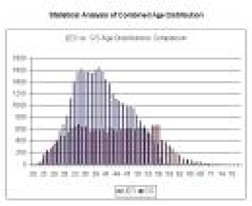
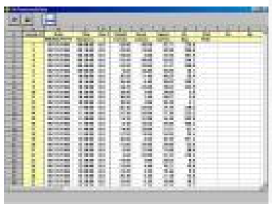
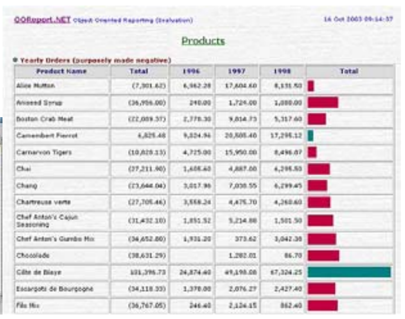
BECY

Tabular – data manipulation language







OOReport.NET Object Oriented Reporting (Evaluation) 16 Oct 2003 09:14:37

Products

● Yearly Orders (purposely made negative)

Product Name	Total	1996	1997	1998	Total
Alice Mutton	(7,301.62)	6,962.28	17,604.60	8,131.50	🟥
Aniseed Syrup	(36,956.00)	240.00	1,724.00	1,080.00	🟥
Boston Crab Meat	(22,009.37)	2,778.30	9,814.73	5,317.60	🟥
Camembert Pierrot	6,825.48	9,024.96	20,505.40	17,295.12	🟩
Carnarvon Tigers	(10,020.13)	4,725.00	15,950.00	8,496.87	🟥
Chai	(37,211.90)	1,605.60	4,887.00	6,295.50	🟥
Chang	(23,644.04)	3,017.96	7,038.55	6,299.45	🟥
Chartreuse verte	(27,705.46)	3,558.24	4,475.70	4,260.60	🟥
Chef Anton's Cajun Seasoning	(31,432.10)	1,851.52	5,214.88	1,501.50	🟥
Chef Anton's Gumbo Mix	(34,652.80)	1,931.20	373.62	3,042.38	🟥
Chocolade	(38,631.29)		1,282.01	86.70	🟥
Côte de Blaye	101,396.73	24,874.40	49,198.08	67,324.25	🟩
Escargots de Bourgogne	(34,118.33)	1,378.00	2,076.27	2,427.40	🟥
Filo Mix	(36,767.05)	246.40	2,124.15	862.40	🟥

Columbia University
Fall 2005

Members:

Bong Koh bdk2109@columbia.edu
Eunchul Bae eb2263@columbia.edu
Cesar Vichdo cv2139@columbia.edu
Yongju Bang yb2149@columbia.edu

1 . INTRODUCTON

1.1. Purpose

The language Becy is a scripting language to manipulate and present tabular data. The ultimate goal of such a language is to provide a simple and flexible scripting language tailored for numerical manipulations of lists of data. Since the goal of Becy is efficiency, it includes many shortcut constructs including implicitly typed variables, abbreviated conditional statement syntax, shortened loop manipulation syntax, and included basic mathematical functions.
Data files formatted in tab-separated values will have inherent access included within the language. Upon the processing of a data file, a program in the language will also be able to generate a formatted HTML file for use with a typical browser through presentation functions included in the keywords of the language.

1.2. Overview

Becy is meant to be a language with a simple interface but great efficiency. In addition, it is a portable and flexible language for tabular manipulations and computations.

The Becy is useful for such tasks as:

• **HTML output**
Aggregating information from text files that contain tables and creating output such as formatted HTML that will enhance users' readability and understanding of the output. Becy's compiler knows how to represent the output in a table structure, and show the result conveniently in HTML.

• **Portability across platforms**
Becy code is parsed and interpreted using the Java language thus is compiled into Java byte code which provides platform independency. Therefore, Becy can be developed on any platform that has the Java compiler, the JVM, and the GUI. Java is a highly portable language thus resulting in the highly portable Becy translator and code. This fact will make distributing software easier.

• **Translating files from one format to another.**

• **Creating small databases.**

• **Performing mathematical operations on files of tabular data.**

1.3. Basic Features

- **Data Types**

Becy Consists of only 4 kinds of data type to ease the complications associated with complicated data types and restrictions. These data types are real numbers (BecyNumber), strings (BecyString), truth values (BecyBool) and tables (BecyTable).

- **Ease-of-use**

Efficiency is the key to Becy as well as its simple Interface and syntax

- **Simple and Light**

Becy keeps its library relatively small, and provide users with more memory space for data manipulation and computation.

- **Built-In Functions**

Becy contains a variety of implicitly implemented functions built into the language itself. Please see [Built-in function in Becy] section.

- **User-Defined Function**

Becy allows the user to define functions providing programming flexibility and ingenuity.

- **Basic table operations and math functions**

Provide most existing arithmetic operations and math functions.

- **Program flow control**

Like other languages, Becy provides users with basic programming language control of events using conditionals and iterations. But to provide the programmer with the most possible convenience, the syntax has been shortened to easy and unique conditional statements and loops

1.4 Models of Computation

Basically, the primary model of computation for Becy is to emulate a simple yet useful scripting language. The ANTLR tool was used to create the lexer and parser from the appropriate grammar files. The lexer and parser were then used to create the Abstract Syntax Tree which was processed by a parse tree walker also written in ANTLR. In the walker, a good deal of functionality and work was developed in an external Interpreter class which also utilized other helper classes. The basic Table types, Number types, String types, Boolean types and the corresponding basic operators have been represented within the parse tree and the walker. However, much of the code for functions, both user-defined and built-in were written in another collection of Java source code. User-defined functions allow a programmer the ability to manipulate the tables and data sources in imaginative ways not implemented natively in the Becy System. Through the process, the final output is accumulated in another class for output and finally produced in an HTML file.

2. TUTORIAL

Becy is useful for simple and quick computational programming for tabular data. Anybody who can write a C program with basic knowledge of language concepts and programming techniques can use Becy with little to no difficulty.

2.1. Getting Started

Now, let's get started. To start Becy, you should verify that all the Becy classes are available and the Java classpath variable is set properly for the correct access. After compiling BecyMain.java and its dependencies, you will eventually enter the following command to parse and process your data file written in Becy:
(Please note that any data source files for inclusion in your program must have read access by Java.)

```
> java BecyMain [your_data_file.becy]
```

Let's start with the simple yet vitally important assignment statement. If you want to assign a value into a variable, then just use the usual assignment symbol, '='

2.2 Variable Declarations

```
Becy> $a = 4;
Becy> $b = $a;
Becy> print $b;
4

Becy> @string_var = “Hello world!”;
Becy> print @string_var;
Hello world!
```

Suppose you want to know the condition of some statement and give some other operations based on the condition, then use the specified Becy conditional syntax (or more commonly referred to as the ternary operator)

(conditional_stmt) ? stmt : stmt ;

```
Becy> ( $a > 1 ) ? $b = 1 : $b = 2;
Becy> print $b;
1
```

In addition, you can use other comparative operations traditionally used with Boolean algebra such as >=, <=, >, <, ==, !=, && (AND), || (OR) in the conditional_stmt. Becy also will support compound conditional statements or nested conditions.

```
        Becy> ($a > 1) ? ($b > 1) ? $c = 1 : $c = 2 : ($b > 1) ? $c = 1 : $c = 2 ;
```

Also, you can use more complicated Boolean algebra for more intricate comparisons:

```
        Becy> ( ( $a > 1 ) == (($b + 1) > 10) ) ? ( $b > 1 ) ? $c = 1 : $c = 2 : ( $b >
1 ) ? $c = 1 : $c = 2 ;
```

Let us now move onto more useful constructs such as looping with Becy's "For" loop.   Here is an example:

```
         for $a TO $b {
            $c = $c + 2;
            $a = $a * 5;
            object1.foo ($a, $c);
          };
```

In addition to those typical operational structures, Becy also supports user-defined functions.   We can define a new function by preceding a block of statements with the keyword "function" and the name of the function defined as so:

```
        function foo1 ($a, $b, 3)
        {
          $a = 1;
          $b = $a;
        };

        function foo2 ($a, $b, 3)
        {
          $a = 1;
          $b = $a;
          ($a == 1) ? ($b == 1) ? $c = 1 : $c = 2 : ($a > 1) ? $a = 1 : $a = 2 ;
        };

        function foo3 ($a, $b, 3)
        {
          $a = 1;
          $b = $a;
          for $a TO $b {
            $a = $a;
            $a = 1;
            object1.foo ($a TO $b);
          };
        };
```

2.3 BecyTable operations

Now, let’s play with the table support in Becy. If you want to import data into the Becy system, then you have to specify the table_object_name for the file and use the keyword include with the file_name as a string enclosed in quotation marks.

```
        Becy> object1 { include "somefile.txt" };
```

Or, you can initialize the table_object with some particular data in mind in a Becy-supported format including the appropriate white space between elements and commas separating rows of the data.

```
        Becy> object1 { Napolean 88 67 43 57 89, Ernie 100 100 100 98 99 };
```

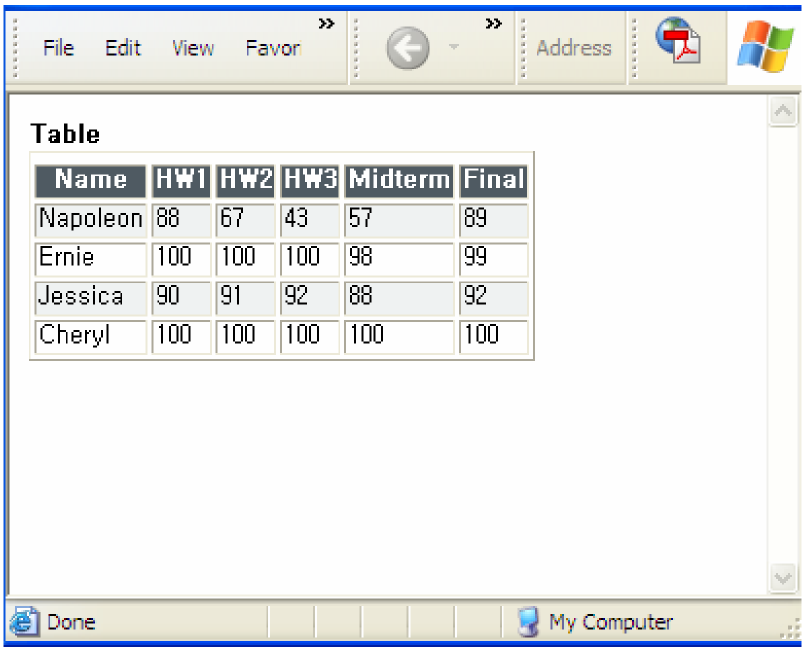
It is easy to use Becy from the command line to perform simple operations on text files. Suppose I have a file named "TestInput_datasource.txt" that describes a grade collection. Each line in the file contains the following information:

Name	HW1	HW2	HW3	Midterm	Final
Napoleon	88	67	43	57	89
Ernie	100	100	100	98	99
Jessica	90	91	92	88	92
Cheryl	100	100	100	100	100

```
        Becy> $a = object1.sum ($c, true);
        Becy> $b = object1.average ($d, true);
        Becy> print $a;
        380
        Becy> print $b;
        94.5
```

If you want to print out the whole table, then use print method with a table_object. The resulting output will show you the whole table in HTML.

```
        Becy> print object1;
```



Table

Name	HW1	HW2	HW3	Midterm	Final
Napoleon	88	67	43	57	89
Ernie	100	100	100	98	99
Jessica	90	91	92	88	92
Cheryl	100	100	100	100	100

```
Becy> $a = object1.max($i, true);
Becy> $b = object1.min($i, true);
Becy> $c = object1.median($i, true);
Becy> print $a;
100
Becy> print $b;
88
Becy> print $c;
95
```

Becy will support most arithmetic functions typically used for tabular data source manipulation such as sum, average, standard deviation etc.

2.4 Built-in functions

Becy provides several built-in functions that are normally associated with lists and tables of data. Thus we provide the functions built into the language to handle these data manipulations.
The following is a list of those functions that are pre-built. As can be seen from the syntax, all built-in functions are invoked in a "pre-fix" manner. You should follow the Becy syntax to specify the row or column identifying number and then the orientation as dictated by a single Boolean value (for example: $d, true)

1) table_object.sum (row or column, bool)
computes the summation of the specific row or column that you specify based on Becy syntax and return the result as real number.

2) table_object.sum2 (row or column, bool)

computes the summation of square of each entry in the specific row or column and return the result as real number.

3) table_object.average (row or column, bool)
computes the average of the specific row or column and return the result as real number.

4) table_object.median (row or column, bool)
return the median entry from the specific row or column.

5) table_object.sd (row or column, bool)
computes the standard deviation of the specific row or column and return the result as real number.

6) table_object.var (row or column, bool)
computes the variation of the specific row or column and return the result as real number.

7) table_object.max (row or column, bool)
return the maximum number among the entries in the specific row or column.

8) table_object.min (row or column, bool)
return the minimum number among the entries in the specific row or column.

9) table_object.round (row or column, $index)
return the rounded number for the specific entries in the row or column. Index specifies the position in the row or column.

9) table_object.sqrt (row or column, $index)
return the square-rooted number for the specific entries in the row or column. Index specifies the position in the row or column.

10) table_object.ceil (row or column, $index)
return the rounded-up number for the specific entries in the row or column. Index specifies the position in the row or column.

11) table_object.floor (row or column, $index)
return the rounded down number for the specific entries in the row or column. Index specifies the position in the row or column.

12) table_object.abs (row or column, $index)
return the absolute value for the specific entries in the row or column. Index specifies the position in the row or column.

13) table_object.power (row or column, $index, $power)
return the number raised to the appropriate exponent for the specific entries in the row or column. Index specifies the position in the row or column.

3. Manual Reference.

The language constructs will be explained by the usual extended BNF(EBNF) using special notation. The “?” and “[ ]” means an optional item. The ‘+’ and ‘*’ means repeat 1 or more times, and 0 or more times, respectively. Non-terminals are shown in italics, keywords are shown in bold, and other terminal symbols are enclosed in single quotes like ‘.’ and ‘;’.

1 - Lexical Conventions

1.1 comments

Comments starts with the characters /* and ends with the characters */.

1.2 Identifiers (Names)

Identifiers in Becy can be any string of letters, digits, and underscores, not beginning with a digit. This coincides with the definition of identifiers in most languages. Becy is case-sensitive. Upper and lower case letters are considered differently.

1.3 Keywords

The following keywords are reserved and cannot be used as identifiers:

return	function	print	for	TO	include
true	false				

1.4 Constants

1.4.1 Numerical constants

Numerical constant is a sequence of digits. It may be written with an optional fraction part. Examples of valid numerical constants are

3 3.0 3.1416

1.4.2 Character constants

Character constants are 1 or 2 characters enclosed in single quotes “ ‘ “. It may contain the number and single symbol. Like C, 2 characters with backslash “\” within matched single quotes are considered special flag.

\n --- newline
\r --- carriage return
\t --- horizontal tab
\v --- vertical tab

1.5 Literal Strings

String will be defined as a sequence of characters confined within matched double quotes. Literals may contain nested " "" " pairs. It may run for several lines and don't interpret any special flag.

1.6 Other tokens

```
+     -     *     /     =   ||
!=    <=    >=    <     >     ==
(     )     {     }     &&    %
;     :     ,     .
```

2. Values and types.

There are three basic types in Becy: boolean, number, and string. Variables will not have explicit type declaration since an indicator such as "@"and "$" will be used preceding any variable. (see. 3.1)

2.1 Boolean
Boolean is the type of the values false and true.

2.2 Number
Number basically represents real numbers. There is no explicit type declaration. (see. 3.1)

2.3 String
String represents a sequence of characters. There is no explicit type declaration. (see. 3.1)

3. Variables

3.1 Basic types of variables
Variables are places that stores values. Becy will utilize two basic types of variables, numbers and strings. However, it will not have explicit type declaration since it will use an indicator character preceding any variable. For numerical variables, the character will be the '$' character and '@' character will be used for string variables. For example, to declare and assign a new string variable and a new numerical variable the values "hello" and 1234, one would write:

```
@string_var1 = "hello";
$num_var1 = 1234;
```

However for Becy's most useful data type, there is no special character preceding the data tables.

4. Statements

Becy supports almost conventional set of statements but in a special form. Basically, the statements are executed in sequence. This set includes assignment, control structures, looping structure, function calls, and table constructors.

4.1 Expression statement
Most statements are expression statements, which have the form

expression ;

Usually, expression statements are assignments or function calls

4.1.1 Assignment
Assignment can take only one form using the symbol "=". This assignment is a direct assignment in the form of:

left-value expression = right-value express;

Left-value must always be a variable whereas the right side can be a constant, variable, the result of an operation or any combination of them. It is necessary to emphasize that the assignation operation always takes place from right to left and never at the inverse.

4.1.2 Function call
Functions in Becy require explicitly identified invoking list objects. (see 6)

4.2 Conditional statement
Becy will also contain support for conditionals only in this manner;

(expression) ? statement1 : statement2 ;

If true, then first statement is evaluated, otherwise second statement.

4.3 Loop statement
Like any other languages, Becy will provide a looping structure. However, since lists are a vital component of Becy's design, its looping syntax reflects the necessity for a convenient method to use lists;

```
for startrow TO endrow
{ statements };
```

4.4 Return statement
Return statement can be used within function body. It returns either a variable's value or a number including null value to the caller. It is mandatory that the statement is ended with a semicolon. A basic return statement as follows:

return ;
return (expression) ;

5. Functions

Functions in Becy require explicitly identified invoking list objects. Thus, the general syntax for using a function follows this format:

```
list_name.func_name ( parameter_list );
```

5.1 Included mathematical functions
Basic mathematical functions on groups of data will be supported by Becy;

```
list_name.average($column_number);
list_name.sd($column_number);
list_name.median($column_number);
list_name.var($column_number);
list_name.sum($column_number);
list_name.max($column_number);
list_name.min($column_number);

list_name.round($column_number, $row_number);
list_name.sqrt($column_number, $row_number);
list_name.fact($column_number, $row_number);
```

5.2 Included display functions

```
print object;
```

5.3 Included data definition, and import functions
Basic data definition and import functions on groups of data such as defining data sources loading a data source will be supported by Becy. All such functions follow the typical function call syntax but will use curly braces '{' and '}' instead of parenthesis:

Definition:

```
list_name {
          data_col1_row1          data_col2_row1,
          data_col1_row2          data_col2_row2
};
```

Import:

```
list_name { include "file_name.txt" };
```

5.4 Manipulation functions.

Adding and removing from the source will provided as follows:

Addition:

```
list_name.push{bob 10 3, kim 6 2};
list_name.push{include "file_name.txt"};
```

Deletion:

```
list_name.remove_row{row1, row2 . . . rown};
list_name.remove_col{ col1, col2 . . . coln };
```

5.5 User-defined Functions

Declaration:

```
function func_name {

        statement1;
  ...
        statementn;
        [return value];
}
```

Invocation:

```
listname.func1(); listname.func2(); …
```

6. Expressions

6.1 Arithmetic Operators

Becy supports the usual arithmetic operators: Unary arithmetic operators "+" and "-" can be prefixed to an expression. The binary "+" (addition), "-" (subtraction), "*" (multiplication), and "/" (division); If the operands are numbers, operations have the usual meaning.

6.2 Relational Operators
The relational operators in Becy are

```
== != < > <= >=
```

These operators always result in false or true. Equality “==” first compares the type of its operands. If the types are different, then the result is false. Otherwise, the values of the operands are compared. Numbers and strings are compared in the usual way. These operations can be applied to tables only if both tables have same numbers of fields, and ‘corresponding’ fields have the same type.

The operator “!=” is exactly the negation of equality “==”.
The order operators work as follows. If both arguments are numbers, then they are compared as such. Otherwise, if both arguments are strings, then their values are compared according to the current locale.

6.3 Logical Operators
The logical operators in Becy are

```
&&      ||
```

Like the control structures all logical operators consider both false and null as false and anything else as true. The conjunction operator “&&”returns its first argument if this value is false or null; otherwise, returns its second argument. The disjunction operator “||” returns its first argument if this value is different from null and false; otherwise, “||” returns its second argument.

6.4 Precedence and Associative.
Operator precedence in Becy follows the below, from lower to higher priority:

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

You can use parentheses to change the precedence in an expression. All other binary operators are left associative.

7. Scope rules

7.1 Lexical scope

Becy is a language with only global scope.

It may result in a program error to re-declare identifiers already declared in the current context because Becy will naturally clobber any pre-defined variable by assigning a new value without complaint. This ease of variable usage provides faster and simpler programming.

7.2 Example of lexical scope

```
$number = 10;
@name = “lucas”;

    function age {
              $number=5;
return $number;
};

 $number2=$number+10;
```

8. Examples

NAME	GRADE1	GRADE2
jean	10	2
peter	8	1
josh	9	3
amber	9	6

```
/*
This is an example of code
*/

/* Adding values to the list*/
list{jean 10 2, peter 8 1, josh 9 3,
   amber 9 6}

/* Sum of grade1 */
$sum = list.sum($a, true);

/* Average of grade 1 */
$avg1 = list.average($a, true)

/* Average of grade 2 */
$avg2 = list.average($a, true)

/* conditional expresion*/
$max_avg = ($avg1>$avg2)?$avg1:$avg2

/* function example*/
function max($n1,$n2){
 $max=$n1;
 ($avg1<$avg2)?$max=$n2;
 return $max;
}

$max_avg = list.max($avg1,$avg2);
```

4. PROJECT PLAN

4.1 General Project Processes

The project development has basically followed the course schedule through out the term. Most of the language specification and its design were decided on during the proposal phase of the project; a few modifications were added during the writing of the Language Reference Manual. We also had a few additional on-going minor adjustments and changes to these decisions during the development phase. Thus we see there are few changes made since submitting the original Language Reference Manual.

4.2 Coding Style

To ease the development process in a multiple team-member working environment, we had to ensure certain things on how to code and develop in a coordinated fashion.

1) Code should be clearly defined and well-commented or clearly specified during team discussions.

2) If any modifications or updates were made, the coder has a responsibility to explain what has been changed and why. It is always helpful to inform other team members what has been changed unless it is trivial.

4.3 Project log

We had set several milestones according to specific deadlines. We had Friday afternoon formal team meeting set up weekly to discuss any occurring problems, and solve any troubles.

The following was the project's timeline:

Proposal	09/30/05
Grammar specifications	10/05/05
Syntax specifications	10/20/05
Reference Manual	10/22/05
Lexer and Parser	11/04/05
Tree walker	11/12/05
Interpreter backend	11/23/05
Code complettion	12/07/05
Final tests and debugging	12/14/05
Final Report	12/20/05

4.4 Team Responsibilities

It was expected that everyone in the group would help in documentation and would help each of the people below with their responsibilities. The table below indicates what each person actually was responsible for and completed. While specific duties were identified, much of the work was shared and worked on by all.

Group Member Responsibilities
Bong Koh: Team leader, grammar design, tree walker construction and testing
Eunchul Bae: Core function implementation, testing, report writing and documentation.
Cesar Vichido: Graphics, table data structure, functions, testing.
Yongju Bang: Parser, implicit language function development, testing and documentation

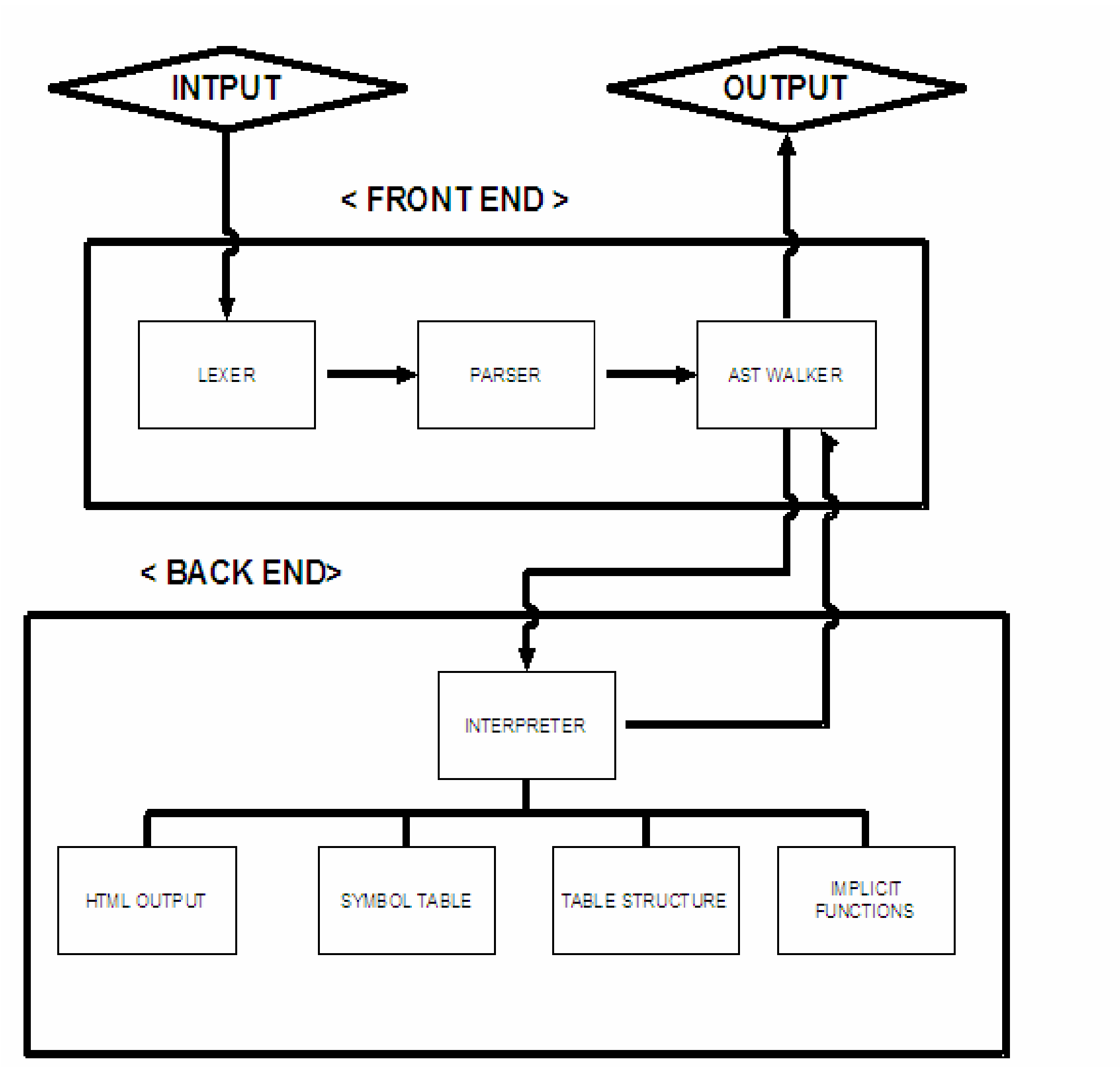
Documentation was generally a group effort. Language Reference Manual and The Final Report were mostly written by Yongju Bang and Eunchul Bae.

4.5 Development environment

All development was done on Windows XP machines. All of the lexer, parser and tree walker were specified using the ANTLR language and generated by the ANTLR tool using Java version 1.4.2. The backend of the interpreter was written in Java, also using Java version 1.4.2.

5. ARCHITECTUAL DESIGN

5.1. Diagram for Becy



5.2 Description of Architecture

There are five main components of the Becy translator and interpreter: lexer, parser, tree walker for semantic analysis, interpreter for execution of more complicated code, and other developed back-end classes used primarily for table manipulation and HTML output.

A user inputs a .becy file to the translator. The first thing that the translator does is use the lexer generated from ANTLR on that input file to produce the tokens that were defined in the grammar.g file. The tokens identified by the lexer are then passed to the parser also defined in the grammar.g file and produced by ANTLR. The parser receives tokens from the lexer as is necessary as it parses the .becy program file to match the file's contents to the grammar rules that we have defined. If there are syntactical errors, the parser reports these errors to the user via standard out and stops parsing the program. If the inputted source file is syntactically correct, the parser creates an Abstract Syntax Tree or AST from the grammar rules. This AST is the main data structure that is used to test both the semantics of the input file and to process the actual program within the file.

Once the AST has been created, it is passed to the tree walker which runs semantic analysis and type checking. This parse tree walker is also defined in the ANTLR language and embedded Java in order to process the AST given by the parser. The walker makes one pass over the AST and creates a single symbol table given the established global scooping of variables. While doing so, it also executes the nodes' contents in a top-down fashion, handling most of the basic operations such as arithmetic or logic while passing on function calls, definitions, data imports etc. to the interpreter. In addition, the language implemented built-in functions can be called during this walk. Some of the many things that the walker checks are: type-checking, appropriate arguments to functions, correct argument structure. Moreover, the static semantic checker examines the following: invalid assignments, mismatched return types, undefined variables or functions, expression checking, function argument checking, logical expression checking, built-in function checking, and java reserved word testing. If the walker detects any errors, it reports these errors to standard out or in some severe cases will cease execution of the program. For the duration of this walk through the tree, the program can be outputting to standard out and also feeding the HTML generator its input. At the conclusion of the program, the HTML is accumulated and output in a aesthetically pleasing format given the Becy's allowance of HTML formatting in its code.

6. TESTING

6.1 Overview

The purpose of our language, BECY is to manipulate and present tabular data. The testing includes BECY's lexical analyzer, parser, tree walker and the backend. The test is essential to ensure that BECY's operations produce correct output of manipulation data given by users.

6.2 Unit Testing

Before going into code testing, the first level of test is the 'unit level'. We test both incrementally and recursively each unit (lexical analyzer, parser, tree walker and the backend).

<LEXICAL AND PARSING ANALYZER>
We used a variety of statements in our intended language to test the rigor of our grammar specification. We used a testing driver to output a parenthesized parse tree representation to analyze that our language was parsing correctly. The following is our testing suite:

```
$a = $a;
$a = 1;
$a = ( 1 + 2 );

( $a > 1 ) ? $b = 1 : $c = 2;
( $a == 1 ) ? $b = 1 : $c = 2;
( true || false ) ? $b = 1 : $c = 2;
( ($a < 1) && true ) ? $b = 1 : $c = 2;
( ($a < 1) && ($b >= 1) ) ? $c = 1 : $d = 2;
( $a == 1 ) ? ( $b == 1 ) ? $d = 1 : $e = 2 : ( $c > 1 ) ? $f = 1 : $g = 2 ;
( ($b >= 1) == ( ($c+1) > ($d*10) ) ) ? $a = 1 : $e = 2 ;
( ($a < 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? ( $b == 1 ) ? $d = 1 : $e = 2 :
( $c > 1 ) ? $f = 1 : $g = 2 ;

for $a TO $b {
  $a = $a;
  $a = 1;
  object1.foo ($a TO $b);
};


function foo1 ($a, $b, 3)
{
  $a = 1;
  $b = $a;
};

function foo2 ($a, $b, 3)
{
  $a = 1;
  $b = $a;
( $a == 1 ) ? ( $a == 1 ) ? $a = 1 : $a = 2 : ( $a > 1 ) ? $a = 1 : $a = 2 ;
};

function foo3 ($a, $b, 3)
{
  $a = 1;
  $b = $a;
  for $a TO $b {
    $a = $a;
    $a = 1;
    object1.foo ($a TO $b);
  };
};

object1 { include "somefile.txt" };
object2 { 1 2 3 "bob", 4 5 6 "jim", 7 8 9 "mary"};
object3 { 3 "bob", 6 "jim", 9 "mary"};

object1.foo (1,2,3);
object1.foo ($a TO $b);

$a = object1.sum ($a TO $b);

print object1;
print $a;

( 1 == 2 ) ? object1.copy(object2) : print object1;

*****************************************************************
            BAD CASES (Cases that should fail):
*****************************************************************

bad_object { $a $b };
( ($a + 1) && true ) ? $a = 1 : $a = 2;
```

<TREE WALKER>
For the tree walker, we began using simple program examples and iteratively escalated the computations required in each test program. The goal was to ensure that the walker was correctly processing a given program in both its parse tree and its final output. Here are the test programs used:

```
$a = 1;
$b = ( 1 + 2 );

print "A: ";
print $a;
print "B: ";
print $b;

$a = 1;
$b = 5;
$c = 0;

for $a TO $b {
  $c = $c + 1;
  print "Iteration:";
  print $c;
};

print $a;
print $b;


$a = 0;
$b = 2;
$c = 10;
$d = 0;

( ($a > 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? $d = 1 : $d = 2 ;

print "$d should be 2.";
print "$d:";
print $d;

$a = 0;
$b = 2;
$c = 10;
$d = 1;

( ($a < 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? ( $b == 1 ) ? $d = 1 : $e = 2 :
( $c > 1 ) ? $f = 1 : $g = 2 ;

print "$g should be 2.";
print "$g:";
print $g;
```

<THE BACKEND>

The interpreter connected the walker and the vital backend components of the translator, the HTMLOutput/Symbol Table/ Table Structure/ Implicit Functions. It maintained much of the details required to correctly execute the more difficult constructs of the Becy language such as the functions and also the output.

6.3 Testing Samples Codes

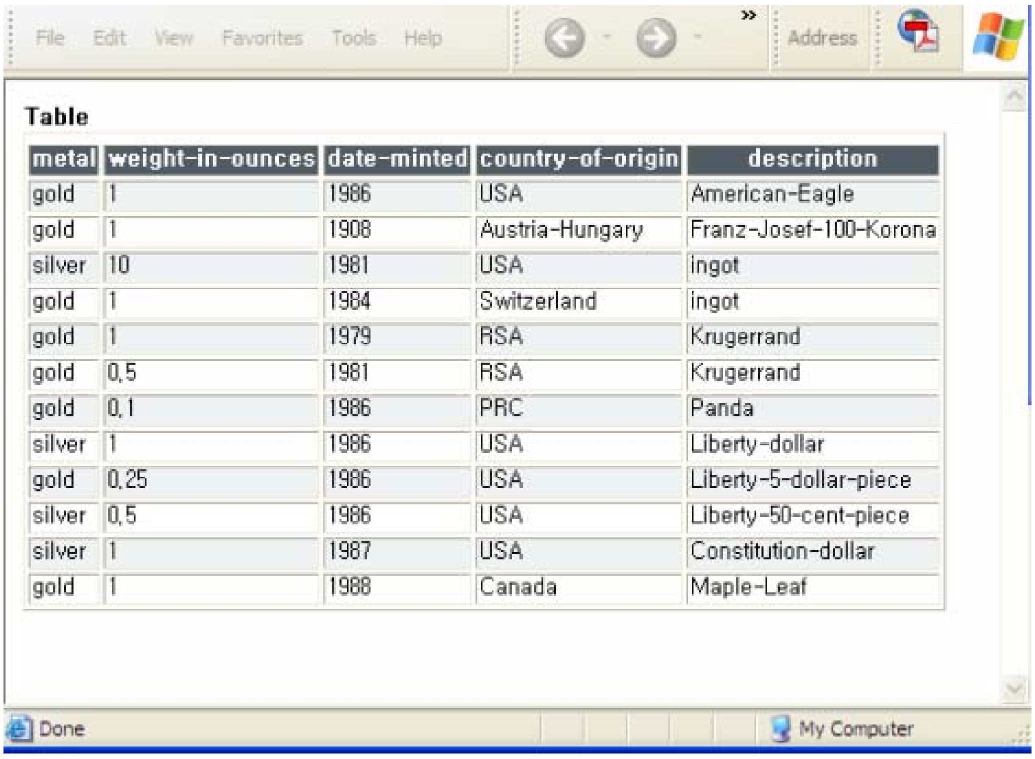
At the last step of our language implementation, we took some sample tests to ensure and demonstrate the correctness and capabilities of our language. These samples show us the functionalities defined in the language working in a complex. These programs also enabled us to gain insight into our language design and to see several benefits and shortcomings to the decisions we made.

Suppose there is a files named "TestInput_datasource2.txt"

```
metal   weight-in-ounces   date-minted   country-of-origin   description
------------------------------------------------------------------------------------------
gold      1                      1986          USA              American-Eagle
gold      1                      1908     Austria-Hungary    Franz-Josef-100-Korona
silver   10                      1981          USA                   ingot
gold      1                      1984      Switzerland               ingot
gold      1                      1979          RSA                Krugerrand
gold     0.5                     1981          RSA                Krugerrand
gold     0.1                     1986          PRC                  Panda
silver    1                      1986          USA               Liberty-dollar
gold     0.25                    1986          USA            Liberty-5-dollar-piece
silver   0.5                     1986          USA              Liberty 50-cent-piece
silver     1                     1987          USA             Constitution-dollar
gold      0.25                   1987          USA       Constitution-5-dollar-piece
gold       1                     1988        Canada               Maple-Leaf
--------------------------------------------------------------------------------------------
```

```
        Becy> object1 { include " TestInput_datasource2.txt" };
        Becy> print object1;
```

Will show the below table.



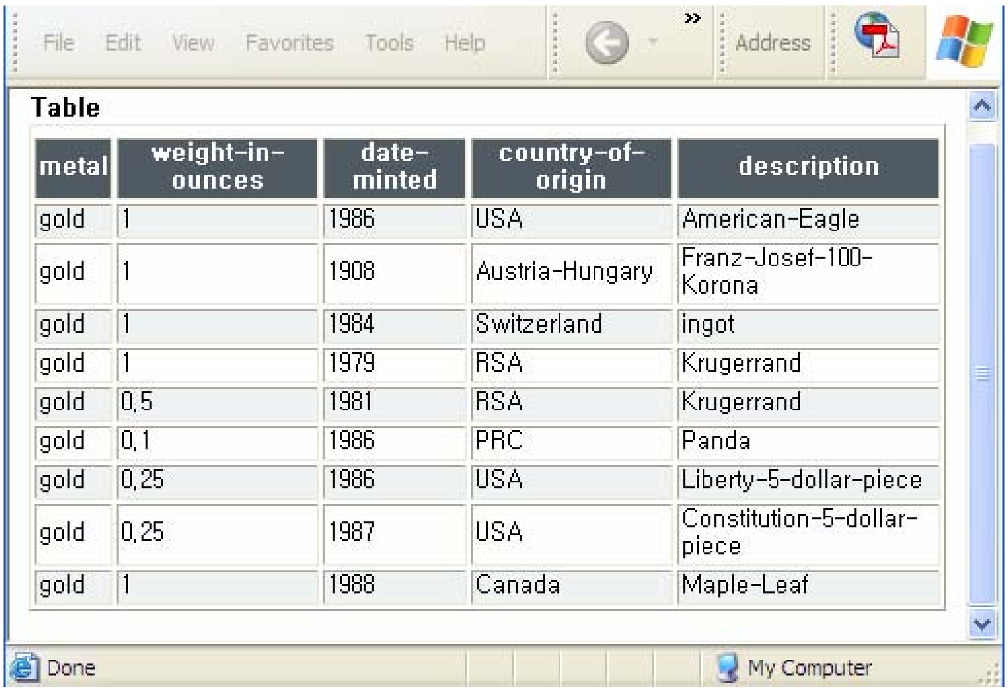
Table

metal	weight-in-ounces	date-minted	country-of-origin	description
gold	1	1986	USA	American-Eagle
gold	1	1908	Austria-Hungary	Franz-Josef-100-Korona
silver	10	1981	USA	ingot
gold	1	1984	Switzerland	ingot
gold	1	1979	RSA	Krugerrand
gold	0.5	1981	RSA	Krugerrand
gold	0.1	1986	PRC	Panda
silver	1	1986	USA	Liberty-dollar
gold	0.25	1986	USA	Liberty-5-dollar-piece
silver	0.5	1986	USA	Liberty-50-cent-piece
silver	1	1987	USA	Constitution-dollar
gold	1	1988	Canada	Maple-Leaf

```
HTMLOutput h = new HTMLOutput();
h.createHTMLFromPool(HTMLPool.getArray());
```

This tells the program to search through the file for lines of text that contain the string "gold", and print them out. The result is:

```
gold    1     1986  USA                  American-Eagle
gold    1     1908  Austria-Hungary      Franz-Josef-100-Korona
gold    1     1984  Switzerland          ingot
gold    1     1979  RSA                  Krugerrand
gold    0.5   1981  RSA                  Krugerrand
gold    0.1   1986  PRC                  Panda
gold    0.25 1986  USA                  Liberty-5-dollar-piece
gold    0.25 1987  USA                  Constitution-5-dollar-piece
gold    1     1988  Canada               Maple-Leaf
```



Table

metal	weight-in-ounces	date-minted	country-of-origin	description
gold	1	1986	USA	American-Eagle
gold	1	1908	Austria-Hungary	Franz-Josef-100-Korona
gold	1	1984	Switzerland	ingot
gold	1	1979	RSA	Krugerrand
gold	0,5	1981	RSA	Krugerrand
gold	0,1	1986	PRC	Panda
gold	0,25	1986	USA	Liberty-5-dollar-piece
gold	0,25	1987	USA	Constitution-5-dollar-piece
gold	1	1988	Canada	Maple-Leaf

7. LESSONS LEARNED

These are few lessons that we had learned during the semester designing and implementing our programming language project. These lessons will be helpful for the future developers or we can take into consideration for other projects in the future.

1. Creativity:

There are two many languages and programs out there for specific purposes, and it is not easy to design a unique programming language that provides useful functionalities to satisfy users. So we tried to focus on designing what our language can do to differentiate itself from other existing languages. We also tried to be creative with our syntax to make the language more efficient and user-friendly.

2. Start Early:

This applies to every team member, I guess. We had learned that with the limited time given between the completion of seeing the abstracts of a translator in class to actually completing a working translator, we should definitely be on the ball in the initial states.

3. Implementing.

Most of all, getting advice and guidelines from those who had already been through a similar experience was very helpful. So we worked with our wonderfully informative and nice TA whenever we were stuck with problems, or when we wanted to step forward to each level of the programming development. We also realized that a small change in the beginning stage can bring a huge difference in the end so that we took extra precautions on defining our grammar.

4. Testing

Many factors should be considered when testing as we have experienced that some very big problems come from something which seem very trivial, or that was thought to not be difficult to deal with.

8. CODE

BecyBool.java

```
public class BecyBool extends BecyDataType{

        boolean bool;

        public BecyBool(boolean bool) {
                this.bool = bool;
        }

        /**
          * returns the string which specifies its type
          * @return return an string representing the type
          */
        public String getType()
        {
                return "BecyBool";
        }

        /**
          * Prints it's boolean value
          */
        public void print()
        {
                System.out.println( bool ? "true" : "false" );
                HTMLPool.add( bool ? "true" : "false" );
        }

        /**
          * return the string representation of its value
          * @return return an string representing its value
          */
        public String toString()
        {
                return new Boolean(bool).toString();
        }

        /**
          * return the boolean value
          * @return return an boolean value
          */
        public boolean getBoolean()
        {
                return bool;
        }

        /**
```

```
      * Set a boolean value
      * @param bool boolean value to be set
      */
     public void setBoolean(boolean bool)
     {
             this.bool = bool;
     }

     /**
      * Assign a boolean value
      * @param b BecyBool to be assigned
  * @return return the boolean value which is BecyBool type
      */
     public BecyBool assign(BecyBool b)
     {
             setBoolean(b.getBoolean());
             return this;
     }

     /**
      * Logical operator "&&"
      * @param bdt BecyDataType to be operated
  * @return return the boolean value which is BecyBool type if the param is consistent
      *            otherwise call exception with operator which is in parent class
      */
public BecyDataType and (BecyDataType bdt)
     {
      if (bdt instanceof BecyBool)
           return new BecyBool( bool && ((BecyBool)bdt).bool );
      return exception( bdt, "and" );
}

     /**
      * Logical operator "||"
      * @param bdt BecyDataType to be operated
      * @return return the boolean value which is BecyBool type if the param is consistent
      *            otherwise call exception with operator which is in parent class
      */
public BecyDataType or(BecyDataType bdt)
     {
      if (bdt instanceof BecyBool)
           return new BecyBool( bool || ((BecyBool)bdt).bool );
      return exception( bdt, "or" );
}

     /**
      * returns its negation value
      * @return return the boolean value which is BecyBool type
      */
public BecyDataType not()
     {
             return new BecyBool( !bool );
        }

            /**
              * Logical operator "=="
              * @param bdt BecyDataType to be operated
              * @return return the boolean value which is BecyBool type if the param is consistent
              *              otherwise call exception with operator which is in parent class
              */
            public BecyDataType eq( BecyDataType bdt )
                            {
            if ( bdt instanceof BecyBool )
                    return new BecyBool( bool == ((BecyBool)bdt).bool );
            return exception( bdt, "==" );
        }

            /**
              * Logical operator "!="
              * @param bdt BecyDataType to be operated
              * @return return the boolean value which is BecyBool type if the param is consistent
              *              otherwise call exception with operator which is in parent class
              */
        public BecyDataType ne( BecyDataType bdt )
            {
            if ( bdt instanceof BecyBool )
                    return new BecyBool( bool != ((BecyBool)bdt).bool );
             return exception( bdt, "!=" );
        }
}
```

BecyData.java

```
public class BecyDataType {

      String name;      // used in hash table

      public BecyDataType(){}

      public BecyDataType(String s) { name = s; }

      public String getType(){
            return "Unknown type";
      }

      public BecyDataType exception(String msg) {
            try{
                    throw new BecyException("[Becy Exception] NullPointer Exception with: "
+ msg);
            }catch(BecyException e){
                    System.out.println(e.getMessage());
                    System.exit(-1);
            }
            return null;
    }

    public BecyDataType exception(BecyDataType bdt, String msg){
                try{
                        if (bdt == null )
                                this.exception(msg);
                        throw new BecyException("[Becy Exception] Type Exception with: '" +
bdt.getType()
                                                                                + "'    operator: '" +
msg + "'");
                }catch (BecyException e){
                        System.out.println(e.getMessage());
                        System.exit(-1);
                }
                return null;
    }

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

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

    public String getName() {
            return name;
    }

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

    public void print() {
                System.out.println("Undefined Variable" );
    }

    public String toString () {
          this.exception("toString" );
          return null;
    }

    public BecyDataType assign(BecyDataType bdt) {
          return exception(bdt, "assign" );
    }

    public BecyDataType uminus(){
                return exception("uminus");
```

```
}

public BecyDataType add( BecyDataType bdt ){
            return exception(bdt, "add");
}

public BecyDataType sub( BecyDataType bdt ) {
       return exception( bdt, "sub" );
}

  public BecyDataType mul( BecyDataType bdt ) {
       return exception( bdt, "mul" );
}

public BecyDataType div( BecyDataType bdt ) {
       return exception( bdt, "div" );
}

public BecyDataType mod( BecyDataType bdt ) {
       return exception( bdt, "mod" );
}

public BecyDataType plus (BecyDataType bdt){
       return exception( bdt, "plus" );
}

public BecyDataType minus (BecyDataType bdt){
       return exception( bdt, "minus" );
}

public BecyDataType times (BecyDataType bdt){
       return exception( bdt, "times" );
  }

public BecyDataType fract (BecyDataType bdt){
       return exception( bdt, "fract" );
  }

public BecyDataType modulus (BecyDataType bdt){
       return exception( bdt, "modulus" );
}

public BecyDataType gt( BecyDataType bdt ) {
       return exception( bdt, "gt" );
}

public BecyDataType ge( BecyDataType bdt ) {
       return exception( bdt, "ge" );
}

public BecyDataType lt( BecyDataType bdt ) {
```

```
            return exception( bdt, "lt" );
      }

      public BecyDataType le( BecyDataType bdt ) {
            return exception( bdt, "le" );
      }

      public BecyDataType eq( BecyDataType bdt ) {
            return exception( bdt, "eq" );
      }

      public BecyDataType ne( BecyDataType bdt ) {
            return exception( bdt, "ne" );
      }

      public BecyDataType and( BecyDataType bdt ) {
            return exception( bdt, "and" );
      }

      public BecyDataType or( BecyDataType bdt ) {
            return exception( bdt, "or" );
      }

      public BecyDataType not() {
                  return exception( "not" );
      }
}
```

BecyException.java

```
public class BecyException extends Exception{
            public BecyException (String s){
                        super(s);
            }
}
```

BecyFunction.java

```
import antlr.collections.AST;

/*
      The function data type (including internal functions)
*/

class BecyFunction extends BecyDataType {
       // we need a reference to the AST for the function entry
       BecyDataType[] params;
       AST body;                        // body = null means an internal function.
       BecySymbolTable pst;      // the symbol table of static parent
       int id;                          // for internal functions only
```

```
        /**
         * Constructor of this class
         * Creates a new instance of this class
         * @param name is the name of this function
         * @param params is the parameter list of the function.
         * @param body is the function body from the AST
         * @param pst is the pointer to symbol table which contains this object.
         */
public BecyFunction( String name, BecyDataType[] params,
                              AST body, BecySymbolTable pst) {
         super( name );
         this.params = params;
         this.body = body;
         this.pst = pst;
}

        /**
         * Constructor of this class
         * Creates a new instance of this class
         * @param name is the name of this function
         * @param id is specific id for function.
         */
public BecyFunction( String name, int id )
        {
         super( name );
         this.params = null;
         this.id = id;
         pst = null;
         body = null;
}

        /**
         * returns the string which specifies its type
         * @return return an string representing the type
         */
public String typename()
        {
         return "function";
}

        /**
         * returns the clone of this class with parameter list
         */
public BecyDataType copy()
        {
         return new BecyFunction( name, params, body, pst );
}

        public void print()
        {
	}

		/**
		  * returns the parameter list of this function
		  * @return return an parameter list
		  */
	public BecyDataType[] getParams()
		{
		 return params;
	}

		/**
		  * returns the symbol table for this function
		  * @return return the symbol table
		  */
	public BecySymbolTable getParentSymbolTable()
		{
		 return pst;
	}

		/**
		  * returns the function body which is in AST
		  * @return return an function body
		  */
	public AST getBody()
		{
		 return body;
	}
}
```

BecyImpFun.java

```
import java.util.*;

/**
  * This class manipulates table input
  * and calculate some math operations with this information.
  * @version 1.2
  */

public class BecyImpFun {

	BecyTable bt;
	private Vector funList;
	private int numFun;

	public BecyImpFun(BecyTable bt) {
		this.bt = bt;
		funList = new Vector();
		// Initialize list of implemented functions;
		// Categorize by function parameters and return types
		// Requires ArrayList a and bool isVert; Returns BecyNumber i
		funList.add("sum");
		funList.add("average");
		funList.add("sd");
		funList.add("median");
		funList.add("var");
		funList.add("max");
		funList.add("min");
		// Requires ArrayList a and BecyNumber i; Returns BecyNumber x
		funList.add("round");
		funList.add("sqrt");
		funList.add("ceil");
		funList.add("floor");
		funList.add("abs");
		funList.add("power");
	}

	public boolean isImp(String name) {
		if (funList.contains(name))
		{
			return true;
		}
		else
		{
			return false;
		}
	}

	public BecyDataType callFun(BecyTable object, String name, BecyDataType[] args)
throws Exception {
		bt = object;
		ArrayList l;
		int i = 0;
		int j = 0;
		boolean isVert = false;

		numFun = funList.indexOf(name);

		if (numFun < 7)

		// Functions requiring ArrayList a and bool isVert;
		{
			i = (int)(((BecyNumber)args[0]).getNumber());
			isVert = ((BecyBool)args[1]).getBoolean();
			// Identify whether to retrieve row or column
			if (isVert)
				try {
					l = bt.getColumn(i);
```

```
                        } catch (Exception e) {
                                throw new Exception("error in bt getting column on
function "+name+"; Error: "+e);
                        }
                else
                        try {
                                l = bt.getRow(i);
                        } catch (Exception e) {
                                throw new Exception("error in bt getting row on function
"+name+"; Error: "+e);
                        }
        } else {
        // Functions requiring ArrayList a and BecyNumber i

                i = (int)(((BecyNumber)args[0]).getNumber());
                try {
                        l = bt.getRow(i);
                } catch (Exception e) {
                        throw new Exception("error in bt getting row on function "+name+";
Error: "+e);
                }
                j = (int)((BecyNumber)args[1]).getNumber();
        }

        // Call appropriate function
        switch (numFun+1) {
          case 1 :      return new BecyNumber(this.sum(l));
          case 2 :      return new BecyNumber(this.average(l));
          case 3 :      return new BecyNumber(this.sd(l));
          case 4 :      return new BecyNumber(this.median(l));
          case 5 :      return new BecyNumber(this.var(l));
          case 6 :      return new BecyNumber(this.max(l));
          case 7 :      return new BecyNumber(this.min(l));
          case 8 :      return new BecyNumber(this.round(l, j));
          case 9 :      return new BecyNumber(this.sqrt(l, j));
          case 10 :     return new BecyNumber(this.ceil(l, j));
          case 11 :     return new BecyNumber(this.floor(l, j));
          case 12 :     return new BecyNumber(this.abs(l, j));
          case 13:
                        double k = ((BecyNumber)args[2]).getNumber();
                        return new BecyNumber(this.power(l, j, k));
           default: throw new Exception("implicit called function does not exist: " + name);
        }
    }

        /**
          * Get the sum of the values of an ArrayList
          * @param list ArrayList list to be sumed
          * @return double result of the sum
          */
```

```
	public double sum(ArrayList list) {
		double sum=0;
		for(int i=0;i<list.size();i++){
			sum += Double.parseDouble((String)list.get(i));
		}
		return sum;
	}

	/**
	 * Get the sum of the square of each entry value of an ArrayList
	 * @param list ArrayList list to be sumed
	 * @return double result of the sum
	 */

	public double sum2(ArrayList list) {
		double sum2=0;
		for(int i=0;i<list.size();i++){
			double sum = Double.parseDouble((String)list.get(i));
			sum2 += (sum* sum);
		}
		return sum2;
	}


	/**
	 * Get the average of the values of an ArrayList
	 * @param list ArrayList list to be calculated
	 * @return double average number from the list
	 */
	public double average(ArrayList list) {
		return (sum(list) / (new Double(list.size())).doubleValue());
	}

	/**
	 * Get the standard deviation of the values of an ArrayList
	 * @param list ArrayList list to be calculated
	 * @return double standard deviation number from the list
	 */
	public double sd(ArrayList list) {
		double avg = this.average(list);
		return (Math.sqrt((this.sum2(list) / (new Double(list.size())).doubleValue()) - (avg *
avg)));
	}

	/**
	 * Get the median number of an ArrayList
	 * @param list ArrayList list to be calculated
	 * @return double median number from the list
	 */
	public double median(ArrayList list){
		list = this.sortNumbers(list);
```

```
        double median = 0;
        int index = (int) list.size()/2;
        if (list.size() % 2 != 0){
                median = Double.parseDouble((String)list.get(index));
        }
        else {
                median = (Double.parseDouble((String)list.get(index-1)) +
                                Double.parseDouble((String)list.get(index)))/2;
        }
        return median;
}

        /**
          * Get the variance number of an ArrayList
          * @param list ArrayList list to be calculated
          * @return double variance number from the list
          */
public double var(ArrayList list)
        {
        double stddev = this.sd(list);
        return (stddev * stddev);
}

        /**
          * Get the maximum number of an ArrayList
          * @param list ArrayList list to be calculated
          * @return double maximum number from the list
          */
public double max(ArrayList list){
        list = this.sortNumbers(list);
        int index = list.size() - 1;
        return Double.parseDouble((String)list.get(index));
}

        /**
          * Get the minimum number of an ArrayList
          * @param list ArrayList list to be calculated
          * @return double minimum number from the list
          */
public double min(ArrayList list){
        list = this.sortNumbers(list);
        return Double.parseDouble((String)list.get(0));
}

        /**
          * Get the rounded number of the specific number in an ArrayList
          * @param list ArrayList list to be calculated
          * @param index index of the ArrayList list
          * @return double rounded number from the list
          */
        public double round(ArrayList list, int index)
```

```
	{
	double number = Double.parseDouble((String)list.get(index));
	return Math.round(number);
}

	/**
	  * Get the square number of the specific number in an ArrayList
	  * @param list ArrayList list to be calculated
	  * @param index index of the ArrayList list
	  * @return double square number from the list
	  */
public double sqrt(ArrayList list, int index)
	{
	double number = Double.parseDouble((String)list.get(index));
	return Math.sqrt(number);
}

	/**
	  * Get the rounded-up number of the specific number in an ArrayList
	  * @param list ArrayList list to be calculated
	  * @param index index of the ArrayList list
	  * @return double rounded-up number from the list
	  */
	public double ceil(ArrayList list, int index)
	{
	double number = Double.parseDouble((String)list.get(index));
	return Math.ceil(number);
}

	/**
	  * Get the rounded-down number of the specific number in an ArrayList
	  * @param list ArrayList list to be calculated
	  * @param index index of the ArrayList list
	  * @return double rounded-down number from the list
	  */
public double floor(ArrayList list, int index)
	{
	double number = Double.parseDouble((String)list.get(index));
	return Math.floor(number);
}

	/**
	  * Get the absolute number of the specific number in an ArrayList
	  * @param list ArrayList list to be calculated
	  * @param index index of the ArrayList list
	  * @return double absolute number from the list
	  */
public double abs(ArrayList list, int index)
	{
	double number = Double.parseDouble((String)list.get(index));
	return Math.abs(number);
```

```
    }

        /**
         * Get the powered-number of the specific number in an ArrayList
         * @param list ArrayList list to be calculated
         * @param index index of the ArrayList list
         * @param power number to be powered
         * @return double powered-number from the list
         */
    public double power(ArrayList list, int index, double power)
        {
        double number = Double.parseDouble((String)list.get(index));
        return Math.pow(number, power);
    }

        /**
         * Get the sorted number list of an ArrayList
         * @param list ArrayList list to be sorted
         * @return ArrayList sorted Array list
         */
    public ArrayList sortNumbers(ArrayList list)
        {
        int a,b;
        int sortTheNumbers = list.size() - 1;
        double temp;

        for (a = 0; a < sortTheNumbers; ++a) {
                for (b = 0; b < sortTheNumbers; ++b){
                        double left = Double.parseDouble((String)list.get(b));
                        double right = Double.parseDouble((String)list.get(b+1));
                        if(left > right){
                                temp = left;
                                left = right;
                                right = temp;
                        }
                                list.set(b, (new Double(left)).toString());
                                list.set(b+1, new Double(right).toString());

                }
        }
        return list;
    }
}
```

BecyInterpreter.java

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

/*
    Interpreter routines that are called directly from the tree walker.
*/

class BecyInterpreter {
        BecySymbolTable symt;
        BecyImpFun impFun;

        final static int fc_none = 0;
        final static int fc_return = 1;

        private int control = fc_none;
        private String label;

        private Random random = new Random();

        public BecyInterpreter() {
                symt = new BecySymbolTable();
        }

        // Converts a Vector of BecyDataType's to an array of them
        public BecyDataType[] convertExprList( Vector v ) {
                /* Note: expr list can be empty */
                BecyDataType[] x = new BecyDataType[v.size()];
                for ( int i=0; i<x.length; i++ )
                        x[i] = (BecyDataType) v.elementAt( i );
                return x;
        }

        // Converts a Vector of strings to an array of them
        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 static BecyDataType getNumber( String s ) {
                return new BecyNumber( Integer.parseInt( s ) );
        }

        // Retrieves a BecyDataType or returns a new BecyDataType with corresponding name
        public BecyDataType getObject( String s ) {
                BecyDataType x = symt.getValue(s);
                if ( x.getName() == "UNDEF" )
                        return new BecyDataType( s );
                return x;
```

```
	}

	public BecyDataType rvalue( BecyDataType a ) {
		return a;
	}

	public BecyDataType assign( BecyDataType a, BecyDataType b ) {
		if ( null != a.name )
		{
			BecyDataType x = rvalue( b );
			x.setName( a.name );
			symt.putValue(x.name, x);
			return x;
		}

		return a.exception( b, "=" );
	}

	public BecyDataType funcInvoke(
		BecyAntlrWalker walker,    BecyTable object,
BecyDataType name,

		BecyDataType[] args ) throws Exception {

	    impFun = new BecyImpFun(object);

	    // check if function is language implemented
	    if ( impFun.isImp(name.getName()))
	    {
	          return impFun.callFun(object, name.getName(), args);
	    }

		// function must exist
		if ( !( name instanceof BecyFunction ) )
			return name.exception( "not a function" );

		BecyDataType[] params = ((BecyFunction)name).getParams();
		if ( args.length != params.length )
			return name.exception( "unmatched length of parameters" );

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

		// call the function body
		BecyDataType r = walker.expr( ((BecyFunction)name).getBody() );
```

```
        return r;
}

public void funcRegister( String name, BecyDataType[] params, AST body ) {
        symt.putValue( name, new BecyFunction( name, params, body, symt ) );
}

public void setReturn( String label ) {
        this.label = label;
        control = fc_return;
}

public void tryResetFlowControl( String loop_label ) {
        if ( null == label || label.equals( loop_label ) )
                control = fc_none;
}

public boolean canProceed() {
        return control == fc_none;
}

/**
  * Opens a source textfile and initiates a new BecyTable
  * @param name BecyString name of source text file
  * @throws BecyException if any exception happen
  * @return BecyTable the newly initiated BecyTable
  */
public BecyTable include( BecyString file ) {
        if ( file instanceof BecyString )
        {
                try
                {
                        return new BecyTable(file.getString());
                }
                catch ( Exception e )
                {
                     System.out.println("include failed; Error: "+e.getMessage());
                     System.exit(-1);
                }
        }
        return new BecyTable();
}

/**
  * Enters a new BecyTable object in the symbol table
  * @param name BecyString name of new data object
  * @param table BecyTable actual BecyTable to be included in symbol table
  * @throws BecyException if any exception happen
  * @return BecyTable the new BecyTable loaded into the symbol table
  */
public BecyTable newDataObject( BecyString name, BecyTable object ) {
        try
        {
            symt.putValue(name.getString(), object);
            return object;
        }
        catch ( Exception e )
        {
                System.out.println("object store in symbol table failed; Error:
"+e.getMessage());
                System.exit(-1);
        }
        return new BecyTable();
    }
}
```

BecyMain.java

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

/*
   The main class
*/

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

            BecyAntlrLexer lexer = new BecyAntlrLexer( input );

            BecyAntlrParser parser = new BecyAntlrParser( lexer );

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

            // Lexer & Parser error check
            if ( lexer.nr_error > 0 || parser.nr_error > 0 )
            {
                System.err.println( "Parsing errors found. Stop." );
                return;
            }
            CommonAST tree = (CommonAST)parser.getAST();

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

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

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

            BecyDataType r = walker.expr( tree );

           HTMLOutput h = new HTMLOutput();
           h.createHTMLFromPool(HTMLPool.getArray());

        } 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 );
        }
    }

    public static void main( String[] args ) {

        if ( args.length == 1 )
            execFile( args[args.length-1] );
       else
           System.out.println("Please add your Becy program to run.");

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

BecyNumber.java

```
public class BecyNumber extends BecyDataType{

        private double number;

        public BecyNumber(double number) {
                this.number = number;
        }
```

```
	public String getType(){
		return "BecyNumber";
	}

	public String toString(){
		return new Double(number).toString();
	}

	public void print() {
		System.out.println(Double.toString(number));
		HTMLPool.add(Double.toString(number));
	}

	public double numValue (BecyDataType bdt){
		if(bdt instanceof BecyNumber)
			return ((BecyNumber) bdt).getNumber();

		exception("Number Exception");
		return 0;
	}

	public double getNumber(){
		return number;
	}

	public void setNumber(double number){
		this.number = number;
	}

	public BecyNumber assign(BecyNumber bdt) {
		setNumber(bdt.getNumber());
		return this;
	}

public BecyDataType uminus(){
	 return new BecyNumber(-number);
}

public BecyDataType add (BecyDataType num){
	number = number + numValue(num);
	return this;
}

public BecyDataType sub (BecyDataType num){
	number = number - numValue(num);
	return this;
}

public BecyDataType mul (BecyDataType num){
	number = number * numValue(num);
	return this;
```

```
    }

  public BecyDataType div (BecyDataType num){
        number = number / numValue(num);
        return this;
    }

  public BecyDataType mod (BecyDataType num){
        number = number % numValue(num);
        return this;
  }

  public BecyDataType plus (BecyDataType num){
        return new BecyNumber(number + numValue(num));
  }

  public BecyDataType minus (BecyDataType num){
        return new BecyNumber(number - numValue(num));
  }

  public BecyDataType times (BecyDataType num){
        return new BecyNumber(number * numValue(num));
    }

  public BecyDataType fract (BecyDataType num){
        return new BecyNumber(number / numValue(num));
    }

  public BecyDataType modulus (BecyDataType num){
        return new BecyNumber(number % numValue(num));
  }

  public BecyDataType gt (BecyDataType num){
        return new BecyBool (number > numValue(num));
}

  public BecyDataType ge (BecyDataType num){
        return new BecyBool (number >= numValue(num));
  }

  public BecyDataType lt (BecyDataType num){
        return new BecyBool (number < numValue(num));
  }

  public BecyDataType le (BecyDataType num){
        return new BecyBool (number <= numValue(num));
  }

  public BecyDataType eq (BecyDataType num){
        return new BecyBool (number == numValue(num));
  }
```

```
	public BecyDataType ne (BecyDataType num){
		return new BecyBool (number != numValue(num));
	}
}
```

BecyString.java

```
public class BecyString extends BecyDataType{

		private String str;

		public BecyString() {
			this.str = null;
		}

		public BecyString( String str ) {
			this.str = str;
		}

		public String getType(){
			return "BecyString";
		}

		public void print() {
			System.out.println( str );
			HTMLPool.add(str);
		}

		public String toString(){
			return str;
		}

		public BecyString assign(BecyString b) {
			setString(b.getString());
			return this;
		}

		public BecyString append (BecyString b) {
			setString(str + b.getString());
			return this;
		}

		public BecyDataType eq (BecyDataType string){
			if (string instanceof BecyString)
				return new BecyBool (str == ((BecyString) string).getString());
			return exception("String eq test Exception");
		}
		public BecyDataType ne (BecyDataType string){
			if (string instanceof BecyString)
				return new BecyBool (str != ((BecyString) string).getString());
			return exception("String ne test Exception");
		}

		public String getString(){
			return str;
		}

		public void setString(String str){
			this.str = str;
		}
}
```

BecySymbolTable.java

```
import java.util.*;

/*
 *	All variables have to be global.
 *	Supports only one scope.
 *
 */

public class BecySymbolTable{

    public Hashtable table;

    public BecySymbolTable() {
        table = new Hashtable();
    }

    public BecyDataType getValue(String keyVar) {
        if (table.containsKey(keyVar))
			return (BecyDataType) table.get(keyVar);
        else
			return new BecyDataType("UNDEF");
    }

    public void putValue(String keyVar, BecyDataType value) {
         table.put(keyVar, value);
    }

    public boolean containsKey(String keyVar) {
        return table.containsKey(keyVar);
    }
    public boolean containsValue(BecyDataType bdt) {
        return table.containsValue(bdt);
    }

    public String typeOf(String keyVar) {
        BecyDataType bdt = (BecyDataType) table.get(keyVar);
        return bdt.getType();
    }

    //Returns a string representation of the Hashtable in the form of set.
    public String toString() {
         Enumeration e = table.keys();

         String str = "{ ";
         while (e.hasMoreElements()){
                 String key = (String) e.nextElement();
                 BecyDataType bdt = (BecyDataType) table.get(key);
                 String value = null;

                 if (bdt instanceof BecyString)
                         value = ((BecyString) bdt).toString();
                 else if(bdt instanceof BecyNumber)
                         value = ((BecyNumber) bdt).toString();
                 else if(bdt instanceof BecyBool)
                         value = ((BecyBool) bdt).toString();

                 str += key + "=" + value + " ";
         }
        str += "}";
        return str;
    }
}
```

BecyTable.java

```
import javax.swing.table.TableModel;
import javax.swing.JTable;
import java.util.*;
import java.io.*;
import java.lang.reflect.Array;

/**
  * This class manipulates the input information and calculate some math operations
  * whit this info.
  * @version 1.2
  */
public class BecyTable extends BecyDataType {

    int cols = 0;
    int rows = 0;
```

```
// Create with initial data
Object[][] cellData;
ArrayList data= new ArrayList();

Object[] columnNames = {"col1", "col2"};
//JTable table =new JTable();
static final int LIMIT=10000;


/** Creates a new instance of BecyTable */
public BecyTable() {
        //table = new JTable();
}

/**
  * Creates a new instance of this class setting the input from a file
  * @param data is the name of the file with the input information
  */
public BecyTable(String data) {
        assert (data!=null);
        try{
                System.out.println("getting data...");
                this.getDataFromFile(data);
        }catch(Exception e){
                System.out.println("Error getting data "+e);
        }
        //table = new JTable(cellData, columnNames);
}

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

  * return the number of the columns of the main table
  * @return return an integer representing the number of columns
  */
public int getCols(){
        return cols;
}

/**
  * Set a number of columns for the main table
  * @param cols integer number representing columns int the main table
  */
public void setCols(int cols){
        assert cols<LIMIT;
        this.cols=cols;
}
/**
  * return the number of the rows of the main table
  * @return return an integer representing the number of rows
  */
public int getRows(){
        return rows;
}

/**
  * Set a number of rows for the main table
  * @param rows integer number representing rows int the main table
  */
public void setRows(int rows){
        assert cols<LIMIT;
        this.rows=rows;
}

/**
  * get the structure with the data of the table
  * @throws java.lang.Exception If any exception happen
  * @return Object array[][] array of 2D with the rows and columns of the data
  */
public ArrayList getData() throws Exception{
        return this.data;
}

/**
  * returns a column of the table
  * @param index int number of the column needed
  * @throws java.lang.Exception if any exception happen
  * @return ArrayList with the info of the column selected
  */
public ArrayList getColumn(int index) throws Exception{
        assert (index>-1)&&(index<this.getCols());
        return this.getColumn(index,1,this.getRows());
}

/**
  * Returns a range of one column of the table
  * @param index int number of the column needed
  * @param begin int begin of the range whitin the column
  * @param end end of the range whitin the column
  * @throws java.lang.Exception if any exception happen
  * @return ArraList with the info of the range whitin the column selected
  */
public ArrayList getColumn(int index, int begin, int end) throws Exception{
        assert (begin>-1)&&(begin<=end);
        assert (end<this.getCols());
        index--;
        begin--;
        end--;
        //System.out.println("getting column from "+begin+" to "+end);
        if (index>this.getCols()-1)
```

```
            throw new Exception("Not valid column index "+index);
        ArrayList col = new ArrayList();
        for (int i=begin; i<end;i++){
            //System.out.println("getting value ("+i+","+index+") = "+table.getValueAt(i,
index));
            //col.add(table.getValueAt(i, index));
            col.add(cellData[i][index]);
        }
        return col;
    }

    /**
     * Returns a range of 1 row of the table
     * @param index int number of the row needed
     * @throws java.lang.Exception if any exception happen
     * @return ArrayList with the info of the range within the row selected
     */
    public ArrayList getRow(int index) throws Exception{
        assert (index>-1) && (index<this.getRows());
        return this.getRow(index,1,this.getCols());
    }

    /**
     * Returns a range of 1 row of the table
     * @param index int number of the row needed
     * @param begin int begin of the range
     * @param end int end of the range
     * @throws java.lang.Exception if any exception happen
     * @return ArrayList with the info of the range within the row selected
     */
    public ArrayList getRow(int index, int begin, int end) throws Exception{
        assert (begin>-1)&&(begin<=end);
        assert (end<this.getCols());
        index--;
        begin--;
        if (index>this.getRows()-2)
            throw new Exception("Not valid row index "+index);
        ArrayList row = new ArrayList();
        for (int i=begin; i<end;i++){
            //row.add(table.getValueAt(index, i));
            row.add(cellData[index][i]);
        }
        return row;
    }

    /**
     * Reads a file and parse it to fill the table attribute of the class
     * @param inFileName String name of the input file
     * @throws java.io.IOException If any exception happen during the I/O
     */
    public void getDataFromFile(String inFileName)throws IOException{
```

```
		assert inFileName!=null;
		assert !inFileName.equals("");
		try {
			File file = new File("");
			System.out.println(file.getAbsolutePath());
			BufferedReader in = new BufferedReader(new FileReader(inFileName));
			String str;
			while ((str = in.readLine()) != null) {
				if (!str.equals("")){
					System.out.println("reading line..."+str);
					process(str);
					this.setRows(this.getRows()+1);
				}
			}
			System.out.println("closing file...");
			in.close();
			System.out.println("Creating list data...");
			this.listToArray();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	/**
	 * Parses a line and fill a list with the info of the table
	 * @param str String line to be parsed
	 */
	public void process(String str){
		assert str!=null;
		ArrayList list = new ArrayList();
			StringTokenizer parser = new StringTokenizer(str);
			while (parser.hasMoreTokens()) {
				list.add(parser.nextToken());
			}
			if (list.size()>this.cols){
				this.setCols(list.size());
			}
			data.add(list);
	}

	/**
	 * Converts the ArrayList with the info of the table in a 2d array of Objects
	 */
	public void listToArray(){
		int extraSize = 101;
		cellData = (String[][])Array.newInstance(String.class, new int[]{this.getRows()-
1+extraSize,this.getCols()+extraSize});

		ArrayList row = (ArrayList)data.get(0);
		//System.out.println("row list = "+row.get(0));
		columnNames=(Object[])row.toArray();
```

```
        for(int i=1 ; i<data.size();i++){
                System.out.println("getting row");
                row = (ArrayList)data.get(i);

                for(int j=0;j<this.getCols();j++){
                        cellData[i-1][j]=row.get(j);
                }
        }
}

/**
  * Add a row to the data of the table
  * @param array Object array with the row is going to be added
  */
public void addRow(Object[] array){
        for(int j=0;j<this.getCols();j++){
                        cellData[this.getRows()-1][j]=array[j];
                }

        this.setRows(this.getRows()+1);
}

/**
  *
  * @param array
  */
public void addCol(Object[] array){

        for(int j=0;j<this.getRows()-1;j++){
                        cellData[j][this.getCols()]=array[j];
                }
        this.setCols(this.getCols()+1);
}

public void print(){
        printTable();
      HTMLOutput h = new HTMLOutput();
      h.setTable(this);
      HTMLPool.add(h.getHTML());
}

/**
  * Prints the content of the table
  */
public void printTable(){
        for (int i=0; i<this.getRows()-1; i++) {
                for(int j=0; j<this.getCols();j++){
                        System.out.println((String)cellData[i][j]);
                      //System.out.println((String)table.getValueAt(i, j));
                }
        }
    }

    /**
      * Prints the content of an ArrayList
      * @param list Arraylist list to be printed
      */
    public void printArray(ArrayList list){
        assert list!=null;
        for (int i=0; i<list.size(); i++) {
                System.out.println((String)list.get(i));
        }
    }

    /**
      * Get the sum of the values of an ArrayList (must be integers)
      * @param list ArrayList list to be sumed
      * @throws java.lang.Exception If any exception happen
      * @return int result of the sum
      */
    public float sum(ArrayList list) throws Exception{
        assert list!=null;
        float sum=0;
        for(int i=0;i<list.size();i++){
            try{
                sum+=Float.parseFloat((String)list.get(i));
            }catch(Exception e){
                throw new Exception("can't sum, invalid number "+list.get(i)+" "+e);
            }
        }
        return sum;
    }

    /**
      * This is a test of the class which select a column and prints into the prompt
      * @param args args from standar input
      * @throws java.lang.Exception if eny exception happen during the execution
      */
    public static void main(String args[]) throws Exception{
      BecyTable mt= new BecyTable(args[0]);
      //mt.printTable();
      mt.printTable();
      //System.out.println(mt.sum(mt.getColumn(1)));
     System.out.println("NEW TABLE");
     mt.addRow(mt.getRow(1).toArray());
     mt.addCol(mt.getColumn(1).toArray());
     mt.printTable();
    }

}
```

BecyVariable.java

```
/*
    The wrapper class for unintialized variables
*/

class BecyVariable extends BecyDataType {
        public BecyVariable( String name ) {
                super( name );
        }

        public String typename() {
                return "undefined-variable";
        }

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

        public void print() {
        }
}
```

HtmlOutput.java

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

public class HTMLOutput {


                String tableSource="dataIn2.txt";
                String htmlName = "out";
                BecyTable mt;
                String[] printPool;


                /**
                  * This method creates a new instance of HTMLOutput.
                  * @param table BecyTable table is going to be displayed as HTML output
                  * @param outputFileName String name of the HTML file of the output (without
".html").
                  * @param printPool String[] pool of Strings needed to print after the table. Used for
results of
                  * operations with the data that can't be shown whitin the BecyTable table print.
                  * Ex. {"Average = 6.4"," Lowest value = 3 ","etc..."}
                  *
                  */
```

```
	public HTMLOutput(BecyTable table, String outputFileName, String[] printPool){
		mt= table;
		this.htmlName = outputFileName;
		this.printPool=printPool;

	}


	public HTMLOutput(){

	}

	/**
	  * Set the name of the inputFileName
	  * @param filename String name of the TXT file that is going to be parsed to create a
new instance
	  * of a BecyTable
	  */
	public void setTableSource(String filename){
		this.tableSource = filename;
	}

	/**
	  * Set the BecyTable from which to generate HTML
	  * @param BecyTable name of the BecyTable
	  */
	public void setTable(BecyTable bt){
		this.mt = bt;
	}

	public String getHTML(){
		StringBuffer out = new StringBuffer("");
		out.append("<table border=1><tr><td>");

		ArrayList row=new ArrayList();
		ArrayList data= new ArrayList();
		try{
		  data=mt.getData();
		}catch(Exception e){
			System.out.println("Error HTMLOUTPUT generating table");
		}
		out.append("<tr bgcolor=#4F5A62>");
		row = (ArrayList)data.get(0);
		for(int j=0;j<mt.getCols();j++){
		  out.append("<td align=center><font
color=#FFFFFF><b>"+row.get(j)+"</b></font></td>");
		}
		out.append("</tr>");
		String color="";
		for(int i=1 ; i<data.size();i++){
				if((i%2)==1)
					color="bgcolor=#EFF2F2";
				else
					color="";
				out.append("<tr "+color+" >");
				row = (ArrayList)data.get(i);
				for(int j=0;j<mt.getCols();j++){
						if(((String)row.get(j)).endsWith(".jpg"))
							out.append("<td><img src="+row.get(j)+" width=180
height=180 /></td>");
						else
							out.append("<td>"+row.get(j)+"</td>");
				}
				out.append("</tr>");
			}

			out.append("</table>");
			return out.toString();
		}



		public void createHTMLFromPool(Object[] pool){
			try {
				BufferedWriter out = new BufferedWriter(new
FileWriter(htmlName+".html"));
				for(int i=0;i<pool.length;i++){
					out.write((String)pool[i]);
					out.write("");
				}
				out.close();
			}catch(Exception e){
				System.out.println(e);
			}
		}

		public void createHTML(){

			try {
			BufferedWriter out = new BufferedWriter(new FileWriter(htmlName+".html"));
			out.write("<html><body>");
			out.write("<b>Table</b><br>");
			out.write("<table border=1><tr><td>");

			ArrayList row=new ArrayList();
			ArrayList data=mt.getData();
				out.write("<tr bgcolor=#4F5A62>");
				row = (ArrayList)data.get(0);
				for(int j=0;j<mt.getCols();j++){
				 out.write("<td align=center><font
color=#FFFFFF><b>"+row.get(j)+"</b></font></td>");
```

```
                }
                out.write("</tr>");
            String color="";
            for(int i=1 ; i<data.size();i++){
                if((i%2)==1)
                    color="bgcolor=#EFF2F2";
                else
                    color="";
                out.write("<tr "+color+" >");
                row = (ArrayList)data.get(i);
                for(int j=0;j<mt.getCols();j++){
                        if(((String)row.get(j)).endsWith(".jpg"))
                            out.write("<td><img src="+row.get(j)+" width=180
height=180 /></td>");
                        else
                            out.write("<td>"+row.get(j)+"</td>");
                }
                out.write("</tr>");
            }

            out.write("</table>");
            out.write("<br><br>");

              out.write("<table border=1><tr><td>");


            color="";
            for(int i=0 ; i<printPool.length;i++){
                if((i%2)==1)
                    color="bgcolor=#EFF2F2";
                else
                    color="";
                out.write("<tr "+color+" ><td>");
                out.write(printPool[i]);
                out.write("</td></tr>");
            }

            out.write("</table>");



            out.write("</body></html>");
            out.close();

        }
        catch (Exception e) {

            e.printStackTrace();
        }

    }   // end doGet method
```

```
		/**
		  * Main method of the class to prove functionality
		  * @param args String Standar Input
		  */
		public static void main(String args[]){
		     BecyTable mt= new BecyTable("dataIn2.txt");
		     String[] pool={"line 1 \n still one","line 2", "line 3...","line number 4......."};
		       HTMLOutput hout = new HTMLOutput(mt,args[0],pool);
		       //hout.createHTML();
		       pool[0]=hout.getHTML();
		       hout.createHTMLFromPool(pool);
		       //System.out.println(hout.getHTML());
		}
}    // end table class
```

HtmlPool.java

```
import java.util.*;

public class HTMLPool {


	public static ArrayList pool = new ArrayList();
	/** Creates a new instance of NewClass */
	public HTMLPool() {
	}

	public static void add(String str){
		pool.add(str);
	}

	public static Object[] getArray(){
		return pool.toArray();
	}

	public static void print(){
		for(int i=0;i<pool.size();i++){
			System.out.println(pool.get(i));
		}
	}
}
```

MakeFile.java

```
JCC = javac

JFLAGS = -source 1.4 -classpath .
```

```
# typing 'make' will invoke the first target entry in the makefile
# (the default one in this case)
#

all: BecyBool.class BecyDataType.class BecyException.class \
        BecyInterpreter.class BecyMain.class BecyNumber.class \
        BecyString.class BecySymbolTable.class BecyTable.class \
        HTMLOutput.class HTMLPool.class BecyImpFun.class \
        TestDriver_Parser.class TestDriver_Table.class


BecyBool.class: BecyBool.java
        $(JCC) $(JFLAGS) BecyBool.java

BecyDataType.class: BecyDataType.java
        $(JCC) $(JFLAGS) BecyDataType.java

BecyException.class: BecyException.java
        $(JCC) $(JFLAGS) BecyException.java

BecyInterpreter.class: BecyInterpreter.java
        $(JCC) $(JFLAGS) BecyInterpreter.java

BecyImpFun.class: BecyImpFun.java
        $(JCC) $(JFLAGS) BecyImpFun.java

BecyMain.class: BecyMain.java
        $(JCC) $(JFLAGS) BecyMain.java

BecyNumber.class: BecyNumber.java
        $(JCC) $(JFLAGS) BecyNumber.java

BecyString.class: BecyString.java
        $(JCC) $(JFLAGS) BecyString.java

BecySymbolTable.class: BecySymbolTable.java
        $(JCC) $(JFLAGS) BecySymbolTable.java

BecyTable.class: BecyTable.java
        $(JCC) $(JFLAGS) BecyTable.java

HTMLOutput.class: HTMLOutput.java
        $(JCC) $(JFLAGS) HTMLOutput.java

HTMLPool.class: HTMLPool.java
        $(JCC) $(JFLAGS) HTMLPool.java

TestDriver_Parser.class: TestDriver_Parser.java
        $(JCC) $(JFLAGS) TestDriver_Parser.java

TestDriver_Table.class: TestDriver_Table.java
```

```
	$(JCC) $(JFLAGS) TestDriver_Table.java



clean:
	$(RM) *.class *.out
```

TestDriver_Parser.java

```
/*
 * Simple front-end for an ANTLR lexer/parser that dumps the AST
 * textually and displays it graphically.   Good for debugging AST
 * construction rules.
 *
 */

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class TestDriver_Parser {
   public static void main(String args[]) {
      try {
         DataInputStream input = new DataInputStream(System.in);

         // Create the lexer and parser and feed them the input
         BecyAntlrLexer lexer = new BecyAntlrLexer(input);
         BecyAntlrParser parser = new BecyAntlrParser(lexer);
         parser.program();

         // Get the AST from the parser
         CommonAST parseTree = (CommonAST)parser.getAST();

         // Print the AST in a human-readable format
         System.out.println(parseTree.toStringList());

      } catch(Exception e) { System.err.println("Exception: "+e); }
   }
}
```

TestDriver_Table.java

```
public class TestDriver_Table {

	/**Test Program for variable in becy
	 * Variable, String, Number, Boolean and Symboltable
	 * @param args
	 */
	public static void main(String[] args) throws Exception{
```

```
//Basic Calculation with BecyNumbers.
//Number basically represents real numbers.
BecyNumber a;
BecyNumber b = new BecyNumber(10);
BecyNumber c = new BecyNumber(3);

a = (BecyNumber) b.plus(c);      //'plus' uses 'call by value'
a.print();                        //a=13.0
b.print();                        //b=10.0

//Now a=13.0, b=10.0, c=3.0
a = (BecyNumber) b.add(c);       //'add' uses 'call by reference'
a.print();                        //a=13.0
b.print();                        //b=13.0

b.assign(c);          //Assign c into b in BecyNumber
b.print();            //b=3.0

//Get the value of b variable
System.out.println("BecyNumber b: " + b.getNumber()); //BecyNumber b:
3.0

//BecyDataType d = new BecyString("test");
//b.assign(d);      //Will give Exception, Can't assign BecyString to
BecyNumber


//Basic operation with String.
BecyString str1 = new BecyString("becy");
str1.print();               //becy

BecyString str2 = new BecyString();
str2.assign(str1);
str2.print();               //becy

BecyString bstr = str2;
bstr.print();               //becy

str2.append(str1);
str2.print();               //becybecy

System.out.println("BecyString str2: " + str2.getString()); //BecyString str2:
becybecy


//Basic operation with Boolean
BecyBool bool1 = (BecyBool) b.eq(c);       //b=3.0, c=3.0
bool1.print();             //true

BecyBool bool2 = (BecyBool) b.lt(c);       //b=3.0, c=3.0
```

```
			bool2.print();			//false

			bool1 = new BecyBool(true);
			bool2.assign(bool1);
			bool2.print();			//true

			bool1.and(bool2);
			bool1.print();			//true

			System.out.println("BecyBool bool1: " + bool1.getBoolean()); //BecyBool
bool1: true


			//Test Symbol table
			BecySymbolTable bst = new BecySymbolTable();
			bst.putValue("a", new BecyString("yongju"));
			bst.putValue("b", new BecyNumber(1.43));


			System.out.print("In symbol table: a = ");
			((BecyString) bst.getValue("a")).print();		//In symbol table: a =
yongju
			System.out.print("In symbol table: b = ");
			((BecyNumber) bst.getValue("b")).print();		//In symbol table: b =
1.43

			//Variable 'a' is defined, but 'c' is not defined.
			System.out.println("variable 'a' is in symbol table: " + bst.containsKey("a"));
//true
			System.out.println("variable 'c' is in symbol table: " + bst.containsKey("c"));
		//false

			System.out.println("Symbol table: " + bst.toString());	//string
representation

			bst.putValue("b", new BecyBool(true));		//In symbol table: b = true


			//Print the hash table with key and value pair
			System.out.println("Symbol table: " + bst.toString());	//string
representation


			//Test Table class
		 BecyTable bt= new BecyTable("dataIn.txt");
		 BecyImpFun t = new BecyImpFun(bt);
		 bt.printArray(bt.getRow(1));

		 //sorted list: [10.0, 10.0, 15.0, 20.0, 30.0, 30.0, 100.0]
		System.out.println("sorted list: " + t.sortNumbers(bt.getColumn(1)));
```

```
            System.out.println("sum: " + t.sum(bt.getColumn(1)));                          //sum:
215.0
            System.out.println("average: " + t.average(bt.getColumn(1)));                  //average:
30.714285714285715
            System.out.println("standard deviation: "+t.sd(bt.getColumn(1)));             //stddev:
29.329235953782586
            System.out.println("median: " + t.median(bt.getColumn(1)));
//median: 20.0
           System.out.println("variation: " + t.var(bt.getColumn(1)));                    //variation:
860.2040816326531
           System.out.println("max: " + t.max(bt.getColumn(1)));                           //max:
100.0
           System.out.println("min: " + t.min(bt.getColumn(1)));                           //min:
10.0

           System.out.println("round: " + t.round(bt.getColumn(1), 0));                    //round:
10.0
           System.out.println("sqrt: " + t.sqrt(bt.getColumn(1), 0));                      //sqrt:
3.1622776601683795
           System.out.println("ceil: " + t.ceil(bt.getColumn(1), 0));                      //ceil: 10.0
           System.out.println("floor: " + t.floor(bt.getColumn(1), 0));                    //floor: 10.0
           System.out.println("abs: " + t.abs(bt.getColumn(1), 0));                          //abs: 10.0
           System.out.println("power: " + t.power(bt.getColumn(1), 0, 2.2));             //power:
158.48931924611142
           }
}
```

TestInput_datasource.txt

```
Name HW1 HW2 HW3 Midterm Final Pic
Napoleon 88 67 43 57 89
http://is2.okcupid.com/users/104/656/10465692962375378952/mt1124409989.jpg
Ernie 100 100 100 98 99 http://www.fantasyworldcostumes.com/images/Create-a-Pak,Nerd-
761.jpg
Jessica 90 91 92 88 92 http://newsfromrussia.com/images/newsline/00-96-nerd.jpg
Cheryl 100 100 100 100 100 http://www.clam.at/uploads/pics/nena200.jpg
```

TestInput_parser.txt

```
$a = $a;
$a = 1;
$a = ( 1 + 2 );

( $a > 1 ) ? $b = 1 : $c = 2;
( $a == 1 ) ? $b = 1 : $c = 2;
( true || false ) ? $b = 1 : $c = 2;
( ($a < 1) && true ) ? $b = 1 : $c = 2;
( ($a < 1) && ($b >= 1) ) ? $c = 1 : $d = 2;
( $a == 1 ) ? ( $b == 1 ) ? $d = 1 : $e = 2 : ( $c > 1 ) ? $f = 1 : $g = 2 ;
( ($b >= 1) == ( ($c+1) > ($d*10) ) ) ? $a = 1 : $e = 2 ;
( ($a < 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? ( $b == 1 ) ? $d = 1 : $e = 2 : ( $c >
1 ) ? $f = 1 : $g = 2 ;

for $a TO $b {
   $a = $a;
   $a = 1;
   object1.foo ($a TO $b);
};


function foo1 ($a, $b, 3)
{
   $a = 1;
   $b = $a;
};

function foo2 ($a, $b, 3)
{
   $a = 1;
   $b = $a;
( $a == 1 ) ? ( $a == 1 ) ? $a = 1 : $a = 2 : ( $a > 1 ) ? $a = 1 : $a = 2 ;
};

function foo3 ($a, $b, 3)
{
   $a = 1;
   $b = $a;
   for $a TO $b {
      $a = $a;
      $a = 1;
      object1.foo ($a TO $b);
   };
};

object1 { include "somefile.txt" };
object2 { 1 2 3 "bob", 4 5 6 "jim", 7 8 9 "mary"};
object3 { 3 "bob", 6 "jim", 9 "mary"};

object1.foo (1,2,3);
object1.foo ($a TO $b);

$a = object1.sum ($a TO $b);

print object1;
print $a;

( 1 == 2 ) ? object1.copy(object2) : print object1;
```

****
BAD CASES (Cases that should fail):
****

```
bad_object { $a $b };
( ($a + 1) && true ) ? $a = 1 : $a = 2;
```

TestInput_v1.txt

```
$a = 1;
$b = ( 1 + 2 );

print "A: ";
print $a;
print "B: ";
print $b;
```

TestInput_v2.txt

```
$a = 1;
$b = 5;
$c = 0;

for $a TO $b {
   $c = $c + 1;
   print "Iteration:";
   print $c;
};

print $a;
print $b;
```

TestInput_v3.txt

```
$a = 0;
$b = 2;
$c = 10;
$d = 0;

( ($a > 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? $d = 1 : $d = 2 ;

print "$d should be 2.";
print "$d:";
print $d;
```

TestInput_v4.txt

```
$a = 0;
$b = 2;
$c = 0;
$d = 1;

( ($a < 1) && ( ($b >= 1) == ( ( $c + 1 ) > ($d * 10) ) ) ) ? ( $b == 1 ) ? $d = 1 : $e = 2 : ( $c >
1 ) ? $f = 1 : $g = 2 ;

print "$g should be 2.";
print "$g:";
print $g;
```

TestInput_v5.txt

```
object1 { include "TestInput_datasource.txt" };

print "Data object:";
print object1;
```

TestInput_v6.txt

```
object1 { include "TestInput_datasource.txt" };

function foo1 ($a, $b, $c)
{
   $d = 1;
   $e = $a;
   return $e;
};

$f = 3;
$g = 4;

$h = object1.foo1 ($f, $g, 10);

print "Interior return value: ";
print $e;
print "Returned value: ";
print $h;
```

TestInput_v7.txt

```
object1 { include "TestInput_datasource.txt" }
;


function foo1 ($a, $b, $c)
```

```
{
   $x = (($a * $b) + $c);
   print "Foo1 . .";
   print "A:";
   print $a;
   print "B:";
   print $b;
   print "C:";
   print $c;
   return $x;
};

$d = 3;
$e = 4;

print "Returned value: ";
$f = object1.foo1 ($d, $e, 10);
print $f;
$f = object1.foo1 (1, 3, 10);
print $f;
print object1.foo1 (42, $e, 18);
print $f;
print "The original table:";
print object1;
```

TestInput_v8.txt

```
object1 { include "TestInput_datasource.txt" };

print "THE ORIGINAL DATA SOURCE AND TABLE:";
print object1;

$a = 1;
$b = 5;
$c = 0;

function HW_ave ($i)
{
   print "<font size = +2>Average on HW";
   print $i;
   print ":</font><br>";
   print "<font size = +1>";
   print object1.average($i + 1, true);
   print "</font>";
   print "<br>";
};

for $a TO $b
{
   $c = $c + 1;
   object1.HW_ave($c);
};
```

Walke.g

```
/*
      walker.g : the AST walker.

*/

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

class BecyAntlrWalker extends TreeParser;
options{
        importVocab = BecyAntlrLexer;
}

{
        static BecyDataType null_data = new BecyDataType();
        BecyInterpreter ipt = new BecyInterpreter();
}

expr returns [ BecyDataType r ]
{
        BecyDataType a, b;
        Vector v;
        BecyDataType[] x;
        String s = null;
        r = null_data;
}
              : #(PROGRAM (program:. { r = expr(#program); } )* )
               | #(OR a=expr right_or:.)
                      {
                              if ( a instanceof BecyBool )
                                      r = ( ((BecyBool)a).bool ? a : expr(#right_or) );
                              else
                                      r = a.or( expr(#right_or) );
                      }
              | #(AND a=expr right_and:.)
                      {
                              if ( a instanceof BecyBool )
                                      r = ( ((BecyBool)a).bool ? expr(#right_and) : a );
                              else
                                      r = a.and( expr(#right_and) );
                      }
              | #(ASGN a=expr b=expr)              { r = ipt.assign( a, b ); }
```

```
		| #(GE a=expr b=expr)			{ r = a.ge( b ); }
		| #(LE a=expr b=expr)			{ r = a.le( b ); }
		| #(GT a=expr b=expr)			{ r = a.gt( b ); }
		| #(LT a=expr b=expr)			{ r = a.lt( b ); }
		| #(EQ a=expr b=expr)			{ r = a.eq( b ); }
		| #(NEQ a=expr b=expr)			{ r = a.ne( b ); }
		| #(PLUS a=expr b=expr)			{ r = a.plus( b );
}

		| #(MINUS a=expr b=expr)		{ r = a.minus( b ); }
		| #(MULT a=expr b=expr)				{ r = a.times( b ); }
		| #(DIV a=expr b=expr)			{ r = a.div( b ); }
		| #(MOD a=expr b=expr)			{ r = a.modulus( b ); }
		| #(UPLUS a=expr)			{ r = a; }
		| #(UMINUS a=expr)			{ r = a.uminus(); }
		| num:NUMBER				{ r = ipt.getNumber( num.getText() ); }
		| str:STRING				{ r = new BecyString( str.getText() ); }
		| "true"			{ r = new BecyBool( true ); }
		| "false"			{ r = new BecyBool( false ); }
		| id:ID					{ r = ipt.getObject( id.getText() ); }

		| #(NUM_VAR nv:ID)			{ r = ipt.getObject( nv.getText() ); }

		| #(STR_VAR sv:ID)			{ r = ipt.getObject( sv.getText() ); }

		| #(QUES a=expr thenp:. elsep:.)
			{
				if ( !( a instanceof BecyBool ) )
					return a.exception( "if: expression should be bool" );
				if ( ((BecyBool)a).bool )
					r = expr( #thenp );
				else if ( null != elsep )
					r = expr( #elsep );
			}

		/* Function Calls */
		| #(FUNC_CALL a=expr b=expr x=mexpr)
			{
			    try {
					r = ipt.funcInvoke( this, (BecyTable)a, b, x );
			    } catch (Exception e) {
					System.out.println("function,<"+b.getName()+"> did not evaluate
completely; Error: " + e);
				}
			}
		| #("for" x=mexpr forbody:.)
			{
			    double i = ((BecyNumber)x[0]).getNumber();
			    double j = ((BecyNumber)x[1]).getNumber();

				for (;i <= j; i++)
```

```
                                {
                                        r = expr( #forbody );
                                }
                        }
                | #(LINE_BLOCK (line:. { r = expr(#line); } )*)
                | #("return"
                                ( a=expr
                                                { r = ipt.rvalue( a ); }
                                )?
                          )
  { ipt.setReturn( null ); }
                | #("function" fname:ID x=mexpr fbody:.)
                        { ipt.funcRegister( fname.getText(), x, #fbody ); }
               /* Data definition - 2 cases */
               | #(DATA_DEF a=expr b=expr)
                        {
                               r = ipt.newDataObject(new BecyString(a.getName()), (BecyTable)b);
                        }
               /* 1. Importing a file. */
               | #("include" a=expr)
                         { r = ipt.include((BecyString)a); }
                /* 2. Creating a new data object.

               | #(DATA_LIST { v = new Vector(); }
                               ( a=expr { v.add(a); } )*
                        dtail:DATA_LIST)
                        {
                                  b = expr(#dtail);
                               ((BecyTable)b).push(v);
                         }
               */
               | #("print" a=expr)
                        {
                               a.print();
                        }
                ;

mexpr returns [ BecyDataType[] rv ]
{
        BecyDataType a,b;
        rv = null;
        Vector v;
}
                : #(ARG_LIST                            { v = new Vector(); }
                                ( a=expr            { v.add( a ); }
                                )*
                        )                               { rv = ipt.convertExprList( v ); }
                | a=expr                    { rv = new BecyDataType[1]; rv[0] = a; }
                | #("TO"                    { v = new Vector(); }
                               a=expr               { v.add(a); }
                b=expr          { v.add(b); }

               )                { rv = ipt.convertExprList( v ); }
        | #(PARAM_LIST                  { v = new Vector(); }
                ( a=expr        { v.add( a ); }
                )*
           )                            { rv = ipt.convertExprList( v ); }
        ;
```

Grammar.g

```
/*
   grammar.g : the lexer and the parser, in ANTLR grammar.
*/

class BecyAntlrLexer extends Lexer;

options{
      testLiterals = false;
      k = 2;
}

{
   int nr_error = 0;
   public void reportError( String s ) {
         super.reportError( s );
         nr_error++;
   }
   public void reportError( RecognitionException e ) {
         super.reportError( e );
         nr_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    : ')';
MULT       : '*';
PLUS       : '+';
MINUS      : '-';
DIV         : '/';
MOD         : '%';
SEMI       : ';';
LBRACE    : '{';
RBRACE    : '}';
ASGN       : '=';
COMMA      : ',';
GE           : ">=";
LE           : "<=";
GT           : '>';
LT           : '<';
EQ           : "==";
AND      : "&&";
OR        : "||";
NEQ         : "!=";
COLON      : ':';
AT         : '@';
DOLLAR             : '$';
DOT      : '.';
QUES    : '?';

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

/* NUMBER example:
    1, 1., 1.1
*/

NUMBER    : (DIGIT)+ ('.' (DIGIT)*)? ;

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


class BecyAntlrParser extends Parser;
```

```
options{
        k = 2;
        buildAST = true;
}

tokens {
      PROGRAM;
      LINE_BLOCK;
      FUNC_CALL;
      ARG_LIST;
      DATA_DEF;
      DATA_LIST;
      PARAM_LIST;
      UPLUS;
      UMINUS;
      NUM_VAR;
      STR_VAR;
}

{
    int nr_error = 0;
    public void reportError( String s ) {
            super.reportError( s );
            nr_error++;
    }
    public void reportError( RecognitionException e ) {
            super.reportError( e );
            nr_error++;
    }
}

program
                : ( line | func_def | data_def )* EOF!
                        {#program = #([PROGRAM,"ROOT"], program);}
                ;

line
              : statement SEMI!
              ;

statement
                : assignment
                | condition
              | for_stmt
                | func_call
              | print
              | rtrn
                ;

assignment
```

```
             : variable ASGN^ r_side
             ;

r_side
             : arith_expr
            | STRING
            ;

variable
            : DOLLAR! ID
               { #variable = #([NUM_VAR, "Num_Var"], variable); }
            | AT! ID
               { #variable = #([STR_VAR, "Str_Var"], variable); }
            ;

condition
             : LPAREN! expr RPAREN! QUES^ statement COLON! statement
             ;

for_stmt
            : "for"^ loop line_block
            ;

loop
            : arith_expr "TO"^ arith_expr
            ;

line_block
            : LBRACE! (line)* RBRACE!
                   {#line_block = #([LINE_BLOCK,"Block:"], line_block);}
            ;

expr
            : logic_term (OR^ logic_term)?

            ;

logic_term
            : logic_comp (AND^ logic_comp)?
            ;

logic_comp
            : logic_factor ((GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) logic_factor)?
            ;

logic_factor
            : t
            | f
            | arith_expr
            ;
```

```
t
            : "true"
            | "TRUE"
            | "1"
            ;

f
            : "false"
            | "FALSE"
            | "0"
            ;

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

arith_term
            : factor (( MULT^ | DIV^ | MOD^ ) factor)*
            ;

factor
            : num
            | func_call
            | variable
            | LPAREN! expr RPAREN!
            ;

num
            : (UPLUS^ | UMINUS^)? NUMBER
            ;

func_call
             : object:ID DOT! name:ID func_cont
               { #func_call = #([FUNC_CALL, "Function_Call"], func_call); }
             ;

func_cont
            : LPAREN! arg_list RPAREN!
            | LBRACE! data RBRACE!
            ;

arg_list
            : ( STRING ( COMMA! arg_op )* )?
               { #arg_list = #([ARG_LIST, "Argument_List:"], arg_list); }
            | (arith_expr
arg_tail)?

               { #arg_list = #([ARG_LIST, "Argument_List:"], arg_list); }
            | (t
arg_tail)?
```

```
                { #arg_list = #([ARG_LIST, "Argument_List:"], arg_list); }
            | (f
arg_tail)?

                { #arg_list = #([ARG_LIST, "Argument_List:"], arg_list); }
            ;

arg_tail
            : ( COMMA! arg_op )*
            | "TO"^ arith_expr
            ;

arg_op
            : STRING
            | arith_expr
            | t
            | f
            ;

print
            : "print"^ ( variable | func_call | ID | num | STRING )
            ;

data_def
            : ID LBRACE! data RBRACE! SEMI!
                { #data_def = #([DATA_DEF, "Data_Definition:"], data_def); }
            ;

data
            : data_list
            | "include"^ STRING
            ;

data_list
             : ( STRING | num )+ (COMMA! data_list)*

                { #data_list = #([DATA_LIST, "Data_List:"], data_list); }


             ;

func_def
            : "function"^ ID LPAREN! param_list RPAREN! line_block SEMI!
            ;

param_list
            : ( variable ( COMMA! variable )* )?
                { #param_list = #([PARAM_LIST, "Parameter_List:"], param_list); }
            ;
rtrn
             : "return" expr
             ;
```