BATS

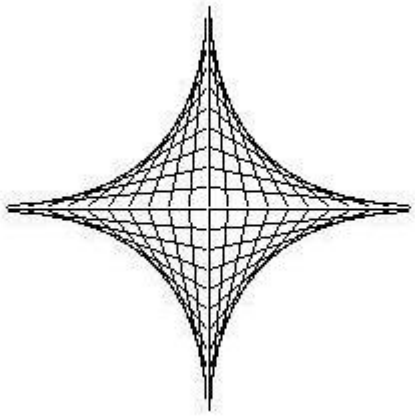
Behrooz Badii (Team Leader) Aleksandr Borovinskiy
Tanya Shtemberg Sui Sum Wong

December 18, 2003

Table of Contents

BATS White Page

Background:



That's an example of something we want to make with BATS. Using the JAVA graphics library through BATS, we can and will be able to create something like the picture above. BATS will be an enjoyable language, since its output is enjoyable, satisfying, and visually gratifying. This creates a language that people can use and that people want to use due to its simplicity and high productivity. What's better than seeing your code come out as a beautiful graphic image or geometrical figure?

In case of curiosity, the title of BATS came from taking the first letter of the first name of each person in the group (**B**ehrooz, **A**lex, **T**anya, **S**ui).

Introduction:

BATS is a geometric figure drawing language using JAVA code to draw simple to complex shapes and figures on a canvas. BATS is a reliable, understandable, efficient, usable, testable, portable, interpreted, high-performance, robust, internet compatible, and visually beautiful programming language.

Reliable:

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."
(C.A.R. Hoare)

Since BATS's compiler was based on JAVA, the reliability of the language is relative to that of JAVA. Programmers can still make errors where their own drawings are not correct (for example creating a triangle instead of a rectangle), but due to the simplicity of the language, these errors would occur quite seldom. Also, if the JAVA Virtual Machine does not have deprecated methods in the Graphics section of the JAVA library later on, BATS can stay reliable.

Understandable:

"Increasingly, people seem to misinterpret complexity as sophistication, which is baffling - the incomprehensible should cause suspicion rather than admiration."
(Niklaus Wirth)

Due to the grammar of the language, and the immediate graphic results, the code is understandable. The language's naming structure gives the programmer the freedom to make reasonable names, bringing an understanding to what each line of code, each variable, and each shape is and does.

Efficient:

"A language that doesn't have everything is actually easier to program in than some that do."
(Dennis M. Ritchie)

BATS gives several possibilities to create a shape or line to draw on the screen. It's also simple to use, so a simple-to-use language creates efficiency. Since the compiler of the language is object oriented (the compiler is code in JAVA), the compiler is efficient due to its simplicity to understand. So, a compiler that can be understood or debugged easily will create efficient code.

Usable:

"I consider it the obligation of scientists and intellectuals to ensure that their ideas are made accessible and thus useful to society instead of being mere playthings for specialists."
(Bjarne Stroustrup)

Programmers and users need a very limited background of understanding graphs and points to be able to use BATS. If programmers know how to make complex graphical images using algorithms, then there is no bound to what they can make; it can range from a rudimentary triangle to an intricate DNA structure, depending on the programmer's need. The usability of the BATS language has a direct relation with the knowledge of the user in creating graphical images and complex two-dimensional geometric figures.

Testable:

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
(Brian Kernighan)

BATS can be programmed and debugged for syntax errors. For semantic errors of incorrect drawing, the error can be immediately seen and corrected. If the programmer puts a circle where it shouldn't be, then they can change it very quickly after seeing the incorrect image he or she created.

Portable:

Most papers in computer science describe how their author learned what someone else already knew. (paraphrase)
(Peter Landin)

BATS is portable because it is based on JAVA. Since JAVA is portable, BATS is portable. Furthermore, the graphics portion of the JAVA library has been relatively unchanged throughout different releases and editions of JAVA. This makes the BATS compiler backwards compatible with releases of JAVA.

Interpreted:

"The best writing is rewriting."
(E. B. White)

BATS is interpreted into JAVA, and the interpreted program will be used to make a graphical image on a frame.

High-performance:

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg."
(Bjarne Stroustrup)

Once again, the language BATS is interpreted into JAVA, giving BATS a great asset: high performance. In addition, BATS can be classified as an extension to JAVA, since it facilitates JAVA by providing an easy to use grammar for graphics. Since JAVA is a high-performance language, BATS becomes a high-performance language. JAVA's Just In Time compiler is high-performance, so that is a boon for BATS.

Robust:

"Any sufficiently advanced bug is indistinguishable from a feature."
(Rich Kulawiec)

Due to BATS's simplicity, it is robust. Since there are simple ways to display complex figures, BATS becomes robust. In terms of drawing, it can do many things. It can draw anything that is two-dimensional. The more the programmer puts into his or her code, the more the power of BATS is realized.

Internet Compatible:

"Languages shouldn't hinder progress by outliving their usefulness."
(Alan Kay)

BATS's graphical images will be displayed on a frame. Since a frame is used to show BATS's output, it can be written on an applet to make it Internet compatible. Internet-compatible JAVA interprets the code, and its output is shown an applet, a feature of JAVA seen very often on websites.

Visually Beautiful:

"Do what you think is interesting, do something that you think is fun and worthwhile, because otherwise you won't do it well anyway."
(Brian Kernighan)

The output of BATS is what makes it stand out. Dealing with graphics, one would assume, and assume correctly, that BATS is visually beautiful. Beauty is in the hands of the programmer. If they can program a beautiful image, then it will come into existence.

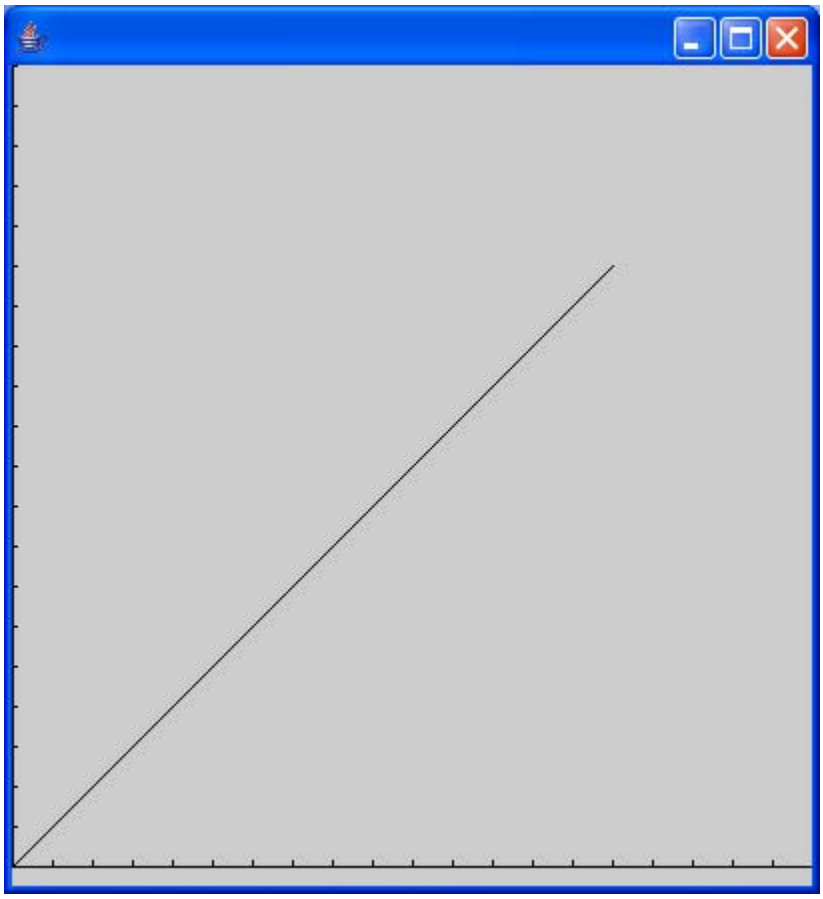
BATS Tutorial

BATS is specialized to create drawings with ease. We firmly believe that the best way to teach someone is by example. With that said, here are a set of examples that range from easy to more difficult.

1. Easy example

Here is the simplest thing one can draw in BATS: a line. Ours will go from the beginning of the x-axis at 45 degrees out to the 300,300 point on the graph.

```
Begin
  Line diagonal;
  Let diagonal <- {^0,0^,^300,300^};
  Draw diagonal;
End
```

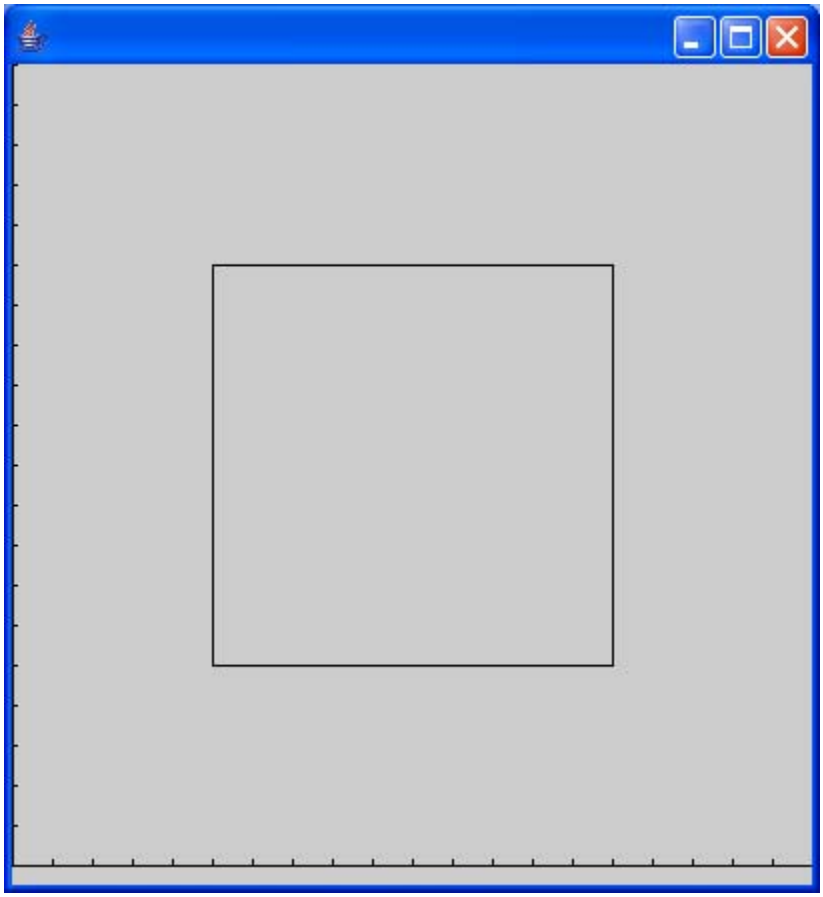


2. Harder Example: Making a polygon

Now let's make a polygon in two ways. In one way, we'll have a polygon with one color for all sides. In another, we'll make it with several colors.

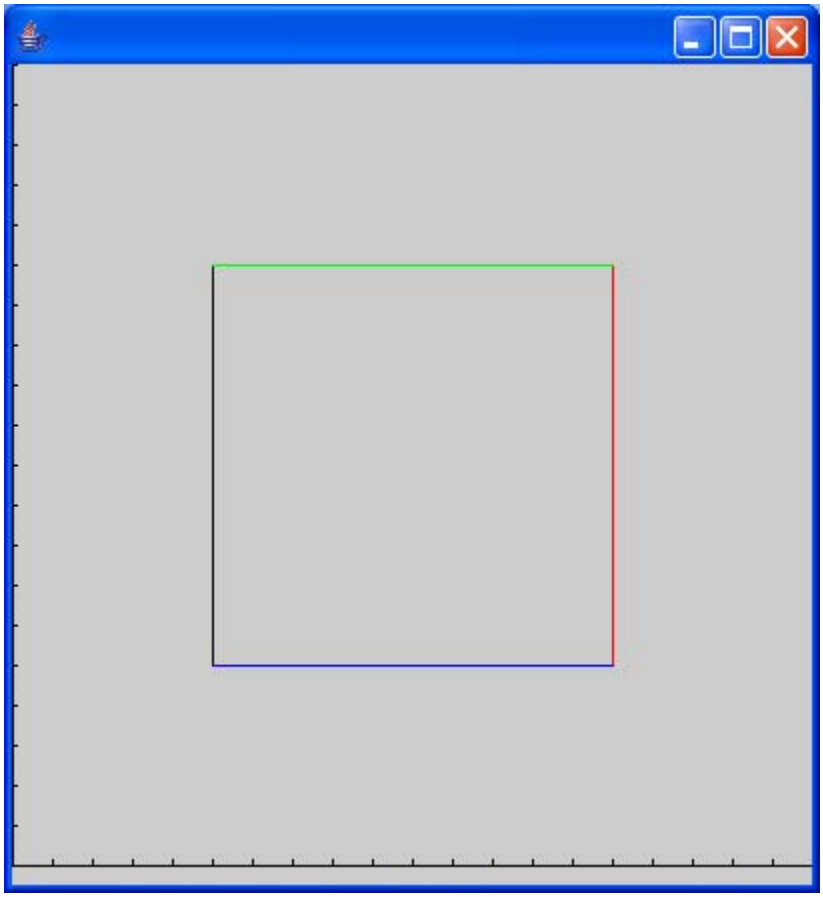
```
Begin
#This is the main file of a BATS
#program, compiler seeks this out
#and starts running code from here

  Line side;
  Let side <- {^100,100^,^300,100^,^300,300^,^100,300^,^100,100^};
  # Notice how the vertices are chained!
  Draw side;
End
```

This makes:



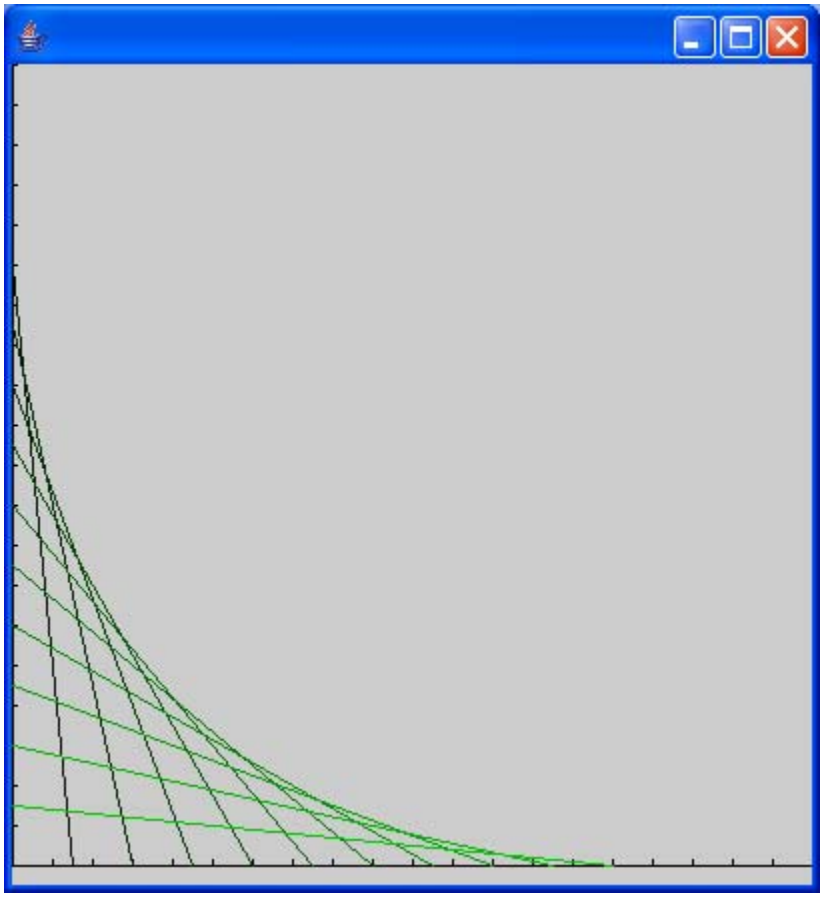
Now let's add color to each line. This means we'll have to break up side:

```
Begin
Line lside, rside, tside, bside;
Point nw, ne, sw, se;
  Let nw <- ^100,300^;
  Let ne <- ^300,300^;
  Let sw <- ^100,100^;
  Let se <- ^300,100^;
  Let lside <- {sw, nw};
  Let rside <- {se, ne};
  Let tside <- {nw, ne};
  Let bside <- {sw, se};
  Draw lside;
  Color 255; 0; 0; #Next lines are red
  Draw rside;
  Color 0; 255; 0; #Next lines are green
  Draw tside;
  Color 0; 0; 255; #Next lines are blue
  Draw bside;
End
```



3. Harder example, using control:

```
Global Line Xshape, Yshape, temp;
                                        Boolean Begin
Int w, z, dacolor;
Let w <- 300;
Let z <- 30;
Let dacolor <- 20;
        While w > 0 Do
                Let temp <- {^ z,0 ^ , ^ 0,w ^ };
                Color 0; dacolor; 0;    #it's going to get more and more green
                Draw temp;              #draw the line Let w <- w – 30;
                Let z <- z + 30;
                Let w <- w – 30;
                Let Dacolor <- dacolor + 20;
        WhileEnd                        #note, indentation is not required
Return True;
End                                     #ends Begin function
```



To use Bats, delete the “input” file if there is any in the Compiler folder. Then, make a file called “batssource.txt”. Create your Bats program there, and run “RunBats.bat”. Then copy the file “input” to the Textual Translator folder. Finally, run “StartBats.bat”.

BATS Language Reference Manual

Introduction:

The BATS language, a geometric figure drawing language, was designed to be simple to use. Through its simplicity, complex figures can be drawn using complex algorithms. The language also includes loops, conditional statements, and functions to modularize a programmer's language. These pieces of block structured programming are essential for effectively representing geometric algorithms in an understandable format. And since this is a drawing language the choice of declaring, assigning, and using lines, as will be seen, will make drawing vastly easier. BATS will use an over-all interpreter that interprets BATS code into the target language – Java.

Syntax (Grammar) notation:

This language reference manual uses a pseudo-ANTLR language to represent the grammar of BATS. Grammars are clearly divided from paragraphing and text to show the grammar exactly. Alternatives are separated by the '|' character. Nonterminals can be placed in an optional set of parentheses, but they are also always italicized. Literals or terminals are placed in single quotes (i.e. '1', 'e'). A set of literals, such as the alphabetic set, can be placed in parentheses, where the first quoted literal is followed by two periods, which is followed by the last literal of the set in single quotes. Optional expressions are placed in parentheses followed by the questions mark character '?'. Iteration is placed in parentheses followed by either the kleene star '*', which means that the parenthesized literals and/or syntactic categories can be matched zero or more times, or the plus character '+', which means that the parenthesized literals and/or syntactic categories must be matched at least once. Epsilon is represented as /*nothing*/. So in the grammar:

Yourname:
    Name ('A' . . 'Z')? (*Name*)?

Name:
    ('A' .. 'Z') ('a'..'z')*

Yourname is the starting symbol. The first name, or Name, is a syntactic subcategory that expands to one uppercase letter followed by any number of lowercase letters. After Name the middle initial is just an optional single uppercase letter. The last name is just Name reiterated to match a last name.

Lexical conventions:

The following are the tokens found in this language: identifiers, keywords, constants, operators, and separators.

Separators:

Blanks (the whitespace), tabs, new lines, carriage returns, and comments are ignored in token creation other than to serve the purpose of separating tokens. The input stream, which is a file containing BATS programs, is parsed so that each token is taken to include the longest string of characters which could possibly constitute a token. This means the parser will keep adding to the token until it hits a whitespace, a tab, a new line, a carriage return, or a comment.

Comments:

The question mark character # introduces a comment, which terminates with a new line.

Identifiers (Names)

An identifier is a sequence of letters and digits where the first character must be alphabetic. The underscore ‘‘_’’ is counted as a letter. Upper and lower case letters are considered different. The following is the grammar for an identifier:

Identifier:

(‘A’..’Z’|’a’..’z’|’_’) (‘A’..’Z’|’a’..’z’|’0’..’9’|’_’)*

These are some examples of identifier names:

hello, hello1, _hi3, I_AM_A_VARIABLE

The following cannot be identifiers:

1joe, thr?ee, b|leach

Keywords

Keywords are reserved words that aid the programmer in creating an understandable program that the compiler will accept. These keywords are used for things such as looping, conditional statements, and type declarations. To make them easier to see in a BATS text, they all start with a capital letter. No identifier can have a name identical to the following keywords:

Int	Boolean	Line	Point	From	To

Draw	Call	Function	Begin	End	While
For	If	Else	WhileEnd	ForEnd	IfEnd
True	False	Do	Color	Then	Append
ForLine	In	Xaxis	Yaxis	Return	Global
Let					

Constants

There are three different types of constants in the BATS language: integer constants, double constants, and Boolean constants. These three are defined as follows:

Integers:

An integer consists of a sequence of ASCII character '1' to '9', and that sequence of ASCII characters represents its real number value. For example, the real number value equivalent of the sequence "123" is 123 or one hundred twenty three. The following is the grammar composing an integer constant:

Integer:
(*Digit*)+

Digit:
('0' . . '9')

Booleans:

There are only two Boolean constants in the BATS language, and they are "True" and "False", representing the logical values of true and false, respectively. The following is the grammar for a Boolean constant:

Boolean:
'True'|'False'

What an Identifier Stands For

An identifier can stand for another identifier, a constant, or a more complex set of constants. The attributes of an identifier are the storage class and its type. The storage class of an identifier defines the scope of an identifier, that is to say, how long and where the identifier exists through the running of a program. Its type is the type of constant or complex set of constants it stands for.

Currently, there are two storage classes an identifier can choose from to have. The first storage class is global. The identifier exists inside and outside all functions until the program has terminated. A global identifier is declared outside all functions, in the global arena. The second storage class is functional. Here, the identifier exists inside only a specific function, and disappears when the function returns to its caller.

The type of an identifier can be of the following simple constants:

- Integer:

  Integers (*Int*) are represented by an actual integer value in the target language of BATS, Java. Integers can have any value from –2147483648 to 2147483647, inclusive.

- Boolean:

  Booleans (*Boolean*) are represented as the string “true” or “false” in Java. Booleans are primarily used for conditional statements.

The type of an identifier can be a complex set of constants, which are:

- Point

  Points (*Point*) are a set of two integer identifiers. These are essentially used for drawing and geometric figures. The following is the grammar of a point:

  Point:

  ‘^‘ *Integer* ’ ‘,’ *Integer* ‘^’

- Line

  Lines (*Line)* are a set of two or more points. These are the building blocks of BATS and they are important enough to be given a separate section further down in this reference manual.

- Array

  Arrays (*Array*) are a set of one type of identifier. One can have an array of points, integers, Booleans, points, or lines. An identifier can also be an array with an index following it. For example, if we have array a, a[0] would be the index of the first element in the array. Array grammar is described in its own section.

- Functions calls

  Functions (*Function)* calls return an object of a given type. They can return integers, Booleans, points, or lines. Function definitions are described in its own section.

An *Identifier* must be one of these, and nothing else.

Variables

Variables are expressions that stand for another expression. Expressions are defined below. Some operators yield variables or expect them, and that detail will be described for each operator.

Expressions

The precedence of expression operators will be in descending order through their explanation in this section. Operators in subsections have equal precedence. Note: 1st level precedence is the highest precedence, 2nd level precedence is second highest, and so on.

1st level precedence

Expressions in the highest precedence are grouped right to left.

Identifiers

Identifiers are in this level. An identifier is an expression if it has been properly declared and it has a variable of the proper type assigned to it. The identifier in an expression gives that type.

Array indices

Obtaining information from an array index or indices (in case of a multi-dimensional array) has first level precedence. This is discussed in more detail in the Arrays section.

Constants

Constants (Integers, Booleans) are in this level.

Parenthetical expression

(*expression*)

An expression with parentheses around it, (*expression*), is at this level. The value of the parenthesized expression is equal to that of the expression inside the parentheses.

Function call

A function call is an expression followed by parentheses containing a list, possibly empty, of expressions that are the arguments of the function. These arguments can be expressions that have to evaluate to the correct type of value in the function definition (discussed in further detail in the Functions section). Arguments are passed by value in function calls, since a copy is made of each actual parameter. Recursive function calls are possible. The following is the grammar for a function call:

Funccall:

‘Call’ *Identifier* ‘(‘ *Arglist* ’)’

Arglist:
expression | *Moreargs*

Moreargs:
‘,’ *expression* | /*nothing*/

Note: the *expression* after ‘Call’ has to be the name of a function.

2nd level precedence

Unary operators are in this level of precedence, and they are grouped right-to-left.

Number Negation

To negate a number, which an integer or double, the following grammar is needed:

Negate:
‘-‘ *expression*

This turns 25 into –25 and –45.7 into 45.7.

Boolean Negation

To negate a Boolean, the following grammar is needed:

NegateBool:
‘!’ *expression*

This turns False into True and True into False

3rd level precedence

Multiplicative operators, *,/, and %, are in this level of precedence, and they are grouped left to right.

Multiplication

Only two numbers can be multiplied. If two integers are multiplied, then the product is an integer. If two doubles are multiplied, then the product is a double. If an integer and a double are multiplied, the product is a double. No other combinations are possible. The grammar is:

Multiply:

expression '' expression*

Division

Only two numbers can be divided. If two integers are multiplied, then the product is an integer. If two doubles are multiplied, then the product is a double. If an integer and a double are multiplied, the product is a double. No other combinations are possible. The grammar is:

Divide:

expression '/' expression

Modular division

Only two integers can be used in modular division. It yields the remainder from the division of the first integer by the second. The grammar is:

ModDivide:

expression '%' expression

4th level precedence

Additive operators, + and -, are in this level of precedence, and they are grouped left to right.

Addition

Only two numbers can be added. If two integers are multiplied, then the sum is an integer. If two doubles are multiplied, then the sum is a double. If an integer and a double are multiplied, the sum is a double. No other combinations are possible. The grammar is:

Addition:

expression '+' expression

Subtraction

Only two numbers can be subtracted. If two integers are subtracted, then the subtraction result is an integer. If two doubles are subtracted, then the subtraction result is a double. If an integer and a double are subtracted, the subtraction result is a double. No other combinations are possible. The grammar is:

Subtraction:

expression '-' expression

5th level precedence

Relational operators are in this level. They are grouped right to left.

Less than
Greater than
Less than or equal
Greater than or equal

The operators <, >, <=, and >= evaluate to true if expression on the left operator is less than, greater than, less than or equal to, and greater than or equal to the expression on the right of the operator, respectively. The expression returns false otherwise. The following are the respective grammars for the operators:

Less:
e*xpression* '<' expression

Greater:
e*xpression* '>' expression

LessEqual:
e*xpression* '<''=' expression

GreaterEqual:
e*xpression* '>''=' expression

6th level precedence

The equality operators are in this level of precedence, and they are grouped right to left. So True == False != False has a result of False.

EqualsEquals
NotEquals

The equalsequals operator '==' and notequals '!=' return true when the value of the expression on the left is equal and not equal to the value of the expression on the right, respectively. It returns false otherwise. The grammars are:

EqualsEquals:
expression '=''=' expression

Notequals:
expression '!''=' expression

7th level precedence

And

The *And* operator is in this level of precedence. It is used to connect Boolean expressions together. It is grouped right to left. The following is the grammar

And:
expression '&'& ' expression

Both expressions must be Boolean. If both sub-expressions return true, the *And* expression returns true, and it returns false otherwise.

8th level precedence

Or

The *Or* operator is in this level of precedence. It is used to connect Boolean expressions together. It is grouped right to left. The following is the grammar

And:
expression '|'|' expression

Both expressions must be Boolean. If both sub-expressions return false, the *Or* expression returns false, and it returns true otherwise.

9th level precedence

Assignment operators, which are found in this level, are grouped right to left. All of them require a variable as their left operand, and the type of that variable being the result of the expression on the right.

Equals

In the Equals expression, the value of the right expression replaces that of the object referred to by the variable in the grammar below. Both the variable type and the expression result on the right must be of the same type (Int, Line, Point, Double, Array, Boolean). The following is the grammar:

Equals:
v*ariable '<''-' expression*

Declarations

Declarations, specifically declaration lists, are used in the global scope (outside of functions) to declare global variables, in function definitions, and in the first part of a function to declare temporary function variables. However, the BATS programming language has two kind of declaration lists, one specific for global variables, and another for local variables. The following is the grammar for the global declaration list and the local declaration list.

DeclarationsGlobal:
‘Global’ *type-specify identifier identifier-list* ‘;’ (*DeclarationsGlobal*)*?*

type-specify:
(‘Int’|’Boolean’|’Line’|’Point’) (“[]”)?

identifier-list:
*‘,’ identifer (identifier-list)?| /*nothing*/*

The following is the grammar used for a declaration list as found in a function body.

Declarations:
type-specify identifier identifier-list ‘;’ (*Declarations*)*?*

type-specify:
(‘Int’|’Boolean’|’Line’|’Point’) (“[]”)?

identifier-list:
‘,’ identifer (identifier-list)?

How Declared identifiers work

Identifiers that are declared become variables. Each variable that has been declared will yield its type and value or object. So if we have Int x. The type of variable x is Int, which is an integer. Using an assignment operator, we can assign an integer to the variable x. After that assignment, any usage of x will yield that assigned integer.

Statements

Statements are executed in sequence unless a conditional, loop, or function is used. Most statements are expressions with the following grammar:

e*xpression* ‘;’

A statement can be a list of statements, or *statement-list* to be executed:

s*tatement-list*:
statement (*statement-list*)?

Conditional Statements

The grammar of a conditional statement is:

IfBlock:

‘If’ *expression Then statement* (‘*else’ statement)? ‘IfEnd*‘

If the evaluation of the *expression* in parentheses is true, then the first *statement* is evaluated. If it is false, then if the optional else clause exists, it is evaluated. After this, the ‘IfEnd’ keyword is matched and the program leaves the if block.

Loops

There are three loops used in BATS: the while loop and for loop and forline loop. The while loop’s grammar is the following:

WhileBlock:
‘While’ *expression* ‘Do’ *statement* ‘WhileEnd’

While the Boolean value of the *expression* is true, then the *statement* is executed. When the *expression* value is false, the WhileEnd keyword is found and the program leaves the while block.

The for loop has the following grammar:

ForBlock:
‘For’ (*expression*)? ‘;’ (*expression*) ‘;’ (*expression*)? ‘Do’ *statement*
‘ForEnd’

The first and last *expressions* are optional. The first *expression* is typically initialization of a variable, like x = 0. The second *expression* is a test before iteration, similar to the *expression* found in *WhileBlock*. The last *expression* is usually used to increment or decrement of the *variable* initialized in the first *expression*. If the first and last *expressions* are not present, then the for loop just becomes a while loop.

ForLineBlock:
‘*ForLine*’ *Identifier* ‘In’ *Identifier* ‘Do’ *statement* ‘ForEnd’

In this special type of for block, the first identifier takes on the value of the first element’s index of the second identifier (which must be a line). Using this first identifier, the for line loop iteratively go through each point in the line and does the statements.

Lack of a break statement

To create cleaner and more of a block structured language, a break statement found in Java and C is not found in the BATS language.

Return statement

A return statement is what a function uses to return to its caller. Each function must return in the BATS language, even if what is returned by the function is not

assigned to anything. For example, a function can just return true and not assign the true value to anything. The grammar for return is the following:

Returnstmt:
'Return' *expression* ';'

Functions

A major part of the BATS language is the use of functions. To be used, they first must be defined in the following manner:

FunctionDecl:
type-specify FunctionDeclarator FunctionBody

FunctionDeclarator:
Identifier '(' (*Parameters*)? ')'

Parameters:
type-specify Identifier Parameter-list

Parameter-list
*',' Parameters | /*nothing*/*

FunctionBody
'{' (Declarations)? Statement '}'

If, in *Parameters*, a Double is stated, but an integer is passed to that function, the integer is automatically converted to a double. The reverse does not occur. Functions cannot be nested; functions are only declared outside other functions in the global scope.

Arrays

Now we must realize how to get to a specific index of an array. The number of bracket sets must be of the same number of brackets when the Array was declared. The following is the grammar that is used to obtain information from an array.

ArrayUse:
Identifier ('[' *Integer* ']')+

All one has to do to use an array is use *ArrayUse* after declaring an array and assigning the correct type of variables to the array. *Arrayuse* is an *expression*, which would evaluate the type and the value at the specified index. It has the same precedence as identifiers, that is, first level precedence.

Lines

Lines are special type of *Variables*, which can be compared to a single array of type *Point*. However, there is a special syntax for Lines.

Line:

‘{‘ *Point* (‘,’ *Point*)+‘}’

Of course, the Point can be a variable that stands for a Point. However, a line is composed of at least two points. More importantly, however, is that braces, to distinguish it in the code, denote it.

Scope

Global variables and functions, which are declared outside any other functions, have scope over the entire program; in other words, they have global scope. They can be used or called anywhere within the program. Any variables in a function have a local or lexical scope limited to the function. After there is a return from the function, the variables are thrown away.

Built-In Functions

This section discusses three important functions that are usable in BATS: Color, Draw, and Append.

Color

The Color function is used to specify the color of the next Line to be drawn (with the built-in function Draw). It takes in three double values (the Red, Green, and Blue value that constitute the RGB value) for the next lines until Color is called again. The default color for a line is black. The following is the grammar for Color

Color:

‘Color’ *expression* ’;’ *expression* ‘;’ *expression*

Calling the built-in function is an *expression*. Therefore, when it’s a statement, a semicolon is added to the end to create something like the following example:

```
Color 176, 176, 176;
```

This makes the next lines gray.

Draw

This is the most important function in BATS, and it’s built-in, ready to be used. Draw takes in a Line and produces it graphically on the screen in the viewer. The following is the grammar for Draw:

Draw

‘Draw’ *Line*

The following example:
Let J <- {^0,100^ , ^100,100^};
Draw J;

draws a line from point 0,100 to point 100,100.

Append

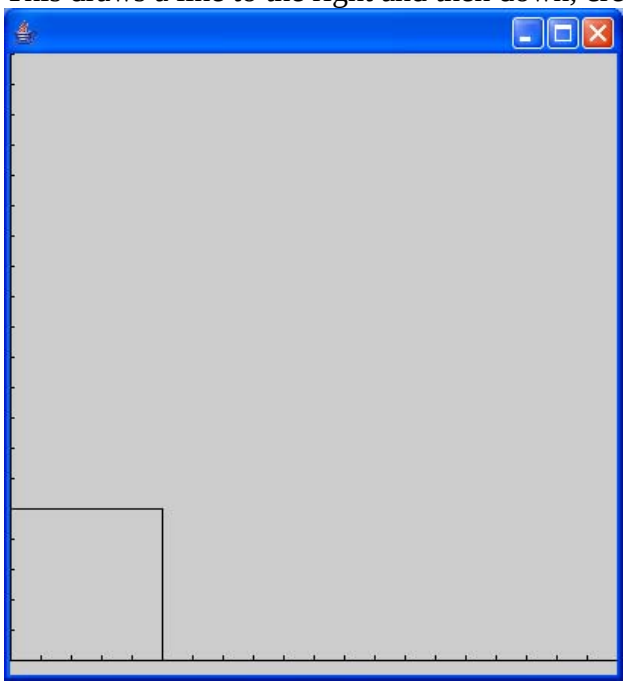
The Append function is used to add a point to a line, a point to a point, or a line to a line. The grammar for the Append function is:

Append:
'Append' *Identifier* 'To' *Identifier*

Calling the built-in function is an *expression*. Append returns a line. So this must be assigned to a line variable or drawn. Since the function is an expression, it is a statement, so a semicolon is added to the end to create something like the following example:

Point P;
Let p <- ^100,0^;
Append P to J;
Draw J;

This draws a line to the right and then down, creating a half of a square:



Project Plan

Section 1: Planning and Development

Planning was a very interactive and democratic experience in this group. People agreed to take parts, but team members were assigned responsibilities if they had no preference as to what part of the project was to be their duty. In this way, team responsibilities discussed in section 2 were assigned.

Development occurred generally independently, but questions, coding milestones, revisions, and changes of ideas were immediately emailed in a reciprocal fashion. Also, communications was eased using AOL Instant Messenger™. Communications was crucial to development, for any clearing up of queries or revisions had to be addressed immediately or as soon as possible.

For anything to start, the grammar was discussed and created. Using ANTLR and its corresponding syntax, the scanner and parser for BATS was created. This gave impetus to independent work on the walker, interpreter, and final translator. Therefore, sequential coding gave way to concurrent work. Agreement between team members about input and output of their corresponding pieces of the compiler gave ease to this. Predictably, the parser and walker were finished first. The symbol table generation was wrapped up along with the walker. Error catching in the walker was an ongoing process. The tree interpreter and interpreter text-to-screen-image translator were finished last, and the input of the translator (the interpreter's output) was strictly and strenuously understood and tested.

Section 2: Team Responsibilities

Below is a table of what each person was assigned to do for the project:

Behrooz Badii	Documentation, scanner, and Parser
Aleksandr Borovinskiy	Interpreter text to screen image translator
Tanya Shtemberg	Symbol Table generation, Walker error catching, Interpreter
Sui Sum Wong	Walker grammar notation, revision of Parser, test cases

Section 3: Project Timeline and Log

The following table notes the dates of major achievements of the project.

White paper finished and delivered by Behrooz	September 23
Language reference manual (LRM) finished and delivered by Behrooz	October 28

Lexer and Parser finished by Behrooz	Nov 12
First Walker draft by Sui	Nov 29
Final Walker and Parser by Sui	Dec 8
Error Checking and Symbol Table Generation and Maintenance begun by Tanya	Dec 8
Final Parser with noted changes by Sui	Dec 12
Final Interpreter text to screen image translator by Alex	Dec 12
Final version of Error Checking and Symbol Table Generation and Maintenance by Tanya	December 15
Interpreter completed by Tanya	December 16
Translator inside Interpreter revised by Tanya	December 17
Final paper finished by Behrooz	December 17
Final paper with all code delivered by BATS team	December 18

Section 4: Development Environment

Each person in the team had a different environment they were using. All were using Windows operating systems to program their pieces in. The primary language used to program any and all code was Java. Behrooz used Java 1.3.1 on a Windows XP machine. He also used JBuilder for text editing. Alex worked on a Windows XP machine using Eclipse software as a development environment. Sui worked on a Windows ME machine using Java 1.4.2. Respective group members used ANTLR to create the Scanner/Lexer, Parser, and Walker. Tanya used Java on the Windows XP operating system. She also programmed using NetBeans.

Section 5: Coding Style

Subsection 1: Introduction

Coding styles are as unique as the number of people coding in the world. It is difficult to put constraints on any number of people about their coding without having objections. However, there were conventional coding techniques used by all the participants that are clearly apparent in coding.

Subsection 2: General overview

Code has to be clear and concise. Comments should be clearly identified. Revisions by other group members should also be clearly identified using comments. Each piece of code must have an author or co-authors.

Subsection 3: Indentation format

Code must be structured well. Specifically, code must adhere to block structuring and also represent its block structure by the proper use of indentation. There is a strict adherence to conventional block (conditional, functional, looping blocks) indentation as seen in ordinary java coding. Opening braces must be accompanied by closing braces at the same level of indentation at a lower line if the corresponding closing brace is not on the same line. The following three fashions of brace and parenthesis placement accepted and are used by respective BATS group members.

Method G
{
…
}
	Method G
	{ … }
		Method G {
		}

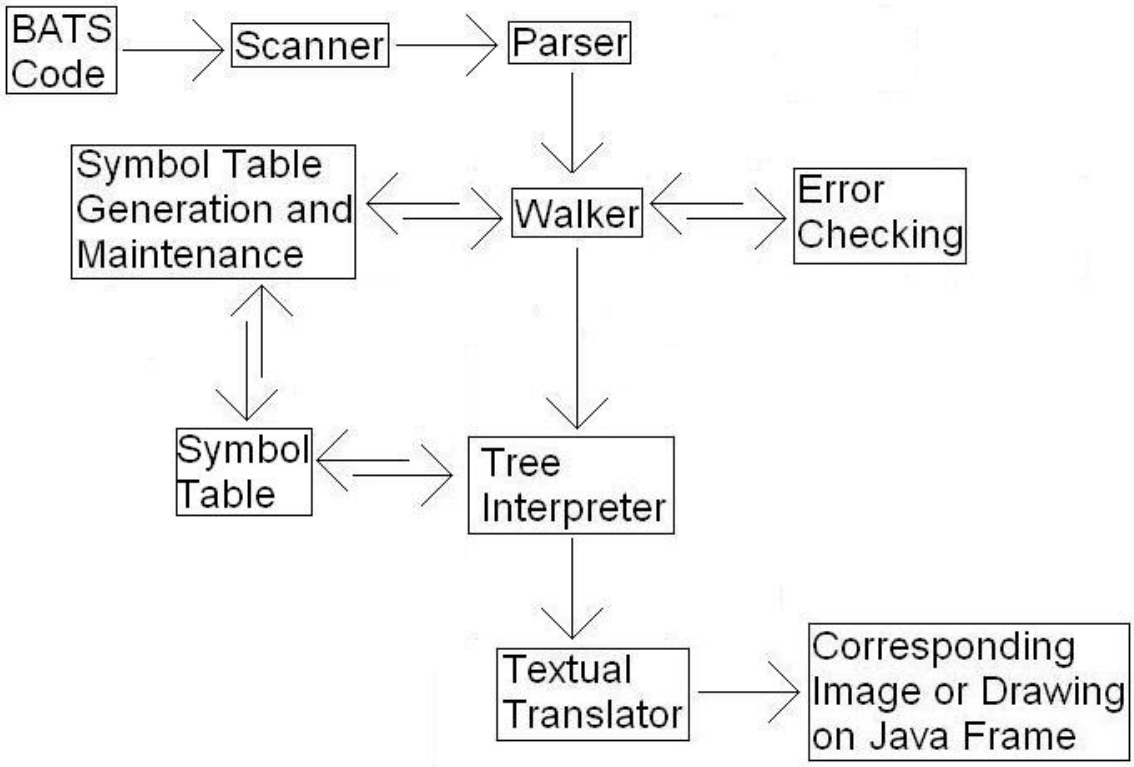
If hard tabs are to be used, they are to be used in a corresponding fashion, where the block is hard tabbed completely.

Subsection 4: Commenting

Commenting should be used vigorously if the code in question has not been strenuously shared and discussed among group members. Large portions of commented code should be either clearly identified or placed at the bottom of the *.java file that encompasses said code.

Architectural Design

Section 1: Diagram of Components



Section 2: I/O between Components

The input of the Scanner is BATS code found in a text file. The scanner divides up the text file into tokens and passes the stream of tokens to the Parser. The Parser syntactically analyzes the stream and passes an ANTLR AST to the Walker. The Walker analyzes the static semantics of the AST. It also creates the Symbol Table and maintains it. The symbol table is comprised of two inner symbol tables. One is the global symbol table, and the other is the local symbol table at that time. The symbol table is saved for the Tree Interpreter, which actually reads the tree and creates an output file. The tree interpreter also creates an intermediary java program. As an example, we will be revisiting a piece of code in the tutorial. This the example:

Global Line Xshape, Yshape, temp;

```
                                        Boolean Begin
Int w, z, dacolor;
Let w <- 300;
Let z <- 30;
Let dacolor <- 20;
        While w > 0 Do
                Let temp <- {^ z,0 ^ , ^ 0,w ^ };
                Color 0; dacolor; 0;    #it's going to get more and more green
                Draw temp;              #draw the line Let w <- w – 30;
                Let z <- z + 30;
                Let w <- w – 30;
                Let Dacolor <- dacolor + 20;
        WhileEnd                        #note, indentation is not required
Return True;
End                                     #ends Begin function
```

The former code gives this as an output to the Textual Translator:

```
2 0 20 0 |
1 30 0 0 300 |
2 0 40 0 |
1 60 0 0 270 |
2 0 60 0 |
1 90 0 0 240 |
2 0 80 0 |
1 120 0 0 210 |
2 0 100 0 |
1 150 0 0 180 |
2 0 120 0 |
1 180 0 0 150 |
2 0 140 0 |
1 210 0 0 120 |
2 0 160 0 |
1 240 0 0 90 |
2 0 180 0 |
1 270 0 0 60 |
2 0 200 0 |
1 300 0 0 30 |
```

Obviously this output needs some explaining. Each line has either a 0,1,or 2 as its first token. 0 denotes a Point to be drawn. 1 denotes a Line with a variable number of x and y pairings. The second and third tokens of a line containing a Line or Point give x and y pairings. A Line goes on to describe more and more x and y pairings (4^{th} and 5^{th} token give a pairing, 6^{th} and 7^{th} give a pairing, and so on). Since the number of tokens for a line is of variable length, the "|" character denotes the end of a line in the input file. When a line in the input file denotes a color, it has only three numbers following the number 2.

These respectively denote the Red, Green, and Blue values of the line. For example, the first color change gives 0 20 0 as its RGB values. This is a light green line. At the end of the file, there is an RGB value of 0 200 0, which is a dark green line.

Section 3: Authorship of Components

Below is a table of the authors of each piece of code:

Behrooz	Lexer and Parser
Alex	Textual Translator
Sui	Parser (revision) and Walker (co-author)
Tanya	Walker (co-author), Tree Interpreter, Error checking, Symbol Table Generation and Maintenance.

Note: There were revisions and re-revisions done by other group members. For example, Tanya and Behrooz revisited the Parser and had to consequently revisit the Walker to ease code interpretation and the coding of the Tree Interpreter.

Test Plan

Section 1: Overview

We believed that the testing of a component should be carried out by more than just the author of that component. For example, Alex did the Textual Translator, but Behrooz tested it. Of course, Alex tested it too, but giving the component to another person to test created a sense of objectivity and consequentially more reliability of the software. Also, the preliminary components were tested more, since they were finished earlier in the semester. Also, since the Textual Translator had its own easy to read input before drawing, one could test the project in reverse. The example used in Architectural Design, Section 2, was actually predicted to give those values to the Textual Translator. The interpreter was then tested to return that value. These two important parts of the BATS engine had to coincide perfectly or near perfectly, or else the last step, the most beautiful, of actually creating images would be fallible. No tools were used in testing; testing was done manually to ensure maximum efficiency within a component and between components.

Section 2: Testing the Whole

During testing, most components had complete components before them. The job of that component was to sequentially implement each feature. For example, at the Tree Interpreter stage, linear programming was tested. Then conditional blocks were tested. After than, loops were implemented and tested and so on. However, most of the objective testing as discussed in Section 1 was done after the component was finalized.

Section 3: Testing Individual Pieces

Before testing BATS as a whole, the pieces of the process, of course, had to be individually tested. Up through the Tree Interpreter stage, the author or authors of the next component tested the component in question. For example, Sui and Tanya vigorously tested the Parser. Tanya tested the Walker for her Interpreter. However, after that testing branched out. Sui and Alex tested the Tree Interpreter for correct output. Behrooz tested the input of the Textual Translator to ensure proper final drawing.

Section 4: Uniting the Pieces

Finally, after testing the pieces, they were slowly united and tested together, producing a unified “one.” The Parser and Walker were tested to ensure correct output for the Tree Interpreter. They were then combined to create textual output, which was

tested with the Textual Translator. This simply ensures continuity and correct communication between components. Finally, all the pieces were tested in a logical fashion with test programs.

Section 5: BATS Program Examples

For the sake of continuity, we will use two of the source programs in the tutorial examples. The first shows the power of color change, and the second shows control flow.

```
Line lside, rside, tside, bside;
Point nw, ne, sw, se;
  Let nw <- ^100,300^;
  Let ne <- ^300,300^;
  Let sw <- ^100,100^;
  Let se <- ^300,100^;
  Let lside <- {sw, nw};
  Let rside <- {se, ne};
  Let tside <- {nw, ne};
  Let bside <- {sw, se};
  Draw lside;
  Color 255; 0; 0; #Next lines are red
  Draw rside;
  Color 0; 255; 0; #Next lines are green
  Draw tside;
  Color 0; 0; 255; #Next lines are blue
  Draw bside;
End
```

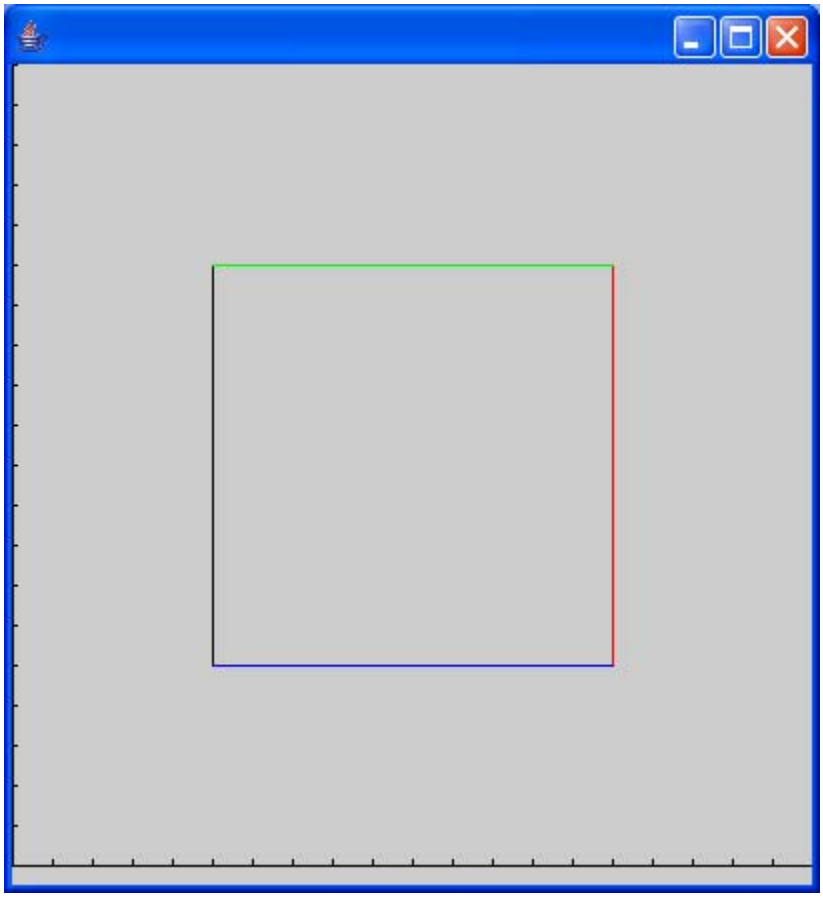
This example creates the following java code in BatsProgram.java:

```
import java.util.*;

public class BatsProgram{

 BatsBuiltIn builtIn = new BatsBuiltIn();

 BatsColor color = new BatsColor();
boolean Begin(){
BatsLine lside, rside, tside, bside;
BatsPoint nw, ne, sw, se;
 nw =  new BatsPoint( 100, 300) ;

 ne =  new BatsPoint( 300, 300) ;

 sw =  new BatsPoint( 100, 100) ;

 se =  new BatsPoint( 300, 100) ;
```

```
 lside =  new BatsLine();
  lside.addPoint(sw); lside.addPoint(nw);
 rside =  new BatsLine();
  rside.addPoint(se); rside.addPoint(ne);
 tside =  new BatsLine();
  tside.addPoint(nw); tside.addPoint(ne);
 bside =  new BatsLine();
  bside.addPoint(sw); bside.addPoint(se);
 builtIn.append( lside );

 color = new BatsColor(255,0, 0);
 builtIn.append( color );

 builtIn.append( rside );

 color = new BatsColor(0,255, 0);
 builtIn.append( color );

 builtIn.append( tside );

 color = new BatsColor(0,0, 255);
 builtIn.append( color );

 builtIn.append( bside );

 return true;

}

 public static void main(String[] argc){
    BatsProgram bpr = new BatsProgram();
    bpr.Begin();
  bpr.builtIn.drawLine();
 }
}
```

This, in turn, gives the following input to the Textual Translator:

```
1 100 100 100 300 |
2 255 0 0 |
1 300 100 300 300 |
2 0 255 0 |
1 100 300 300 300 |
2 0 0 255 |
1 100 100 300 100 |
```



The other example is the following:

```
Global Line Xshape, Yshape, temp;
Boolean Begin ()
Int w, z, dacolor;
Let w <- 300;
Let z <- 30;
Let dacolor <- 20;
While w > 0 Do
      Let temp <- {^ z,0 ^ , ^ 0,w ^ };
       Color 0; dacolor; 0;
       Draw temp;              #draw the line
       Let z <- z + 30;
       Let w <- w - 30;
       Let dacolor <- dacolor + 20;
WhileEnd                    #note, indentation is not required
Return true;
End
```

The translator within the interpreter returns the following Java Program:

```
import java.util.*;
```

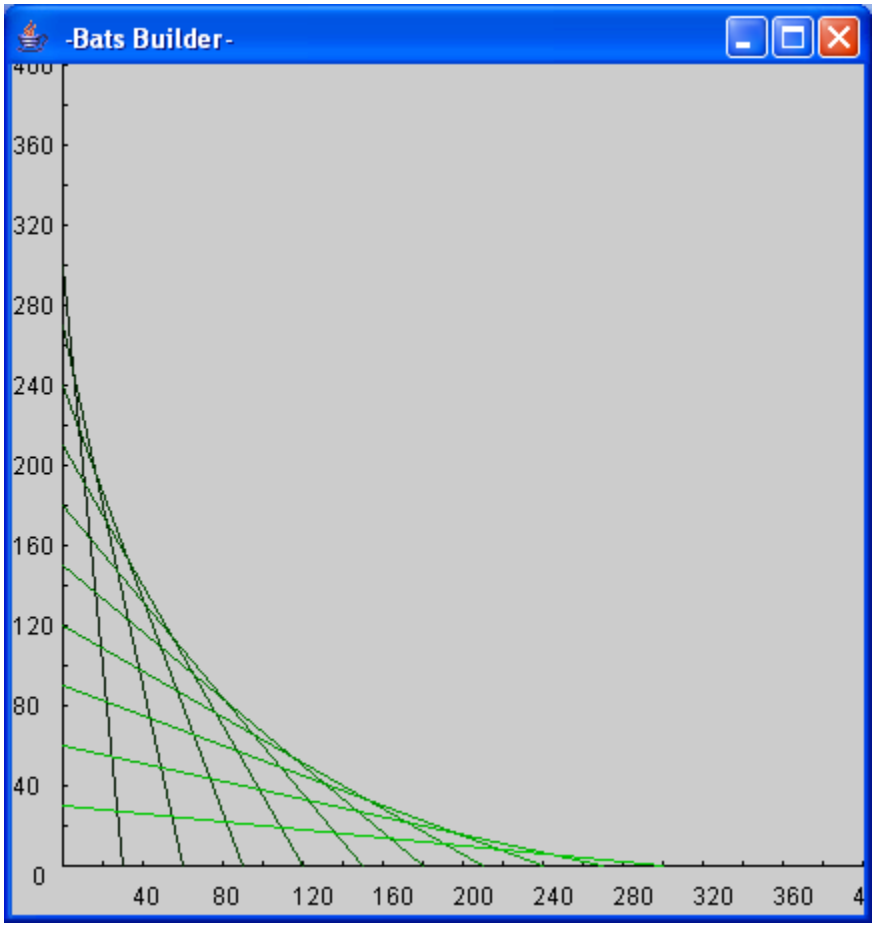
```
public class BatsProgram{

 BatsBuiltIn builtIn = new BatsBuiltIn();

 BatsColor color = new BatsColor();
BatsLine Xshape, Yshape, temp;
boolean Begin(){
int w, z, dacolor;
 w = 300 ;

 z = 30 ;

 dacolor = 20 ;
while( w > 0 ){

 temp =  new BatsLine();
  temp.addPoint( new BatsPoint( z, 0));

 temp.addPoint( new BatsPoint( 0, w) );
 color = new BatsColor(0,dacolor, 0);
 builtIn.append( color );

 builtIn.append( temp );

 z = z + 30 ;

 w = w - 30 ;

 dacolor = dacolor + 20 ;

}

 return true;

}

 public static void main(String[] argc){
    BatsProgram bpr = new BatsProgram();
    bpr.Begin();
  bpr.builtIn.drawLine();
 }
}
```

Which in turn creates the following input file.

```
2 0 20 0 |
1 30 0 0 300 |
2 0 40 0 |
1 60 0 0 270 |
2 0 60 0 |
1 90 0 0 240 |
2 0 80 0 |
1 120 0 0 210 |
2 0 100 0 |
```

1 150 0 0 180 |
2 0 120 0 |
1 180 0 0 150 |
2 0 140 0 |
1 210 0 0 120 |
2 0 160 0 |
1 240 0 0 90 |
2 0 180 0 |
1 270 0 0 60 |
2 0 200 0 |
1 300 0 0 30 |

Then, the textual translator creates the following graphic:



These examples were chosen to show the color and draw functions. However, more importantly, they were chosen to show that the BATS language supports the essence of structured programming and control flow: loops and conditionals.

Section 6: Testing Automation and Method

Testing was done “by hand” when individual pieces were tested. That is, the group did not use formal assert values as shown in class. Group members used their own personal print statements to find errors if anything unexpected occurred. Thanks to heavy testing by the authors of each component, not too many errors came up. This informality adds more focus to the task of testing. If there were errors in a piece of software, the testers would meet to deal with the problem, share their ideas, and ultimately fix the malicious bug.

Section 7: Responsibility of testing

The following is a table of the testers of each step of the software (the steps would be lexical and syntactic analysis, semantic analysis, Interpretation, and Graphical Translation):

Scanner/Parser	Behrooz, Sui
Walker/Symbol Table/Error Checking	Sui, Tanya, Behrooz
Tree Interpreter	Tanya, Sui, Alex
Textual Translator	Behrooz

Lessons Learned

Section 1: Individual Lessons

Sui:

I think the most important thing I learned is that it is important to ask questions when you are stuck with one problem for more than 2 days, and it is also important to have faith in the people whom you work with. My group partners gave me lots of support during the process of creating the walker. We solved problems together, and that made everything much easier.

Sui

Behrooz:

Communication, communication, communication. It's the key part to working in a team. It reflected the amount of work we did. In the beginning, communication was minimal, and work was almost non-existent. But by the end, we were emailing each other on an almost hourly basis. Then, when we started to use AOL Instant Messenger™, work was being done at an almost alarming rate. Testing was occurring and output was examined by all team members in a matter of seconds instead of hours. And we were open to the ideas of people that had become our friends through this monstrous escapade of work and revision.

What I learned was that we should start early. I had first agreed to the documentation part, because, due to the first slides in class, that was designated as one whole quarter of the work. Since we're four people, it seemed sensible. After the LRM, work had not started, so after giving impetus to the rest of the group by doing the Parser, they started to work at full force. I wish, however, that the division of work occurred quicker and more equally. This would have been easier for everybody. I empathized with the other group members when they had so much trouble on a crunched schedule, and we could have and should have seen to it that that wouldn't occur.

My last lesson is to have faith in my group. To be honest, I had a lot of qualms with their work in terms of when they started and how fast they did it. But as deadlines came up, they delivered every time. It truly changed my opinion, and I was honored to work with these fine people by the end.

Behrooz

Tanya:

I think that the work partition is a difficult issue. It's hard to see the project as a whole and in parts, but the most important thing is communication. For example, instant messaging is the most powerful and helpful for of communication so far. It moves ideas and code along quickly, I recommend it for other groups.

Tanya

Alex:

There are two kinds of lessons that I learned and mastered during the duration of the projects. The first lesson, and probably, the hardest one, is organization. Initially, it was very hard for us to work and synergize together due to the different educational and cultural backgrounds. But once we had ironed out all the kinks, we settled into a predictable routine, thereby accomplishing the given project in a quick fashion. The moral lesson to be extracted is that communication leads to organization, and that, consequently, leads to results.

From the technical perspective, I've learned the beauty and fundamental simplicity that many languages exhibit. I always thought of the inherent intricacy and perplexity as a building block for a language, due to its awesome power. Moreover, writing our language gave me the opportunity to put theory, which I learned in the Computability class, into real-world practice.

Alex

Section 2: Future Work

If we were given more time, these would be things that we would improve in our language:

- We would like to have better error handling. The more and more we can check errors, the stronger our language would be.
- We would like to give programmers the ability to write and use libraries. A graphics language makes much more sense with a set of methods that can be included on demand. For example, the libraries can include methods that create often-used polygons or shapes. Also, it can include a set of pre-set colors like Red, Green, Black, White, Magenta, etc. Other libraries can deal with midpoints of lines, distances between points, and so on.
- We would like to consider a polygon type. Due to the way a closed polygon is created in BATS, the interpreter can notice that a line closes in on itself, and it can automatically give it polygon status. Of course, with a polygon type, would want to be able to fill in that polygon with a specific color, shrink it, enlarge it, or change the dimension of the polygon.
- We would like the ability to include Text of different sizes and fonts in our language.
- We would like to include a braced or bracketed statement that directly implements java code. We saw this in the Pencil language, and it seemed very attractive to us. It's an easy to use Java's massive library without implementing all of it.

Section 3: Advice to Future Teams

To future teams, we would like to give the following advice:

- Start early. Make the work easier on yourself by giving yourself more time to code and test your ideas.
- Have a strong channel of communication. The speed at which our inboxes were growing was unbelievable in the last few weeks. Also, use a messenger service like MSN™ of AOL Instant Messenger™
- Have faith in your group members, for they will always pull through. Keep track of them using a deadline system, but never try to control their creativity.
- Never be afraid to ask questions from teaching assistants, the professor, or other group members.
- Be open to ideas from all group members. Even though the team is recommended to be a dictatorship, recommendations and changes should be welcomed.
- Work together. The more often the group works together, the less discrepancies there are. This is extremely important, because it reduces the overall obscurity of the language. It even helps in giving more knowledge of the language to all group members. If a group starts early, this point can be easily followed.

Appendix: Code Listing

Explanation of Code Listing:

The code listing occurs in two parts: One part is the Textual Translator; the other part is the rest of the software. These two parts should be saved in different folders.

Textual Translator:

Bats.java:

```
/*
Author: Aleksandr Borovinskiy
Content: GraphicalOUTPUT ON THE FRAME: Bats class
*/

import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;


class Bats{

 static Vector figures=new Vector();


  Bats ()
  {
      BatsGraphics  BG= new BatsGraphics(figures);
    JFrame appFrame= new JFrame(" -Bats Builder-");
    appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    appFrame.setContentPane(BG);
    appFrame.pack();
      appFrame.show();

  }




  public static void main(String[] args){


           String filename="input";
```

```
           try{
                 BufferedReader reader=new BufferedReader(new
FileReader(filename));
               String strFile=new String();
                 for (String
line=reader.readLine();line!=null;line=reader.readLine())
                      strFile+=line+" ";

                 StringTokenizer st=new StringTokenizer(strFile);
               boolean indexFlag=false;

             while(st.hasMoreTokens())
                 {
                   String  temp=st.nextToken();
                       int index=Integer.parseInt(temp);



                       if(index==0)
                       {
                            BatsPoint  bp= new
BatsPoint(Integer.parseInt(st.nextToken()),Integer.parseInt(st.nextToke
n()));
                            figures.addElement(bp);
                       }

                       else if (index==1)
                       {
                            BatsLine bl=new BatsLine();

while(((temp=st.nextToken()).equals("|"))==false)
                               bl.addPoint( new BatsPoint (
Integer.parseInt(temp) ,Integer.parseInt (st.nextToken()) ) );
                              figures.addElement(bl);
                       }
                       else if (index==2)
                       {
                            int r=Integer.parseInt(st.nextToken()),
b=Integer.parseInt(st.nextToken());
                            int g=Integer.parseInt(st.nextToken());
                            figures.addElement(new BatsColor(r,b,g));

                       }
                       while((temp.equals("|"))==false)
                         temp=st.nextToken();
                }




           }
           catch (Exception ioe){
              System.err.println("Error reading file:" + ioe);
              System.exit(1);
           }
           Enumeration enum=figures.elements();
           while(enum.hasMoreElements())
```

```
            {
                  Object ob=enum.nextElement();
                  if(ob.toString().equals("BatsPoint")){
                        BatsPoint b=(BatsPoint)ob;
                        int p1=b.getXValue(), p2=b.getYValue();

                        System.out.print("0 ");
                        System.out.print(p1);
                        System.out.print(" ");
                        System.out.println(p2);
                     }
                  if(ob.toString().equals("BatsColor")){
                      BatsColor b=(BatsColor)ob;
                      int p1=b.getRValue(), p2=b.getBValue(),
p3=b.getGValue();

                  System.out.print("2 ");
                        System.out.print(p1);
                    System.out.print(" ");
                  System.out.print(p2);
                      System.out.print(" ");
                  System.out.println(p3);
                   }


                      if(ob.toString().equals("BatsLine")){
                         BatsLine bl=(BatsLine)ob;
                         System.out.print("1 ");
                   Enumeration e=bl.getLine().elements();
                     while(e.hasMoreElements())
                         {
                            BatsPoint b=(BatsPoint)e.nextElement();
                            int p1=b.getXValue(), p2=b.getYValue();
                      System.out.print(p1);
                      System.out.print(" ");
                      System.out.print(p2);
                        System.out.print(" ");
                   }
                          System.out.println();

                       }
             }


          new Bats();


     }

 }



 class BatsGraphics extends JPanel
 {
        private Vector figures=new Vector();
```

```
        public BatsGraphics(Vector figures)
        {

               this.figures=figures;
                 setPreferredSize(new Dimension(426,425));

        }



  public void paint(Graphics g)
  {

       g.setColor(Color.black);
       int i=0;

      g.drawLine(25,0,25,400);//g.drawLine(0,0,0,400);
      g.drawLine(25,400,425,400); //g.drawLine(0,400,400,400);
       while(i<=400)
       {

            g.drawLine(25,i,27,i);//g.drawLine(0,i,2,i);
            if (i==0)
               g.drawString("0",10,410);

            else if(i%40==0){
               g.drawString(String.valueOf(i),i+20,420);
               g.drawString(String.valueOf(i),0,405-i);
            }
            g.drawLine(i+25,400,i+25,398);//g.drawLine(i,399,i,397);
            i+=20;
       }

/*================================================*/
          Enumeration enum=figures.elements();
               while(enum.hasMoreElements())
               {
                         Object ob=enum.nextElement();
                         if(ob.toString().equals("BatsPoint"))
                         {
                                 BatsPoint b=(BatsPoint)ob;
                                 int p1=b.getXValue(),
p2=b.getYValue();
                              g.drawOval(p1+25,400-p2,1,1);
//g.drawOval(p1,400-p2,1,1);

                         }
                         if(ob.toString().equals("BatsColor"))
                         {
                               BatsColor b=(BatsColor)ob;
                               int p1=b.getRValue(), p2=b.getBValue(),
p3=b.getGValue();
                     g.setColor(new Color(p1,p2,p3));
                          }

                  if(ob.toString().equals("BatsLine"))
                  {
```

```
                                    BatsLine bl=(BatsLine)ob;

                            Enumeration e=bl.getLine().elements();
                                    int temp1=-1,temp2=-1;
                                while(e.hasMoreElements())
                                    {

                                       BatsPoint
b=(BatsPoint)e.nextElement();
                                       int p1=b.getXValue(),
p2=b.getYValue();
                                       if(temp1!=-1){
                                          g.drawLine(temp1+25,400-
temp2,p1+25,400-p2);
                                       }
                                     temp1=p1;
                                     temp2=p2;

                            }


                    }


        }

      }
```

BatsDataType.java:

```
import java.io.PrintWriter;
//import antlr.CommonAST;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * changed by Tanya
 * @version $Id: BatsDataType.java,v 1.17 2003/05/12 23:44:34 hanhua
Exp $
 */
public class BatsDataType
{
    String name;   // used in hash table
    String scope;
    String category;
    String dataType;
    String value;
    public void setDataType(String dt){
         this.dataType = dt;
    }
```

```
    public void setScope(String sc){
         this.scope = sc;
    }
    public String getDataType(){
         return this.dataType;
    }
    public void setCategory(String categ){
         this.category = categ;
    }
    public void setValue(String value){
         this.value = value;
    }
     public String getValue(){
         return this.value;
    }
    public String getScope(){
         return this.scope;
    }
    public String getName(){
         return this.name;
    }
     public String getCategory(){
         return this.category;
    }

    public BatsDataType() {
        name = null;
    }


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

/*   public String typename() {
        return "unknown";
    }

    public BatsDataType copy() {
        return new BatsDataType();
    }

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

    public BatsDataType error( String msg ) {
        throw new BatsException( "illegal operation: " + msg
                                 + "( <" + typename() + "> "
                                 + ( name != null ? name : "<?>" )
                                 + " )" );
    }

    public BatsDataType error( BatsDataType b, String msg ) {
        if ( null == b )
            return error( msg );
        throw new BatsException(
```

```
        "illegal operation: " + msg
        + "( <" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " )" );
}

public void print( PrintWriter w ) {
    if ( name != null )
        w.print( name + " = " );
    w.println( "<undefined>" );
}

public void print() {
    print( new PrintWriter( System.out, true ) );
}

public void what( PrintWriter w ) {
    w.print( "<" + typename() + ">  " );
    print( w );
}

public void what() {
    what( new PrintWriter( System.out, true ) );
}

public BatsDataType assign( BatsDataType b ) {
    return error( b, "=" );
}

public BatsDataType transpose() {
    return error( "\'" );
}

public BatsDataType uminus() {
    return error( "-" );
}

public BatsDataType plus( BatsDataType b ) {
    return error( b, "+" );
}

public BatsDataType add( BatsDataType b ) {
    return error( b, "+=" );
}

public BatsDataType minus( BatsDataType b ) {
    return error( b, "-" );
}

public BatsDataType sub( BatsDataType b ) {
    return error( b, "-=" );
}

public BatsDataType times( BatsDataType b ) {
```

```
    return error( b, "*" );
}

public BatsDataType mul( BatsDataType b ) {
    return error( b, "*=" );
}

public BatsDataType lfracts( BatsDataType b ) {
    return error( b, "/" );
}

public BatsDataType rfracts( BatsDataType b ) {
    return error( b, "/\'" );
}

public BatsDataType ldiv( BatsDataType b ) {
    return error( b, "/=" );
}

public BatsDataType rdiv( BatsDataType b ) {
    return error( b, "/\'=" );
}

public BatsDataType modulus( BatsDataType b ) {
    return error( b, "%" );
}

public BatsDataType rem( BatsDataType b ) {
    return error( b, "%=" );
}

public BatsDataType gt( BatsDataType b ) {
    return error( b, ">" );
}

public BatsDataType ge( BatsDataType b ) {
    return error( b, ">=" );
}

public BatsDataType lt( BatsDataType b ) {
    return error( b, "<" );
}

public BatsDataType le( BatsDataType b ) {
    return error( b, "<=" );
}

public BatsDataType eq( BatsDataType b ) {
    return error( b, "==" );
}

public BatsDataType ne( BatsDataType b ) {
    return error( b, "!=" );
}

public BatsDataType and( BatsDataType b ) {
    return error( b, "and" );
```

```
    }

    public BatsDataType or( BatsDataType b ) {
        return error( b, "or" );
    }

    public BatsDataType not() {
        return error( "not" );
    }*/
}
```

BatsPoint.java:

```
//import antlr.CommonAST;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * changed by Tanya
 * @version $Id: BatsDataType.java,v 1.17 2003/05/12 23:44:34 hanhua
Exp $
 */
public class BatsPoint
    extends BatsDataType {
  String name; // used in hash table
  String scope;
  String category;
  String dataType;
  String value;
  int xValue, yValue;

  public BatsPoint() {
    this.xValue = 0;
    this.yValue = 0;
  }

  public BatsPoint(BatsDataType bd) {
    this.xValue = 0;
    this.yValue = 0;
  }

  public BatsPoint(int x, int y) {
    this.xValue = x;
    this.yValue = y;
  }

  public void setXValue(int x) {
    this.xValue = x;
  }
```

```
  public int getXValue() {
    return this.xValue;
  }

  public void setYValue(int y) {
    this.yValue = y;
  }

  public int getYValue() {
    return this.yValue;
  }

  public void setDataType(String dt) {
    this.dataType = dt;
  }

  public void setScope(String sc) {
    this.scope = sc;
  }

  public String getDataType() {
    return this.dataType;
  }

  public void setCategory(String categ) {
    this.category = categ;
  }

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

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

  public String getScope() {
    return this.scope;
  }

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

  public String getCategory() {
    return this.category;
  }

  public String toString() {
    return "BatsPoint";
  }

}
```

BatsColor.java:

```
// @author: Tanya

import java.io.PrintWriter;

public class BatsColor extends BatsDataType
{

    String name;   // used in hash table
    String scope;
    String category;
    String dataType;
    private  int r, b, g;

   public BatsColor( ) {
        this.r = 0;
        this.b = 0;
        this.g = 0;
    }
   public BatsColor( BatsDataType bd ) {
      this.r = 0;
      this.b = 0;
      this.g = 0;

    }
    public BatsColor( int r, int b, int g) {
      this.r = r;
      this.b = b;
      this.g = g;


    }

    public void setRBG(int r,int b, int g){
        this.r = r;
      this.b = b;
      this.g = g;

    }
    public int getRValue(){
         return this.r;
    }
    public int getBValue(){
         return this. b;
    }
    public int getGValue(){
         return this.g;
    }
    public void setDataType(String dt){
         this.dataType = dt;
    }
    public void setScope(String sc){
         this.scope = sc;
    }
```

```
    public String getDataType(){
         return this.dataType;
    }
    public void setCategory(String categ){
         this.category = categ;
    }
     public void setValue(String value){
         this.value = value;
    }
     public String getValue(){
         return this.value;
    }
    public String getScope(){
         return this.scope;
    }
    public String getName(){
         return this.name;
    }
     public String getCategory(){
         return this.category;
    }

   public String toString(){ return "BatsColor";}

}
```

BatsLine.java:

```
// @author: Tanya

import java.io.PrintWriter;
//import antlr.CommonAST;
import java.util.*;

public class BatsLine extends BatsDataType
{
    String name;   // used in hash table
    String scope;
    String category;
    String dataType;
    String value;
    Vector points = new Vector();


    public BatsLine( ) {
    }
    public BatsLine( BatsDataType bd ) {
    }
    public BatsLine( BatsPoint bp ) {
         this.points.add(bp);
    }
    public void addPoint(BatsPoint bp){
            this.points.addElement(bp);
```

```
    }
    public void addPoints(Vector vbp){
         for(int i = 0; i < vbp.size(); i++){
            BatsPoint bp = (BatsPoint) vbp.elementAt(i);
            this.points.add(bp);
         }
     }
    public Vector getLine(){ return points;}

   public String toString(){ return "BatsLine";}

}
```

StartBats.bat:

javac *.java
java Bats

Rest of Compiler:

Bats.g:

/*
Author: Behrooz Badii
Content: BATS lexer and parser
*/

/*
Notes: This is basically a grammar, but in ANTLR, + means one or more
* means zero or more, ? means zero or one, ^ means make that token the root of
a subtree, and ! means eliminate the token from the AST
*/

options {
	language="Java";
}
class BATSLexer extends Lexer;
options {
	testLiterals = false;
	exportVocab = BATStokens;
	k = 2; }

/*The following are all keywords for easy reference*/
tokens {
	INT = "Int";
	LINESTR = "Line";
	POINTSTR = "Point";
	BOOLSTR = "Boolean";

```
        IFSTR = "If";
        THEN = "Then";
        ELSE = "Else";
        IFEND = "IfEnd";
        WHILESTR = "While";
        DO = "Do";
        WHILEEND = "WhileEnd";
        FORSTR = "For";
        FOREND = "ForEnd";
        FORLINESTR = "ForLine";
        IN = "In";
        XAXIS = "XAxis";
        YAXIS = "YAxis";
        RETURNSTR = "Return";
        COLORSTR = "Color";
        DRAWSTR = "Draw";
        APPENDSTR = "Append";
        TRUE = "True";
        FALSE = "False";
        LET = "Let";
        TO = "To";
        CALL = "Call";
        GLOBAL = "Global";
    END = "End";
        }

PLUS  : '+' ;
MINUS         : '-' ;
MUL   : '*' ;
DIV    : '/' ;
MOD   : '%' ;
ASSIGN        : "<-";
COMMA       : ',' ;
PERIOD        : '.' ;
GREATER     : '>' ;
LESS  : '<' ;
CARRAT       : '^' ;
GEQ   : ">=";
LEQ   : "<=";
AND   : "&&";
OR     : "||";
EQEQ : "==";
NEQ   : "!=";
LPARENS : '(' ;
RPARENS : ')' ;
SEMI  : ';' ;
```

```
LBRACE       : '{' ;
RBRACE       : '}' ;
BRACKET     : "[]";
LBRACKET: '[' ;
RBRACKET: ']' ;
NOT    : '!' ;


protected LETTER : ('a'..'z' | 'A'..'Z') ;
protected DIGIT : '0'..'9' ;

ID options { testLiterals = true; }
: LETTER (LETTER | DIGIT | '_')* ;

INTEGER : (DIGIT)+ ;
/*If it is a double, set the type of the token to DOUBLE*/

EXPONENT : ('e'|'E') ('+'|'-') DIGIT DIGIT ;

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

COMMENT:
'#' (~('\n' | '\r'))* ('\n' {newline();}| '\r' {newline();}) { $setType(Token.SKIP); };
/* comments start with # and end with a newline */

class BATSParser extends Parser;
options {
        k = 2;
        buildAST = true;
        importVocab = BATStokens;
        }

/*The following are the tokens created to flesh out the AST*/
/*Sui: changed the order of the tokens*/
tokens {
   PROGRAM;
        GLOBVARDECL;
   FUNCDECL;
   VARDECL;
        PARAMLIST;
        STMTBLOCK;
        ASSIGNSTMT;
        EXPR;
        POINT;
        LINE;
```

```
        ARGLIST;

        }

/*added the program rule to have an overall root*/
program :
        (globvarDecl)* (funcDecl)+  EOF!
  { #program = #([PROGRAM, "program"], #program); }
  ;

globvarDecl :
        GLOBAL type ID (COMMA! ID)* SEMI!
        { #globvarDecl = #([GLOBVARDECL, "globvarDecl"], #globvarDecl); }
        ;

/*took away the optoinal clauses*/
funcDecl:
        type ID LPARENS! paramlist RPARENS! varDecl stmtblock END!
        { #funcDecl = #([FUNCDECL,"funcDecl"], #funcDecl); }
        ;

varDecl :
        (type ID (COMMA! ID)* SEMI!)*
        { #varDecl = #([VARDECL, "varDecl"], #varDecl); }
        ;

paramlist       :
                (parameter (COMMA! parameter)*)?
                { #paramlist = #([PARAMLIST, "paramlist"], #paramlist); }
                ;

parameter       :
                type ID;

stmtblock       :
                (stmt)+
                { #stmtblock = #([STMTBLOCK, "stmtblock"], #stmtblock); }
                ;

stmt    :
        expr SEMI! | assignstmt SEMI! | ifBlock | whileBlock | forblock | returnstmt
SEMI! | builtinFunc SEMI!;
/*expressions are the only stmts that have fundamental types like Int, Double, or Line,
etc.*/

type    :
```

```
        (INT|LINESTR|POINTSTR|BOOLSTR) (BRACKET)?;

ifBlock :
        IFSTR^ expr THEN! stmtblock (ELSE! stmtblock)? IFEND!;

whileBlock      :
                WHILESTR^ expr DO! stmtblock WHILEEND!;

forblock        :
                forblockDefault | forblockLine;

forblockDefault         :
                FORSTR^ (assignstmt)? SEMI! (expr) SEMI! (assignstmt)? DO!
stmtblock FOREND!;

forblockLine    :
                FORLINESTR^ ID IN! ID DO! stmtblock FOREND!;

/*changed*/
assignstmt      :
                LET varaccess ASSIGN! relExpr
                { #assignstmt = #([ASSIGNSTMT, "assignstmt"], #assignstmt); }
                ;

varaccess       :
                ID (LBRACKET! expr RBRACKET!)? | ID PERIOD!
(XAXIS|YAXIS|INTEGER);

/*changed*/
returnstmt      :
                RETURNSTR^ relExpr;

builtinFunc     :
                colorFunc | drawFunc | appendFunc;

/*changed*/
colorFunc       :
                COLORSTR^ factor SEMI! factor SEMI! factor;

/*changed*/
drawFunc        :
                DRAWSTR^ (ID|point|line) (COMMA! (ID|point|line))*;

/*changed */
appendFunc      :
                APPENDSTR^ (ID|point|line) TO! (ID|point|line);
```

```
expr    :
        primaryExpr (primaryOp primaryExpr)?
        { #expr = #([EXPR, "expr"], #expr); }
        ;

primaryOp       :
                AND | OR;

primaryExpr     :
                eqeqExpr (eqeqOp eqeqExpr)?;

eqeqOp          :
        EQEQ | NEQ;

eqeqExpr:
        relExpr (relOp relExpr)?;

relOp   :
        LESS | LEQ | GREATER | GEQ;

relExpr:
        (MINUS)? term (addOp term)*;

addOp :
        PLUS | MINUS;

term    :
        factor (mulOp factor)*;

mulOp :
        MUL | DIV | MOD;

factor  :
        constant | varaccess | LPAREN! (constant|varaccess) addOp (constant|varaccess)
RPAREN! | funccall | NOT factor;


constant:
        INTEGER | bool | point | line;

bool    :
```

```
        TRUE|FALSE;

/*changed for new format*/
point   :
        CARRAT! (ID|INTEGER) COMMA! (ID|INTEGER) CARRAT!
        { #point = #([POINT, "point"], #point); }
        ;
/*changed to only take in points*/
line    :
        LBRACE! (ID|point) (COMMA! (ID|point))+ RBRACE!
        { #line = #([LINE, "line"], #line); }
        ;

funccall:
        CALL^ ID LPARENS! (arglist)? RPARENS! SEMI!;

arglist  :
        (constant|ID) (COMMA! (constant|ID))*
        { #arglist = #([ARGLIST, "arglist"], #arglist); }
        ;
```

BatsWalker.g:

```
/*
 * Batswalker.g : the AST walker.
 *
 * @author Sui
 *
 * @version $Id: walker.g,v 1.24 2003/05/12 21:42:43 hanhua Exp $
 */

{
import java.io.*;
import java.util.*;
import antlr.CommonAST;
}

class BatsWalker extends TreeParser;
options{
   buildAST = true;
   importVocab = BATStokens;
}

{
```

```
//  BatsInterpreter biptr = new BatsInterpreter();
 // BatsBuiltIn builtIn = new BatsBuiltIn();
//  BatsSymbolT smt = new BatsSymbolT();
}

program returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(PROGRAM (a=globvarDecl)* (b=funcDecl)+ (c=funccall)* );

globvarDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(GLOBVARDECL GLOBAL  t:type v=var );

funcDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(FUNCDECL   t:type  funName:ID  a=paramlist (b=varDecl)* (a=stmtblock ));

varDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(VARDECL (t:type (id:ID )+ )*);

paramlist returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
```

```
    a = new CommonAST();

}: #(PARAMLIST  (pt:type pn:ID )*);

type: (INT|LINESTR|POINTSTR|BOOLSTR);

stmtblock returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();

}: #(STMTBLOCK (a=stmt)+);

stmt returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();

}: a=assignstmt
| a=expr
| a=returnstmt
| a=ifBlock
| a=whileBlock
| a=forBlock
| a=builtinFunc
;

assignstmt returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();


}:#(ASSIGNSTMT LET b=varaccess  a=relExpr );

varaccess returns [CommonAST r]
  {
     CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
```

```
    a = new CommonAST();

   }
   : #(id:ID  (XAXIS|YAXIS|INTEGER|(a=expr)?));


ifBlock returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();
   boolean res = false;

}:#(IFSTR  a=expr  ( b=stmtblock)(c=stmtblock)?);

whileBlock returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();
   boolean res = false;

}:#(WHILESTR  a=expr ( b=stmtblock) );

forBlock returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(FORSTR (b=assignstmt)? a=expr (b=assignstmt)? c=stmtblock)
| #(FORLINESTR ID ID a=stmtblock);

returnstmt returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(RETURNSTR a=relExpr  );

builtinFunc      returns [ CommonAST r ]
```

```
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:a=colorFunc
| a=drawFunc
| a=appendFunc
;

colorFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(COLORSTR a=relExpr  b=relExpr  c=relExpr );

drawFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(DRAWSTR (id:ID|p:point)+ );

appendFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(APPENDSTR (ID|point|line) (ID|point|line));



expr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();
```

```
}:#(EXPR a=primaryExpr  (p:primaryOp  b=primaryExpr)?);

primaryOp:AND|OR;

primaryExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:a=eqeqExpr  (b=eqeqOp c=eqeqExpr)?;

eqeqExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: b=relExpr (relOp a=relExpr)?;

eqeqOp returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: a=relExpr (relOp b=relExpr)?;

relOp{
  String ro = "";
}
: (LESS)|(LEQ)|(GREATER)|(GEQ) ;

relExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: a=term (addOp   b=term )*  ;
```

```
addOp: (PLUS)|(MINUS)
  ;

term returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:  a=factor  (mulOp b=factor)*;

mulOp:  (MUL)|(DIV)|(MOD);

factor returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();
   boolean cmp = false;
}:  (((constant |id:ID) (options {greedy=true;}:(addOp (constant|ID)))*)|b=funccall|NOT
b=factor);

constant: ( INTEGER |b:bool|p:point|l:line )

 ;

bool: (TRUE | FALSE ) ;

point{
   String x = "";
   String y = "";
}: #(POINT  (INTEGER|ID ) (INTEGER|ID) );

line: #(LINE (ID |point ) (ID|mp:point )+);

funccall returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(CALL {System.out.println("CALL");}fc:ID  a=arglist);
```

```
arglist returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(ARGLIST (arg:ID|constant)+);



var returns [ Vector v ]
{
  v=new Vector();
}
:(p:ID { v.add( p.getText() ); } )+
;
```

preBatsWalker.g: //this is before changes

```
/*
 * Batswalker.g : the AST walker.
 *
 * @author Tanya, Sui
 *
 * @version $Id: walker.g,v 1.24 2003/05/12 21:42:43 hanhua Exp $
 */

{
import java.io.*;
import java.util.*;
import antlr.CommonAST;
}

class BatsWalker extends TreeParser;
options{
   buildAST = true;
   importVocab = BATStokens;
}

{
  BatsInterpreter biptr = new BatsInterpreter();
  BatsBuiltIn builtIn = new BatsBuiltIn();
//  BatsSymbolT smt = new BatsSymbolT();
}
```

```
program returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(PROGRAM (a=globvarDecl)* (b=funcDecl)+ (c=funccall)* {biptr.drawPicture();
biptr.checkBegin(); biptr.printHash();});

globvarDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(GLOBVARDECL GLOBAL ( { biptr.setScope("Global"); } ) t:type {
biptr.setDataType(t.getText());} v=var { biptr.setGlobalVariables(v);});

funcDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(FUNCDECL {biptr.setScope("Function"); biptr.funcStart=true;}  t:type {
            biptr.setDataType(t.getText());} funName:ID {
            biptr.setFunVar(funName.getText(),"fun");} a=paramlist (b=varDecl)*
(a=stmtblock ){
            biptr.funcEnd(); biptr.setScope("Global"); });

varDecl returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(VARDECL (t:type {biptr.setDataType(t.getText());} (id:ID  {
biptr.setFunVar(id.getText(),"var"); })+ )*);

paramlist returns [ CommonAST r ]
{
```

```
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(PARAMLIST  (pt:type {biptr.setDataType(pt.getText());} pn:ID
{biptr.setFunVar(pn.getText(),"parameter");})*);

type: (INT|LINESTR|POINTSTR|BOOLSTR);

stmtblock returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(STMTBLOCK (a=stmt)+);

stmt returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: a=assignstmt
| a=expr
| a=returnstmt
| a=ifBlock
| a=whileBlock
| a=forBlock
| a=builtinFunc
;

assignstmt returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();


}:#(ASSIGNSTMT LET {biptr.startLet(); } b=varaccess  a=relExpr {
     if(biptr.mathSign != null && !biptr.mathSign.equals(""))
     { biptr.mathOper(); biptr.assign();} biptr.endLet();});
```

```
varaccess returns [CommonAST r]
  {
     CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();

   }
   : #(id:ID  { biptr.setLvalue(id.getText());  } (XAXIS|YAXIS|INTEGER|(a=expr)?));


ifBlock returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();
    boolean res = false;

}:#(IFSTR  {biptr.startWhile();} a=expr {res = biptr.compare(); } ({if(res)}
b=stmtblock)(c=stmtblock)?);

whileBlock returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();
    boolean res = false;

}:#(WHILESTR { biptr.startWhile();} a=expr { res = biptr.compare(); }( b=stmtblock)
{biptr.endWhile();});

forBlock returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();

}:#(FORSTR (b=assignstmt)? a=expr (b=assignstmt)? c=stmtblock)
| #(FORLINESTR ID ID a=stmtblock);

returnstmt returns [ CommonAST r ]
{
```

```
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(RETURNSTR {biptr.startReturn=true; }a=relExpr {biptr.startReturn=false;} );

builtinFunc     returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:a=colorFunc
| a=drawFunc
| a=appendFunc
;

colorFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(COLORSTR { biptr.startColor("red");} a=relExpr {biptr.startColor("green");}
b=relExpr { biptr.startColor("blue");} c=relExpr {biptr.appendColor();});

drawFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(DRAWSTR (id:ID|p:point)+ { biptr.draw(id.getText());});

appendFunc returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(APPENDSTR (ID|point|line) (ID|point|line));
```

```
expr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:#(EXPR a=primaryExpr  (p:primaryOp  b=primaryExpr)?);

primaryOp:AND|OR;

primaryExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:a=eqeqExpr  (b=eqeqOp c=eqeqExpr)?;

eqeqExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: b=relExpr (ro:relOp { biptr.relOp(ro.getText());  if(biptr.startWhile)
biptr.setLeft(biptr.getRvalue());
            } a=relExpr)?;

eqeqOp returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: a=relExpr (relOp b=relExpr)?;

relOp{
  String ro = "";
}
```

```
: (ls:LESS { if(ls.getText() != null) biptr.setRelOp(ls.getText());
   })|(le:LEQ {if(le.getText() != null) ro = le.getText();
   } )|(gr:GREATER {if(gr.getText() != null) ro = gr.getText();}
   )|(ge:GEQ {if(ge.getText() != null) ro = ge.getText();}) ;

relExpr returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: a=term (addOp   b=term )*  ;

addOp: (p:PLUS { if(p != null && p.getText() != null) biptr.mOp(p.getText());
     })|(m:MINUS { if(m != null && m.getText() != null) biptr.mOp(m.getText());
       })
 ;

term returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}:  a=factor  (mulOp b=factor)*;

mulOp:  (ml:MUL  { if(ml != null && ml.getText() != null) biptr.mOp(ml.getText());
     })|(d:DIV  { if(d != null && d.getText() != null) biptr.mOp(d.getText());
     })|(md:MOD  { if(md != null && md.getText() != null) biptr.mOp(md.getText());
     });

factor returns [ CommonAST r ]
{
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();
   boolean cmp = false;
}:  ((( con:constant { biptr.setRvalue(con.getText());
 if(biptr.startWhile){ biptr.setRight(con.getText()); cmp = biptr.compare(); }
biptr.assign();
 }|id:ID { biptr.setRvalue(id.getText());
 if(biptr.startWhile)  { biptr.setWleft(id.getText());
 }
```

```
 else {

 biptr.setBoolVar(id.getText());
 biptr.assign();

 }
 if(biptr.startReturn) biptr.setReturn(id.getText());} ) (options  {greedy=true;}:(addOp (
constant|ID)))*)|b=funccall|NOT b=factor);

constant: ( i:INTEGER { if(i != null && i.getText() != null && biptr.mathSign != null ) {
          try{if(!biptr.rValue.equals("")) biptr.leftOper = Integer.parseInt(biptr.rValue);
}catch(Exception e){}
          biptr.rightOper = Integer.parseInt(i.getText());
           biptr.mathOper();  biptr.assign(); }
           }|b:bool|p:point|l:line
         )

 ;

bool: (t:TRUE {biptr.setBoolVar(t.getText());} | f:FALSE
{biptr.setBoolVar(f.getText());} )({if(!biptr.funcName.equals(""))  biptr.assignFunVar();
else biptr.assignGlobVar();}) ;

point{
   String x = "";
   String y = "";
}: #(POINT  (a:INTEGER|id1:ID ) (b:INTEGER|id2:ID){ if(a != null && a.getText() !=
null) x = a.getText(); else x = id1.getText();
if(b != null && b.getText() != null) y = b.getText(); else y = id2.getText();
biptr.setPointVar(x,y);} );

line: #(LINE {biptr.startLine();} (id:ID { biptr.setLineVar(id.getText());}|p:point {
biptr.startLine();  biptr.mPoint();}) (mid:ID{
biptr.setMultiPointLine(mid.getText());}|mp:point )+);

funccall returns [ CommonAST r ]
{
    CommonAST a,b,c;
    Vector v = new Vector();
    r = null;
    a = new CommonAST();

}:#(CALL {System.out.println("CALL");}fc:ID { biptr.funCall(fc.getText());} a=arglist);

arglist returns [ CommonAST r ]
{
```

```
   CommonAST a,b,c;
   Vector v = new Vector();
   r = null;
   a = new CommonAST();

}: #(ARGLIST (arg:ID{biptr.funCallArgs(arg.getText());}|constant)+);



var returns [ Vector v ]
{
  v=new Vector();
}
:(p:ID { v.add( p.getText() ); } )+
;
```

BatsBoolean.java:

```
import java.io.PrintWriter;
import antlr.CommonAST;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * changed by Tanya
 * @version $Id: BatsDataType.java,v 1.17 2003/05/12 23:44:34 hanhua
Exp $
 */
public class BatsBoolean extends BatsDataType
{
   boolean value;
    public BatsBoolean( ) {
        super.value = "false"; //default is false
    }
    public BatsBoolean(String value ) {
        super.value = value;
    }
     public BatsBoolean(boolean value ) {
        this.value = value;
    }

    public BatsBoolean(  String name,String scope) {
        super(name,scope,"Boolean","var");

    }
    public boolean getBoolValue(){
       return this.value;
    }
    public String toString(){
```

```
       return super.getValue();
    }
}
```

BatsBuiltIn.java:

```
import java.io.*;
import java.util.*;
/**
*  The class is created to implement builtin Bats functions such as
draw, append and color.
*/
public class BatsBuiltIn {
   Vector drawLine;
   FileOutputStream fos;
   DataOutputStream dos;
   public BatsBuiltIn (){
      this.drawLine = new Vector();
    try{ //create a chain
      this.fos = new FileOutputStream("input");
      this.dos = new DataOutputStream(fos);
    }catch(IOException e){System.out.println("IOException
"+e.toString());}
   }

    public void drawLine(BatsLine bl){
      try{

            Vector blPoints = bl.getLine();
            dos.writeBytes("1 ");  //BatsLine drawing

            for (int i = 0; i < blPoints.size(); i++){

               BatsPoint bp = (BatsPoint)blPoints.elementAt(i);

               String xVal = new Integer(bp.getXValue()).toString()+"
";

               String yVal = new Integer(bp.getYValue()).toString();
    System.out.println("++||++||++|| BUILT IN DRAW i "+i+" "+" xVal
"+xVal+" yVal"+yVal);
               dos.writeBytes(xVal);
               dos.writeBytes(yVal);
               dos.writeBytes(" ");
            }
         dos.writeBytes("|\n");

      }catch(IOException e){System.out.println("IOException
"+e.toString());}
   }

 public void drawLine(){
      try{

         for (int j = 0; j < this.drawLine.size(); j++){
```

```
            if(this.drawLine.elementAt(j) instanceof BatsLine ){
               BatsLine bl = (BatsLine)drawLine.elementAt(j);
               drawLine(bl);
            }//if

            if(this.drawLine.elementAt(j) instanceof BatsColor ){
               BatsColor bc = (BatsColor)this.drawLine.elementAt(j);
               drawColor(bc);
            }//if
             if(this.drawLine.elementAt(j) instanceof BatsPoint ){
                 BatsPoint bp = (BatsPoint)this.drawLine.elementAt(j);
                 drawPoint(bp);
            }//if
         }//for
     dos.close();
       }catch(IOException e){System.out.println("IOException
"+e.toString());}
 }

/**********************************************************************
**/
 public void drawPoint(BatsPoint bp){
      try{
         dos.writeBytes("0 ");  //BatsLine drawing
         String xVal = new Integer(bp.getXValue()).toString()+" ";
         String yVal = new Integer(bp.getYValue()).toString();

         dos.writeBytes(xVal);
         dos.writeBytes(yVal);
         dos.writeBytes("|\n");
      }catch(IOException e){System.out.println("IOException
"+e.toString());}
   }
/**********************************************************************
*/
 public void drawColor(BatsColor bc){
      try{
         dos.writeBytes("2 ");  //BatsColor drawing
         String rVal = new Integer(bc.getRValue()).toString()+" ";
         String gVal = new Integer(bc.getGValue()).toString()+ " ";
         String bVal = new Integer(bc.getBValue()).toString()+ " ";

         dos.writeBytes(rVal);
         dos.writeBytes(gVal);
         dos.writeBytes(bVal);
         dos.writeBytes("|\n");
      }catch(IOException e){System.out.println("IOException
"+e.toString());}
   }
/**********************************************************************
****/
   public void append(BatsLine bl){
      System.out.println("=========APPEND LINE");
     this.drawLine.add(bl);
   }
```

```
/**************************************************************************
***/
   public void append(BatsColor bc){
     System.out.println("======APPEND COLORR "+bc.getRValue() +" G
"+bc.getGValue()+" B "+bc.getBValue());
     this.drawLine.add(bc);
   }

/**************************************************************************
***/
   public void append(BatsPoint bp){
      System.out.println("=========APPEND COLOR");
      this.drawLine.add(bp);
   }

}
```

BatsColor.java:

```
import java.io.PrintWriter;
/**
 * The base Bats function data type class
 *
 * @author Alex, Tanya
 * @version $Id: BatsFunction.java,v 1.17 2004/12/18 23:44:34
 */
public class BatsColor extends BatsDataType
{

   private  int r, b, g;

   public BatsColor( ) {
        this.r = 0;
        this.b = 0;
        this.g = 0;
        super.setDataType("Color");
   }
   public BatsColor( BatsDataType bd ) {
      this.r = 0;
      this.b = 0;
      this.g = 0;

    }
    public BatsColor( int r, int g, int b) {
         this.r = r;
      this.g = g;
      this.b = b;

    }

    public void setRBG(int r,int g, int b){
        this.r = r;
           this.g = g;
        this.b = b;
```

```
    }
    public void setRValue(int r){
         this.r = r;
    }
    public void setBValue(int b){
         this.b = b;
    }
    public void setGValue(int g){
         this.g = g;
    }
    public int getRValue(){
         return this.r;
    }
    public int getBValue(){
         return this.b;
    }
    public int getGValue(){
         return this.g;
    }

}
```

BatsDatatype.java:

```
import java.io.PrintWriter;
import antlr.CommonAST;

/**
 * The base Bats function data type class
 *
 * @author Tanya
 * @version $Id: BatsFunction.java,v 1.17 2004/12/18 23:44:34 hanhua
Exp $
 */
public class BatsDataType
{
    String name;   // used in hash table
    String scope;
    String category;
    String dataType;
    String value;
    int paramOrder;

    public BatsDataType(  ) {
        this.name = "";
    }
    public BatsDataType( String name ) {
        this.name = name;
    }
    public BatsDataType( String name, String scope,String dataType,
String category ) {
        this.name = name;
        this.scope = scope;
        this.dataType = dataType;
        this.category = category;
    }
```

```
    public void setParamOrder(int po){
         this.paramOrder = po;
    }
    public int getParamOrder(){
         return this.paramOrder;
    }
    public void setDataType(String dt){
         this.dataType = dt;
    }
    public void setScope(String sc){
         this.scope = sc;
    }
    public void SetDataType(String dt){
         this.dataType = dt;
    }
    public String getDataType(){
         return this.dataType;
    }
    public void setCategory(String categ){
         this.category = categ;
    }
     public String getCategory(){
         return this.category;
    }
    public void setValue(String value){
         this.value = value;
    }
     public String getValue(){
         return this.value;
    }
    public String getScope(){
         return this.scope;
    }
     public void setName(String n){
         this.name = n;
    }
    public String getName(){
         return this.name;
    }

}
```

BatsFunction.java:

```
import java.io.PrintWriter;
import antlr.CommonAST;
import java.util.*;

/**
 * The base Bats function data type class
 *
 * @author Tanya
 * @version $Id: BatsFunction.java,v 1.17 2004/12/18 23:44:34 hanhua
Exp $
 */
```

```
public class BatsFunction extends BatsDataType
{
    String name;   // used in hash table
    String scope;
    String category;
    String dataType;
    String returnType;
    String returnName;
    BatsDataType returnValue = new BatsDataType();
    public  Hashtable parameters = new Hashtable();
    public  Hashtable variables = new Hashtable();

    public BatsFunction() {
        this.name = "";
        this.scope = "";
        this.category = "";
        this.dataType = "";
        this.returnType = "";
    }
    public BatsFunction( String name ) {
        this.name = name;
        this.scope = "";
        this.category = "";
        this.dataType = "";
        this.returnType = "";

    }
    public void setReturnName(String rn){
         this.returnName = rn;
    }
    public String getReturnName(){
         return this.returnName;
    }
    public void setReturnType(String rt){
         this.returnType = rt;
    }
    public void setReturnValue(BatsDataType rv){
         this.returnValue = rv;
    }
    public BatsDataType getReturnValue(){
         return this.returnValue;
    }
    public String getReturnType(){
         return this.returnType;
    }

    public Hashtable getVariables(){
         return this.variables;
    }

}
```

BatsInteger.java:

```
import java.io.PrintWriter;
import antlr.CommonAST;
/**
```

```
 * The base Bats function data type class
 *
 * @author Tanya
 * @version $Id: BatsFunction.java,v 1.17 2004/12/18 23:44:34 hanhua
Exp $
 */
public class BatsInteger extends BatsDataType
{

    public BatsInteger( ) {
        super.value = "0";
    }

    public BatsInteger( String name,String scope) {
        super(name,scope,"Int","var");
    }
    public int getInt(){
System.out.println("super.vallue "+super.value);
      return Integer.parseInt(super.value);
    }

  }
```

BatsInterpreter.java:

```
import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


/** Interpreter routines that is called directly from the tree walker.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: BatsInterpreter.java,v 1.27 2003/05/12 21:42:43 hanhua
Exp $
 */
public class BatsInterpreter {
    BatsBuiltIn builtIn = new BatsBuiltIn();
    Hashtable varst = new Hashtable();
    Hashtable funst = new Hashtable();
    boolean
beginFun,funcStart,let,startLine,startReturn,mPoints,startColor,startWh
ile,
            loopProceed;
    String dataType,scope,category,funcName,rValue,
            lValue,rbg,wLeft,relOp,wRight,mathSign,relSign,conSign;
    BatsFunction funCall = new BatsFunction();
    int leftOper, rightOper, rConValue;
    BatsBoolean boolVar = new BatsBoolean();
    BatsPoint pointVar = new BatsPoint();
    BatsLine lineVar = new BatsLine();
```

```
    Vector linePoints = new Vector();
    BatsColor bColor = new BatsColor();
    String msg = "";
    Vector lVector = new Vector();

   public BatsInterpreter() {
       this.dataType="";
       this.scope="";
       this.category="";
       this.funcName="";
   }
   public void funCall(String funCallName){
      if(funst.containsKey(funCallName)){

         funCall = (BatsFunction) funst.get(funCallName);
      }
      else{
         printError("Function call name is not recognized.");
      }
   }
    public void funCallArgs(String callArgName){
     boolean paramFound = false;
     String paramDT = "";
     BatsDataType param = new BatsDataType();
     if(funcName.equals("")){
       if(varst.containsKey(callArgName)){
         param = (BatsDataType)varst.get(callArgName);
         paramDT = param.getDataType();
       }

     }
     else if(funst.containsKey(funcName)){
         BatsFunction bf =  (BatsFunction) funst.get(funcName);
         param = (BatsDataType)bf.variables.get(callArgName);
         paramDT = param.getDataType();

     }

     for (Enumeration e = funCall.parameters.elements() ;
e.hasMoreElements() ;) {
          BatsDataType prm = (BatsDataType) e.nextElement();

          if(prm.getDataType().equals(paramDT) && prm.getParamOrder()
== param.getParamOrder()){

             paramFound = true;
             if(funst.containsKey(funCall.getName())){
               funst.put(funCall.getName(),param);
             }
             break;
          }
      }//for

      if(!paramFound){
         printError("Parameter is not defined in signature for the
function call.");
      }
```

```
   }
   public void mPoint(){

      this.mPoints = true;
   }
   public void setBoolVar(String boolV){
      this.boolVar = new BatsBoolean(boolV);
   }
   public void  setWleft(String wLeft){
      if(startWhile && !wLeft.equals("")){
         this.wRight = wLeft;
      }
      else
         this.wLeft = wLeft;
   }
   public void startLine(){
      startLine = true;
   }
   public void setLineVar(BatsPoint bp){
      this.lineVar = new BatsLine(bp);
   }
   public void setLineVar(String pointName){

      BatsPoint bp = new BatsPoint();
      if(varst.containsKey(pointName)){

         bp = (BatsPoint) varst.get(pointName);
         this.lineVar = new BatsLine(bp);

      }
      else if(!funcName.equals("") && funst.containsKey(funcName)){
           BatsFunction funSymT = (BatsFunction) funst.get(funcName);
           if(funSymT.variables != null &&
(funSymT.getVariables().containsKey(pointName))){
               bp = (BatsPoint) funSymT.getVariables().get(pointName);

               this.lineVar = new BatsLine(bp);
           }//if
           else { //Error pointName does not exist
               msg = "Variable "+pointName+" is not defined";
               printError(msg);
           }
      }//else if

   }
   /*****************************************************************/
   public void setPointVar(String x, String y){
     int X = 0;
     int Y = 0;
     try{
       X = Integer.parseInt(x);
     }catch(NumberFormatException e){
       X = getVarValue(x);
     }
     try{
       Y = Integer.parseInt(y);
     }catch(NumberFormatException e){
```

```
       Y = getVarValue(y);
     }

      this.pointVar = new BatsPoint(X, Y);
     if(startLine){
      //if the point is not the first in line
        for(int i = 0; i < this.lineVar.getLine().size(); i++){
            BatsPoint bp = (BatsPoint)
this.lineVar.getLine().elementAt(i);
            this.linePoints.add(bp);

        }
         this.lineVar = new BatsLine();
         this.linePoints.add(this.pointVar);

      }

   }

/**********************************************************************
******/
 public int getVarValue(String xy){
   int res = 0;
   BatsDataType bdt = new BatsDataType();
    if(!funcName.equals("") && funst.containsKey(funcName)){
           BatsFunction funSymT = (BatsFunction) funst.get(funcName);
           if(funSymT.variables != null &&
funSymT.variables.containsKey(xy)){
              bdt = (BatsDataType) funSymT.getVariables().get(xy);

               if(bdt.getDataType().equals("Int")){
                  res = Integer.parseInt(bdt.getValue());
                  return res;
               }
           }
      }
    if(varst != null && varst.containsKey(xy)){
        bdt = (BatsDataType) varst.get(xy);
        if(bdt.getDataType().equals("Int")){
           res = Integer.parseInt(bdt.getValue());
        }
    }
            return res;
 }
/*****************************************************************/
   public void setMultiPointLine(String mpName){
       if(!funcName.equals("") && funst.containsKey(funcName)){
           BatsFunction funSymT = (BatsFunction) funst.get(funcName);
           if(funSymT.variables != null &&
(funSymT.getVariables().containsKey(mpName))){
              Hashtable funVar = funSymT.getVariables();
              addPointsToLine(funVar, mpName);
              return;
           }
       }
       if (varst.containsKey(mpName)){
              addPointsToLine(varst, mpName);
```

```
        }
    }

    public void addPointsToLine(Hashtable vfHash, String mpName){
        BatsDataType bdt = (BatsDataType) vfHash.get(mpName);
        if(bdt.getDataType().equals("Point")){
             bdt = (BatsPoint) vfHash.get(mpName);
             for(int i = 0; i < this.lineVar.getLine().size(); i++){
                BatsPoint bp = (BatsPoint)
this.lineVar.getLine().elementAt(i);
                this.linePoints.add(bp);
            }
             this.linePoints.add(bdt);

          }else{
             printError("Unable to set any bats data type other than
Point.");
          }
    }
/**
* The method sets used to check the function return value
*/

    public void setReturn(String retVarName){
       BatsDataType bdt = new BatsDataType();
       String bdtRetType = "";

        if(!funcName.equals("") && funst.containsKey(funcName)){
          BatsFunction funSymT = (BatsFunction) funst.get(funcName);
          if(retVarName.equals("true") || retVarName.equals("false")){
//boolean
              bdt = new BatsBoolean(retVarName);
              funSymT.setReturnValue(bdt);
           }
          //check if return variable exists in the function
          else if(funSymT.variables.containsKey(retVarName) ){
             bdt = (BatsDataType)funSymT.variables.get(retVarName);
             bdtRetType = bdt.getDataType();
             if(bdtRetType.equals(funSymT.getReturnType())){
                funSymT.setReturnName(retVarName);
                funSymT.setReturnValue(bdt);
             }
             else
                printError("Mismatch return type in function");
          }
          //check if return variable exists in the global ST
           else if(varst.containsKey(retVarName) ){
                bdt = (BatsDataType)varst.get(retVarName);
                bdtRetType = bdt.getDataType();
                if(bdtRetType.equals(funSymT.getReturnType())){
                   funSymT.setReturnName(retVarName);
                   funSymT.setReturnValue(bdt);
                }
           }
           else {//if varName doesn't exist neither in global or the
funtion's scopes
             printError("No return variable defined in function");
```

```
        }
       }
   }
    public void setReturn(){

      String funRetType = "";

       if(!funcName.equals("") && funst.containsKey(funcName)){
         BatsFunction funSymT = (BatsFunction) funst.get(funcName);

         funRetType = funSymT.getReturnType();
         if(funRetType.equals("Boolean") && (rValue.equals("true") ||
rValue.equals("false"))){
           BatsDataType returnBdt = new BatsDataType();
           returnBdt.setDataType("Boolean");
           returnBdt.setValue(rValue);
           returnBdt.setName("return"+funcName);
           funSymT.setReturnValue(returnBdt);
         }
         if(funRetType.equals("Point") ){
           BatsPoint returnBdt = new BatsPoint();
           returnBdt.setDataType("Point");
           returnBdt = this.pointVar;
           returnBdt.setName("return"+funcName);
           funSymT.setReturnValue(returnBdt);
         }
         if(funRetType.equals("Line") ){
           BatsLine returnBdt = new BatsLine();
           returnBdt.setDataType("Line");
           returnBdt = this.lineVar;
           returnBdt.setName("return"+funcName);
           funSymT.setReturnValue(returnBdt);
         }
         if(funRetType.equals("Int") ){
           BatsDataType returnBdt = new BatsDataType();
           returnBdt.setDataType("Int");
           returnBdt.setValue(rValue);
           returnBdt.setName("return"+funcName);
           funSymT.setReturnValue(returnBdt);
         }
      }
   }

   public BatsPoint getPointVar(){
      return this.pointVar;
   }

   public void funcEnd(){
      funcStart = false;
      this.funcName = "";
   }
    public void startLet(){
      let = true;
   }
    public void setRvalue(String rV){

      this.rValue = rV;
```

```
  System.out.println("SET RVLAUE2");
   }

   public String getRvalue(){
      return this.rValue;
   }
   public void setLvalue(String lV){
         lValue = lV;
   }
    public void endLet(){
      let = false;
   }
    public void setDataType(String dt){
      this.dataType = dt;
   }
   public void setScope(String sc){
      this.scope = sc;
   }
   public String getScope(){
      return this.scope;
   }
   public boolean varExists(String varName, String scope){
      boolean result = false;
      if(scope.equals("Global")){
        if(varst.containsKey(varName)){
         result = true;
        }
     }
      return result;
   }
   public boolean funExists( String funName, String scope){
      if(funst.containsKey(funName))
         return true;
      else return false;
   }

 /**
 *    The method used to set Global variables to global variables
symbol table
 */
   public void setGlobalVariables(Vector vars)
   {
      boolean exists = false;
      BatsDataType globVar = null;
      for (int i = 0; i < vars.size(); i++){

         String paramName = ((String)vars.elementAt(i)).toString();
         if(this.dataType.equals("Int")){
            globVar = new BatsInteger();
            globVar = new BatsInteger( paramName,this.scope);
         }
         else if(this.dataType.equals("Boolean")){
            globVar = new BatsBoolean();
            globVar = new BatsBoolean(paramName,this.scope);
         }
         else if(this.dataType.equals("Point")){
            globVar = new BatsPoint();
```

```
            globVar = new BatsPoint(paramName,this.scope);
          }
           else if(this.dataType.equals("Line")){
              globVar = new BatsLine();
              globVar = new BatsLine(paramName,this.scope);
           }

           exists = varExists(globVar.getName(), this.scope);
           if(exists)
              printError("The identifier name1 '"+globVar.getName()+"'
has been already defined.");
           if(!exists){
              varst.put(globVar.getName(),globVar);
           }

        }
     }

 /**
 *    The method used to set functions ids and variables inside of
function to function's symbol
 *    table
 */
   public void setFunVar(String pName,String paramVar)
   {
         if(funcStart && funcName.equals("")){
            BatsFunction batsFDT = new BatsFunction(pName);
            this.scope = "function";
            batsFDT.setScope(this.scope);
            batsFDT.setCategory("function");
            batsFDT.setReturnType(this.dataType);
            this.funcName = pName;
            if(!beginFun && funcName.equals("Begin")){
               beginFun = true;
            }

            batsFDT.setName(this.funcName);
            if(!funExists(batsFDT.getName(), this.scope)){//a new
function Name
                  funst.put(batsFDT.getName(),batsFDT);
                  this.scope = pName;

                 System.out.println("Function Name
"+batsFDT.getName()+" dataType "+batsFDT.getReturnType()+" scope
"+batsFDT.getScope()
                  +" category "+batsFDT.getCategory());
            }
            else{ //error function name exists
               printError("Not allowed to have two functions with the
same name in the program");
            }
         }
         else if(funcStart && !funcName.equals("")){
            BatsDataType batsDT = new BatsDataType(pName);
            this.scope = funcName;
            batsDT.setDataType(this.dataType);
            batsDT.setScope(this.scope);
```

```
            if(paramVar.equals("var"))
               batsDT.setCategory("var");
            else
               batsDT.setCategory("param");
            if(funst.containsKey(funcName)){
               BatsFunction funSymT = (BatsFunction)
funst.get(funcName);
               if(funSymT.variables != null &&
!(funSymT.getVariables().containsKey(batsDT.getName()))){
                  funSymT.variables.put(batsDT.getName(), batsDT);
                  if( batsDT.getCategory().equals("param")){

                      int po = funSymT.parameters.size();
                      batsDT.setParamOrder(po);
                      funSymT.parameters.put(batsDT.getName(), batsDT);
                  }
               }//if no such variables in function
               else{
                   printError("The identifier name2
'"+batsDT.getName()+"' has been already defined in function.");
               }

            }
         }
   }
/*********************************************************************/
  public void printHash(){
   for (Enumeration e = varst.elements() ; e.hasMoreElements() ;) {
         BatsDataType bd = (BatsDataType)e.nextElement();
         System.out.println("varst "+bd.getName()+ " type
"+bd.getDataType());
     }
    for (Enumeration e = funst.elements() ; e.hasMoreElements() ;) {
         BatsFunction bf = (BatsFunction)e.nextElement();
         System.out.println("funst "+bf+" retName "+bf.getReturnName()+
         " retType "+bf.getReturnType()+" retValue
"+((BatsDataType)bf.getReturnValue()).getValue()+" fVars
"+bf.variables);
         Hashtable h = bf.variables;
     }
  }

 /**   The method used to indicate start of Color statement **/

  public void startColor( String rbgName){
      this.startColor = true;
      this.rbg = rbgName;
  }

 /**   The method used to indicate start of while statement **/

    public void startWhile( ){
      this.startWhile = true;
   //   this.iteration.add(new Vector(););
  }
   public void endWhile( ){
      this.startWhile = false;
```

```
 }
 public void setRelOp(String relOp){
  this.relOp = relOp;
 }

/**
*    The method used to assign values to global and local variables
*/
public void assign(){

  System.out.println("IN Assign lValue "+lValue+" rValue
"+this.rValue);

   if(startReturn){
      setReturn();
   }

   else if(startColor){
     try{
      if(this.rbg.equals("red")){
         bColor = new BatsColor();
         bColor.setRValue(Integer.parseInt(rValue));
      }
      else if(this.rbg.equals("green")){
         bColor.setGValue(Integer.parseInt(rValue));
      }
      else if(this.rbg.equals("blue")){
         bColor.setBValue(Integer.parseInt(rValue));

      }
      }catch(Exception e){}
System.out.println("=====rbgName "+bColor.getRValue());
   }
   else if(lValue == null || rValue == null){
      printError("Type mismatch.");
   }

   //global variables assignment
   else if(rValue != null && varst.containsKey(lValue) ){
         assignGlobVar();
   }
   //function's variables asignment
   else if(rValue != null && !funcName.equals("")){
        assignFunVar();
   }
   else if(rValue != null && !varst.containsKey(lValue)){
       printError("lValue "+lValue+" is not properly declared");
   }

}

public void assignGlobVar()
{

    BatsDataType bdt = (BatsDataType)(varst.get(lValue));
    String dType = bdt.getDataType();
```

```
System.out.println("in assignGlobVar() dType "+bdt.getDataType());
      if(dType.equals("Line") && rValue.equals("line")){ //for Line
variable
          BatsLine bl = new BatsLine(lValue,"Global");

System.out.println("++++++++POINTS "+ this.linePoints);
System.out.println("lineVar "+ lineVar.getName() +" LINE TYPE
"+lineVar.getDataType());

          bl.addPoints(this.linePoints);
          this.linePoints = new Vector();
System.out.println("bl line "+bl+" bl points "+ bl.points);
for(int i = 0; i < bl.points.size(); i++){
   BatsPoint bp = (BatsPoint) bl.points.elementAt(i);
   System.out.println("POINT IN LINE "+bp.xValue+" YValue "+bp.yValue);
}
System.out.println("lVALUE " +lValue + " BL "+bl);
          varst.put( lValue, bl);
      }
      else if(dType.equals("Point") && rValue.equals("point")){ //for
Point variable
          BatsPoint bp = new BatsPoint( );
          bp = pointVar;
          bp.setName(lValue);
          bp.setDataType(dType);
System.out.println("POINT lValue "+ lValue + " bp.X "+bp.xValue+" bp.Y
"+bp.yValue);
         varst.put( lValue, bp);
      }
        else if(dType.equals("Boolean")){
            if(rValue.equals("true") || rValue.equals("false")){
               bdt.setValue(rValue);
               varst.put( lValue, bdt);
            }
            else{
               printError("Type mismatch for Boolean type");
            }
        }
      else{ // for Int
         bdt.setValue(rValue);
      }
 }

 public void assignFunVar()
 {
   int rv = 0;
 System.out.println("In assignFunVar ");
   if(funst.containsKey(funcName)){
            BatsFunction funSymT = (BatsFunction) funst.get(funcName);
            if(funSymT.variables != null &&
(funSymT.getVariables().containsKey(lValue))){
                BatsDataType bdt =
(BatsDataType)(funSymT.getVariables().get(lValue));
                String dType = bdt.getDataType();
             try{
                 if(dType.equals("Line") && rValue.equals("line")){
//for Line variable
```

```
                   BatsLine bl = new BatsLine(
bdt.getName(),bdt.getScope());

                  bl.addPoints(this.linePoints);

         /*for(int i = 0; i < bl.points.size(); i++){
            BatsPoint bp = (BatsPoint) bl.points.elementAt(i);
            System.out.println("POINTS IN LINE IN FUNCTION"+bp.xValue+"
YValue "+bp.yValue);
         }*/
                   funSymT.variables.put( lValue, bl);
                   this.linePoints = new Vector();
                 }
                 else if(dType.equals("Point") &&
rValue.equals("point")){ //for Point variable
                    BatsPoint bp = new BatsPoint( bdt.getName(),
bdt.getScope() );
                    bp.setXValue(this.pointVar.getXValue());
                    bp.setYValue(this.pointVar.getYValue());

                    funSymT.variables.put( lValue, bp);
                  }
                  else if(dType.equals("Int")){

                     try{
                        rv = Integer.parseInt(rValue);

                    }catch(NumberFormatException e){
                         rv = getVarValue(rValue);
                         rValue = new Integer(rv).toString();

                    }
                     if(bdt != null){
                        bdt.setValue(new Integer(rv).toString());
                        funSymT.variables.put( lValue, bdt);
                     }
                  }
                  else if(dType.equals("Boolean")){
                     if(rValue.equals("true") ||
rValue.equals("false")){
                        bdt = this.boolVar;
                        funSymT.variables.put( lValue, bdt);
                      }
                      else{
                        printError("Type mismatch for Boolean type");
                      }
                  }
               }catch(Exception e){
                  printError("rValue NumberFormatException during
assignment");
               }
            }
        }//funcName
 }

 public void  mathOper(){
   System.out.println("IN MATHOP" +this.mathSign);
```

```
    char[] chArr = this.mathSign.toCharArray();

      switch (chArr[0]) {
         case '+': add(); break;
         case '-': {
         System.out.println("IN SWITCH");
         minus(); break;}
         case '*': mult(); break;
         case '/': div(); break;
         default: printError("The operation does not exist");
      }

 }
  public boolean compare(){
    boolean res = false;

   if(this.relSign != null ){
 System.out.println("IN Compare" +this.relSign);
    if(this.relSign.equals(">")){
      gt();
    }
    else if(this.relSign.equals("<")){
      lt();
    }
    else if(this.relSign.equals(">=")){
      ge();
    }
    else if(this.relSign.equals("<=")){
      le();
    }
    else if(this.relSign.equals("&&")){
      and();
    }
    else if(this.relSign.equals("||")){
      or();
    }
    this.rConValue = this.rightOper;
    this.conSign = this.relSign;
System.out.println("--======BOOL "+this.rValue + "ConValue
"+this.rConValue);

      if( this.rValue.equals("true"))
         res = true;
      else
         res = false;

  }//if relSign != null
      return res;

  }
  public void  cleanResources(){
    this.rValue = "";
    this.lValue = "";
    this.relSign = "";
    this.mathSign = "";
    this.wLeft = "";
    this.wRight = "";
```

```
 }
 public void mOp(String sign){
   this.mathSign = sign;
 }
 public void relOp(String sign){
   this.relSign = sign;
 }
 public void add(){
//  System.out.println("IN ADD lValue "+leftOper+" rOper "+rightOper);
   this.rValue = new Integer(leftOper + rightOper).toString();
 }

 public void minus(){
   this.rValue = new Integer(leftOper - rightOper).toString();

 }

 public void mult(){
   this.rValue = new Integer(leftOper * rightOper).toString();
 }

 public void div(){
   this.rValue = new Integer(leftOper / rightOper).toString();
 }

 public void mod(){
   this.rValue = new Integer(leftOper % rightOper).toString();
 }

 public void gt(){
   this.rValue = new Boolean(leftOper > rightOper).toString();

 }

 public void ge(){
   this.rValue = new Boolean(leftOper > rightOper).toString();
 }

 public void lt(){
   this.rValue = new Boolean(leftOper > rightOper).toString();
 }

 public void le(){
   this.rValue = new Boolean(leftOper > rightOper).toString();
 }

 public void and(){
   this.rValue = new Boolean(leftOper > rightOper).toString();
 }
 public void or(){
    this.rValue = new Boolean(leftOper > rightOper).toString();
 }

 public void setLeft(String operat){
   try{
   System.out.println("leftOper "+operat);
      this.leftOper = Integer.parseInt(operat);
```

```
   }catch(NumberFormatException e){
       this.leftOper = getVarValue(operat);

//   printError("Wrong data type for the left operand to "+operat);
   }
 }
 public void setRight(String operat){
   try{
     System.out.println("rightOper "+operat);
      this.rightOper = Integer.parseInt(operat);
   }catch(NumberFormatException e){
       this.rightOper = getVarValue(operat);
   //printError("Wrong data type right operand to "+operat);
   }
 }
 public void checkBegin(){
   if (!beginFun){
       printError("No Begin method found in the program.");
   }
 }
 public void printError(String msg){
     System.out.println("ERROR: "+msg);
    System.exit(1);
 }
 public boolean Cond(){
;
   boolean res = false;

   if(conSign != null ){
// System.out.println("IN WHileCompare" +this.conSign+" leftOper
"+this.leftOper+" rightOper "+this.rightOper);
    if(conSign.equals(">")){
      gt();
    }
    else if(conSign.equals("<")){
      lt();
    }
    else if(conSign.equals(">=")){
      ge();
    }
    else if(conSign.equals("<=")){
      le();
    }
    else if(conSign.equals("&&")){
      and();
    }
    else if(conSign.equals("||")){
      or();
    }
   }

    if( this.rValue.equals("true"))
       res = true;
    else
       res = false;
        System.out.println("IN WHileCompare"+res);
   return res;
```

```
 }
 public void mkWhile(){
  // System.out.println("LVALUE "+lValue+"RValue "+rValue+" wLeft
"+wLeft+" wRight "+ wRight);
   //System.out.println(" leftOper "+leftOper+" rightOper "+rightOper+"
wLeft "+wLeft+" wRight "+ wRight);
//System.out.println("rConValue "+ rConValue+" conSign "+conSign+"
mathSign "+mathSign);
    int loopCntr = 0;
    this.leftOper = Integer.parseInt(rValue);
    this.rightOper = rightOper;
    boolean res = true;

    while(res){
       mathOper();
       this.leftOper = Integer.parseInt(this.rValue);
       System.out.println("HELLO1"+loopCntr + " "+this.leftOper);
       if(loopCntr > 40)
          break;
       loopCntr++;
      res = Cond( );
   }

 }

 public void draw( String varPL){
   if(!funcName.equals("") && funst.containsKey(funcName)){
      BatsFunction bf = (BatsFunction) funst.get(funcName);

      // variable varPL (point or line) is in function scope
      if(bf.variables != null &&
(bf.getVariables().containsKey(varPL))){
         BatsDataType bdt =
(BatsDataType)(bf.getVariables().get(varPL));
         String dType = bdt.getDataType();
         if(dType.equals("Line") ){
            BatsLine bl = (BatsLine)bf.getVariables().get(varPL);

           builtIn.append(bl);
         }
      }
      //if varPL not in function but in global scope
      if(varst != null && varst.containsKey(varPL)){
         BatsDataType bdt = (BatsDataType) varst.get(varPL);
         String dType = bdt.getDataType();

         if(dType.equals("Point") || dType.equals("Line") ){
            BatsLine bl = (BatsLine)varst.get(varPL);

          //  builtIn.draw(bl);
          builtIn.append(bl);
         }
         else{
            printError("Expecting Point or Line, sent "+dType + " to
builtin function 'draw'");
         }//if Point or Line
```

```
      }//for global var
   }//if

 }

 /**
 * The method used to append color to the input data file
 */
 public void appendColor(){
     builtIn.append(bColor);
 }
public void drawPicture(){
   builtIn.drawLine();
}

}
```

BatsKeywordMap.java:

```
import java.util.*;

/**
 */
public class BatsKeywordMap {

   private static HashMap _keyWords;

   static {
      _keyWords = new HashMap();

      _keyWords.put("Int",       "int");
      _keyWords.put("Boolean",   "boolean");
      _keyWords.put("Line",      "BatsLine");
      _keyWords.put("Point",     "BatsPoint");
      _keyWords.put("While",     "while");
      _keyWords.put("If",        "if");
       _keyWords.put("True",      "true");
       _keyWords.put("False",     "false");
   }

   /***/
   public static String toJava(String batsTerm) {
      return (String)_keyWords.get(batsTerm);
   }

}
```

BatsLexer.java:

```
// $ANTLR 2.7.2: "BATS.g" -> "BATSLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
```

```
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class BATSLexer extends antlr.CharScanner implements
BATStokensTokenTypes, TokenStream
 {
public BATSLexer(InputStream in) {
      this(new ByteBuffer(in));
}
public BATSLexer(Reader in) {
      this(new CharBuffer(in));
}
public BATSLexer(InputBuffer ib) {
      this(new LexerSharedInputState(ib));
}
public BATSLexer(LexerSharedInputState state) {
      super(state);
      caseSensitiveLiterals = true;
      setCaseSensitive(true);
      literals = new Hashtable();
      literals.put(new ANTLRHashString("IfEnd", this), new
Integer(11));
      literals.put(new ANTLRHashString("Then", this), new Integer(9));
      literals.put(new ANTLRHashString("Int", this), new Integer(4));
      literals.put(new ANTLRHashString("To", this), new Integer(28));
      literals.put(new ANTLRHashString("ForEnd", this), new
Integer(16));
      literals.put(new ANTLRHashString("Call", this), new Integer(29));
      literals.put(new ANTLRHashString("False", this), new
Integer(26));
      literals.put(new ANTLRHashString("Color", this), new
Integer(22));
      literals.put(new ANTLRHashString("True", this), new Integer(25));
      literals.put(new ANTLRHashString("For", this), new Integer(15));
      literals.put(new ANTLRHashString("WhileEnd", this), new
Integer(14));
      literals.put(new ANTLRHashString("XAxis", this), new
Integer(19));
      literals.put(new ANTLRHashString("Let", this), new Integer(27));
```

```
      literals.put(new ANTLRHashString("Boolean", this), new
Integer(7));
      literals.put(new ANTLRHashString("Append", this), new
Integer(24));
      literals.put(new ANTLRHashString("End", this), new Integer(31));
      literals.put(new ANTLRHashString("YAxis", this), new
Integer(20));
      literals.put(new ANTLRHashString("Do", this), new Integer(13));
      literals.put(new ANTLRHashString("In", this), new Integer(18));
      literals.put(new ANTLRHashString("Return", this), new
Integer(21));
      literals.put(new ANTLRHashString("While", this), new
Integer(12));
      literals.put(new ANTLRHashString("Line", this), new Integer(5));
      literals.put(new ANTLRHashString("Point", this), new Integer(6));
      literals.put(new ANTLRHashString("If", this), new Integer(8));
      literals.put(new ANTLRHashString("Global", this), new
Integer(30));
      literals.put(new ANTLRHashString("Draw", this), new Integer(23));
      literals.put(new ANTLRHashString("Else", this), new Integer(10));
      literals.put(new ANTLRHashString("ForLine", this), new
Integer(17));
}

public Token nextToken() throws TokenStreamException {
      Token theRetToken=null;
tryAgain:
      for (;;) {
            Token _token = null;
            int _ttype = Token.INVALID_TYPE;
            resetText();
            try {   // for char stream error handling
                  try {   // for lexical error handling
                        switch ( LA(1)) {
                        case '+':
                        {
                              mPLUS(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '-':
                        {
                              mMINUS(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '*':
                        {
                              mMUL(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '/':
                        {
                              mDIV(true);
                              theRetToken=_returnToken;
                              break;
```

```
}
case '%':
{
      mMOD(true);
      theRetToken=_returnToken;
      break;
}
case ',':
{
      mCOMMA(true);
      theRetToken=_returnToken;
      break;
}
case '.':
{
      mPERIOD(true);
      theRetToken=_returnToken;
      break;
}
case '^':
{
      mCARRAT(true);
      theRetToken=_returnToken;
      break;
}
case '&':
{
      mAND(true);
      theRetToken=_returnToken;
      break;
}
case '|':
{
      mOR(true);
      theRetToken=_returnToken;
      break;
}
case '=':
{
      mEQEQ(true);
      theRetToken=_returnToken;
      break;
}
case '(':
{
      mLPARENS(true);
      theRetToken=_returnToken;
      break;
}
case ')':
{
      mRPARENS(true);
      theRetToken=_returnToken;
      break;
}
case ';':
{
```

```
      mSEMI(true);
      theRetToken=_returnToken;
      break;
}
case '{':
{
      mLBRACE(true);
      theRetToken=_returnToken;
      break;
}
case '}':
{
      mRBRACE(true);
      theRetToken=_returnToken;
      break;
}
case ']':
{
      mRBRACKET(true);
      theRetToken=_returnToken;
      break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
      mINTEGER(true);
      theRetToken=_returnToken;
      break;
}
case '\t':  case '\n':  case '\r':  case ' ':
{
      mWS(true);
      theRetToken=_returnToken;
      break;
}
case '#':
{
      mCOMMENT(true);
      theRetToken=_returnToken;
      break;
}
default:
      if ((LA(1)=='<') && (LA(2)=='-')) {
            mASSIGN(true);
            theRetToken=_returnToken;
      }
      else if ((LA(1)=='>') && (LA(2)=='=')) {
            mGEQ(true);
            theRetToken=_returnToken;
      }
      else if ((LA(1)=='<') && (LA(2)=='=')) {
            mLEQ(true);
            theRetToken=_returnToken;
      }
      else if ((LA(1)=='!') && (LA(2)=='=')) {
            mNEQ(true);
```

```
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='[') && (LA(2)==']')) {
                                    mBRACKET(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='E'||LA(1)=='e') &&
(LA(2)=='+'||LA(2)=='-')) {
                                    mEXPONENT(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='>') && (true)) {
                                    mGREATER(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='<') && (true)) {
                                    mLESS(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='[') && (true)) {
                                    mLBRACKET(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='!') && (true)) {
                                    mNOT(true);
                                    theRetToken=_returnToken;
                              }
                              else if ((_tokenSet_0.member(LA(1))) &&
(true)) {
                                    mID(true);
                                    theRetToken=_returnToken;
                              }
                        else {
                              if (LA(1)==EOF_CHAR) {uponEOF();
_returnToken = makeToken(Token.EOF_TYPE);}
                        else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                        }
                        }
                        if ( _returnToken==null ) continue tryAgain; //
found SKIP token
                        _ttype = _returnToken.getType();
                        _returnToken.setType(_ttype);
                        return _returnToken;
                  }
                  catch (RecognitionException e) {
                        throw new TokenStreamRecognitionException(e);
                  }
            }
            catch (CharStreamException cse) {
                  if ( cse instanceof CharStreamIOException ) {
                        throw new
TokenStreamIOException(((CharStreamIOException)cse).io);
                  }
                  else {
```

```
                        throw new
TokenStreamException(cse.getMessage());
                  }
            }
      }
}

      public final void mPLUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUS;
            int _saveIndex;

            match('+');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMINUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MINUS;
            int _saveIndex;

            match('-');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMUL(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MUL;
            int _saveIndex;

            match('*');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mDIV(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIV;
            int _saveIndex;
```

```
            match('/');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMOD(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MOD;
            int _saveIndex;

            match('%');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mASSIGN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = ASSIGN;
            int _saveIndex;

            match("<-");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMA;
            int _saveIndex;

            match(',');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mPERIOD(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
```

```
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PERIOD;
            int _saveIndex;

            match('.');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mGREATER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = GREATER;
            int _saveIndex;

            match('>');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLESS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LESS;
            int _saveIndex;

            match('<');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCARRAT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = CARRAT;
            int _saveIndex;

            match('^');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
	public final void mGEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GEQ;
		int _saveIndex;

		match(">=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LEQ;
		int _saveIndex;

		match("<=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mAND(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = AND;
		int _saveIndex;

		match("&&");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mOR(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = OR;
		int _saveIndex;

		match("||");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
```

```
            }
            _returnToken = _token;
      }

      public final void mEQEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = EQEQ;
            int _saveIndex;

            match("==");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NEQ;
            int _saveIndex;

            match("!=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLPARENS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LPARENS;
            int _saveIndex;

            match('(');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRPARENS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RPARENS;
            int _saveIndex;

            match(')');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mSEMI(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = SEMI;
            int _saveIndex;

            match(';');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LBRACE;
            int _saveIndex;

            match('{');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RBRACE;
            int _saveIndex;

            match('}');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mBRACKET(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = BRACKET;
            int _saveIndex;
```

```
            match("[]");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLBRACKET(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LBRACKET;
            int _saveIndex;

            match('[');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRBRACKET(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RBRACKET;
            int _saveIndex;

            match(']');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNOT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NOT;
            int _saveIndex;

            match('!');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      protected final void mLETTER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
```

```
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LETTER;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case 'a':  case 'b':  case 'c':  case 'd':
            case 'e':  case 'f':  case 'g':  case 'h':
            case 'i':  case 'j':  case 'k':  case 'l':
            case 'm':  case 'n':  case 'o':  case 'p':
            case 'q':  case 'r':  case 's':  case 't':
            case 'u':  case 'v':  case 'w':  case 'x':
            case 'y':  case 'z':
            {
                  matchRange('a','z');
                  break;
            }
            case 'A':  case 'B':  case 'C':  case 'D':
            case 'E':  case 'F':  case 'G':  case 'H':
            case 'I':  case 'J':  case 'K':  case 'L':
            case 'M':  case 'N':  case 'O':  case 'P':
            case 'Q':  case 'R':  case 'S':  case 'T':
            case 'U':  case 'V':  case 'W':  case 'X':
            case 'Y':  case 'Z':
            {
                  matchRange('A','Z');
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      protected final void mDIGIT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIGIT;
            int _saveIndex;

            matchRange('0','9');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
      public final void mID(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = ID;
            int _saveIndex;

            mLETTER(false);
            {
            _loop32:
            do {
                  switch ( LA(1)) {
                  case 'A':  case 'B':  case 'C':  case 'D':
                  case 'E':  case 'F':  case 'G':  case 'H':
                  case 'I':  case 'J':  case 'K':  case 'L':
                  case 'M':  case 'N':  case 'O':  case 'P':
                  case 'Q':  case 'R':  case 'S':  case 'T':
                  case 'U':  case 'V':  case 'W':  case 'X':
                  case 'Y':  case 'Z':  case 'a':  case 'b':
                  case 'c':  case 'd':  case 'e':  case 'f':
                  case 'g':  case 'h':  case 'i':  case 'j':
                  case 'k':  case 'l':  case 'm':  case 'n':
                  case 'o':  case 'p':  case 'q':  case 'r':
                  case 's':  case 't':  case 'u':  case 'v':
                  case 'w':  case 'x':  case 'y':  case 'z':
                  {
                        mLETTER(false);
                        break;
                  }
                  case '0':  case '1':  case '2':  case '3':
                  case '4':  case '5':  case '6':  case '7':
                  case '8':  case '9':
                  {
                        mDIGIT(false);
                        break;
                  }
                  case '_':
                  {
                        match('_');
                        break;
                  }
                  default:
                  {
                        break _loop32;
                  }
                  }
            } while (true);
            }
            _ttype = testLiteralsTable(_ttype);
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
	public final void mINTEGER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INTEGER;
		int _saveIndex;

		{
		int _cnt35=0;
		_loop35:
		do {
			if (((LA(1) >= '0' && LA(1) <= '9'))) {
				mDIGIT(false);
			}
			else {
				if ( _cnt35>=1 ) { break _loop35; } else {throw
new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
			}

			_cnt35++;
		} while (true);
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mEXPONENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = EXPONENT;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'e':
		{
			match('e');
			break;
		}
		case 'E':
		{
			match('E');
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
		}
		}
		}
		{
		switch ( LA(1)) {
```

```
            case '+':
            {
                  match('+');
                  break;
            }
            case '-':
            {
                  match('-');
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            mDIGIT(false);
            mDIGIT(false);
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mWS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = WS;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case ' ':
            {
                  match(' ');
                  break;
            }
            case '\t':
            {
                  match('\t');
                  break;
            }
            case '\n':
            {
                  match('\n');
                  newline();
                  break;
            }
            case '\r':
            {
                  match('\r');
                  break;
            }
            default:
```

```
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            _ttype = Token.SKIP;
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMENT;
            int _saveIndex;

            match('#');
            {
            _loop44:
            do {
                  if ((_tokenSet_1.member(LA(1)))) {
                        {
                        match(_tokenSet_1);
                        }
                  }
                  else {
                        break _loop44;
                  }

            } while (true);
            }
            {
            switch ( LA(1)) {
            case '\n':
            {
                  match('\n');
                  newline();
                  break;
            }
            case '\r':
            {
                  match('\r');
                  newline();
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
```

```
            _ttype = Token.SKIP;
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }


      private static final long[] mk_tokenSet_0() {
            long[] data = { 0L, 576460743847706622L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
      private static final long[] mk_tokenSet_1() {
            long[] data = { 8935141020752937472L, 4611686013863985150L,
0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_1 = new
BitSet(mk_tokenSet_1());

      }
```

BatsLine.java:

```
import java.io.PrintWriter;
import antlr.CommonAST;
import java.util.*;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: BatsDataType.java,v 1.17 2003/05/12 23:44:34 hanhua
Exp $
 */
public class BatsLine extends BatsDataType
{
   Vector points = new Vector();


    public BatsLine( ) {
    }
    public BatsLine(String name,String scope ) {
       super(name,scope,"Line","var");
    }
/*    public BatsLine( BatsDataType bd ) {
    }*/
    public BatsLine(Vector points){
      this.points = points;
```

```
    }
    public BatsLine( BatsPoint bp ) {
         this.points.add(bp);
    }
    public BatsLine( BatsPoint bp1, BatsPoint bp2 ) {
         this.points.add(bp1);
         this.points.add(bp2);
    }
    public void addPoint(BatsPoint bp){
           this.points.add(bp);
    }
    public void addPoints(Vector vbp){
         for(int i = 0; i < vbp.size(); i++){
            BatsPoint bp = (BatsPoint) vbp.elementAt(i);
            this.points.add(bp);
         }
    }
    public Vector getLine(){ return points;}
    public String toString() {return "BatsLine";}

}
```

BatsParser.java:

```
// $ANTLR 2.7.2: "BATS.g" -> "BATSParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class BATSParser extends antlr.LLkParser       implements
BATSParserTokenTypes
 {

protected BATSParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
```

```
public BATSParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected BATSParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public BATSParser(TokenStream lexer) {
  this(lexer,2);
}

public BATSParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

      public final void program() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST program_AST = null;

            try {      // for error handling
                  {
                  _loop48:
                  do {
                        if ((LA(1)==GLOBAL)) {
                              globvarDecl();
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              break _loop48;
                        }

                  } while (true);
                  }
                  {
                  int _cnt50=0;
                  _loop50:
                  do {
                        if (((LA(1) >= INT && LA(1) <= BOOLSTR))) {
                              funcDecl();
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              if ( _cnt50>=1 ) { break _loop50; } else
{throw new NoViableAltException(LT(1), getFilename());}
```

```
                        }

                        _cnt50++;
                  } while (true);
                  }
                  match(Token.EOF_TYPE);
                  program_AST = (AST)currentAST.root;
                  program_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PROGRAM,"program")).add(program_AST)
);
                  currentAST.root = program_AST;
                  currentAST.child = program_AST!=null
&&program_AST.getFirstChild()!=null ?
                        program_AST.getFirstChild() : program_AST;
                  currentAST.advanceChildToEnd();
                  program_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_0);
            }
            returnAST = program_AST;
      }

      public final void globvarDecl() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST globvarDecl_AST = null;

            try {      // for error handling
                  AST tmp2_AST = null;
                  tmp2_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp2_AST);
                  match(GLOBAL);
                  type();
                  astFactory.addASTChild(currentAST, returnAST);
                  AST tmp3_AST = null;
                  tmp3_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp3_AST);
                  match(ID);
                  {
                  _loop53:
                  do {
                        if ((LA(1)==COMMA)) {
                              match(COMMA);
                              AST tmp5_AST = null;
                              tmp5_AST = astFactory.create(LT(1));
                              astFactory.addASTChild(currentAST,
tmp5_AST);
                              match(ID);
                        }
                        else {
                              break _loop53;
                        }
```

```
                  } while (true);
                  }
                  match(SEMI);
                  globvarDecl_AST = (AST)currentAST.root;
                  globvarDecl_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(GLOBVARDECL,"globvarDecl")).add(glob
varDecl_AST));
                  currentAST.root = globvarDecl_AST;
                  currentAST.child = globvarDecl_AST!=null
&&globvarDecl_AST.getFirstChild()!=null ?
                        globvarDecl_AST.getFirstChild() :
globvarDecl_AST;
                  currentAST.advanceChildToEnd();
                  globvarDecl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_1);
            }
            returnAST = globvarDecl_AST;
      }

      public final void funcDecl() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funcDecl_AST = null;

            try {      // for error handling
                  type();
                  astFactory.addASTChild(currentAST, returnAST);
                  AST tmp7_AST = null;
                  tmp7_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp7_AST);
                  match(ID);
                  match(LPARENS);
                  paramlist();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RPARENS);
                  varDecl();
                  astFactory.addASTChild(currentAST, returnAST);
                  stmtblock();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(END);
                  funcDecl_AST = (AST)currentAST.root;
                  funcDecl_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(FUNCDECL,"funcDecl")).add(funcDecl_A
ST));
                  currentAST.root = funcDecl_AST;
                  currentAST.child = funcDecl_AST!=null
&&funcDecl_AST.getFirstChild()!=null ?
                        funcDecl_AST.getFirstChild() : funcDecl_AST;
                  currentAST.advanceChildToEnd();
                  funcDecl_AST = (AST)currentAST.root;
```

```
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_2);
            }
            returnAST = funcDecl_AST;
      }

      public final void type() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST type_AST = null;

            try {      // for error handling
                  {
                  switch ( LA(1)) {
                  case INT:
                  {
                        AST tmp11_AST = null;
                        tmp11_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp11_AST);
                        match(INT);
                        break;
                  }
                  case LINESTR:
                  {
                        AST tmp12_AST = null;
                        tmp12_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp12_AST);
                        match(LINESTR);
                        break;
                  }
                  case POINTSTR:
                  {
                        AST tmp13_AST = null;
                        tmp13_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp13_AST);
                        match(POINTSTR);
                        break;
                  }
                  case BOOLSTR:
                  {
                        AST tmp14_AST = null;
                        tmp14_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp14_AST);
                        match(BOOLSTR);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
```

```
                  }
                  {
                  switch ( LA(1)) {
                  case BRACKET:
                  {
                        AST tmp15_AST = null;
                        tmp15_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp15_AST);
                        match(BRACKET);
                        break;
                  }
                  case ID:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  type_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_3);
            }
            returnAST = type_AST;
      }

      public final void paramlist() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST paramlist_AST = null;

            try {      // for error handling
                  {
                  switch ( LA(1)) {
                  case INT:
                  case LINESTR:
                  case POINTSTR:
                  case BOOLSTR:
                  {
                        parameter();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop63:
                        do {
                              if ((LA(1)==COMMA)) {
                                    match(COMMA);
                                    parameter();
                                    astFactory.addASTChild(currentAST,
returnAST);
```

```
                              }
                              else {
                                    break _loop63;
                              }

                        } while (true);
                        }
                        break;
                  }
                  case RPARENS:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  paramlist_AST = (AST)currentAST.root;
                  paramlist_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PARAMLIST,"paramlist")).add(paramlis
t_AST));
                  currentAST.root = paramlist_AST;
                  currentAST.child = paramlist_AST!=null
&&paramlist_AST.getFirstChild()!=null ?
                        paramlist_AST.getFirstChild() : paramlist_AST;
                  currentAST.advanceChildToEnd();
                  paramlist_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_4);
            }
            returnAST = paramlist_AST;
      }

      public final void varDecl() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST varDecl_AST = null;

            try {      // for error handling
                  {
                  _loop59:
                  do {
                        if (((LA(1) >= INT && LA(1) <= BOOLSTR))) {
                              type();
                              astFactory.addASTChild(currentAST,
returnAST);
                              AST tmp17_AST = null;
                              tmp17_AST = astFactory.create(LT(1));
```

```
                              astFactory.addASTChild(currentAST,
tmp17_AST);
                              match(ID);
                              {
                              _loop58:
                              do {
                                    if ((LA(1)==COMMA)) {
                                          match(COMMA);
                                          AST tmp19_AST = null;
                                          tmp19_AST =
astFactory.create(LT(1));

      astFactory.addASTChild(currentAST, tmp19_AST);
                                          match(ID);
                                    }
                                    else {
                                          break _loop58;
                                    }

                              } while (true);
                              }
                              match(SEMI);
                        }
                        else {
                              break _loop59;
                        }

                  } while (true);
                  }
                  varDecl_AST = (AST)currentAST.root;
                  varDecl_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(VARDECL,"varDecl")).add(varDecl_AST)
);
                  currentAST.root = varDecl_AST;
                  currentAST.child = varDecl_AST!=null
&&varDecl_AST.getFirstChild()!=null ?
                        varDecl_AST.getFirstChild() : varDecl_AST;
                  currentAST.advanceChildToEnd();
                  varDecl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_5);
            }
            returnAST = varDecl_AST;
      }

      public final void stmtblock() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST stmtblock_AST = null;

            try {      // for error handling
                  {
```

```
                  int _cnt67=0;
                  _loop67:
                  do {
                        if ((_tokenSet_5.member(LA(1)))) {
                              stmt();
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              if ( _cnt67>=1 ) { break _loop67; } else
{throw new NoViableAltException(LT(1), getFilename());}
                        }

                        _cnt67++;
                  } while (true);
                  }
                  stmtblock_AST = (AST)currentAST.root;
                  stmtblock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(STMTBLOCK,"stmtblock")).add(stmtbloc
k_AST));
                  currentAST.root = stmtblock_AST;
                  currentAST.child = stmtblock_AST!=null
&&stmtblock_AST.getFirstChild()!=null ?
                        stmtblock_AST.getFirstChild() : stmtblock_AST;
                  currentAST.advanceChildToEnd();
                  stmtblock_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_6);
            }
            returnAST = stmtblock_AST;
      }

      public final void parameter() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST parameter_AST = null;

            try {      // for error handling
                  type();
                  astFactory.addASTChild(currentAST, returnAST);
                  AST tmp21_AST = null;
                  tmp21_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp21_AST);
                  match(ID);
                  parameter_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_7);
            }
            returnAST = parameter_AST;
```

```
	}

	public final void stmt() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST stmt_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case TRUE:
			case FALSE:
			case CALL:
			case MINUS:
			case CARRAT:
			case LBRACE:
			case NOT:
			case ID:
			case INTEGER:
			case LPAREN:
			{
				expr();
				astFactory.addASTChild(currentAST, returnAST);
				match(SEMI);
				stmt_AST = (AST)currentAST.root;
				break;
			}
			case LET:
			{
				assignstmt();
				astFactory.addASTChild(currentAST, returnAST);
				match(SEMI);
				stmt_AST = (AST)currentAST.root;
				break;
			}
			case IFSTR:
			{
				ifBlock();
				astFactory.addASTChild(currentAST, returnAST);
				stmt_AST = (AST)currentAST.root;
				break;
			}
			case WHILESTR:
			{
				whileBlock();
				astFactory.addASTChild(currentAST, returnAST);
				stmt_AST = (AST)currentAST.root;
				break;
			}
			case FORSTR:
			case FORLINESTR:
			{
				forblock();
				astFactory.addASTChild(currentAST, returnAST);
				stmt_AST = (AST)currentAST.root;
				break;
```

```
                  }
                  case RETURNSTR:
                  {
                        returnstmt();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(SEMI);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  case COLORSTR:
                  case DRAWSTR:
                  case APPENDSTR:
                  {
                        builtinFunc();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(SEMI);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_8);
            }
            returnAST = stmt_AST;
      }

      public final void expr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expr_AST = null;

            try {      // for error handling
                  primaryExpr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  switch ( LA(1)) {
                  case AND:
                  case OR:
                  {
                        primaryOp();
                        astFactory.addASTChild(currentAST, returnAST);
                        primaryExpr();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case THEN:
                  case DO:
```

```
                  case SEMI:
                  case RBRACKET:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  expr_AST = (AST)currentAST.root;
                  expr_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(EXPR,"expr")).add(expr_AST));
                  currentAST.root = expr_AST;
                  currentAST.child = expr_AST!=null
&&expr_AST.getFirstChild()!=null ?
                        expr_AST.getFirstChild() : expr_AST;
                  currentAST.advanceChildToEnd();
                  expr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_9);
            }
            returnAST = expr_AST;
      }

      public final void assignstmt() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST assignstmt_AST = null;

            try {      // for error handling
                  AST tmp26_AST = null;
                  tmp26_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp26_AST);
                  match(LET);
                  varaccess();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(ASSIGN);
                  relExpr();
                  astFactory.addASTChild(currentAST, returnAST);
                  assignstmt_AST = (AST)currentAST.root;
                  assignstmt_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(ASSIGNSTMT,"assignstmt")).add(assign
stmt_AST));
                  currentAST.root = assignstmt_AST;
                  currentAST.child = assignstmt_AST!=null
&&assignstmt_AST.getFirstChild()!=null ?
                        assignstmt_AST.getFirstChild() :
assignstmt_AST;
                  currentAST.advanceChildToEnd();
```

```
                assignstmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_10);
            }
            returnAST = assignstmt_AST;
      }

      public final void ifBlock() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST ifBlock_AST = null;

            try {      // for error handling
                AST tmp28_AST = null;
                tmp28_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp28_AST);
                match(IFSTR);
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(THEN);
                stmtblock();
                astFactory.addASTChild(currentAST, returnAST);
                {
                switch ( LA(1)) {
                case ELSE:
                {
                      match(ELSE);
                      stmtblock();
                      astFactory.addASTChild(currentAST, returnAST);
                      break;
                }
                case IFEND:
                {
                      break;
                }
                default:
                {
                      throw new NoViableAltException(LT(1),
getFilename());
                }
                }
                }
                match(IFEND);
                ifBlock_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_8);
            }
            returnAST = ifBlock_AST;
      }
```

```
	public final void whileBlock() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST whileBlock_AST = null;

		try {	// for error handling
			AST tmp32_AST = null;
			tmp32_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp32_AST);
			match(WHILESTR);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(DO);
			stmtblock();
			astFactory.addASTChild(currentAST, returnAST);
			match(WHILEEND);
			whileBlock_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_8);
		}
		returnAST = whileBlock_AST;
	}

	public final void forblock() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST forblock_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case FORSTR:
			{
				forblockDefault();
				astFactory.addASTChild(currentAST, returnAST);
				forblock_AST = (AST)currentAST.root;
				break;
			}
			case FORLINESTR:
			{
				forblockLine();
				astFactory.addASTChild(currentAST, returnAST);
				forblock_AST = (AST)currentAST.root;
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1),
getFilename());
			}
```

```
                }
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_8);
            }
            returnAST = forblock_AST;
        }

        public final void returnstmt() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST returnstmt_AST = null;

            try {      // for error handling
                AST tmp35_AST = null;
                tmp35_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp35_AST);
                match(RETURNSTR);
                relExpr();
                astFactory.addASTChild(currentAST, returnAST);
                returnstmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_11);
            }
            returnAST = returnstmt_AST;
        }

        public final void builtinFunc() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST builtinFunc_AST = null;

            try {      // for error handling
                switch ( LA(1)) {
                case COLORSTR:
                {
                    colorFunc();
                    astFactory.addASTChild(currentAST, returnAST);
                    builtinFunc_AST = (AST)currentAST.root;
                    break;
                }
                case DRAWSTR:
                {
                    drawFunc();
                    astFactory.addASTChild(currentAST, returnAST);
                    builtinFunc_AST = (AST)currentAST.root;
                    break;
                }
```

```
                  case APPENDSTR:
                  {
                        appendFunc();
                        astFactory.addASTChild(currentAST, returnAST);
                        builtinFunc_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_11);
            }
            returnAST = builtinFunc_AST;
      }

      public final void forblockDefault() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST forblockDefault_AST = null;

            try {      // for error handling
                  AST tmp36_AST = null;
                  tmp36_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp36_AST);
                  match(FORSTR);
                  {
                  switch ( LA(1)) {
                  case LET:
                  {
                        assignstmt();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case SEMI:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  match(SEMI);
                  {
                  expr();
                  astFactory.addASTChild(currentAST, returnAST);
```

```
                  }
                  match(SEMI);
                  {
                  switch ( LA(1)) {
                  case LET:
                  {
                        assignstmt();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case DO:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  match(DO);
                  stmtblock();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(FOREND);
                  forblockDefault_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_8);
            }
            returnAST = forblockDefault_AST;
      }

      public final void forblockLine() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST forblockLine_AST = null;

            try {      // for error handling
                  AST tmp41_AST = null;
                  tmp41_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp41_AST);
                  match(FORLINESTR);
                  AST tmp42_AST = null;
                  tmp42_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp42_AST);
                  match(ID);
                  match(IN);
                  AST tmp44_AST = null;
                  tmp44_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp44_AST);
                  match(ID);
                  match(DO);
```

```
                  stmtblock();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(FOREND);
                  forblockLine_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_8);
            }
            returnAST = forblockLine_AST;
      }

      public final void varaccess() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST varaccess_AST = null;

            try {      // for error handling
                  if ((LA(1)==ID) && (_tokenSet_12.member(LA(2)))) {
                        AST tmp47_AST = null;
                        tmp47_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp47_AST);
                        match(ID);
                        {
                        switch ( LA(1)) {
                        case LBRACKET:
                        {
                              match(LBRACKET);
                              expr();
                              astFactory.addASTChild(currentAST,
returnAST);
                              match(RBRACKET);
                              break;
                        }
                        case THEN:
                        case DO:
                        case PLUS:
                        case MINUS:
                        case MUL:
                        case DIV:
                        case MOD:
                        case ASSIGN:
                        case GREATER:
                        case LESS:
                        case GEQ:
                        case LEQ:
                        case AND:
                        case OR:
                        case EQEQ:
                        case NEQ:
                        case SEMI:
                        case RBRACKET:
                        case RPAREN:
                        {
```

```
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(LT(1),
getFilename());
                        }
                        }
                        }
                        varaccess_AST = (AST)currentAST.root;
                  }
                  else if ((LA(1)==ID) && (LA(2)==PERIOD)) {
                        AST tmp50_AST = null;
                        tmp50_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp50_AST);
                        match(ID);
                        match(PERIOD);
                        {
                        switch ( LA(1)) {
                        case XAXIS:
                        {
                              AST tmp52_AST = null;
                              tmp52_AST = astFactory.create(LT(1));
                              astFactory.addASTChild(currentAST,
tmp52_AST);
                              match(XAXIS);
                              break;
                        }
                        case YAXIS:
                        {
                              AST tmp53_AST = null;
                              tmp53_AST = astFactory.create(LT(1));
                              astFactory.addASTChild(currentAST,
tmp53_AST);
                              match(YAXIS);
                              break;
                        }
                        case INTEGER:
                        {
                              AST tmp54_AST = null;
                              tmp54_AST = astFactory.create(LT(1));
                              astFactory.addASTChild(currentAST,
tmp54_AST);
                              match(INTEGER);
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(LT(1),
getFilename());
                        }
                        }
                        }
                        varaccess_AST = (AST)currentAST.root;
                  }
                  else {
```

```
                        throw new NoViableAltException(LT(1),
getFilename());
                  }

            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_13);
            }
            returnAST = varaccess_AST;
      }

      public final void relExpr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST relExpr_AST = null;

            try {      // for error handling
                  {
                  switch ( LA(1)) {
                  case MINUS:
                  {
                        AST tmp55_AST = null;
                        tmp55_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp55_AST);
                        match(MINUS);
                        break;
                  }
                  case TRUE:
                  case FALSE:
                  case CALL:
                  case CARRAT:
                  case LBRACE:
                  case NOT:
                  case ID:
                  case INTEGER:
                  case LPAREN:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  term();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop108:
                  do {
                        if ((LA(1)==PLUS||LA(1)==MINUS)) {
                              addOp();
```

```
                              astFactory.addASTChild(currentAST,
returnAST);
                              term();
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              break _loop108;
                        }

                  } while (true);
                  }
                  relExpr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_14);
            }
            returnAST = relExpr_AST;
      }

      public final void colorFunc() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST colorFunc_AST = null;

            try {      // for error handling
                  AST tmp56_AST = null;
                  tmp56_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp56_AST);
                  match(COLORSTR);
                  factor();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(SEMI);
                  factor();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(SEMI);
                  factor();
                  astFactory.addASTChild(currentAST, returnAST);
                  colorFunc_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_11);
            }
            returnAST = colorFunc_AST;
      }

      public final void drawFunc() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
```

```
		AST drawFunc_AST = null;

		try {	// for error handling
			AST tmp59_AST = null;
			tmp59_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp59_AST);
			match(DRAWSTR);
			{
			switch ( LA(1)) {
			case ID:
			{
				AST tmp60_AST = null;
				tmp60_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp60_AST);
				match(ID);
				break;
			}
			case CARRAT:
			{
				point();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case LBRACE:
			{
				line();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1),
getFilename());
			}
			}
			}
			{
			_loop92:
			do {
				if ((LA(1)==COMMA)) {
					match(COMMA);
					{
					switch ( LA(1)) {
					case ID:
					{
						AST tmp62_AST = null;
						tmp62_AST =
astFactory.create(LT(1));
						astFactory.addASTChild(currentAST,
tmp62_AST);
						match(ID);
						break;
					}
					case CARRAT:
					{
						point();
```

```
                                    astFactory.addASTChild(currentAST,
returnAST);
                                    break;
                              }
                              case LBRACE:
                              {
                                    line();
                                    astFactory.addASTChild(currentAST,
returnAST);
                                    break;
                              }
                              default:
                              {
                                    throw new
NoViableAltException(LT(1), getFilename());
                              }
                              }
                              }
                        }
                        else {
                              break _loop92;
                        }

                  } while (true);
                  }
                  drawFunc_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_11);
            }
            returnAST = drawFunc_AST;
      }

      public final void appendFunc() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST appendFunc_AST = null;

            try {      // for error handling
                  AST tmp63_AST = null;
                  tmp63_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp63_AST);
                  match(APPENDSTR);
                  {
                  switch ( LA(1)) {
                  case ID:
                  {
                        AST tmp64_AST = null;
                        tmp64_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp64_AST);
                        match(ID);
                        break;
                  }
```

```
                    case CARRAT:
                    {
                          point();
                          astFactory.addASTChild(currentAST, returnAST);
                          break;
                    }
                    case LBRACE:
                    {
                          line();
                          astFactory.addASTChild(currentAST, returnAST);
                          break;
                    }
                    default:
                    {
                          throw new NoViableAltException(LT(1),
getFilename());
                    }
                    }
                    }
                    match(TO);
                    {
                    switch ( LA(1)) {
                    case ID:
                    {
                          AST tmp66_AST = null;
                          tmp66_AST = astFactory.create(LT(1));
                          astFactory.addASTChild(currentAST, tmp66_AST);
                          match(ID);
                          break;
                    }
                    case CARRAT:
                    {
                          point();
                          astFactory.addASTChild(currentAST, returnAST);
                          break;
                    }
                    case LBRACE:
                    {
                          line();
                          astFactory.addASTChild(currentAST, returnAST);
                          break;
                    }
                    default:
                    {
                          throw new NoViableAltException(LT(1),
getFilename());
                    }
                    }
                    }
                    appendFunc_AST = (AST)currentAST.root;
              }
              catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_11);
              }
              returnAST = appendFunc_AST;
```

```
	}

	public final void factor() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST factor_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case TRUE:
			case FALSE:
			case CARRAT:
			case LBRACE:
			case INTEGER:
			{
				constant();
				astFactory.addASTChild(currentAST, returnAST);
				factor_AST = (AST)currentAST.root;
				break;
			}
			case ID:
			{
				varaccess();
				astFactory.addASTChild(currentAST, returnAST);
				factor_AST = (AST)currentAST.root;
				break;
			}
			case LPAREN:
			{
				match(LPAREN);
				{
				switch ( LA(1)) {
				case TRUE:
				case FALSE:
				case CARRAT:
				case LBRACE:
				case INTEGER:
				{
					constant();
					astFactory.addASTChild(currentAST,
returnAST);
					break;
				}
				case ID:
				{
					varaccess();
					astFactory.addASTChild(currentAST,
returnAST);
					break;
				}
				default:
				{
					throw new NoViableAltException(LT(1),
getFilename());
				}
```

```
                        }
                        }
                        addOp();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        switch ( LA(1)) {
                        case TRUE:
                        case FALSE:
                        case CARRAT:
                        case LBRACE:
                        case INTEGER:
                        {
                              constant();
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        case ID:
                        {
                              varaccess();
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(LT(1),
getFilename());
                        }
                        }
                        }
                        match(RPAREN);
                        factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case CALL:
                  {
                        funccall();
                        astFactory.addASTChild(currentAST, returnAST);
                        factor_AST = (AST)currentAST.root;
                        break;
                  }
                  case NOT:
                  {
                        AST tmp69_AST = null;
                        tmp69_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp69_AST);
                        match(NOT);
                        factor();
                        astFactory.addASTChild(currentAST, returnAST);
                        factor_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
```

```
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_15);
            }
            returnAST = factor_AST;
      }

      public final void point() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST point_AST = null;

            try {      // for error handling
                  match(CARRAT);
                  {
                  switch ( LA(1)) {
                  case ID:
                  {
                        AST tmp71_AST = null;
                        tmp71_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp71_AST);
                        match(ID);
                        break;
                  }
                  case INTEGER:
                  {
                        AST tmp72_AST = null;
                        tmp72_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp72_AST);
                        match(INTEGER);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  match(COMMA);
                  {
                  switch ( LA(1)) {
                  case ID:
                  {
                        AST tmp74_AST = null;
                        tmp74_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp74_AST);
                        match(ID);
                        break;
                  }
                  case INTEGER:
```

```
                  {
                        AST tmp75_AST = null;
                        tmp75_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp75_AST);
                        match(INTEGER);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  match(CARRAT);
                  point_AST = (AST)currentAST.root;
                  point_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(POINT,"point")).add(point_AST));
                  currentAST.root = point_AST;
                  currentAST.child = point_AST!=null
&&point_AST.getFirstChild()!=null ?
                        point_AST.getFirstChild() : point_AST;
                  currentAST.advanceChildToEnd();
                  point_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_16);
            }
            returnAST = point_AST;
      }

      public final void line() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST line_AST = null;

            try {      // for error handling
                  match(LBRACE);
                  {
                  switch ( LA(1)) {
                  case ID:
                  {
                        AST tmp78_AST = null;
                        tmp78_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp78_AST);
                        match(ID);
                        break;
                  }
                  case CARRAT:
                  {
                        point();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
```

```
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  {
                  int _cnt126=0;
                  _loop126:
                  do {
                        if ((LA(1)==COMMA)) {
                              match(COMMA);
                              {
                              switch ( LA(1)) {
                              case ID:
                              {
                                    AST tmp80_AST = null;
                                    tmp80_AST =
astFactory.create(LT(1));
                                    astFactory.addASTChild(currentAST,
tmp80_AST);
                                    match(ID);
                                    break;
                              }
                              case CARRAT:
                              {
                                    point();
                                    astFactory.addASTChild(currentAST,
returnAST);
                                    break;
                              }
                              default:
                              {
                                    throw new
NoViableAltException(LT(1), getFilename());
                              }
                              }
                              }
                        }
                        else {
                              if ( _cnt126>=1 ) { break _loop126; }
else {throw new NoViableAltException(LT(1), getFilename());}
                        }

                        _cnt126++;
                  } while (true);
                  }
                  match(RBRACE);
                  line_AST = (AST)currentAST.root;
                  line_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(LINE,"line")).add(line_AST));
                  currentAST.root = line_AST;
                  currentAST.child = line_AST!=null
&&line_AST.getFirstChild()!=null ?
                        line_AST.getFirstChild() : line_AST;
```

```
                  currentAST.advanceChildToEnd();
                  line_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_17);
            }
            returnAST = line_AST;
      }

      public final void primaryExpr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST primaryExpr_AST = null;

            try {      // for error handling
                  eqeqExpr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  switch ( LA(1)) {
                  case EQEQ:
                  case NEQ:
                  {
                        eqeqOp();
                        astFactory.addASTChild(currentAST, returnAST);
                        eqeqExpr();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case THEN:
                  case DO:
                  case AND:
                  case OR:
                  case SEMI:
                  case RBRACKET:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  primaryExpr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_18);
            }
            returnAST = primaryExpr_AST;
      }
```

```
	public final void primaryOp() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST primaryOp_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case AND:
			{
				AST tmp82_AST = null;
				tmp82_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp82_AST);
				match(AND);
				primaryOp_AST = (AST)currentAST.root;
				break;
			}
			case OR:
			{
				AST tmp83_AST = null;
				tmp83_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp83_AST);
				match(OR);
				primaryOp_AST = (AST)currentAST.root;
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1),
getFilename());
			}
			}
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_19);
		}
		returnAST = primaryOp_AST;
	}

	public final void eqeqExpr() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST eqeqExpr_AST = null;

		try {	// for error handling
			relExpr();
			astFactory.addASTChild(currentAST, returnAST);
			{
			switch ( LA(1)) {
			case GREATER:
			case LESS:
```

```
                  case GEQ:
                  case LEQ:
                  {
                        relOp();
                        astFactory.addASTChild(currentAST, returnAST);
                        relExpr();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case THEN:
                  case DO:
                  case AND:
                  case OR:
                  case EQEQ:
                  case NEQ:
                  case SEMI:
                  case RBRACKET:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  eqeqExpr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_20);
            }
            returnAST = eqeqExpr_AST;
      }

      public final void eqeqOp() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST eqeqOp_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case EQEQ:
                  {
                        AST tmp84_AST = null;
                        tmp84_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp84_AST);
                        match(EQEQ);
                        eqeqOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case NEQ:
                  {
```

```
                        AST tmp85_AST = null;
                        tmp85_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp85_AST);
                        match(NEQ);
                        eqeqOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_19);
            }
            returnAST = eqeqOp_AST;
      }

      public final void relOp() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST relOp_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case LESS:
                  {
                        AST tmp86_AST = null;
                        tmp86_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp86_AST);
                        match(LESS);
                        relOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case LEQ:
                  {
                        AST tmp87_AST = null;
                        tmp87_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp87_AST);
                        match(LEQ);
                        relOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case GREATER:
                  {
                        AST tmp88_AST = null;
                        tmp88_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp88_AST);
                        match(GREATER);
                        relOp_AST = (AST)currentAST.root;
                        break;
```

```
                  }
                  case GEQ:
                  {
                        AST tmp89_AST = null;
                        tmp89_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp89_AST);
                        match(GEQ);
                        relOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_19);
            }
            returnAST = relOp_AST;
      }

      public final void term() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST term_AST = null;

            try {      // for error handling
                  factor();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop112:
                  do {
                        if (((LA(1) >= MUL && LA(1) <= MOD))) {
                              mulOp();
                              astFactory.addASTChild(currentAST,
returnAST);
                              factor();
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              break _loop112;
                        }

                  } while (true);
                  }
                  term_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
```

```
                consumeUntil(_tokenSet_21);
            }
            returnAST = term_AST;
      }

      public final void addOp() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST addOp_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case PLUS:
                  {
                        AST tmp90_AST = null;
                        tmp90_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp90_AST);
                        match(PLUS);
                        addOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case MINUS:
                  {
                        AST tmp91_AST = null;
                        tmp91_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp91_AST);
                        match(MINUS);
                        addOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_22);
            }
            returnAST = addOp_AST;
      }

      public final void mulOp() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST mulOp_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case MUL:
```

```
                  {
                        AST tmp92_AST = null;
                        tmp92_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp92_AST);
                        match(MUL);
                        mulOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case DIV:
                  {
                        AST tmp93_AST = null;
                        tmp93_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp93_AST);
                        match(DIV);
                        mulOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case MOD:
                  {
                        AST tmp94_AST = null;
                        tmp94_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp94_AST);
                        match(MOD);
                        mulOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_22);
            }
            returnAST = mulOp_AST;
      }

      public final void constant() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST constant_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case INTEGER:
                  {
                        AST tmp95_AST = null;
                        tmp95_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp95_AST);
                        match(INTEGER);
                        constant_AST = (AST)currentAST.root;
```

```
                        break;
                  }
                  case TRUE:
                  case FALSE:
                  {
                        bool();
                        astFactory.addASTChild(currentAST, returnAST);
                        constant_AST = (AST)currentAST.root;
                        break;
                  }
                  case CARRAT:
                  {
                        point();
                        astFactory.addASTChild(currentAST, returnAST);
                        constant_AST = (AST)currentAST.root;
                        break;
                  }
                  case LBRACE:
                  {
                        line();
                        astFactory.addASTChild(currentAST, returnAST);
                        constant_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_23);
            }
            returnAST = constant_AST;
      }

      public final void funccall() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funccall_AST = null;

            try {      // for error handling
                  AST tmp96_AST = null;
                  tmp96_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp96_AST);
                  match(CALL);
                  AST tmp97_AST = null;
                  tmp97_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp97_AST);
                  match(ID);
                  match(LPARENS);
                  {
```

```
                  switch ( LA(1)) {
                  case TRUE:
                  case FALSE:
                  case CARRAT:
                  case LBRACE:
                  case ID:
                  case INTEGER:
                  {
                        arglist();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case RPARENS:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  match(RPARENS);
                  match(SEMI);
                  funccall_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_15);
            }
            returnAST = funccall_AST;
      }

      public final void bool() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST bool_AST = null;

            try {      // for error handling
                  switch ( LA(1)) {
                  case TRUE:
                  {
                        AST tmp101_AST = null;
                        tmp101_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp101_AST);
                        match(TRUE);
                        bool_AST = (AST)currentAST.root;
                        break;
                  }
                  case FALSE:
                  {
                        AST tmp102_AST = null;
                        tmp102_AST = astFactory.create(LT(1));
```

```
                        astFactory.addASTChild(currentAST, tmp102_AST);
                        match(FALSE);
                        bool_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_23);
            }
            returnAST = bool_AST;
      }

      public final void arglist() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST arglist_AST = null;

            try {      // for error handling
                  {
                  switch ( LA(1)) {
                  case TRUE:
                  case FALSE:
                  case CARRAT:
                  case LBRACE:
                  case INTEGER:
                  {
                        constant();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case ID:
                  {
                        AST tmp103_AST = null;
                        tmp103_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp103_AST);
                        match(ID);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(LT(1),
getFilename());
                  }
                  }
                  }
                  {
                  _loop133:
```

```
                  do {
                        if ((LA(1)==COMMA)) {
                              match(COMMA);
                              {
                              switch ( LA(1)) {
                              case TRUE:
                              case FALSE:
                              case CARRAT:
                              case LBRACE:
                              case INTEGER:
                              {
                                    constant();
                                    astFactory.addASTChild(currentAST,
returnAST);
                                    break;
                              }
                              case ID:
                              {
                                    AST tmp105_AST = null;
                                    tmp105_AST =
astFactory.create(LT(1));
                                    astFactory.addASTChild(currentAST,
tmp105_AST);
                                    match(ID);
                                    break;
                              }
                              default:
                              {
                                    throw new
NoViableAltException(LT(1), getFilename());
                              }
                              }
                              }
                        }
                        else {
                              break _loop133;
                        }

                  } while (true);
                  }
                  arglist_AST = (AST)currentAST.root;
                  arglist_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(ARGLIST,"arglist")).add(arglist_AST)
);
                  currentAST.root = arglist_AST;
                  currentAST.child = arglist_AST!=null
&&arglist_AST.getFirstChild()!=null ?
                        arglist_AST.getFirstChild() : arglist_AST;
                  currentAST.advanceChildToEnd();
                  arglist_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_4);
            }
            returnAST = arglist_AST;
```

```
}


public static final String[] _tokenNames = {
	"<0>",
	"EOF",
	"<2>",
	"NULL_TREE_LOOKAHEAD",
	"\"Int\"",
	"\"Line\"",
	"\"Point\"",
	"\"Boolean\"",
	"\"If\"",
	"\"Then\"",
	"\"Else\"",
	"\"IfEnd\"",
	"\"While\"",
	"\"Do\"",
	"\"WhileEnd\"",
	"\"For\"",
	"\"ForEnd\"",
	"\"ForLine\"",
	"\"In\"",
	"\"XAxis\"",
	"\"YAxis\"",
	"\"Return\"",
	"\"Color\"",
	"\"Draw\"",
	"\"Append\"",
	"\"True\"",
	"\"False\"",
	"\"Let\"",
	"\"To\"",
	"\"Call\"",
	"\"Global\"",
	"\"End\"",
	"PLUS",
	"MINUS",
	"MUL",
	"DIV",
	"MOD",
	"ASSIGN",
	"COMMA",
	"PERIOD",
	"GREATER",
	"LESS",
	"CARRAT",
	"GEQ",
	"LEQ",
	"AND",
	"OR",
	"EQEQ",
	"NEQ",
	"LPARENS",
	"RPARENS",
	"SEMI",
	"LBRACE",
```

```
            "RBRACE",
            "BRACKET",
            "LBRACKET",
            "RBRACKET",
            "NOT",
            "LETTER",
            "DIGIT",
            "ID",
            "INTEGER",
            "EXPONENT",
            "WS",
            "COMMENT",
            "PROGRAM",
            "GLOBVARDECL",
            "FUNCDECL",
            "VARDECL",
            "PARAMLIST",
            "STMTBLOCK",
            "ASSIGNSTMT",
            "EXPR",
            "POINT",
            "LINE",
            "ARGLIST",
            "LPAREN",
            "RPAREN"
      };

      protected void buildTokenTypeASTClassMap() {
            tokenTypeToASTClassMap=null;
      };

      private static final long[] mk_tokenSet_0() {
            long[] data = { 2L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
      private static final long[] mk_tokenSet_1() {
            long[] data = { 1073742064L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_1 = new
BitSet(mk_tokenSet_1());
      private static final long[] mk_tokenSet_2() {
            long[] data = { 242L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_2 = new
BitSet(mk_tokenSet_2());
      private static final long[] mk_tokenSet_3() {
            long[] data = { 1152921504606846976L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_3 = new
BitSet(mk_tokenSet_3());
      private static final long[] mk_tokenSet_4() {
            long[] data = { 1125899906842624L, 0L};
```

```
            return data;
      }
      public static final BitSet _tokenSet_4 = new
BitSet(mk_tokenSet_4());
      private static final long[] mk_tokenSet_5() {
            long[] data = { 3607387708963590400L, 4096L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_5 = new
BitSet(mk_tokenSet_5());
      private static final long[] mk_tokenSet_6() {
            long[] data = { 2147568640L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_6 = new
BitSet(mk_tokenSet_6());
      private static final long[] mk_tokenSet_7() {
            long[] data = { 1126174784749568L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_7 = new
BitSet(mk_tokenSet_7());
      private static final long[] mk_tokenSet_8() {
            long[] data = { 3607387711111159040L, 4096L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_8 = new
BitSet(mk_tokenSet_8());
      private static final long[] mk_tokenSet_9() {
            long[] data = { 74309393851621888L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_9 = new
BitSet(mk_tokenSet_9());
      private static final long[] mk_tokenSet_10() {
            long[] data = { 2251799813693440L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_10 = new
BitSet(mk_tokenSet_10());
      private static final long[] mk_tokenSet_11() {
            long[] data = { 2251799813685248L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_11 = new
BitSet(mk_tokenSet_11());
      private static final long[] mk_tokenSet_12() {
            long[] data = { 110895913848807936L, 8192L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_12 = new
BitSet(mk_tokenSet_12());
      private static final long[] mk_tokenSet_13() {
            long[] data = { 74867116829843968L, 8192L, 0L, 0L};
            return data;
      }
```

```
      public static final BitSet _tokenSet_13 = new
BitSet(mk_tokenSet_13());
      private static final long[] mk_tokenSet_14() {
            long[] data = { 74866846246904320L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_14 = new
BitSet(mk_tokenSet_14());
      private static final long[] mk_tokenSet_15() {
            long[] data = { 74866979390890496L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_15 = new
BitSet(mk_tokenSet_15());
      private static final long[] mk_tokenSet_16() {
            long[] data = { 85000353698816512L, 8192L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_16 = new
BitSet(mk_tokenSet_16());
      private static final long[] mk_tokenSet_17() {
            long[] data = { 75993154444075520L, 8192L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_17 = new
BitSet(mk_tokenSet_17());
      private static final long[] mk_tokenSet_18() {
            long[] data = { 74414946967888384L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_18 = new
BitSet(mk_tokenSet_18());
      private static final long[] mk_tokenSet_19() {
            long[] data = { 3607387708797747200L, 4096L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_19 = new
BitSet(mk_tokenSet_19());
      private static final long[] mk_tokenSet_20() {
            long[] data = { 74837159432954368L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_20 = new
BitSet(mk_tokenSet_20());
      private static final long[] mk_tokenSet_21() {
            long[] data = { 74866859131806208L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_21 = new
BitSet(mk_tokenSet_21());
      private static final long[] mk_tokenSet_22() {
            long[] data = { 3607387700207812608L, 4096L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_22 = new
BitSet(mk_tokenSet_22());
      private static final long[] mk_tokenSet_23() {
```

```
            long[] data = { 75993154175640064L, 8192L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_23 = new
BitSet(mk_tokenSet_23());

      }
```

BatsParser.java

```
// $ANTLR 2.7.2: "BATS.g" -> "BATSLexer.java"$

public interface BATSParserTokenTypes {
      int EOF = 1;
      int NULL_TREE_LOOKAHEAD = 3;
      int INT = 4;
      int LINESTR = 5;
      int POINTSTR = 6;
      int BOOLSTR = 7;
      int IFSTR = 8;
      int THEN = 9;
      int ELSE = 10;
      int IFEND = 11;
      int WHILESTR = 12;
      int DO = 13;
      int WHILEEND = 14;
      int FORSTR = 15;
      int FOREND = 16;
      int FORLINESTR = 17;
      int IN = 18;
      int XAXIS = 19;
      int YAXIS = 20;
      int RETURNSTR = 21;
      int COLORSTR = 22;
      int DRAWSTR = 23;
      int APPENDSTR = 24;
      int TRUE = 25;
      int FALSE = 26;
      int LET = 27;
      int TO = 28;
      int CALL = 29;
      int GLOBAL = 30;
      int END = 31;
      int PLUS = 32;
      int MINUS = 33;
      int MUL = 34;
      int DIV = 35;
      int MOD = 36;
      int ASSIGN = 37;
      int COMMA = 38;
      int PERIOD = 39;
      int GREATER = 40;
      int LESS = 41;
      int CARRAT = 42;
      int GEQ = 43;
      int LEQ = 44;
      int AND = 45;
```

```
      int OR = 46;
      int EQEQ = 47;
      int NEQ = 48;
      int LPARENS = 49;
      int RPARENS = 50;
      int SEMI = 51;
      int LBRACE = 52;
      int RBRACE = 53;
      int BRACKET = 54;
      int LBRACKET = 55;
      int RBRACKET = 56;
      int NOT = 57;
      int LETTER = 58;
      int DIGIT = 59;
      int ID = 60;
      int INTEGER = 61;
      int EXPONENT = 62;
      int WS = 63;
      int COMMENT = 64;
      int PROGRAM = 65;
      int GLOBVARDECL = 66;
      int FUNCDECL = 67;
      int VARDECL = 68;
      int PARAMLIST = 69;
      int STMTBLOCK = 70;
      int ASSIGNSTMT = 71;
      int EXPR = 72;
      int POINT = 73;
      int LINE = 74;
      int ARGLIST = 75;
      int LPAREN = 76;
      int RPAREN = 77;
}
```

BatsPoint.java

```
import java.io.PrintWriter;
import antlr.CommonAST;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * changed by Tanya
 * @version $Id: BatsDataType.java,v 1.17 2003/05/12 23:44:34 hanhua
Exp $
 */
public class BatsPoint extends BatsDataType
{
    int xValue, yValue;

    public BatsPoint( ) {
    }
    public BatsPoint(  String name,String scope) {
        super(name,scope,"Point","var");
```

```
    }
    public BatsPoint( int x, int y) {
        super.setValue("^"+x+","+y+"^");
        this.xValue = x;
        this.yValue = y;
    }
    public BatsPoint( BatsDataType bd ) {
        this.xValue = 0;
        this.yValue = 0;
    }
     public BatsPoint( BatsPoint bp ) {
        this.xValue = bp.getXValue();
        this.yValue = bp.getYValue();
    }


    public void setXValue(int x){
         this.xValue = x;
    }
    public int getXValue(){
         return this.xValue;
    }
    public void setYValue(int y){
         this.yValue = y;
    }
    public int getYValue(){
         return this.yValue;
    }

}
```

BatsProgram.java: //This is an example that is generated

```
import java.util.*;

public class BatsProgram{

 BatsBuiltIn builtIn = new BatsBuiltIn();

 BatsColor color = new BatsColor();
BatsLine Xshape, Yshape, temp;
boolean Begin(){
int w, z, dacolor;
 w = 300 ;

 z = 30 ;

 dacolor = 20 ;
while( w > 0 ){

 temp =  new BatsLine();
  temp.addPoint( new BatsPoint( z, 0));

 temp.addPoint( new BatsPoint( 0, w) );
```

```
 color = new BatsColor(0,dacolor, 0);
 builtIn.append( color );

 builtIn.append( temp );

 z = z + 30 ;

 w = w - 30 ;

 dacolor = dacolor + 20 ;

}

 return true;

}

 public static void main(String[] argc){
    BatsProgram bpr = new BatsProgram();
    bpr.Begin();
  bpr.builtIn.drawLine();
 }
}
```

BatstokensTokenTypes.java:

```
// $ANTLR 2.7.2: "BATS.g" -> "BATSLexer.java"$

public interface BATStokensTokenTypes {
      int EOF = 1;
      int NULL_TREE_LOOKAHEAD = 3;
      int INT = 4;
      int LINESTR = 5;
      int POINTSTR = 6;
      int BOOLSTR = 7;
      int IFSTR = 8;
      int THEN = 9;
      int ELSE = 10;
      int IFEND = 11;
      int WHILESTR = 12;
      int DO = 13;
      int WHILEEND = 14;
      int FORSTR = 15;
      int FOREND = 16;
      int FORLINESTR = 17;
      int IN = 18;
      int XAXIS = 19;
      int YAXIS = 20;
      int RETURNSTR = 21;
      int COLORSTR = 22;
      int DRAWSTR = 23;
      int APPENDSTR = 24;
      int TRUE = 25;
      int FALSE = 26;
      int LET = 27;
```

```
	int TO = 28;
	int CALL = 29;
	int GLOBAL = 30;
	int END = 31;
	int PLUS = 32;
	int MINUS = 33;
	int MUL = 34;
	int DIV = 35;
	int MOD = 36;
	int ASSIGN = 37;
	int COMMA = 38;
	int PERIOD = 39;
	int GREATER = 40;
	int LESS = 41;
	int CARRAT = 42;
	int GEQ = 43;
	int LEQ = 44;
	int AND = 45;
	int OR = 46;
	int EQEQ = 47;
	int NEQ = 48;
	int LPARENS = 49;
	int RPARENS = 50;
	int SEMI = 51;
	int LBRACE = 52;
	int RBRACE = 53;
	int BRACKET = 54;
	int LBRACKET = 55;
	int RBRACKET = 56;
	int NOT = 57;
	int LETTER = 58;
	int DIGIT = 59;
	int ID = 60;
	int INTEGER = 61;
	int EXPONENT = 62;
	int WS = 63;
	int COMMENT = 64;
}
```

BatsTranslator.java:

```
import antlr.*;
import antlr.Token;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.ASTFactory;


/**
 * The class is created as a translator for Bats program in to
BatsProgram.java
 * with the following execution and creating 'input' file for drawing
purpoces
 *
```

```
 * @author Tanya
 * @version $Id: .java,v 1.17 2004/12/18 23:44:34
 */

public class BatsTranslator{
   CommonAST _t;
   StringBuffer str;
   AST letId;
   boolean forOn,forEndSt;

   public BatsTranslator( CommonAST t){
      _t = t;
      str = new StringBuffer();
      str.append("import java.util.*;\n\n");
      str.append("public class BatsProgram{\n");
      str.append("\n BatsBuiltIn builtIn = new BatsBuiltIn();\n");
      str.append("\n BatsColor color = new BatsColor();\n");
   }

   public void interpret(){
      // step thru top level nodes

      CommonAST topChild = (CommonAST) _t.getFirstChild();

      doTopElement(topChild);

      while( (topChild = (CommonAST)topChild.getNextSibling()) != null
) {
         doTopElement(topChild);
      }

      str.append("\n public static void main(String[] argc){ \n"+
                 "    BatsProgram bpr = new BatsProgram();\n"+
                 "    bpr.Begin();\n  bpr.builtIn.drawLine(); ");
      str.append("\n }\n}");
     try{
      FileOutputStream fos = new FileOutputStream("BatsProgram.java");
      DataOutputStream dos = new DataOutputStream(fos);
      dos.writeBytes(str.toString());
     }catch(IOException e){System.out.println("IOException
"+e.toString());}

   }


   void doTopElement(CommonAST topChild) {

      String nodeType = topChild.getText();

      // here is two possible options globvarDecl and funcDecl
      if ("globvarDecl".equalsIgnoreCase(nodeType)) {
         doGlobalVar(topChild);
      }  else if ("funcDecl".equalsIgnoreCase(nodeType)) {
         doFunction(topChild);
         str.append("\n}\n");
      }
```

```
   } // doTopElement(CommonAST topChild)


   void doGlobalVar(CommonAST globVar) {

      // ignore the first one "Global"
     if( globVar.getFirstChild().getNextSibling() != null){
      AST varChild = globVar.getFirstChild().getNextSibling();

      String varType = varChild.getText();
      str.append(BatsKeywordMap.toJava(varType));
      str.append(" ");

      varChild = varChild.getNextSibling();

      String firstId = varChild.getText();
      str.append(firstId);


      while((varChild = varChild.getNextSibling()) != null){
         String varId   = varChild.getText();
         str.append(", ");
         str.append(varId);
      }
      str.append(";\n");
      }

   }


   void doFunction(CommonAST function) {
      String retType;
      String funId;
      AST argList;

      //1. Function return type in declaration
      AST funChild = function.getFirstChild();
      retType = funChild.getText();

      //2. next funtion id
      funChild = funChild.getNextSibling();
      funId = funChild.getText();
      str.append(BatsKeywordMap.toJava(retType) + " " + funId + "(" );

      //3. get arg list
      argList = funChild.getNextSibling();

         // go thru arg list children
         if(argList.getFirstChild() != null){
            AST argListElement = argList.getFirstChild();


            str.append(argListElement.getText()+"");
            boolean pair = false;
            while((argListElement = argListElement.getNextSibling()) !=
null){
               if(pair){
```

```
                   str.append(", ");
                   pair = true;
                }
                str.append(" "+argListElement.getText());
                pair = !pair;
             }
          }
            str.append("){\n");

      //4. varDecl in function
      AST varDecl = argList.getNextSibling();

      if( varDecl.getFirstChild() != null){
          AST varType = varDecl.getFirstChild();
          String fvarType = varType.getText();
          str.append(BatsKeywordMap.toJava(fvarType) + " ");

          while((varType = varType.getNextSibling()) != null){
            if(BatsKeywordMap.toJava(varType.getText()) != null){

str.append(";\n"+BatsKeywordMap.toJava(varType.getText()) + " ");
               continue;
            }
            else{
               str.append(varType.getText());
               if(varType.getNextSibling() != null &&

BatsKeywordMap.toJava(varType.getNextSibling().getText()) == null)
                  str.append(", ");
            }

         }//while
      str.append(";");

      }//if

       //5. stmtblock

       AST stmtblock = varDecl.getNextSibling();


       AST stmt  = stmtblock.getFirstChild();
       doStmt(stmt);

   }
/*************************************************************/
    void  doStmt(AST stmt){
       String stmtName = stmt.getText();
       if(stmtName.equals("assignstmt")){
         doAssigstmt(stmt);

       }//if
       else if(stmtName.equals("If")){
         AST expr = stmt.getFirstChild();
         str.append("if(");
         doStmt(expr);
       }
```

```
    else if(stmtName.equals("expr")) {
      doExpr(stmt);
    }
    else if(stmtName.equals("stmtblock")) {
      stmt = stmt.getFirstChild();
      doStmt(stmt);
      return;
    }
    if(stmtName.equals("Return")) {
      stmt = stmt.getFirstChild();
      str.append("\n return "+ stmt.getText().toLowerCase()+";\n");
      return;
    }
    else if(stmtName.equals("While")) {
      AST expr = stmt.getFirstChild();
      str.append("while(");
      doStmt(expr);
      str.append("\n}\n");
     } //if
    else if(stmtName.equals("ForLine")) {
      doForLine(stmt);
     } //if
    else if(stmtName.equals("For")) {
      forOn = true;
      doFor(stmt);
     } //if
    else if(stmtName.equals("Draw")) {
      AST pl = stmt.getFirstChild();
      str.append("\n builtIn.append( "+pl+" );\n");
    } //if
    else if(stmtName.equals("Color")) {
      doColor(stmt);
    } //if
    if((stmt  = stmt.getNextSibling()) != null){
       doStmt(stmt);
    }
 }
 /*************************************************************/
 void  doFor(AST stmt){

      str.append("  \n for(int ");
      AST assigStmt = stmt.getFirstChild();
      forOn = true;
      doAssigstmt(assigStmt);
      AST expr = assigStmt.getNextSibling();
      doExpr(expr);
      forOn = false;
      forEndSt = true;
      assigStmt = expr.getNextSibling();
      doStmt(assigStmt);

      str.append("\n}\n");

 }
/*************************************************************/
void doForLine(AST stmtFor){
     AST indPt = stmtFor.getFirstChild();
```

```
       if(indPt.getNextSibling() != null){
         AST line = indPt.getNextSibling();
         str.append("\n Vector points = "+line+".getLine();\n");
         str.append("   for(int i = 0; i < points.size(); i++){\n");
         str.append("     BatsPoint bp = new
BatsPoint((BatsPoint)points.elementAt(i));\n");
         str.append("     builtIn.append(bp); \n  }\n");
       }

  }
 /*****************************************************************/
  void doColor(AST color){
      AST red = new CommonAST();
      AST green= new CommonAST();
      AST blue = new CommonAST();
      if(color.getFirstChild() != null){
         red = color.getFirstChild();
         if(red != null && red.getNextSibling() != null){
            green =  red.getNextSibling();
         }
         if(green != null && green.getNextSibling() != null){
            blue =  green.getNextSibling();
         }
      }
System.out.println("----in doCOLOR:== "+red+green+blue);
//    str.append("\n color.setRBG("+red+","+green+", "+blue+");");
      str.append("\n color = new BatsColor("+red+","+green+",
"+blue+");");
      str.append("\n builtIn.append( color );\n");
 }
/*****************************************************************/
   void  doAssigstmt(AST stmt){
         AST letIdN;

         letId = stmt.getFirstChild().getNextSibling();
         if(forOn){
            str.append(letId + " = ");
         }else{
            str.append("\n "+letId + " = ");
         }
         while((letIdN = letId.getNextSibling()) != null){

            if((letIdN.getText()).equalsIgnoreCase("point")) {
              doPoint(letIdN);
              str.append(";\n");
              return;
            }
            else if((letIdN.getText()).equalsIgnoreCase("line")) {

              str.append(" new BatsLine();\n ");
              doLine(letIdN);
              return;
            }
             else{ // if no line nor Point

                str.append(letIdN +" ");
```

```
            while((letIdN = letIdN.getNextSibling()) != null){
                 str.append(letIdN+ " ");
            }
            if(forEndSt){
              str.append("){\n");
              forEndSt = false;
              return;
            }else{
               str.append(";\n");
               return;
            }
         }
        }
    }
/**************************************************************************
*/
    void  doExpr(AST expr){
         AST leftOp = expr.getFirstChild();
         str.append(" "+leftOp + " ");

         while((leftOp = leftOp.getNextSibling()) != null){
            str.append(leftOp +" ");
         }
         if(forOn){
            str.append(";");
         }else{
           str.append("){\n");
         }
    }

/**************************************************************************
*/
    void  doPoint(AST point){
         point = point.getFirstChild();
         //add X

         str.append(" new BatsPoint( "+point + ", ");

         //add Y
         while((point = point.getNextSibling()) != null){
             str.append(point +") ");
         }
      //   str.append(";\n");
    }
/**************************************************************************
*/
    void  doLine(AST line){
      AST pointN;

        if(line.getText().equals("point")){
           str.append("\n "+letId+".addPoint(");
           doPoint(line);
           str.append("); ");
        }
        else if(line.getFirstChild() != null){

           AST point = line.getFirstChild();
```

```
            if((point.getText()).equalsIgnoreCase("point")) {

            str.append(" "+letId+".addPoint(");
             pointN = point.getFirstChild();
             str.append(" new BatsPoint( "+pointN + ", ");

               while((pointN = pointN.getNextSibling()) != null){
                     str.append(pointN+"));\n");
               }
                while((point = point.getNextSibling()) != null)
                   doLine(point);
             } else{

                 str.append(" "+letId+".addPoint("+point+");");

                 while((point = point.getNextSibling()) != null){

                   if((point.getText()).equalsIgnoreCase("point")) {
                      str.append(" "+letId+".addPoint(bp);\n");
                       doPoint(point);
                   }else {

                      str.append(" "+letId+".addPoint("+point+");");
                   }
                  }//while

              }//else
          }//if there are points
    }

} // class
```

BatsWalker.java

```
// $ANTLR 2.7.2: "BatsWalker.g" -> "BatsWalker.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
import java.util.*;
import antlr.CommonAST;
```

```
public class BatsWalker extends antlr.TreeParser       implements
BatsWalkerTokenTypes
 {

//  BatsInterpreter biptr = new BatsInterpreter();
 // BatsBuiltIn builtIn = new BatsBuiltIn();
//  BatsSymbolT smt = new BatsSymbolT();
public BatsWalker() {
      tokenNames = _tokenNames;
}

      public final  CommonAST  program(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST program_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST program_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t2 = _t;
                  AST tmp1_AST = null;
                  AST tmp1_AST_in = null;
                  tmp1_AST = astFactory.create((AST)_t);
                  tmp1_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp1_AST);
                  ASTPair __currentAST2 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,PROGRAM);
                  _t = _t.getFirstChild();
                  {
                  _loop4:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==GLOBVARDECL)) {
                              a=globvarDecl(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              break _loop4;
                        }

                  } while (true);
                  }
                  {
                  int _cnt6=0;
```

```
                  _loop6:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==FUNCDECL)) {
                              b=funcDecl(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              if ( _cnt6>=1 ) { break _loop6; } else
{throw new NoViableAltException(_t);}
                        }

                        _cnt6++;
                  } while (true);
                  }
                  {
                  _loop8:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==CALL)) {
                              c=funccall(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              break _loop8;
                        }

                  } while (true);
                  }
                  currentAST = __currentAST2;
                  _t = __t2;
                  _t = _t.getNextSibling();
                  program_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = program_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  globvarDecl(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST globvarDecl_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST globvarDecl_AST = null;
            AST t_AST = null;
            AST t = null;
```

```
            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t10 = _t;
                  AST tmp2_AST = null;
                  AST tmp2_AST_in = null;
                  tmp2_AST = astFactory.create((AST)_t);
                  tmp2_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp2_AST);
                  ASTPair __currentAST10 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,GLOBVARDECL);
                  _t = _t.getFirstChild();
                  AST tmp3_AST = null;
                  AST tmp3_AST_in = null;
                  tmp3_AST = astFactory.create((AST)_t);
                  tmp3_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp3_AST);
                  match(_t,GLOBAL);
                  _t = _t.getNextSibling();
                  t = _t==ASTNULL ? null : (AST)_t;
                  type(_t);
                  _t = _retTree;
                  t_AST = (AST)returnAST;
                  astFactory.addASTChild(currentAST, returnAST);
                  v=var(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  currentAST = __currentAST10;
                  _t = __t10;
                  _t = _t.getNextSibling();
                  globvarDecl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = globvarDecl_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  funcDecl(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST funcDecl_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funcDecl_AST = null;
```

```
AST t_AST = null;
AST t = null;
AST funName = null;
AST funName_AST = null;

CommonAST a,b,c;
Vector v = new Vector();
r = null;
a = new CommonAST();



try {      // for error handling
      AST __t12 = _t;
      AST tmp4_AST = null;
      AST tmp4_AST_in = null;
      tmp4_AST = astFactory.create((AST)_t);
      tmp4_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp4_AST);
      ASTPair __currentAST12 = currentAST.copy();
      currentAST.root = currentAST.child;
      currentAST.child = null;
      match(_t,FUNCDECL);
      _t = _t.getFirstChild();
      t = _t==ASTNULL ? null : (AST)_t;
      type(_t);
      _t = _retTree;
      t_AST = (AST)returnAST;
      astFactory.addASTChild(currentAST, returnAST);
      funName = (AST)_t;
      AST funName_AST_in = null;
      funName_AST = astFactory.create(funName);
      astFactory.addASTChild(currentAST, funName_AST);
      match(_t,ID);
      _t = _t.getNextSibling();
      a=paramlist(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      {
      _loop14:
      do {
            if (_t==null) _t=ASTNULL;
            if ((_t.getType()==VARDECL)) {
                  b=varDecl(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST,
returnAST);
            }
            else {
                  break _loop14;
            }

      } while (true);
      }
      {
      a=stmtblock(_t);
      _t = _retTree;
```

```
                astFactory.addASTChild(currentAST, returnAST);
                }
                currentAST = __currentAST12;
                _t = __t12;
                _t = _t.getNextSibling();
                funcDecl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = funcDecl_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  funccall(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST funccall_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funccall_AST = null;
            AST fc = null;
            AST fc_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t114 = _t;
                  AST tmp5_AST = null;
                  AST tmp5_AST_in = null;
                  tmp5_AST = astFactory.create((AST)_t);
                  tmp5_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp5_AST);
                  ASTPair __currentAST114 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,CALL);
                  _t = _t.getFirstChild();
                  System.out.println("CALL");
                  fc = (AST)_t;
                  AST fc_AST_in = null;
                  fc_AST = astFactory.create(fc);
                  astFactory.addASTChild(currentAST, fc_AST);
                  match(_t,ID);
                  _t = _t.getNextSibling();
                  a=arglist(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  currentAST = __currentAST114;
```

```
            _t = __t114;
            _t = _t.getNextSibling();
            funccall_AST = (AST)currentAST.root;
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      returnAST = funccall_AST;
      _retTree = _t;
      return r ;
}

public final void type(AST _t) throws RecognitionException {

      AST type_AST_in = (AST)_t;
      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST type_AST = null;

      try {      // for error handling
            {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case INT:
            {
                  AST tmp6_AST = null;
                  AST tmp6_AST_in = null;
                  tmp6_AST = astFactory.create((AST)_t);
                  tmp6_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp6_AST);
                  match(_t,INT);
                  _t = _t.getNextSibling();
                  break;
            }
            case LINESTR:
            {
                  AST tmp7_AST = null;
                  AST tmp7_AST_in = null;
                  tmp7_AST = astFactory.create((AST)_t);
                  tmp7_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp7_AST);
                  match(_t,LINESTR);
                  _t = _t.getNextSibling();
                  break;
            }
            case POINTSTR:
            {
                  AST tmp8_AST = null;
                  AST tmp8_AST_in = null;
                  tmp8_AST = astFactory.create((AST)_t);
                  tmp8_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp8_AST);
                  match(_t,POINTSTR);
                  _t = _t.getNextSibling();
                  break;
            }
```

```
                case BOOLSTR:
                {
                      AST tmp9_AST = null;
                      AST tmp9_AST_in = null;
                      tmp9_AST = astFactory.create((AST)_t);
                      tmp9_AST_in = (AST)_t;
                      astFactory.addASTChild(currentAST, tmp9_AST);
                      match(_t,BOOLSTR);
                      _t = _t.getNextSibling();
                      break;
                }
                default:
                {
                      throw new NoViableAltException(_t);
                }
                }
                }
                type_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
          }
          returnAST = type_AST;
          _retTree = _t;
    }

    public final  Vector  var(AST _t) throws RecognitionException {
           Vector v ;

          AST var_AST_in = (AST)_t;
          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST var_AST = null;
          AST p = null;
          AST p_AST = null;

          v=new Vector();


          try {      // for error handling
                {
                int _cnt121=0;
                _loop121:
                do {
                      if (_t==null) _t=ASTNULL;
                      if ((_t.getType()==ID)) {
                            p = (AST)_t;
                            AST p_AST_in = null;
                            p_AST = astFactory.create(p);
                            astFactory.addASTChild(currentAST,
p_AST);
                            match(_t,ID);
                            _t = _t.getNextSibling();
                            v.add( p.getText() );
                      }
                      else {
```

```
                              if ( _cnt121>=1 ) { break _loop121; }
else {throw new NoViableAltException(_t);}
                        }

                        _cnt121++;
                  } while (true);
                  }
                  var_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = var_AST;
            _retTree = _t;
            return v ;
      }

      public final  CommonAST  paramlist(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST paramlist_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST paramlist_AST = null;
            AST pt_AST = null;
            AST pt = null;
            AST pn = null;
            AST pn_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t23 = _t;
                  AST tmp10_AST = null;
                  AST tmp10_AST_in = null;
                  tmp10_AST = astFactory.create((AST)_t);
                  tmp10_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp10_AST);
                  ASTPair __currentAST23 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,PARAMLIST);
                  _t = _t.getFirstChild();
                  {
                  _loop25:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if (((_t.getType() >= INT && _t.getType() <=
BOOLSTR))) {
                              pt = _t==ASTNULL ? null : (AST)_t;
```

```
                              type(_t);
                              _t = _retTree;
                              pt_AST = (AST)returnAST;
                              astFactory.addASTChild(currentAST,
returnAST);
                              pn = (AST)_t;
                              AST pn_AST_in = null;
                              pn_AST = astFactory.create(pn);
                              astFactory.addASTChild(currentAST,
pn_AST);
                              match(_t,ID);
                              _t = _t.getNextSibling();
                        }
                        else {
                              break _loop25;
                        }

                  } while (true);
                  }
                  currentAST = __currentAST23;
                  _t = __t23;
                  _t = _t.getNextSibling();
                  paramlist_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = paramlist_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  varDecl(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST varDecl_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST varDecl_AST = null;
            AST t_AST = null;
            AST t = null;
            AST id = null;
            AST id_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t17 = _t;
                  AST tmp11_AST = null;
                  AST tmp11_AST_in = null;
```

```
                  tmp11_AST = astFactory.create((AST)_t);
                  tmp11_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp11_AST);
                  ASTPair __currentAST17 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,VARDECL);
                  _t = _t.getFirstChild();
                  {
                  _loop21:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if (((_t.getType() >= INT && _t.getType() <=
BOOLSTR))) {
                              t = _t==ASTNULL ? null : (AST)_t;
                              type(_t);
                              _t = _retTree;
                              t_AST = (AST)returnAST;
                              astFactory.addASTChild(currentAST,
returnAST);
                              {
                              int _cnt20=0;
                              _loop20:
                              do {
                                    if (_t==null) _t=ASTNULL;
                                    if ((_t.getType()==ID)) {
                                          id = (AST)_t;
                                          AST id_AST_in = null;
                                          id_AST =
astFactory.create(id);

      astFactory.addASTChild(currentAST, id_AST);
                                          match(_t,ID);
                                          _t = _t.getNextSibling();
                                    }
                                    else {
                                          if ( _cnt20>=1 ) { break
_loop20; } else {throw new NoViableAltException(_t);}
                                    }

                                    _cnt20++;
                              } while (true);
                              }
                        }
                        else {
                              break _loop21;
                        }

                  } while (true);
                  }
                  currentAST = __currentAST17;
                  _t = __t17;
                  _t = _t.getNextSibling();
                  varDecl_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
```

```
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = varDecl_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  stmtblock(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST stmtblock_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST stmtblock_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t29 = _t;
                  AST tmp12_AST = null;
                  AST tmp12_AST_in = null;
                  tmp12_AST = astFactory.create((AST)_t);
                  tmp12_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp12_AST);
                  ASTPair __currentAST29 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,STMTBLOCK);
                  _t = _t.getFirstChild();
                  {
                  int _cnt31=0;
                  _loop31:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_tokenSet_0.member(_t.getType()))) {
                              a=stmt(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
                        else {
                              if ( _cnt31>=1 ) { break _loop31; } else
{throw new NoViableAltException(_t);}
                        }

                        _cnt31++;
                  } while (true);
                  }
                  currentAST = __currentAST29;
                  _t = __t29;
                  _t = _t.getNextSibling();
```

```
                stmtblock_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = stmtblock_AST;
            _retTree = _t;
            return r ;
        }

        public final  CommonAST  stmt(AST _t) throws RecognitionException
{
             CommonAST r ;

            AST stmt_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST stmt_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case ASSIGNSTMT:
                {
                      a=assignstmt(_t);
                      _t = _retTree;
                      astFactory.addASTChild(currentAST, returnAST);
                      stmt_AST = (AST)currentAST.root;
                      break;
                }
                case EXPR:
                {
                      a=expr(_t);
                      _t = _retTree;
                      astFactory.addASTChild(currentAST, returnAST);
                      stmt_AST = (AST)currentAST.root;
                      break;
                }
                case RETURNSTR:
                {
                      a=returnstmt(_t);
                      _t = _retTree;
                      astFactory.addASTChild(currentAST, returnAST);
                      stmt_AST = (AST)currentAST.root;
                      break;
                }
                case IFSTR:
                {
                      a=ifBlock(_t);
```

```
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  case WHILESTR:
                  {
                        a=whileBlock(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  case FORSTR:
                  case FORLINESTR:
                  {
                        a=forBlock(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  case COLORSTR:
                  case DRAWSTR:
                  case APPENDSTR:
                  {
                        a=builtinFunc(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        stmt_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = stmt_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  assignstmt(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST assignstmt_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST assignstmt_AST = null;

            CommonAST a,b,c;
```

```
		Vector v = new Vector();
		r = null;
		a = new CommonAST();



		try {	// for error handling
			AST __t34 = _t;
			AST tmp13_AST = null;
			AST tmp13_AST_in = null;
			tmp13_AST = astFactory.create((AST)_t);
			tmp13_AST_in = (AST)_t;
			astFactory.addASTChild(currentAST, tmp13_AST);
			ASTPair __currentAST34 = currentAST.copy();
			currentAST.root = currentAST.child;
			currentAST.child = null;
			match(_t,ASSIGNSTMT);
			_t = _t.getFirstChild();
			AST tmp14_AST = null;
			AST tmp14_AST_in = null;
			tmp14_AST = astFactory.create((AST)_t);
			tmp14_AST_in = (AST)_t;
			astFactory.addASTChild(currentAST, tmp14_AST);
			match(_t,LET);
			_t = _t.getNextSibling();
			b=varaccess(_t);
			_t = _retTree;
			astFactory.addASTChild(currentAST, returnAST);
			a=relExpr(_t);
			_t = _retTree;
			astFactory.addASTChild(currentAST, returnAST);
			currentAST = __currentAST34;
			_t = __t34;
			_t = _t.getNextSibling();
			assignstmt_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		returnAST = assignstmt_AST;
		_retTree = _t;
		return r ;
	}

	public final  CommonAST  expr(AST _t) throws RecognitionException
{
		 CommonAST r ;

		AST expr_AST_in = (AST)_t;
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST expr_AST = null;
		AST p_AST = null;
		AST p = null;
```

```
CommonAST a,b,c;
Vector v = new Vector();
r = null;
a = new CommonAST();



try {      // for error handling
      AST __t65 = _t;
      AST tmp15_AST = null;
      AST tmp15_AST_in = null;
      tmp15_AST = astFactory.create((AST)_t);
      tmp15_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp15_AST);
      ASTPair __currentAST65 = currentAST.copy();
      currentAST.root = currentAST.child;
      currentAST.child = null;
      match(_t,EXPR);
      _t = _t.getFirstChild();
      a=primaryExpr(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      {
      if (_t==null) _t=ASTNULL;
      switch ( _t.getType()) {
      case AND:
      case OR:
      {
            p = _t==ASTNULL ? null : (AST)_t;
            primaryOp(_t);
            _t = _retTree;
            p_AST = (AST)returnAST;
            astFactory.addASTChild(currentAST, returnAST);
            b=primaryExpr(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            break;
      }
      case 3:
      {
            break;
      }
      default:
      {
            throw new NoViableAltException(_t);
      }
      }
      }
      currentAST = __currentAST65;
      _t = __t65;
      _t = _t.getNextSibling();
      expr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
      reportError(ex);
      if (_t!=null) {_t = _t.getNextSibling();}
}
```

```
            returnAST = expr_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  returnstmt(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST returnstmt_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST returnstmt_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t52 = _t;
                  AST tmp16_AST = null;
                  AST tmp16_AST_in = null;
                  tmp16_AST = astFactory.create((AST)_t);
                  tmp16_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp16_AST);
                  ASTPair __currentAST52 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,RETURNSTR);
                  _t = _t.getFirstChild();
                  a=relExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  currentAST = __currentAST52;
                  _t = __t52;
                  _t = _t.getNextSibling();
                  returnstmt_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = returnstmt_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  ifBlock(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST ifBlock_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
```

```
AST ifBlock_AST = null;

CommonAST a,b,c;
Vector v = new Vector();
r = null;
a = new CommonAST();
boolean res = false;



try {      // for error handling
      AST __t40 = _t;
      AST tmp17_AST = null;
      AST tmp17_AST_in = null;
      tmp17_AST = astFactory.create((AST)_t);
      tmp17_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp17_AST);
      ASTPair __currentAST40 = currentAST.copy();
      currentAST.root = currentAST.child;
      currentAST.child = null;
      match(_t,IFSTR);
      _t = _t.getFirstChild();
      a=expr(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      {
      b=stmtblock(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      }
      {
      if (_t==null) _t=ASTNULL;
      switch ( _t.getType()) {
      case STMTBLOCK:
      {
            c=stmtblock(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            break;
      }
      case 3:
      {
            break;
      }
      default:
      {
            throw new NoViableAltException(_t);
      }
      }
      }
      currentAST = __currentAST40;
      _t = __t40;
      _t = _t.getNextSibling();
      ifBlock_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
      reportError(ex);
```

```
            if (_t!=null) {_t = _t.getNextSibling();}
        }
        returnAST = ifBlock_AST;
        _retTree = _t;
        return r ;
    }

    public final  CommonAST  whileBlock(AST _t) throws
RecognitionException {
         CommonAST r ;

        AST whileBlock_AST_in = (AST)_t;
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST whileBlock_AST = null;

        CommonAST a,b,c;
        Vector v = new Vector();
        r = null;
        a = new CommonAST();
        boolean res = false;



        try {      // for error handling
            AST __t44 = _t;
            AST tmp18_AST = null;
            AST tmp18_AST_in = null;
            tmp18_AST = astFactory.create((AST)_t);
            tmp18_AST_in = (AST)_t;
            astFactory.addASTChild(currentAST, tmp18_AST);
            ASTPair __currentAST44 = currentAST.copy();
            currentAST.root = currentAST.child;
            currentAST.child = null;
            match(_t,WHILESTR);
            _t = _t.getFirstChild();
            a=expr(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            {
            b=stmtblock(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            }
            currentAST = __currentAST44;
            _t = __t44;
            _t = _t.getNextSibling();
            whileBlock_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
        }
        returnAST = whileBlock_AST;
        _retTree = _t;
        return r ;
    }
```

```
      public final  CommonAST  forBlock(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST forBlock_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST forBlock_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case FORSTR:
                  {
                        AST __t47 = _t;
                        AST tmp19_AST = null;
                        AST tmp19_AST_in = null;
                        tmp19_AST = astFactory.create((AST)_t);
                        tmp19_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp19_AST);
                        ASTPair __currentAST47 = currentAST.copy();
                        currentAST.root = currentAST.child;
                        currentAST.child = null;
                        match(_t,FORSTR);
                        _t = _t.getFirstChild();
                        {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ASSIGNSTMT:
                        {
                              b=assignstmt(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        case EXPR:
                        {
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(_t);
                        }
                        }
                        }
                        a=expr(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
```

```
                        {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ASSIGNSTMT:
                        {
                              b=assignstmt(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        case STMTBLOCK:
                        {
                              break;
                        }
                        default:
                        {
                              throw new NoViableAltException(_t);
                        }
                        }
                        }
                        c=stmtblock(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        currentAST = __currentAST47;
                        _t = __t47;
                        _t = _t.getNextSibling();
                        forBlock_AST = (AST)currentAST.root;
                        break;
                  }
                  case FORLINESTR:
                  {
                        AST __t50 = _t;
                        AST tmp20_AST = null;
                        AST tmp20_AST_in = null;
                        tmp20_AST = astFactory.create((AST)_t);
                        tmp20_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp20_AST);
                        ASTPair __currentAST50 = currentAST.copy();
                        currentAST.root = currentAST.child;
                        currentAST.child = null;
                        match(_t,FORLINESTR);
                        _t = _t.getFirstChild();
                        AST tmp21_AST = null;
                        AST tmp21_AST_in = null;
                        tmp21_AST = astFactory.create((AST)_t);
                        tmp21_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp21_AST);
                        match(_t,ID);
                        _t = _t.getNextSibling();
                        AST tmp22_AST = null;
                        AST tmp22_AST_in = null;
                        tmp22_AST = astFactory.create((AST)_t);
                        tmp22_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp22_AST);
                        match(_t,ID);
                        _t = _t.getNextSibling();
```

```
                        a=stmtblock(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        currentAST = __currentAST50;
                        _t = __t50;
                        _t = _t.getNextSibling();
                        forBlock_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = forBlock_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  builtinFunc(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST builtinFunc_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST builtinFunc_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case COLORSTR:
                  {
                        a=colorFunc(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        builtinFunc_AST = (AST)currentAST.root;
                        break;
                  }
                  case DRAWSTR:
                  {
                        a=drawFunc(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        builtinFunc_AST = (AST)currentAST.root;
```

```
                        break;
                  }
                  case APPENDSTR:
                  {
                        a=appendFunc(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        builtinFunc_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = builtinFunc_AST;
            _retTree = _t;
            return r ;
      }

      public final CommonAST  varaccess(AST _t) throws
RecognitionException {
            CommonAST r;

            AST varaccess_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST varaccess_AST = null;
            AST id = null;
            AST id_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t36 = _t;
                  id = _t==ASTNULL ? null :(AST)_t;
                  AST id_AST_in = null;
                  id_AST = astFactory.create(id);
                  astFactory.addASTChild(currentAST, id_AST);
                  ASTPair __currentAST36 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,ID);
                  _t = _t.getFirstChild();
                  {
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
```

```
                    case XAXIS:
                    {
                          AST tmp23_AST = null;
                          AST tmp23_AST_in = null;
                          tmp23_AST = astFactory.create((AST)_t);
                          tmp23_AST_in = (AST)_t;
                          astFactory.addASTChild(currentAST, tmp23_AST);
                          match(_t,XAXIS);
                          _t = _t.getNextSibling();
                          break;
                    }
                    case YAXIS:
                    {
                          AST tmp24_AST = null;
                          AST tmp24_AST_in = null;
                          tmp24_AST = astFactory.create((AST)_t);
                          tmp24_AST_in = (AST)_t;
                          astFactory.addASTChild(currentAST, tmp24_AST);
                          match(_t,YAXIS);
                          _t = _t.getNextSibling();
                          break;
                    }
                    case INTEGER:
                    {
                          AST tmp25_AST = null;
                          AST tmp25_AST_in = null;
                          tmp25_AST = astFactory.create((AST)_t);
                          tmp25_AST_in = (AST)_t;
                          astFactory.addASTChild(currentAST, tmp25_AST);
                          match(_t,INTEGER);
                          _t = _t.getNextSibling();
                          break;
                    }
                    case 3:
                    case EXPR:
                    {
                          {
                          if (_t==null) _t=ASTNULL;
                          switch ( _t.getType()) {
                          case EXPR:
                          {
                                a=expr(_t);
                                _t = _retTree;
                                astFactory.addASTChild(currentAST,
returnAST);
                                break;
                          }
                          case 3:
                          {
                                break;
                          }
                          default:
                          {
                                throw new NoViableAltException(_t);
                          }
                          }
                          }
```

```
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
                  }
                  currentAST = __currentAST36;
                  _t = __t36;
                  _t = _t.getNextSibling();
                  varaccess_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = varaccess_AST;
            _retTree = _t;
            return r;
      }

      public final  CommonAST  relExpr(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST relExpr_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST relExpr_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  a=term(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  _loop81:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==PLUS||_t.getType()==MINUS))
{
                              addOp(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                              b=term(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                        }
```

```
                        else {
                              break _loop81;
                        }

                  } while (true);
                  }
                  relExpr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = relExpr_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  colorFunc(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST colorFunc_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST colorFunc_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t55 = _t;
                  AST tmp26_AST = null;
                  AST tmp26_AST_in = null;
                  tmp26_AST = astFactory.create((AST)_t);
                  tmp26_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp26_AST);
                  ASTPair __currentAST55 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,COLORSTR);
                  _t = _t.getFirstChild();
                  a=relExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  b=relExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  c=relExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  currentAST = __currentAST55;
                  _t = __t55;
                  _t = _t.getNextSibling();
```

```
                colorFunc_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = colorFunc_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  drawFunc(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST drawFunc_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST drawFunc_AST = null;
            AST id = null;
            AST id_AST = null;
            AST p_AST = null;
            AST p = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t57 = _t;
                  AST tmp27_AST = null;
                  AST tmp27_AST_in = null;
                  tmp27_AST = astFactory.create((AST)_t);
                  tmp27_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp27_AST);
                  ASTPair __currentAST57 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,DRAWSTR);
                  _t = _t.getFirstChild();
                  {
                  int _cnt59=0;
                  _loop59:
                  do {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ID:
                        {
                              id = (AST)_t;
                              AST id_AST_in = null;
                              id_AST = astFactory.create(id);
                              astFactory.addASTChild(currentAST,
id_AST);
                              match(_t,ID);
```

```
                              _t = _t.getNextSibling();
                              break;
                        }
                        case POINT:
                        {
                              p = _t==ASTNULL ? null : (AST)_t;
                              point(_t);
                              _t = _retTree;
                              p_AST = (AST)returnAST;
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        default:
                        {
                              if ( _cnt59>=1 ) { break _loop59; } else
{throw new NoViableAltException(_t);}
                        }
                        }
                        _cnt59++;
                  } while (true);
                  }
                  currentAST = __currentAST57;
                  _t = __t57;
                  _t = _t.getNextSibling();
                  drawFunc_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = drawFunc_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  appendFunc(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST appendFunc_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST appendFunc_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t61 = _t;
                  AST tmp28_AST = null;
                  AST tmp28_AST_in = null;
                  tmp28_AST = astFactory.create((AST)_t);
```

```
tmp28_AST_in = (AST)_t;
astFactory.addASTChild(currentAST, tmp28_AST);
ASTPair __currentAST61 = currentAST.copy();
currentAST.root = currentAST.child;
currentAST.child = null;
match(_t,APPENDSTR);
_t = _t.getFirstChild();
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ID:
{
      AST tmp29_AST = null;
      AST tmp29_AST_in = null;
      tmp29_AST = astFactory.create((AST)_t);
      tmp29_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp29_AST);
      match(_t,ID);
      _t = _t.getNextSibling();
      break;
}
case POINT:
{
      point(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      break;
}
case LINE:
{
      line(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      break;
}
default:
{
      throw new NoViableAltException(_t);
}
}
}
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ID:
{
      AST tmp30_AST = null;
      AST tmp30_AST_in = null;
      tmp30_AST = astFactory.create((AST)_t);
      tmp30_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp30_AST);
      match(_t,ID);
      _t = _t.getNextSibling();
      break;
}
case POINT:
{
```

```
                        point(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                }
                case LINE:
                {
                        line(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
                }
                currentAST = __currentAST61;
                _t = __t61;
                _t = _t.getNextSibling();
                appendFunc_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        returnAST = appendFunc_AST;
        _retTree = _t;
        return r ;
}

public final void point(AST _t) throws RecognitionException {

        AST point_AST_in = (AST)_t;
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST point_AST = null;

        String x = "";
        String y = "";


        try {      // for error handling
                AST __t105 = _t;
                AST tmp31_AST = null;
                AST tmp31_AST_in = null;
                tmp31_AST = astFactory.create((AST)_t);
                tmp31_AST_in = (AST)_t;
                astFactory.addASTChild(currentAST, tmp31_AST);
                ASTPair __currentAST105 = currentAST.copy();
                currentAST.root = currentAST.child;
                currentAST.child = null;
                match(_t,POINT);
                _t = _t.getFirstChild();
                {
                if (_t==null) _t=ASTNULL;
```

```
switch ( _t.getType()) {
case INTEGER:
{
      AST tmp32_AST = null;
      AST tmp32_AST_in = null;
      tmp32_AST = astFactory.create((AST)_t);
      tmp32_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp32_AST);
      match(_t,INTEGER);
      _t = _t.getNextSibling();
      break;
}
case ID:
{
      AST tmp33_AST = null;
      AST tmp33_AST_in = null;
      tmp33_AST = astFactory.create((AST)_t);
      tmp33_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp33_AST);
      match(_t,ID);
      _t = _t.getNextSibling();
      break;
}
default:
{
      throw new NoViableAltException(_t);
}
}
}
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case INTEGER:
{
      AST tmp34_AST = null;
      AST tmp34_AST_in = null;
      tmp34_AST = astFactory.create((AST)_t);
      tmp34_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp34_AST);
      match(_t,INTEGER);
      _t = _t.getNextSibling();
      break;
}
case ID:
{
      AST tmp35_AST = null;
      AST tmp35_AST_in = null;
      tmp35_AST = astFactory.create((AST)_t);
      tmp35_AST_in = (AST)_t;
      astFactory.addASTChild(currentAST, tmp35_AST);
      match(_t,ID);
      _t = _t.getNextSibling();
      break;
}
default:
{
      throw new NoViableAltException(_t);
```

```
            }
            }
            }
            currentAST = __currentAST105;
            _t = __t105;
            _t = _t.getNextSibling();
            point_AST = (AST)currentAST.root;
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      returnAST = point_AST;
      _retTree = _t;
}

public final void line(AST _t) throws RecognitionException {

      AST line_AST_in = (AST)_t;
      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST line_AST = null;
      AST mp_AST = null;
      AST mp = null;

      try {      // for error handling
            AST __t109 = _t;
            AST tmp36_AST = null;
            AST tmp36_AST_in = null;
            tmp36_AST = astFactory.create((AST)_t);
            tmp36_AST_in = (AST)_t;
            astFactory.addASTChild(currentAST, tmp36_AST);
            ASTPair __currentAST109 = currentAST.copy();
            currentAST.root = currentAST.child;
            currentAST.child = null;
            match(_t,LINE);
            _t = _t.getFirstChild();
            {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case ID:
            {
                  AST tmp37_AST = null;
                  AST tmp37_AST_in = null;
                  tmp37_AST = astFactory.create((AST)_t);
                  tmp37_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp37_AST);
                  match(_t,ID);
                  _t = _t.getNextSibling();
                  break;
            }
            case POINT:
            {
                  point(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  break;
```

```
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
                  }
                  {
                  int _cnt112=0;
                  _loop112:
                  do {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ID:
                        {
                              AST tmp38_AST = null;
                              AST tmp38_AST_in = null;
                              tmp38_AST = astFactory.create((AST)_t);
                              tmp38_AST_in = (AST)_t;
                              astFactory.addASTChild(currentAST,
tmp38_AST);
                              match(_t,ID);
                              _t = _t.getNextSibling();
                              break;
                        }
                        case POINT:
                        {
                              mp = _t==ASTNULL ? null : (AST)_t;
                              point(_t);
                              _t = _retTree;
                              mp_AST = (AST)returnAST;
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        default:
                        {
                              if ( _cnt112>=1 ) { break _loop112; }
else {throw new NoViableAltException(_t);}
                        }
                        }
                        _cnt112++;
                  } while (true);
                  }
                  currentAST = __currentAST109;
                  _t = __t109;
                  _t = _t.getNextSibling();
                  line_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = line_AST;
            _retTree = _t;
      }
```

```
      public final  CommonAST  primaryExpr(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST primaryExpr_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST primaryExpr_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  a=eqeqExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case TRUE:
                  case FALSE:
                  case CALL:
                  case NOT:
                  case ID:
                  case INTEGER:
                  case POINT:
                  case LINE:
                  {
                        b=eqeqOp(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        c=eqeqExpr(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case 3:
                  case AND:
                  case OR:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
                  }
                  primaryExpr_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
```

```
		}
		returnAST = primaryExpr_AST;
		_retTree = _t;
		return r ;
	}

	public final void primaryOp(AST _t) throws RecognitionException {

		AST primaryOp_AST_in = (AST)_t;
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST primaryOp_AST = null;

		try {	// for error handling
			if (_t==null) _t=ASTNULL;
			switch ( _t.getType()) {
			case AND:
			{
				AST tmp39_AST = null;
				AST tmp39_AST_in = null;
				tmp39_AST = astFactory.create((AST)_t);
				tmp39_AST_in = (AST)_t;
				astFactory.addASTChild(currentAST, tmp39_AST);
				match(_t,AND);
				_t = _t.getNextSibling();
				primaryOp_AST = (AST)currentAST.root;
				break;
			}
			case OR:
			{
				AST tmp40_AST = null;
				AST tmp40_AST_in = null;
				tmp40_AST = astFactory.create((AST)_t);
				tmp40_AST_in = (AST)_t;
				astFactory.addASTChild(currentAST, tmp40_AST);
				match(_t,OR);
				_t = _t.getNextSibling();
				primaryOp_AST = (AST)currentAST.root;
				break;
			}
			default:
			{
				throw new NoViableAltException(_t);
			}
			}
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		returnAST = primaryOp_AST;
		_retTree = _t;
	}

	public final  CommonAST  eqeqExpr(AST _t) throws
RecognitionException {
		 CommonAST r ;
```

```
AST eqeqExpr_AST_in = (AST)_t;
returnAST = null;
ASTPair currentAST = new ASTPair();
AST eqeqExpr_AST = null;

CommonAST a,b,c;
Vector v = new Vector();
r = null;
a = new CommonAST();



try {      // for error handling
      b=relExpr(_t);
      _t = _retTree;
      astFactory.addASTChild(currentAST, returnAST);
      {
      if (_t==null) _t=ASTNULL;
      switch ( _t.getType()) {
      case GREATER:
      case LESS:
      case GEQ:
      case LEQ:
      {
            relOp(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            a=relExpr(_t);
            _t = _retTree;
            astFactory.addASTChild(currentAST, returnAST);
            break;
      }
      case 3:
      case TRUE:
      case FALSE:
      case CALL:
      case AND:
      case OR:
      case NOT:
      case ID:
      case INTEGER:
      case POINT:
      case LINE:
      {
            break;
      }
      default:
      {
            throw new NoViableAltException(_t);
      }
      }
      }
      eqeqExpr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
      reportError(ex);
```

```
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = eqeqExpr_AST;
            _retTree = _t;
            return r ;
      }

      public final  CommonAST  eqeqOp(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST eqeqOp_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST eqeqOp_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  a=relExpr(_t);
                  _t = _retTree;
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case GREATER:
                  case LESS:
                  case GEQ:
                  case LEQ:
                  {
                        relOp(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        b=relExpr(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case TRUE:
                  case FALSE:
                  case CALL:
                  case NOT:
                  case ID:
                  case INTEGER:
                  case POINT:
                  case LINE:
                  {
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
```

```
            }
            }
            }
            eqeqOp_AST = (AST)currentAST.root;
      }
      catch (RecognitionException ex) {
            reportError(ex);
            if (_t!=null) {_t = _t.getNextSibling();}
      }
      returnAST = eqeqOp_AST;
      _retTree = _t;
      return r ;
}

public final void relOp(AST _t) throws RecognitionException {

      AST relOp_AST_in = (AST)_t;
      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST relOp_AST = null;

      String ro = "";


      try {      // for error handling
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case LESS:
            {
                  {
                  AST tmp41_AST = null;
                  AST tmp41_AST_in = null;
                  tmp41_AST = astFactory.create((AST)_t);
                  tmp41_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp41_AST);
                  match(_t,LESS);
                  _t = _t.getNextSibling();
                  }
                  relOp_AST = (AST)currentAST.root;
                  break;
            }
            case LEQ:
            {
                  {
                  AST tmp42_AST = null;
                  AST tmp42_AST_in = null;
                  tmp42_AST = astFactory.create((AST)_t);
                  tmp42_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp42_AST);
                  match(_t,LEQ);
                  _t = _t.getNextSibling();
                  }
                  relOp_AST = (AST)currentAST.root;
                  break;
            }
            case GREATER:
            {
```

```
                        {
                        AST tmp43_AST = null;
                        AST tmp43_AST_in = null;
                        tmp43_AST = astFactory.create((AST)_t);
                        tmp43_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp43_AST);
                        match(_t,GREATER);
                        _t = _t.getNextSibling();
                        }
                        relOp_AST = (AST)currentAST.root;
                        break;
                  }
                  case GEQ:
                  {
                        {
                        AST tmp44_AST = null;
                        AST tmp44_AST_in = null;
                        tmp44_AST = astFactory.create((AST)_t);
                        tmp44_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp44_AST);
                        match(_t,GEQ);
                        _t = _t.getNextSibling();
                        }
                        relOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = relOp_AST;
            _retTree = _t;
      }

      public final  CommonAST  term(AST _t) throws RecognitionException
{
             CommonAST r ;

            AST term_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST term_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
```

```
                a=factor(_t);
                _t = _retTree;
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop87:
                do {
                      if (_t==null) _t=ASTNULL;
                      if (((_t.getType() >= MUL && _t.getType() <=
MOD))) {
                            mulOp(_t);
                            _t = _retTree;
                            astFactory.addASTChild(currentAST,
returnAST);
                            b=factor(_t);
                            _t = _retTree;
                            astFactory.addASTChild(currentAST,
returnAST);
                      }
                      else {
                            break _loop87;
                      }

                } while (true);
                }
                term_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
          }
          returnAST = term_AST;
          _retTree = _t;
          return r ;
     }

     public final void addOp(AST _t) throws RecognitionException {

          AST addOp_AST_in = (AST)_t;
          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST addOp_AST = null;

          try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case PLUS:
                {
                      {
                      AST tmp45_AST = null;
                      AST tmp45_AST_in = null;
                      tmp45_AST = astFactory.create((AST)_t);
                      tmp45_AST_in = (AST)_t;
                      astFactory.addASTChild(currentAST, tmp45_AST);
                      match(_t,PLUS);
                      _t = _t.getNextSibling();
                      }
                      addOp_AST = (AST)currentAST.root;
```

```
                        break;
                  }
                  case MINUS:
                  {
                        {
                        AST tmp46_AST = null;
                        AST tmp46_AST_in = null;
                        tmp46_AST = astFactory.create((AST)_t);
                        tmp46_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp46_AST);
                        match(_t,MINUS);
                        _t = _t.getNextSibling();
                        }
                        addOp_AST = (AST)currentAST.root;
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = addOp_AST;
            _retTree = _t;
      }

      public final  CommonAST  factor(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST factor_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST factor_AST = null;
            AST id = null;
            AST id_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();
            boolean cmp = false;


            try {      // for error handling
                  {
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case TRUE:
                  case FALSE:
                  case ID:
                  case INTEGER:
                  case POINT:
```

```
                    case LINE:
                    {
                          {
                          {
                          if (_t==null) _t=ASTNULL;
                          switch ( _t.getType()) {
                          case TRUE:
                          case FALSE:
                          case INTEGER:
                          case POINT:
                          case LINE:
                          {
                                constant(_t);
                                _t = _retTree;
                                astFactory.addASTChild(currentAST,
returnAST);
                                break;
                          }
                          case ID:
                          {
                                id = (AST)_t;
                                AST id_AST_in = null;
                                id_AST = astFactory.create(id);
                                astFactory.addASTChild(currentAST,
id_AST);
                                match(_t,ID);
                                _t = _t.getNextSibling();
                                break;
                          }
                          default:
                          {
                                throw new NoViableAltException(_t);
                          }
                          }
                          }
                          {
                          _loop99:
                          do {
                                if (_t==null) _t=ASTNULL;
                                if
((_t.getType()==PLUS||_t.getType()==MINUS)) {
                                      {
                                      addOp(_t);
                                      _t = _retTree;
                                      astFactory.addASTChild(currentAST,
returnAST);
                                      {
                                      if (_t==null) _t=ASTNULL;
                                      switch ( _t.getType()) {
                                      case TRUE:
                                      case FALSE:
                                      case INTEGER:
                                      case POINT:
                                      case LINE:
                                      {
                                            constant(_t);
                                            _t = _retTree;
```

```
      astFactory.addASTChild(currentAST, returnAST);
                                          break;
                                    }
                                    case ID:
                                    {
                                          AST tmp47_AST = null;
                                          AST tmp47_AST_in = null;
                                          tmp47_AST =
astFactory.create((AST)_t);
                                          tmp47_AST_in = (AST)_t;

      astFactory.addASTChild(currentAST, tmp47_AST);
                                          match(_t,ID);
                                          _t = _t.getNextSibling();
                                          break;
                                    }
                                    default:
                                    {
                                          throw new
NoViableAltException(_t);
                                    }
                                    }
                                    }
                                    }
                              }
                              else {
                                    break _loop99;
                              }

                        } while (true);
                        }
                        }
                        break;
                  }
                  case CALL:
                  {
                        b=funccall(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case NOT:
                  {
                        AST tmp48_AST = null;
                        AST tmp48_AST_in = null;
                        tmp48_AST = astFactory.create((AST)_t);
                        tmp48_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp48_AST);
                        match(_t,NOT);
                        _t = _t.getNextSibling();
                        b=factor(_t);
                        _t = _retTree;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  default:
```

```
                {
                        throw new NoViableAltException(_t);
                }
                }
                }
                factor_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        returnAST = factor_AST;
        _retTree = _t;
        return r ;
}

public final void mulOp(AST _t) throws RecognitionException {

        AST mulOp_AST_in = (AST)_t;
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST mulOp_AST = null;

        try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case MUL:
                {
                        {
                        AST tmp49_AST = null;
                        AST tmp49_AST_in = null;
                        tmp49_AST = astFactory.create((AST)_t);
                        tmp49_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp49_AST);
                        match(_t,MUL);
                        _t = _t.getNextSibling();
                        }
                        mulOp_AST = (AST)currentAST.root;
                        break;
                }
                case DIV:
                {
                        {
                        AST tmp50_AST = null;
                        AST tmp50_AST_in = null;
                        tmp50_AST = astFactory.create((AST)_t);
                        tmp50_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp50_AST);
                        match(_t,DIV);
                        _t = _t.getNextSibling();
                        }
                        mulOp_AST = (AST)currentAST.root;
                        break;
                }
                case MOD:
                {
                        {
```

```
                    AST tmp51_AST = null;
                    AST tmp51_AST_in = null;
                    tmp51_AST = astFactory.create((AST)_t);
                    tmp51_AST_in = (AST)_t;
                    astFactory.addASTChild(currentAST, tmp51_AST);
                    match(_t,MOD);
                    _t = _t.getNextSibling();
                    }
                    mulOp_AST = (AST)currentAST.root;
                    break;
              }
              default:
              {
                    throw new NoViableAltException(_t);
              }
              }
        }
        catch (RecognitionException ex) {
              reportError(ex);
              if (_t!=null) {_t = _t.getNextSibling();}
        }
        returnAST = mulOp_AST;
        _retTree = _t;
}

public final void constant(AST _t) throws RecognitionException {

        AST constant_AST_in = (AST)_t;
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST constant_AST = null;
        AST b_AST = null;
        AST b = null;
        AST p_AST = null;
        AST p = null;
        AST l_AST = null;
        AST l = null;

        try {      // for error handling
              {
              if (_t==null) _t=ASTNULL;
              switch ( _t.getType()) {
              case INTEGER:
              {
                    AST tmp52_AST = null;
                    AST tmp52_AST_in = null;
                    tmp52_AST = astFactory.create((AST)_t);
                    tmp52_AST_in = (AST)_t;
                    astFactory.addASTChild(currentAST, tmp52_AST);
                    match(_t,INTEGER);
                    _t = _t.getNextSibling();
                    break;
              }
              case TRUE:
              case FALSE:
              {
                    b = _t==ASTNULL ? null : (AST)_t;
```

```
                        bool(_t);
                        _t = _retTree;
                        b_AST = (AST)returnAST;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case POINT:
                  {
                        p = _t==ASTNULL ? null : (AST)_t;
                        point(_t);
                        _t = _retTree;
                        p_AST = (AST)returnAST;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case LINE:
                  {
                        l = _t==ASTNULL ? null : (AST)_t;
                        line(_t);
                        _t = _retTree;
                        l_AST = (AST)returnAST;
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
                  }
                  constant_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = constant_AST;
            _retTree = _t;
      }

      public final void bool(AST _t) throws RecognitionException {

            AST bool_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST bool_AST = null;

            try {      // for error handling
                  {
                  if (_t==null) _t=ASTNULL;
                  switch ( _t.getType()) {
                  case TRUE:
                  {
                        AST tmp53_AST = null;
                        AST tmp53_AST_in = null;
                        tmp53_AST = astFactory.create((AST)_t);
                        tmp53_AST_in = (AST)_t;
```

```
                        astFactory.addASTChild(currentAST, tmp53_AST);
                        match(_t,TRUE);
                        _t = _t.getNextSibling();
                        break;
                  }
                  case FALSE:
                  {
                        AST tmp54_AST = null;
                        AST tmp54_AST_in = null;
                        tmp54_AST = astFactory.create((AST)_t);
                        tmp54_AST_in = (AST)_t;
                        astFactory.addASTChild(currentAST, tmp54_AST);
                        match(_t,FALSE);
                        _t = _t.getNextSibling();
                        break;
                  }
                  default:
                  {
                        throw new NoViableAltException(_t);
                  }
                  }
                  }
                  bool_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = bool_AST;
            _retTree = _t;
      }

      public final  CommonAST  arglist(AST _t) throws
RecognitionException {
             CommonAST r ;

            AST arglist_AST_in = (AST)_t;
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST arglist_AST = null;
            AST arg = null;
            AST arg_AST = null;

            CommonAST a,b,c;
            Vector v = new Vector();
            r = null;
            a = new CommonAST();



            try {      // for error handling
                  AST __t116 = _t;
                  AST tmp55_AST = null;
                  AST tmp55_AST_in = null;
                  tmp55_AST = astFactory.create((AST)_t);
                  tmp55_AST_in = (AST)_t;
                  astFactory.addASTChild(currentAST, tmp55_AST);
```

```
                  ASTPair __currentAST116 = currentAST.copy();
                  currentAST.root = currentAST.child;
                  currentAST.child = null;
                  match(_t,ARGLIST);
                  _t = _t.getFirstChild();
                  {
                  int _cnt118=0;
                  _loop118:
                  do {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType()) {
                        case ID:
                        {
                              arg = (AST)_t;
                              AST arg_AST_in = null;
                              arg_AST = astFactory.create(arg);
                              astFactory.addASTChild(currentAST,
arg_AST);
                              match(_t,ID);
                              _t = _t.getNextSibling();
                              break;
                        }
                        case TRUE:
                        case FALSE:
                        case INTEGER:
                        case POINT:
                        case LINE:
                        {
                              constant(_t);
                              _t = _retTree;
                              astFactory.addASTChild(currentAST,
returnAST);
                              break;
                        }
                        default:
                        {
                              if ( _cnt118>=1 ) { break _loop118; }
else {throw new NoViableAltException(_t);}
                        }
                        }
                        _cnt118++;
                  } while (true);
                  }
                  currentAST = __currentAST116;
                  _t = __t116;
                  _t = _t.getNextSibling();
                  arglist_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            returnAST = arglist_AST;
            _retTree = _t;
            return r ;
      }
```

```
public static final String[] _tokenNames = {
      "<0>",
      "EOF",
      "<2>",
      "NULL_TREE_LOOKAHEAD",
      "\"Int\"",
      "\"Line\"",
      "\"Point\"",
      "\"Boolean\"",
      "\"If\"",
      "\"Then\"",
      "\"Else\"",
      "\"IfEnd\"",
      "\"While\"",
      "\"Do\"",
      "\"WhileEnd\"",
      "\"For\"",
      "\"ForEnd\"",
      "\"ForLine\"",
      "\"In\"",
      "\"XAxis\"",
      "\"YAxis\"",
      "\"Return\"",
      "\"Color\"",
      "\"Draw\"",
      "\"Append\"",
      "\"True\"",
      "\"False\"",
      "\"Let\"",
      "\"To\"",
      "\"Call\"",
      "\"Global\"",
      "\"End\"",
      "PLUS",
      "MINUS",
      "MUL",
      "DIV",
      "MOD",
      "ASSIGN",
      "COMMA",
      "PERIOD",
      "GREATER",
      "LESS",
      "CARRAT",
      "GEQ",
      "LEQ",
      "AND",
      "OR",
      "EQEQ",
      "NEQ",
      "LPARENS",
      "RPARENS",
      "SEMI",
      "LBRACE",
      "RBRACE",
      "BRACKET",
```

```
            "LBRACKET",
            "RBRACKET",
            "NOT",
            "LETTER",
            "DIGIT",
            "ID",
            "INTEGER",
            "EXPONENT",
            "WS",
            "COMMENT",
            "PROGRAM",
            "GLOBVARDECL",
            "FUNCDECL",
            "VARDECL",
            "PARAMLIST",
            "STMTBLOCK",
            "ASSIGNSTMT",
            "EXPR",
            "POINT",
            "LINE",
            "ARGLIST"
      };

      private static final long[] mk_tokenSet_0() {
            long[] data = { 31625472L, 384L, 0L, 0L};
            return data;
      }
      public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
      }
```

BatsWalkerTokenTypes.java:

```
// $ANTLR 2.7.2: "BatsWalker.g" -> "BatsWalker.java"$

public interface BatsWalkerTokenTypes {
      int EOF = 1;
      int NULL_TREE_LOOKAHEAD = 3;
      int INT = 4;
      int LINESTR = 5;
      int POINTSTR = 6;
      int BOOLSTR = 7;
      int IFSTR = 8;
      int THEN = 9;
      int ELSE = 10;
      int IFEND = 11;
      int WHILESTR = 12;
      int DO = 13;
      int WHILEEND = 14;
      int FORSTR = 15;
      int FOREND = 16;
      int FORLINESTR = 17;
      int IN = 18;
      int XAXIS = 19;
```

```
        int YAXIS = 20;
        int RETURNSTR = 21;
        int COLORSTR = 22;
        int DRAWSTR = 23;
        int APPENDSTR = 24;
        int TRUE = 25;
        int FALSE = 26;
        int LET = 27;
        int TO = 28;
        int CALL = 29;
        int GLOBAL = 30;
        int END = 31;
        int PLUS = 32;
        int MINUS = 33;
        int MUL = 34;
        int DIV = 35;
        int MOD = 36;
        int ASSIGN = 37;
        int COMMA = 38;
        int PERIOD = 39;
        int GREATER = 40;
        int LESS = 41;
        int CARRAT = 42;
        int GEQ = 43;
        int LEQ = 44;
        int AND = 45;
        int OR = 46;
        int EQEQ = 47;
        int NEQ = 48;
        int LPARENS = 49;
        int RPARENS = 50;
        int SEMI = 51;
        int LBRACE = 52;
        int RBRACE = 53;
        int BRACKET = 54;
        int LBRACKET = 55;
        int RBRACKET = 56;
        int NOT = 57;
        int LETTER = 58;
        int DIGIT = 59;
        int ID = 60;
        int INTEGER = 61;
        int EXPONENT = 62;
        int WS = 63;
        int COMMENT = 64;
        int PROGRAM = 65;
        int GLOBVARDECL = 66;
        int FUNCDECL = 67;
        int VARDECL = 68;
        int PARAMLIST = 69;
        int STMTBLOCK = 70;
        int ASSIGNSTMT = 71;
        int EXPR = 72;
        int POINT = 73;
        int LINE = 74;
        int ARGLIST = 75;
}
```

Main.java:

```
import antlr.*;
import antlr.Token;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.ASTFactory;
import antlr.debug.misc.ASTFrame;
/**
*  The class created is a driver for a Bats program to be parsed by
BatsParser with AST
* generation walked the tree and refined it and sent to Bats'
interpreter to translate it
* to BatsProgram.java class with the following compiling and running. A
data "input" file
* is created for drawing by Frame.
*/

public class Main{
 public static void main(String[] args) {
    try {
      BufferedReader inFile=null;
      BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
      String fileName = "batssource.txt";
      inFile = new BufferedReader(new FileReader(fileName));
      BATSLexer l = new BATSLexer(inFile);
      BATSParser p = new BATSParser(l);

      p.program();
      String s = p.getAST().toStringList();
      CommonAST t = (CommonAST)p.getAST();

      ASTFactory factory = new ASTFactory();

      ASTFrame frame = new ASTFrame("AST JTree Example", t);
      frame.setVisible(true);
      BatsWalker w = new BatsWalker();
      System.out.println("TO WALKER "+p.getAST().toStringTree());

      w.program(p.getAST());
      t = (CommonAST)w.getAST();

      System.out.println("WALKER: "+t.toStringTree());

      BatsTranslator bTrans = new BatsTranslator(t);

      bTrans.interpret();

      System.out.println("java line :: " + bTrans.str.toString());


      /*      T walker = new T();
```

```
      walker.startRule(parser.getAST()); // walk tree
      AST results = walker.getAST();
      DumpASTVisitor visitor = new DumpASTVisitor();
      visitor.visit(results);*/

    } catch (Exception e) {
      System.err.println(e);
    }
  }
}
```

RunBats.bat

java antlr.Tool BATS.g
java antlr.Tool BatsWalker.g
javac *.java
java Main
javac BatsProgram.java
java BatsProgram