SASSi

A Language For Statistics

Programming Languages & Translators
Fall 2003

Carl Morgan – Group Leader
Paul Salama
Xiaotang Zhang

Table of Contents

1. INTRODUCTION

1.1. Purpose

SASSi's purpose is to create an efficient and easy to use scheme for handling statistics. The language is designed to let the user make quick and simple programs to deal with various statistical problems.

The hope is that SASSi will be seen as an alternative to the disaster that is SAS. Despite its longer name, SASSi would be smaller than the unnecessarily bloated SAS. It would also be free, allowing those who can't stomach SAS's $120 price tag, namely everybody. Compute variance and standard deviations graphically. Learn more from a generated pie chart with a small glance at a computer screen. Think about never having to figure out what libraries are needed and how to use them. SASSi makes it all possible.

1.2. Overview

SASSi: A simple, powerful, efficient, easily made graphical, customizable, and architecture neutral language.

Borrowing the form set forth by the Java white paper, we are using similar buzzwords to describe the features of our language. This only seems natural since our language will compile into Java code. SASSi has features that are helpful to any organization or student using statistics and can easily be learned and applied. The following explanation of the previous buzzwords will further help you to understand the characteristics of SASSi:

Simple When designing a system for statistics, there are two focuses: the set up of the model and the representation of that model. Because SASSi inherently has its own data types and constructs, it is simple to set up a statistics system. Also, it is easy for any user to make additions to the model or change behavior of an entire program. The language is simple and easy to learn because of its similarity to English. This parallel also helps in understanding exactly what each command does and simplifies how to use the language.

Powerful	SASSi gives you the power to make statistical models a reality in just a few lines of code.
Easily Made Graphical	Since SASSi is compiled into Java code, creating a GUI is simple to do. The user need not worry about creating any code for graphical representation. The compiler knows how to represent everything and where to place what where. All the user needs to do is choose what models he or she wishes to see.
Efficient	The main goal of this language is to take the strain off of the user and the computer. By keeping the libraries small and the manual compact, anyone can use SASSi. Using smaller libraries with SASSI allows the user to utilize more memory for the computations and other various tasks on his/her computer. The algorithms will also be efficient so as to maximize the utility of the computer.
Customizable	Different aspects of SASSi's implementation are customizable. The users will be able to design their own procedures and are encouraged to so. If they aren't satisfied with the procedures given to them, they can easily implement new procedures and not experience the headache of trying to find software that supports it.
Architecture Neutral	SASSi code is compiled into Java byte code which makes it architecture neutral. This means that SASSi can be developed on any platform that has the Java compiler, the JVM, and the GUI. This will make distributing the software easier.

1.3. Functionality

Graphics	SASSi offers several nice graphical options. The users can decide how the results are seen. The user can choose from a variety of formats including but not limited to: bar chart, pie chart, and curves. The users

also have an easy way of seeing the results in a text form. This simplicity allows every onlooker to quickly understand what is going on with the statistics system and if their model is behaving the way that is desired.

Data Types SASSi will support two types of data: numbers and vectors. The numbers will follow the basic patterns that are defined in most mathematical applications, i.e. integer and (double) floating point number. The vectors will be used to represent M by 1 matrices of data. The basic algebraic functions (addition, subtraction, multiplication, and division) will all be supported for both number and vector arguments.

Algorithms It will also support a number of various algorithms. Some of these are built into the language, but more complex ones can be imported using libraries pertaining to the functions you will be using. This will save space and make the program run more efficiently. For example: one library will allow you to do variance and standard deviation functions. Regressions will have their own library: including linear, multiple, and nonlinear regressions. Various libraries will be incorporated as well for variance, expectation, and a few other functions needed for statistics.

2. TUTORIAL

2.1. Getting Started

The first thing we need to do is open our text editor, and save a file with the suffix "ssi". Let's call ours test.ssi.

Although it isn't required, it's a good idea to begin with a comment describing the file, so here it is:

```
//Test SASSi program to be used for our final
//document
```

This is also where you would place any include statements to gain access to various statistics libraries if you wanted to do so. We do not need any libraries so we require no include statement.

2.2. Alternative: Command-line SASSi

SASSi also supports a command-line interface, and anything that is done in a SASSi file can be done at the SASSi prompt:

```
SASSi>
```

2.3. Variable Declarations

Here are some variables, and their original assignments:

```
a=1;
b=2;
c=6;

d=0.0;
e=0;
f=0;
v=[a,b,b,c,b,c];
```

SASSi will interpret d as a double, and e and f as integers.

The vector v is of length 6, and is assigned the values [1,2,2,6,2,6] from the constants above.

2.4. *Vector Procedure*

Here we use probably the most basic statistics function, the mean. The procedure takes our vector v, and assigns the double d to the mean of its contained values.

```
d=mean(v);
```

2.5. *Print Functions*

Here we display both functions implementable in SASSi: numbers and strings. Notice that you can only do one at a time.

```
print("The mean of doubles in vector v is:\n");
print(d);
print("\n");
```

This would print out:

```
The mean of doubles in vector v is:
3.1666
```

2.6. *Vector Arithmetic*

Here we're simply adding the values of positions zero and five in the vector v, and storing the result in e. This should assign e to 8.

```
e=v[0]+v[5];
```

2.7. *Statistics Procedure*

Here we use the built-in factorial procedure, which will take the value of e (8), and assign f to the result: 40320.

```
f=factorial(e);
```

2.8. *if/else Statements*

The factorial procedure usually produces a large result, so we'll actually do some error checking of the input variable e, using if and else statements.

```
if(e<=10)
{
  f=factorial(e);
  print(f);
}
else
{
  print("The number you have entered: ");
  print(e);
  print("is too large to do a factorial.\n");
}
```

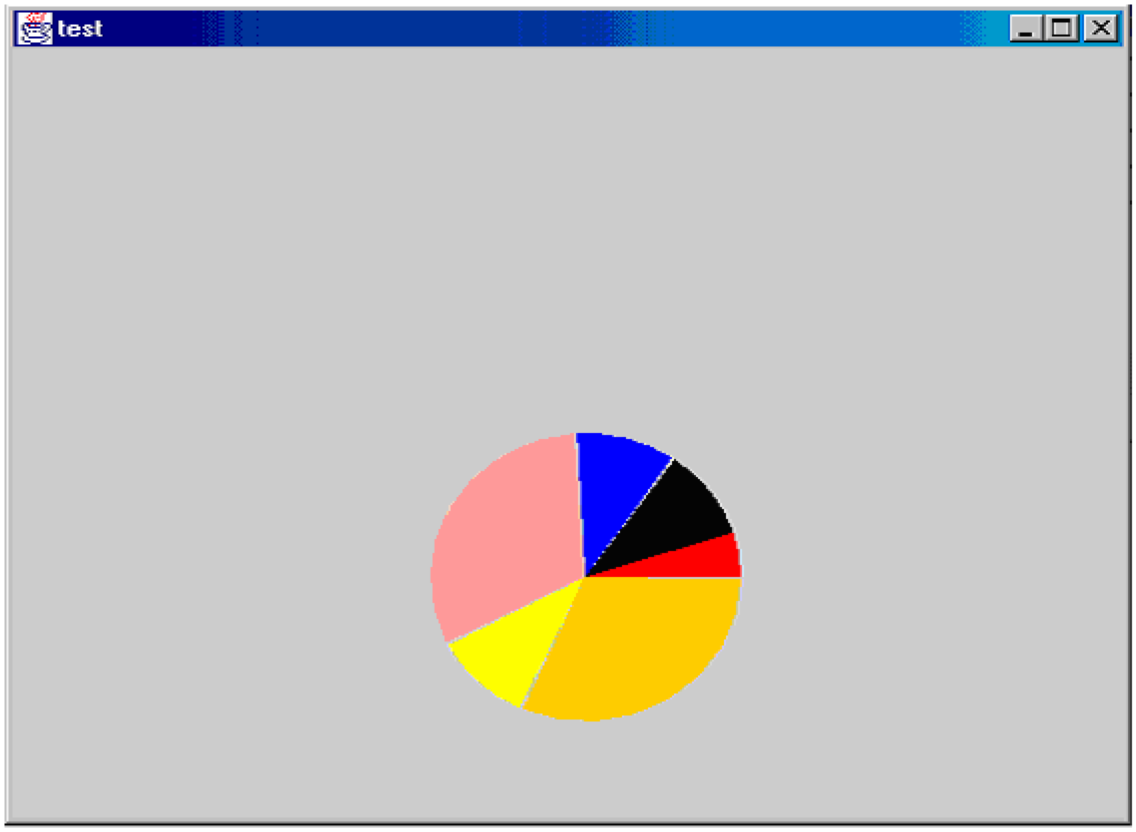
Here we only did the factorial if e is less than or equal to 10, which it is, so what's in the else statement won't be printed. Also, notice the arrangement of the braces, and the two-space indentation of the code in the statement blocks.

2.9. Plotting Procedure

Here we're simply showing one of the five plotting procedures built into the SASSi language.

```
plot("pie", v);
```

The plot procedure is called with type of plot as a string, and then a vector. We selected "pie" or pie chart simply because it was the most difficult to implement… and also because our TA didn't think we could do it. This plot procedure outputs the following plot:



2.10. Finishing Up

After all this we save the file “test.ssi”, and run it through the SASSi interpreter.

3. LANGUAGE REFERENCE MANUAL

3.1. Lexical Conventions

3.1.1. Comments

Comments are enclosed by the characters “/*” and “*/”. Also, the “//” style of commenting for a single line applies.

3.1.2. Identifiers

An identifier is under the same rules that bind their name in C. It must consist of letters, digits, and underscores (“_”) for long names. The first character must be a letter. All identifiers are case sensitive meaning that “A” is considered completely different from “a”; this assists in checking uniqueness.

3.1.3. Keywords

The following words are reserved and thus can not be used as identifiers by the user:

for	if	else	return	include	procedure	load
true	false	plot	print	const	xconst	err
bool	string	vector	inf	pi	exp	

3.1.4. Numbers

An integer is a set of digits preceded by an optional “-” signifying a negative integer.

A double floating-point number follows the same rules that are applied in C. It will consist of digits, an optional decimal point “.”, another set of digits, and an “E” or “e” for exponential numbers followed by a signed integer. Integers and double floating point numbers will be distinguishable. It should be noted, a double floating-point number need not have an exponential part. Also, regular integers will be accepted.

3.1.5. String Literals

String literals will be defined as anything confined within a set of double quotes. Double quotes inside the string literal will be preceded by a backslash, i.e. \". Also note that there is no support for chars. As an alternative, we do support strings of length one.

3.1.6. *Other Tokens*

There are a few reserved characters or pairs of characters that must be used correctly in the language. The reserved characters and pairs are as follows:

{	}	(	)	[	]	=
+	-	*	/	<	>	,
==	<>	<=	>=	;		

3.2. *Types*

bool:	Boolean values that can either be **true** or **false** (and True/False and TRUE/FALSE)
int:	standard 32-bit integers
double:	64-bit IEEE double floating-point format
string:	a string of characters
procedure:	a user defined procedure
vector:	an M by 1 vector of **doubles**

3.3. *Expression*

3.3.1. *Primary Expression*

Primary expressions are identifiers, constants, procedure calls, and accesses to vector types. They also include any expression in between parentheses—"(" and ")".

3.3.2. *Identifier*

An identifier is a left-value expression. It will be evaluated to some value delimited by the restrictions of left-value expressions, its type.

3.3.3. *Constant*

A constant is a right-value expression. It will be evaluated to the constant itself.

3.3.4. *Procedure Calls*

A procedure call consists of a procedure identifier, followed by the appropriate list of parameters that are enclosed in parentheses—"(" and ")". The list of parameters contains zero or more identifiers all separated by commas. Each parameter is an expression and each procedure call is itself left-valued.

3.3.5. *Access to Vector Types*

The primary expression consists of a vector identifier or other left-valued expression. This is followed by a list of indexes enclosed by brackets—"[" and "]". The list of indexes can contain as many numbers as the vector is defined to hold. The expression itself is left-valued.

3.3.6. *(expression)*

A parenthesized expression is a primary expression. It returns the value of the enclosed expression. The presence of the parentheses gives the expression a higher priority than the other expressions in the statement.

3.3.7. *Arithmetic Expressions*

Arithmetic expressions take primary expressions as operands.

Unary Arithmetic operators

The operators "+" and "-" can be prefixed to an expression. The "+"operator returns the expression in a positive form and the "-" returns the expression in a negative form. They are applicable only to the following: int, double, and vector.

Multiplicative Operators

Binary operators "*" and "/" indicate multiplication and division respectively. They are grouped left to right. They are applicable only to the following: int, double, and vector.

Additive operators

Binary operators "+" and "-" indicate addition and subtraction respectively. They are grouped left to right. They are applicable only to the following: int, double, and vector.

3.3.8. Relational Expressions

Binary relational operators "<=", ">=", "==", "<", ">", and "<>" indicate if the first operand is less than or equal to, greater than or equal to, equal to, less than, greater than, or not equal to the second operand, respectively. A bool value is returned in the case that both operands are numbers (i.e. not characters, strings, or vectors). Exactly two operands are needed.

Only "==" and "<>" are used in the vectors. If two vectors are compared these will tell you if they are exactly equal or exactly not equal by checking the values in each vector. Also, these only work if the vectors being compared have the same size.

3.4. Vector Indexes

Vector indices can have only one component (i.e. m[3]). Index components can be a single arithmetic expression with an integer compatible value. Any 1-by-1 sub-vector is handled just as an element that matches the type of the vector.

3.5. Variable and Constant Declaration

There are two types of data: variables and constants. Constants are usually defined at the beginning of programs and procedures. They are of the form:

somename = *number*;
.
.

The little dots are used to denote that you can put more statements in the const section. Declaring all constants at the beginning is an easy way to separate from variables, which can be declared anywhere.

Variables can be declared anywhere in the program. It takes a similar form as the constant declaration and looks like :

somename;

or

somename=*initial value*;

The second version of the variable declaration takes an initial value and automatically assigns the variable to that value. This notation saves space. Also, it should be noted that the initial value has to match the type that the variable is. The type must be one of the ones defined previously in the type section(Section 2).

3.6. Statements

Statements are the basis of a program. They take on form and structure with certain notation. Most instructions are executed sequentially but recursion is a viable option. Items that are to be replaced by the user are in italics.

3.6.1. Statements in "{" and "}"s

A group of zero or more statements can be enclosed by curly brackets. This modularizes the program. Any statements that are contained within curly brackets are grouped and run together in a sequential manner.

3.6.2. Assignments

An assignment can take one forms. The assignment form is a direct assignment in the form of:

Left-valued expression = *Right-valued expression*

This form assigns whatever is on the right of the equal sign to whatever is on the left.

3.6.3. Conditional Statements

The conditional statement can take one of two forms. The first form is a simple if and looks exactly like:

```
if ( relational expression )
{
        statement
}
```

This form executes whatever is within the curly brackets if the logical expression evaluates to true. Otherwise, it skips execution of what is within the curly brackets and continues with the program. The second form is the first version followed by an else clause. It looks like:

```
if ( relational expression )
{
        statement
}
else
{
        statement
}
```

This executes the same way as the first form except for the fact that if the logical expression evaluates to false then what is enclosed within the curly brackets after the else statement are evaluated. It is also important to know that these two forms of conditional statements can be nested within each other. By nesting conditional statements you effectively reduce execution time.

3.6.4. Procedure Calls

Procedures are different than any other expression. They follow the following form:

```
procedure_name ( parameter list );
```

The *procedure_name* must be a procedure that the user has already defined and must be spelled the correct way. The *parameter list* is the name of the appropriate variables separated by a comma (i.e. foobar(who, knows, why)). The *parameter list* may also be empty, this means that the procedure doesn't take anything. The *parameter list* variables must match the types they are being sent to be assigned. This prevents type mismatching like assigning an **int** to a procedure variable that requires a **double**.

3.6.5. Return Statements

The return statement can be used within the procedure definition body. It returns either a variable's value or a number to the caller. It is mandatory that you end the statement with a semicolon. A basic return statement would look like:

```
return 0;
```

3.6.6. Including Other Files

The keyword include is used to include other files. The form is as follows:

include <*path name*>;

This allows you to include previous work or procedures that you wish. It also reduces the amount of memory you consume by letting you choose which things you wish to include in your palette of procedures. The *path name* must be exact. The file will not be included if the *path name* entered is not correct. Also, this declaration must be made at the top of your program.

3.7. Procedure Definitions

A procedure takes only one form but what may be accomplished is unlimited. It follows the form below:

```
procedure procedure_name ( variable list)
{
        statements
}
```

The return type is decided when the code is interpreted The *procedure_name* is any name that you give but it must follow the guidelines defined by **identifiers**. The *variable list* is a list of parameters defined in the form of type and then identifier and each is separated by a comma. An example of this is:

```
procedure doesnothing(int x, double y)
{

}
```

It is important to note that the *variable list* can be empty. You will also notice that it is not necessary to have any statements in the body of the procedure. It will compile with no statements. It is of course, your prerogative to have a procedure that does nothing so it is supported.

3.8. Internal Functions

3.8.1. Print

The function print() has two different versions. The first version takes a variable as an argument and prints out the value of the variable. This is only used for non-vector types. It takes the form:

```
print(v);
```

The second version takes a string and prints it out to the screen. The string must be enclosed in double quotes and takes the form:

```
print("Who knows why you name something foo?");
```

3.8.2. What

There are two forms of the what() procedure. The first is called as below:

```
what();
```

This returns all of the variables that have been type defined in the program. Or the second form can be used:

```
what(a);
```

This only returns the type of the variable a.

3.8.3. Plot

The plot procedure draws a plot that you request. The types of plots aren't limited to just the ones listed below because you can implement your own plot type. The plot command follows the form:

plot(*type_of_plot*, *type_of_data*);

The *type_of_data* can be only one of two types: vector or ordered series. The ordered series take the form:

```
[x1,x2…], [y1,y2…]
```

This form is simple to implement and allows the user just to put in the numbers they wish to plot. The vector already has its numbers defined so the call reacts exactly the same with either type. The *type_of_plot* falls under a user defined type or one of the following predefined types:

bar:	a bar graph
pie:	a pie chart
line:	a line graph
error	a line graph with error bars added
curve:	multiple points forming a line

It is important to note that a program utilizing an error plot must have a variable of type **double** that is called **err**. The **err** value must be set before the plot call is made. This value can be changed after the plot call and thus is not necessarily a constant.

3.8.4. Built-in Statistics Constants

These are constants that are used in mathematical evaluations. They are provided as a convenience to the user.

```
inf=maxInt;
```

This is the value for infinity. It is defined at the maxInt of the system or the largest integer that can be represented by your computer.

```
pi= 3.14159265358979323846264338327950;
```

This is the value of pi. Since pi is a very important number and curious within it self, we have defined it to a semi-exact point. If we were to implement it to the millionth decimal it would be quite a waste of space and take considerable time to be used in calculation.

```
exp=2.71828182846;
```

This is the value of the exponential. This is included because it is used in various other functions and is frequently used in mathematical calculations. Please note that imaginary numbers are not supported and there are no tools for conversion from an exponential notation to the imaginary notation.

3.8.5. Built-in Statistics Procedures

Each of the following procedures is an available function to the user. This is the default set of statistical procedures. Others are implemented and can be used, they however must be included by putting an include statement in your program. The procedures below do not require any include statements to be added to your program.

Note that SASSi interprets the procedure and expression types, so even though we specify types in the expression below, it is only because we're showing what types the procedures expect as input, and what the procedures output.

double mean(SsVector vect)

This procedure takes in a vector of doubles as its only argument. It finds the mean of all the elements with in that vector and returns the value as a double.

double median(SsVector vect)

This procedure takes in a vector as its only argument. It finds the median of all the elements within that vector and returns the value as an int. The return type matches the type of elements in the vector.

SsVector sort(SsVector vect)

This procedure takes in a vector of doubles as its only argument. It sorts the entire vector in ascending order. This is very useful for an n-by-one vector.

double range(SsVector vect)

This procedure takes in a vector of doubles as its only argument. It finds the highest and lowest values within the vector and returns the difference between them.

double variance(SsVector vect)

This procedure takes in a vector of doubles. It finds the variance of the vector and returns it as a double.

double stdDev(SsVector vect)

This procedure takes in a vector of doubles as its only variable. It returns the standard deviation of the elements that are stored in the vector.

SsVector intersect(SsVector vect1,SsVector vect2)

This procedure takes in two vectors of the same type. It returns a vector of the same type that was passed. The vector returned contains the intersection of the two vectors passed in which means that it contains all the values that were in the first and all the values that were in the second with no repeated values. This is helpful when dealing with set theory.

SsVector union(SsVector vect1,SsVector vect2)

This procedure takes in two vectors of the same type. It returns a vector of the same type that was passed. The vector returned contains the union of the two vectors passed in which means that it contains all the values that were in the first and in the second but not any values that were just in one or the other. This is another helpful set theory procedure.

boolean contains(SsVector vect, double number)

This procedure takes in a vector of doubles and a double. The procedure searches through the vector that was passed in for the double that was passed in to the procedure. If the double is in the vector then the index of that number is returned. If the double is not found a negative one is returned. A zero is not returned since it could be the first element in the vector which is denoted as zero.

int factorial(int x)

This procedure is essential for statistics. It takes in an integer and returns the factorial of that integer.

double sqrt(double x)

This procedure simply uses Java's sqrt function. It is essential for building statistics procedures. It takes and returns a double.

procedure size(SsVector vect)

This procedure is very simple yet very helpful. It takes in a vector of integers and returns the number of elements in the vector. This is used a lot for size checking and is very helpful.

double expectation (SsVector events, SsVector probabilities)

This procedure returns the mathematical expectation of discrete events with given probabilities.

double sxy (SsVector x,SsVector y)

This returns the value of (Σ(x * y) - (Σ(x)) * (Σ(x))) / n. Where n is the size of vector x and y. This is used for calculating linear regressions.

Discrete distributions
f(x) is the probability function and p(X) is the probability for all x <= X.

double binoDistri(int x, int n, double p)

The procedure takes in a value x which is smaller than n and returns the binomial distribution with probability p.

double geoDistri(int x, double p)

This is a procedure that takes in a value x and the probability p and returns the geometric distribution

Continuous distribution
p(X) returns the probability for all x <= X

double normalDistri (double x, double sigma, double mean)

This procedure performs the Normal Distribution.

double stdDistri (double x)

This procedure performs the Standard Distribution using only a double x.

4. PROJECT PLAN

4.1. Team Responsibilities

Our team has met regularly for about 45 minutes, twice a week to discuss our progress, problems, and solutions. In our group meetings, we all decided what functions should and shouldn't be included in SASSi (mostly shouldn't). Each member was assigned different tasks to complete, thus we all had individual goals to help the group reach its overall goal of completion.

Since each member has a varying level of experience programming, we tried to break down the task according to each person's strengths and weaknesses, both on a programming level, and a comunication level. Below is a break down of what each member was responsible for during the life of the project.

Carl Morgan:	Team leader; front-end: lexical analyzer, parser, tree walker; documentation
Paul Salama:	Graphics; documentation writer and editor; presentation; tester
Xiaotang Zhang:	Statistics libraries and functions; back-end;

4.2. Programming Style (Coding Conventions)

4.2.1. ANTLR Coding Style

ANTLR is a language that is designed for writing parsers, lexical analyzers, and tree walkers, amongst other things. If one wants to define a rule that is short and takes no action, then it can be done using only one line of code. The format for the code is as follows:

name: *statement* ;

If the *name* is to be a token the all its letters are capitalized. If the *name* is to be used syntactically or for non terminal items, all the letters are in lower case. It should be noted that using an underscore(_) is valid in either case. This convention makes it very simple to distinguish between the two forms. The colon (:) is always placed after the name and denotes the start of code. The *statement* actually can be a series of different options or cases of the name. If you are to have multiple *statements* they must be separated by a bar (|) and there must be no ambiguity between the two different options. Every declaration is terminated by a semi colon(;). Action coding, i.e. things that affect the tree walker, are done in the Java coding style.

4.2.2. Java Coding Style

All of the group members have their own programming style, even if they don't know it. We tried to come up with a uniform set of rules that we could all adhere to; it's listed below.

Indentation and spacing:

1) Indentation of each level is 2 spaces.
2) The left brace ({) occupies its own line and is aligned with the first letter of the previous line.
3) The right brace (}) also occupies its own line and is aligned with the left brace.
4) There is one space between the left parenthesis and the first variable.
5) There is no space between the name of the procedure and the left parenthesis of the argument list
6) Add spaces between operators and operands for outer expressions.
7) Use spaces, do not use tabs.
8) The action statements for an if/else/for statement must not be on the same line as the left brace for the respective sections. These also adhere to the same brace rules explained above.

Names:

1) Class names start with "SASSi" followed by English words whose first character is capitalized.
2) Variables are in lower cases, and words are separated by underscores (_).
3) Brevity should always be used in naming of local variables.

4.2.3. SASSi Coding Style

Since our back-end programming was done in Java, we (unintentionally) decided to utilize a Java-like form for writing in SASSi, borrowing from the above rules for Java.

Indentation and spacing:

1) Indentation of each level is 2 spaces.
2) The left brace ({) occupies its own line and is aligned with the first letter of the previous line.
3) The right brace (}) also occupies its own line and is aligned with the left brace.
4) There is no space between the name of the procedure and the left parenthesis of the argument list
5) Use spaces, do not use tabs.
6) The action statements for an if/else/for statement must not be on the same line as the left brace for the respective sections. These also adhere to the same brace rules explained above.

Names:

1) SASSi files have the file suffix "ssi", so files are of the form *name.ssi*.
2) Variables are in lower cases, and words are separated by underscores (_).
3) Brevity should always be used in naming of local variables.
4) Procedure names are English words whose first characters are capitalized.

4.3. Project Timeline

Here are the deadlines that we defined for ourselves to keep on track

September	21	Language White Paper Draft done
September	23	Language White Paper
October	21	First version of Language Reference Manual done
October	28	Language Reference Manual final version completed

		Finished Lexical Analyzer and Parser
		Start Tree Walker
November	6	Finished Tree Walker
November	20	Tuning of work to date
November	28	Finish First Version of Final Manual
December	7	Presentation Finished
December	9	Presentation
December	17	Entire Project Finished
December	19	Final Meeting with Professor Edwards

4.4. Software Development Environment

Most of the programs are written in Java. The lexical analyzer, parser, and tree walker are written in ANTLR, and will be translated into java code.

4.4.1. Operating Systems

Since this language is developed in pure Java, it can be used anywhere a Java VM is running. We took advantage of this by programming in Linux in the CLIC lab and Windows everywhere else.

4.4.2. Java 1.3.1

Java is a "simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language."

4.4.3. ANTLR 2.7.2

The language parser, lexical analyzer, and tree walker are written in ANTLR, "a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions."

4.4.4. Programming Tools

Each group member had their own programs of choice for programming and editing.

Carl Morgan:	NEdit 5.3 for everything (Highly Recommended)
Paul Salama:	NetBeans IDE 3.5.1 for Java programming, TextPad 4.7.2 for SASSi file editing
Xiaotang Zhang:	VisualAge for Java 4.0 (even though it will be discontinued by the end of the year)

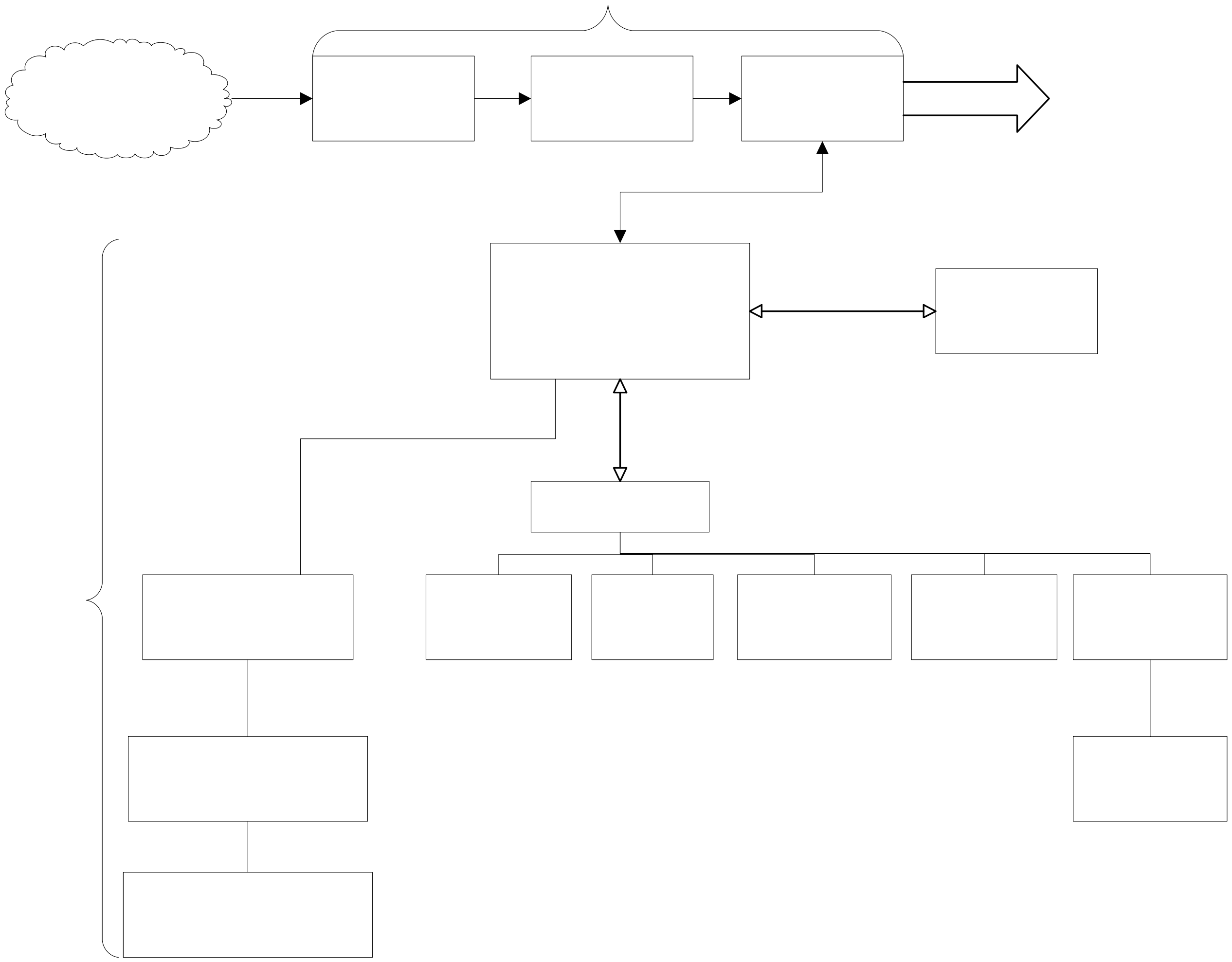
Documentation was written in Microsoft Word, and then converted to PDF using Adobe Acrobat 6.0. Our presentation was made with Microsoft PowerPoint. Diagrams were made with Microsoft Visio.

4.5. Project Log

Here are the dates of significant project milestones. This is a much more realistic version of what happened than what we planned.

September	22	Decided to create SASSi
September	23	Language White Paper
September	30	Define Framework
October	7	Define Proper Rules / Start Lexical Analyzer & Parser
October	14	Define Supported Functions
October	21	First version of Language Reference Manual done
October	28	Language Reference Manual final version completed Finished Lexical Analyzer and Parser Start Tree Walker
November	6	Framework of Graphics library Finished Tree Walker
November	20	Tuning of work to date
November	28	Finish First Version of Final Manual
December	5	Front End/ Code Generation Finished
December	8	Presentation Finished
December	9	Presentation
December	15-17	Error Testing, more Error Testing
December	17	Combined Documentation Entire Project Finished
December	18	Tuning Final Meeting With Professor Edwards

5. ARCHITECTUAL DESIGN



Command-line input, or *.ssi file (usually made by Paul)

6. TEST PLAN

6.1. *Goal*

Since our compiler is supposed to conduct statistical operations, testing is essential, because the compiler must be reliable. Testing has evolved throughout the whole project, from when the grammar was implemented to the very end. Each part is tested thoroughly before being merged into the final project.

6.2. *The First Stage: Command-Line Testing*

There are many things that can go wrong, and many things did. Our command-line allowed us to test components on a small scale. We first wanted to test the basic mathematics of SASSi, because it's essential for all further procedures in SASSi. We then moved on to building simple procedures, and build more complex ones from there. This was our basic mode of testing.

6.3. *The Second Stage: Advanced Testing Examples*

The procedures below are very useful statistics functions. They happen to be complex so they have the can be utilized as complex tests for our programming language. They also have the added benefit of proving that our less-is-more implementation (not providing many statistics functions), is a workable approach.

6.4. *permute.ssi*

Just to show that we could implement statistics functions, we included a file which has a procedure that takes two integer inputs and prints out the permutation of those two numbers. In the file permute.ssi we called the procedure with the number 9 and 6, which should hopefully give us the answer for 9 nPr 6. So we run the file using the following command:

```
java –jar sassi.jar permute.ssi
```

We get the following results:

```
n
r
done = 60480
```

60,480 is in fact the answer we were looking for. The code for this program is in the appendix.

6.5. school.ssi

Here's a program for student who might want to calculate various aspects of their grades.

```
homework=0.2;
project=0.3;
bullsh_t=0.1;
midterm=0.15;
final=0.25;
```

These are the percentages that each component of the chosen course counts towards the final grade.

```
homeworks=[36,24,39,27];
average=mean(homeworks);
print(average);
```

Here we have four homeworks that are out of 40, and we print the average, and get the result:

```
average = 31.5
```

Now we could find out what the grades are out of 100, with the following lines:

```
homeworks1=homeworks*2.5;
average=mean(homeworks1);
```

We also have a standard deviation function, if you're into that sorta thing:

```
std=stdDev(homeworks);
```

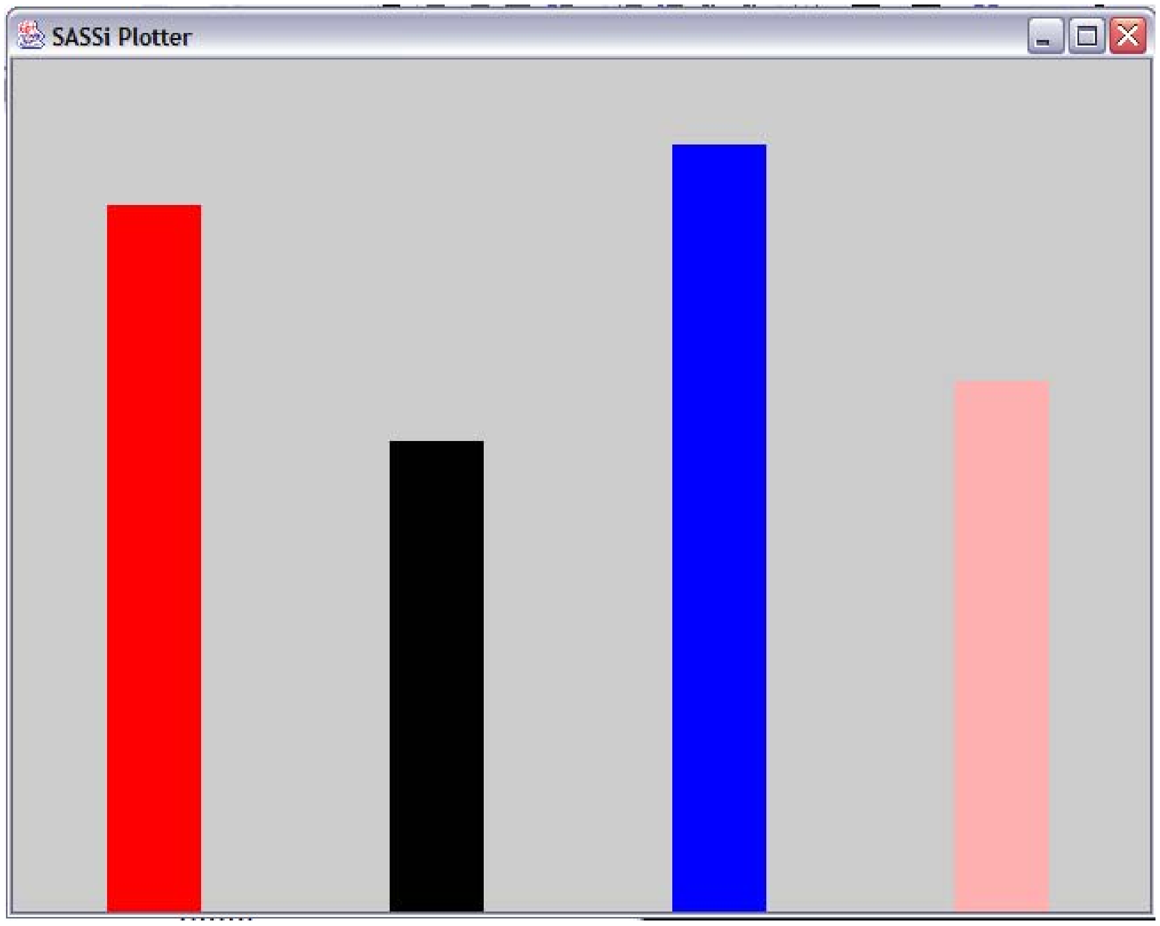
```
print(std);
```

We get the result:

```
std = 37.0675059 (etc…)
```

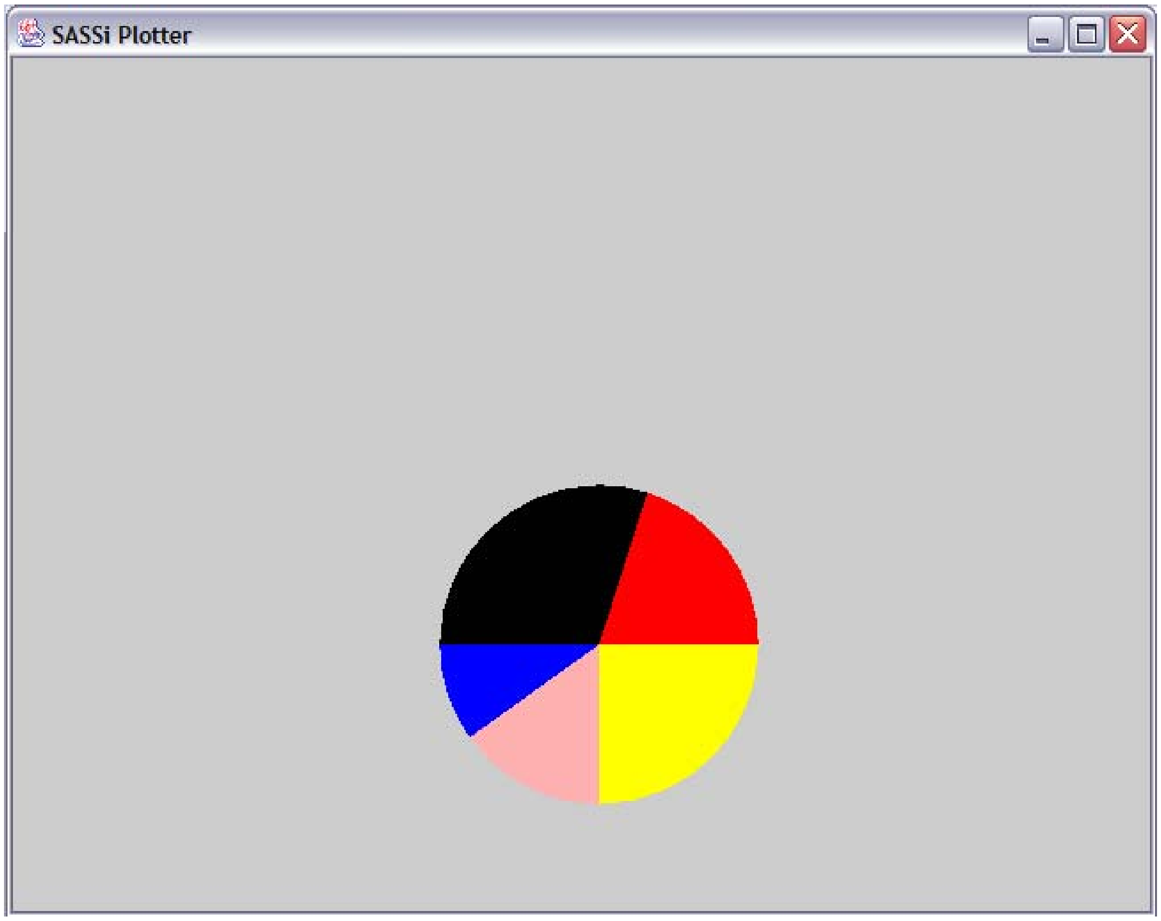
We can also plot the various grades of homework in a bar graph with:

```
plot("bar",homeworks);
```



Now we put all of the different components into a vector, then multiply it by one hundred to get whole numbers. Then we can plot it as a pie chart:

```
total=[homework,project,bullsh_t,midterm,final];
total=total*100;
plot("pie",total);
```



And we're done.

7. LESSONS LEARNED

These are some of the lessons that we have learned throughout the process of creating this project. As the saying goes, "Learn from others so that you don't make the mistakes yourself."

7.1. Carl's Lessons

Get feedback from the teaching assistant. They are there to help us and have been through the same troubles you're having now.

Think big, build small. Our project was way too big at the start, meaning we tried to include too many functions from C and statistics. By trimming back what we were going to include, we made the project far more manageable. Besides, we can always go back and build on our solid foundation.

7.2. Paul's Lessons

Schedule two meetings a week. That way, if something comes up and someone can't make it, you don't worry about it. Also, it's easier to have two one-hour meetings instead of one two-hour meeting.

It's nice to have a group leader who is on top of things. It occurred on a number of occasions that I'd see something was required of our group at various times, and not know at all what our status was. I'd ask Carl about it, and he'd always already have it planned out, if not already done.

7.3. Xiaotang's Lessons

Decide early on if your language is really better being a language or a program. If it is better off being a program then there is no need to implement it as a new language. This is a direct result of the fact that the night before the white paper was due we switched the complete idea of our project after realizing that the initial project would best be implemented as a program and not a language to build programs.

8. APPENDIX

8.1. SASSiBool.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 10:11:12)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;

class SASSiBool extends SASSiDataType {
    boolean var;

    SASSiBool( boolean var ) {
        this.var = var;
    }
    public SASSiDataType and( SASSiDataType b ) {
        if ( b instanceof SASSiBool )
            return new SASSiBool( var && ((SASSiBool)b).var );
        return error( b, "and" );
    }
    public SASSiDataType copy() {
        return new SASSiBool( var );
    }
    public SASSiDataType eq( SASSiDataType b ) {
        if ( b instanceof SASSiBool )
            return new SASSiBool( ( var && ((SASSiBool)b).var )
                                  || ( !var && !((SASSiBool)b).var
) );
        return error( b, "==" );
    }
    public SASSiDataType ne( SASSiDataType b ) {
        if ( b instanceof SASSiBool )
            return new SASSiBool( ( var && !((SASSiBool)b).var )
                                  || ( !var && ((SASSiBool)b).var )
);
        return error( b, "!=" );
    }
    public SASSiDataType not() {
        return new SASSiBool( !var );
    }
    public SASSiDataType or( SASSiDataType b ) {
        if ( b instanceof SASSiBool )
```

```
            return new SASSiBool( var || ((SASSiBool)b).var );
        return error( b, "or" );
    }
    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( var ? "true" : "false" );
    }
    public String typename() {
        return "bool";
    }
}
```

8.2. *SASSiBuiltInProcedure.java*

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-10 19:18:20)
 * @author: Xiaotang Zhang
 */
import java.io.IOException;


class SASSiBuiltInProcedure {
    static int anIntValue = 0;
    static double aDoubleValue = 0;
     static boolean aBool = false;
    static SASSiStatisticalMethords aMethod  = new
SASSiStatisticalMethords();
    static SASSiPlotter plotter = null;

    final static int p_print = 0;
    final static int p_what = 1;
    final static int p_plot = 2;
    final static int p_binoDistri = 3;
    final static int p_contains = 4;
    final static int p_expectation = 5;
    final static int p_factorial = 6;
    final static int p_geoDistri = 7;
    final static int p_intersect = 8;
    final static int p_linearReg = 9;
    final static int p_mean = 10;
    final static int p_median = 11;
    final static int p_sxy = 12;
    final static int p_normalDistri = 13;
    final static int p_range = 14;
    final static int p_size = 15;
    final static int p_sort = 16;
    final static int p_stdDev = 17;
    final static int p_stdDistri = 18;
    final static int p_union = 19;
    final static int p_variance = 20;

    public static void register( SASSiSymbolTable st ) {

              st.put( "print", new SASSiProcedureCall( null,
p_print ) );  //0
              st.put( "what", new SASSiProcedureCall( null,
p_what ) );  //1
              st.put( "plot", new SASSiProcedureCall( null,
p_plot ) );  //2
              st.put( "binoDistri", new SASSiProcedureCall(
null, p_binoDistri ) ); //3
              st.put( "contains", new SASSiProcedureCall( null,
p_contains ) ); //4
              st.put( "expectation", new SASSiProcedureCall(
null, p_expectation ) ); //5
              st.put( "factorial", new SASSiProcedureCall( null,
p_factorial ) ); //6
              st.put( "geoDistri", new SASSiProcedureCall( null,
p_geoDistri ) ); //7
              st.put( "intersect", new SASSiProcedureCall( null,
p_intersect ) ); //8
              st.put( "linearReg", new SASSiProcedureCall( null,
p_linearReg ) ); //9
              st.put( "mean", new SASSiProcedureCall( null,
p_mean ) ); //10
              st.put( "median", new SASSiProcedureCall( null,
p_median ) ); //11
              st.put( "sxy", new SASSiProcedureCall( null, p_sxy
) ); //12
              st.put( "normalDistri", new SASSiProcedureCall(
null, p_normalDistri ) ); //13
              st.put( "range", new SASSiProcedureCall( null,
p_range ) ); //14
              st.put( "size ", new SASSiProcedureCall( null,
p_size ) ); //15
              st.put( "sort", new SASSiProcedureCall( null,
p_sort ) ); //16
              st.put( "stdDev", new SASSiProcedureCall( null,
p_stdDev ) ); //17
```

```
              st.put( "stdDistri", new SASSiProcedureCall( null,
p_stdDistri ) ); //18
              st.put( "union", new SASSiProcedureCall( null,
p_union ) ); //19
              st.put( "variance", new SASSiProcedureCall( null,
p_variance ) ); //20
              st.put( "E", new SASSiDouble( Math.E ) );
            st.put( "PI", new SASSiDouble( Math.PI ) );
    }
    public static SASSiDataType run( SASSiSymbolTable st,
                              int id, SASSiDataType[] params
) {
         switch ( id )
        {
        case p_print:
            for ( int i=0; i<params.length; i++ )
                params[i].print();
            return null;

        case p_what:
            if ( params.length > 0 )
            {
                for ( int i=0; i<params.length; i++ )
                        params[i].what();
            }
            else
                st.what();
            return null;

        case p_plot:
            if ( 2 != params.length )
                throw new SASSiException( "plot() needs 2
parameters" );;
            if ( !( params[0] instanceof SASSiString ) )
                throw new SASSiException( "First arg should be a
string" );
            if ( !( params[1] instanceof SASSiVector ) )
                throw new SASSiException( "Second arg should be
a vector" );
            plotter = SASSiPlotter.create( "SASSi Plotter", 640,
480 );
            SsVector vect = ((SASSiVector)params[1]).vect;
            String type = ((SASSiString)params[0]).var;
            if(type.equals("bar"))
            {

     plotter.drawBarChart(((SASSiVector)params[1]).vect);
```

```
               return null;
            }
            else if (type.equals("pie"))
            {

     plotter.drawPieChart(((SASSiVector)params[1]).vect);
               return null;
            }
            else if (type.equals("line"))
            {
                 if(((SASSiVector)params[1]).vect.size() < 2)
                    throw new SASSiException( "To print a line,
the vector should contain at least 2 elements!" );
               plotter.drawLine(((SASSiVector)params[1]).vect);
               return null;
            }
            else if (type.equals("polyline"))
            {

     plotter.drawPolyLine(((SASSiVector)params[1]).vect);
               return null;
            }
            else if (type.equals("error"))
            {
               plotter.drawError(((SASSiVector)params[1]).vect);
               return null;
            }
            else
               throw new SASSiException( "SASSi is not
supporting plot with type " + type );

         case p_binoDistri: //3
            if ( params.length != 3  )
                throw new SASSiException( "binoDistri() accepts
3 parameter(s)" );;
            aDoubleValue =
aMethod.binoDistri(SASSiInt.intValue(params[0]),

SASSiInt.intValue(params[1]),

SASSiDouble.doubleValue(params[2]));
            return new SASSiDouble(aDoubleValue);

        case p_contains: //4
            if ( params.length != 2  )
                throw new SASSiException( "contains() accepts 2
parameter(s)" );;
```

```
            aBool =
aMethod.contains(((SASSiVector)params[0]).vect,

SASSiDouble.doubleValue(params[1]));
            return new SASSiBool(aBool);

          case p_expectation: //5
            if ( params.length != 2  )
                throw new SASSiException( "expectation() accepts
2 parameter(s)" );;
            aDoubleValue =
aMethod.expectation(((SASSiVector)params[0]).vect,

((SASSiVector)params[1]).vect);
            return new SASSiDouble(aDoubleValue);

          case p_factorial: //6
            if ( params.length != 1  )
                throw new SASSiException( "factorial() accepts 1
parameter(s)" );;
            anIntValue =
aMethod.factorial(SASSiInt.intValue(params[0]));
            return new SASSiInt(anIntValue);

          case p_geoDistri: //7
            if ( params.length != 2  )
                throw new SASSiException( "geoDistri() accepts 2
parameter(s)" );;
            aDoubleValue =
aMethod.geoDistri(SASSiInt.intValue(params[0]),

SASSiDouble.doubleValue(params[1]));
            return new SASSiDouble(aDoubleValue);

        case p_intersect: //8
            if ( params.length != 2  )
                throw new SASSiException( "intersect() accepts 2
parameter(s)" );

            return new
SASSiVector(aMethod.intersect(((SASSiVector)params[0]).vect,

((SASSiVector)params[1]).vect));

        case p_linearReg: //9
            if ( params.length != 2  )
```

```
                throw new SASSiException( "linearReg() accepts 2
parameter(s)" );

            return new
SASSiVector(aMethod.linearReg(((SASSiVector)params[0]).vect,

((SASSiVector)params[1]).vect));

          case p_mean: //10
            if ( params.length != 1  )
                throw new SASSiException( "mean() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.mean(((SASSiVector)params[0]).vect);
            return new SASSiDouble(aDoubleValue);

        case p_median: //11
            if ( params.length != 1  )
                throw new SASSiException( "median() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.median(((SASSiVector)params[0]).vect);
            return new SASSiDouble(aDoubleValue);

        case p_sxy: //12
            if ( params.length != 3  )
                throw new SASSiException( "sxy() accepts 2
parameter(s)" );;
            return new
SASSiDouble(aMethod.sxy(((SASSiVector)params[0]).vect,

((SASSiVector)params[1]).vect));

        case p_normalDistri: //13
            if ( params.length != 3  )
                throw new SASSiException( "normalDistri()
accepts 3   parameter(s)" );;
            aDoubleValue =
aMethod.normalDistri(SASSiDouble.doubleValue(params[0]),

SASSiDouble.doubleValue(params[1]),

SASSiDouble.doubleValue(params[2]));
            return new SASSiDouble(aDoubleValue);

       case p_range: //14
            if ( params.length != 1 )
```

```
                throw new SASSiException( "range() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.range(((SASSiVector)params[0]).vect);
            return new SASSiDouble(aDoubleValue);

       case p_size: //15
            if ( params.length != 1  )
                throw new SASSiException( "colors() accepts 1
parameter(s)" );;
            anIntValue =
aMethod.size(((SASSiVector)params[0]).vect);
            return new SASSiInt(anIntValue);

       case p_sort: //16
            if ( params.length != 1  )
                throw new SASSiException( "sort() accepts 1
parameter(s)" );;
            return new
SASSiVector(aMethod.sort(((SASSiVector)params[0]).vect));

       case p_stdDev: //17
            if ( params.length != 1  )
                throw new SASSiException( "colors() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.stdDev(((SASSiVector)params[0]).vect);
            return new SASSiDouble(aDoubleValue);

       case p_stdDistri: //18
            if ( params.length != 1  )
                throw new SASSiException( "stdDistri() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.stdDistri(SASSiDouble.doubleValue(params[0]));
            return new SASSiDouble(aDoubleValue);

       case p_union: //19
            if ( params.length != 2  )
                throw new SASSiException( "union() accepts 2
parameter(s)" );

            return new
SASSiVector(aMethod.union(((SASSiVector)params[0]).vect,

((SASSiVector)params[1]).vect));
```

```
       case p_variance: //20
            if ( params.length != 1  )
                throw new SASSiException( "variance() accepts 1
parameter(s)" );;
            aDoubleValue =
aMethod.variance(((SASSiVector)params[0]).vect);
            return new SASSiDouble(aDoubleValue);

       default:
            throw new SASSiException( "unknown built-in
procedure" );
        }
    }
}
```

8.3. SASSiDataType.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 10:19:22)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 */
public class SASSiDataType
{
    String name;   // used in hash table

    public SASSiDataType() {
        name = null;
    }
    public SASSiDataType( String name ) {
        this.name = name;
    }
    public SASSiDataType add( SASSiDataType b ) {
        return error( b, "+=" );
    }
    public SASSiDataType and( SASSiDataType b ) {
        return error( b, "and" );
    }
```

```
public SASSiDataType assign( SASSiDataType b ) {
    return error( b, "=" );
}
public SASSiDataType copy() {
    return new SASSiDataType();
}
public SASSiDataType eq( SASSiDataType b ) {
    return error( b, "==" );
}
public SASSiDataType error( String msg ) {
    throw new SASSiException( "illegal operation: " + msg
                           + "( <" + typename() + "> "
                           + ( name != null ? name : "<?>" )
                           + " )" );
}
public SASSiDataType error( SASSiDataType b, String msg ) {
    if ( null == b )
        return error( msg );
    throw new SASSiException(
        "illegal operation: " + msg
        + "( <" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + typename() + "> "
        + ( name != null ? name : "<?>" )
        + " )" );
}
public SASSiDataType ge( SASSiDataType b ) {
    return error( b, ">=" );
}
public SASSiDataType gt( SASSiDataType b ) {
    return error( b, ">" );
}
public SASSiDataType ldiv( SASSiDataType b ) {
    return error( b, "/=" );
}
public SASSiDataType le( SASSiDataType b ) {
    return error( b, "<=" );
}
public SASSiDataType lfracts( SASSiDataType b ) {
    return error( b, "/" );
}
public SASSiDataType lt( SASSiDataType b ) {
    return error( b, "<" );
}
public SASSiDataType minus( SASSiDataType b ) {
    return error( b, "-" );
```

```
}
public SASSiDataType mul( SASSiDataType b ) {
    return error( b, "*=" );
}
public SASSiDataType ne( SASSiDataType b ) {
    return error( b, "<>" );
}
public SASSiDataType not() {
    return error( "not" );
}
public SASSiDataType or( SASSiDataType b ) {
    return error( b, "or" );
}
public SASSiDataType plus( SASSiDataType b ) {
    return error( b, "+" );
}
public void print() {
    print( new PrintWriter( System.out, true ) );
}
public void print( PrintWriter w ) {
    if ( name != null )
        w.print( name + " = " );
    w.println( "<undefined>" );
}
public SASSiDataType rfracts( SASSiDataType b ) {
    return error( b, "/" );
}
public void setName( String name ) {
    this.name = name;
}
public SASSiDataType times( SASSiDataType b ) {
    return error( b, "*" );
}
public String typename() {
    return "unknown";
}
public SASSiDataType uminus() {
    return error( "-" );
}
public SASSiDataType uplus( SASSiDataType b ) {
    return error( b, "+" );
}
public void what() {
    what( new PrintWriter( System.out, true ) );
}
public void what( PrintWriter w ) {
    w.print( "<" + typename() + ">  " );
```

```
        print( w );
    }
}
```

8.4. SASSiDouble.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 16:26:29)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;

class SASSiDouble extends SASSiDataType {
    double var;

    public SASSiDouble( double x ) {
        var = x;
    }
    public SASSiDataType add( SASSiDataType b ) {
        var += doubleValue( b );
        return this;
    }
    public SASSiDataType copy() {
        return new SASSiDouble( var );
    }
    public static double doubleValue( SASSiDataType b ) {
        if ( b instanceof SASSiDouble )
            return ((SASSiDouble)b).var;
        if ( b instanceof SASSiInt )
            return (double) ((SASSiInt)b).var;
        b.error( "cast to double" );
        return 0;
    }
    public SASSiDataType eq( SASSiDataType b ) {
        return new SASSiBool( var == doubleValue(b) );
    }
    public SASSiDataType ge( SASSiDataType b ) {
        return new SASSiBool( var >= doubleValue(b) );
    }
    public SASSiDataType gt( SASSiDataType b ) {
        return new SASSiBool( var > doubleValue(b) );
    }
    public SASSiDataType ldiv( SASSiDataType b ) {
        var /= doubleValue(b);
        return this;
    }
    public SASSiDataType le( SASSiDataType b ) {
        return new SASSiBool( var <= doubleValue(b) );
    }
    public SASSiDataType lfracts( SASSiDataType b ) {
        return new SASSiDouble( var / doubleValue(b) );
    }
    public SASSiDataType lt( SASSiDataType b ) {
        return new SASSiBool( var < doubleValue(b) );
    }
    public SASSiDataType minus( SASSiDataType b ) {
        return new SASSiDouble( var - doubleValue(b) );
    }
    public SASSiDataType mul( SASSiDataType b ) {
        var *= doubleValue( b );
        return this;
    }
    public SASSiDataType ne( SASSiDataType b ) {
        return new SASSiBool( var != doubleValue(b) );
    }
    public SASSiDataType plus( SASSiDataType b ) {
        return new SASSiDouble( var + doubleValue(b) );
    }
    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Double.toString( var ) );
    }
    public SASSiDataType rdiv( SASSiDataType b ) {
        return ldiv( b );
    }
    public SASSiDataType rfracts( SASSiDataType b ) {
        return lfracts( b );
    }
    public SASSiDataType sub( SASSiDataType b ) {
        var -= doubleValue( b );
        return this;
    }
    public SASSiDataType times( SASSiDataType b ) {
        if ( b instanceof SASSiVector )
            return b.times( this );
        return new SASSiDouble( var * doubleValue(b) );
    }
    public String typename() {
        return "double";
    }
    public SASSiDataType uminus() {
        return new SASSiDouble( -var );
    }
}
```

8.5. *SASSiException.java*

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 15:36:22)
 * @author: Xiaotang Zhang
 */
class SASSiException extends RuntimeException {
    SASSiException( String msg ) {
        System.err.println( "Error: " + msg );
    }
}
```

8.6. *SASSiInt.java*

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-8 9:11:22)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;


class SASSiInt extends SASSiDataType {
    int var;

    public SASSiInt( int x ) {
        var = x;
    }
    public SASSiDataType add( SASSiDataType b ) {
        var += intValue( b );
        return this;
    }
    public SASSiDataType copy() {
        return new SASSiInt( var );
    }
    public SASSiDataType eq( SASSiDataType b ) {
        if ( b instanceof SASSiInt )
```

```
                return new SASSiBool( var == intValue(b) );
            return b.eq( this );
        }
        public SASSiDataType ge( SASSiDataType b ) {
            if ( b instanceof SASSiInt )
                return new SASSiBool( var >= intValue(b) );
            return b.le( this );
        }
        public SASSiDataType gt( SASSiDataType b ) {
            if ( b instanceof SASSiInt )
                return new SASSiBool( var > intValue(b) );
            return b.lt( this );
        }
        public static int intValue( SASSiDataType b ) {
            if ( b instanceof SASSiDouble )
                return (int) ((SASSiDouble)b).var;
            if ( b instanceof SASSiInt )
                return ((SASSiInt)b).var;
            b.error( "cast to int" );
            return 0;
        }
        public SASSiDataType ldiv( SASSiDataType b ) {
            var /= intValue(b);
            return this;
        }
        public SASSiDataType le( SASSiDataType b ) {
            if ( b instanceof SASSiInt )
                return new SASSiBool( var <= intValue(b) );
            return b.ge( this );
        }
    public SASSiDataType lfracts( SASSiDataType b ) {
         if ( b instanceof SASSiInt )
           return new SASSiInt( var / intValue(b) );
        return new SASSiDouble( var / SASSiDouble.doubleValue(b) );
    }
        public SASSiDataType lt( SASSiDataType b ) {
            if ( b instanceof SASSiInt )
                return new SASSiBool( var < intValue(b) );
            return b.gt( this );
        }
        public SASSiDataType minus( SASSiDataType b ) {
            if ( b instanceof SASSiInt )
                return new SASSiInt( var - intValue(b) );
            return new SASSiDouble( var - SASSiDouble.doubleValue(b)
    );
        }
        public SASSiDataType mul( SASSiDataType b ) {
```

```
        var *= intValue( b );
        return this;
    }
    public SASSiDataType ne( SASSiDataType b ) {
        if ( b instanceof SASSiInt )
            return new SASSiBool( var != intValue(b) );
        return b.ne( this );
    }
    public SASSiDataType plus( SASSiDataType b ) {
        if ( b instanceof SASSiInt )
            return new SASSiInt( var + intValue(b) );
        return new SASSiDouble( var + SASSiDouble.doubleValue(b)
);
    }
    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Integer.toString( var ) );
    }
    public SASSiDataType rdiv( SASSiDataType b ) {
        return ldiv( b );
    }
    public SASSiDataType rfracts( SASSiDataType b ) {
        return lfracts( b );
    }
    public SASSiDataType sub( SASSiDataType b ) {
        var -= intValue( b );
        return this;
    }
    public SASSiDataType times( SASSiDataType b ) {
        if ( b instanceof SASSiVector )
            return b.times( this );
        if ( b instanceof SASSiInt )
            return new SASSiInt( var * intValue(b) );
        return new SASSiDouble( var * SASSiDouble.doubleValue(b)
);
    }
    public String typename() {
        return "int";
    }
    public SASSiDataType uminus() {
        return new SASSiInt( -var );
    }
}
```

8.7. SASSiInterpreter.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 12:11:01)
 * @author: Xiaotang Zhang
 */
import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


class SASSiInterpreter {
    SASSiSymbolTable symt;

    final static int fc_none = 0;
    final static int fc_break = 1;
    final static int fc_continue = 2;
    final static int fc_return = 3;

    private int control = fc_none;
    private String label;

    private Random random = new Random();

    public SASSiInterpreter() {
        symt = new SASSiSymbolTable( null, null );
        registerBuiltIn();
    }
    public SASSiDataType assign( SASSiDataType a, SASSiDataType
b ) {

         if ( null != a.name )
        {
            SASSiDataType x = deepRvalue( b );
            x.setName( a.name );
            symt.setValue( x.name, x, true, 0 );
            return x;
        }

         if ( a instanceof SASSiVector )
        {
            if ( b instanceof SASSiVector )
```

```
                    ((SASSiVector)a).vect.assign(
((SASSiVector)b).vect );

            return a;
        }

        return a.error( b, "=" );
    }
    public boolean canProceed() {
        return control == fc_none;
    }
     public SASSiDataType[] convertExprList( Vector v ) {
         /* Note: expr list can be empty */
        SASSiDataType[] x = new SASSiDataType[v.size()];
        for ( int i=0; i<x.length; i++ )
            x[i] = (SASSiDataType) v.elementAt( i );
        return x;
    }
    public static String[] convertVarList( Vector v ) {
        /* Note: var list can be empty */
        String[] sv = new String[ v.size() ];
        for ( int i=0; i<sv.length; i++ )
            sv[i] = (String) v.elementAt( i );
        return sv;
    }
    public SASSiDataType deepRvalue( SASSiDataType a ) {
        if ( null == a.name )
            return a;
        if ( a instanceof SASSiVector )
            return ((SASSiVector)a).deepCopy();
        return a.copy();
    }
    public SASSiDataType execInternal( int id, SASSiDataType[]
params ) {
        return SASSiBuiltInProcedure.run( symt, id, params );
    }
    public boolean forCanProceed( SASSiDataType[] mexpr,
SASSiInt[] values ) {
        if ( control != fc_none )
            return false;

        for ( int i=mexpr.length-1; ; i-- )
        {
            if ( 0 == i )
                return false;
        }
    }
    public void forEnd( SASSiDataType[] mexpr ) {
        if ( control == fc_break )
            tryResetFlowControl( mexpr[0].name );

        // remove this symbol table
        symt = symt.dynamicParent();
    }
    public SASSiInt[] forInit( SASSiDataType[] mexpr ) {
        // very much like function call
  /*      for ( int i=0; i<mexpr.length; i++ )
            if ( ! ( mexpr[i] instanceof SASSiRange )  )
            {
                mexpr[i].error( "for: [range expected]" );
                return null;
            }
*/
        // create a new symbol table
        symt = new SASSiSymbolTable( symt, symt );

        SASSiInt[] x = new SASSiInt[mexpr.length];
   /*     for ( int i=0; i<mexpr.length; i++ )
        {
            x[i] = new SASSiInt(
((SASSiRange)mexpr[i]).range.first() );
            x[i].setName( mexpr[i].name );
            symt.setValue( mexpr[i].name, x[i], false, 0 );
        }
*/
        symt.setReadOnly();
        return x;
    }
    public void forNext( SASSiDataType[] mexpr, SASSiInt[]
values ) {

        values[ values.length-1 ].var++;
        if ( control == fc_continue )
            tryResetFlowControl( mexpr[0].name );
    }
    public static SASSiDataType getNumber( String s ) {
        if ( s.indexOf( '.' ) >= 0
             || s.indexOf( 'e' ) >= 0 || s.indexOf( 'E' ) >= 0 )
            return new SASSiDouble( Double.parseDouble( s ) );
        return new SASSiInt( Integer.parseInt( s ) );
    }
    public SASSiDataType getVariable( String s ) {
        // default static scoping
        SASSiDataType x = symt.getValue( s, true, 0 );
        if ( null == x )
            return new SASSiVariable( s );
        return x;
    }
    public SASSiDataType include( SASSiDataType file ) {
        if ( file instanceof SASSiString )
        {
            try
            {
                InputStream input =
                    (InputStream) new FileInputStream(
((SASSiString)file).var );

                SASSiLexer lexer = new SASSiLexer( input );
                SASSiParser parser = new SASSiParser( lexer );

                parser.program();
           //     if ( lexer.nr_error > 0 || parser.nr_error > 0
)
            //        throw new SASSiException( "parsing erros"
);
                CommonAST tree = (CommonAST)parser.getAST();
                SASSiWalker walker = new SASSiWalker();
                // share the symbol table
                walker.interp.symt = this.symt;
                return walker.expr( tree );
            }
            catch ( Exception e )
            {
                throw new SASSiException( "include failed" );
            }
        }

        throw new SASSiException( "why parser let another data
type go through?" );
    }
    public void loopEnd( String loop_label ) {
        if ( control == fc_break )
            tryResetFlowControl( loop_label );
    }
    public void loopNext( String loop_label ) {
        if ( control == fc_continue )
            tryResetFlowControl( loop_label );
    }
    public SASSiDataType procedureInvoke(
        SASSiWalker walker,  SASSiDataType func,
        SASSiDataType[] params ) throws
antlr.RecognitionException {

        // func must be an existing ProcedureCall
        if ( !( func instanceof SASSiProcedureCall ) )
            return func.error( "not a ProcedureCall" );

        // Is this ProcedureCall an internal ProcedureCall?
        // Names of formal args are not necessary for internal
ProcedureCalls.
        if ( ((SASSiProcedureCall)func).isInternal() )
            return execInternal(
((SASSiProcedureCall)func).getInternalId(),
                                 params );

        // otherwise check numbers of actual and formal
arguments
        String[] args = ((SASSiProcedureCall)func).getArgs();
        if ( args.length != params.length )
            return func.error( "unmatched length of parameters"
);

        // create a new symbol table
        symt = new SASSiSymbolTable(
((SASSiProcedureCall)func).getParentSymbolTable(),
                                  symt );

        // assign actual parameters to formal arguments
        for ( int i=0; i<args.length; i++ )
        {
            SASSiDataType d = rvalue( params[i] );
            d.setName( args[i] );
            symt.setValue( args[i], d, false, 0 );
        }

        // call the ProcedureCall body
        SASSiDataType r = walker.expr(
((SASSiProcedureCall)func).getBody() );

        // no break or continue can go through the ProcedureCall
        if ( control == fc_break || control == fc_continue )
            throw new SASSiException( "nowhere to break or
continue" );

        // if a return was called
        if ( control == fc_return )
```

```
            tryResetFlowControl( ((SASSiProcedureCall)func).name
);

        // remove this symbol table and return
        symt = symt.dynamicParent();

        return r;
    }
    public void procedureRegister( String name, String[] args,
AST body ) {
        symt.put( name, new SASSiProcedureCall( name, args,
body, symt ) );
    }
    public void registerBuiltIn() {
        SASSiBuiltInProcedure.register( symt );
    }
    public SASSiDataType rvalue( SASSiDataType a ) {
        if ( null == a.name )
            return a;
        return a.copy();
    }
    public void setBreak( String label ) {
        this.label = label;
        control = fc_break;
    }
    public void setContinue( String label ) {
        this.label = label;
        control = fc_continue;
    }
    public void setReturn( String label ) {
        this.label = label;
        control = fc_return;
    }
     public SASSiDataType subMatrix( SASSiDataType x,
SASSiDataType[] vexpr ) {

         if ( x instanceof SASSiVector )
        {
            if ( 1 == vexpr.length && (vexpr[0] instanceof
SASSiInt
                     || vexpr[0] instanceof SASSiDouble ))
            {
                return new SASSiDouble(

((SASSiVector)x).vect.elementAt(SASSiInt.intValue( vexpr[0])) );
            }
        }
        return x.error( "vector" );
    }
    public void tryResetFlowControl( String loop_label ) {
        if ( null == label || label.equals( loop_label ) )
            control = fc_none;
    }
}
```

8.8. SASSiLexer.java

```
package sassi;

// $ANTLR 2.7.2: "blah.g" -> "SASSiLexer.java"$

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 SASSiLexer extends antlr.CharScanner implements
SASSiTokenTypes, TokenStream
 {

  int line_error=0;

     public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
```

```
     public static final BitSet _tokenSet_1 = new
BitSet(mk_tokenSet_1());


public SASSiLexer(InputBuffer ib) {
     this(new LexerSharedInputState(ib));
}
public SASSiLexer(LexerSharedInputState state) {
     super(state);
     caseSensitiveLiterals = true;
     setCaseSensitive(true);
     literals = new Hashtable();
     literals.put(new ANTLRHashString("FALSE", this), new
Integer(55));
     literals.put(new ANTLRHashString("if", this), new
Integer(46));
     literals.put(new ANTLRHashString("for", this), new
Integer(45));
     literals.put(new ANTLRHashString("xconst", this), new
Integer(44));
     literals.put(new ANTLRHashString("False", this), new
Integer(54));
     literals.put(new ANTLRHashString("const", this), new
Integer(43));
     literals.put(new ANTLRHashString("True", this), new
Integer(52));
     literals.put(new ANTLRHashString("TRUE", this), new
Integer(53));
     literals.put(new ANTLRHashString("else", this), new
Integer(47));
     literals.put(new ANTLRHashString("include", this), new
Integer(49));
     literals.put(new ANTLRHashString("true", this), new
Integer(51));
     literals.put(new ANTLRHashString("exit", this), new
Integer(57));
     literals.put(new ANTLRHashString("procedure", this), new
Integer(50));
     literals.put(new ANTLRHashString("false", this), new
Integer(56));
     literals.put(new ANTLRHashString("return", this), new
Integer(48));
}
public SASSiLexer(InputStream in) {
     this(new ByteBuffer(in));
}
public SASSiLexer(Reader in) {
     this(new CharBuffer(in));
}
     protected final void mALPHA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = ALPHA;
          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;
          }
          case '_':
          {
               match('_');
               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;
     }
     public final void mASGN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = ASGN;
          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 mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = COMMENT;
          int _saveIndex;
          {
          if ((LA(1)=='/') && (LA(2)=='*')) {
               match("/*");
               {
               _loop14:
               do {
                    // nongreedy exit test
                    if ((LA(1)=='*') && (LA(2)=='/')) break
_loop14;
                    if ((_tokenSet_0.member(LA(1))) && ((LA(2)
>= '\u0003' && LA(2) <= '\u00ff'))) {
                         {
                         match(_tokenSet_0);
                         }
                    }
                    else if ((LA(1)=='\n'||LA(1)=='\r')) {
                         mNL(false);
                    }
                    else {
                         break _loop14;
                    }

               } while (true);
               }
               match("*/");
          }
          else if ((LA(1)=='/') && (LA(2)=='/')) {
               match("//");
               {
               _loop17:
               do {
                    if ((_tokenSet_0.member(LA(1)))) {
                         {
                         match(_tokenSet_0);
                         }
                    }
                    else {
                         break _loop17;
                    }

               } while (true);
               }
               mNL(false);
          }
          else {
```

```
			throw new
NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
		}

		}
		if ( inputState.guessing==0 ) {
			_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 mDEC(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
		int _ttype; Token _token=null; int
_begin=text.length();
		_ttype = DEC;
		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 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 mEQ(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = EQ;
          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 mGE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = GE;
          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 mGT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = GT;
          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 mID(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = ID;
          int _saveIndex;

          mALPHA(false);
          {
          _loop41:
          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 '_':  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':
               {
                    mALPHA(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;
               }
```

```
                default:
                {
                     break _loop41;
                }
                }
          } 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 mINC(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = INC;
          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;
     }
     private static final long[] mk_tokenSet_0() {
          long[] data = new long[8];
          data[0]=-9224L;
          for (int i = 1; i<=3; i++) { data[i]=-1L; }
          return data;
     }
     private static final long[] mk_tokenSet_1() {
          long[] data = new long[8];
          data[0]=-17179870216L;
          for (int i = 1; i<=3; i++) { data[i]=-1L; }
          return data;
     }
     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 mLBRK(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = LBRK;
          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 mLE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = LE;
          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 mLPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = LPAREN;
          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 mLT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = LT;
          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 mMULT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = MULT;
          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 mNL(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
		int _ttype; Token _token=null; int
_begin=text.length();
		_ttype = NL;
		int _saveIndex;

		{
		boolean synPredMatched9 = false;
		if (((LA(1)=='\r') && (LA(2)=='\n'))) {
			int _m9 = mark();
			synPredMatched9 = true;
			inputState.guessing++;
			try {
				{
				match('\r');
				match('\n');
				}
			}
			catch (RecognitionException pe) {
				synPredMatched9 = false;
			}
			rewind(_m9);
			inputState.guessing--;
		}
		if ( synPredMatched9 ) {
			match('\r');
			match('\n');
		}
		else if ((LA(1)=='\n')) {
			match('\n');
		}
		else if ((LA(1)=='\r') && (true)) {
			match('\r');
		}
		else {
			throw new
NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
		}

		}
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP; newline();
		}
		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 mNUMBER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
          int _ttype; Token _token=null; int
_begin=text.length();
          _ttype = NUMBER;
          int _saveIndex;

          {
          int _cnt44=0;
          _loop44:
          do {
               if (((LA(1) >= '0' && LA(1) <= '9'))) {
                    mDIGIT(false);
               }
               else {
                    if ( _cnt44>=1 ) { break _loop44; } else
{throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
               }

               _cnt44++;
          } while (true);
          }
          {
          if ((LA(1)=='.')) {
               match('.');
               {
               _loop47:
               do {
                    if (((LA(1) >= '0' && LA(1) <= '9'))) {
                         mDIGIT(false);
                    }
                    else {
                         break _loop47;
                    }

               } while (true);
               }
          }
          else {
          }
          }
          {
          if ((LA(1)=='E'||LA(1)=='e')) {
               {
               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;
               }
               case '0':  case '1':  case '2':  case '3':
               case '4':  case '5':  case '6':  case '7':
               case '8':  case '9':
               {
                    break;
               }
               default:
               {
                    throw new
NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
```

```
                }
                }
                }
                {
                int _cnt52=0;
                _loop52:
                do {
                     if (((LA(1) >= '0' && LA(1) <= '9'))) {
                          mDIGIT(false);
                     }
                     else {
                          if ( _cnt52>=1 ) { break _loop52; }
else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
                     }

                     _cnt52++;
                } while (true);
                }
           }
           else {
           }

           }
           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 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 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 mRBRK(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
		int _ttype; Token _token=null; int
_begin=text.length();
		_ttype = RBRK;
		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 mRDV(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
		int _ttype; Token _token=null; int
_begin=text.length();
		_ttype = RDV;
		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 mRPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
            int _ttype; Token _token=null; int
_begin=text.length();
            _ttype = RPAREN;
            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 mSTRING(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException
{
            int _ttype; Token _token=null; int
_begin=text.length();
            _ttype = STRING;
            int _saveIndex;
```

```
          _saveIndex=text.length();
          match('"');
          text.setLength(_saveIndex);
          {
          _loop57:
          do {
               if ((LA(1)=='"') && (LA(2)=='"')) {
                    {
                    _saveIndex=text.length();
                    match('"');
                    text.setLength(_saveIndex);
                    match('"');
                    }
               }
               else if ((_tokenSet_1.member(LA(1)))) {
                    {
                    match(_tokenSet_1);
                    }
               }
               else {
                    break _loop57;
               }

          } while (true);
          }
          _saveIndex=text.length();
          match('"');
          text.setLength(_saveIndex);
          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;

          {
          int _cnt5=0;
          _loop5:
          do {
               switch ( LA(1)) {
               case ' ':
               {
                    match(' ');
                    break;
               }
               case '\t':
               {
                    match('\t');
                    break;
               }
               default:
               {
                    if ( _cnt5>=1 ) { break _loop5; } else
{throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
               }
               }
               _cnt5++;
          } while (true);
          }
          if ( inputState.guessing==0 ) {
               _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 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 '\t':  case ' ':
                    {
                         mWS(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case '\n':  case '\r':
                    {
                         mNL(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case '(':
                    {
                         mLPAREN(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case ')':
                    {
                         mRPAREN(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case '{':
                    {
                         mLBRACE(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case '}':
                    {
                         mRBRACE(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case '[':
                    {
                         mLBRK(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case ']':
                    {
                         mRBRK(true);
                         theRetToken=_returnToken;
                         break;
                    }
                    case ',':
                    {
```

```
          mCOMMA(true);
          theRetToken=_returnToken;
          break;
     }
     case '*':
     {
          mMULT(true);
          theRetToken=_returnToken;
          break;
     }
     case ';':
     {
          mSEMI(true);
          theRetToken=_returnToken;
          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':  case '_':  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':
     {
          mID(true);
          theRetToken=_returnToken;
          break;
     }
     case '0':  case '1':  case '2':  case '3':
     case '4':  case '5':  case '6':  case '7':
     case '8':  case '9':
     {
          mNUMBER(true);
          theRetToken=_returnToken;
          break;
     }
     case '"':
     {
          mSTRING(true);
          theRetToken=_returnToken;
```

```
                              break;
                         }
                         default:
                              if ((LA(1)=='/') &&
(LA(2)=='*'||LA(2)=='/')) {
                                   mCOMMENT(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='>') && (LA(2)=='='))
{
                                   mGE(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='<') && (LA(2)=='='))
{
                                   mLE(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='=') && (LA(2)=='='))
{
                                   mEQ(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='<') && (LA(2)=='>'))
{
                                   mNEQ(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='+') && (LA(2)=='+'))
{
                                   mINC(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='-') && (LA(2)=='-'))
{
                                   mDEC(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='=') && (true)) {
                                   mASGN(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='+') && (true)) {
                                   mPLUS(true);
                                   theRetToken=_returnToken;
                              }
                              else if ((LA(1)=='-') && (true)) {
```

```
                              mMINUS(true);
                              theRetToken=_returnToken;
                         }
                         else if ((LA(1)=='/') && (true)) {
                              mRDV(true);
                              theRetToken=_returnToken;
                         }
                         else if ((LA(1)=='>') && (true)) {
                              mGT(true);
                              theRetToken=_returnToken;
                         }
                         else if ((LA(1)=='<') && (true)) {
                              mLT(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 void reportError(RecognitionException excep)
  {
   super.reportError(excep);
   line_error++;
  }
  public void reportError(String str)
  {
   super.reportError(str);
   line_error++;
  }
}
```

8.9. SASSiMain.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-6 11:11:22)
 * @author: Xiaotang Zhang
 */
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;


class SASSiMain {
    static boolean verbose = false;

    public static void commandLine() {
        InputStream input = (InputStream) new DataInputStream(
System.in );
        SASSiWalker walker = new SASSiWalker();

        for ( ;; )
        {
            try
            {
                while( input.available() > 0 )
                    input.read();
            }
            catch ( IOException e ) {}

            System.out.print( "SASSi> " );
            System.out.flush();
            SASSiLexer lexer = new SASSiLexer( input );
            SASSiParser parser = new SASSiParser( lexer );

            try
            {
                parser.cl_statement();
                CommonAST tree = (CommonAST)parser.getAST();
                SASSiDataType r = walker.expr( tree );
                if ( verbose && r != null)
                    r.print();
            } catch( RecognitionException e ) {
                System.err.println( "Recognition exception: " +
e );
            } catch( TokenStreamException e ) {
                if ( e instanceof TokenStreamIOException ) {
                    System.err.println( "Token I/O exception" );
                    break;
                }
                System.err.println( "Error: Token stream: " + e
);
            } catch( SASSiException e ) {
                System.err.println( "Error: Interpretive: " + e
);
                e.printStackTrace();
            } catch( RuntimeException e ) {
                System.err.println( "Error: Runtime: " + e );
                e.printStackTrace();
            } catch( Exception e ) {
                System.err.println( "Error: " + e );
                e.printStackTrace();
            }

        }
    }
    public static void execFile( String filename ) {
        try
        {
            InputStream input = ( null != filename ) ?
                (InputStream) new FileInputStream( filename ) :
                (InputStream) System.in;

            SASSiLexer lexer = new SASSiLexer( input );

            SASSiParser parser = new SASSiParser( lexer );

            // Parse the input program
            parser.program();

            if ( lexer.line_error > 0 || parser.line_error > 0 )
            {
                System.err.println( "Parsing errors found.
Stop." );
                return;
            }

            CommonAST tree = (CommonAST)parser.getAST();

            if ( verbose )
            {
                // Print the resulting tree out in LISP notation
                System.out.println(
                    "================ tree structure
======================" );
                System.out.println( tree.toStringList() );
            }

            SASSiWalker walker = new SASSiWalker();
            // Traverse the tree created by the parser

            if ( verbose )
                System.out.println(
                    "================ program output
======================" );

            SASSiDataType r = walker.expr( tree );

            if ( verbose )
                System.out.println(
                    "================ program return
======================" );
            if ( null != r )
                r.print();

            if ( verbose )
            {
                System.out.println(
                    "================ global variables
====================" );
                walker.interp.symt.what();
            }

        } catch( IOException e ) {
            System.err.println( "Error: I/O: " + e );
        } catch( RecognitionException e ) {
            System.err.println( "Error: Recognition: " + e );
        } catch( TokenStreamException e ) {
            System.err.println( "Error: Token stream: " + e );
        } catch( Exception e ) {
            System.err.println( "Error: " + e );
        }

   /*     while ( Painter.frameCount() > 0 ||
Plotter.frameCount() > 0 )
        {
            try
            {
                Thread.sleep( 1000 );
            } catch ( InterruptedException e ) {}
        }*/
    }
    public static void main( String[] args ) {

        verbose = args.length >= 1 && args[0].equals( "-v" );

        boolean batch = args.length >=1 && args[0].equals( "-b"
);

        if ( args.length >= 1 && args[args.length-1].charAt(0)
!= '-' )
            execFile( args[args.length-1] );
        else if ( batch )
            execFile( null );
        else
            commandLine();

        System.exit( 0 );
    }
}
```

8.10. SASSiParser.java

```
package sassi;

// $ANTLR 2.7.2: "blah.g" -> "SASSiParser.java"$

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     SASSiParser     extends     antlr.LLkParser
implements SASSiTokenTypes
 {

  int line_error=0;

     public static final String[] _tokenNames = {
          "<0>",
          "EOF",
          "<2>",
          "NULL_TREE_LOOKAHEAD",
          "ALPHA",
          "DIGIT",
          "WS",
          "NL",
          "COMMENT",
          "LPAREN",
          "RPAREN",
          "LBRACE",
          "RBRACE",
          "LBRK",
          "RBRK",
          "ASGN",
          "COMMA",
          "MULT",
          "PLUS",
          "MINUS",
          "RDV",
          "GE",
          "LE",
          "GT",
          "LT",
          "EQ",
          "NEQ",
          "INC",
          "DEC",
          "SEMI",
          "ID",
          "NUMBER",
          "STRING",
          "STATEMENT",
          "FOR_LOOP",
          "VAR_LIST",
          "VECTOR",
          "EXPRESSION_LIST",
          "CONSTANT_STATEMENT",
          "PROCEDURE_CALL",
          "FOR_CON",
          "UPLUS",
          "UMINUS",
          "\"const\"",
          "\"xconst\"",
          "\"for\"",
          "\"if\"",
          "\"else\"",
          "\"return\"",
          "\"include\"",
          "\"procedure\"",
          "\"true\"",
          "\"True\"",
          "\"TRUE\"",
          "\"False\"",
          "\"FALSE\"",
          "\"false\"",
          "\"exit\""
     };

     public    static    final    BitSet    _tokenSet_0    =    new
BitSet(mk_tokenSet_0());
     public    static    final    BitSet    _tokenSet_1    =    new
BitSet(mk_tokenSet_1());
     public    static    final    BitSet    _tokenSet_2    =    new
BitSet(mk_tokenSet_2());
     public    static    final    BitSet    _tokenSet_3    =    new
BitSet(mk_tokenSet_3());
     public    static    final    BitSet    _tokenSet_4    =    new
BitSet(mk_tokenSet_4());
     public    static    final    BitSet    _tokenSet_5    =    new
BitSet(mk_tokenSet_5());
```

```
     public    static    final    BitSet    _tokenSet_6    =    new
BitSet(mk_tokenSet_6());
     public    static    final    BitSet    _tokenSet_7    =    new
BitSet(mk_tokenSet_7());
     public    static    final    BitSet    _tokenSet_8    =    new
BitSet(mk_tokenSet_8());
     public    static    final    BitSet    _tokenSet_9    =    new
BitSet(mk_tokenSet_9());
     public    static    final    BitSet    _tokenSet_10    =    new
BitSet(mk_tokenSet_10());
     public    static    final    BitSet    _tokenSet_11    =    new
BitSet(mk_tokenSet_11());
     public    static    final    BitSet    _tokenSet_12    =    new
BitSet(mk_tokenSet_12());
     public    static    final    BitSet    _tokenSet_13    =    new
BitSet(mk_tokenSet_13());
     public    static    final    BitSet    _tokenSet_14    =    new
BitSet(mk_tokenSet_14());
     public    static    final    BitSet    _tokenSet_15    =    new
BitSet(mk_tokenSet_15());
     public    static    final    BitSet    _tokenSet_16    =    new
BitSet(mk_tokenSet_16());
     public    static    final    BitSet    _tokenSet_17    =    new
BitSet(mk_tokenSet_17());


public SASSiParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
public SASSiParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}
protected SASSiParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
public SASSiParser(TokenStream lexer) {
  this(lexer,2);
}
protected SASSiParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
     public     final     void     arith_expression()     throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               arith_term();
               astFactory.addASTChild(currentAST, returnAST);
               {
               _loop101:
               do {
                    if ((LA(1)==PLUS||LA(1)==MINUS)) {
                         {
                         switch ( LA(1)) {
                         case PLUS:
                         {
                              AST tmp40_AST = null;
                              tmp40_AST                          =
astFactory.create(LT(1));
                              astFactory.makeASTRoot(currentAST,
tmp40_AST);
                              match(PLUS);
                              break;
                         }
                         case MINUS:
                         {
                              AST tmp41_AST = null;
                              tmp41_AST                          =
astFactory.create(LT(1));
                              astFactory.makeASTRoot(currentAST,
tmp41_AST);
                              match(MINUS);
                              break;
                         }
                         default:
                         {
                              throw                           new
NoViableAltException(LT(1), getFilename());
                         }
                         }
                         }
                         arith_term();
```

```
                              astFactory.addASTChild(currentAST,
returnAST);
                         }
                         else {
                              break _loop101;
                         }
                    
                    } while (true);
                    }
                    arith_expression_AST = (AST)currentAST.root;
               }
               catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_13);
               }
               returnAST = arith_expression_AST;
          }
          public      final      void      arith_factor()      throws
RecognitionException, TokenStreamException {
               
               returnAST = null;
               ASTPair currentAST = new ASTPair();
               AST arith_factor_AST = null;
               
               try {      // for error handling
                    switch ( LA(1)) {
                    case PLUS:
                    {
                         match(PLUS);
                         r_value();
                         astFactory.addASTChild(currentAST,
returnAST);
                         arith_factor_AST = (AST)currentAST.root;
                         arith_factor_AST    =    (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(UPLUS,"UPLUS")).add(arith_fac
tor_AST));
                         currentAST.root = arith_factor_AST;
                         currentAST.child   =   arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                              arith_factor_AST.getFirstChild()       :
arith_factor_AST;
                         currentAST.advanceChildToEnd();
                         arith_factor_AST = (AST)currentAST.root;
                         break;
                    }
               case MINUS:
               {
                    match(MINUS);
                    r_value();
                    astFactory.addASTChild(currentAST,
returnAST);
                    arith_factor_AST = (AST)currentAST.root;
                    arith_factor_AST    =    (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(UMINUS,"UMINUS")).add(arith_f
actor_AST));
                    currentAST.root = arith_factor_AST;
                    currentAST.child   =   arith_factor_AST!=null
&&arith_factor_AST.getFirstChild()!=null ?
                         arith_factor_AST.getFirstChild()       :
arith_factor_AST;
                    currentAST.advanceChildToEnd();
                    arith_factor_AST = (AST)currentAST.root;
                    break;
               }
               case LPAREN:
               case LBRK:
               case ID:
               case NUMBER:
               case STRING:
               case LITERAL_true:
               case LITERAL_True:
               case LITERAL_TRUE:
               case LITERAL_False:
               case LITERAL_FALSE:
               case LITERAL_false:
               {
                    r_value();
                    astFactory.addASTChild(currentAST,
returnAST);
                    arith_factor_AST = (AST)currentAST.root;
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
          }
          catch (RecognitionException ex) {
               reportError(ex);
```

```
                consume();
                consumeUntil(_tokenSet_14);
            }
            returnAST = arith_factor_AST;
        }
        public final void arith_term() throws RecognitionException,
TokenStreamException {

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

            try {      // for error handling
                arith_factor();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop105:
                do {
                    if ((LA(1)==MULT||LA(1)==RDV)) {
                        {
                        switch ( LA(1)) {
                        case MULT:
                        {
                            AST tmp51_AST = null;
                            tmp51_AST                          =
astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST,
tmp51_AST);
                            match(MULT);
                            break;
                        }
                        case RDV:
                        {
                            AST tmp52_AST = null;
                            tmp52_AST                          =
astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST,
tmp52_AST);
                            match(RDV);
                            break;
                        }
                        default:
                        {
                            throw                            new
NoViableAltException(LT(1), getFilename());
                        }
                        }
```

```
                         }
                         arith_factor();
                         astFactory.addASTChild(currentAST,
returnAST);
                    }
                    else {
                         break _loop105;
                    }

               } while (true);
               }
               arith_term_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_15);
          }
          returnAST = arith_term_AST;
     }
     public final void assignment() throws RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               l_value();
               astFactory.addASTChild(currentAST, returnAST);
               AST tmp10_AST = null;
               tmp10_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp10_AST);
               match(ASGN);
               arith_expression();
               astFactory.addASTChild(currentAST, returnAST);
               match(SEMI);
               assignment_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_6);
          }
          returnAST = assignment_AST;
     }
     protected void buildTokenTypeASTClassMap() {
          tokenTypeToASTClassMap=null;
     }
     public      final      void      cl_statement()      throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               switch ( LA(1)) {
               case LBRACE:
               case ID:
               case LITERAL_for:
               case LITERAL_if:
               case LITERAL_return:
               case LITERAL_include:
               case LITERAL_procedure:
               {
                    {
                    switch ( LA(1)) {
                    case LBRACE:
                    case ID:
                    case LITERAL_for:
                    case LITERAL_if:
                    case LITERAL_return:
                    case LITERAL_include:
                    {
                         statement();
                         astFactory.addASTChild(currentAST,
returnAST);
                         break;
                    }
                    case LITERAL_procedure:
                    {
                         procedure_def();
                         astFactory.addASTChild(currentAST,
returnAST);
                         break;
                    }
                    default:
                    {
                         throw   new   NoViableAltException(LT(1),
getFilename());
                    }
                    }
                    }
```

```
                         cl_statement_AST = (AST)currentAST.root;
                         break;
                    }
                    case LITERAL_exit:
                    {
                         AST tmp68_AST = null;
                         tmp68_AST = astFactory.create(LT(1));
                         astFactory.addASTChild(currentAST,
tmp68_AST);
                         match(LITERAL_exit);
                         System.exit(0);
                         cl_statement_AST = (AST)currentAST.root;
                         break;
                    }
                    case EOF:
                    {
                         match(Token.EOF_TYPE);
                         System.exit(0);
                         cl_statement_AST = (AST)currentAST.root;
                         break;
                    }
                    default:
                    {
                         throw     new     NoViableAltException(LT(1),
getFilename());
                    }
                    }
               }
               catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_0);
               }
               returnAST = cl_statement_AST;
          }
          public      final      void      constant_group()      throws
RecognitionException, TokenStreamException {

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

               try {      // for error handling
                    AST tmp2_AST = null;
                    tmp2_AST = astFactory.create(LT(1));
                    astFactory.makeASTRoot(currentAST, tmp2_AST);
                    match(LITERAL_const);
```

```
                {
                _loop64:
                do {
                     if ((LA(1)==ID)) {
                          constant_statement();
                          astFactory.addASTChild(currentAST,
returnAST);
                     }
                     else {
                          break _loop64;
                     }

                } while (true);
                }
                AST tmp3_AST = null;
                tmp3_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp3_AST);
                match(LITERAL_xconst);
                constant_group_AST = (AST)currentAST.root;
           }
           catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_1);
           }
           returnAST = constant_group_AST;
     }
     public     final     void     constant_statement()     throws
RecognitionException, TokenStreamException {

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

           try {      // for error handling
                assignment();
                astFactory.addASTChild(currentAST, returnAST);
                constant_statement_AST = (AST)currentAST.root;
                constant_statement_AST   =   (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(CONSTANT_STATEMENT,"CONSTANT_
STATEMENT")).add(constant_statement_AST));
                currentAST.root = constant_statement_AST;
                currentAST.child   =   constant_statement_AST!=null
&&constant_statement_AST.getFirstChild()!=null ?
                     constant_statement_AST.getFirstChild()      :
constant_statement_AST;
```

```
                currentAST.advanceChildToEnd();
                constant_statement_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_5);
            }
            returnAST = constant_statement_AST;
        }
        public final void expression() throws RecognitionException,
    TokenStreamException {

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

            try {      // for error handling
                relat_expression();
                astFactory.addASTChild(currentAST, returnAST);
                expression_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_11);
            }
            returnAST = expression_AST;
        }
        public     final     void     expression_list()     throws
    RecognitionException, TokenStreamException {

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

            try {      // for error handling
                switch ( LA(1)) {
                case LPAREN:
                case LBRK:
                case PLUS:
                case MINUS:
                case ID:
                case NUMBER:
                case STRING:
                case LITERAL_true:
                case LITERAL_True:
```

```
               case LITERAL_TRUE:
               case LITERAL_False:
               case LITERAL_FALSE:
               case LITERAL_false:
               {
                    expression();
                    astFactory.addASTChild(currentAST,
returnAST);
                    {
                    _loop86:
                    do {
                         if ((LA(1)==COMMA)) {
                              match(COMMA);
                              expression();
                              astFactory.addASTChild(currentAST,
returnAST);
                         }
                         else {
                              break _loop86;
                         }

                    } while (true);
                    }
                    expression_list_AST = (AST)currentAST.root;
                    expression_list_AST  =  (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(EXPRESSION_LIST,"EXPRESSION_L
IST")).add(expression_list_AST));
                    currentAST.root = expression_list_AST;
                    currentAST.child  =  expression_list_AST!=null
&&expression_list_AST.getFirstChild()!=null ?
                         expression_list_AST.getFirstChild()    :
expression_list_AST;
                    currentAST.advanceChildToEnd();
                    expression_list_AST = (AST)currentAST.root;
                    break;
               }
               case RPAREN:
               {
                    expression_list_AST = (AST)currentAST.root;
                    expression_list_AST  =  (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(EXPRESSION_LIST,"EXPRESSION_L
IST")).add(expression_list_AST));
                    currentAST.root = expression_list_AST;
                    currentAST.child  =  expression_list_AST!=null
&&expression_list_AST.getFirstChild()!=null ?
```

```
                         expression_list_AST.getFirstChild()    :
expression_list_AST;
                    currentAST.advanceChildToEnd();
                    expression_list_AST = (AST)currentAST.root;
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_8);
          }
          returnAST = expression_list_AST;
     }
     public  final  void  for_con()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               loop_init();
               astFactory.addASTChild(currentAST, returnAST);
               loop_cond();
               astFactory.addASTChild(currentAST, returnAST);
               loop_incr();
               astFactory.addASTChild(currentAST, returnAST);
               for_con_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_8);
          }
          returnAST = for_con_AST;
     }
     public  final  void  for_loop()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               AST tmp12_AST = null;
               tmp12_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp12_AST);
               match(LITERAL_for);
               match(LPAREN);
               for_con();
               astFactory.addASTChild(currentAST, returnAST);
               match(RPAREN);
               statement();
               astFactory.addASTChild(currentAST, returnAST);
               for_loop_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_3);
          }
          returnAST = for_loop_AST;
     }
     public      final      void      if_statement()      throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               AST tmp15_AST = null;
               tmp15_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp15_AST);
               match(LITERAL_if);
               match(LPAREN);
               expression();
               astFactory.addASTChild(currentAST, returnAST);
               match(RPAREN);
               statement();
               astFactory.addASTChild(currentAST, returnAST);
               {
               if           ((LA(1)==LITERAL_else)            &&
(_tokenSet_2.member(LA(2)))) {
                    match(LITERAL_else);
                    statement();
                    astFactory.addASTChild(currentAST,
returnAST);
               }
               else    if    ((_tokenSet_3.member(LA(1)))    &&
(_tokenSet_7.member(LA(2)))) {
               }
               else {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }

               }
               if_statement_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_3);
          }
          returnAST = if_statement_AST;
     }
     public  final  void  index()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               arith_expression();
               astFactory.addASTChild(currentAST, returnAST);
               index_AST = (AST)currentAST.root;
               index_AST     =     (AST)astFactory.make(     (new
ASTArray(2)).add(astFactory.create(EXPRESSION_LIST,"INDEX")).add
(index_AST));
               currentAST.root = index_AST;
               currentAST.child        =        index_AST!=null
&&index_AST.getFirstChild()!=null ?
                    index_AST.getFirstChild() : index_AST;
               currentAST.advanceChildToEnd();
               index_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_17);
          }
          returnAST = index_AST;
     }
     public  final  void  l_value()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               AST tmp37_AST = null;
               tmp37_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp37_AST);
               match(ID);
               {
               _loop110:
               do {
                    if ((LA(1)==LBRK)) {
                         match(LBRK);
                         index();
                         astFactory.addASTChild(currentAST,
returnAST);
                         match(RBRK);
                    }
                    else {
                         break _loop110;
                    }

               } while (true);
               }
               l_value_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_12);
          }
          returnAST = l_value_AST;
     }
     public     final     void     load_statement()     throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
```

```
                AST tmp21_AST = null;
                tmp21_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp21_AST);
                match(LITERAL_include);
                match(LT);
                AST tmp23_AST = null;
                tmp23_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp23_AST);
                match(STRING);
                match(GT);
                match(SEMI);
                load_statement_AST = (AST)currentAST.root;
           }
           catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
           }
           returnAST = load_statement_AST;
      }
      public  final  void  loop_cond()  throws  RecognitionException,
TokenStreamException {

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

           try {      // for error handling
                switch ( LA(1)) {
                case SEMI:
                {
                     match(SEMI);
                     loop_cond_AST = (AST)currentAST.root;
                     loop_cond_AST  =  (AST)astFactory.make(  (new
ASTArray(1)).add(astFactory.create(null,"null_cond")));
                     currentAST.root = loop_cond_AST;
                     currentAST.child    =    loop_cond_AST!=null
&&loop_cond_AST.getFirstChild()!=null ?
                          loop_cond_AST.getFirstChild()          :
loop_cond_AST;
                     currentAST.advanceChildToEnd();
                     loop_cond_AST = (AST)currentAST.root;
                     break;
                }
                case LPAREN:
                case LBRK:
                case PLUS:
```

```
                case MINUS:
                case ID:
                case NUMBER:
                case STRING:
                case LITERAL_true:
                case LITERAL_True:
                case LITERAL_TRUE:
                case LITERAL_False:
                case LITERAL_FALSE:
                case LITERAL_false:
                {
                     expression();
                     astFactory.addASTChild(currentAST,
returnAST);
                     match(SEMI);
                     loop_cond_AST = (AST)currentAST.root;
                     break;
                }
                default:
                {
                     throw     new     NoViableAltException(LT(1),
getFilename());
                }
                }
           }
           catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_10);
           }
           returnAST = loop_cond_AST;
     }
     public final void loop_incr() throws RecognitionException,
TokenStreamException {

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

           try {      // for error handling
                switch ( LA(1)) {
                case RPAREN:
                {
                     {
                     }
                     loop_incr_AST = (AST)currentAST.root;
```

```
                    loop_incr_AST  =  (AST)astFactory.make(  (new
ASTArray(1)).add(astFactory.create(null,"null_incr")));
                    currentAST.root = loop_incr_AST;
                    currentAST.child     =     loop_incr_AST!=null
&&loop_incr_AST.getFirstChild()!=null ?
                         loop_incr_AST.getFirstChild()          :
loop_incr_AST;
                    currentAST.advanceChildToEnd();
                    loop_incr_AST = (AST)currentAST.root;
                    break;
               }
               case ID:
               {
                    assignment();
                    astFactory.addASTChild(currentAST,
returnAST);
                    loop_incr_AST = (AST)currentAST.root;
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_8);
          }
          returnAST = loop_incr_AST;
     }
     public  final  void  loop_init()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               switch ( LA(1)) {
               case SEMI:
               {
                    match(SEMI);
                    loop_init_AST = (AST)currentAST.root;
                    loop_init_AST  =  (AST)astFactory.make(  (new
ASTArray(1)).add(astFactory.create(null,"null_init")));
                    currentAST.root = loop_init_AST;
                    currentAST.child    =    loop_init_AST!=null
&&loop_init_AST.getFirstChild()!=null ?
                         loop_init_AST.getFirstChild()         :
loop_init_AST;
                    currentAST.advanceChildToEnd();
                    loop_init_AST = (AST)currentAST.root;
                    break;
               }
               case ID:
               {
                    assignment();
                    astFactory.addASTChild(currentAST,
returnAST);
                    match(SEMI);
                    loop_init_AST = (AST)currentAST.root;
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_9);
          }
          returnAST = loop_init_AST;
     }
     private static final long[] mk_tokenSet_0() {
          long[] data = { 2L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_1() {
          long[] data = { 207587902698905 8L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_10() {
          long[] data = { 1073742848L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_11() {
          long[] data = { 536937472L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_12() {
          long[] data = { 671073280L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_13() {
          long[] data = { 669074432L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_14() {
          long[] data = { 671040512L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_15() {
          long[] data = { 669860864L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_16() {
          long[] data = { 671048704L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_17() {
          long[] data = { 16384L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_2() {
          long[] data = { 949979120142336L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_3() {
          long[] data = { 2216616515344386L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_4() {
          long[] data = { 2075879026984962L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_5() {
          long[] data = { 17593259786240L, 0L};
          return data;
     }
     private static final long[] mk_tokenSet_6() {
          long[] data = { 2234209238260738L, 0L};
          return data;
     }
```

```
	private static final long[] mk_tokenSet_7() {
		long[] data = { 144080011774441986L, 0L};
		return data;
	}
	private static final long[] mk_tokenSet_8() {
		long[] data = { 1024L, 0L};
		return data;
	}
	private static final long[] mk_tokenSet_9() {
		long[] data = { 141863396316029440L, 0L};
		return data;
	}
	public      final      void      procedure_body()      throws
RecognitionException, TokenStreamException {

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

		try {      // for error handling
			match(LBRACE);
			{
			switch ( LA(1)) {
			case LITERAL_const:
			{
				constant_group();
				astFactory.addASTChild(currentAST,
returnAST);
				break;
			}
			case LBRACE:
			case RBRACE:
			case ID:
			case LITERAL_for:
			case LITERAL_if:
			case LITERAL_return:
			case LITERAL_include:
			{
				break;
			}
			default:
			{
				throw     new     NoViableAltException(LT(1),
getFilename());
			}
			}
			}
```

```
               {
               _loop94:
               do {
                    if ((_tokenSet_2.member(LA(1)))) {
                         statement();
                         astFactory.addASTChild(currentAST,
returnAST);
                    }
                    else {
                         break _loop94;
                    }

               } while (true);
               }
               match(RBRACE);
               procedure_body_AST = (AST)currentAST.root;
               procedure_body_AST  =  (AST)astFactory.make(  (new
ASTArray(2)).add(astFactory.create(STATEMENT,"PROCEDURE_BODY")).
add(procedure_body_AST));
               currentAST.root = procedure_body_AST;
               currentAST.child     =     procedure_body_AST!=null
&&procedure_body_AST.getFirstChild()!=null ?
                    procedure_body_AST.getFirstChild()          :
procedure_body_AST;
               currentAST.advanceChildToEnd();
               procedure_body_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_4);
          }
          returnAST = procedure_body_AST;
     }
     public      final      void      procedure_call()      throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               AST tmp42_AST = null;
               tmp42_AST = astFactory.create(LT(1));
               astFactory.addASTChild(currentAST, tmp42_AST);
               match(ID);
               match(LPAREN);
               expression_list();
               astFactory.addASTChild(currentAST, returnAST);
               match(RPAREN);
               procedure_call_AST = (AST)currentAST.root;
               procedure_call_AST  =  (AST)astFactory.make(  (new
ASTArray(2)).add(astFactory.create(PROCEDURE_CALL,"PROCEDURE_CAL
L")).add(procedure_call_AST));
               currentAST.root = procedure_call_AST;
               currentAST.child     =     procedure_call_AST!=null
&&procedure_call_AST.getFirstChild()!=null ?
                    procedure_call_AST.getFirstChild()          :
procedure_call_AST;
               currentAST.advanceChildToEnd();
               procedure_call_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_14);
          }
          returnAST = procedure_call_AST;
     }
     public   final   void   procedure_call_statement()   throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               procedure_call();
               astFactory.addASTChild(currentAST, returnAST);
               match(SEMI);
               procedure_call_statement_AST                     =
(AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_3);
          }
          returnAST = procedure_call_statement_AST;
     }
     public      final      void      procedure_def()      throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               AST tmp6_AST = null;
               tmp6_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp6_AST);
               match(LITERAL_procedure);
               AST tmp7_AST = null;
               tmp7_AST = astFactory.create(LT(1));
               astFactory.addASTChild(currentAST, tmp7_AST);
               match(ID);
               match(LPAREN);
               var_list();
               astFactory.addASTChild(currentAST, returnAST);
               match(RPAREN);
               procedure_body();
               astFactory.addASTChild(currentAST, returnAST);
               procedure_def_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_4);
          }
          returnAST = procedure_def_AST;
     }
     public  final  void  program()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               {
               switch ( LA(1)) {
               case LITERAL_const:
               {
                    constant_group();
                    astFactory.addASTChild(currentAST,
returnAST);
                    break;
               }
               case EOF:
               case LBRACE:
               case ID:
               case LITERAL_for:
               case LITERAL_if:
               case LITERAL_return:
               case LITERAL_include:
               case LITERAL_procedure:
               {
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
               }
               {
               _loop61:
               do {
                    switch ( LA(1)) {
                    case LBRACE:
                    case ID:
                    case LITERAL_for:
                    case LITERAL_if:
                    case LITERAL_return:
                    case LITERAL_include:
                    {
                         statement();
                         astFactory.addASTChild(currentAST,
returnAST);
                         break;
                    }
                    case LITERAL_procedure:
                    {
                         procedure_def();
                         astFactory.addASTChild(currentAST,
returnAST);
                         break;
                    }
                    default:
                    {
                         break _loop61;
                    }
                    }
               } while (true);
               }
               match(Token.EOF_TYPE);
               program_AST = (AST)currentAST.root;
```

```
               program_AST    =    (AST)astFactory.make(    (new
ASTArray(2)).add(astFactory.create(STATEMENT,"PROG")).add(progra
m_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  r_value()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               switch ( LA(1)) {
               case NUMBER:
               {
                    AST tmp55_AST = null;
                    tmp55_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp55_AST);
                    match(NUMBER);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case STRING:
               {
                    AST tmp56_AST = null;
                    tmp56_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp56_AST);
                    match(STRING);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_true:
```

```
               {
                    AST tmp57_AST = null;
                    tmp57_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp57_AST);
                    match(LITERAL_true);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_True:
               {
                    AST tmp58_AST = null;
                    tmp58_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp58_AST);
                    match(LITERAL_True);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_TRUE:
               {
                    AST tmp59_AST = null;
                    tmp59_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp59_AST);
                    match(LITERAL_TRUE);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_False:
               {
                    AST tmp60_AST = null;
                    tmp60_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp60_AST);
                    match(LITERAL_False);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_FALSE:
               {
                    AST tmp61_AST = null;
                    tmp61_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp61_AST);
                    match(LITERAL_FALSE);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_false:
               {
                    AST tmp62_AST = null;
                    tmp62_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST,
tmp62_AST);
                    match(LITERAL_false);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LBRK:
               {
                    vector();
                    astFactory.addASTChild(currentAST,
returnAST);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               case LPAREN:
               {
                    match(LPAREN);
                    expression();
                    astFactory.addASTChild(currentAST,
returnAST);
                    match(RPAREN);
                    r_value_AST = (AST)currentAST.root;
                    break;
               }
               default:
                    if              ((LA(1)==ID)              &&
(_tokenSet_16.member(LA(2)))) {
                         l_value();
                         astFactory.addASTChild(currentAST,
returnAST);
                         r_value_AST = (AST)currentAST.root;
                    }
                    else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
                         procedure_call();
                         astFactory.addASTChild(currentAST,
returnAST);
                         r_value_AST = (AST)currentAST.root;
                    }
               else {
                    throw     new     NoViableAltException(LT(1),
getFilename());
```

```
                    }
                    }
               }
               catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_14);
               }
               returnAST = r_value_AST;
          }
          public     final     void     relat_expression()     throws
     RecognitionException, TokenStreamException {

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

               try {      // for error handling
                    arith_expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    {
                    switch ( LA(1)) {
                    case GE:
                    case LE:
                    case GT:
                    case LT:
                    case EQ:
                    case NEQ:
                    {
                         {
                         switch ( LA(1)) {
                         case GE:
                         {
                              AST tmp31_AST = null;
                              tmp31_AST = astFactory.create(LT(1));
                              astFactory.makeASTRoot(currentAST,
     tmp31_AST);
                              match(GE);
                              break;
                         }
                         case LE:
                         {
                              AST tmp32_AST = null;
                              tmp32_AST = astFactory.create(LT(1));
                              astFactory.makeASTRoot(currentAST,
     tmp32_AST);
                              match(LE);
                         break;
                    }
                    case GT:
                    {
                         AST tmp33_AST = null;
                         tmp33_AST = astFactory.create(LT(1));
                         astFactory.makeASTRoot(currentAST,
tmp33_AST);
                         match(GT);
                         break;
                    }
                    case LT:
                    {
                         AST tmp34_AST = null;
                         tmp34_AST = astFactory.create(LT(1));
                         astFactory.makeASTRoot(currentAST,
tmp34_AST);
                         match(LT);
                         break;
                    }
                    case EQ:
                    {
                         AST tmp35_AST = null;
                         tmp35_AST = astFactory.create(LT(1));
                         astFactory.makeASTRoot(currentAST,
tmp35_AST);
                         match(EQ);
                         break;
                    }
                    case NEQ:
                    {
                         AST tmp36_AST = null;
                         tmp36_AST = astFactory.create(LT(1));
                         astFactory.makeASTRoot(currentAST,
tmp36_AST);
                         match(NEQ);
                         break;
                    }
                    default:
                    {
                         throw   new   NoViableAltException(LT(1),
getFilename());
                    }
                    }
                    }
                    arith_expression();
                    astFactory.addASTChild(currentAST,
returnAST);
                    break;
               }
               case RPAREN:
               case COMMA:
               case SEMI:
               {
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
               }
               relat_expression_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_11);
          }
          returnAST = relat_expression_AST;
     }
  public void reportError(RecognitionException excep)
  {
   super.reportError(excep);
   line_error++;
  }
  public void reportError(String str)
  {
   super.reportError(str);
   line_error++;
  }
     public     final     void     return_statement()     throws
RecognitionException, TokenStreamException {

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

          try {      // for error handling
               AST tmp19_AST = null;
               tmp19_AST = astFactory.create(LT(1));
               astFactory.makeASTRoot(currentAST, tmp19_AST);
               match(LITERAL_return);
               {
               switch ( LA(1)) {
               case LPAREN:
               case LBRK:
               case PLUS:
               case MINUS:
               case ID:
               case NUMBER:
               case STRING:
               case LITERAL_true:
               case LITERAL_True:
               case LITERAL_TRUE:
               case LITERAL_False:
               case LITERAL_FALSE:
               case LITERAL_false:
               {
                    expression();
                    astFactory.addASTChild(currentAST,
returnAST);
                    break;
               }
               case SEMI:
               {
                    break;
               }
               default:
               {
                    throw     new     NoViableAltException(LT(1),
getFilename());
               }
               }
               }
               match(SEMI);
               return_statement_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_3);
          }
          returnAST = return_statement_AST;
     }
     public final void statement() throws RecognitionException,
TokenStreamException {

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

          try {      // for error handling
               switch ( LA(1)) {
               case LITERAL_for:
               {
                    for_loop();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_if:
               {
                    if_statement();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_return:
               {
                    return_statement();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
                    break;
               }
               case LITERAL_include:
               {
                    load_statement();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
                    break;
               }
               case LBRACE:
               {
                    match(LBRACE);
                    {
                    _loop68:
                    do {
                         if ((_tokenSet_2.member(LA(1)))) {
                              statement();
                              astFactory.addASTChild(currentAST,
returnAST);
```

```
                    }
                    else {
                         break _loop68;
                    }

               } while (true);
               }
               match(RBRACE);
               statement_AST = (AST)currentAST.root;
               statement_AST  =  (AST)astFactory.make(  (new
ASTArray(2)).add(astFactory.create(STATEMENT,"STATEMENT")).add(s
tatement_AST));
               currentAST.root = statement_AST;
               currentAST.child     =     statement_AST!=null
&&statement_AST.getFirstChild()!=null ?
                    statement_AST.getFirstChild()          :
statement_AST;
               currentAST.advanceChildToEnd();
               statement_AST = (AST)currentAST.root;
               break;
          }
          default:
               if              ((LA(1)==ID)              &&
(LA(2)==LBRK||LA(2)==ASGN)) {
                    assignment();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
               }
               else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
                    procedure_call_statement();
                    astFactory.addASTChild(currentAST,
returnAST);
                    statement_AST = (AST)currentAST.root;
               }
          else {
               throw     new     NoViableAltException(LT(1),
getFilename());
          }
          }
     }
     catch (RecognitionException ex) {
          reportError(ex);
          consume();
          consumeUntil(_tokenSet_3);
     }
     returnAST = statement_AST;
}
public  final  void  var_list()  throws  RecognitionException,
TokenStreamException {

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

          try {      // for error handling
          switch ( LA(1)) {
          case ID:
          {
               AST tmp46_AST = null;
               tmp46_AST = astFactory.create(LT(1));
               astFactory.addASTChild(currentAST,
tmp46_AST);
               match(ID);
               {
               _loop90:
               do {
                    if ((LA(1)==COMMA)) {
                         match(COMMA);
                         AST tmp48_AST = null;
                         tmp48_AST                         =
astFactory.create(LT(1));
                         astFactory.addASTChild(currentAST,
tmp48_AST);
                         match(ID);
                    }
                    else {
                         break _loop90;
                    }

               } while (true);
               }
               var_list_AST = (AST)currentAST.root;
               var_list_AST  =  (AST)astFactory.make(  (new
ASTArray(2)).add(astFactory.create(VAR_LIST,"VAR_LIST")).add(var
_list_AST));
               currentAST.root = var_list_AST;
               currentAST.child     =     var_list_AST!=null
&&var_list_AST.getFirstChild()!=null ?
                    var_list_AST.getFirstChild()           :
var_list_AST;
               currentAST.advanceChildToEnd();
               var_list_AST = (AST)currentAST.root;
               break;
```

```
				}
				case RPAREN:
				{
					var_list_AST = (AST)currentAST.root;
					var_list_AST   =   (AST)astFactory.make(   (new
ASTArray(2)).add(astFactory.create(VAR_LIST,"VAR_LIST")).add(var
_list_AST));
					currentAST.root = var_list_AST;
					currentAST.child     =     var_list_AST!=null
&&var_list_AST.getFirstChild()!=null ?
						var_list_AST.getFirstChild()           :
var_list_AST;
					currentAST.advanceChildToEnd();
					var_list_AST = (AST)currentAST.root;
					break;
				}
				default:
				{
					throw     new     NoViableAltException(LT(1),
getFilename());
				}
				}
			}
			catch (RecognitionException ex) {
				reportError(ex);
				consume();
				consumeUntil(_tokenSet_8);
			}
			returnAST = var_list_AST;
		}
		public  final  void  vector()  throws  RecognitionException,
TokenStreamException {

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

			try {      // for error handling
				match(LBRK);
				arith_expression();
				astFactory.addASTChild(currentAST, returnAST);
				{
				_loop114:
				do {
					if ((LA(1)==COMMA)) {
						match(COMMA);
						arith_expression();
```

```
                         astFactory.addASTChild(currentAST,
returnAST);
                    }
                    else {
                         break _loop114;
                    }

               } while (true);
               }
               match(RBRK);
               vector_AST = (AST)currentAST.root;
               vector_AST     =     (AST)astFactory.make(     (new
ASTArray(2)).add(astFactory.create(VECTOR,"VECTOR")).add(vector_
AST));
               currentAST.root = vector_AST;
               currentAST.child        =        vector_AST!=null
&&vector_AST.getFirstChild()!=null ?
                    vector_AST.getFirstChild() : vector_AST;
               currentAST.advanceChildToEnd();
               vector_AST = (AST)currentAST.root;
          }
          catch (RecognitionException ex) {
               reportError(ex);
               consume();
               consumeUntil(_tokenSet_14);
          }
          returnAST = vector_AST;
     }
}
```

8.11. SASSiPlotter.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-15 12:11:01)
 * @author: Xiaotang Zhang
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.Color.*;
import java.lang.reflect.Array;
```

```
public class SASSiPlotter extends JPanel {
    JFrame frame;
    Font tick_font;
    Font label_font;

     private static Color[] colors = {
      Color.red, Color.black, Color.blue,
      Color.pink, Color.yellow, Color.orange,
      Color.cyan, Color.pink, Color.magenta,
      Color.green };

     Insets border = new Insets( 32, 64, 32, 64 );
     private static int frame_counter = 0;
     float[] range;
    private static double[] x;
    private static double[] y;
    private static double[] z;
    private static int num;
    private static int maxVal;

    private static String type = "";

    /** the constructor
     * @param frame    the window that contains this sassiplot
panel
     */
    public SASSiPlotter( JFrame frame ) {
        this.frame = frame;
        tick_font = new Font( "Monospaced", Font.PLAIN, 10 );
        label_font = new Font( null, Font.PLAIN, 12 );
        range = new float[24];
        for ( int i=0; i<24; i+=2 )
        {
            range[i] = 0f;
            range[i+1] = 1f;
        }
    }
    /** create a sassiplot window with title and drawing area
size
     * @param title    the title of the window
     * @param width    the width of drawing area
     * @param height   the height of drawing area
     */
    public static SASSiPlotter create( String title, int width,
int height ) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName()
);
        } catch (Exception e) {}

        JFrame frame = new JFrame( title );
        SASSiPlotter pane = new SASSiPlotter( frame );
        frame.getContentPane().add( pane, BorderLayout.CENTER );

        frame_counter++;

        frame.addWindowListener(
            new WindowAdapter()
            {
                public void windowClosing( WindowEvent e ) {
                    frame_counter--;
                    /* System.exit(0); */
                }
            });

        pane.newsize( width, height );

  //      frame.setLocationRelativeTo( null );
        frame.setVisible( true );

        return pane;
    }
      public void drawBarChart(SsVector piece){
        int sum=0;
        type = "bar";
        num= piece.size();
        maxVal = (int)(piece.max());
        x = new double[num];
        for ( int i=0; i<num; i++ ) {
            x[i] = piece.get(i);
        }
        repaint();
    }
    public void drawError(SsVector piece){
        int sum=0;

        type = "error";
        num= piece.size();
        maxVal = (int)(piece.max());
        x = new double[num];
        for (int i = 0; i < num; i++)
          x[i] = piece.elementAt(i);
        SsVector yValue = new SsVector(num);
        for (int j = 0; j < num; j++)
          yValue.set(j, j+1);

        y = new double[2];
        SsVector lr = new SsVector(2);
        SASSiStatisticalMethords       aMethod       =       new
SASSiStatisticalMethords();
        lr = aMethod.linearReg(yValue, piece);
        y[0] = lr.elementAt(0);
        y[1] = lr.elementAt(1);
        repaint();
    }
    public void drawLine(SsVector piece){
        int sum=0;

        type = "line";
        num= piece.size();
        maxVal = (int)(piece.max());
        x = new double[num];
        for (int i = 0; i < num; i++)
          x[i] = piece.elementAt(i);

        repaint();
    }
    public void drawPieChart(SsVector piece){
        int sum=0;
        type = "pie";
        num= piece.size();
        maxVal = (int)(piece.max());
        y = new double[num+1];

        for(int i=0; i<num; i++){
            sum += piece.elementAt(i);
        }

        y[0]= 0;
        double percent = 360.0/sum;
        z = new double[num];
        for(int j=0; j<num; j++){
             z[j] = piece.elementAt(j) * percent;
            y[j + 1] = z[j] + y[j];

        }
        repaint();
    }
    public void drawPolyLine(SsVector piece){
        type = "polyline";
        num= piece.size();
        maxVal = (int)(piece.max());
        x = new double[num];
        y = new double[num];

        for (int i = 0; i < num; i++)
        {
             x[i] = i;
             y[i] = piece.elementAt(i);
        }

        repaint();
    }
    /** return the number of sassiplot windows */
    public static int frameCount() {
        return frame_counter;
    }
    /* change the dimensions of the drawing panel */
    private void newsize( int width, int height ) {
        setPreferredSize( new Dimension( width, height ) );
        frame.pack();
    }
    public void paintComponent( Graphics g ) {
        super.paintComponent (g);

         if (type.equals("line"))
         {
              int y0 = (int)(x[0]);
              int y1 = (int)(x[1] * getWidth() + x[0]);
              g.drawLine(0,   getHeight()   -   y0,   getWidth(),
getHeight() - y1);
         }
         else if (type.equals("error"))
         {
              int    y0    =    (int)(y[0]    +x[0]/maxVal    *
getHeight()*0.9) ;
              int  y1  =  (int)(y[1]  *  getWidth()  +  y[0]  +
x[0]/maxVal * getHeight()*0.9);
              g.drawLine(0,   getHeight()   -   y0,   getWidth(),
getHeight() - y1);

              g.setColor(Color.red);
              for (int i = 0 ; i < num; i++)
```

```
               g.fillRect((int)((i+0.01)/num*getWidth()*0.9),
(int)(getHeight()    -    x[i]/maxVal    *    getHeight()*0.9),
(int)(getWidth()*0.01), (int)(getHeight()* 0.01));
          }
          else if (type.equals("polyline"))
          {
               int[] xVal = new int[num];
               int[] yVal = new int[num];
               for (int i = 0; i < num; i++)
               {
                    xVal[i]    =    (int)(x[i]/num*getWidth()    +
0.5*getWidth()/num);
                    yVal[i]      =      (int)(getHeight()       -
y[i]*getHeight()/maxVal * 0.8);
               }

               g.setColor(Color.red);
               for (int i = 0 ; i < num; i++)
                g.drawPolyline(xVal, yVal, num);
          }
          else if (type.equals("bar"))
          {
               int xSpace = (int)(getWidth()/(3*num));
               for (int i = 0; i < num ; i++)
               {
                    int  height  =  (int)(getHeight()  *  x[i]  /
maxVal * 0.9);
                    int index = i % colors.length;
                    g.setColor(colors[index]);


g.fillRect((int)(xSpace*(3*i+1)),(int)(getHeight()   -   height),
xSpace, height);
               }
          }
          else if (type.equals("pie"))
          {
               int  center  =  Math.min(getWidth(),  getHeight())  /
2;
               int diameter = (int)(center * (0.75));

                for (int i = 0; i < num; i++)
                {
                     int index = i % colors.length;
                     g.setColor(colors[index]);
                     g.fillArc(center,     center,     diameter,
diameter, (int)(y[i]), (int)(z[i]));
```

```
            }
        }

    }
}
```

8.12. SASSiProcedureCall.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-10 15:1:22)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;
import antlr.collections.AST;


class SASSiProcedureCall extends SASSiDataType {
    // we need a reference to the AST for the procedure entry
    String[] args;
    AST body;            // body = null means a built-in
procedure call
    SASSiSymbolTable pst;   // the symbol table of static parent
    int id;              // for internal functions only

    public SASSiProcedureCall( String name, String[] args,
                       AST body, SASSiSymbolTable pst) {
        super( name );
        this.args = args;
        this.body = body;
        this.pst = pst;
    }
    public SASSiProcedureCall( String name, int id ) {
        super( name );
        this.args = null;
        this.id = id;
        pst = null;
        body = null;
    }
    public SASSiDataType copy() {
        return new SASSiProcedureCall( name, args, body, pst );
    }
    public String[] getArgs() {
        return args;
    }
    public AST getBody() {
        return body;
    }
    public final int getInternalId() {
        return id;
    }
    public SASSiSymbolTable getParentSymbolTable() {
        return pst;
    }
    public final boolean isInternal() {
        return body == null;
    }
    public void print( PrintWriter w ) {
        if ( body == null )
        {
            w.println( name + " = <built-in procedure call> #" +
id );
        }
        else
        {
            if ( name != null )
                w.print( name + " = " );
            w.print( "<procedure>(" );
            for ( int i=0; ; i++ )
            {
                w.print( args[i] );
                if ( i >= args.length - 1 )
                    break;
                w.print( "," );
            }
            w.println( ")" );
        }
    }
    public String typename() {
        return "function";
    }
}
```

8.13. SASSiStatisticalMethords.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-10 15:11:22)
 * @author: Xiaotang Zhang
 */
```

```
import java.util.*;
import java.io.PrintWriter;
import java.io.IOException;


public class SASSiStatisticalMethords implements Cloneable{

     private static double[] stdDestriTable
                         = new double[]{ 0.0000, 0.0040, 0.0080,
0.0120, 0.0160, 0.0199, 0.0239, 0.0279, 0.0319, 0.0359,
                                             0.0398,     0.0438,
0.0478, 0.0517, 0.0557, 0.0596, 0.0636, 0.0675, 0.0714, 0.0753,
                                             0.0793,     0.0832,
0.0871, 0.0910, 0.0948, 0.0987, 0.1026, 0.1064, 0.1103, 0.1141,
                                             0.1179,     0.1217,
0.1255, 0.1293, 0.1331, 0.1368, 0.1406, 0.1443, 0.1480, 0.1517,
                                             0.1915,     0.1950,
0.1985, 0.2019, 0.2054, 0.2088, 0.2123, 0.2157, 0.2190, 0.2224,

                                             0.2257,     0.2291,
0.2324, 0.2357, 0.2389, 0.2422, 0.2454, 0.2486, 0.2517, 0.2549,
                                             0.2580,     0.2611,
0.2642, 0.2673, 0.2704, 0.2734, 0.2764, 0.2794, 0.2823, 0.2852,
                                             0.2881,     0.2910,
0.2939, 0.2967, 0.2995, 0.3023, 0.3051, 0.3078, 0.3106, 0.3133,
                                             0.3159,     0.3186,
0.3212, 0.3238, 0.3264, 0.3289, 0.3315, 0.3340, 0.3365, 0.3389,
                                             0.3413,     0.3438,
0.3461, 0.3485, 0.3508, 0.3531, 0.3554, 0.3577, 0.3599, 0.3621,

                                             0.3643,     0.3665,
0.3686, 0.3708, 0.3729, 0.3749, 0.3770, 0.3790, 0.3810, 0.3830,
                                             0.3849,     0.3869,
0.3888, 0.3907, 0.3925, 0.3944, 0.3962, 0.3980, 0.3997, 0.4015,
                                             0.4032,     0.4049,
0.4066, 0.4082, 0.4099, 0.4115, 0.4131, 0.4147, 0.4162, 0.4177,
                                             0.4192,     0.4207,
0.4222, 0.4236, 0.4251, 0.4265, 0.4279, 0.4292, 0.4306, 0.4319,
                                             0.4332,     0.4345,
0.4357, 0.4370, 0.4382, 0.4394, 0.4406, 0.4418, 0.4429, 0.4441,

                                             0.4452,     0.4463,
0.4474, 0.4484, 0.4495, 0.4505, 0.4515, 0.4525, 0.4535, 0.4545,
                                             0.4554,     0.4565,
0.4573, 0.4582, 0.4591, 0.4599, 0.4608, 0.4616, 0.4625, 0.4633,
                                             0.4641,     0.4649,
0.4656, 0.4664, 0.4671, 0.4678, 0.4686, 0.4693, 0.4699, 0.4706,
                                                      0.4713,       0.4719,
0.4726, 0.4732, 0.4738, 0.4744, 0.4750, 0.4756, 0.4761, 0.4767,
                                                      0.4772,       0.4778,
0.4783, 0.4788, 0.4793, 0.4798, 0.4803, 0.4808, 0.4812, 0.4817,

                                                      0.4821,       0.4826,
0.4830, 0.4834, 0.4838, 0.4842, 0.4846, 0.4850, 0.4854, 0.4857,
                                                      0.4961,       0.4964,
0.4868, 0.4871, 0.4875, 0.4878, 0.4881, 0.4884, 0.4887, 0.4890,
                                                      0.4893,       0.4896,
0.4898, 0.4901, 0.4904, 0.4906, 0.4909, 0.4911, 0.4913, 0.4916,
                                                      0.4918,       0.4920,
0.4922, 0.4925, 0.4927, 0.4929, 0.4931, 0.4932, 0.4934, 0.4936,
                                                      0.4938,       0.4940,
0.4941, 0.4943, 0.4945, 0.4946, 0.4948, 0.4949, 0.4951, 0.4952,

                                                      0.4953,       0.4955,
0.4956, 0.4957, 0.4959, 0.4960, 0.4961, 0.4962, 0.4963, 0.4964,
                                                      0.4965,       0.4966,
0.4967, 0.4968, 0.4969, 0.4970, 0.4971, 0.4972, 0.4973, 0.4974,
                                                      0.4974,       0.4975,
0.4976, 0.4977, 0.4977, 0.4978, 0.4979, 0.4978, 0.7980, 0.4981,
                                                      0.4981,       0.4982,
0.4982, 0.4983, 0.4984, 0.4984, 0.4985, 0.4985, 0.4986, 0.4986,
                                                      0.4987,       0.4987,
0.4987,  0.4988,  0.4988,  0.4989,  0.4989,  0.4989,  0.4990,  0.4990
     };
/**
 * SASSiFunctionCalls constructor comment.
 */
public SASSiStatisticalMethords() {
     super();
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.Vector
 */
public double binoDistri(int x, int n, double p) {

     double px = 1;
     for (int i = 0; i <x; i ++)
          px = px * p;
     double pnx = 1;
     for (int j = 0; j < n-x; j++)
          pnx = pnx * (1-p);
     double   probability   =   factorial(n)   /   (factorial(x)   *
factorial(n-x)) * px * pnx;

     return probability;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public boolean contains (SsVector vect, double number) {

     return vect.contains(number);
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public   double   expectation   (SsVector   events,   SsVector
probabilities) {

     double expectation = 0;
     int size = Math.min(events.size(), probabilities.size());
     for (int i = 0 ; i < size; i ++)
          expectation          +=          events.elementAt(i)*
probabilities.elementAt(i);

     return expectation;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.Vector
 */
public int factorial(int i) {
     int factorial = 1;
     for (int j = 1; j < i + 1; j++)
          factorial *= j;

     return factorial;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.Vector
 */
public double geoDistri(int x, double p) {

     double px = 1;
     for (int i=0; i<x-2; i++)
          px = px*(1-p);

     double probability = p*px;
     return probability;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public SsVector intersect (SsVector vect1, SsVector vect2) {

     SsVector vect = new SsVector(vect1.size() + vect2.size());
     vect1 = sort(vect1);
     vect2 = sort(vect2);
     int k = 0;
     while (!vect1.isEmpty() && !vect2.isEmpty())
     {
          double currentElementIn1 = vect1.firstElement();
          double currentElementIn2 = vect2.firstElement();
          if (currentElementIn1 < currentElementIn2)
               vect1.removeElementAt(0);
          else if (currentElementIn1 > currentElementIn2)
               vect2.removeElementAt(0);
          else
               vect.set(k++, currentElementIn1);
     }

     vect.n = k;
     return vect;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public SsVector linearReg (SsVector x, SsVector y) {
     SsVector vect = new SsVector(2);
     double b = sxy(x, y) / sxy(x, x);
     double a = mean(y) - b * mean(x);
     vect.set(0, a);
     vect.set(1, b);
     return vect;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public double mean(SsVector vect) {
     double mean = 0;
     int i = 1;
     for (i = 0; i < vect.size(); i++)
     {
          mean += vect.elementAt(i);
     }

     return mean/i;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public double median(SsVector vect) {
     double median = 0;
     int i = 1;
     if(!vect.isEmpty())
     {
          int size = vect.size();
          SsVector sorted = sort(vect);
          median = sorted.elementAt(size/2);
     }

     return median;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.Vector
 */
public double normalDistri(double x, double sigma, double mean)
{

     double std = (x-mean) / sigma;

     return stdDistri(std);
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public double range(SsVector vect) {
     double range = vect.max() - vect.min();

     return range;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect sassi.SsVector
 */
public int size(SsVector vect) {

     return vect.size();
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:36:31)
 * @return java.util.SsVector
 */
public SsVector sort(SsVector vect) {
     SsVector sortedVect = new SsVector(vect.size());
     SsVector tempVect = new SsVector(vect);

     int position = 0;
     int size = vect.size();
     int innerSize = size;

     for (int i = 0; i < size; i ++)
     {
          double mimValue = tempVect.elementAt(0);
          for (int j=0; j < innerSize; j ++)
          {
               double currentElement = vect.elementAt(j);
               if(currentElement < mimValue )
               {
                    mimValue = currentElement;
                    position = j;
               }
          }
          sortedVect.set(i, mimValue);
          tempVect.removeElementAt(position);
          innerSize--;
     }

     return sortedVect;
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public double stdDev(SsVector vect) {

     return Math.sqrt(variance(vect));
}
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.Vector
 */
public double stdDistri(double x) {

     double probability = 0;

     double aDistr = 0;
     int row = (int) (Math.abs(x)/0.10);
     int column = (int) (Math.abs(x) * 100) % 10;
     int index = row * 10 + column;

     if(index > stdDestriTable.length)
          aDistr = 0.5;
     else
          aDistr = stdDestriTable[index];

     if (x >= 0)
          probability = aDistr + 0.5;
     else
```

```
			probability = 0.5 - aDistr;

		return probability;
	}
	/**
	 * Insert the method's description here.
	 * Creation date: (2003-12-10 15:18:10)
	 * @return double
	 * @param vect plt.SsVector
	 */
	public SsVector union (SsVector vect1, SsVector vect2) {

		SsVector vect = new SsVector(vect1.size() + vect2.size());
		int k = 0;
		for (int i = 0 ; i < vect1.size(); i ++)
			if(!vect.contains(vect1.elementAt(i)))
				vect.set(k++, vect1.elementAt(i));

		for (int j = 0 ; j < vect2.size(); j ++)
			if(!vect.contains(vect2.elementAt(j)))
				vect.set(k++, vect2.elementAt(j));

		vect.n = k;
		return vect;
	}
	/**
	 * Insert the method's description here.
	 * Creation date: (2003-12-10 15:18:10)
	 * @return double
	 * @param vect plt.Vector
	 */
	public double variance(SsVector vect) {

		if(!vect.isEmpty())
		{
			int size = vect.size();
			double sum = 0;
			for (int i =0 ; i < size; i ++)
			{
				double currentElement = vect.elementAt(i);
				sum += currentElement * currentElement;
			}
			return sum/ (size -1);
		}

		return 0;
	}
```

```
/**
 * Insert the method's description here.
 * Creation date: (2003-12-10 15:18:10)
 * @return double
 * @param vect plt.SsVector
 */
public double sxy (SsVector x1, SsVector x2) {

     double sxy;

     int size = Math.min(x1.size(), x2.size());

     double sumxy2 = 0;
     double sumx=0;
     double sumy=0;

     for (int i = 0; i < size; i++)
     {
          sumxy2 += x1.elementAt(i) * x2.elementAt(i);
          sumx += x1.elementAt(i);
          sumy += x2.elementAt(i);
     }

     sxy = sumxy2 - sumx * sumy / size;
     return sxy;
}
}
```

8.14. SASSiString.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-11 11:28:11)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;

class SASSiString extends SASSiDataType {
    String var;

    public SASSiString( String str ) {
        this.var = str;
    }
    public SASSiDataType add( SASSiDataType b ) {
        if ( b instanceof SASSiString )
        {
            var = var + ((SASSiString)b).var;
            return this;
        }

        return error( b, "+=" );
    }
    public SASSiDataType copy() {
        return new SASSiString( var );
    }
    public SASSiDataType plus( SASSiDataType b ) {
        if ( b instanceof SASSiString )
            return new SASSiString( var + ((SASSiString)b).var
);

        return error( b, "+" );
    }
    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.print( var );
        w.println();
    }
    public String typename() {
        return "string";
    }
}
```

8.15. SASSiSymbolTable.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-11 11:28:13)
 * @author: Xiaotang Zhang
 */
import java.util.*;
import java.io.PrintWriter;


class SASSiSymbolTable extends HashMap {
    SASSiSymbolTable static_parent, dynamic_parent;
    boolean read_only;
    public SASSiSymbolTable( SASSiSymbolTable sparent,
SASSiSymbolTable dparent ) {
        static_parent = sparent;
        dynamic_parent = dparent;
        read_only = false;
    }
    public final boolean containsVar( String name ) {
        return containsKey( name );
    }
    public final SASSiSymbolTable dynamicParent() {
        return dynamic_parent;
    }
    public final SASSiDataType getValue( String name, boolean
is_static,
                                             int level ) {
        SASSiSymbolTable st = gotoLevel( level, is_static );
        Object x = st.get( name );

        while ( null == x && null != st.parent( is_static ) )
        {
            st = st.parent( is_static );
            x = st.get( name );
        }

        return (SASSiDataType) x;
    }
    private final SASSiSymbolTable gotoLevel( int level, boolean
is_static ) {
        SASSiSymbolTable st = this;

        if ( level < 0 )
        {
            // global variable
            while ( null != st.static_parent )
                st = st.parent( is_static );
        }
        else
        {
            // local variable
            for ( int i=level; i>0; i-- )
            {
                while ( st.read_only )
                {
                    st = st.parent( is_static );
            //        assert st expecting != null;
                }
                if ( null != st.parent( is_static ) )
                    st = st.parent( is_static );
                else
                    break;
            }
        }

        return st;
    }
    public final SASSiSymbolTable parent( boolean is_static ) {
        return is_static ? static_parent : dynamic_parent;
    }
    public void setReadOnly() {
        read_only = true;
    }
    public final void setValue( String name, SASSiDataType data,
                                boolean is_static, int level ) {

        SASSiSymbolTable st = gotoLevel( level, is_static );
        while ( st.read_only )
        {
            st = st.parent( is_static );
        //    assert st != null;
        }

        st.put( name, data );
    }
    public final SASSiSymbolTable staticParent() {
        return static_parent;
    }
    public void what() {
        what( new PrintWriter( System.out, true ) );
    }
    public void what( PrintWriter output ) {
        for ( Iterator it = values().iterator() ; it.hasNext();
)
        {
            SASSiDataType d = ((SASSiDataType)(it.next()));
            if ( !( d instanceof SASSiProcedureCall &&
((SASSiProcedureCall)d).isInternal() ) )
                d.what( output );
        }
    }
}
```

8.16. SASSiTokenTypes.java

```
package sassi;

// $ANTLR 2.7.2: "blah.g" -> "SASSiParser.java"$

public interface SASSiTokenTypes {
     int EOF = 1;
     int NULL_TREE_LOOKAHEAD = 3;
     int ALPHA = 4;
     int DIGIT = 5;
     int WS = 6;
     int NL = 7;
     int COMMENT = 8;
     int LPAREN = 9;
     int RPAREN = 10;
     int LBRACE = 11;
     int RBRACE = 12;
     int LBRK = 13;
     int RBRK = 14;
     int ASGN = 15;
     int COMMA = 16;
     int MULT = 17;
     int PLUS = 18;
     int MINUS = 19;
     int RDV = 20;
     int GE = 21;
     int LE = 22;
     int GT = 23;
     int LT = 24;
     int EQ = 25;
     int NEQ = 26;
     int INC = 27;
     int DEC = 28;
     int SEMI = 29;
     int ID = 30;
     int NUMBER = 31;
     int STRING = 32;
     int STATEMENT = 33;
     int FOR_LOOP = 34;
     int VAR_LIST = 35;
     int VECTOR = 36;
     int EXPRESSION_LIST = 37;
     int CONSTANT_STATEMENT = 38;
     int PROCEDURE_CALL = 39;
     int FOR_CON = 40;
     int UPLUS = 41;
     int UMINUS = 42;
     int LITERAL_const = 43;
     int LITERAL_xconst = 44;
     int LITERAL_for = 45;
     int LITERAL_if = 46;
     int LITERAL_else = 47;
     int LITERAL_return = 48;
     int LITERAL_include = 49;
     int LITERAL_procedure = 50;
     int LITERAL_true = 51;
     int LITERAL_True = 52;
     int LITERAL_TRUE = 53;
     int LITERAL_False = 54;
     int LITERAL_FALSE = 55;
     int LITERAL_false = 56;
     int LITERAL_exit = 57;
}
```

8.17. SASSiVariable.java

```
package sassi;

/**
 * Insert the type's description here.
 * Creation date: (2003-12-11 11:28:15)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;

class SASSiVariable extends SASSiDataType {
    public SASSiVariable( String name ) {
        super( name );
    }
    public SASSiDataType copy() {
        throw new SASSiException( "Variable " + name + " has not 
been defined" );
    }
    public void print( PrintWriter w ) {
        w.println( name + " = <undefined>" );
    }
    public String typename() {
        return "undefined-variable";
    }
}
```

8.18. SASSiVector.java

```
package sassi;
```

```
/**
 * Insert the type's description here.
 * Creation date: (2003-12-8 17:36:15)
 * @author: Xiaotang Zhang
 */
import java.io.PrintWriter;
import java.util.*;

class SASSiVector extends SASSiDataType {
    SsVector  vect;
    SASSiVector( SsVector vect ) {
        this.vect = vect;
    }
    public SASSiDataType copy() {
        return new SASSiVector( vect);
    }
        public SASSiDataType deepCopy() {
        return new SASSiVector( vect.copy() );
    }
    public SASSiDataType eq( SASSiDataType b ) {
        if ( b instanceof SASSiVector)
        {
             return new
SASSiBool(vect.eq(((SASSiVector)b).vect));
        }
        return b.eq( this );
    }
       private static final int getDim( SASSiDataType x) {
        if ( x instanceof SASSiDouble || x instanceof SASSiInt )
            return 1;
        if ( x instanceof SASSiVector)
            return ((SASSiVector)x).vect.size();
        x.error( "array [none number/array element]" );
        return 0;
    }
    public static SASSiDataType joinVert( SASSiDataType [] x ) {
        if ( x.length == 0 )
            throw new IllegalArgumentException( "no data in
array" );

        int vectorSize = x.length;

        SASSiVector y = new SASSiVector( new SsVector(
vectorSize ) );

        int colp = 0;
        for ( int i=0; i<vectorSize; i++ )
```

```
            y.vect.set( colp++, SASSiDouble.doubleValue( x[i] )
);

        return y;
    }
    public SASSiDataType mul( SASSiDataType b ){

        vect.selfmul( SASSiDouble.doubleValue( b ) );
        return this;
    }
    public SASSiDataType ne( SASSiDataType b ) {
          if ( b instanceof SASSiVector)
            return new
SASSiBool(vect.ne(((SASSiVector)b).vect));

        return b.ne( this );
    }
    public SASSiDataType times( SASSiDataType b ) {

         return new SASSiVector( vect.times(
SASSiDouble.doubleValue( b ) ) );
    }
    public String typename() {
        return "vector";
    }
    public SASSiDataType uminus() {

        return new SASSiVector (vect.uminus());
    }
    public void what( PrintWriter w ) {
        w.print( "<" + typename() + ">  " );
        if ( name != null )
            w.print( name + "  " );
        w.println( " " + vect.size() + " elements" );
     }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.println( name + " = " );
        vect.print( w, 8, 4);

    }

    public SASSiDataType ldiv( SASSiDataType b ) {
        return new SASSiVector( vect.div(
SASSiDouble.doubleValue( b ) ) );
    }
    public SASSiDataType rdiv( SASSiDataType b ) {
        return ldiv( b );
    }
}
```

8.19. SASSiWalker.java

```
package sassi;

// $ANTLR 2.7.2: "blahwalk3.g" -> "SASSiWalker.java"$

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.*;


public class SASSiWalker extends antlr.TreeParser
implements SASSiWalkerTokenTypes
 {

    static SASSiDataType null_data = new SASSiDataType( "<NULL>"
);
    SASSiInterpreter interp = new SASSiInterpreter();
     public static final String[] _tokenNames = {
          "<0>",
          "EOF",
          "<2>",
          "NULL_TREE_LOOKAHEAD",
          "ALPHA",
          "DIGIT",
          "WS",
          "NL",
          "COMMENT",
          "LPAREN",
          "RPAREN",
          "LBRACE",
           "RBRACE",
           "LBRK",
           "RBRK",
          "ASGN",
          "COMMA",
          "MULT",
          "PLUS",
          "MINUS",
          "RDV",
          "GE",
          "LE",
          "GT",
          "LT",
          "EQ",
          "NEQ",
          "INC",
          "DEC",
          "SEMI",
          "ID",
          "NUMBER",
          "STRING",
          "STATEMENT",
          "FOR_LOOP",
          "VAR_LIST",
          "VECTOR",
          "EXPRESSION_LIST",
          "CONSTANT_STATEMENT",
          "PROCEDURE_CALL",
          "FOR_CON",
          "UPLUS",
          "UMINUS",
          "\"const\"",
          "\"xconst\"",
          "\"for\"",
          "\"if\"",
          "\"else\"",
          "\"return\"",
          "\"include\"",
          "\"procedure\"",
          "\"true\"",
          "\"True\"",
          "\"TRUE\"",
          "\"False\"",
          "\"FALSE\"",
          "\"false\"",
          "\"exit\""
```

```
     };

     public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
     public static final BitSet _tokenSet_1 = new
BitSet(mk_tokenSet_1());

public SASSiWalker() {
     tokenNames = _tokenNames;
}
     public final  SASSiDataType  expr(AST _t) throws
RecognitionException {
           SASSiDataType r ;

          AST expr_AST_in = (AST)_t;
          AST num = null;
          AST str = null;
          AST id = null;
          AST thenp = null;
          AST elsep = null;
          AST stmt = null;
          AST procedure_name = null;
          AST probody = null;

          SASSiDataType a, b;
          Vector v;
          SASSiDataType[] x;
          String s = null;
          String[] sx;
          r = null_data;


          try {      // for error handling
               if (_t==null) _t=ASTNULL;
               switch ( _t.getType()) {
               case GE:
               {
                    AST __t2 = _t;
                    AST tmp1_AST_in = (AST)_t;
                    match(_t,GE);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    b=expr(_t);
                    _t = _retTree;
                    _t = __t2;
                    _t = _t.getNextSibling();
                    r = a.ge( b ); //System.out.println(r);
                    break;
               }
               case LE:
               {
                    AST __t3 = _t;
                    AST tmp2_AST_in = (AST)_t;
                    match(_t,LE);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    b=expr(_t);
                    _t = _retTree;
                    _t = __t3;
                    _t = _t.getNextSibling();
                    r = a.le( b );// System.out.println(r);
                    break;
               }
               case GT:
               {
                    AST __t4 = _t;
                    AST tmp3_AST_in = (AST)_t;
                    match(_t,GT);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    b=expr(_t);
                    _t = _retTree;
                    _t = __t4;
                    _t = _t.getNextSibling();
                    r = a.gt( b ); //System.out.println(r);
                    break;
               }
               case LT:
               {
                    AST __t5 = _t;
                    AST tmp4_AST_in = (AST)_t;
                    match(_t,LT);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    b=expr(_t);
                    _t = _retTree;
                    _t = __t5;
                    _t = _t.getNextSibling();
                    r = a.lt( b );// System.out.println(r);
                    break;
```

```
}
case EQ:
{
     AST __t6 = _t;
     AST tmp5_AST_in = (AST)_t;
     match(_t,EQ);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t6;
     _t = _t.getNextSibling();
     r = a.eq( b );// System.out.println(r);
     break;
}
case NEQ:
{
     AST __t7 = _t;
     AST tmp6_AST_in = (AST)_t;
     match(_t,NEQ);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t7;
     _t = _t.getNextSibling();
     r = a.ne( b ); //System.out.println(r);
     break;
}
case PLUS:
{
     AST __t8 = _t;
     AST tmp7_AST_in = (AST)_t;
     match(_t,PLUS);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t8;
     _t = _t.getNextSibling();
     r = a.plus( b );// System.out.println(r);
     break;
}
case MINUS:
```

```
{
     AST __t9 = _t;
     AST tmp8_AST_in = (AST)_t;
     match(_t,MINUS);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t9;
     _t = _t.getNextSibling();
     r = a.minus( b ); //System.out.println(r);
     break;
}
case MULT:
{
     AST __t10 = _t;
     AST tmp9_AST_in = (AST)_t;
     match(_t,MULT);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t10;
     _t = _t.getNextSibling();
     r = a.times( b ); //System.out.println(r);
     break;
}
case RDV:
{
     AST __t11 = _t;
     AST tmp10_AST_in = (AST)_t;
     match(_t,RDV);
     _t = _t.getFirstChild();
     a=expr(_t);
     _t = _retTree;
     b=expr(_t);
     _t = _retTree;
     _t = __t11;
     _t = _t.getNextSibling();
     r = a.rfracts( b );// System.out.println(r);
     break;
}
case UPLUS:
{
     AST __t12 = _t;
```

```
                    AST tmp11_AST_in = (AST)_t;
                    match(_t,UPLUS);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    _t = __t12;
                    _t = _t.getNextSibling();
                    r = a; //System.out.println(r);
                    break;
               }
               case UMINUS:
               {
                    AST __t13 = _t;
                    AST tmp12_AST_in = (AST)_t;
                    match(_t,UMINUS);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    _t = __t13;
                    _t = _t.getNextSibling();
                    r = a.uminus(); //System.out.println(r);
                    break;
               }
               case ASGN:
               {
                    AST __t14 = _t;
                    AST tmp13_AST_in = (AST)_t;
                    match(_t,ASGN);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    b=expr(_t);
                    _t = _retTree;
                    _t = __t14;
                    _t = _t.getNextSibling();
                    r = interp.assign( a, b );//
System.out.println(r);
                    break;
               }
               case PROCEDURE_CALL:
               {
                    AST __t15 = _t;
                    AST tmp14_AST_in = (AST)_t;
                    match(_t,PROCEDURE_CALL);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    x=vexpr(_t);
                    _t = _retTree;
                    _t = __t15;
                    _t = _t.getNextSibling();
                    r = interp.procedureInvoke( this, a, x );
//System.out.println(r);
                    break;
               }
               case VECTOR:
               {
                    AST __t16 = _t;
                    AST tmp15_AST_in = (AST)_t;
                    match(_t,VECTOR);
                    _t = _t.getFirstChild();
                    v = new Vector();
                    {
                    _loop18:
                    do {
                         if (_t==null) _t=ASTNULL;
                         if ((_tokenSet_0.member(_t.getType())))
{
                              a=expr(_t);
                              _t = _retTree;
                              v.add( a );
//System.out.println(a);
                         }
                         else {
                              break _loop18;
                         }

                    } while (true);
                    }
                    _t = __t16;
                    _t = _t.getNextSibling();
                    r =
SASSiVector.joinVert(interp.convertExprList( v ) );
                    break;
               }
               case NUMBER:
               {
                    num = (AST)_t;
                    match(_t,NUMBER);
                    _t = _t.getNextSibling();
                    r = interp.getNumber( num.getText() );
//System.out.println(r);
                    break;
               }
               case STRING:
               {
                    str = (AST)_t;
                    match(_t,STRING);
                    _t = _t.getNextSibling();
                    r = new SASSiString( str.getText() );//
System.out.println(r);
                    break;
               }
               case LITERAL_true:
               {
                    AST tmp16_AST_in = (AST)_t;
                    match(_t,LITERAL_true);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( true );
//System.out.println(r);
                    break;
               }
               case LITERAL_True:
               {
                    AST tmp17_AST_in = (AST)_t;
                    match(_t,LITERAL_True);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( true );
//System.out.println(r);
                    break;
               }
               case LITERAL_TRUE:
               {
                    AST tmp18_AST_in = (AST)_t;
                    match(_t,LITERAL_TRUE);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( true );
//System.out.println(r);
                    break;
               }
               case LITERAL_false:
               {
                    AST tmp19_AST_in = (AST)_t;
                    match(_t,LITERAL_false);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( false );//
System.out.println(r);
                    break;
               }
               case LITERAL_False:
               {
                    AST tmp20_AST_in = (AST)_t;
                    match(_t,LITERAL_False);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( false );
//System.out.println(r);
                    break;
               }
               case LITERAL_FALSE:
               {
                    AST tmp21_AST_in = (AST)_t;
                    match(_t,LITERAL_FALSE);
                    _t = _t.getNextSibling();
                    r = new SASSiBool( false );
//System.out.println(r);
                    break;
               }
               case ID:
               {
                    AST __t19 = _t;
                    id = _t==ASTNULL ? null :(AST)_t;
                    match(_t,ID);
                    _t = _t.getFirstChild();
                    r = interp.getVariable( id.getText() );
                    {
                    _loop21:
                    do {
                         if (_t==null) _t=ASTNULL;
                         if ((_tokenSet_1.member(_t.getType())))
{
                              x=vexpr(_t);
                              _t = _retTree;
                              r = interp.subMatrix( r, x );
                         }
                         else {
                              break _loop21;
                         }

                    } while (true);
                    }
                    _t = __t19;
                    _t = _t.getNextSibling();
                    break;
               }
               case LITERAL_if:
               {
                    AST __t22 = _t;
                    AST tmp22_AST_in = (AST)_t;
                    match(_t,LITERAL_if);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    thenp = (AST)_t;
                    if ( _t==null ) throw new
MismatchedTokenException();
                    _t = _t.getNextSibling();
                    {
                    if (_t==null) _t=ASTNULL;
                    switch ( _t.getType()) {
                    case ALPHA:
                    case DIGIT:
                    case WS:
                    case NL:
                    case COMMENT:
                    case LPAREN:
                    case RPAREN:
                    case LBRACE:
                    case RBRACE:
                    case LBRK:
                    case RBRK:
                    case ASGN:
                    case COMMA:
                    case MULT:
                    case PLUS:
                    case MINUS:
                    case RDV:
                    case GE:
                    case LE:
                    case GT:
                    case LT:
                    case EQ:
                    case NEQ:
                    case INC:
                    case DEC:
                    case SEMI:
                    case ID:
                    case NUMBER:
                    case STRING:
                    case STATEMENT:
                    case FOR_LOOP:
                    case VAR_LIST:
                    case VECTOR:
                    case EXPRESSION_LIST:
                    case CONSTANT_STATEMENT:
                    case PROCEDURE_CALL:
```

```
                    case FOR_CON:
                    case UPLUS:
                    case UMINUS:
                    case LITERAL_const:
                    case LITERAL_xconst:
                    case LITERAL_for:
                    case LITERAL_if:
                    case LITERAL_else:
                    case LITERAL_return:
                    case LITERAL_include:
                    case LITERAL_procedure:
                    case LITERAL_true:
                    case LITERAL_True:
                    case LITERAL_TRUE:
                    case LITERAL_False:
                    case LITERAL_FALSE:
                    case LITERAL_false:
                    case LITERAL_exit:
                    {
                         elsep = (AST)_t;
                         if ( _t==null ) throw new
MismatchedTokenException();
                         _t = _t.getNextSibling();
                         break;
                    }
                    case 3:
                    {
                         break;
                    }
                    default:
                    {
                         throw new NoViableAltException(_t);
                    }
                    }
                    }
                    _t = __t22;
                    _t = _t.getNextSibling();

                    if ( !( a instanceof SASSiBool ) )
                    {
                                  return a.error( "if:
expression should be bool" );
                                 }
                    if ( ((SASSiBool)a).var )
                    {
                                   r = expr( thenp );
                                 //  System.out.println(r);
```

```
                                  }
                    else if ( null != elsep )
                    {r = expr( elsep );
                                  // System.out.println(r);
                                  }

                    break;
               }
               case STATEMENT:
               {
                    AST __t24 = _t;
                    AST tmp23_AST_in = (AST)_t;
                    match(_t,STATEMENT);
                    _t = _t.getFirstChild();
                    {
                    _loop26:
                    do {
                         if (_t==null) _t=ASTNULL;
                         if (((_t.getType() >= ALPHA &&
_t.getType() <= LITERAL_exit))) {
                              stmt = (AST)_t;
                              if ( _t==null ) throw new
MismatchedTokenException();
                              _t = _t.getNextSibling();
                              if ( interp.canProceed() ) r =
expr(stmt); //System.out.println(r);
                         }
                         else {
                              break _loop26;
                         }

                    } while (true);
                    }
                    _t = __t24;
                    _t = _t.getNextSibling();
                    break;
               }
               case LITERAL_return:
               {
                    AST __t27 = _t;
                    AST tmp24_AST_in = (AST)_t;
                    match(_t,LITERAL_return);
                    _t = _t.getFirstChild();
                    {
                    if (_t==null) _t=ASTNULL;
                    switch ( _t.getType()) {
                    case ASGN:
```

```
                    case MULT:
                    case PLUS:
                    case MINUS:
                    case RDV:
                    case GE:
                    case LE:
                    case GT:
                    case LT:
                    case EQ:
                    case NEQ:
                    case ID:
                    case NUMBER:
                    case STRING:
                    case STATEMENT:
                    case VECTOR:
                    case PROCEDURE_CALL:
                    case UPLUS:
                    case UMINUS:
                    case LITERAL_if:
                    case LITERAL_return:
                    case LITERAL_include:
                    case LITERAL_procedure:
                    case LITERAL_true:
                    case LITERAL_True:
                    case LITERAL_TRUE:
                    case LITERAL_False:
                    case LITERAL_FALSE:
                    case LITERAL_false:
                    {
                         a=expr(_t);
                         _t = _retTree;
                         r = interp.rvalue( a );//
System.out.println(r);
                         break;
                    }
                    case 3:
                    {
                         break;
                    }
                    default:
                    {
                         throw new NoViableAltException(_t);
                    }
                    }
                    }
                    _t = __t27;
                    _t = _t.getNextSibling();
                    interp.setReturn( null );
                    break;
               }
               case LITERAL_procedure:
               {
                    AST __t29 = _t;
                    AST tmp25_AST_in = (AST)_t;
                    match(_t,LITERAL_procedure);
                    _t = _t.getFirstChild();
                    procedure_name = (AST)_t;
                    match(_t,ID);
                    _t = _t.getNextSibling();
                    sx=vlist(_t);
                    _t = _retTree;
                    probody = (AST)_t;
                    if ( _t==null ) throw new
MismatchedTokenException();
                    _t = _t.getNextSibling();
                    _t = __t29;
                    _t = _t.getNextSibling();
                    interp.procedureRegister(
procedure_name.getText(), sx, probody );
                    break;
               }
               case LITERAL_include:
               {
                    AST __t30 = _t;
                    AST tmp26_AST_in = (AST)_t;
                    match(_t,LITERAL_include);
                    _t = _t.getFirstChild();
                    a=expr(_t);
                    _t = _retTree;
                    _t = __t30;
                    _t = _t.getNextSibling();
                    interp.include( a );//
System.out.println(a);
                    break;
               }
               default:
               {
                    throw new NoViableAltException(_t);
               }
               }
          }
          catch (RecognitionException ex) {
               reportError(ex);
               if (_t!=null) {_t = _t.getNextSibling();}
```

```
            }
            _retTree = _t;
            return r ;
        }
        private static final long[] mk_tokenSet_0() {
            long[] data = { 1439113136286269 44L, 0L};
            return data;
        }
        private static final long[] mk_tokenSet_1() {
            long[] data = { 1439114510675804 16L, 0L};
            return data;
        }
        public final  SASSiDataType[]  vexpr(AST _t) throws
    RecognitionException {
             SASSiDataType[] rv ;

            AST vexpr_AST_in = (AST)_t;

            SASSiDataType a;
            rv = null;
            Vector v;


            try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case EXPRESSION_LIST:
                {
                    AST __t32 = _t;
                    AST tmp27_AST_in = (AST)_t;
                    match(_t,EXPRESSION_LIST);
                    _t = _t.getFirstChild();
                    v = new Vector();
                    {
                    _loop34:
                    do {
                        if (_t==null) _t=ASTNULL;
                        if ((_tokenSet_0.member(_t.getType())))
    {
                            a=expr(_t);
                            _t = _retTree;
                            v.add( a );
                        }
                        else {
                            break _loop34;
                        }
```

```
          } while (true);
          }
          _t = __t32;
          _t = _t.getNextSibling();
          rv = interp.convertExprList( v );
          break;
     }
     case ASGN:
     case MULT:
     case PLUS:
     case MINUS:
     case RDV:
     case GE:
     case LE:
     case GT:
     case LT:
     case EQ:
     case NEQ:
     case ID:
     case NUMBER:
     case STRING:
     case STATEMENT:
     case VECTOR:
     case PROCEDURE_CALL:
     case UPLUS:
     case UMINUS:
     case LITERAL_if:
     case LITERAL_return:
     case LITERAL_include:
     case LITERAL_procedure:
     case LITERAL_true:
     case LITERAL_True:
     case LITERAL_TRUE:
     case LITERAL_False:
     case LITERAL_FALSE:
     case LITERAL_false:
     {
          a=expr(_t);
          _t = _retTree;
          rv = new SASSiDataType[1]; rv[0] = a;
          break;
     }
     default:
     {
          throw new NoViableAltException(_t);
     }
     }
```

```
          }
          catch (RecognitionException ex) {
               reportError(ex);
               if (_t!=null) {_t = _t.getNextSibling();}
          }
          _retTree = _t;
          return rv ;
     }
     public final  String[]  vlist(AST _t) throws
RecognitionException {
           String[] sv ;

          AST vlist_AST_in = (AST)_t;
          AST s = null;

          Vector v;
          sv = null;


          try {      // for error handling
               AST __t36 = _t;
               AST tmp28_AST_in = (AST)_t;
               match(_t,VAR_LIST);
               _t = _t.getFirstChild();
               v = new Vector();
               {
               _loop38:
               do {
                    if (_t==null) _t=ASTNULL;
                    if ((_t.getType()==ID)) {
                         s = (AST)_t;
                         match(_t,ID);
                         _t = _t.getNextSibling();
                         v.add( s.getText() );
System.out.println(s);
                    }
                    else {
                         break _loop38;
                    }

               } while (true);
               }
               _t = __t36;
               _t = _t.getNextSibling();
               sv = interp.convertVarList( v );
          }
          catch (RecognitionException ex) {
               reportError(ex);
               if (_t!=null) {_t = _t.getNextSibling();}
          }
          _retTree = _t;
          return sv ;
     }
}
```

8.20. SASSiWalkerTokenType.java

```
package sassi;

// $ANTLR 2.7.2: "blahwalk3.g" -> "SASSiWalker.java"$

public interface SASSiWalkerTokenTypes {
     int EOF = 1;
     int NULL_TREE_LOOKAHEAD = 3;
     int ALPHA = 4;
     int DIGIT = 5;
     int WS = 6;
     int NL = 7;
     int COMMENT = 8;
     int LPAREN = 9;
     int RPAREN = 10;
     int LBRACE = 11;
     int RBRACE = 12;
     int LBRK = 13;
     int RBRK = 14;
     int ASGN = 15;
     int COMMA = 16;
     int MULT = 17;
     int PLUS = 18;
     int MINUS = 19;
     int RDV = 20;
     int GE = 21;
     int LE = 22;
     int GT = 23;
     int LT = 24;
     int EQ = 25;
     int NEQ = 26;
     int INC = 27;
     int DEC = 28;
     int SEMI = 29;
     int ID = 30;
     int NUMBER = 31;
     int STRING = 32;
     int STATEMENT = 33;
     int FOR_LOOP = 34;
     int VAR_LIST = 35;
     int VECTOR = 36;
     int EXPRESSION_LIST = 37;
     int CONSTANT_STATEMENT = 38;
     int PROCEDURE_CALL = 39;
     int FOR_CON = 40;
     int UPLUS = 41;
     int UMINUS = 42;
     int LITERAL_const = 43;
     int LITERAL_xconst = 44;
     int LITERAL_for = 45;
     int LITERAL_if = 46;
     int LITERAL_else = 47;
     int LITERAL_return = 48;
     int LITERAL_include = 49;
     int LITERAL_procedure = 50;
     int LITERAL_true = 51;
     int LITERAL_True = 52;
     int LITERAL_TRUE = 53;
     int LITERAL_False = 54;
     int LITERAL_FALSE = 55;
     int LITERAL_false = 56;
     int LITERAL_exit = 57;
}
```

8.21. SsVector.java

```
package sassi;

import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.io.IOException;
import java.text.NumberFormat;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.Locale;

public class SsVector  implements Cloneable {

    /* internal data structure
     * a:  array to store matrix entries
     * n:  number of columns
     */
    double [] a;
    int n;
    public SsVector( int n ) {
        this.n = n;
        a = new double[n];
    }
    public SsVector( SsVector vect ) {
        n = vect.n;
        a = vect.a;
    }
    public SsVector assign( SsVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
);
        for ( int i=0; i<n; i++ )
            set( i, b.get( i ) );
        return this;
    }
    /** Implement the method in interface Cloneable
     * @return      copy of this vector as an object
     */
    public Object clone() {
        return copy();
    }
    public boolean contains( double x ) {
        for ( int i=0; i<n; i++ )
          if ( get(i) == x )
               return true;
        return false;
    }
    public final SsVector copy() {
        SsVector x = new SsVector(n );
        for ( int i=0; i<n; i++ )
        {
            x.a[i] = a[i];
        }

        return x;
    }
    // Return an string indicating an error for array dimensions
    private final String dimErrMsg() {
        return "Invalid dimensions: " + n;
    }
    private final String dimErrMsg( SsVector b ) {
        return "Dimensions not match: " + n + " <=> "
            + b.n;
    }
    public double elementAt(int i) {
       return a[i];
    }
    public boolean eq( SsVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
);

        for ( int i=0; i<n; i++ )
            if ( !(get(i) == b.get(i)) )
               return false;
        return true;
    }
    public double firstElement() {
       return a[0];
    }
    public final double get( int i) {
        return a[i];
    }
    public boolean isEmpty() {
         if (n == 0)
          return true;
        return false;
    }
    public double lastElement() {
       return a[n-1];
    }
    public double max() {
        double d = Double.MIN_VALUE;

        for ( int i=0; i<n; i++ )
            if ( get(i) > d )
               d = get(i);
        return d;
    }
    public double min() {
        double d = Double.MAX_VALUE;

        for ( int i=0; i<n; i++ )
          if ( get(i) < d )
               d = get(i);

        return d;
    }
    public boolean ne( SsVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
);
```

```
        for ( int i=0; i<n; i++ )
          if ( get(i) != b.get(i) )
                    return true;
        return false;
    }
    public final void removeElementAt( int i ) {
         for (int j = i; j < n - 1; j ++)
          a[j] = a[j+1];
         n = n-1;
    }
    public SsVector selfadd( SsVector b ) {
        if ( n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
);
        for ( int i=0; i<n; i++ )
            set( i , get(i) + b.get(i) );
        return this;
    }
    public SsVector selfmul( double f ) {
        for ( int i=0; i<n; i++ )
                setMul( i, f );
        return this;
    }
    /** Negate all elements of a Vector
     * @return      this Vector
     */
    public SsVector selfneg() {
        for ( int i=0; i<n; i++ )
            set( i,  -get(i) );
        return this;
    }
    public SsVector selfsub( SsVector b ) {
        if (  n != b.n )
            throw new IllegalArgumentException( dimErrMsg( b )
);
        for ( int i=0; i<n; i++ )
            setSub( i , b.get(i) );
        return this;
    }
    public final void set( int i,  double x ) {
        a[i] = x;
    }
    public final void setAdd( int i,  double x ) {
        a[i] += x;
    }
    public final void setMul( int i, double x ) {
        a[i] *= x;
    }
    public final void setSub( int i,double x ) {
        a[i] -= x;
    }
    /** Get number of elements
     * @return      the number of elements
     */
    public final int size() {
        return n;
    }
    /** Compute the product of this Vector by a scalar (C=fA)
     * @param f     scalar operand
     * @return      new product Vector
     */
    public SsVector times( double f ) {
        SsVector x = new SsVector( n );
        for ( int i=0; i<n; i++ )
          x.set( i, get(i) * f );
        return x;
    }
    /** Generate a new Vector that is the negative of this
Vector
     * @return      new negative Vector
     */
    public SsVector uminus() {
        SsVector x = new SsVector( n );
        for ( int i=0; i<n; i++ )
                x.set( i, -get(i) );
        return x;
    }

  public void print( PrintWriter output, int width, int
precision
                       ) {
        DecimalFormat fmt = new DecimalFormat();
        fmt.setDecimalFormatSymbols( new
DecimalFormatSymbols(Locale.US) );
        fmt.setMinimumIntegerDigits( 1 );
        fmt.setMaximumFractionDigits( precision );
        fmt.setMinimumFractionDigits( precision );
        fmt.setGroupingUsed( false );
        print( output, fmt, width );
    }

  public void print( PrintWriter output, NumberFormat format ,
int width) {
        for ( int i=0; i<n; i++ )
        {
                String str = format.format( get(i) );
                int space = width - str.length();
                output.print( ' ' );
                space--;

                for ( ; space>0; space-- )
                    output.print( ' ' );
                output.print( str );

        }
        output.println();
    }

    /** Compute the product of this Vector by a scalar (C=fA)
     * @param f     scalar operand
     * @return      new product Vector
     */
    public SsVector div( double f ) {
        SsVector x = new SsVector( n );
        for ( int i=0; i<n; i++ )
          x.set( i, get(i) / f );
        return x;
    }
}
```

8.22. blah.g

```
class SASSiLexer extends Lexer;
options {
k = 2;
charVocabulary = '\3'..'\377';
testLiterals = false;
exportVocab = SASSi;
}

{
  int line_error=0;

  public void reportError(String str)
  {
   super.reportError(str);
   line_error++;
  }
  public void reportError(RecognitionException excep)
  {
   super.reportError(excep);
   line_error++;
  }
}

protected ALPHA :   'a'..'z' | 'A'..'Z' | '_';
protected DIGIT :   '0'..'9';
WS :      (' ' | '\t')+ { $setType(Token.SKIP); } ;
NL :      ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
     { $setType(Token.SKIP); newline(); } ;

COMMENT :      ( "/*" ( options {greedy=false;} :NL | ~( '\n' |
'\r' ))* "*/"
          | "//" (~( '\n' | '\r' ))* NL)
          { $setType(Token.SKIP); } ;

LPAREN    : '(';
RPAREN    : ')';
LBRACE    : '{';
RBRACE    : '}';
LBRK      : '[';
RBRK      : ']';
ASGN      : '=';
COMMA     : ',';
MULT      : '*';
PLUS      : '+';
MINUS     : '-';
RDV       : '/';
GE        : ">=";
LE        : "<=";
GT        : '>';
LT        : '<';
EQ        : "==";
NEQ       : "<>";
INC       :"++";
DEC       :"--";
SEMI      :';';

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

NUMBER :  (DIGIT)+ ('.' (DIGIT)*)? (('E'|'e') ('+'|'-')?
(DIGIT)+)?
                  ;
```

```
STRING :  '"'!
               ( ~('"' | '\n')
               | ('"'! '"')
               )*
          '"'!
          ;




class SASSiParser extends Parser;

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

tokens {
     STATEMENT;
     FOR_LOOP;
     VAR_LIST;
     VECTOR;
     EXPRESSION_LIST;
     CONSTANT_STATEMENT;
     PROCEDURE_CALL;
        FOR_CON;
     UPLUS;
     UMINUS;
       }

{
  int line_error=0;

  public void reportError(String str)
  {
   super.reportError(str);
   line_error++;
  }
  public void reportError(RecognitionException excep)
  {
   super.reportError(excep);
   line_error++;
  }
}

program : (constant_group)?
```

```
 (statement | procedure_def )* EOF!
{ #program = #([STATEMENT,"PROG"], program); }
;

constant_group      : "const"^ (constant_statement)* "xconst"^
     ;

constant_statement  : assignment
                         {#constant_statement =
#([CONSTANT_STATEMENT,"CONSTANT_STATEMENT"],
constant_statement); }

     ;

statement
: for_loop
| if_statement
| return_statement
| load_statement
| assignment
| LBRACE! (statement)* RBRACE!
{#statement = #([STATEMENT,"STATEMENT"], statement); }
;


for_loop
          : "for"^ LPAREN! for_con RPAREN! statement
          ;

for_con
          : loop_init loop_cond loop_incr
          ;
loop_init
  : SEMI!
  { #loop_init = #([null, "null_init"]); }
  | assignment SEMI!
  ;

loop_cond
  : SEMI!
  { #loop_cond = #([null, "null_cond"]); }
  | expression SEMI!
  ;

loop_incr
  : ()
  { #loop_incr = #([null, "null_incr"]); }
```

```
    | assignment
    ;

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

expression:  relat_expression

          ;

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

load_statement
        : "include"^  LT! STRING GT! SEMI!
        ;

assignment
        : l_value ASGN^ arith_expression SEMI!

        ;

procedure_call_statement
        : procedure_call SEMI!
        ;

procedure_call
        : ID LPAREN! expression_list RPAREN!
            {#procedure_call =
#([PROCEDURE_CALL,"PROCEDURE_CALL"], procedure_call); }
        ;

expression_list
        : expression ( COMMA! expression )*
            {#expression_list =
#([EXPRESSION_LIST,"EXPRESSION_LIST"], expression_list); }
        |
            {#expression_list =
#([EXPRESSION_LIST,"EXPRESSION_LIST"], expression_list); }
        ;

procedure_def  : "procedure"^ ID LPAREN! var_list RPAREN!
procedure_body
     ;
```

```
var_list
        : ID ( COMMA! ID )*
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        |
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

procedure_body : LBRACE!(constant_group)? (statement)* RBRACE!
{#procedure_body = #([STATEMENT,"PROCEDURE_BODY"],
procedure_body); }
          ;

relat_expression
          : arith_expression ( (GE^ | LE^ | GT^ | LT^ | EQ^ |
NEQ^)arith_expression)?
          ;

arith_expression
          : arith_term ( (PLUS^ | MINUS^) arith_term )*
          ;

arith_term
          : arith_factor ( (MULT^ | RDV^) arith_factor )*
          ;

arith_factor
          : PLUS! r_value
                 {#arith_factor = #([UPLUS,"UPLUS"],
arith_factor); }
           | MINUS! r_value
                  {#arith_factor = #([UMINUS,"UMINUS"],
arith_factor); }
           | r_value
     ;


r_value
        : l_value
        | procedure_call
        | NUMBER
        | STRING

        | "true"
       | "True"
       | "TRUE"
       | "False"
       | "FALSE"
        | "false"
        | vector
        | LPAREN! expression RPAREN!
        ;

l_value
        : ID^ ( LBRK! index  RBRK! )*
        ;

index
        :arith_expression
            {#index = #([EXPRESSION_LIST,"INDEX"], index); }
        ;

vector
        : LBRK! arith_expression (COMMA! arith_expression)*
RBRK!
            {#vector = #([VECTOR,"VECTOR"], vector); }
        ;

cl_statement
        : ( statement | procedure_def )
        | "exit"
            { System.exit(0); }
        | EOF!
            { System.exit(0); }
         ;
```

8.23. blahwalk3.g

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

class SASSiWalker extends TreeParser;
options{
    importVocab = SASSi;
}

{
    static SASSiDataType null_data = new SASSiDataType( "<NULL>"
);
    SASSiInterpreter interp = new SASSiInterpreter();
}
```

```
expr returns [ SASSiDataType r ]
{
    SASSiDataType a, b;
    Vector v;
    SASSiDataType[] x;
    String s = null;
    String[] sx;
    r = null_data;
}
       :  #(GE a=expr b=expr)        { r = a.ge( b );
System.out.println(r);}
        | #(LE a=expr b=expr)        { r = a.le( b );
System.out.println(r);}
        | #(GT a=expr b=expr)        { r = a.gt( b );
System.out.println(r);}
        | #(LT a=expr b=expr)        { r = a.lt( b );
System.out.println(r);}
        | #(EQ a=expr b=expr)        { r = a.eq( b );
System.out.println(r);}
        | #(NEQ a=expr b=expr)       { r = a.ne( b );
System.out.println(r);}
        | #(PLUS a=expr b=expr)      { r = a.plus( b );
System.out.println(r);}
        | #(MINUS a=expr b=expr)     { r = a.minus( b );
System.out.println(r);}
        | #(MULT a=expr b=expr)      { r = a.times( b );
System.out.println(r);}
        | #(RDV a=expr b=expr)       { r = a.rfracts( b );
System.out.println(r);}
        | #(UPLUS a=expr)            { r = a;
System.out.println(r);}
        | #(UMINUS a=expr)           { r = a.uminus();
System.out.println(r);}
        | #(ASGN a=expr b=expr)      { r = interp.assign( a, b
); System.out.println(r);}
        | #(PROCEDURE_CALL a=expr x=vexpr)
            { r = interp.procedureInvoke( this, a, x );
System.out.println(r);}
        | #(VECTOR                   { v = new Vector(); }
             (a=expr                 { v.add( a );
System.out.println(a);}
             )*
           )
       {r = SASSiVector.joinVert(interp.convertExprList( v ) );}
        | num:NUMBER                 { r = interp.getNumber(
num.getText() ); System.out.println(r);}
```

```
        | str:STRING                 { r = new SASSiString(
str.getText() ); System.out.println(r);}
        | "true"                     { r = new SASSiBool( true
); System.out.println(r);}
     | "True"                     { r = new SASSiBool( true );
System.out.println(r);}
     | "TRUE"                     { r = new SASSiBool( true );
System.out.println(r);}
        | "false"                    { r = new SASSiBool( false
); System.out.println(r);}
        | "False"                    { r = new SASSiBool( false
); System.out.println(r);}
     | "FALSE"                    { r = new SASSiBool( false );
System.out.println(r);}
     | #(id:ID                    { r = interp.getVariable(
id.getText() ); }
              ( x=vexpr              { r = interp.subMatrix( r,
x ); }
              )*
           )
/*      | #("for" x=vexpr forbody:.)
            {
                SASSiInt[] values = interp.forInit( x );
                while ( interp.forCanProceed( x, values ) )
                {
                    r = expr( #forbody );
                    interp.forNext( x, values );
                }
                interp.forEnd( x );
            }
*/
        | #("if" a=expr thenp:. (elsep:.)?)
            {
                if ( !( a instanceof SASSiBool ) )
                    {
              return a.error( "if: expression should be bool" );
             }
                if ( ((SASSiBool)a).var )
                     {
               r = expr( #thenp );
               System.out.println(r);
              }
                else if ( null != elsep )
                    {r = expr( #elsep );
               System.out.println(r);
              }
            }
        | #(STATEMENT (stmt:. { if ( interp.canProceed() ) r =
expr(#stmt); System.out.println(r);}         )*)
        | #("return" ( a=expr        { r = interp.rvalue( a );
System.out.println(r);}
                     )?
           )                         { interp.setReturn( null );
}
        | #("procedure" procedure_name:ID sx=vlist probody:.)
             { interp.procedureRegister(
procedure_name.getText(), sx, #probody ); }
        | #("include" a=expr)        { interp.include( a );
System.out.println(a);}

        ;

vexpr returns [ SASSiDataType[] rv ]
{
    SASSiDataType a;
    rv = null;
    Vector v;
}
        : #(EXPRESSION_LIST         { v = new Vector(); }
                ( a=expr      { v.add( a ); }
                )*
           )                  { rv = interp.convertExprList( v
); }
        | a=expr              { rv = new SASSiDataType[1]; rv[0]
= a; }
/*      |  #(FOR_CON          { v = new Vector(); }
                ( s:ID a=expr { a.setName( s.getText() );
v.add(a); }
                )+
            )                 { rv = interp.convertExprList( v
); }
*/
        ;

vlist returns [ String[] sv ]
{
    Vector v;
    sv = null;
}
        : #(VAR_LIST     { v = new Vector(); }
              (s:ID      { v.add( s.getText() );
System.out.println(s);}
              )*
          )              { sv = interp.convertVarList( v ); }
        ;
```

8.24. permute.ssi

```
procedure permutation(n,r)
{
  a=0;
  b=0;
  c=0;
  d=0;

  a=n-r;
  b=factorial(n);
  c=factorial(a);

  if(c<0)
  {
    return -1;
  }
  else
  {
    d=b/c;
    if(d<=0)
    {
      return -1;
    }
    else
    {
      return d;
    }
  }
}


done=permutation(9,6);
print(done);
```

8.25. school.ssi

```
homework=0.2;
project=0.3;
bullsh_t=0.1;
midterm=0.15;
final=0.25;

homeworks=[36,24,39,27];
average=mean(homeworks);
```

```
print(average);
homeworks1=homeworks*2.5;
average=mean(homeworks1);
std=stdDev(homeworks);
print(std);
plot("bar",homeworks);

total=[homework,project,bullsh_t,midterm,final];
total=total*100;
plot("pie",total);
```