Programming Languages and Translators
COMS W4115

Prof. Stephen Edwards
Summer 2008

VINL
VINL Is Not Logo

Final Project Report

Yang Cao
August 11, 2008

Table of Content

White Paper of VINL

Introducing VINL

VINL is graphic programming language targeting introducing younger children to the world of programming. VINL's visual power to show programming result on screen with its simple syntax makes programming in VINL a fun activity. VINL has its root in Logo programming language and is largely a subset of Logo's turtle graph drawing ability with refined syntax.

Why VINL while we already have Logo?

While Logo is feature complete in terms of turtle graphic ability, its syntax is showing age -- it's just not fun to write “repeat / end repeat” all the time. Some dialect of Logo replaces the “repeat / end repeat” sequence with brackets [] but the syntax is not standardized. Furthermore, Logo's syntax is very different from popular programming languages widely in use today. On the other hand, VINL is designed based on popular programming languages such as C/C++/Java syntactically, and we believe it's easier for the young users of VINL to make a leap to C/C++/Java in the future.

Language Overview

Turtle

The turtle starts in the middle of the screen, and it can leave a trail when it has been moved. The turtle knows where it is at (the X and Y coordinates) and which direction it is facing (the polar angle in polar coordinate system). It can only move straight forward or backward.

Data Types

We have the following data types in VINL:

- decimal
- boolean
- void

Unlike C, boolean and decimal are fundamental different data types and one can not use those two data types interchangeably.

Functions

Function declaration is somewhat relaxed in VINL. One does not have to specify a return type compared with Java's function declaration. Function declaration starts with the keyword “funct” followed by function name, then followed by a list of parameters enclosed by (), and lastly a function body which consists of multiple statements enclosed by {}.

Flow control statements and their syntax

- if -- if (boolean) {...statements...}

the statements in {} will be executed once if and only if the boolean value in () followed by keyword “if” is evaluated to true. In the case an expression is used inside (), the expression will be evaluated to be a boolean value first then passed to if.

- while – while (boolean) {...statements...}

while is similar to if. The only difference is the statements in {} will be executed as long as the boolean value remains to be true after each iteration.

- for – for(initialization;condition;post-action) {...statements...}

for can be rewrite as a while loop:

```
initialization
while(condition) {
        ...statements...
        post-action
}
```

Operators

VINL supports all common arithmetic operations including: +.-.*,/, and %. The assignment operator is denoted with <-. Comparison between two expressions are supported by the following operators: <, >, =, and !=. VINL also support logical operator && and ||.

Comments

Comments start with //. Everything follows // on the same line is considered comments. If there are multiple // on the same line, everything follows the first // is considered comments. The subsequent // has no meaning.

Language Tutorial

Core VINL

Unlike most other language tutorial that starts with a “hello world” program, there is none in VINL. I consider the following statement as the “hello world” program:

```
forward 100;
```

The effect of this statement would be move the turtle forward by 100 steps and leave a straight vertical line on screen because the turtle starts facing north. We can also turn the turtle:

```
left 90;
```

This has no visible effect, but it turns turtle 90 degrees to the left, now the turtle is facing west. Repeat those two statements 3 more times, and you will have a square on screen. The essence of VINL is to feed a sequence of commands to the turtle so it will draw interesting pictures on screen.

Now let's look at some examples:

```
//This program shows ability of function definition, functions calls and for/while loops operation
//on top of regular variable declaration, assignment, and VINL's drawing abilities.

funct circle(decimal stepsize) {
        var decimal i <- 0;
        while( i < 360 ) {
                fw stepsize;
                rt 1;
                i<-i+1;
        }
}

funct drawMultiCircles(decimal num, decimal stepsize) {
        //only to show how function return a value
        var decimal turn_deg <- calcDeg(num);
        for(var decimal i<-0; i<num; i<-i+1) {
                circle(stepsize);
                left turn_deg;
        }
}

funct calcDeg(decimal num) {
```

```
23          return (360 / num);
24 }
25 drawMultiCircles(24, 2);
```

Let's analysis this program in detail. Line 1 and 2 are comments, they start with // and everything follows on the same line is considered comments and are ignored by the language interpreter. Similarly line 3, 12, and 21 are empty lines, they are also ignored.

```
 4 funct circle(decimal stepsize) {
 5          var decimal i <- 0;
 6          while( i < 360 ) {
 7                  fw stepsize;
 8                  rt 1;
 9                  i<-i+1;
10          }
11 }
```

Line 4 to 11 defines a new function “circle”, this function takes one parameter called “stepsize”, and declares its own local variable “i” which is initialized to 0 at line 5. Both variables are type decimal and they only live in this code block. Line 6 to 9 declares a standard while loop, note at line 7 and 8, VINL understands some abbreviations of popular command such as forward, backward, left, right, penup, pendown and clear. The abbreviations are fw, bw, lf, rt, pu, pd and cl respectively. Convince yourself that this while loop will indeed draw a circle. With the declaration of this function, we can simply call “circle” with a decimal to draw a circle on screen instead of typing the while loop each time. Pretty neat.

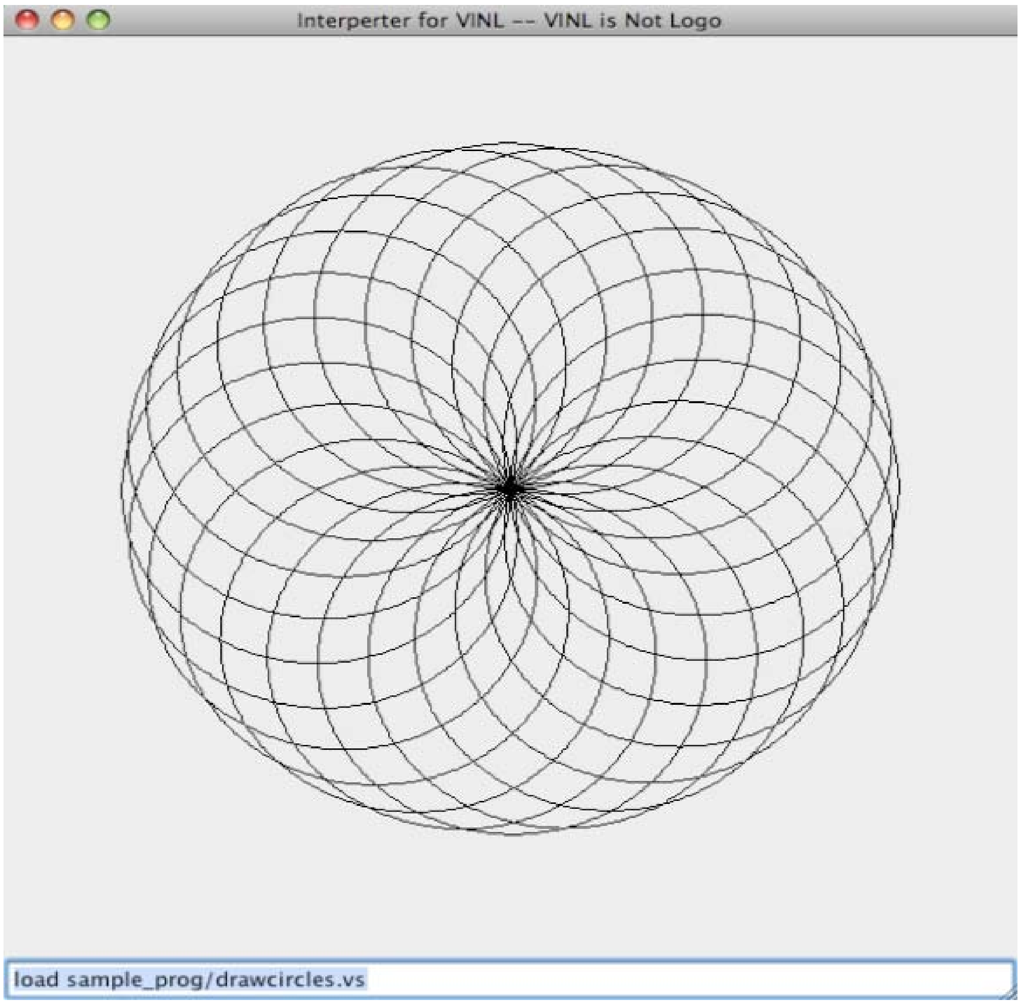
```
13 funct drawMultiCircles(decimal num, decimal stepsize) {
14          //only to show how function return a value
15          var decimal turn_deg <- calcDeg(num);
16          for(var decimal i<-0; i<num; i<-i+1) {
17                  circle(stepsize);
18                  left turn_deg;
19          }
20 }
```

Line 13 to 20 defines another function that takes two decimal variables. Notice at line 15, we initialize the local variable turn_deg by calling another function. In addition to grouping statements together, function could also compute values and return such value. We will take a look at function “calcDeg” in next section. Line 16 to 19 defines a standard “for” loop that calls our first function circle, recalls that “circle” will indeed draw a circle on screen, turns toward left for some degree and repeat this process multiple times.

```
  22 funct calcDeg(decimal num) {
 23         return (360 / num);
 24 }
```

Line 22 to 24 defines a function that has nothing to do with the turtle. It simply calculates a number and returns this number when it's called. "return" keyword terminates the function call, if there are any statements that come between line 23 and 24, they will never be executed.

Lastly at line 25, we call upon our function and this is the result:



```
load sample_prog/drawcircles.vs
```

This program source is stored in a file called "drawcircles.vs", we can simply loaded it in the interpreter to avoid all the typing. This functionality will be discussed in the interpreter section coming up.

Let's step back and take a bird view of this program, notice function "drawMultiCircles" references “calcDeg” before “calcDeg” is defined. This is okay in VINL. However, the actually function call at line 25 happens at the end, this is required otherwise we will get a “symbol not defined” error. The requirement is all dependencies must be resolved before a statement can issue a function call. If “drawMultiCircles” does not call “calcDeg”, then it's all right to move line 25 up to line 21, since “drawMultiCircles” would only depend on “circles”, and it has been declared, so all dependencies can be resolved. However given the current implementation, the function call must occur at line 25 or later.

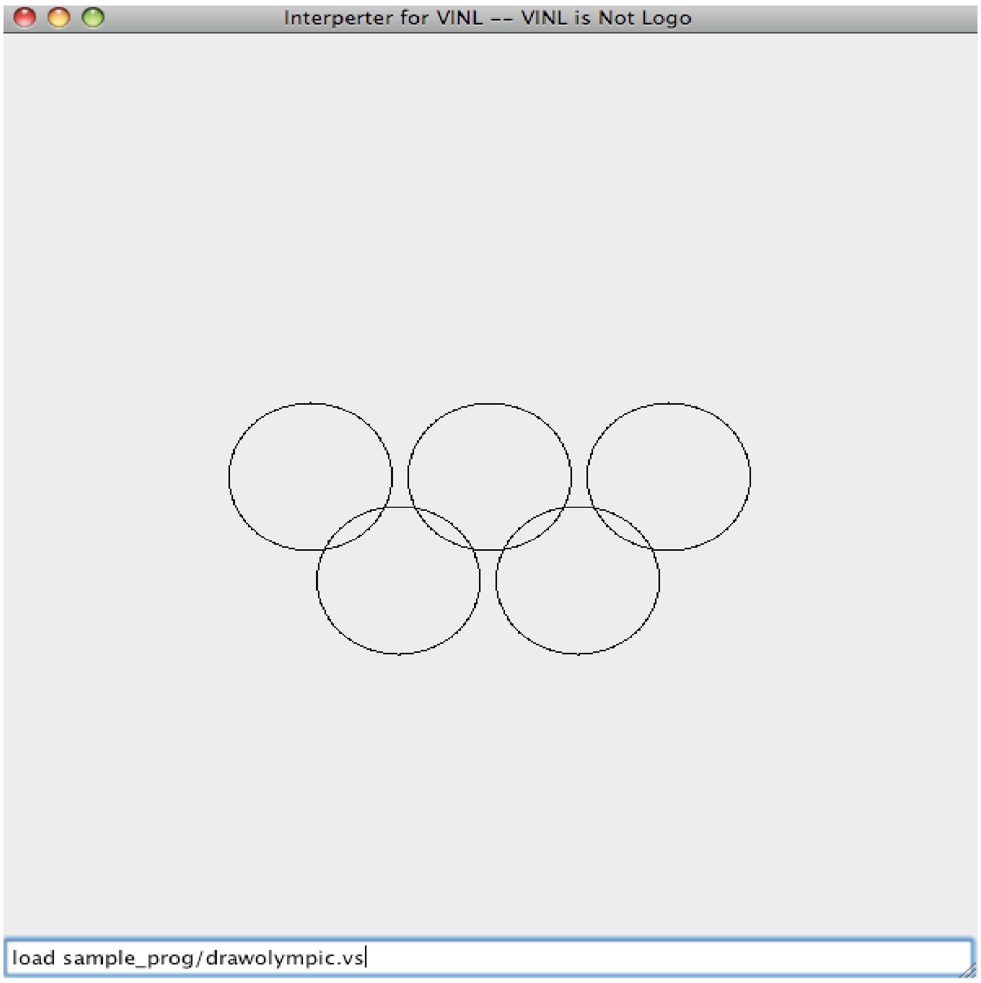
Let's examine another program in Olympic spirit during this exciting time:

```
 1 //different approach to draw a circle centered at current turtle location
 2 funct rcircle(decimal radius) {
 3         var decimal stepsize <- radius*2*3.1415926/360;
 4         pu; left 90; fw radius; right 90; pd;
 5         for(var decimal i; i<360; i<-i+1) {
 6                 fw stepsize;
 7                 rt 1;
 8         }
 9         pu; right 90; fw radius; left 90; pd;
10 }
11
12 funct drawOlympicFlag(void) {
13         pu; rt 90; bw 110; pd;
14         rcircle(50);
15         pu; fw 110; pd;
16         rcircle(50);
17         pu; fw 110; pd;
18         rcircle(50);
19         pu; rt 90; fw 70; rt 90; fw 55; pd;
20         rcircle(50);
21         pu; fw 110; pd;
22         rcircle(50);
23 }
24
25 drawOlympicFlag(void);
```

Nothing groundbreaking in line 1 to 10, but notice we never initialized variable “i” in line 5. By default, all decimal variables are initialized to 0 and all boolean variables are initialized to false. Also we see we can put multiple statements on the same line. Line 12 defines a new function without any inputs; however a place holder “void” is required in such situations. The same idea applies to line 25, it also applies to “for” loops and line 5 to 8 can be rewritten as:

```
        var decimal i;
        for(void; i<360; void) {
                fw stepsize;
                rt 1;
                i<-i+1;
        }
```

The output of this program is:



```
load sample_prog/drawolympic.vs
```

There are other programs that can be found under the “sample_prog” folder. They are also attached on the code listing section. They can all be executed by loading them in the interpreter.

Interpreter Overview

As the previous section suggests, the execution of a VINL program is done in an interpreter. I provided an interpreter implementation that supports all VINL defined functionalities.

Starting the interpreter is straight forward a Java call. The main class is VinlInterpreterGui.

```
            java -classpath .:lib/antlr.jar VinlInterpreterGui
```

Once the GUI is started, there will be an input bar at the bottom. It accepts one statement a time; “enter” key will feed the statement to the interpreter. It also has a basic history feature, press the up/down array will bring up previously entered commands. By starting the command with a “load” command, it will read the file follows the load command. Notice “load” is not a VINL defined keyword, and this load command is not a VINL statement – it is wrong to terminate it with a semi colon. You are free to use “load” as identifiers in your program. Also, load a program into current session does not create new scope, so every variable defined in the source file will be defined in current session as well. Think it as #include in C.

To quit the interpreter, type “exit”, “bye” or “quit” as the input.

Language Reference Manual

Lexical Conventions

Tokens

There are five classes of tokens: identifiers, keywords, constants, operators and other separators. White-space consists of space, tab and new line, and is separators of tokens.

Comments

Comments starts with // and terminates with end of line.

Identifiers

An identifier is a sequence of letters, digits and underscores. Identifiers can not start with a digit. Identifiers are case sensitive.

Keywords

The following identifiers are reserved for use as keywords, and may not be defined as variable names or function names:

Variable Types:
```
boolean   decimal   void
```

Logic Flow Control:
```
break     continue  else      for
if        return    while
```

Turtle Operations:
```
bakdward  bw        clear     cl        forward   fw
home      left      lf        right     rt
```

Declaration:
```
var       funct
```

Debugging:
```
echo      status
```

Floating Constants

Floating constants consists of a mandatory integer part, an optional decimal point and fraction part.

Other Tokens

Other tokens used in VINL including: +, -, *, /, %, <-, (,), {,},&&, ||, <, >, <=, >=,!=,=

Types

VINL is statically typed. All variables must be declared as either a decimal or a boolean. There will be no explicit nor implicit casting in VINL between boolean and decimals.

decimal

A floating constant. The memory storage size of decimal type is not specified. In this implementation, it will be 64bit value. Behaviors on operations on a decimal resulting in overflow , divide check and other exceptions are also not specified.

boolean

Either true or false

void

Denotes nothingness. One cannot create a variable of type void. Used only in function declaration to show the function does not return any value.

Expressions

Primary Expressions

Primary expressions include identifiers, constants, function calls, and expressions in parentheses.

- ***identifiers***

  identifiers are lvalues and are evaluated to values stored in this memory space

- ***constants***

  constants are evaluated to itself.

- ***Function Calls***

  A function call consists of a identifier declared as a function, followed by optional white space characters, followed by a list of arguments surrounded by “(“ and “)”.

Operative Expressions

The operators are listed by precedence, highest to lowest; they take primary expressions as operands:

- ***Multiplicative Operators***

  *, /, and % group left to right. Only defined for decimal type constants and identifiers.

- ***Additive Operators***

  + and – group left to right. They are only defined for decimal type constants and identifiers.

- ***Relational Operators***

  <,>,<=,>= ,!=, and = can be used to compare two decimals, and they yield boolean values. Those operators can not be chained, e.g. (a<b)<c since (boolean)<c is not allowed.

- ***Logical AND Operator***

  && group left to right and can be used in between two boolean expressions. It yields boolean values. It evaluates to true if and only if both boolean expressions are true.

- ***Logical OR Operator***

  || group left to right and can be used in between two boolean expressions. It yields boolean values. It evaluates to true if and only if at least one of the operands is true.

Assignment Expressions

There is only one assignment operator <- and it groups right to left. It requires a type matching identifier on the left side of the operator. The value of the expression on the right of the operator will be stored in the left identifier.

Statements

Statements are executed in sequence unless otherwise noted. They do not have values, and are executed for their effects. Multiple statements maybe grouped together by "{" and "}", in that case they are treated as if they are a single statement.

Expression Statement

Expression statements have the form:

```
expression-stmt: (expression)? ;
```

All effects of the expression is completed before the next statement is executed. An empty expression statement is a null statement and is often used for grouping related statements.

Selection Statement

Selection statements have the form:

```
                  selection-stmt : if ( expression ) statement
                                 | if ( expression ) statement else statement
                                 ;
```

The “else” groups with the closest else-less if.

Iteration Statement

Iteration statements have the form:

```
             iteration-stmt  : while ( expression ) statement
                             | for (expression;expression;expression) statement
                             ;
```

The while loop stops if and only if the expression evaluates to false. The for loop is equivalent to the following while loop:

```
initialization; //first optional expression
while (expression) //second optional expression
      statement
      post-action //third optional expression
```

The keyword break and continue may only appear in the statement inside a loop body. Break will stop the execution of the inner-most loop immediately; continue will stop the current iteration of the inner-most loop and starts next iteration of this loop immediately.

Keyword statement

```
Keyword statements have the form:
     keyword-stmt : forward decimal
                  | right decimal
                  | … (other keyword statements)
                  ;
```

Scope

All source code can only be stored in one file.

Global variables are variables defined outside of any function body, and can be accessed from everywhere in the source.   The variable can be reference only if it's already declared prior.

Local variables are variables defined inside a function body, and can only be referenced inside the function where it has been declared.

No other statement declare any kind of scope.   If a nested “for” loop has the following form:

```
for(var decimal i<-0; i<x; i<-i+1) {
     for(var decimal j<-0; j<x; j<-j+1) {
     }
}
```

This is wrong because the second for loop does not declare a new scope, j is placed on the function scope or global scope, the next iteration will try to redefine j as a new variable, and redefinition of variables are not supported in VINL.

Project Plan

Programming Style Guide

All Java source code is written with the Java style guide from Sun Microsystems in mind.

Antlr code is written with the following rules:

1.If it fits in one line cleanly, do so. However don't be afraid to break a simple statement into multiple lines if it adds clarity.

2.For one liner, the format is: token <\t> : statements ;

3.For multiple lines, the format is: token <\n> <\t> : statements <\n> <\t> | statements <\n> <\t> ;

4.In the walker, there were embedded Java code, this piece of Java code is written with C style in mind. Namely place the left bracket on a new line and match the terminating right bracket on the same column.

Since this is a one man project, I didn't experience any hardship regarding dissatisfaction of programming style used whatsoever. I am happy with the style that I used. Never had a bug or had a debugging difficulty due to programming style used.

Project Plan

Logo was the first programming language I was exposed to on a very old and slow 8 bit Nintendo clone gaming system. As soon as I registered for this class, I knew I wanted to write a project that mimics Logo. I was deeply awed by Logo's ability as a kid and I wanted to create something similar.

My plan was to start the project in three phases:

1.Creating a GUI that could display turtle movements

2.Creating a parser for VINL

3.Creating a walker that interprets VINL

The first phase does not involve any ANTLR or any principles studied in PLT, so I started on that as my first step. I created a GUI that responds to keystrokes. If I press 'f' key, it would go forward 100 steps; if I press 'l', it turns 90 degrees, and etc. I finished this phase around mid of June.

Then I started working on the parser around the time that Language Reference Manual is due. I used the reference manual literally as a manual to write the parser. The progress on writing the parser was slower than I had anticipated and due to other engagement I only finished the first working draft of parser around the first week of July. I've created a testing file that utilizes the simple AST frame code on the course website and this file has proven to be enormous help in both testing the parser and creating the walker. Even when working on walker, I adjusted the parser numerous times—I added new keywords to simplify debugging, changed variable/function declaration rules, and changed the AST root child relationship for many tokens.

I started on the walker one week later and it's not going well at first. At first I didn't really get a hang of the walker at all, and even after reading over several ANTLR tutorials online, it's still not clear to me. Then I decided to just go ahead and try to make the simplest command “forward” work. I modified the GUI to take command inputs instead of keystrokes, and created a suite of Java classes that represents elements needed to make VINL work. Mx walker really helped me get going. I used Mx's structure, but I did not consult Mx's Java implementation of the interpreter. Once the forward command works, it became quite clear how the parser, walker and the backend Vinl Java code worked together and I really picked up on speed for writing the walker.

I had a few obstacles during writing the walker. The first one is how to break and continue a loop. I implemented a very inefficient way that all statements after break and continue are still examined, but they are not executed if a flag is set. I am still searching for a better way to really break/continue the loop. The same goes to the return statement as well. There are a few things that I'd like to add to VINL in the future, this is definitely on the list.

I finished the entire project around end of July, but my code was not rigorously tested at that time. I started testing my code after the final exam on August 4th.

Development Environment

Antlr version used: 2.7.7

Java version used: 1.5.0_13

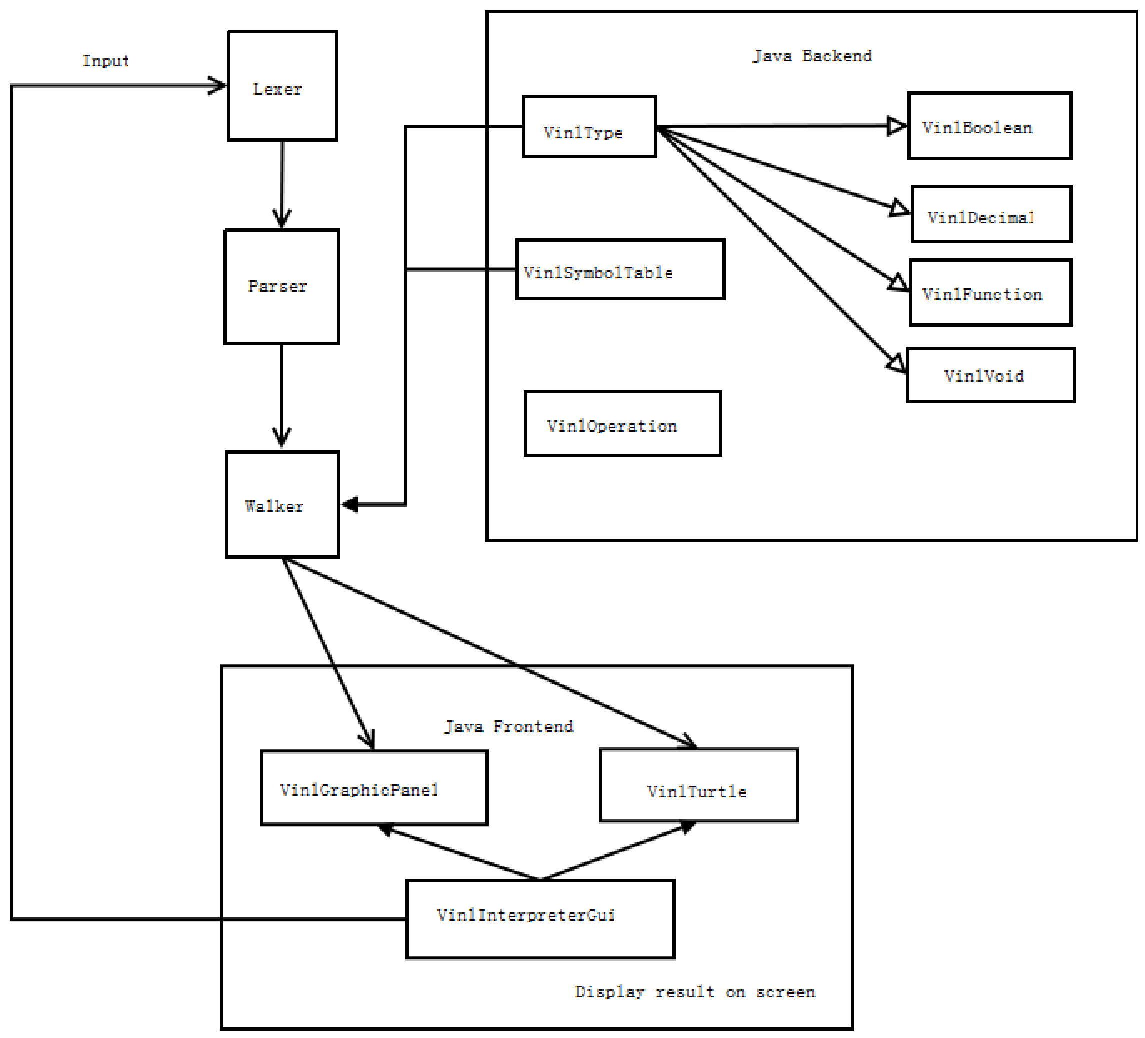
The project is done on a macbook.

I started the Java portion of project in NetBean 6.1 environment, but as most changes were in the antlr source, and I couldn't figure out a way to automatically compile antlr source in NetBean, I dropped it in favor of plain old bash shell with vim. At least 2/3 of the coding is done in vi.

To automatically compile the antlr sources with Java sources, I created a very simple Makefile that lists both antlr source and java source as prerequisites of the default target. Although very simple, it gets the job done. I also created another launch script run.sh that starts VINL interpreter GUI. It has saved me a lot of typing as well.

Since this is a one man project, I didn't find a version control system irreplaceable. I registered for google project hosting service but didn't utilize it much. NetBean auto saves local history changes so I didn't need version control while I was working with NetBean. After I switched to vi, vi also provided unlimited undo so I just had to make sure I don't close vi accidentally. It worked well for one man project. Had I worked in a team, I am certain the version control software could become more important, just as version control database is probably the most important piece of software at any software development site.

Architectural Design



The VinlInterpreterGui is the starter. It gathers input and sends it to the lexer, parser and walker. Walker uses VinlType and VinlSymbolTable to parse the AST tree to objects and functions understandable by VinlGraphicPanel and VinlTurtle. Walker uses the java backend to control the frontend. VinlInterpreterGui shares the same VinlGraphicPanel and VinlTurtle object Walker uses, so every output sent from Walker to VinlGraphicPanel and VinlTurtle will be reflected in VinlInterpreterGui as on screen display, this is the output visible to user.

VinlType is an interface, VinlBoolean, VinlDecimal, VinlFunction, and VinlVoid implements this interface.

Testing & Debugging Plan

Since VINL is a visual language, I did not find a good way to automatically test the whole project. So I broke the testing into 2 phases:

1. Testing the parser and lexer could be automated because the AST tree node has a toStringList() function. I created a cases.txt which contains phrases that should satisfy the lexer and parser, and feed this text file to a driver that instantiate lexer and parser instance. The error checking are two folding, first I can easily check if there are any errors in the stdout. If there is none, it means it satisfies the grammar, and this part takes no effect. Secondly, my driver also creates an AST tree frame, so I can check each AST sub tree to see if it's in the format I expected. This sounds like a lot of work, but it's actually not, because after the initial work, every time I am only changing a small subset of the entire grammar, and I am confident it only affects a small subset of AST trees, so it is actually workable. I start this process from a script which first compiles the parser so we are confident we are testing the latest source and starts the driver with the text file after compiling parser.

2. Testing the walker is more trouble; I have created logging method that can display messages according to a debug level. I have also provided means to log the turtle's coordinates and its angle. Most of my testing is done as unit test while creating the walker—I would send different variation of the AST node I am working on that time to see if the walker parse it correctly. Later on when I am done with the project, I tested the walker with the programs in "sample_prog" folder. After I make changes, I usually tests a few unit cases then load those programs one by one and make sure nothing is broken. My interpreter does not have a mean to automatically load a program at startup, so I couldn't automatically test the interpreter. However loading each sample program is easy and it's fun to look at the displays.

The testing cases can be found in appendix code listing as well as in the source "test" folder.

Lessons Learned

A good Language Reference Manual goes a long way. It makes writing the grammar like a translator's work. It is also part of the final project, so it really pays off to spend more time on the Language Reference Manual at the beginning of the course.

ANTLR has unfamiliar syntax to me, but in the end it is still Java. There were times that I just couldn't figure out what is wrong in my antlr source, so I dived in to the generated JAVA code and although it's not the easiest thing to debug, but you can modify the code to help you. You can add print lines, check object values and once I added many stack trace calls to the code and found out the bug eventually.

Think like a user and make the language more enjoyable to write. After all, you will probably spend the most time writing in your own languages. I made the choice of using <- as the assign operator, and I later felt it is not very user friendly as you need to hold the shift key to type '<' but you immediately need to release the shift key to type '-'. I probably will change the assign operator to ':' in the future. It still requires the shift key, but it only has one character.

Plans for Future

- I added keyword 'var' and 'funct' to the grammar because it's easier to parse in parser. I was getting non-determinism errors without those two keywords. However I really don't like the extra typing.
- More subtle scoping rule. Should create a new scope every time we see brackets pairs. Then we don't have to worry about define the index first before executing nested for loops. The current scope rule only provides global and function level scope, it could be improved.
- Support recursive function calls. Currently my implementation of symbol table does not support recursive function because I treat functions like a static instance, so in a recursive call, each function instances share the same variables and symbol table. Need to add the recursive call to the current symbol table as a new instance of this function.
- Creating colors. It would be a lot more fun if the pictures are colorful. This is very easy to implement but I feel it is too basic to add any academic value to this project, so I decided to leave it out for this course, but will add it to VINL in the future.
- There is no visible turtle on screen yet. I have the code to draw the turtle but it slows down the execution dramatically. That’s because I draw the turtle every time the forward function is called. In the future, I want to implement a work queue and I can place all forward calls on the queue, and draw the turtle if and only if the queue is empty. This should improve the performance significantly.

Appendix Code Listing

VINL Source Code:

vinl_grammar.g:

```
/*
 * vinl_grammar.g : the lexer and parser in ANTLR grammar for VINL
 *
 * @author Yang Cao
 *
 * @version $id:
 */

class VinlLexer extends Lexer;

options {
      k = 2;
      charVocabulary = '\3'..'\377';
      testLiterals = false;
      exportVocab = VinlAntlr;
}

protected
ALPHA : 'a'..'z' | 'A'..'Z' | '_';

protected
DIGIT : '0'..'9';

WS    : (' ' | '\t')+         {$setType(Token.SKIP);};

EOL  : ('\n' | ('\r' '\n') => '\r' '\n' | '\r') {$setType(Token.SKIP); newline();};

COMMENT     : "//" (~('\n' | '\r'))* (EOL) {$setType(Token.SKIP);};

LPAREN      : '(';
RPAREN      : ')';
MULT  : '*';
DIV   : '/';
MOD   : '%';
PLUS  : '+';
MINUS : '-';
SEMI  : ';';
LBRACE      : '{';
RBRACE      : '}';
ASSIGN      : "<-";
COMMA : ',';
EQ    : '=';
NEQ   : "!=";
GT    : '>';
GE    : ">=";
LT    : '<';
```

```
LE    : "<=";
AND   : "&&";
OR    : "||";

ID    options { testLiterals = true; }
      : ALPHA (ALPHA|DIGIT)*
      ;

/*
 * DECIMAL: consists of an mandator integer part with an optional fraction part
 * e.g. 1, 2.34, 567
 */
DECIMAL     : (DIGIT)+ ('.' (DIGIT)+)? ;

/*
 * STRING: a sequence of chars surrounded by double quotes ",
 * double quote inside a string is escaped by \.  New line is not allowed.
 */
/* string is not useful in vinl
STRING      : '"'!
            (~('"' | '\n') | ('\'!'"'))*
        '"'!
      ;
*/



/************************************************************
 *  parser
 ************************************************************/

class VinlParser extends Parser;

options {
      k = 2;
      buildAST = true;
      exportVocab = VinlAntlr;
}

tokens {
      STATEMENT;
      FUNC_CALL;
      EXPR_LIST;
      PARAMS_LIST;
      FUNC_BODY;
}

cmd
      : ( statement | func_def )
      | ( "exit" | "bye" | "quit" )
            { System.exit(0); }
      | EOF!
            { System.exit(0); }
      ;

program
      : ( statement | func_def )* EOF!
            { #program = #([STATEMENT, "PROGRAM"], program); }
      ;

statement
      : selection_stmt //if
      | expression_stmt
      | iteration_stmt //while and for
      | keyword_stmt
      | LBRACE! (statement)* RBRACE!
            {#statement = #([STATEMENT,"STATEMENT"], statement);}
      ;

expression_stmt
      : expression SEMI!
      ;

selection_stmt
      : "if"^ LPAREN! expression RPAREN! statement
            (options {greedy = true;}: "else"! statement)?
      ;

iteration_stmt
      : while_stmt
      | for_stmt
      ;

while_stmt
      : "while"^ LPAREN! expression RPAREN! statement
      ;

for_stmt
      : "for"^ LPAREN! expression SEMI! expression SEMI! expression RPAREN!
statement
      ;

keyword_stmt
      : pendown_stmt
      | penup_stmt
      | home_stmt
      | forward_stmt
      | backward_stmt
      | left_stmt
      | right_stmt
      | clear_stmt
      | break_stmt
      | continue_stmt
      | return_stmt
      | echo_stmt
      | status_stmt
      ;

break_stmt
      : "break" SEMI!
      ;

continue_stmt
      : "continue" SEMI!
      ;

return_stmt
      : "return"^ (expression)? SEMI!
      ;

pendown_stmt
      : ("pendown" | "pd") SEMI!
      ;

penup_stmt
      : ("penup" | "pu") SEMI!
      ;

home_stmt
      : ("home") SEMI!
      ;

forward_stmt
      : ("forward"^ | "fw"^) expression SEMI!
      ;

backward_stmt
      : ("backward"^ | "bw"^) expression SEMI!
      ;

right_stmt
      : ("right"^ | "rt"^) expression SEMI!
      ;

left_stmt
      : ("left"^ | "lf"^) expression SEMI!
      ;

echo_stmt
      : ("echo"^) expression SEMI!
      ;

clear_stmt
      : ("clear" | "cl") SEMI!
      ;

status_stmt
      : ("status") SEMI!
      ;

type_specifier
      : "decimal"
      | "boolean"
      ;

return_type_specifier
      : type_specifier
      | "void"
      ;

func_def
      : "funct"^ ID LPAREN! params_list RPAREN! func_body
      ;

func_body
      : LBRACE! (statement)* RBRACE!
            {#func_body = #([STATEMENT, "FUNC_BODY"], func_body); }
      ;

params_list
      : param (COMMA! param)*
            {#params_list = #([PARAMS_LIST, "PARAMS_LIST"], params_list);}
      ;

param
      : type_specifier ID
      | "void"
      ;


expression
      : operative_expr //includes primary_expr
      | variable_decl_expr
      | assignment_expr
      ;

primary_expr
      : ID
      | constant_expr
      | func_call_expr
      ;
```

```
constant_expr
      : "true"
      | "false"
      | DECIMAL
      ;

//func_call_stmt
func_call_expr
      : ID LPAREN! expr_list RPAREN!
            {#func_call_expr=#([FUNC_CALL,"FUNC_CALL"], func_call_expr);}
      ;

expr_list
      : expression (COMMA! expression)*
            {#expr_list = #([EXPR_LIST, "EXPR_LIST"], expr_list);}
      ;

variable_decl_expr
      : "var"^ type_specifier ID (ASSIGN^ expression)?
      | "void" //null variable place holder expression
      ;

assignment_expr
      : lvalue ASSIGN^ expression
      ;

operative_expr
      : logical_expr (OR^ logical_expr)*
      ;

logical_expr
      : relational_expr (AND^ relational_expr)*
      ;

//no chaining of relational expressions
relational_expr
      : additive_expr ( (LT^ | LE^ | GT^ | GE^ | EQ^ | NEQ^) additive_expr)?
      ;

additive_expr
      : multiplicative_expr ( (PLUS^ | MINUS^) multiplicative_expr)*
      ;

multiplicative_expr
      : rvalue ( (MULT^ | DIV^ | MOD^) rvalue)*
      ;

rvalue
      : primary_expr
      | LPAREN! expression RPAREN!
      ;

lvalue
      : ID
      ;
```

vinl_walker.g:

```
/*
 * vinl_walker.g : the AST walker in ANTLR grammar for VINL
 *
 * @author Yang Cao
 *
 * @version $id:
 */

{
      import java.io.*;
      import java.util.*;
      import java.awt.*;
}

class VinlAstWalker extends TreeParser;
options {
      importVocab = VinlAntlr;
}

{
      static Hashtable<String, VinlFunction> function_table = new Hashtable();
      static VinlSymbolTable global_table = new VinlSymbolTable();
      static VinlSymbolTable curr_table = global_table;
      static Hashtable<String, String> keywords = new Hashtable();
      static VinlGraphicPanel gp;
      static VinlTurtle turtle;
      VinlType ret = null; // used in function return

      /* control flows */
      boolean inloop = false;
      boolean breaked = false;
      boolean continued = false;
      boolean proceed = true;
      boolean returned = false;

      /* DEBUG LEVEL */
      public static final int INFO = 3;
      public static final int DEBUG = 2;
      public static final int DEBUG_FINE = 1;

      public static int curr_level = 4; //no messages

      static {
            /* vinl keywords should not be used as identifiers.
               Should also forbid java keywords, however java keywords are not VINL
limitation,
               so I will left it for Java compiler to catch such error.
            */
            keywords.put("backward", "backward");
            keywords.put("bw", "bw");
            keywords.put("boolean", "boolean");
            keywords.put("break", "break");
            keywords.put("cl", "cl");
            keywords.put("clear", "clear");
            keywords.put("continue", "continue");
            keywords.put("decimal", "decimal");
            keywords.put("echo", "echo");
            keywords.put("for", "for");
            keywords.put("forward", "forward");
            keywords.put("funct", "funct");
            keywords.put("fw", "fw");
            keywords.put("home", "home");
            keywords.put("if", "if");
            keywords.put("left", "left");
            keywords.put("lt", "lt");
            keywords.put("pendown", "pendown");
            keywords.put("pd", "pd");
            keywords.put("penup", "penup");
            keywords.put("pu", "pu");
            keywords.put("right", "right");
            keywords.put("rt", "rt");
            keywords.put("var", "var");
            keywords.put("void", "void");

            gp = VinlInterpreterGui.getGraphicPanel();
      }

      boolean isKeyword(String name) {
            return keywords.containsKey(name);
      }

      static void log(String s, int level)  {
            if(level > curr_level)
            {
                  System.out.println(s);
            }
      }
}

expr returns [ VinlType r ]
{
      VinlType a, b;
      r=null;
      turtle = gp.getTurtle();
      String[] func_params;
      VinlType[] expr_list;
}
      : #(STATEMENT (stmt:. { r=expr(#stmt); })* )
      | #(PLUS a=expr b=expr)            {  if(proceed)  r=a.eval(VinlOperation.PLUS,
a, b); }
      | #(MINUS a=expr b=expr)            { if(proceed)
r=a.eval(VinlOperation.MINUS, a, b); }
      | #(MULT a=expr b=expr)             { if(proceed)
r=a.eval(VinlOperation.MULTI, a, b); }
      | #(DIV a=expr b=expr)              { if(proceed) r=a.eval(VinlOperation.DIV,
a, b); }
      | #(MOD a=expr b=expr)              { if(proceed) r=a.eval(VinlOperation.MOD,
a, b); }
      | #(LT a=expr b=expr)               { if(proceed) r=a.eval(VinlOperation.LT,
a, b); }
      | #(GT a=expr b=expr)               { if(proceed) r=a.eval(VinlOperation.GT,
a, b); }
      | #(GE a=expr b=expr)               { if(proceed) r=a.eval(VinlOperation.GE,
a, b); }
      | #(EQ a=expr b=expr)               { if(proceed) r=a.eval(VinlOperation.EQ,
a, b); }
      | #(NEQ a=expr b=expr)              { if(proceed) r=a.eval(VinlOperation.NEQ,
a, b); }
      | #(OR a=expr right_or:.)
            {     /* short circuit or */
                  if(proceed)
                  {
                        if (a instanceof VinlBoolean)
                              r=((VinlBoolean) a).getValue()?new
VinlBoolean(true):expr(#right_or);
                  }
            }
      | #(AND a=expr right_and:.)
            {
                  /* short circuit and */
                  if(proceed)
                  {
                        if (a instanceof VinlBoolean)
                              r=((VinlBoolean) a).getValue()?expr(#right_and):new
VinlBoolean(false);
                  }
            }
      | #(ASSIGN a=expr b=expr)
            {
                  if(proceed)
                  {
                        if(curr_table.get(a.getName())==null)
                        {
                              throw new RecognitionException("ERROR: Could not find
variable: "+a.getName());
                        } else {
                              if(a instanceof VinlBoolean && b instanceof
VinlBoolean)
                              {
                                    ((VinlBoolean) a).setValue(((VinlBoolean)
b).getValue());
                              }
                              else if (a instanceof VinlDecimal && b instanceof
VinlDecimal)
                              {
                                    ((VinlDecimal) a).setValue(((VinlDecimal)
b).getValue());
                              }
                              else
                              {
                                    log("Type of variable:
"+((a==null)?"null":a.getType())+ ", type of rvalue:
"+((b==null)?"null":b.getType()), INFO);
                                    throw new RecognitionException("ERROR: Type
mismatched assignment.");
                              }
                        }
                  }
            }
      | "true"                      { if(proceed) r=new VinlBoolean(true); }
      | "false"                     { if(proceed) r=new VinlBoolean(false); }
      | "void"                      { if(proceed) r = new VinlVoid(); }
      | "clear"                     { if(proceed) gp.clearScreen(); }
      | "cl"                              { if(proceed) gp.clearScreen(); }
      | "home"                      { if(proceed) gp.homeTurtle(); }
      | "penup"                     { if(proceed) turtle.setPenDown(false); }
      | "pu"                              { if(proceed) turtle.setPenDown(false); }
      | "pendown"                   { if(proceed) turtle.setPenDown(true); }
      | "pd"                              { if(proceed) turtle.setPenDown(true); }
      | "break"
            {
                  if(proceed)
                  {
                        if(inloop)
                        {
                              proceed = false;
                              breaked = true;
                              log("break further stmts", DEBUG);
                        }
                        else
                              throw new RecognitionException("ERROR: Illegal break
out side of any while/for loop");
                  }
            }
      | "continue"
            {
                  if(proceed)
                  {
                        if(inloop)
                        {
                              proceed = false;
                              continued = true;
                              log("Stop further stmts", DEBUG);
                        }
                        else
                              throw new RecognitionException("ERROR: Illegal break
out side of any while/for loop");
                  }
            }
      | #("var" type:.  id:ID)
            {
                  if(proceed)
                  {
                        if(isKeyword(id.getText()))
                              throw new RecognitionException("ERROR: Illegal use of
keyword *"+id.getText()+"*");

                        if(type.getText().equals("decimal"))
                        {
                              r = new VinlDecimal(id.getText());
                              curr_table.put(id.getText(), r);
                        }
                        else if (type.getText().equals("boolean"))
                        {
                              r = new VinlBoolean(id.getText());
                              curr_table.put(id.getText(), r);
                        }
                        else
                        {
                              throw new RecognitionException("ERROR: Unrecognized
type specified.");
                        }
                  }
            }
      | var_id:ID
            {
                  if(proceed)
                  {
                        if(curr_table.get(var_id.getText())!=null)
                              r=curr_table.get(var_id.getText());
                        else
                              throw new RecognitionException("ERROR: Undeclared
variable: "+var_id.getText());
                  }
            }
      | decimal:DECIMAL       { if(proceed) r=new
VinlDecimal(Double.parseDouble(decimal.getText())); }
      | #("echo" a=expr)
            {
                  if(proceed)
                  {
                        String value="";
                        if(a instanceof VinlBoolean)
                              value += ((VinlBoolean)a).getValue();
                        else if(a instanceof VinlDecimal)
                              value += ((VinlDecimal)a).getValue();
                        else if(a instanceof VinlFunction)
                              value += "a function";
                        else
                              value="not valid";
                        log("The rvalue is: "+value, curr_level+1); //always print
                  }
            }
      | #("forward" a=expr)
            {
                  if(proceed)
                  {
                        log("Going forward: "+((VinlDecimal)a).getValue(),
DEBUG_FINE);
                        gp.forward(((VinlDecimal)a).getValue());
                  }
            }
      | #("fw" a=expr)
```

```
            {
                  if(proceed)
                  {
                        log("Going forward: "+((VinlDecimal)a).getValue(),
DEBUG_FINE);
                        gp.forward(((VinlDecimal)a).getValue());
                  }
            }
      | #("backward" a=expr)
            {
                  if(proceed)
                  {
                        log("Going backward: "+((VinlDecimal)a).getValue(),
DEBUG_FINE);
                        gp.backward(((VinlDecimal)a).getValue());
                  }
            }
      | #("bw" a=expr)
            {
                  if(proceed)
                  {
                        log("Going backward: "+((VinlDecimal)a).getValue(),
DEBUG_FINE);
                        gp.backward(((VinlDecimal)a).getValue());
                  }
            }
      | #("left" a=expr)
            {
                  if(proceed)
                  {
                        log("Turn left: "+((VinlDecimal)a).getValue(), DEBUG_FINE);
                        gp.left(((VinlDecimal)a).getValue());
                  }
            }
      | #("lf" a=expr)
            {
                  if(proceed)
                  {
                        log("Turn left: "+((VinlDecimal)a).getValue(), DEBUG_FINE);
                        gp.left(((VinlDecimal)a).getValue());
                  }
            }
      | #("right" a=expr)
            {
                  if(proceed)
                  {
                       log("Turn right: "+((VinlDecimal)a).getValue(), DEBUG_FINE);
                        gp.right(((VinlDecimal)a).getValue());
                  }
            }
      | #("rt" a=expr)
            {
                  if(proceed)
                  {
                       log("Turn right: "+((VinlDecimal)a).getValue(), DEBUG_FINE);
                        gp.right(((VinlDecimal)a).getValue());
                  }
            }
      | #("if" a=expr ifbody:. (elsebody:.)?)
            {
                  if(proceed)
                  {
                        if(a instanceof VinlBoolean)
                        {
                              if(((VinlBoolean)a).getValue()) {
                                    //do if part
                                    r = expr(#ifbody);
                              }
                              else
                              {
                                    //do else part
                                    r = expr(#elsebody);
                              }
                        }
                        else
                        {
                              throw new RecognitionException("ERROR: Only accept
boolean express as if condition");
                        }
                  }
            }
      | #("return"
            (
                  a=expr { if(proceed) { ret = a; r = a; } }
            )?
         )
            {
                  /* if returned, it's same as breaked in the sense no stmt should
run anymore */
                  proceed = false;
                  returned = true;
            }
      | #("while" while_condition:. while_body:.)
            {
                  if(proceed)
                  {
                        inloop = true;
                        VinlType while_cond = expr(#while_condition);
                        if(!(while_cond instanceof VinlBoolean))
                              throw new RecognitionException("ERROR: Condition in
for/while loop must be boolean type");
                        while(((VinlBoolean)while_cond).getValue())
                        {
                              r = expr(#while_body);
                              if(breaked)
                              {
                                    //reset break for future statements
                                    proceed = true;
                                    breaked = false;
                                    break;
                              }
                              else if(continued)
                              {
                                    proceed=true;
                                    continued=false;
                              }
                              while_cond=expr(#while_condition);
                        }
                        inloop = false;
                  }
            }
      | #("for" a=expr for_condition:.  post_action:. for_body:.)
            {
                  if(proceed)
                  {
                        inloop = true;
                        VinlType for_cond = expr(#for_condition);
                        if(!(for_cond instanceof VinlBoolean))
                              throw new RecognitionException("ERROR: Condition in
for/while loop must be boolean type");
                        while(((VinlBoolean)for_cond).getValue())
                        {
                              r = expr(#for_body);
                              if(breaked)
```

```
                                {
                                      //reset break for future statements
                                      proceed = true;
                                      breaked = false;
                                      break;
                                }
                                else if(continued)
                                {
                                      proceed=true;
                                      continued=false;
                                }
                                r = expr(#post_action);
                                for_cond=expr(#for_condition);
                          }
                          inloop = false;
                    }
              }
        | "status"
              {
                    /* printout internal states*/
                    log(String.format("Control flows: proceed=(%s), inloop=(%s), 
breaked=(%s), continued=(%s), returned=(%s)", proceed, inloop, breaked, continued, 
returned), curr_level+1);
                    log(String.format("turtle coordinates: x=%f y=%f angle(in 
degree)=%f", turtle.getX(), turtle.getY(), turtle.getAngle()), curr_level+1);
              }
        | #(FUNC_CALL a=expr expr_list=mexpr)
              {
                    if(proceed) {
                          if(!(a instanceof VinlFunction))
                                throw new RecognitionException("ERROR: 
"+a.getName()+" is not a function.");

                          VinlFunction func = (VinlFunction) a;
                          /* every function starts its own scope with current scope 
as it's parent scope */
                          log("Current symbol table: " + curr_table, DEBUG_FINE);
                          func.getSymTable().setParent(curr_table);
                          curr_table = func.getSymTable();
                          log("New symbol table, going to functions: 
"+func.getName()+"'s scope: " + curr_table, DEBUG_FINE);
                          /* evaluate arguments and put them in symbol table */
                          String[] params = func.getParams();
                          int j = 0; //expr_list index
                          for(int i = 0; i<params.length; i++) {
                                if(params[i].equals("void")) {
                                      if(expr_list[j]==null || !(expr_list[j] 
instanceof VinlVoid))
                                            throw new RecognitionException("ERROR: 
Wrong argument passed to function at index: "+j);
                                } else if (params[i].equals("decimal") && 
params[++i] != null) {
                                      if(expr_list[j]==null || !(expr_list[j] 
instanceof VinlDecimal))
                                            throw new RecognitionException("ERROR: 
Wrong argument passed to function at index: "+j);
                                      log("putting "+params[i]+" in sym 
table",DEBUG_FINE);
                                      ((VinlDecimal)expr_list[j]).setName(params[i]);
                                      curr_table.put(params[i], expr_list[j++]);
                                } else if (params[i].equals("boolean") && 
params[++i] != null) {
                                      if(expr_list[j]==null || !(expr_list[j] 
instanceof VinlBoolean))
                                            throw new RecognitionException("ERROR: 
Wrong argument passed to function at index: "+j);
                                      log("putting "+params[i]+" in sym
table",DEBUG_FINE);
                                      ((VinlBoolean)expr_list[j]).setName(params[i]);
                                      curr_table.put(params[i], expr_list[j++]);
                                }
                          }
                          /* invoke this function */
                          r=expr(func.getBody());
                          if(returned) {
                                proceed = true;
                                returned = false;
                                if(ret != null) {
                                      r = ret;
                                }
                                ret = null;
                          }
                          /* coming out of the function so resume parent scope now */
                          log("Out of function: "+func.getName()+"'s scope: " +
curr_table, DEBUG_FINE);
                          curr_table = curr_table.getParent();
                          log("Back to "+func.getName()+"'s parent scope: " +
curr_table, DEBUG_FINE);
                          /* destory the function symbol table */
                          func.setSymTable(new VinlSymbolTable());
                    }
              }
        | #("funct" func_name:ID func_params=vlist func_body:.)
              {
                    r = new VinlFunction(func_name.getText(), func_params,
#func_body);
                    curr_table.put(func_name.getText(), r);
              }
        ;
mexpr returns [ VinlType[] mret ]
{
        ArrayList<VinlType> list = new ArrayList();
        mret = null;
        VinlType a;
}
        : #(EXPR_LIST     (a=expr {list.add(a);})+)
              {
                    mret = new VinlType[list.size()];
                    mret = list.toArray(mret);
              }
        ;

vlist returns [ String[] ret ]
{
        /* type and variable name alternating unless it's void*/
        ArrayList<String> list=new ArrayList();
        ret = null;
}
        : #(PARAMS_LIST   (s:.  {list.add(s.getText());})+)
              {
                    for(int i = 0; i < list.size(); i++){
                          log(list.get(i), DEBUG_FINE);
                    }
                    ret = new String[list.size()];
                    ret = list.toArray(ret);
              }
        ;
```

VinlInterpreterGui.java:

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

import antlr.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.io.*;
import java.util.ArrayList;

/**
 * This class is the main GUI class
 * @author ycao
 */
public class VinlInterpreterGui extends JFrame implements KeyListener {

      private static VinlGraphicPanel gp;
      private static JTextField editor;
      private ArrayList<String> cmdlist = new ArrayList();
      private int cmd_index = 0;

      public VinlInterpreterGui() {
            gp = new VinlGraphicPanel();
            gp.setPreferredSize(new Dimension(600, 600));
            editor = new JTextField();
            editor.setPreferredSize(new Dimension(500, 30));
            editor.addKeyListener(this);

            this.getContentPane().setLayout(new java.awt.BorderLayout(0, 10));

      this.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
            this.getContentPane().add(gp, java.awt.BorderLayout.CENTER);
            this.getContentPane().add(editor, java.awt.BorderLayout.PAGE_END);
            this.setTitle("Interperter for VINL -- VINL is Not Logo");

            pack();
            gp.init(gp.getSize());
      }

      public static VinlGraphicPanel getGraphicPanel() {
            return gp;
      }

      private void drawCircles(int num) {
            System.out.println("Drawing circles starting: " + new
java.util.Date().toString());
            int turn = 360 / num;
            for (int i = 0; i < num; i++) {
                  for (int j = 0; j < 360; j++) {
                        gp.forward(1);
                        gp.right(1);
                  }
                  gp.left(turn);
            }
            gp.repaint();
            System.out.println("Drawing circles end: " + new
java.util.Date().toString());
      }

      private void drawSquares(int iteration) {
            for (int i = 0; i < iteration; i++) {
                  gp.forward(i + 2);
                  gp.right(89);
            }
      }
```

```
public void keyPressed(KeyEvent e) {
}


public void keyTyped(KeyEvent e) {
}
/*
public void keyTyped(KeyEvent e) {
char c = e.getKeyChar();
switch (c) {
case 'f':
gp.forward(50);
break;
case 'b':
gp.back(50);
break;
case 'c':
gp.clearScreen();
break;
case 'q':
System.exit(0);
break;
case 'd':
drawSquares(40);
break;
case 'v':
drawCircles(24);
break;
case 'p':
gp.repaint();
break;
case 'r':
gp.right(90);
break;
case 'l':
gp.left(90);
break;
default:
break;
}
e.consume();
repaint();
}
 */

public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                  new VinlInterpreterGui().setVisible(true);
            }
      });
}

InputStream input;
VinlLexer lexer;
VinlParser parser;
VinlAstWalker walker;

public InputStream strToInputStream(String cmd) {
      InputStream in = null;
      if(cmd==null) return null;
      try {
            in = new ByteArrayInputStream(cmd.getBytes());
      } catch (Exception e) {}
      return in;
}
```

```
	public void loadProgram(String filename) throws Exception {
		File src = new File(filename);
		FileInputStream fis = new FileInputStream(src);
		lexer = new VinlLexer(fis);
		parser = new VinlParser(lexer);
		parser.program();
		walker = new VinlAstWalker();
		CommonAST tree = (CommonAST) parser.getAST();
		//System.out.println(tree.toStringList());
		VinlType r = walker.expr(tree);
	}

	public void keyReleased(KeyEvent e) {
		switch(e.getKeyCode()) {
			case KeyEvent.VK_ENTER:
				try {
					String cmd = editor.getText();
					cmdlist.add(cmd);
					cmd_index = cmdlist.size();
					if(cmd.startsWith("load ")) {
						loadProgram(cmd.substring(5).trim());
					} else {
						input = strToInputStream(cmd.trim());
						lexer = new VinlLexer(input);
						parser = new VinlParser(lexer);
						parser.cmd();
						walker = new VinlAstWalker();
						CommonAST tree = (CommonAST) parser.getAST();
						//System.out.println(tree.toStringList());
						VinlType r = walker.expr(tree);
					}
				} catch (Exception ex) {
					System.out.println(ex);
					VinlAstWalker.log(ex.getMessage(),
VinlAstWalker.DEBUG_FINE);
					//ex.printStackTrace();
				}
				editor.setText("");
				break;
			case KeyEvent.VK_UP:
				if(cmd_index-1>=0) {
					editor.setText(cmdlist.get(--cmd_index));
				}
				break;
			case KeyEvent.VK_DOWN:
				if(cmd_index+1<cmdlist.size()) {
					editor.setText(cmdlist.get(++cmd_index));
				}
				break;
			default:
				break;
		}
		e.consume();
		repaint();
	}
}
```

VinlGraphicPanel.java:

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * This is the panel where all the drawings are actually performed
 * @author ycao
 */
public class VinlGraphicPanel extends JPanel {

      private Image buf_img;

      private Graphics2D buf_g;

      private Dimension d;

      private VinlTurtle turtle;

      private Color penColor;

      /**
       * Default Constructor
       */
      public VinlGraphicPanel() {
      }

      public void init(Dimension d) {
            this.d = d;
            this.setSize(d);
            buf_img = this.createImage(d.width, d.height);
            buf_g = (Graphics2D)buf_img.getGraphics();
            penColor = Color.BLACK;
            buf_g.setColor(penColor);
            this.setBackground(Color.WHITE);
            turtle = new VinlTurtle(d.width/2, d.height/2, 90);
      }

      /**
       * Draw a line length of r from the current turtle position
       * in the turtle facing direction
       * @param r -- distance from current point
       */
      public void forward(double r) {
            double currX = turtle.getX();
            double currY = turtle.getY();
            double theta = turtle.getAngle() * Math.PI / 180;
            double x = currX + (Math.cos(theta) * r);
            //because y coordinate in Java goes down, do minus instead of plus here
            double y = currY - (Math.sin(theta) * r);
            if (turtle.isPenDown()) {
                  //need to sacrafice some accuracy
                  buf_g.drawLine((int)currX, (int)currY, (int)x, (int)y);
                  VinlAstWalker.log(String.format("coordinates: %f %f %f %f. %n",
currX, currY, x, y), VinlAstWalker.DEBUG_FINE);
            }
            turtle.setX(x);
            turtle.setY(y);
      }

      /**
       * Draw a line length of r from the current turtle position
       * in the direction opposite of the turtle facing direction
       * @param r -- distance from current point
```

```
	 */
	public void backward(double r) {
		forward((-1)*r);
	}

	/**
	 * turn the turtle x degrees to the right
	 * @param x -- in degrees
	 */
	public void right(double x) {
		turtle.setAngle(turtle.getAngle()-x);
	}

	/**
	 * turn the turtle x degrees to the right
	 * @param x -- in degrees
	 */
	public void left(double x) {
		turtle.setAngle(turtle.getAngle()+x);
	}

	/**
	 * Clear the entire screen
	 */
	public void clearScreen() {
		//empty the buffer and start a new
		this.init(d);
	}

	public void homeTurtle() {
		turtle = new VinlTurtle(d.width/2, d.height/2, 90);
	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		g.drawImage(buf_img, 0, 0, this);
	}

	public void drawTurtle() {
		//create a triangle based on the current x, y and angle
		Polygon triangle = new Polygon();
		double theta = turtle.getAngle() * Math.PI / 180;
		double x = turtle.getX() + (Math.cos(theta) * 10);
		double y = turtle.getY() - (Math.sin(theta) * 10);
		triangle.addPoint((int)x, (int)y);
		triangle.addPoint((int)(turtle.getX()+(Math.sin(theta)*5)),
(int)(turtle.getY()+(Math.cos(theta)*5)) );
		triangle.addPoint((int)(turtle.getX()-(Math.sin(theta)*5)),
(int)(turtle.getY()-(Math.cos(theta)*5)) );
		buf_g.draw(triangle);
		turtle.setTriangle(triangle);
	}

	public void eraseTurtle() {
		Polygon triangle = turtle.getTriangle();
		if(triangle != null) {
			buf_g.setColor(buf_g.getBackground());
			buf_g.draw(triangle);
			buf_g.setColor(penColor);
		}
	}

	public VinlTurtle getTurtle() {
		return this.turtle;
	}

}
```

VinlTurtle.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author ycao
 */
public class VinlTurtle {

      //x coordinate of turtle
      private double x = 0;

      //y coordinate of turtle
      private double y = 0;

      //turtle facing angle in degree, 90 = facing north
      private double angle = 90;

      //if pen is down, will leave footprints
      private boolean penDown = true;

      //triangle graphical representation of a turtle
      private java.awt.Polygon triangle;

      public VinlTurtle(double x, double y, int angle) {
            this.x = x;
            this.y = y;
            this.angle = angle;
      }

      public double getX() {
            return x;
      }

      public void setX(double x) {
            this.x = x;
      }

      public double getY() {
            return y;
      }

      public void setY(double y) {
            this.y = y;
      }

      public double getAngle() {
            return angle;
      }

      public void setAngle(double angle) {
            this.angle = angle;
      }

      public boolean isPenDown() {
            return penDown;
      }

      public void setPenDown(boolean isPenDown) {
            this.penDown = isPenDown;
      }

      public void setTriangle(java.awt.Polygon triangle) {
            this.triangle = triangle;
      }

      public java.awt.Polygon getTriangle() {
            return triangle;
      }

}
```

VinlType.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



/**
 *
 * @author ycao
 */
public interface VinlType {

      //real types
      final int DECIMAL =  0;
      final int BOOLEAN =  1;
      final int VOID          = -1;
      final int FUNCTION      =  2;

      //types
      int getType();

      String getName();

      VinlType eval(int op, VinlType op1, VinlType op2) throws
antlr.RecognitionException;
}
```

Vinl_Boolean.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author ycao
 */
public class VinlBoolean implements VinlType {

      private boolean value;
      private String name=null;

      public VinlBoolean() {
            value = false;
      }

      //This is useful for tmp variables that do not have name
	public VinlBoolean(boolean value) {
		this.value=value;
	}

	public VinlBoolean(String name) {
		this.name=name;
	}

	public VinlBoolean(String name, boolean value) {
		this.name = name;
		this.value = value;
	}

	public int getType() {
		return VinlType.BOOLEAN;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return this.name;
	}

	public void setValue(boolean value) {
		this.value=value;
	}

	public boolean getValue() {
		return this.value;
	}

	public VinlType eval(int op, VinlType operand1, VinlType operand2) throws
antlr.RecognitionException {
		if (!checkValid(operand1)) {
			throw new antlr.RecognitionException("First operand is not
decimal or it has no value");
		}
		if (!checkValid(operand2)) {
			throw new antlr.RecognitionException("Second operand is not
decimal or it has no value");
		}
		VinlBoolean op1 = (VinlBoolean) operand1;
		VinlBoolean op2 = (VinlBoolean) operand2;

		/* boolean only supports logical operaionts */
		switch (op) {
			case VinlOperation.OR:
				return op1.or(op2);
			case VinlOperation.AND:
				return op1.and(op2);
			case VinlOperation.EQ:
				return op1.eq(op2);
			case VinlOperation.NEQ:
				return op1.neq(op2);
			default:
				throw new antlr.RecognitionException("Decimal does not
support this operation");
		}
	}

	public boolean checkValid(VinlType operand) {
		return (operand.getType() == VinlType.BOOLEAN);
	}

	//logical operations
	public VinlBoolean or(VinlBoolean b) {
		return new VinlBoolean(this.value||b.getValue());
	}

	public VinlBoolean and(VinlBoolean b) {
		return new VinlBoolean(this.value&&b.getValue());
	}

	public VinlBoolean eq(VinlBoolean b) {
		return new VinlBoolean(this.value==b.getValue());
	}

	public VinlBoolean neq(VinlBoolean b) {
		return new VinlBoolean(this.value!=b.getValue());
	}
}
```

VinlDecimal.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



import antlr.RecognitionException;

/**
 *
 * @author ycao
 */
public class VinlDecimal implements VinlType {

	private double value;
	private String name;

	public VinlDecimal() {
		//auto initialize all decimal to 0 if no value is specified
		value = 0;
	}

	//use for tmp variables
	public VinlDecimal(double value) {
		this.value = value;
	}

	public VinlDecimal(String name) {
		this.name = name;
	}

	public VinlDecimal(String name, double value) {
		this.name = name;
		this.value = value;
	}


	public int getType() {
		return VinlType.DECIMAL;
	}

	public void setValue(double v) {
		this.value = v;
	}
```

```
      public double getValue() {
            return this.value;
      }

      public void setName(String name) {
            this.name=name;
      }

      public String getName() {
            return this.name;
      }

      public VinlType eval(int op, VinlType operand1, VinlType operand2) throws
RecognitionException {

            if(!checkValid(operand1)) {
                  throw new RecognitionException("First operand is not decimal or
it has no value");
            }
            if(!checkValid(operand2)) {
                  throw new RecognitionException("Second operand is not decimal or
it has no value");
            }
            VinlDecimal op1 = (VinlDecimal)operand1;
            VinlDecimal op2 = (VinlDecimal)operand2;
            /* decimal supports all arithmetical and relational operaionts */
            switch(op) {
                  case VinlOperation.PLUS:
                        return op1.plus(op2);
                  case VinlOperation.MINUS:
                        return op1.minus(op2);
                  case VinlOperation.MULTI:
                        return op1.multi(op2);
                  case VinlOperation.DIV:
                        return op1.div(op2);
                  case VinlOperation.MOD:
                        return op1.mod(op2);
                  case VinlOperation.EQ:
                        return op1.eq(op2);
                  case VinlOperation.NEQ:
                        return op1.neq(op2);
                  case VinlOperation.LT:
                        return op1.lt(op2);
                  case VinlOperation.GT:
                        return op1.gt(op2);
                  case VinlOperation.LE:
                        return op1.le(op2);
                  case VinlOperation.GE:
                        return op1.ge(op2);
                  default:
                        throw new antlr.RecognitionException("Decimal does not
support this operation");
            }
      }

      public boolean checkValid(VinlType operand) {
            return (operand.getType()==VinlType.DECIMAL);
      }

      //arithmetic oeprations
      public VinlDecimal plus(VinlDecimal b) {
            return new VinlDecimal(value+b.getValue());
      }

      public VinlDecimal minus(VinlDecimal b) {
            return new VinlDecimal(value-b.getValue());
      }
```

```
	public VinlDecimal multi(VinlDecimal b) {
		return new VinlDecimal(value*b.getValue());
	}

	public VinlDecimal div(VinlDecimal b) {
		return new VinlDecimal(value/b.getValue());
	}

	public VinlDecimal mod(VinlDecimal b) {
		return new VinlDecimal(value%b.getValue());
	}

	//relational operations
	public VinlBoolean eq(VinlDecimal b) {
		return new VinlBoolean(this.value == b.getValue());
	}

	public VinlBoolean neq(VinlDecimal b) {
		return new VinlBoolean(this.value != b.getValue());
	}

	public VinlBoolean lt(VinlDecimal b) {
		return new VinlBoolean(this.value < b.getValue());
	}

	public VinlBoolean le(VinlDecimal b) {
		return new VinlBoolean(this.value <= b.getValue());
	}

	public VinlBoolean gt(VinlDecimal b) {
		return new VinlBoolean(this.value > b.getValue());
	}

	public VinlBoolean ge(VinlDecimal b) {
		return new VinlBoolean(this.value >= b.getValue());
	}
}
```

VinlFunction.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



import antlr.RecognitionException;
import antlr.collections.AST;

/**
 *
 * @author ycao
 */
public class VinlFunction implements VinlType {

	private String name;
	private String[] params;
	private AST body;
	private VinlSymbolTable sym_table = new VinlSymbolTable();

	public VinlFunction(String name, String[] params, AST body) {
		this.name = name;
		this.params = params;
```

```
            this.body = body;
      }

      public VinlType eval(int op, VinlType op1, VinlType op2) throws
RecognitionException {
            throw new UnsupportedOperationException("Not supported yet.");
      }

      public int getType() {
            return VinlType.FUNCTION;
      }

      public AST getBody() {
            return body;
      }

      public VinlSymbolTable getSymTable() {
            return sym_table;
      }

      public void setSymTable(VinlSymbolTable sym_table) {
            this.sym_table = sym_table;
      }

      public void setBody(AST body) {
            this.body = body;
      }

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
      }

      public String[] getParams() {
            return params;
      }

      public void setParams(String[] params) {
            this.params = params;
      }

}
```

VinlVoid.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



/**
 *
 * @author ycao
 */
public class VinlVoid implements VinlType{

      public int getType() {
            return VinlType.VOID;
      }
      public VinlType eval(int op, VinlType op1, VinlType op2) throws
antlr.RecognitionException {
            throw new antlr.RecognitionException("VOID does not support any
operations.");
      }

      public String getName() {
            return "void";
      }

}
```

VinlOperation.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */



/**
 *
 * @author ycao
 */
public class VinlOperation {

      //Arithmetic
      public static  final int ARITH_BASE = 0;
      public static  final int PLUS       = ARITH_BASE+0;
      public static  final int MINUS            = ARITH_BASE+1;
      public static  final int MULTI            = ARITH_BASE+2;
      public static  final int DIV        = ARITH_BASE+3;
      public static  final int MOD        = ARITH_BASE+4;

      //Relational
      public static  final int RELATION_BASE    = 20;
      public static  final int LT         = RELATION_BASE+1;
      public static  final int GT         = RELATION_BASE+2;
      public static  final int LE         = RELATION_BASE+3;
      public static  final int GE         = RELATION_BASE+4;
      public static  final int EQ         = RELATION_BASE+5;
      public static  final int NEQ        = RELATION_BASE+6;

      //Logical
      public static  final int LOGICAL_BASE     = 40;
      public static  final int OR         = LOGICAL_BASE+0;
      public static  final int AND        = LOGICAL_BASE+1;

}
```

VinlSymbolTable.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

import antlr.RecognitionException;
import java.util.Hashtable;

/**
```

```
 * Vinl uses static scoping
 * @author ycao
 */
public class VinlSymbolTable {

      private Hashtable<String, VinlType> symbols = null;
      private VinlSymbolTable parent = null;

      public VinlSymbolTable() {
            symbols = new Hashtable();
      }

      public VinlSymbolTable(VinlSymbolTable parent) {
            symbols = new Hashtable();
            this.parent = parent;
      }

      public VinlSymbolTable getParent() {
            return parent;
      }

      public void setParent(VinlSymbolTable parent) {
            this.parent = parent;
      }

      public Hashtable getSymbols() {
            return this.symbols;
      }

      public void setSymbols(Hashtable symbols) {
            this.symbols = symbols;
      }

      public void put(String name, VinlType obj, boolean enable_redef) throws
RecognitionException {
            //don't really expect this since parser should catch this error,
            //but better safe than sorry
            int type = obj.getType();
            if(type != VinlType.BOOLEAN && type != VinlType.VOID &&
                        type != VinlType.DECIMAL && type != VinlType.FUNCTION) {
                  throw new RecognitionException("ERROR: Undefined type.");
            }
            //Do not allow declare the same varible name twice unless override by
enable_redef
            if(symbols.containsKey(name) && !enable_redef) {
                  throw new RecognitionException("ERROR: Redefinition of variable:
"+name);
            }
            symbols.put(name, obj);
      }

      public void put(String name, VinlType obj) throws RecognitionException { //
this is used in tail recursion
            put(name, obj, false);
      }

      public VinlType get(String name) {
            VinlType ret = null;
            if(symbols.get(name) != null) {
                  ret = symbols.get(name);
            } else {
                  //check in parents' table
                  VinlSymbolTable par = this.getParent();
                  while(par != null) {
                        if(par.getSymbols().get(name)!=null) {
                              ret = (VinlType)(par.getSymbols().get(name));
                              break;
                        } else {
                              par = par.getParent();
                        }
                  }
            }
            return ret;
      }
}
```

AST Testing Case:

```
//this is a test for comment

//expressions statments, everything needs to end with a semi ;

//constants
1;
1.5;
true;
false;

//identifiers
x;
x1;
x1x;
x1x_;

//function calls
x(void);
x(x, x);
x(x, y);
x(x, void, y);
drawcircle(a, b);

//operative expressions
1*2;
x*2;
x*x*2;
1%2;
x%2;
x%x%2;
```

```
1/2;
x/2;
x/x/2;
1+2;
x+2;
x+x+2;
1-2;
x-2;
x-x-2;
x<1;
x<y;
x=1;
x=y;
x>1;
x>y;
x>=1;
x>=y;
x<=1;
x<=y;
x!=1;
x!=y;
x||true;
x||y;
x&&false;
x&&y;

//assignment expressions
x<-1;
x<-true;
x<-y;
x<-5%8;
x<-y(void);
x<-y(x,y);

//keyword statements
fw 10;
forward 10;
fw x;
forward x;
```

```
bw 10;
backward 10;
bw x;
backward x;
rt 10;
right 10;
rt x;
right x;
lf 10;
left 10;
lf x;
left x;
penup;
pu;
pendown;
pd;
clear;
cl;
return x;
return 10;
return 10+x;
echo x;
echo 10;
echo 10+x;
break;
continue;
status;

//variable declaration statement
var decimal x;
var decimal x<-0+10;
var boolean y;
var boolean y<-false;

//function declaration statement
//mainly test param list declaration since we take body as a block
funct x(void) {
      fw 10;
}
```

```
funct drawcircle(decimal x, boolean y) {
      for ( var decimal x <- 0; x <= 36; x <- x+1) {
            forward 1;
            left x;
      }
}



//Control flow statments
for ( var decimal x <- 0; x <= 36; x <- x+1) {
      forward x+10;
      left x;
}

for (void; x<=36; void) { //condition can't be void
      forward x+10;
      left x;
}

while(x < 4) {
      forward 100;
      right 90;
      x<-x+1;
}

if(x=2) break;

if(x<100) {} else { left 90; }

if(true) {
      forward 1;
}



//tail recursion
//this passed the parser but does not work in walker

funct drawSpiral(decimal step_size) {
      if(step_size = 100) {
            return; //base case
      }
      fw step_size;
      rt 1;
        drawSpiral(step_size+1);
}
```

Sample Programs:

drawcircles.vs

```
//This program shows ability of function definition, functions calls and for/while
loops operation
//on top of regular variable declaration, assignment, and VINL's drawing abilities.

funct circle(decimal stepsize) {
      var decimal i <- 0;
      while( i < 360 ) {
            fw stepsize;
            rt 1;
            i<-i+1;
      }
}

funct drawMultiCircles(decimal num, decimal stepsize) {
      //only to show how function return a value
      var decimal turn_deg <- calcDeg(num);
      for(var decimal i<-0; i<num; i<-i+1) {
            circle(stepsize);
            left turn_deg;
      }
}

funct calcDeg(decimal num) {
      return (360 / num);
}
drawMultiCircles(24, 2);
```

drawflower.vs:

```
//copied from Bob DuCharme's logo tutorial


funct drawFlower(void) {
      var decimal j; //can't have this in the for loop, since for {} does not
define new scope
      var decimal i;
      //flower
      for(i<-0; i<24; i<-i+1) {
            for(j<-0; j<3; j<-j+1) {
                  fw 35;
                  rt 120;
            }
            rt 15;
      }
      //stem
      home;
      bw 140;

      //left leaf
      left 45;
      fw 70;
      lf 10;
      bw 60;

      //right leaf
      rt 110;
      fw 60;
      left 10;
      bw 71;
      left 45;

      //one more line for stem
      fw 140;
}

drawFlower(void);
```

drawolympics.vs:

```
//different approach to draw a circle centered at current turtle location
funct rcircle(decimal radius) {
      var decimal stepsize <- radius*2*3.1415926/360;
      pu; left 90; fw radius; right 90; pd;
      for(var decimal i; i<360; i<-i+1) {
            fw stepsize;
            rt 1;
      }
      pu; right 90; fw radius; left 90; pd;
}

funct drawOlympicFlag(void) {
      pu; rt 90; bw 110; pd;
      rcircle(50);
      pu; fw 110; pd;
      rcircle(50);
      pu; fw 110; pd;
      rcircle(50);
      pu; rt 90; fw 70; rt 90; fw 55; pd;
      rcircle(50);
      pu; fw 110; pd;
      rcircle(50);
}

drawOlympicFlag(void);
```

drawsnow.vs:

```
funct snowflakes(decimal len) {
      var decimal j;
      for(var decimal i<-0; i<5; i<-i+1) {
            for(j<-0; j<8; j<-j+1) {
                  fw len;
                  pu;
                  bw len;
                  rt 45;
                  pd;
            }
            pu;
            fw 200;
            rt 136;
            fw 242;
            pd;
      }
}

snowflakes(30)
```

drawspiral.vs:

```
funct drawSpiral(decimal start_step_size, decimal steps) {
      while(steps > 0) {
            fw start_step_size;
            rt 15;
            steps <- steps - 1;
            start_step_size <- start_step_size+1;
      }
}

drawSpiral(1, 50)
```

drawstar.vs:

```
funct drawStar(decimal len) {
      //star should be centered on screen
      pu;
      fw len/3;
      rt 90;
      bw len/2;
      pendown;

      //the tip angle should be 36 degree
      //(180 - 540/5) / 2 = 36
      for (var decimal i<-0; i<5; i<-i+1) {
            fw len;
            right 180-(180-540/5)/2;
      }
}

drawStar(300)
```