #(REMOVE a:ID)

    {

     if (!list.ifNotExists(a.getText())){

        list.remove(a.getText());

     }

     else{

        System.out.println(a.getText()+" was not previously declared.");

        System.exit(1);

     }

};
```

```
hide: #(HIDE a:ID)
    {
            list.hide(a.getText());
};

save: SAVE
{
            list.save(filename+Integer.toString(filenamecounter)+".mdl");
            filenamecounter++;
};

assignment: ((#(OBJEQUALSOBJ a:ID b:ID )
                {
                    //System.out.println("OBJEQUALSOBJ, "+a.getText());
                    EmdlNode node1=list.getNode(a.getText());
                    EmdlNode node2=list.getNode(b.getText());

                    if (node1.type.equals(node2.type)){
                        int check = list.makeEqual(a.getText(), b.getText());
                        if (check==-1){
                            System.exit(1);
                        }
                    }
                    else{
                        System.exit(1);
                    }
```

```
                    //list.printVar(a.getText());
})
 | (#(OBJEQULASINT j:ID k:INT)
                {
                    EmdlNode node;
                    node = list.getNode(j.getText());

                    if (node!=null){
                        if (node.type.equals("integer")){
                            list.remove(j.getText());
                            node.x=Integer.parseInt(k.getText());
                            list.insert(node);
                        }
                        else{System.out.println(j.getText()+" is not an
integer.");}
                    }
                    else{System.out.println(j.getText()+" does not exist.");}
                    //list.printVar(j.getText());
})
 | (#(SUBEQUALSSUB c:ID d:ID e:ID f:ID)
                {
                    EmdlNode node1 =list.getNode(c.getText());
                    EmdlNode node2 =list.getNode(e.getText());
                    int sub1=list.getSub(c.getText(), d.getText());
                    int sub2=list.getSub(e.getText(), f.getText());
                    //System.out.println("sub1: "+sub1);
                    //System.out.println("sub2: "+sub2);
```

```
                    if (node1!=null && node2!=null && sub1!=-1 && sub2!=-1){

                        //list.remove(j.getText());

                        //node1.x=Integer.parseInt(k.getText());

                        int check=list.putSub(c.getText(),d.getText(),sub2);

                        if (check==-1){

                            System.out.println("Error1:
"+c.getText()+"."+d.getText()

                                +" cannot be set to
"+e.getText()+"."+f.getText());

                            System.exit(1);

                        }

                    }

                    else{

                        System.out.println("Error2:
"+c.getText()+"."+d.getText()

                            +" cannot be set to "+e.getText()+"."+f.getText());

                        System.exit(1);

                    }

                    //list.printVar(c.getText());



})

 | (#(SUBEQUALSINT g:ID h:ID i:INT)

                {



                    EmdlNode node1 =list.getNode(g.getText());

                    int sub1=list.getSub(g.getText(), h.getText());

                    if (node1!=null && sub1!=-1){

                        int check=list.putSub(g.getText(),h.getText(),
Integer.parseInt(i.getText()));
```

```
                        if (check==-1){

                            System.out.println("Error1:
"+g.getText()+"."+h.getText()

                                +" cannot be set to
"+Integer.parseInt(i.getText()));

                            System.exit(1);

                        }

                    }

                    else{

                        System.out.println("Error1:
"+g.getText()+"."+h.getText()

                            +" cannot be set to
"+Integer.parseInt(i.getText()));

                        System.exit(1);

                    }

                    //list.printVar(g.getText());



})

 | (#(SUBEQUALSSUBBINOPSUB l:ID m:ID n:ID o:ID p:BINARY_OP q:ID r:ID)

                {

                    EmdlNode node1 =list.getNode(l.getText());

                    EmdlNode node2 =list.getNode(n.getText());

                    EmdlNode node3 =list.getNode(q.getText());

                    int sub1=list.getSub(l.getText(), m.getText());

                    int sub2=list.getSub(n.getText(), o.getText());

                    int sub3=list.getSub(q.getText(), r.getText());



                    if (node1!=null && node2!=null && node3!=null && sub1!=-1
&& sub2!=-1 && sub3!=-1){

                        int result = binop(sub2, p.getText(), sub3);
```

```
                        if (result==-1){
                            System.out.println(n.getText()+"."+o.getText()+"
"+p.getText()
                                +" "+q.getText()+"."+r.getText()+" cannot be
done.");
                            System.exit(1);
                        }
                        int check=list.putSub(l.getText(),n.getText(),result);
                        if (check==-1){

                            System.out.println(l.getText()+"."+m.getText()+"
cannot be set to "+
                                n.getText()+"."+o.getText()+" "+p.getText()
                                +" "+q.getText()+"."+r.getText());
                            System.exit(1);
                        }
                    }
                    else{
                        System.out.println(l.getText()+"."+m.getText()+" cannot
be set to "+
                            n.getText()+"."+o.getText()+" "+p.getText()
                            +" "+q.getText()+"."+r.getText());
                        System.exit(1);
                    }
                    //list.printVar(l.getText());

})
 | (#(SUBEQUALSINTBINOPSUB aa:ID bb:ID cc:INT dd:BINARY_OP ee:ID ff:ID)
                {
```

```
                    EmdlNode node1 =list.getNode(aa.getText());
                    EmdlNode node2 =list.getNode(ee.getText());
                    int sub1=list.getSub(aa.getText(), bb.getText());
                    int sub2=list.getSub(ee.getText(), ff.getText());

                    if (node1!=null && node2!=null && sub1!=-1 && sub2!=-1){
                        int result = binop(Integer.parseInt(cc.getText()),
dd.getText(), sub2);
                        if (result==-1){
                            System.out.println(Integer.parseInt(cc.getText())+"
"+p.getText()
                                +" "+e.getText()+"."+f.getText()+" cannot be
done.");
                            System.exit(1);
                        }
                        int
check=list.putSub(aa.getText(),bb.getText(),result);
                        if (check==-1){

                            System.out.println(aa.getText()+"."+bb.getText()+"
cannot be set to "+
                                Integer.parseInt(cc.getText())+" "+dd.getText()
                                +" "+ee.getText()+"."+ff.getText());
                            System.exit(1);
                        }
                    }
                    else{
```

```
                        System.out.println(aa.getText()+"."+bb.getText()+"
cannot be set to "+

                            Integer.parseInt(cc.getText())+" "+dd.getText()

                            +" "+ee.getText()+"."+ff.getText());

                        System.exit(1);

                    }

                    //list.printVar(aa.getText());




})

 | (#(SUBEQUALSSUBBINOPINT ll:ID mm:ID nn:ID oo:ID pp:BINARY_OP qq:INT)

                {

                    EmdlNode node1 =list.getNode(ll.getText());

                    EmdlNode node2 =list.getNode(nn.getText());

                    int sub1=list.getSub(ll.getText(), mm.getText());

                    int sub2=list.getSub(nn.getText(), oo.getText());



                    if (node1!=null && node2!=null && sub1!=-1 && sub2!=-1){

                        int result = binop(sub2, pp.getText(),
Integer.parseInt(qq.getText()));

                        if (result==-1){

                            System.out.println(nn.getText()+"."+oo.getText()+"
"+pp.getText()

                                +" "+Integer.parseInt(qq.getText())+" cannot be
done.");

                            System.exit(1);

                        }

                        int
check=list.putSub(ll.getText(),mm.getText(),result);

                        if (check==-1){
```

```
                            System.out.println(ll.getText()+"."+mm.getText()+"
cannot be set to "+
                                nn.getText()+"."+oo.getText()+" "+pp.getText()
                                +" "+Integer.parseInt(qq.getText()));
                            System.exit(1);
                        }
                    }
                    else{
                        System.out.println(ll.getText()+"."+mm.getText()+"
cannot be set to "+
                            nn.getText()+"."+oo.getText()+" "+pp.getText()
                            +" "+Integer.parseInt(qq.getText()));
                        System.exit(1);
                    }

                    //list.printVar(ll.getText());



})
 | (#(SUBEQUALSINTBINOPINT rr:ID ss:ID tt:INT uu:BINARY_OP vv:INT)
               {

                    EmdlNode node1 =list.getNode(ll.getText());
                    int sub1=list.getSub(ll.getText(), mm.getText());

                    if (node1!=null && sub1!=-1 ){
                        int result = binop(Integer.parseInt(tt.getText()),
uu.getText(), Integer.parseInt(vv.getText()));
```

```
                        if (result==-1){

                            System.out.println(Integer.parseInt(tt.getText())+"
"+pp.getText()

                                +" "+Integer.parseInt(uu.getText())+" cannot be
done.");

                            System.exit(1);

                        }

                        int
check=list.putSub(aa.getText(),bb.getText(),result);

                        if (check==-1){

                        System.out.println(rr.getText()+"."+ss.getText()+"
cannot be set to "+

                            Integer.parseInt(tt.getText())+" "+pp.getText()

                            +" "+Integer.parseInt(uu.getText()));

                        System.exit(1);

                        }

                    }

                    else{

                        System.out.println(rr.getText()+"."+ss.getText()+"
cannot be set to "+

                            Integer.parseInt(tt.getText())+" "+pp.getText()

                            +" "+Integer.parseInt(uu.getText()));

                        System.exit(1);

                    }

                    //list.printVar(rr.getText());

})

 | (#(SUBEQUALSOBJ a1:ID a2:ID a3:ID)

               {



                    EmdlNode node1 =list.getNode(a1.getText());
```

```
                    int sub1=list.getSub(a1.getText(), a2.getText());
                    EmdlNode node2 =list.getNode(a3.getText());
                    if (node2!=null && !node2.type.equals("integer")) {
                        System.exit(1);
                    }
                    int sub2=list.getSub(a3.getText(), "x");




                    if (node1!=null && sub1!=-1 ){


                       int check=list.putSub(a1.getText(),a2.getText(),sub2);
                       if (check==-1){
                            System.exit(1);
                       }
                    }
                    else{
                        System.exit(1);
                    }
                    //list.printVar(a1.getText());



})
 | (#(OBJEQUALSSUB b1:ID b2:ID b3:ID)
                {

                    EmdlNode node1 =list.getNode(b1.getText());
```

```
                    EmdlNode node2 =list.getNode(b2.getText());
                    if (node1!=null && !node1.type.equals("integer")) {
                        System.exit(1);
                    }

                    int sub1=list.getSub(b1.getText(), "x");

                    int sub2=list.getSub(b2.getText(), b3.getText());

                    if (node2!=null && sub2!=-1 ){

                       int check=list.putSub(b1.getText(),"x",sub2);
                       if (check==-1){
                            System.exit(1);
                       }
                    }
                    else{
                        System.exit(1);
                    }
                    //list.printVar(b1.getText());
})

 | (#(OBJEQUALSSUBBINOPSUB c1:ID c2:ID c3:ID c4:BINARY_OP c5:ID c6:ID)
                {

                    EmdlNode node1 =list.getNode(c1.getText());
```

```
EmdlNode node2 =list.getNode(c2.getText());
EmdlNode node3 =list.getNode(c5.getText());

if (node1!=null && !node1.type.equals("integer")) {
    System.exit(1);
}

int sub1=list.getSub(c1.getText(), "x");

int sub2=list.getSub(c2.getText(), c3.getText());
int sub3=list.getSub(c5.getText(), c6.getText());

if (node2!=null && node3!=null && sub2!=-1 && sub3!=-1){

    int result=binop(sub2,c4.getText(),sub3);
    if (result==-1){
        System.exit(1);
    }

    int check=list.putSub(c1.getText(),"x",result);
    if (check==-1){
        System.exit(1);
   }
}
else{
    System.exit(1);
```

```
                    }
                    //list.printVar(c1.getText());
})

 | (#(OBJEQUALSSUBBINOPINT d1:ID d2:ID d3:ID d4:BINARY_OP d5:INT)
                {

                    EmdlNode node1 =list.getNode(d1.getText());

                    EmdlNode node2 =list.getNode(d2.getText());

                    if (node1!=null && !node1.type.equals("integer")) {
                        System.exit(1);
                    }

                    int sub1=list.getSub(d1.getText(), "x");

                    int sub2=list.getSub(d2.getText(), d3.getText());

                    if (node2!=null && sub2!=-1){

                        int
result=binop(sub2,d4.getText(),Integer.parseInt(d5.getText()));
                        if (result==-1){
                            System.exit(1);
                        }
```

```
                    int check=list.putSub(d1.getText(),"x",result);
                    if (check==-1){
                        System.exit(1);
                    }
                }
                else{
                    System.exit(1);
                }
                //list.printVar(d1.getText());


})

 | (#(OBJEQUALSINTBINOPSUB e1:ID e2:INT e4:BINARY_OP e5:ID e6:ID)
            {

                EmdlNode node1 =list.getNode(e1.getText());

                EmdlNode node3 =list.getNode(e5.getText());

                if (node1!=null && !node1.type.equals("integer")) {
                    System.exit(1);
                }

                int sub1=list.getSub(e1.getText(), "x");

                int sub3=list.getSub(e5.getText(), e6.getText());
```

```
                    if (node3!=null && sub3!=-1){


                        int
result=binop(Integer.parseInt(e2.getText()),e4.getText(),sub3);
                        if (result==-1){
                            System.exit(1);
                        }

                        int check=list.putSub(e1.getText(),"x",result);
                        if (check==-1){
                            System.exit(1);
                       }
                    }
                    else{
                        System.exit(1);
                    }
                    //list.printVar(e1.getText());


})


 | (#(OBJEQUALSINTBINOPINT f1:ID f2:INT f4:BINARY_OP f5:INT)
                {

                    EmdlNode node1 =list.getNode(f1.getText());

                    if (node1!=null && !node1.type.equals("integer")) {
```

```
                        System.exit(1);

                    }



                    int sub1=list.getSub(c1.getText(), "x");




                    int
result=binop(Integer.parseInt(f2.getText()),f4.getText(),Integer.parseInt(f5.ge
tText()));

                    if (result==-1){

                        System.exit(1);

                    }



                    int check=list.putSub(c1.getText(),"x",result);

                    if (check==-1){

                        System.exit(1);

                    }



                    //list.printVar(c1.getText());



})



)

;



//THINGS TO ADD: SUBEQUALSOBJ, OBJEQUALSOBJ==GOOD, OBJEQUALSSUB,
OBJEQUALSSUBBINOPSUB, OBJEQUALSINTBINOPSUB

//OBJEQUALSSUBBINOPINT, OBJEQUALSINTBINOPINT
```

```
repeat: #(REPEATTEXT a:INT (repeatbody)*);



repeatbody: #(REPEATBODY (body)*);



//move object <x,y,z>

move: #(MOVETEXT a:ID b:INT c:INT d:INT)

        {
list.move(a.getText(),Integer.parseInt(b.getText()),Integer.parseInt(c.getText(
)),

                Integer.parseInt(d.getText()));

};



// scale object <x,y,z>

scale: #(SCALETEXT a:ID b:INT c:INT d:INT)

        {




list.scale(a.getText(),Integer.parseInt(b.getText()),Integer.parseInt(c.getText
()),

                Integer.parseInt(d.getText()));

};



unaryop: ((#(UNARYOPOBJ a:ID DOT b:ID c:UNARY_OP)

        {




                    EmdlNode node1 =list.getNode(a.getText());

                    int sub1=list.getSub(a.getText(), b.getText());
```

```
                    if (node1!=null && sub1!=-1 ){
                        int result = unaryop(sub1, c.getText() );
                        if (result==-1)
                        {
                            System.out.println(a.getText()+"."+b.getText()+"
"+c.getText()
                                +" cannot be done.");
                            System.exit(1);
                        }
                        int check=list.putSub(a.getText(),b.getText(),result);
                        if (check==-1){
                            System.out.println(a.getText()+"."+b.getText()+"
"+c.getText()
                                +" cannot be done.");
                            System.exit(1);
                        }
                    }
                    else{
                        System.out.println(a.getText()+"."+b.getText()+"
"+c.getText()
                            +" cannot be done.");
                        System.exit(1);
                    }
                   //list.printVar(a.getText());

})
|(#(UNARYOPINT d:ID e:UNARY_OP )
        {
```

```
EmdlNode node1 =list.getNode(d.getText());
if (!node1.type.equals("integer")) {
    System.out.println(d.getText()+e.getText()
        +" cannot be done.");
    System.exit(1);
}
int sub1=list.getSub(d.getText(), "x");

if (node1!=null && sub1!=-1 ){

    int result = unaryop(sub1, e.getText() );
    if (result==-1)
    {
        System.out.println(d.getText()+e.getText()
            +" cannot be done.");
        System.exit(1);
    }
    int check=list.putSub(d.getText(),"x",result);
    if (check==-1){
        System.out.println(d.getText()+e.getText()
            +" cannot be done.");
        System.exit(1);
    }
}
else{
    System.out.println(d.getText()+e.getText()
```

```
                            +" cannot be done.");
                        System.exit(1);
                    }
                    //list.printVar(d.getText());
})))
;

functionCall: #(FUNCTIONCALL a:ID)
{
    //emdlTreeWalker walker = new emdlTreeWalker();
    //doTreeAction("tree.txt",astFactory.dupTree(rootHolder),
    this.functionFind(_t, astFactory.dupList(rootHolder),a.getText());

};
```

emdlnode.java:

```
public class EmdlNode{

    /*
      @author Rajesh Banik - rb861@columbia.edu
     */

    // Constructor for int
    EmdlNode( String name, int x )
    {
```

```
       this.type = "integer";

       this.x = x;

       this.name = name;

    }



    // Constructor for line and box.  differentiate by extra varible type

    EmdlNode( String type, String name, int x, int y, int z, int x2, int y2,
int z2 )

    {

       this.type = type;

       this.name = name;

       this.x = x;

       this.y = y;

       this.z = z;

       this.x2 = x2;

       this.y2 = y2;

       this.z2 = z2;

    }



    // Constructor for sphere

    EmdlNode( String name, int x, int y, int z, int r )

    {

       this.type = "sphere";

       this.name = name;

       this.x = x;

       this.y = y;

       this.z = z;
```

```
   this.r = r;
}

// Constructor for torus
EmdlNode( String name, int x, int y, int z, int r, int R )
{
   this.type = "torus";
   this.name = name;
   this.x = x;
   this.y = y;
   this.z = z;
   this.r = r;
   this.R = R;
}

// elements of this Node
String name;

/* types will a String with the following values :
   integer
   line
   box
   sphere
   torus
*/
String type;
int x = -1, y = -1, z = -1, x2 =-1, y2=-1, z2=-1, r=-1, R=-1;
```

```
    // pointer to next Node
    EmdlNode next;

    //boolean to show if this object is hidden
    boolean hidden = false;

}
```

emdllistitr.java:

```
public class EmdlListItr
{
    /*
      @author Rajesh Banik - rb861@columbia.edu
     */

    /**
     * Construct the list iterator
     * @param theNode any node in the linked list.
     */
    EmdlListItr( EmdlNode theNode )
    {
       current = theNode;
    }

    /**
```

```
     * Test if the current position is past the end of the list.
     * @return true if the current position is null.
     */
    public boolean isPastEnd( )
    {
       return current == null;
    }

    /**
     * Advance the current position to the next node in the list.
     * If the current position is null, then do nothing.
     */
    public void advance( )
    {
       if( !isPastEnd( ) )
           current = current.next;
    }

    EmdlNode current;    // Current position
}
```

emdllist.java:

```
import java.io.*;

public class EmdlList
{
```

```
/*
  @author Rajesh Banik - rb861@columbia.edu
*/

/**
 * Construct the list
 */
public EmdlList( )
{
   header = null;
}

/**
 * Test if the list is logically empty.
 * @return true if empty, false otherwise.
 */
public boolean isEmpty( )
{
   return header == null;
}

/**
 * Make the list logically empty.
 */
public void makeEmpty( )
{
   header = null;
```

```
}



/**
 * Return an iterator representing the header node.
 */
public EmdlListItr head( )
{
   return new EmdlListItr( header );
}


/**
 * Insert after p.
 * @param x the item to insert.
 * @param p the position prior to the newly inserted item.
 */
public void insert( EmdlNode node )
{
   if (node != null){
       node.next = header;
       header = node;
   }
}


/**
 * Find a node in the list that corresponds to a varible name
 * and return an iterator pointing to its node
```

```
 */
public EmdlListItr find( String x )
{
   EmdlNode itr = header;

   while( itr != null && !itr.name.equals( x ) )
       itr = itr.next;

   return new EmdlListItr( itr );
}



/**
 * Given a varible name, remove it from the linked list.
 * This is one of the functions that users might want to use
 * to get rid of a variable from their pic.
 */
public void remove( String x )
{

   if (header.name.equals(x)){
       header = header.next;
       return;
   }

   EmdlListItr p = findPrevious( x );
```

```
        if( p.current.next != null )

            p.current.next = p.current.next.next;  // Bypass deleted node

    }




    /**

     * Finds a node that is previous to the given varible name x.

     * This is helpful when deleting a node from the list.

     */

    public EmdlListItr findPrevious( String x )

    {

       EmdlNode itr = this.header;



       //for( ; !itr.isPastEnd( ) && !itr.current.next.name.equals(x);
itr.advance( ) ){

       //

       //}



       while( itr.next != null && !itr.next.name.equals( x ) )

         itr = itr.next;



       return new EmdlListItr(itr);

    }



    /**

     * This is the main lookup function of the variable linked list

     * it will be used in the variable declaration portion of the code.
```

```
 * It tells the tree walker if the given variable name is already in use.
 */
public boolean ifNotExists  ( String x){
   if( this.isEmpty( ) )
       {
         return true;
       }
   else
        {
            EmdlListItr itr = this.head( );
            for( ; !itr.isPastEnd( ); itr.advance( ) ){
                if (x.equals(itr.current.name)){
                return false;
             }
         }
        }
   return true;
}

/**
 * returns a node for the given varible name.
 * is helpful during the assignments section of the code.
 */
public EmdlNode getNode (String x ){
   if( this.isEmpty( ) )
       {
         return null;
```

```
        }
    else
          {
              EmdlListItr itr = this.head( );
              for( ; !itr.isPastEnd( ); itr.advance( ) ){
                  if (x.equals(itr.current.name)){
                  return itr.current;
               }
          }
          }
    return null;

}

/**
 * Returns an int corresponding to the requested variable name
 * Used for assignments involving sub ops.
 */
public int getSub (String x, String sub){
    if( this.isEmpty( ) )
        {
          //System.out.println("I'm empty;");
          return -1;
        }
    else
          {
              EmdlListItr itr = this.head( );
```

```
                for( ; !itr.isPastEnd( ); itr.advance( ) ){

                    //System.out.println("searching " + itr.current.name + "
for " + x + "." + sub);

                 if (x.equals(itr.current.name)){

                    if (sub.equals("x")){

                        return itr.current.x;

                    } else if (sub.equals("y")){

                        return itr.current.y;

                    } else if (sub.equals("z")){

                        return itr.current.z;

                    } else if (sub.equals("x2")){

                        return itr.current.x2;

                    } else if (sub.equals("y2")){

                        return itr.current.y2;

                    } else if (sub.equals("z2")){

                        return itr.current.z2;

                    } else if (sub.equals("r")){

                        return itr.current.r;

                    } else if (sub.equals("R")){

                        return itr.current.R;

                    } else {

                        return -1;

                    }

                 }

              }

             }

       return -1;
```

```
}

/**
 * Given two varible names, set the values to be the same.
 * Sets a to the values of b.
 */
public int makeEqual(String a, String b){
   EmdlNode nodeB = getNode(b);

   if (nodeB == null){
       return -1;
   } else {
       EmdlListItr itr = this.head( );
       for( ; !itr.isPastEnd( ); itr.advance( ) ){
          if (itr.current.name.equals(a)){
              itr.current.x = nodeB.x;
              itr.current.y = nodeB.y;
              itr.current.z = nodeB.z;
              itr.current.x2 = nodeB.x2;
              itr.current.y2 = nodeB.y2;
              itr.current.z2 = nodeB.z2;
              itr.current.r = nodeB.r;
              itr.current.R = nodeB.R;
              itr.current.hidden = nodeB.hidden;
              return 0;
          }
```

```
        }
    }
    return -1;
}

/**
 * Given a varible name, makes the varible hidden.
 */
public void hide ( String x ){
    if( this.isEmpty( ) )
        {
          return;
        }
    else
         {
             EmdlListItr itr = this.head( );
             for( ; !itr.isPastEnd( ); itr.advance( ) )
              if (itr.current.name.equals(x)){
                 if (itr.current.hidden==true)
                     itr.current.hidden=false;
                 else
                     itr.current.hidden=true;
                 //itr.current.hidden = !itr.current.hidden;
              }
         }

    return;
```

```
}

/**
 * The walker will tell us which variable and sub variable to set
 * and then we find the variable in the list and set its sub
 * variable equal to the value that was given to us
 */
public int putSub (String x, String sub, int newSub){
   if( this.isEmpty( ) )
       {
          return -1;
       }
   else
        {
            EmdlListItr itr = this.head( );
            for( ; !itr.isPastEnd( ); itr.advance( ) ){
                if (x.equals(itr.current.name)){
                if (sub.equals("x")){
                    itr.current.x = newSub;
                    return 0;
                } else if (sub.equals("y")){
                    itr.current.y = newSub;
                    return 0;
                } else if (sub.equals("z")){
                    itr.current.z = newSub;
                    return 0;
                } else if (sub.equals("x2")){
```

```
                    itr.current.x2 = newSub;
                    return 0;
                } else if (sub.equals("y2")){
                    itr.current.y2 = newSub;
                    return 0;
                } else if (sub.equals("z2")){
                    itr.current.z2 = newSub;
                    return 0;
                } else if (sub.equals("r")){
                    itr.current.r = newSub;
                    return 0;
                } else if (sub.equals("R")){
                    itr.current.R = newSub;
                    return 0;
                } else {
                    return -1;
                }
            }
         }
        }
   return -1;

}

/**
 * Prints out the linked variable list.
 * Used for debugging purposes only.
```

```
     */

    public void print( )

    {

       if( this.isEmpty( ) )

           {

              //System.out.println("printing is empty.");

              return;

           }

       else

            {

                EmdlListItr itr = this.head( );

                for( ; !itr.isPastEnd( ); itr.advance( ) )

                 if (!itr.current.hidden){

                    if (itr.current.type.equals("line")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.x2 + ", " + itr.current.y2 + ", " + itr.current.z2);

                    } else if (itr.current.type.equals("box")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.x2 + ", " + itr.current.y2 + ", " + itr.current.z2);

                    } else if (itr.current.type.equals("sphere")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.r);

                    } else if (itr.current.type.equals("torus")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.R + ", " + itr.current.r);

                    } else if (itr.current.type.equals("integer")){

                        System.out.println( itr.current.type + " " +
itr.current.x);
```

```
                    }

                }

            }


    }


    /**
     * Prints out the values of the requested variable.
     * Will only print info for ONE variable
     * Please use only for debugging.
     */
    public void printVar(String x )
    {
       if( this.isEmpty( ) )
           {
             //System.out.println("printing is empty.");
             return;
           }
       else
            {
                EmdlListItr itr = this.head( );
                for( ; !itr.isPastEnd( ); itr.advance( ) ){
                 if (itr.current.name.equals(x)){
                    System.out.println( itr.current.name );
                    if (itr.current.type.equals("line")){
                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.x2 + ", " + itr.current.y2 + ", " + itr.current.z2);
```

```
                    } else if (itr.current.type.equals("box")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.x2 + ", " + itr.current.y2 + ", " + itr.current.z2);

                    } else if (itr.current.type.equals("sphere")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.r);

                    } else if (itr.current.type.equals("torus")){

                        System.out.println( itr.current.type + " " +
itr.current.x + ", " + itr.current.y + ", " + itr.current.z + ", " +
itr.current.R + ", " + itr.current.r);

                    } else if (itr.current.type.equals("integer")){

                        System.out.println( itr.current.type + " " +
itr.current.x);

                    }

                }

            }



        }

    }




    /**

     * Apply a move to one object.

     */

    public void move (String var, int x, int y, int z){

       if( this.isEmpty( ) )

           {

              return;

           }
```

```
   else
        {
            EmdlListItr itr = this.head( );
            for( ; !itr.isPastEnd( ); itr.advance( ) ){
             if (itr.current.name.equals(var)){
                itr.current.x += x;
                itr.current.y += y;
                itr.current.z += z;
                if (itr.current.type.equals("line")){
                    itr.current.x2 += x;
                    itr.current.y2 += y;
                    itr.current.z2 += z;
                }
             }
         }
       }

   return;
}

/**
 * Scale an object.
 */
public void scale (String var, int x, int y, int z){
   if( this.isEmpty( ) )
```

```
            {

               return;

            }

        else

             {

                EmdlListItr itr = this.head( );

                for( ; !itr.isPastEnd( ); itr.advance( ) ){

                  if (itr.current.name.equals(var)){

                     if (itr.current.type.equals("box") ||
itr.current.type.equals("line")){

                         itr.current.x2 *= x;

                         itr.current.y2 *= y;

                         itr.current.z2 *= z;

                     } else if (itr.current.type.equals("sphere")){

                         itr.current.r *= (x+y+z)/3;

                     } else if (itr.current.type.equals("int")){

                         itr.current.x *= x;

                     } else if (itr.current.type.equals("torus")){

                         itr.current.r *= (x+y+z)/3;

                         itr.current.R *= (x+y+z)/3;

                     }

                  }

               }



            }



        return;
```

```
    }


    /**
     * Saves the variable linked list into an mdl file with the
     * given filename.
     */
    public void save(String filename)
    {
       try {


            PrintWriter pr = new PrintWriter(new FileOutputStream(filename));
            pr.println("rotate x 0");


            if( this.isEmpty( ) )
               {
               }
            else
               {
                    EmdlListItr itr = this.head( );
                    for( ; !itr.isPastEnd( ); itr.advance( ) )
                       if (itr.current.hidden==false){
                           if (itr.current.type.equals("line")){
                              pr.println( itr.current.type + " " + itr.current.x +
" " + itr.current.y + " " + itr.current.z + " " + itr.current.x2 + " " +
itr.current.y2 + " " + itr.current.z2);
                           } else if (itr.current.type.equals("box")){
                              pr.println( itr.current.type + " " + itr.current.x +
" " + itr.current.y + " " + itr.current.z + " " + itr.current.x2 + " " +
itr.current.y2 + " " + itr.current.z2);
```

```
                        } else if (itr.current.type.equals("sphere")){

                           pr.println( itr.current.type + " " + itr.current.x +
" " + itr.current.y + " " + itr.current.z + " " + itr.current.r);

                        } else if (itr.current.type.equals("torus")){

                           pr.println( itr.current.type + " " + itr.current.x +
" " + itr.current.y + " " + itr.current.z + " " + itr.current.R + " " +
itr.current.r);

                        }

                    }

             }



           pr.close( );

       } catch (IOException iox){

           System.out.println("Error writing file: " + filename);

           System.exit(1);

       }



    }




    // member variable to store the head of the list

    private EmdlNode header;



}



emdlLexer.g:



class P extends Parser;
```

```
options{
    k=9;
    buildAST = true;
    exportVocab=emdl;
}

tokens {
PARAMETERS;
FUNCTIONBEGIN="function";
TOPLEVELBODY;
REPEATTEXT="repeat";
ROTATETEXT="rotate";
MOVETEXT="move";
SCALETEXT="scale";
STARTRULE;
FUNCTIONDECLARATIONS;
VARIABLEDECLARATIONS;
FUNCTION;
FUNCTIONBODY;
VARIABLE;
BODY;
FUNCTIONCALL;
OBJEQUALSOBJ;
SUBEQUALSSUB;
OBJEQUALSINT;
SUBEQUALSINT;
SUBEQUALSSUBBINOPSUB;
```

```
SUBEQUALSSUBBINOPINT;

SUBEQUALSINTBINOPSUB;

SUBEQUALSINTBINOPINT;

STARTBODY;

FUNCTIONDECSTART;

VARIABLEDECSTART;

SAVE="save";

REPEATBODY;

FUNCT="funct";

SPHERE="sphere";

LINE="line";

BOX="box";

TORUS="torus";

INTEGER="integer";

REMOVE="remove";

FILENAME="filename";

UNARYOPOBJ;

UNARYOPINT;

SUBEQUALSOBJ;

OBJEQUALSSUB;

OBJEQUALSSUBBINOPSUB;

OBJEQUALSINTBINOPSUB;

OBJEQUALSSUBBINOPINT;

OBJEQUALSINTBINOPINT;

HIDE="hide";

}
```

```
{

/*

@author Jerin Kurian - jk1243@columbia.edu

@author Rajesh Banik - rb861@columbia.edu

*/



    public void reportError(RecognitionException ex) {

        try {

            System.err.println("ANTLR Parsing Error: "+ex + " token name:" +
tokenNames[LA(1)]);

            //ex.printStackTrace(System.err);

            System.exit(1);

        }

        catch (TokenStreamException e) {

            System.err.println("ANTLR Parsing Error: "+ex);

            //ex.printStackTrace(System.err);

            System.exit(1);

        }

    }

    public void reportError(String s) {

        System.err.println("ANTLR Parsing Error from String: " + s);

        System.exit(1);

    }

    public void reportWarning(String s) {

        System.err.println("ANTLR Parsing Warning from String: " + s);

        System.exit(1);

    }
```

```
}

startRule: (fileName)? start1 start2 start3;

fileName: FILENAME^ a:ID SEMICOLON!;

start1: (fd:functionDeclarations )*
{#start1 = #(#[FUNCTIONDECSTART,"FUNCTIONDECSTART"], #start1);};
start2: (vd:variableDeclarations)*
{#start2 = #(#[VARIABLEDECSTART,"VARIABLEDECSTART"], #start2);};
start3: (bd:body)*
{#start3 = #(#[STARTBODY,"STARTBODY"], #start3);};


type: ("box"|"sphere"|"int"|"torus"|"line");

functionDeclarations: FUNCTIONBEGIN^ a:ID functionBody;

functionBody: OPENBRACE! (bd:body)* CLOSEBRACE!
{#functionBody = #(#[FUNCTIONBODY,"FUNCTIONBODY"], #functionBody);};

//variable declaration will look like this:
// set x as box <2,2,2,10,10,10>

variableDeclarations: (
```

```
("set"! a:ID "as"! BOX^ LT! x1:INT COMMA! y1:INT COMMA! z1:INT COMMA! x2:INT
COMMA! y2:INT COMMA! z2:INT GT! SEMICOLON!)

| ("set"! b:ID "as"! LINE^ LT! xx1:INT COMMA! yy1:INT COMMA! zz1:INT COMMA!
xx2:INT COMMA! yy2:INT COMMA! zz2:INT GT! SEMICOLON!)

| ("set"! c:ID "as"! SPHERE^ LT! x:INT COMMA! y:INT COMMA! z:INT COMMA! r:INT
GT! SEMICOLON!)

| ("set"! d:ID "as"! TORUS^ LT! xx:INT COMMA! yy:INT COMMA! zz:INT COMMA!
rr:INT COMMA! bigR:INT GT! SEMICOLON!)

| ("set"! e:ID "as"! INTEGER^ LT! i:INT GT! SEMICOLON!)

);



// We currently don't have if conditionals in the body

body: (assignment|functionCall|unaryop|repeat|move|scale|save|remove|hide)
SEMICOLON!;



save: SAVE^;



remove: (REMOVE^ a:ID);



hide: (HIDE^ a:ID);



assignment: (((a:ID EQUALS! b:ID )

                {#assignment = #(#[OBJEQUALSOBJ,"OBJEQUALSOBJ"],
#assignment);})

 | ((j:ID EQUALS! LT! k:INT GT!)

                {#assignment = #(#[OBJEQUALSINT,"OBJEQUALSINT"],
#assignment);})

 | ((c:ID DOT! d:ID) EQUALS! (e:ID DOT! f:ID)

                {#assignment = #(#[SUBEQUALSSUB,"SUBEQUALSSUB"],
#assignment);})

 | ((g:ID DOT! h:ID EQUALS! LT! i:INT GT! )
```

```
                {#assignment = #(#[SUBEQUALSINT,"SUBEQUALSINT"],
#assignment);})

 | ((l:ID DOT! m:ID EQUALS! n:ID DOT! o:ID p:BINARY_OP q:ID DOT! r:ID)

                {#assignment =
#(#[SUBEQUALSSUBBINOPSUB,"SUBEQUALSSUBBINOPSUB"], #assignment);})

 | ((aa:ID DOT! bb:ID EQUALS! (cc:INT) dd:BINARY_OP ee:ID DOT! ff:ID)

                {#assignment =
#(#[SUBEQUALSINTBINOPSUB,"SUBEQUALSINTBINOPSUB"], #assignment);})

 | ((ll:ID DOT! mm:ID EQUALS! nn:ID DOT! oo:ID pp:BINARY_OP qq:INT)

                {#assignment =
#(#[SUBEQUALSSUBBINOPINT,"SUBEQUALSSUBBINOPINT"], #assignment);})

 | ((rr:ID DOT! ss:ID EQUALS! tt:INT uu:BINARY_OP vv:INT)

                {#assignment =
#(#[SUBEQUALSINTBINOPINT,"SUBEQUALSINTBINOPINT"], #assignment);})



 | ((a1:ID DOT! b1:ID EQUALS! c1:ID )

                {#assignment = #(#[SUBEQUALSOBJ,"SUBEQUALSOBJ"],
#assignment);})

 | ((a2:ID EQUALS! b2:ID DOT! c2:ID)

                {#assignment = #(#[OBJEQUALSSUB,"OBJEQUALSSUB"],
#assignment);})

 | ((ID EQUALS! ID DOT! ID BINARY_OP ID DOT! ID)

                {#assignment =
#(#[OBJEQUALSSUBBINOPSUB,"OBJEQUALSSUBBINOPSUB"], #assignment);})

 | ((ID EQUALS! ID DOT! ID BINARY_OP INT)

                {#assignment =
#(#[OBJEQUALSSUBBINOPINT,"OBJEQUALSSUBBINOPINT"], #assignment);})

 | ((ID EQUALS! INT BINARY_OP ID DOT! ID)

                {#assignment =
#(#[OBJEQUALSINTBINOPSUB,"OBJEQUALSINTBINOPSUB"], #assignment);})

 | ((ID EQUALS! INT BINARY_OP INT)

                {#assignment =
#(#[OBJEQUALSINTBINOPINT,"OBJEQUALSINTBINOPINT"], #assignment);})
```

```
)

;



//THINGS TO ADD: SUBEQUALSOBJ, OBJEQUALSOBJ==GOOD, OBJEQUALSSUB,
OBJEQUALSSUBBINOPSUB, OBJEQUALSINTBINOPSUB

//OBJEQUALSSUBBINOPINT, OBJEQUALSINTBINOPINT




//repeat <> {

// body

//};

repeat: REPEATTEXT^ LT! a:INT GT! OPENBRACE!

    {



            repeatbody();



            AST tmp=returnAST;



            for (int i=1;i<Integer.parseInt(a.getText());i++ ){

                tmp.setNextSibling(astFactory.dupTree(tmp));

                tmp=tmp.getNextSibling();

            }

        if ((Integer.parseInt(a.getText()))>0) {

            astFactory.addASTChild(currentAST, returnAST);

        }

    }
```

```
    CLOSEBRACE!;


repeatbody: ((bd:body)*)

{#repeatbody= #(#[REPEATBODY,"REPEATBODY"], #repeatbody);};



//move object <x,y,z>

move: MOVETEXT^ a:ID LT! (b:INT|NEGATIVEINT) COMMA! (c:INT|NEGATIVEINT) COMMA!
(d:INT|NEGATIVEINT) GT!;



// scale object <x,y,z>

scale: SCALETEXT^ a:ID LT! (b:INT|NEGATIVEINT) COMMA! (c:INT|NEGATIVEINT)
COMMA! (d:INT|NEGATIVEINT) GT!;



unaryop: ((a:ID DOT b:ID c:UNARY_OP

        {#unaryop = #(#[UNARYOPOBJ,"UNARYOPOBJ"], #unaryop);

        })



| (d:ID e:UNARY_OP

        {#unaryop = #(#[UNARYOPINT,"UNARYOPINT"], #unaryop);

}));



functionCall: FUNCT! a:ID

{#functionCall = #(#[FUNCTIONCALL,"FUNCTIONCALL"], #functionCall);};




class L extends Lexer;



options
```

```
{
    k=2;
}

//SPACE_PLUS: (' ')+;
DOT: ".";
COMMA: ',';
OPENBRACE: "{";
CLOSEBRACE: "}";
EQUALS: "=";
DOUBLE_EQUAL: "==";
LT: "<";
GT: ">";
//FUNCTION_BEGIN: "function ";

SEMICOLON: ';';

INT: ('0'..'9')+;
NEGATIVEINT: ('~') ('0'..'9')+;

ID : ('a'..'z' | 'A'..'Z') ('a'..'z' | 'A'..'Z' | '_' | '0'..'9')* ;

//COMMENTLINE
//: '%'
```

```
COMMENTLINE:  '#' ((options {greedy=false;}: 'a'..'z' | 'A'..'Z' | '_' |
'0'..'9' | ' ' | ';' | '.' |BINARY_OP|COMMA|OPENBRACE|CLOSEBRACE|EQUALS|LT|GT|
'!' | '@' | '#' | '$' | '^' | '&' | '?' | '[' | ']' | '~' | LPAREN | RPAREN)*
('\n'))

{ _ttype = Token.SKIP; };



//COMMENTLINE: '#' (options {greedy=false;}: ((~'\n')))* ('\n')

//{ _ttype = Token.SKIP; };




//NEWLINE

//    :   '\r' '\n'   // DOS

//    |   '\n'        // UNIX

//    ;




//COMMENT: '#' COMMENTLINE { _ttype = Token.SKIP; };



UNARY_OP: ("++"|"--");



BINARY_OP:

       ('+'|'-'|'*'|'/'|'%')

       ;



LPAREN: '(';

RPAREN: ')';



// Whitespace -- ignored

WS     :      (      ' '
```

```
            |      '\t'
            |      '\f'
                   // handle newlines
            |      (      options {generateAmbigWarnings=false;}
                   :      "\r\n"  // Evil DOS
                   |      '\r'    // Macintosh
                   |      '\n'    // Unix (the right way)
                   )
                   { newline(); }
            )+
            { _ttype = Token.SKIP; }
      ;
```

eMDL.java:

```
/*
      @author Jerin Kurian jk1243
    */
import java.io.*;
import antlr.collections.AST;
import antlr.collections.impl.*;
import antlr.debug.misc.*;
import antlr.*;
import java.awt.event.*;

class eMDL {
    public static void main(String[] args) {
```

```
    try {
        if (args.length != 1){
         System.err.println("Correct usage: java eMDL <source file>");
         System.exit(1);
       }

       String filename = args[0];

       L lexer = new L(new FileInputStream(filename));
       //L lexer = new L(new DataInputStream(System.in));
        P parser = new P(lexer);
        parser.startRule();
       doTreeAction("test.txt", parser.getAST(),parser.getTokenNames());
       emdlTreeWalker walker = new emdlTreeWalker();
       walker.startRule(parser.getAST());
    } catch(Exception e) {
        System.err.println("exception: "+e);
    }
}

public static void doTreeAction(String f, AST t, String[] tokenNames) {
   if ( t==null ) return;
   ((CommonAST)t).setVerboseStringConversion(true, tokenNames);
   ASTFactory factory = new ASTFactory();
   AST r = factory.create(0,"AST ROOT");
   r.setFirstChild(t);
```

```
        /*
        final ASTFrame frame = new ASTFrame("Java AST", r);
        frame.setVisible(true);
        frame.addWindowListener(
                            new WindowAdapter() {
                                public void windowClosing (WindowEvent e) {
                                   frame.setVisible(false); // hide the Frame
                                   frame.dispose();
                                   System.exit(0);
                                }
                            }
                            );
        */
        // System.out.println(t.toStringList());
        //JavaTreeParser tparse = new JavaTreeParser();
        //try {
        //tparse.compilationUnit(t);
        // System.out.println("successful walk of result AST for "+f);
        //}
        //catch (RecognitionException e) {
        //   System.err.println(e.getMessage());
        //   e.printStackTrace();
        //}


    }
}
```