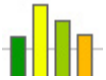


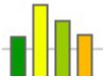
EZGraphs

A graphs and charts generating language

Final Report
05/07/07

Team:
Vincent Dobrev **vd2006@columbia.edu**
Edlira Kumbarce **ek2248@columbia.edu**

COMS W4115: Programming Languages and Translators
Prof. Stephen A. Edwards
Spring 2007



Chapter 1

Introduction

1.1 Background

Information graphics are visual representations of data, knowledge and information. They are used anywhere information needs to be explained simply and quickly, especially in mathematics, statistics and computer science where they aid in the process of developing and communicating conceptual information and scientific data. The first information graphics date back to early humans and their cave paintings, and later can be seen in the design of maps. The first data graphs appear in William Playfair's book, *The Commercial and Political Atlas*, in 1786.

Graphs such as bar and pie charts visually communicate data that, if presented in words or in tables, could appear cumbersome and difficult to understand. All sorts of complicated information can be presented using graphs, ranging form a simple curve representing the relationship of temperature and time to a bubble and line hierarchical graph depicting the HTML structure of a website. Because of the intrinsic simplicity of graphs in presenting scientific information, they are introduced to children in schools very early on.

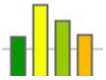
1.2 Objective

Our goal is to design a language that can be used to create graphs. Although there are various tools available for generating graphs, most of these tools are not very intuitive and require a certain amount of learning before they can be used. EZGraphs aims in becoming a simple way for any user with a little programming knowledge to draw a graph. EZGraphs has a more organized structure than Ploticus and provides more design options than the graphs wizard in Microsoft Excel. It strives to be user-friendly with clear, unadorned syntax rules and informative error messages for easy debugging, so that anyone writing EZGraphs scripts finds the experience pleasant and relatively effortless.

1.3 Features

Although EZGraphs was designed to be simple and easy to use, it offers many features that give the user the power to write programs that perform from simple arithmetic calculations to complicated sorting algorithms, all of which complement the ability to design chart and graph images.

1.3.1 Simple Syntax



The target users for this language are those with a little prior programming experience in two of the most popular languages of today, C and Java. Therefore, the syntax of our language are as close to C and Java as possible. Most keywords and pre-defined function name in the language aim to be intuitive. Each would be the first thing that comes to mind when thinking about the specific object or action it corresponds to. This makes the language syntax very easy to learn and memorize.

1.3.2 Easy Debugging

As programmers, we know that the amount of information we are given in the case of an error makes a world of difference, particularly so when we're learning a new language or trying to use a new application. Therefore, EZGraphs will have non-cryptic and informative error messages that will make it easy to use and debug. Aside from an informative text message indicating the error, the user will be given a line and column number of where the error occurred, as well as the name of the file that contains the offending piece of code. This will take any guesswork out of determining the error allowing the programmer to focus on how to correct the error.

1.3.3 Portability

Because is it based on Java, EZGraphs is very portable and can be used on machines running Windows, Linux, or Mac OS. It constitutes of a few classes which can very easily be put in a single JAR file. The tool can then be used on any computer that has Java Runtime Environment installed.

1.3.4 Data Types

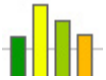
EZGraphs offers a variety of data types. It supports boolean types for easy logical operations. The presence of both integer and floating-point data types gives the user more flexibility and the power to perform numerous mathematical operations. The language also supports character strings. Another feature that is particularly important for graphing, which needs sets of data, is the support for arrays. Arrays can be of any of the basic data types mentioned previously and can have many dimensions.

1.3.5 File Inclusion

To promote modularity and code reuse, as well as to give the programmer the option of better organizing the code, EZGraphs allows file inclusion. This will make it possible for code to be written once and then reused as a library.

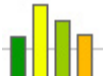
1.3.6 Control Flow Statements

The language allows the programmer to direct the flow of control in a program. It supports conditional (if-then-else) and iterative (looping) statements, which in turn provide the opportunity to create recursive functions.



1.3.7 Simple Drawing Functions

Its main goal being the generation of graphs and charts, EZGraphs provides simple functions for drawing. However, these functions are generic enough that they do not limit drawing to graphs. Instead, they are used to draw shapes and text in a canvas, this way leaving the elaborateness of the images drawn up to the user's imagination and creativity.



Chapter 2

Tutorial

2.1 Running EZGraphs

To execute EZGraphs programs, first write the code in a file and save it with the .ezg extension. Then, run the following command:

```
java ezg.EzgMain [OPTION]... FILE
```

The `FILE` argument specifies the .ezg file that contains the EZGraphs source code. The system will interpret the code and generate output accordingly. The interpreter can be run with various options (switches) that allow specifying the amount of information printed to the console as the program is running, setting the size of the tab character, etc. Use the `--help` option to see a list of all the available options.

The output of an EZGraphs program can be text on the console, an image file, an image displayed in a window on-screen, or a combination of any of these options, depending on what output functions are called.

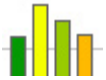
The interpreter will terminate execution automatically after the program has executed completely, unless there is an image window showing, in which case it will terminate once the window is closed.

2.2 Non-Graphing Operations

Although the main purpose of this language is to be used for drawing charts and graphs, EZGraphs programs can be written to perform various other operations.

For example, a small program that computes the factorial of a number can be written as follows:

```
/*
 * Program that calculates the factorial of a number.
 */

void main() {
    int num = 5;
    println(fact(num));
}
int fact(int n) {
    if (n <= 1)
        return 1;
    else
        return n * fact(n-1);
}
```



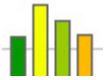
Execution starts in the `main()` method, where a new integer variable `num` is declared and initialized to a value of 5. The pre-defined function `println()` is then called, which prints its parameter to the console. Before printing, the value of the parameter has to be evaluated. The parameter passed to `println()` in this example is a user-defined function called `fact()`. The function `fact()` takes an integer as a parameter. After it executes, calling itself recursively, it will return to where it was called in `main()`. The value returned by `fact()` will then be printed to the console, and the program execution will terminate. The output is:

```
120
```

EZGraphs can also be used to sort a list of elements stored in an array, as shown in the simple implementation of a Bubble Sort algorithm in the following program:

```
/*
 * Implementation of a Bubble Sort algorithm.
 */

void main() {
    int[] a = new int[6];
    a[0] = 5;
    a[1] = 2;
    a[2] = 4;
    a[3] = 1;
    a[4] = 3;
    a[5] = 4;

    println(a);
    bubbleSort(a);
    println(a);
}

void bubbleSort(int[] a) {
    bool swapped;
    int n = size(a), temp;

    do {
        swapped = false;
        n--;
        for (int i = 0; i < n; i++) {
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                swapped = true;
            }
        }
    } while (swapped);
}
```

The array a is declared and initialized in `main()`. It is then passed to the `println()` function, which will print a string representation of the array contents on the console. Then, function `bubbleSort()` is called with the array as a parameter. Inside `bubbleSort()`, we use loops to make iterative passes through the array elements, swapping them as necessary until all the elements are in order.



The output is:
```
{5,2,4,1,3,4}
{1,2,3,4,4,5}
```

2.3 *Reading a Data File*

While EZGraphs code can be written to generate sets of data that are then converted into charts or graphs, this language provides for a simple way to read data from an external file. The data that is read from the file is stored into a two-dimensional `string` array, and can then be manipulated and used in constructing graphs and charts. The data in the file is expected to be formatted such that each data element is separated by a delimiting character, and each record composed of data elements is in a separate line. For example, a file "students.txt" used in the example would look like the following:

```
1,Jane Doe,jd2007,COMS W4115
2,John Smith,js1234,COMS W4111
3,Adam Brown,acb25,MECE E6700
4,Mary Stones,ms2250,IEOR E4407
```

The data elements are delimited by commas, and each line represents data belonging to one student.
The following program reads the data from the file, stores it into an array, and prints the array contents.

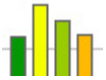
```
/*
 * Reading data from an input file into an array.
 */

void main() {
    string [][] data;
    data = data("samples\\students.txt", ",");
    println(data);
}
```

The pre-defined function `data()` is passed the relative path to the data file and the delimiting character. It reads all the data and returns it in the array `data`.
The output is:
```
{{1,Jane Doe,jd2007,COMS W4115},{2,John Smith,js1234,COMS W4111},
{3,Adam Brown,acb25,MECE E6700},{4,Mary Stones,ms2250,IEOR E4407}}
```

2.4 *Drawing*

Making use of the simple drawing pre-defined functions that EZGraphs offers, users can write programs that draw many things other than graphs and charts.

The following EZGraphs program uses recursion to generate a Sierpinski Triangle fractal.



```
/*
 * Sierpinski Triangle.
 */

void main() {
    canvas(600, 600);
    scale(1, -1);
    translate(0, -600);
    triangles(50, 50, 550, 50, 300, 550, 1);
    show();
}

void triangles(int x1, int y1, int x2, int y2,
               int x3, int y3, int level) {
    line(x1, y1, x2, y2);
    line(x2, y2, x3, y3);
    line(x3, y3, x1, y1);

    int xp1 = (x2+x1) / 2;
    int yp1 = (y2+y1) / 2;
    int xp2 = (x3+x2) / 2;
    int yp2 = (y3+y2) / 2;
    int xp3 = (x1+x3) / 2;
    int yp3 = (y1+y3) / 2;

    if (level < 8) {
        triangles(x1, y1, xp1, yp1, xp3, yp3, level+1);
        triangles(xp1, yp1, x2, y2, xp2, yp2, level+1);
        triangles(xp3, yp3, xp2, yp2, x3, y3, level+1);
    }
}
```

To begin drawing, a canvas needs to be defined first. In this example, a new canvas with dimensions 600 x 600 pixels is defined using the pre-defined function `canvas(600, 600)`. By default, the origin (point (0,0)) of the coordinate system in the canvas is at the upper-left corner. The x-coordinates of a point in the canvas increase from left to right, while the y-coordinates increase from top to bottom. Sometimes, this is not the desired setup. In the program above, we're using a combination of two pre-defined transformations: A call to `scale(1, -1)` will cause the y-coordinates to increase from bottom to top, with no effect to the x-coordinates; a call to `translate(0, -600)` will then move all points in the canvas down 600 pixels, causing the location of the origin (0,0) to be the lower-left corner of the canvas. Once this is set up, the user-defined recursive function `triangles()` is called, which takes as parameters integers indicating the (x,y) coordinates for the three corners of a triangle, as well as the current level we are in during the recursive run. Each time inside `triangles()`, the pre-defined function `line()` is called three times to draw three lines, taking as parameters the (x,y) coordinates of the two endpoints of the line. This will cause a triangle to be drawn on the canvas. As the function runs recursively, it will keep drawing triangles at the midpoints of the edges of bigger triangles, until the level reaches to 8, constructing this was a Sierpinski Triangle fractal.
The pre-defined function `show()` is called in the end of the program to show the canvas in a frame on-screen. The output is shown in Fig. 2.1.

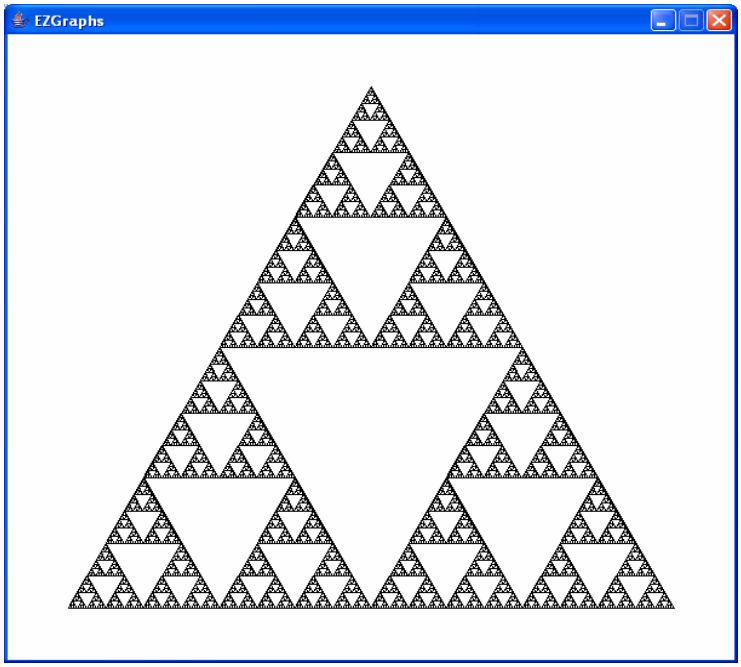
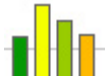


Fig. 2.1: Sierpinski Triangle

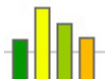
2.5 *A Pie Chart*

Pie charts can easily be generated using a small EZGraphs program. An example is shown below.

```
/*
 * A pie chart illustrating distribution of income.
 */

void main() {
    int fields = 7;

    /* Colors. */
    int[][] c = new int[fields][3];
    c[0][0] = 255; c[0][1] = 50;  c[0][2] = 0;
    c[1][0] = 0;   c[1][1] = 255; c[1][2] = 0;
    c[2][0] = 0;   c[2][1] = 128; c[2][2] = 255;
    c[3][0] = 255; c[3][1] = 128; c[3][2] = 0;
    c[4][0] = 255; c[4][1] = 0;   c[4][2] = 128;
    c[5][0] = 0;   c[5][1] = 128; c[5][2] = 0;
    c[6][0] = 255; c[6][1] = 255; c[6][2] = 0;

    /* Labels. */
    string[] l = new string[fields];
    l[0] = "Savings";
    l[1] = "Insurance";
    l[2] = "Entertainment";
    l[3] = "Auto";
    l[4] = "Clothing";
```



```
l[5] = "Food";
l[6] = "Housing";

/* Percentages. */
int[] p = new int[fields];
p[0] = 9;
p[1] = 11;
p[2] = 12;
p[3] = 19;
p[4] = 15;
p[5] = 14;
p[6] = 21;

canvas(400,400);
background(244,244,244);

string title = "Income Distribution";
font("Arial", 0, 20);
int width = width(title);
text(title, 200 - width/2, 40);

stroke(2);
line(50,50,350,50);

translate(100,100);

int start = 0, end = 0; /* Angles. */
for (int i = 0; i < fields; i++) {
    color(c[i][0],c[i][1],c[i][2]);
    start += end;
    end = p[i] * 360 / 100;
    fillArc(0,0,200,200,start,end);
}

font("Verdana", 1, 10);

/* Savings. */
color(c[0][0],c[0][1],c[0][2]);
text(l[0] + " (" + p[0] + "%)",200,65);
/* Insurance. */
color(c[1][0],c[1][1],c[1][2]);
text(l[1] + " (" + p[1] + "%)",165,20);
/* Entertainment. */
color(c[2][0],c[2][1],c[2][2]);
text(l[2] + " (" + p[2] + "%)",35,-10);
/* Auto. */
color(c[3][0],c[3][1],c[3][2]);
text(l[3] + " (" + p[3] + "%)",-60,50);
/* Clothing. */
color(c[4][0],c[4][1],c[4][2]);
text(l[4] + " (" + p[4] + "%)",-80,150);
/* Food. */
color(c[5][0],c[5][1],c[5][2]);
text(l[5] + " (" + p[5] + "%)",30,215);
/* Housing. */
color(0,0,0);
text(l[6] + " (" + p[6] + "%)",190,160);
```



```
        show();
    }
```

This example, which constructs a pie chart to depict distribution of income, makes use of arrays to store colors, labels, and percentages for each slice of the pie. It then uses pre-defined drawing and transformation functions to render the chart shown in Fig. 2.2.

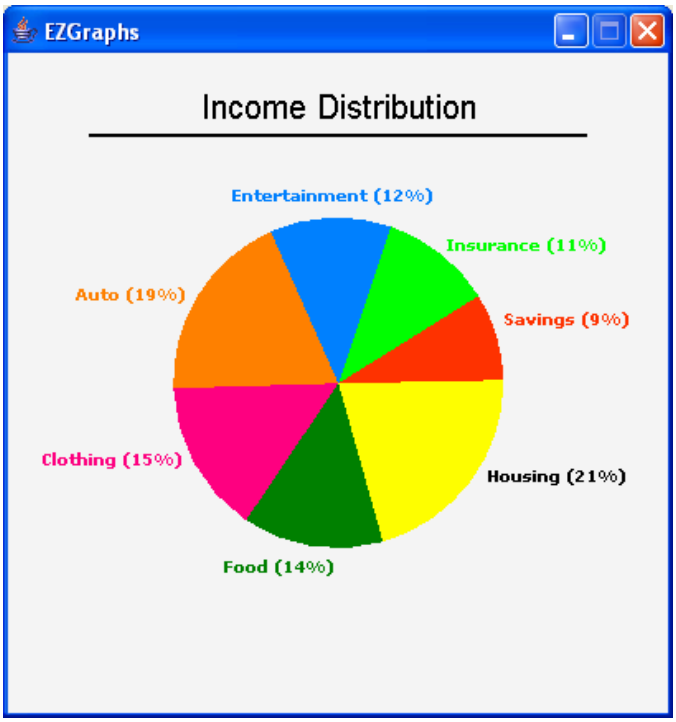
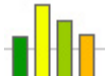


Fig. 2.2: Sample Pie Chart

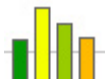
2.6 *A Bar Chart*

Attractive bar charts can also be constructed using EZGraphs. The following example shows how data for the chart is read from an input file and stored in an array using the data() function. User-defined functions `setOrigin()`, `xAxis()`, and `yAxis()` are used to draw the axes and set up the coordinate system.

```
/*
 * A bar chart illustrating exam scores.
 */

int canvW = 550, canvH = 200; /* Canvas dimensions. */
float origX = 50, origY = 50; /* Origin coordinates. */
string[][] data;

void main() {
    int xMin = 0, xMax = 450, yMin = 0, yMax = 100;
```



```
    /* Read data from the input file. */
    data = data("samples\\scores.txt", ",");

    canvas(canvW, canvH);

    color(100, 100, 100);
    string title = "Student Exam Scores";
    font("Verdana", 1, 18);
    int width = width(title);
    text(title, canvW/2 - width/2, 20);

    font("Times", 0, 10);
    color(0, 0, 0);
    setOrigin();
    xAxis(xMin, xMax);
    yAxis(yMin, yMax);

    for (int i = 0, j = xMin; i < size(data); i++, j = j + 50) {
        color(200+i*6, 0, 100+i*10);
        fillRect(j+35, 0, 30, strToInt(data[i][1]));
        push();
        reset();
        translate(origX, canvH-origY);
        text(""+data[i][1], j+50, -strToInt(data[i][1])-5);
        pop();
    }

    show();
}

void setOrigin() {
    scale(1.0, -1.0);
    translate(origX, origY-canvH);
}

void xAxis(int min, int max) {
    line(min, 0, max, 0);
    for (int i = min, j = 0; i <= max; i = i + 50) {
        line(i, -2, i, 0);
        if (j < size(data)) {
            push();
            reset();
            translate(origX, canvH-origY);
            text(data[j][0], (i+50)-width(data[j][0])/2, 10);
            pop();
            j++;
        }
    }
}

void yAxis(int min, int max) {
    line(0, min, 0, max);
    for (int i = min; i <= max; i=i+10) {
        line(-2, i, 0, i);
        push();
        reset();
        translate(origX, origY);
```



```
            text(""+i, -15, max-i);
            pop();
        }
    }
```

The image shown on screen after calling show() is illustrated by Fig. 2.3.

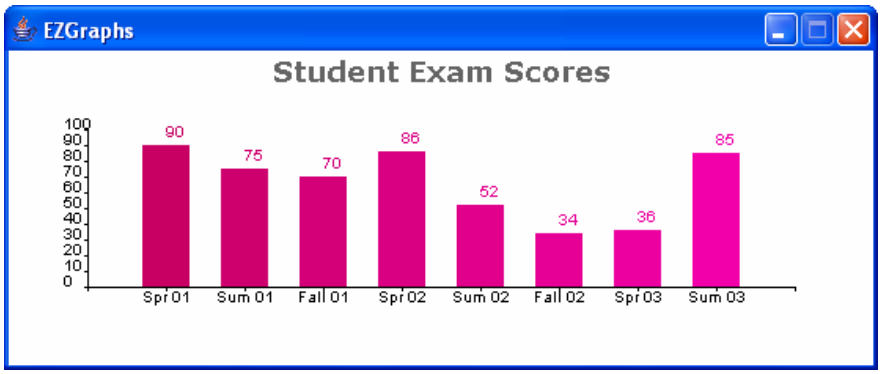
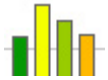


Fig. 2.3: Sample Bar Chart

2.7 *A Point Graph*

EZGraphs can be used to generate plots of functions. In the following example, the function y = x^3 / 1000 is plotted with points indicating (x,y) pairs.

```
/*
 * A point plot of function y = x^3/1000.
 */

int canvW = 300, canvH = 300;       /* Canvas dimensions. */
float origX = canvW/2, origY = 100; /* Origin coordinates. */

void main() {
    int xMin = -300, xMax = 300, yMin = -100, yMax = 300;

    canvas(canvW,canvH);

    string title = "y = x^3/1000";
    font("Verdana", 1, 10);
    text(title, 10, 20);

    font("Times", 0, 10);
    color(0, 0, 0);

    setOrigin();
    xAxis(xMin, xMax);
    yAxis(yMin, yMax);

    stroke(4);
    color(0,0,255);
    int y;
    for (int x = xMin; x <= xMax; x++) {
```



```
            y = x * x * x / 1000;
            point(x, y);
        }
        show();
    }

    void setOrigin() {
        scale(1.0, -1.0);
        translate(origX, origY-canvH);
    }

    void xAxis(int min, int max) {
        line(min, 0, max, 0);
        for (int i = min; i <= max; i=i+10) {
            line(i, -2, i, 2);
            if (0 == i%4) {
                push();
                reset();
                translate(origX, canvH-origY);
                text("" + i, i-width(""+i)/2, 10);
                pop();
            }
        }
    }

    void yAxis(int min, int max) {
        line(0, min, 0, max);
        for (int i = min; i <= max; i=i+10) {
            line(-2, i, 2, i);
            if (0 == i%4 && 0 != i) {
                push();
                reset();
                translate(origX, origY - canvH);
                text(""+i, -20, canvH - i + max - 200);
                pop();
            }
        }
   }
```

The output generated on-screen is shown in Fig. 2.4.

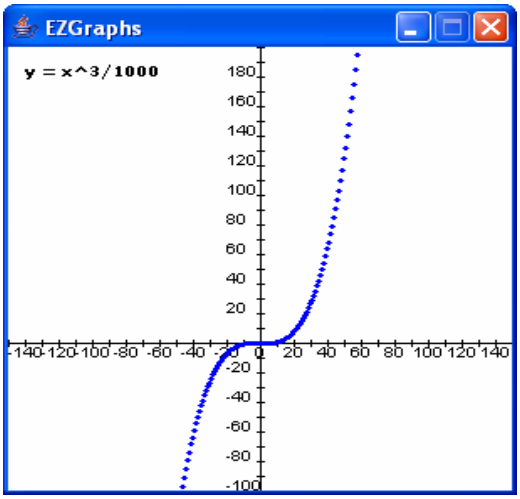
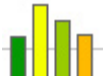


Fig. 2.4: Plot of y = x^3 / 1000



Chapter 3

Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

Single-line comments begin with the characters "`//`" and terminate with an end-of-line marker. Multi-line comments start with the characters "`/*`" and end with "`*/`". Comments cannot be nested. They are recognized during lexical analysis and then ignored.

3.1.2 Identifiers

An identifier consists of any sequence of letters, digits, and underscores. Its first character must be a letter or an underscore. Identifiers are case-sensitive.

3.1.3 Keywords

Some identifiers have special meaning in the language. They are reserved as keywords and cannot be used otherwise. The following identifiers are considered keywords:

```
include         break
void            continue
bool            return
int             if
float           else
string          while
true            do
false           for
```

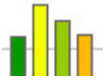
3.1.4 Booleans

Boolean constants can only have the values `true` or `false`.

3.1.5 Integers

Integer constants are represented by any sequence of digits.

3.1.6 Floats



Floating point constants consist of any one of the following sequences:

- An integer, a decimal point "."
- An integer, a decimal point ".", a fractional part
- An integer, a decimal point ".", an exponent part
- An integer, a decimal point ".", a fractional part, an exponent part
- An integer, an exponent part
- A decimal point ".", a fractional part
- A decimal point ".", a fractional part, an exponent part

The exponent part consists of either "e" or "E", followed by an optionally signed integer.

3.1.7 Strings

A string constant is any sequence of characters surrounded by double quotes "". Within a string constant a double quote must be escaped by a back-slash "\" as do the following characters:

"\"	back-slash itself
"\n"	line feed
"\t"	horizontal tab

3.1.8 Line Terminators

Lines are terminated by either one of "\r\n", "\r", or "\n". These characters are recognized and then ignored.

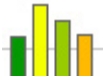
3.1.9 Whitespace

The blank space and tab characters are considered whitespace and serve to separate tokens. Like comments and line terminators, whitespace is recognized during lexical analysis and then ignored.

3.1.10 Operator Tokens

Operators are special symbols that perform specific operations on operands and return a result. They may be unary or binary and are evaluated based on an order of precedence, which is indicated in the table that follows, in descending order.

Operator Type	***Operators***
unary	+ - ! ++ --
multiplicative	* / %
additive	+ -
relational	< > <= >=
equality	== !=
logical AND	&&
logical OR	||



assignment	=	+=	-=	*=	/=	%=

3.1.11 Separator Tokens

The following symbols are used to separate tokens or combinations of tokens:

```
{   }   (   )   [   ]   ;   ,
```

3.2 Data Types

The following data types are supported in the language:

	Basic Types
`bool`	a boolean value (`true` or `false`)
`int`	a 32-bit integer
`float`	a single-precision floating point
`string`	a sequence of characters
	Derived Types
`array`	a set of values of a basic type

3.3 Arrays

An array is a list of items ordered sequentially. Each item in the array can be accessed using an index indicating its location in the list. The index starts from zero. The number of items in an array determines its length. The length of an array is established when the array is created and is fixed from that point on. The items of the array can be of any of the basic types; however, they must all be of the same type.

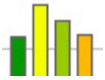
A two-dimensional array can be created by making an array of arrays of a basic type. The "inner" arrays need not be of the same size, but they have to be of the same type. The same process can be applied recursively to create multi-dimensional arrays.

3.4 Operators and Type Conversions

Different operators can be applied to constants and variable of certain types. The rules are presented in the following tables:

Logical not: `!`

	`!`
`array`	
`bool`	✓
`int`	
`float`	
`string`	



Prefix / postfix increment / decrement; unary plus / minus: `++ -- + -`

	++	--	+	-
array				
bool				
int	✓	✓	✓	✓
float	✓	✓	✓	✓
string				

Subtraction / multiplication / division / modulo: `- * / %`

a \ b	array	bool	int	float	string
array					
bool					
int			✓	✓	
float			✓	✓	
string					

Addition / concatenation: `+`

a \ b	array	bool	int	float	string
array					
bool					
int			✓	✓	
float			✓	✓	
string	✓	✓	✓	✓	✓

Relational: `< > <= >=`

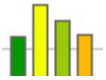
a \ b	array	bool	int	float	string
array					
bool					
int			✓	✓	
float			✓	✓	
string					✓

Equality / inequality: `== !=`

a \ b	array	bool	int	float	string
array					
bool		✓			
int			✓	✓	
float			✓	✓	
string					✓

Logical or / and: `|| &&`

a \ b	array	bool	int	float	string
array					
bool		✓			
int					
float					
string					



Assignment after subtraction / multiplication / division / modulo: `-= *= /= %=`

a \ b	array	bool	int	float	string
array					
bool					
int			✓	✓	
float			✓	✓	
string					

Assignment after addition / concatenation: `+=`

a \ b	array	bool	int	float	string
array					
bool					
int			✓	✓	
float			✓	✓	
string	✓	✓	✓	✓	✓

Assignment: `=`

a \ b	array	bool	int	float	string
array	✓				
bool		✓			
int			✓		
float			✓	✓	
string					✓

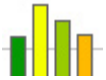
3.5 Expressions

Expressions are the core components of statements. They are constructed using operators, identifiers, constants, function calls, and other expressions. Expressions evaluate to some value in the end by applying operators to constants, function calls, or other expressions using the operator precedence rules as indicated in the section on operator tokens. An expression can be of one of the following:

primary-expression
new-expression
unary-expression
multiplicative-expression
additive-expression
relational-expression
logical-expression
assignment-expression

3.5.1 Primary Expressions

Primary expressions consist of l-values (identifiers or arrays), prefix/postfix increments and decrements of l-values, function calls, constants, or other expressions enclosed in parentheses.



Function calls are of the form

identifier (*expression-list*$_{opt}$)

where *identifier* is the name of the function and *expression-list* is an optional comma-delimited list of expressions to be passed as arguments to the function that is being called.

Constants can be boolean, character, integer, floating-point, or string constants.

3.5.2 New Expressions

New expressions are used to create arrays. They are of the form

new *base-type index-list*

base-type:
`bool`
`int`
`float`
`string`

index-list:
[expression]
[expression] index-list

3.5.3 Unary Expressions

Unary expressions are expressions prefixed by a unary operator. They are of the form

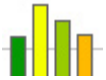
`+` *expression*
`-` *expression*
`!` *expression*

The unary + operator returns the value of *expression* unchanged. The unary – operator returns the negative value of *expression*. The logical negation operator `!` returns `false` if *expression* evaluates to `true` and it returns `true` if *expression* evaluates to `false`. Unary expressions associate from right to left.

3.5.4 Multiplicative Expressions

Multiplicative expressions consist of two *expression* operands and one of the multiplicative operators. They can be applied to integers and floats but not to booleans or strings. Multiplicative expressions are of the form

expression * *expression*
expression / *expression*
expression % *expression*



The * operator performs a multiplication on the two operands and returns the product. The / operator divides the second operand into the first, returning the quotient. The % operator returns the remainder of dividing the second operand into the second. They all group from left to right.

3.5.5 Additive Expressions

Additive expressions consist of two *expression* operands and one of the additive operators. The + operator can be applied to integers, floats, and strings, while the – operator only applies to integers and floats. Additive expressions are of the form

expression + *expression*
expression - *expression*

They group from left to right.

3.5.6 Relational Expressions

Relational expressions consist of two expression operands and one of the relational operators. They are applicable to integers and floats, and return a boolean value of `true` if:

==	the two operands are equal
!=	the two operands are not equal
<	the first operand is less than the second operand
>	the first operand is greater than the second operand
<=	the first operand is less than or equal to the second operand
>=	the first operand is greater than or equals to the second operand

3.5.7 Logical Expressions

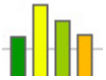
Logical expressions are only applicable to boolean values. They contain two *expression* operands and either the logical AND operator (&&) or the logical OR operator (||). The AND operator has higher precedence that the OR operator. The logical expressions associate from left to right and are of the form

expression && *expression*
expression || *expression*

3.5.8 Assignment Expressions

Assignment expressions consist of an l-value as a left operand, one of the assignment operators, and *expression* (whose type is the same as that of the l-value) as the right operand. They group from right to left. Assignment expressions evaluate to the value contained in the left operand and are of the form

lvalue = *expression*
lvalue += *expression*



lvalue -= expression
*lvalue *= expression*
lvalue /= expression
lvalue %/ expression

3.6 Declarations

Declarations have the form

type declarator-list

type:
    type-specifier brackets$_{opt}$

type-specifier:
    `bool`
    `int`
    `float`
    `string`

brackets:
    `[]`
    `[]` *brackets*

declarator-list:
    identifier variable-initialization$_{opt}$
    identifier variable-initialization$_{opt}$ `,` *declarator-list*

variable-initialization:
    `=` *expression*

3.7 Statements

Statements are building blocks of a program and are executed in the order they appear in the program provided control flow statements do not dictate jumping to different locations.

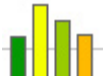
3.7.1 Empty Statement

A semicolon by itself is a statement.

3.7.2 Declaration Statement

A declaration followed by a semicolon (*declaration* `;`) is also a statement.

3.7.3 Conditional Statement



Conditional statements are used to make decisions at various points in a program. They are of the form

```
if ( expression ) statement
if ( expression ) statement else statement
```

where *expression* has to evaluate to either `true` or `false`. If *expression* evaluates to `true`, the first statement is executed. If it is `false`, the second statement is executed, if present. Conditional statements can be nested.

3.7.4 While Statement

The `while` statement is of the form

```
while ( expression ) statement
```

The *expression* must evaluate to a boolean and *statement* continues to be executed as long as expression is `true`. The test takes place before each execution of the *statement*.

3.7.5 Do Statement

The do statement is of the form

```
do statement while ( expression ) ;
```

The *expression* must evaluate to a boolean and *statement* continues to be executed as long as expression is `true`. The test takes place after each execution of the *statement*.

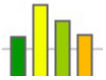
3.7.6 For Statement

The `for` statement is of the form

```
for ( for-clause ) statement

for-clause:
        declaration_opt ; expression_opt ; expression-list_opt
        expression-list_opt ; expression_opt ; expression-list_opt

expression_list:
        expression
        expression , expression_list
```

The statement is equivalent to

```
declaration ;
while ( expression ) {
        statement
```



```
        expression-list ;
    }
```

and to

```
    expression-list ;
    while ( expression ) {
        statement
        expression-list ;
    }
```

3.7.7 Break Statement

The statement

```
    break ;
```

is used to exit from iterative statements such as `while`, `do`, and `for`.

3.7.8 Continue Statement

The statement

```
    continue ;
```

is used to terminate only the current iteration of iterative statements such as `while`, `do`, and `for`.

3.7.9 Return Statement

A function returns to its caller by means of the `return` statement, which has the form

```
    return expression_opt ;
```

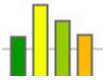
3.7.10 Expression Statement

An expression statement is of the form

```
    expression ;
```

3.7.11 Compound Statement

A block of zero or more statements enclosed in curly brackets is also a statement.



3.8 External Definitions

A program consists of a sequence of external definitions which are as follows:

3.8.1 Include Declarations

The include declaration

```
include "filename" ;
```

results in the replacement of that line by the entire content of the file *filename*. Includes can be nested. For example, the program file can include "lib1.ezg", and "lib1.ezg" can in turn include "lib2.ezg". However, the includes cannot form a circular reference to each other. That means, if "lib1.ezg" includes "lib2.ezg", the latter cannot include the former.

3.8.2 Function Definitions

Function definitions have the form

function-prefix function-body

function-prefix:
    `void` *identifier* (*argument-list*$_{opt}$)
    type identifier (*argument-list*$_{opt}$)

argument-list:
    type identifier
    type identifier , *argument-list*

function-body:
    { *statement-list*$_{opt}$ }

statement-list:
    statement
    statement statement-list

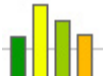
3.8.3 Declarations

Any *declaration* ; that appears outside the body of a function.

3.9 Scope Rules

The notion of static, open scoping will be applied as follows:

Variables defined in a program externally (outside of functions) are global. They are accessible from inside any function in any source file.



Each function in the program has its own scope. Variables defined inside a function can be accessed only within the function's body. The same rule applies for parameters to functions.

Blocks of statements enclosed in curly braces also have their own scope and variables defined inside each block are accessible only within the block.

A new scope is also started right before the head of a `for`-loop. It ends right after the body of the loop terminates.

Variables that have not been initialized are invalid even within their scope area and attempts to use them will generate "uninitialized variable" errors.

3.10 Predefined Functions

3.10.1 Data Acquiring Functions

```
string[][] data(string filename, string delims, bool header, bool quoted)
```

Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data; a `delims` string containing the character(s) separating each data element (token) in the file; a `header` boolean to indicate whether the data file has a header line, which would be ignored when retrieving the data; and a `quoted` boolean to indicate whether some of the tokens are enclosed by quotes, in which case any delimiter occurring within a pair of double-quotes is ignored and not used in splitting the data tokens. This function returns the 2D array containing the data.

```
string[][] data(string filename, string delims, bool header)
```

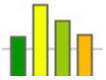
Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data; a `delims` string containing the character(s) separating each data element (token) in the data file; and a `header` boolean to indicate whether the data file has a header line, which would be ignored when retrieving the data. This function returns the 2D array containing the data.

```
string[][] data(string filename, bool header, bool quoted)
```

Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data; a `header` boolean to indicate whether the data file has a header line, which would be ignored when retrieving the data; and a `quoted` boolean to indicate whether some of the tokens are enclosed by quotes, in which case any delimiter occurring within a pair of double-quotes is ignored and not used in splitting the data tokens. The default delimiter set " \t\n\r\f" is used. This function returns the 2D array containing the data.

```
string[][] data(string filename, string delims)
```

Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data and a `delims` string containing the character(s)



separating each data element (token) in the file. The data file is assumed to have no header and no quoted tokens. This function returns the 2D array containing the data.

```
string[][] data(string filename, bool header)
```

Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data and a header boolean to indicate whether the data file has a header line, which would be ignored when retrieving the data. The default delimiter set " \t\n\r\f" is used and the data file is assumed to have no quoted tokens. This function returns the 2D array containing the data.

```
string[][] data(string filename)
```

Retrieves data from an input file and stores it into a 2D string array. It expects the name or path of the file containing the data. The default delimiter set " \t\n\r\f" is used. The data file is assumed to have no header and no quoted tokens. This function returns the 2D array containing the data.

3.10.2 Drawing Functions

```
void canvas(int width, int height)
```

Creates a new drawing canvas with the given dimensions. The new canvas overwrites the old one; therefore, all previous drawing will be lost.

```
void background(int r, int g, int b)
```

Clears the canvas with the specified color. The `r, g, and b` parameters represent the amounts of red, green, and blue the make up the color. Each parameter must be an integer between 0 and 255, inclusive.

```
void color(int r, int g, int b)
```

Sets the current drawing color. The `r, g, and b` parameters represent the amounts of red, green, and blue the make up the color. Each parameter must be an integer between 0 and 255, inclusive.

```
void stroke(int width)
```

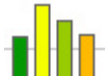
Sets the current drawing stroke width. Accepts an integer value from 1 to 10 representing the width of the stroke in pixels.

```
void font(string name, int style, int size)
```

Sets the current font used to draw text on the canvas. Style is defined as follows

0	plain
1	bold
2	italic
3	bold/italic

```
void point(int x, int y)
```



Draws a point (x,y) using the current stroke and color.

```
void line(int x1, int y1, int x2, int y2)
```

Draws a line starting at point (x1,y1) and ending at point (x2,y2) using the current stroke and color.

```
void rect(int x, int y, int width, int height)
```

Draws a rectangle with upper-left cornet at point (x,y) and specified width and height using the current stroke and color.

```
void fillRect(int x, int w, int width,int height)
```

Fills a rectangle with upper-left cornet at point (x,y) and specified width and height using the current color.

```
void polygon(int[] x, int[] y)
```

Draws a polygon with corners at points (x[i],y[i]) using the current stroke and color.

```
void fillPolygon(int[] x, int[] y)
```

Fills a polygon with corners at points (x[i],y[i]) using the current color.

```
void arc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

Draws an arc with the specified parameter using the current stroke and color.

```
void fillArc(int x, int y, int width, int height,
             int startAngle, int arcAngle)
```

Fills an arc with the specified parameter using the current color.

```
void oval(int x, int y, int width, int height)
```

Draws an oval with the specified parameter using the current stroke and color.

```
void fillOval(int x, int y, int width, int height)
```

Fills an oval with the specified parameter using the current color.

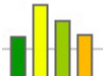
```
void text(string text, int x, int y)
```

Draws the specified text starting at point (x,y). It uses the current color and font.

3.10.3 Transformation Functions

The following functions add the corresponding transformation matrix to the current matrix

```
void scale(float sx, float sy)
void shear(float shx, float shy)
```



```
void rotate(float theta)
void translate(float tx, float ty)

void reset()
```

Reset the current transformation matrix to the identity matrix.

```
void push()
```

Push the current transformation matrix onto the stack.

```
void pop()
```

Pop the top transformation matrix from the stack.

3.10.4 Math Functions

The following math function are supported

```
float random()
int round(float a)
int floor(float a)
int ceil(float a)
int abs(int a)
float abs(float a)
float sqrt(float a)
float pow(float a, float b)
float exp(float a)
float log(float a)
float sin(float a)
float cos(float a)
float tan(float a)
float asin(float a)
float acos(float a)
float atan(float a)
```

3.10.5 Output Functions

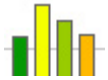
```
void print()
```

Prints to console the string representation of the parameter(s) passed to it. It accepts one or more parameters and prints them all in the same line.

```
void println()
```

Prints to console the string representation of the parameter(s) passed to it. It accepts zero or more parameters and prints each parameter in a new line. If no parameters are passed, it prints a blank line.

```
void save(string filename)
```



Saves the canvas contents to a .png file. It expects the name or relative/absolute path to the file. If the directories specified in the path do not exist, they will be created automatically. If only the name of the file has been supplied, the file will be created in the directory where the .ezg source file is run from.

```
void show()
```

Displays the canvas contents in a window.

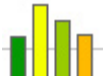
3.10.6 Auxiliary Functions

```
int strToInt(string str)
float strToFloat(string str)
int index(string str, string substr)
int lastIndex(string str, string substr)
string substring(string str, int begin, int end)
int size(string str)
int size(type[] array)

int width(string str)
```

Returns the width of the given string in points using the current font.

```
int height()
```

Returns the height of the any character in points using the current font.



Chapter 4

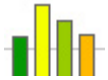
Project Plan

4.1 Planning

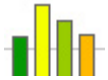
Since we were just two team members, we held regular ½ hour meetings (longer if discussions arose) to catch up to what each other had done the night before. We also had two meetings with Prof. Edwards through the semester course to show our progress and receive advice on how to proceed. As each of us completed a piece of code, we'd commit the new changes to the version control database.

4.2 Project Log

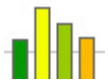
Rev	Date	User	Comment
1	2/11/07 12:01 AM	vincent	Initial repository layout
2	2/24/07 6:14 PM	vincent	Initial project layout
3	2/24/07 6:20 PM	edlira	Initial Lexer code
4	2/25/07 1:44 AM	vincent	Added Language Syntax and modified Lexer
5	2/25/07 5:49 PM	edlira	Added syntax rules for var-list, type-spec
6	2/26/07 1:07 AM	edlira	Added AST building annotations to parser rules; changed Main so it builds & outputs an AST
7	2/26/07 2:44 AM	vincent	Minor changes to Lexer, Parser, Main, and Language Syntax
8	2/26/07 7:46 PM	vincent	Fixed ID rule; removed some extra parenthesis; renamed type_decl to type_def
9	2/26/07 7:58 PM	vincent	Switched INT and EXP rules; added Main.printlnTokens()
10	2/27/07 12:17 AM	vincent	Fixed arg_list rule to add a tree node even if the arg_list is empty
11	3/02/07 12:39 AM	edlira	Adding Proposal.doc (final version) and LRM.doc (first draft)
12	3/02/07 1:05 AM	vincent	Fixed nondeterminism between INT, FLOAT, and DOT rules; added some Parser rules
13	3/04/07 2:09 AM	vincent	Added more syntactic Parser rules
14	3/04/07 3:42 AM	edlira	Added more expression-related parser rules; small changes in LRM.doc
15	3/04/07 11:58 PM	edlira	Added expression-related sections to LRM.doc; small changes in grammar.g
16	3/10/07 3:37 PM	vincent	Finalized LRM.doc before submitting; updated format of Proposal.doc
17	3/11/07 1:09 AM	edlira	Initial walker version (added file walker.g); minor changes to grammar.g & Main.java



18	3/17/07 7:35 PM	vincent	Lexer: fixed STRING and added xASGN, ESC and CHAR; Parser: added/updated rules
19	3/21/07 11:01 PM	vincent	Renamed all classes by adding prefix "Ezg" or "EzgAntlr"
20	3/22/07 12:21 PM	vincent	Added build.xml file
21	3/23/07 9:16 PM	edlira	Type classes; walker with only expressions
22	3/24/07 2:48 PM	vincent	Target clean in build.xml now deletes all files and subdirectories of classes
23	3/25/07 10:28 AM	edlira	Added test-related tasks
24	3/25/07 10:29 PM	vincent	Added rules to walker and fixed lexer, parser, and other classes as needed
25	3/28/07 7:39 PM	vincent	Walker now registers functions and variables; cleaned up lexer and parser
26	3/31/07 10:23 PM	vincent	Fixed func and var registration, func calls, and assignments; added scopes
27	4/01/07 1:16 AM	edlira	More test-related changes; added test-case files, and corresponding golden-ref files
28	4/02/07 11:45 PM	vincent	Fixed tests and expression evaluation for all operators on all types
29	4/03/07 12:38 AM	edlira	Started internal functions: added print(...), println(...), scope()
30	4/03/07 6:28 PM	vincent	Tests now run from a batch file called from build.xml
31	4/03/07 11:19 PM	vincent	Removed character support; deleted unused token DOT from lexer
32	4/03/07 11:33 PM	vincent	Added PLASGN, MINASGN, MULASGN, and DIVASGN to walker
33	4/04/07 1:40 AM	edlira	Support for "return" statements
34	4/04/07 11:59 PM	vincent	Added array declarations
35	4/05/07 10:40 PM	vincent	Escaped characters in strings are now correctly registered and displayed
36	4/07/07 6:12 PM	edlira	Added support for while/do loops and break/continue statements
37	4/07/07 8:21 PM	vincent	Added checks for assignments to non-lvalues and use of uninitialized variables
38	4/08/07 6:11 PM	edlira	Passing parameters to functions; placeholders for drawPoint and drawLine internals
39	4/08/07 11:03 PM	vincent	Fixed some bugs in arithmetic calculations; added test cases
40	4/12/07 1:40 AM	edlira	Added some initial drawing functionality (EzgPainter.java, drawXXX() internal functions)
41	4/12/07 3:24 PM	vincent	Fixed array declarations, assignments, and access (indexing)
42	4/15/07 1:13 AM	edlira	Added polygons and slices for drawing internals; worked on passing arrays as parameters to internals
43	4/20/07 10:14 PM	vincent	Added for loop; fixed bug with new_expr; removed endOfProgram() and labels
44	4/21/07 2:18 AM	vincent	Fixed array bug; added checks for statement and EzgStringTokenizer.java
45	4/21/07 8:14 PM	edlira	Internal functions: getData(), strToInt(), strToFloat(); tests for getData(); modulo operator



46	4/23/07 8:33 PM	vincent	Organized command line flags that EzgMain accepts
47	4/23/07 11:25 PM	edlira	Printing drawing info for verbose and test modes
48	4/24/07 12:00 AM	vincent	EzgMain now accepts only .ezg source files; added include declarations
49	4/24/07 12:58 AM	vincent	Made project compatible with Java 1.4; organized error messages
50	4/24/07 11:11 PM	vincent	Removed some command line flags; organized debug messages
51	4/25/07 10:36 PM	edlira	Added internal function saveGraph()
52	4/26/07 11:05 PM	vincent	Added filename:line:column in the beginning of error messages
53	4/26/07 11:34 PM	vincent	Moved source to package ezg; renamed package types to type; errors not fixed
54	4/26/07 11:50 PM	vincent	Fixed errors introduced with package changes
55	4/27/07 3:17 AM	edlira	Modified EzgPainter.saveGraph() to route invalid filename errors; added internal.ezg and scope.ezg test cases
56	4/28/07 5:33 PM	vincent	Fixed line and column values for some tree nodes
57	4/29/07 2:29 PM	edlira	Added code to detect filename correctness based on OS for saveGraph()
58	5/02/07 8:41 PM	edlira	Added length() internal function to get length of an array
59	5/02/07 9:54 PM	vincent	Made condition in for loop optional
60	5/05/07 4:20 PM	vincent	Added and organized internal functions
61	5/05/07 5:16 PM	vincent	Renamed some internal functions
62	5/05/07 10:46 PM	vincent	Added math internal functions; fixed bug in background()
63	5/06/07 2:49 AM	edlira	Added test cases for arrays, for-loops, drawing and transformation functions
64	5/06/07 10:55 AM	edlira	Added drawing test cases
65	5/06/07 3:49 PM	edlira	Updated the syntax file
66	5/06/07 4:14 PM	vincent	Ints are now promoted to floats when needed; floats cannot be assigned to ints
67	5/06/07 6:16 PM	vincent	Fixed int-to-float promotion for internal functions; added test cases
68	5/06/07 10:34 PM	edlira	Added sample EZGraphs programs
69	5/06/07 11:24 PM	vincent	Cleaned up some code; fixed a tests bug



Chapter 5

Architectural Design

The EZGraphs interpreter consists of the following main components:

- **Lexer**
  Gets an EZGraphs program and performs lexical analysis on it character-by-character. It produces a stream of tokens.
- **Parser**
  Gets the stream of tokens produced by the Lexer and performs syntactic analysis. It then produces an Abstract Syntax Tree.
- **Walker**
  Traverses the Abstract Syntax Tree, performs semantic analysis, and communicates back and forth with the Interpreter.
- **Interpreter**
  Directs control flow, looks up symbol tables, interacts with Walker, reads input files when necessary, and generates output accordingly.
- **Type System**
  Collection of data types, all inheriting from a base data type, each being aware of compatibility with different operators and the other data types.
- **Exception Handler**
  Captures, formats, and outputs error messages sent from any of the previous components.

The block diagram in Fig. 5.1 shows the components as well as the connections between them. A connecting arrow means transfer of control to the component the arrow points to. A connecting line indicates the classes communicate with each other.

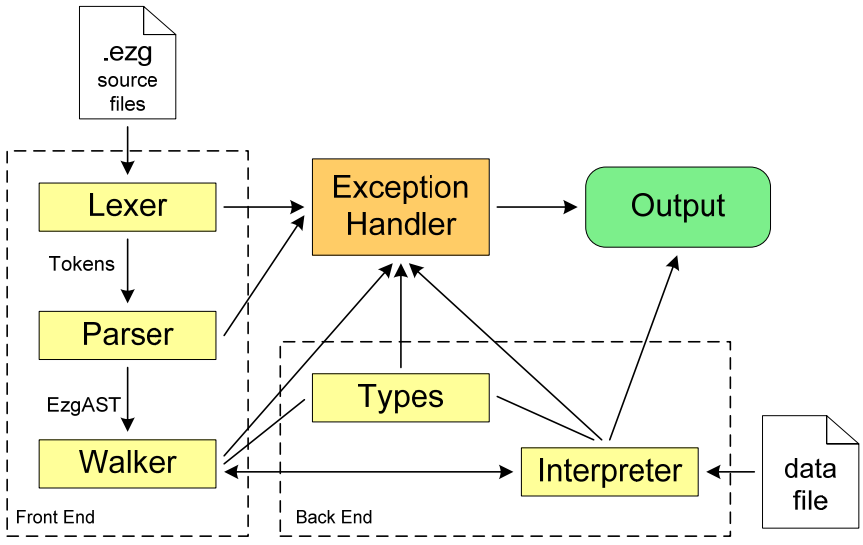
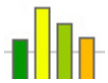


Fig. 5.1: Main architectural design block diagram



The front-end of the interpreter is implemented using ANTLR. The **Lexer** and **Parser** reside in file *grammar.g* while the **Walker** is in file *walker.g*. The **Parser** is instructed to use our own custom nodes (found in class *EzgAST*) when constructing the Abstract Syntax Tree. The custom nodes have added line, column and filename information, which is used to generate informative error messages for the user. The **Walker** traverses the Abstract Syntax Tree created by the **Parser** and constantly communicates and transfers control to and from the Interpreter, which is implemented in class *EzgInterpreter*.

When running an EZGraphs program, the starting point is the class *EzgMain*, which contains the main() method. It reads in the source file and any optional switches supplied by the user (for running the Interpreter with various options), and initializes the **Lexer**, **Parser**, and **Walker**. It is there that the custom node type is set for the Parser. *EzgMain* also contains the **Exception Handler** component that organizes, formats and prints error messages.

The back-end of the interpreter is composed of a **Type System** and the collection of classes that make up the core of the Interpreter.

The **Type System** consists of classes representing each of the various data types the language supports. It contains the base class *EzgType*, classes *EzgArray*, *EzgBool*, *EzgInt*, *EzgFloat*, *EzgString*, and *EzgVoid* (for functions returning a `void` data type) . All the other type classes inherit from the base class *EzgType*. The *EzgArray* class itself may contain any of the type classes within itself, including another *EzgArray*, except for the *EzgVoid* and *EzgType* classes. This is so the language supports multi-dimensional arrays of various types. Fig. 5.2 illustrates the hierarchical structure of the **Type System**.

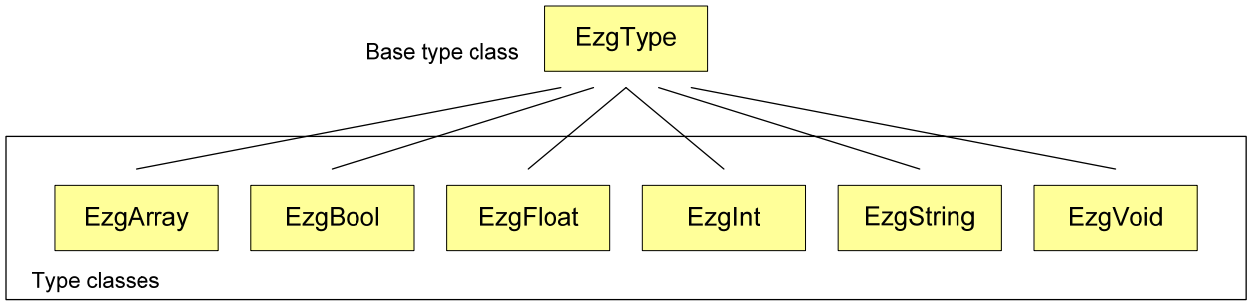
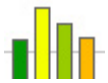


Fig. 5.2: Type System

The **Interpreter** component is itself composed of various classes. It starts from an *EzgInterpreter* class, which performs function execution, knows about variable scope and functions. The *EzgInterpreter* class has an *EzgSymbolTable*, where the variables are registered as they are declared, which serves as an activation record for the program as well. It also maintains a list of declared functions in a hash table, and functions are represented by the *EzgFunction* class, which knows about function name, return type, and arguments.

As it handles control flow, the *EzgInterpreter* class may transfer control to the class that handles pre-defined (internal) functions in the language, *EzgInternalFunctions*. The *EzgInternalFunctions* class in turn may get an input data file and use an EzgStringTokenizer class to parse the data. Based on the internal function called at a particular point, the *EzgInternalFunctions* class may output directly to the console, or transfer control to an



EzgPainter class, which contains methods to create images. The *EzgPainter* class in the end may either display an image in a frame on-screen, or create a .png file with the image. Fig. 5.3 shows the building blocks of the **Interpreter** component.

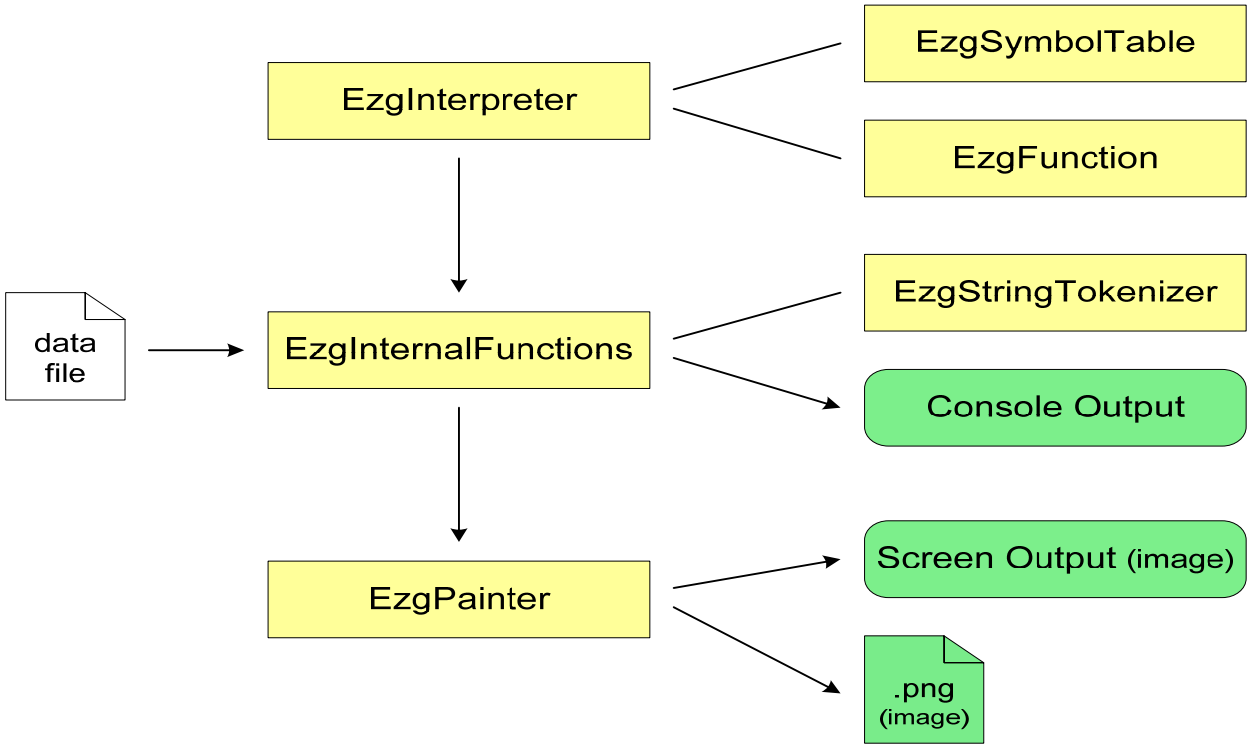
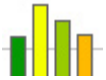


Fig. 5.3: Interpreter building blocks



Chapter 6

Test Plan

6.1 Idea

Our goal was to create a language that is simple and easy to use for those with a little programming experience. This meant reducing the number of bugs in the language to zero. Since an interpreter is a complicated and elaborate piece of software, it has a high probability for mistakes that might not be simple to catch or prevent. Therefore, we tried to test as much of the code as possible.

6.2 Unit Tests

We created small test cases that were run parallel to the development process. Each small test case was run to test the piece of the software that was being developed at that particular time. This allowed for early detection of bugs, and made this detection simple since it restricted the area that we needed to check for errors to just the one being developed at the time.

6.3 Regression Tests

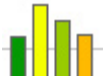
Because sometimes errors can occur not in separate sections of the code but rather when pieces of code is merged together, or when new code is added, we performed regression testing each time a new feature was added, this way making sure that the existing code worked well with the new code being added.

6.3.1 Sample Test Cases

Here are a couple of test samples:

1) Logical Operator Test Case

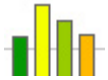
```
void main() {
    bool a = true, b = false;

    println(a && a);
    println(b && b);
    println(a && b);
    println(b && a);

    println(a || a);
    println(b || b);
```



```
    println(a || b);
    println(b || a);

    println(!a);
    println(!b);
}
```

2) Drawing Functionality Test Case

```
void main() {
    canvas(700, 700);
    points();
    lines();
    rectangles();
    ovals();
    show();
}

void points() {
    for (int i = 1; i <= 25; i++) {
        color(i*5, 10, 200+i*2);
        stroke(i);
        point(i*20, i*2);
    }
}

void lines() {
    stroke(1);
    for (int i = 0; i < 25; i++) {
        color(200+i/2, 0, 0);
        line(700, i*20, 400, 700);
    }

    color(0,0,0);
    for (int i = 10; i > 0; i--) {
        stroke(10-i);
        line(600, 700-i*10, 700, 700-i*10);
    }
}

void rectangles() {
    color(0, 150, 80);
    stroke(2);
    for (int i = 0; i < 50; i=i+10)
        rect(i+50, i+50, 100-i*2, 100-i*2);

    for (int i = 0; i < 40; i=i+6) {
        color(100+i*3, 37+i*5, 71+i*4);
        fillRect(i+200, i+50, 120, 60);
    }
}

void ovals() {
    color(13, 77, 18);
    stroke(3);
    for (int i = 0; i < 50; i=i+10)
        oval(i+450, i+150, 100-i*2, 100-i*2);
```

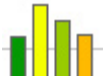


```
    for (int i = 0; i < 40; i=i+6) {
        color(100+i*4, 37+i*3, 71+i*2);
        fillOval(i+350, i+260, 120, 60);
    }
}
```

Running of test cases was automated using ANT build files and batch files that ran all the tests, captured output for each one, and compared the generated output to the golden references for each test. Drawing functionality was tested by printing out what the drawing function was doing along the way, and matching against a golden reference that listed the expected steps.

6.4 More Advanced Testing

We used our tutorial examples as more advanced testing cases, since they covered larger areas in the code, and the interaction between these areas. The examples are closer to the real-life programs expected from users, so they gave us more confidence in our language.



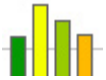
Chapter 7

Lessons Learned

Make sure that the tools you want to use with your system actually do what you expect them to do. This has to be done beforehand with simple testing programs before trying to integrate and use such tools with your compiler/interpreter, otherwise, you might have to make considerable changes to your software when you do not have enough time.

Testing is the most important part of developing large software as it is impossible to think about all possible things that can go wrong. This is even more important when working in a team, where each member might not know the others' code as well, which leads to even more bugs.

As this is a quite large project, start the soonest possible and have a sense of urgency from the beginning of the semester.



Appendix A

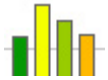
Language Syntax

A.1 Lexical Rules

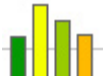
```
digit      -> '0'..'9'
letter     -> 'a'..'z' | 'A'..'Z'
int        -> (digit)+
exp        -> ('e' | 'E') ('+' | '-')? int
float      -> int ('.' (int)? (exp)? | exp) | '.' int (exp)?
id         -> (letter | '_') (letter | digit | '_')*
esc        -> '\\' ('n' | 't' | '"' | '\\')
string     -> '"' (esc | ~('"' | '\\' | '\n' | '\r'))* '"'
newline    -> '\r\n' | '\r' | '\n'
whitespace -> (' ' | '\t')+
comment    -> '/*' (~'*/')* '*/' | '//' (~newline)* newline
```

A.2 Syntactic Rules

```
/* External definitions. */
program       -> (extern_def)*
extern_def    -> include_decl | func_def | declaration ';'
include_decl  -> 'include' string ';'
func_def      -> func_prefix func_body
func_prefix   -> ('void' | type) id '(' arg_list ')'
arg_list      -> (type id (',' type id)*)?
func_body     -> '{' (statement)* '}'

/* Statements. */
statement     -> ';' | declaration ';' | if_stmt | while_stmt | do_stmt
                 | for_stmt | break_stmt | continue_stmt | return_stmt
                 | expression ';' | '{' (statement)* '}'
if_stmt       -> 'if' '(' expression ')' statement ('else' statement)?
while_stmt    -> 'while' '(' expression ')' statement
do_stmt       -> 'do' statement 'while' '(' expression ')' ';'
for_stmt      -> 'for' '(' for_clause ')' statement
for_clause    -> (declaration | expr_list) ';' for_cond ';' expr_list
for_cond      -> (expression)?
expr_list     -> (expression (',' expression)*)?
break_stmt    -> 'break' ';'
continue_stmt -> 'continue' ';'
return_stmt   -> 'return' (expression)? ';'
```



```
/* Declarations. */
declaration   -> type decl_list
type          -> type_spec ('[' ']')*
type_spec     -> 'bool' | 'int' | 'float' | 'string'
decl_list     -> id var_init (',' id var_init)*
var_init      -> ('=' expression)?

/* Expressions. */
expression    -> assignment
assignment    -> logic_expr (('=' | '+=' | '-=' | '*=' | '/=' | '%=')
                 assignment)?
logic_expr    -> logic_term ('||' logic_term)*
logic_term    -> equal_expr ('&&' equal_expr)*
equal_expr    -> relat_expr (('==' | '!=') relat_expr)?
relat_expr    -> arith_expr (('<' | '>' | '<=' | '>=') arith_expr)?
arith_expr    -> arith_term (('+' | '-') arith_term)*
arith_term    -> arith_factor (('*' | '/' | '%') arith_factor)*
arith_factor  -> ('++' | '--' | '+' | '-')? logic_factor
logic_factor  -> new_expr | ('!')? postfix_expr
new_expr      -> 'new' base_type index_list
base_type     -> type_spec
index_list    -> ('[' expression ']')+
postfix_expr  -> primary_expr (index_list)? ('++' | '--')?
primary_expr  -> id ('(' expr_list ')')? | constant | '(' expression ')'
constant      -> int | float | string | 'true' | 'false'
```



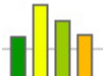
Appendix B

Code Listing

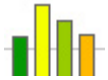
B.1 Parser

B.1.1 src/ezg/grammar.g

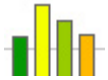
```
/*
 * Title:       grammar.g
 * Description: Lexer and parser for the EzgGraphs language.
 * Modified:    2007-05-06
 */

header {
    package ezg;
}

class EzgAntlrLexer extends Lexer;

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

tokens {
    INT;
    FLOAT;
}

{
    public void reportError(String s) {
        throw new RuntimeException(s);
    }

    public void reportError(RecognitionException e) {
        throw new RuntimeException(e.toString());
    }
}

LBRACE  : '{';
RBRACE  : '}';
LPAREN  : '(';
RPAREN  : ')';
LBRACK  : '[';
RBRACK  : ']';
SEMI    : ';';
COMMA   : ',';
```



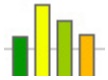
```
ASSIGN  : '=';
PLASGN  : "+=";
MINASGN : "-=";
MULASGN : "*=";
DIVASGN : "/=";
MODASGN : "%=";
OR      : "||";
AND     : "&&";
EQ      : "==";
NEQ     : "!=";
LT      : '<';
GT      : '>';
LTEQ    : "<=";
GTEQ    : ">=";
PLUS    : '+';
MINUS   : '-';
MULT    : '*';
DIV     : '/';
MOD     : '%';
INC     : "++";
DEC     : "--";
NOT     : '!';

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

protected
LETTER  : 'a'..'z' | 'A'..'Z'
        ;

protected
EXP     : ('e' | 'E') ('+' | '-')? (DIGIT)+
        ;

INT_FLOAT
        : (DIGIT)+                      { $setType(INT); }
          (('.' (DIGIT)* (EXP)? | EXP) { $setType(FLOAT); })?
        | '.' (DIGIT)+ (EXP)?           { $setType(FLOAT); }
        ;

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

protected
ESC     : '\\' ('n' | 't' | '"' | '\\')
        ;

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

NEWLINE : ( ("\r\n") => "\r\n" /* DOS */
          | '\r'               /* MAC */
          | '\n'               /* UNIX */
          )
          { $setType(Token.SKIP); newline(); }
```



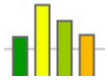
```
        ;

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

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



class EzgAntlrParser extends Parser;

options {
    k = 1;
    buildAST = true;
    exportVocab = EzgAntlrVocab;
}

tokens {
    PROGRAM;
    FUNC_DEF;
    ARG_LIST;

    STATEMENT;
    FOR_COND;
    EXPR_LIST;
    BLOCK;

    DECLARATION;
    TYPE;

    UPLUS;
    UMINUS;
    INDEX;
    PREF_INC;
    PREF_DEC;
    POST_INC;
    POST_DEC;
    FUNC_CALL;
}

{
    public void reportError(String s) {
        throw new RuntimeException(s);
    }

    public void reportError(RecognitionException e) {
```



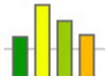
```
        throw new RuntimeException(e.toString());
    }
}


/* External definitions. */

program
        : (extern_def)* EOF!
          { #program = #([PROGRAM, "program"], program); }
        ;

extern_def
        : include_decl
        | (func_prefix) => func_def
        | declaration SEMI!
        ;

include_decl
        : "include"^ STRING SEMI!
        ;

func_def
        : func_prefix func_body
          { #func_def = #([FUNC_DEF, "func_def"], func_def); }
        ;

protected
func_prefix
        : ("void" | type) ID LPAREN! arg_list RPAREN!
        ;

arg_list
        : ( type ID (COMMA! type ID)*
          | /* Nothing. Needed to add a node for an empty arg_list. */
          )
          { #arg_list = #([ARG_LIST, "arg_list"], arg_list); }
        ;

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


/* Statements. */

statement
        : SEMI
        | declaration SEMI!
        | if_stmt
        | while_stmt
        | do_stmt
        | for_stmt
        | break_stmt
        | continue_stmt
        | return_stmt
```



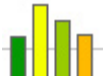
```
        | expression SEMI!
        | LBRACE! (statement)* RBRACE!
          { #statement = #([BLOCK, "block"], statement); }
        ;

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

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

do_stmt
        : "do"^ statement "while"! LPAREN! expression RPAREN! SEMI!
        ;

for_stmt
        : "for"^ LPAREN! for_clause RPAREN! statement
        ;

for_clause
        : (declaration | expr_list) SEMI! for_cond SEMI! expr_list
        ;

for_cond
        : ( expression
          | /* Nothing. Needed to add a node for an empty for_cond. */
          )
          { #for_cond = #([FOR_COND, "for_cond"], for_cond); }
        ;

expr_list
        : ( expression (COMMA! expression)*
          | /* Nothing. Needed to add a node for an empty expr_list. */
          )
          { #expr_list = #([EXPR_LIST, "expr_list"], expr_list); }
        ;

break_stmt
        : "break"^ SEMI!
        ;

continue_stmt
        : "continue"^ SEMI!
        ;

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


/* Declarations. */
```



```
declaration
        : type decl_list
          { #declaration = #([DECLARATION, "declaration"], declaration); }
        ;

type
        : type_spec (LBRACK RBRACK!)*
          { #type = #([TYPE, "type"], type); }
        ;

type_spec
        : "bool" | "int" | "float" | "string"
        ;

decl_list
        : ID var_init (COMMA! ID var_init)*
        ;

var_init
        : (ASSIGN^ expression)?
        ;


/* Expressions. */

expression
        : assignment
        ;

assignment
        : logic_expr
          ( (ASSIGN^ | PLASGN^ | MINASGN^ | MULASGN^ | DIVASGN^ | MODASGN^)
            assignment
          )?
        ;

logic_expr
        : logic_term (OR^ logic_term)*
        ;

logic_term
        : equal_expr (AND^ equal_expr)*
        ;

equal_expr
        : relat_expr ((EQ^ | NEQ^) relat_expr)?
        ;

relat_expr
        : arith_expr ((LT^ | GT^ | LTEQ^ | GTEQ^) arith_expr)?
        ;

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

arith_term
```

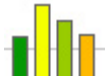


```
        : arith_factor ((MULT^ | DIV^ | MOD^) arith_factor)*
        ;

arith_factor
        : inc:INC^ logic_factor { #inc.setType(PREF_INC); }
        | dec:DEC^ logic_factor { #dec.setType(PREF_DEC); }
        | p:PLUS^  logic_factor { #p.setType(UPLUS); }
        | m:MINUS^ logic_factor { #m.setType(UMINUS); }
        | logic_factor
        ;

logic_factor
        : new_expr
        | (NOT^)? postfix_expr
        ;

new_expr
        : "new"^ base_type index_list
        ;

base_type
        : type_spec
          { #base_type = #([TYPE, "base_type"], base_type); }
        ;

index_list
        : (LBRACK! expression RBRACK!)+
          { #index_list = #([EXPR_LIST, "index_list"], index_list); }
        ;

postfix_expr
        : primary_expr
          ( index_list
            { #postfix_expr = #([INDEX, "index"], postfix_expr); }
          )?
          ( inc:INC^ { #inc.setType(POST_INC); }
          | dec:DEC^ { #dec.setType(POST_DEC); }
          )?
        ;

primary_expr
        : ID
          ( LPAREN! expr_list RPAREN!
            { #primary_expr = #([FUNC_CALL, "func_call"], primary_expr); }
          )?
        | constant
        | LPAREN! expression RPAREN!
        ;

constant
        : INT
        | FLOAT
        | STRING
        | "true"
        | "false"
        ;
```

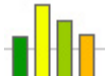


B.1.2 src/ezg/walker.g

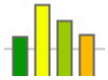
```
/*
 * Title:       walker.g
 * Description: A parser for the Abstract Syntax Tree generated by the
parser.
 * Modified:    2007-05-02
 */

header {
    package ezg;
}

{
    import java.util.Vector;
    import ezg.type.*;
}

class EzgAntlrWalker extends TreeParser;

options {
    importVocab = EzgAntlrVocab;
}

{
    private EzgInterpreter intpr = new EzgInterpreter();
    private int line;
    private int column;
    private String filename;

    protected void match(AST t, int ttype) throws MismatchedTokenException {
        super.match(t, ttype);
        line = t.getLine();
        column = t.getColumn();
        if (null != ((EzgAST)t).getFilename())
            filename = ((EzgAST)t).getFilename();
    }

    public int getLine() {
        return line;
    }

    public int getColumn() {
        return column;
    }

    public String getFilename() {
        return filename;
    }
}

extern_def
{
    EzgType a, rt;
    EzgType[] al;
```



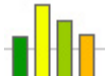
```
}
        : #("include" a=expression) { intpr.includeFile(this, a); }
        | #(FUNC_DEF rt=type fname:ID al=arg_list fbody:.)
          { intpr.registerFunction(fname.getText(), rt, al, #fbody); }
        | decl:DECLARATION          { declaration(#decl); }
        | #(PROGRAM (extd:.         { extern_def(#extd); })*)
        ;

main_func
        : . { intpr.invokeFunction(this, "main", new EzgType[0]); }
        ;

arg_list returns [ EzgType[] al ]
{
    Vector v;
    EzgType t;
    al = null;
}
        : #(ARG_LIST        { v = new Vector(); }
            (t=type arg:ID  { t.setName(arg.getText()); v.add(t); })*
          )                 { al = EzgInterpreter.convertList(v); }
        ;

statement returns [ EzgType r ]
{
    EzgType a;
    r = null;
}
        : SEMI /* Skip. */
        | decl:DECLARATION { declaration(#decl); }
        | #("if" a=expression thenp:. (elsep:.)?)
          {
              if (!(a instanceof EzgBool))
                  throw new RuntimeException("if condition must evaluate to "
+
                                             "bool");
              EzgInterpreter.mustBeStmt(#thenp);
              if (null != elsep)
                  EzgInterpreter.mustBeStmt(#elsep);
              if (((EzgBool)a).getVar())
                  r = statement(#thenp);
              else if (null != elsep)
                  r = statement(#elsep);
          }
        | #("while" wcond:. wbody:.)
          {
              EzgInterpreter.mustBeStmt(#wbody);
              while (intpr.loopCanProceed(this, #wcond)) {
                  r = statement(#wbody);
                  intpr.handleContinue();
              }
              intpr.handleBreak();
          }
        | #("do" dbody:. dcond:.)
          {
              EzgInterpreter.mustBeStmt(#dbody);
              do {
```



```
                r = statement(#dbody);
                intpr.handleContinue();
            } while (intpr.loopCanProceed(this, #dcond));
            intpr.handleBreak();
        }
      | #("for" finit:. fcond:. fiter:. fbody:.)
        { intpr.executeForLoop(this, #finit, #fcond, #fiter, #fbody); }
      | "break"           { intpr.setBreak(); }
      | "continue"        { intpr.setContinue(); }
      | #("return"
          (a=expression   { r = EzgInterpreter.rValue(a); })?
        )                 { intpr.setReturn(); }
      | a=expr:expression { EzgInterpreter.mustBeExpr(#expr); }
      | #(EXPR_LIST
          ( lexpr:.
          {
              EzgInterpreter.mustBeExpr(#lexpr);
              expression(#lexpr);
          }
          )*
        )
      | #(STATEMENT
          ( stmt:.
          {
              EzgInterpreter.mustBeStmtOrDecl(#stmt);
              if (intpr.canProceed())
                  r = statement(#stmt);
          }
          )*
        )
      | #(block:BLOCK
        {
            #block.setType(STATEMENT);
            if (intpr.canProceed())
                r = intpr.executeBlock(this, #block);
            #block.setType(BLOCK);
        }
        )
      ;

declaration
{
    EzgType t;
    EzgType a = null;
}
      : #(DECLARATION t=type
          ( vname:ID (#(ASSIGN a=expression))?
          { intpr.registerVariable(vname.getText(), t, a); a = null; }
          )*
        )
      ;

type returns [ EzgType t ]
{
    t = null;
}
      : "void"       { t = new EzgVoid(); }
```



```
        | #(TYPE
            ( "bool"   { t = new EzgBool(); }
            | "int"    { t = new EzgInt(); }
            | "float"  { t = new EzgFloat(); }
            | "string" { t = new EzgString(); }
            )
            (LBRACK    { t = new EzgArray(t); })*
          )
        ;

expr_vec returns [ Vector ev ]
{
    EzgType a;
    ev = null;
}
        : #(EXPR_LIST     { ev = new Vector(); }
            (a=expression { ev.add(a); })*
          )
        ;

expr_list returns [ EzgType[] el ]
{
    Vector ev;
    el = null;
}
        : #(elist:EXPR_LIST
          {
              ev = expr_vec(#elist);
              el = EzgInterpreter.convertList(ev);
          }
          )
        ;

int_list returns [ int[] il ]
{
    Vector ev;
    il = null;
}
        : #(elist:EXPR_LIST
          {
              ev = expr_vec(#elist);
              il = EzgInterpreter.convertIntList(ev);
          }
          )
        ;

expression returns [ EzgType r ]
{
    EzgType a, b, bt;
    EzgType[] el;
    int[] il;
    r = null;
}
        : #(ASSIGN    a=expression b=expression) { r = a.assign(b); }
        | #(PLASGN    a=expression b=expression) { r = a.plasgn(b); }
        | #(MINASGN   a=expression b=expression) { r = a.minasgn(b); }
        | #(MULASGN   a=expression b=expression) { r = a.mulasgn(b); }
```

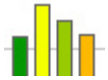


```
        | #(DIVASGN   a=expression b=expression) { r = a.divasgn(b); }
        | #(MODASGN   a=expression b=expression) { r = a.modasgn(b); }
        | #(OR        a=expression b=expression) { r = a.or(b); }
        | #(AND       a=expression b=expression) { r = a.and(b); }
        | #(EQ        a=expression b=expression) { r = a.eq(b); }
        | #(NEQ       a=expression b=expression) { r = a.neq(b); }
        | #(LT        a=expression b=expression) { r = a.lt(b); }
        | #(GT        a=expression b=expression) { r = a.gt(b); }
        | #(LTEQ      a=expression b=expression) { r = a.lteq(b); }
        | #(GTEQ      a=expression b=expression) { r = a.gteq(b); }
        | #(PLUS      a=expression b=expression) { r = a.plus(b); }
        | #(MINUS     a=expression b=expression) { r = a.minus(b); }
        | #(MULT      a=expression b=expression) { r = a.mult(b); }
        | #(DIV       a=expression b=expression) { r = a.div(b); }
        | #(MOD       a=expression b=expression) { r = a.mod(b); }
        | #(PREF_INC  a=expression)              { r = a.prefinc(); }
        | #(PREF_DEC  a=expression)              { r = a.prefdec(); }
        | #(UPLUS     a=expression)              { r = a.uplus(); }
        | #(UMINUS    a=expression)              { r = a.uminus(); }
        | #(NOT       a=expression)              { r = a.not(); }
        | #(INDEX     a=expression il=int_list)  { r = a.index(il); }
        | #(POST_INC  a=expression)              { r = a.postinc(); }
        | #(POST_DEC  a=expression)              { r = a.postdec(); }
        | #(FUNC_CALL fname:ID el=expr_list)
          { r = intpr.invokeFunction(this, fname.getText(),el); }
        | #("new" bt=type il=int_list)
          {
              for (int j = 1; j < il.length; j++)
                  bt = new EzgArray(bt);
              r = new EzgArray(bt, il);
          }
        | id:ID    { r = intpr.getVariable(id.getText()); }
        | i:INT    { r = new EzgInt(Integer.parseInt(i.getText())); }
        | f:FLOAT  { r = new EzgFloat(Float.parseFloat(f.getText())); }
        | s:STRING { r = new EzgString(s.getText()); }
        | "true"   { r = new EzgBool(true); }
        | "false"  { r = new EzgBool(false); }
        ;
```

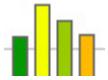
B.2 Interpreter

B.2.1 src/ezg/EzgMain.java

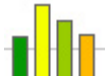
```
/**
 * Title:       EzgMain.java
 * Description: Main class.
 * Modified:    2007-05-05
 */

package ezg;

import java.io.File;
import java.io.FileReader;
import antlr.debug.misc.ASTFrame;
```



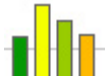
```
public class EzgMain {
    /** Operating system name. */
    private static final String OS =
        System.getProperty("os.name").toLowerCase();

    /**
     * The system-dependent default name-separator character, represented
     * as a string for convenience.
     */
    private static final String FS = System.getProperty("file.separator");

    public  static int     tabSize;
    public  static boolean verbose;
    public  static boolean test;
    public  static boolean debug;
    private static boolean help;
    private static boolean version;
    private static String  filename;

    public static boolean parserError;

    private EzgMain() {
        /* This class cannot be instantiated. */
    }

    public static void main(String[] args) {
        EzgAntlrLexer lexer;
        EzgAntlrParser parser;
        EzgAST tree;
        EzgAntlrWalker walker = null;

        try {
            parseArguments(args);
            if (help) {
                printHelp();
                System.exit(0);
            }
            if (version) {
                printVersion();
                System.exit(0);
            }

            parserError = false;
            lexer = new EzgAntlrLexer(new FileReader(filename));
            lexer.setFilename(filename);
            lexer.setTabSize(tabSize);
            parser = new EzgAntlrParser(lexer);
            parser.setFilename(filename);
            parser.setASTNodeClass(EzgAST.class.getName());
            parser.program();
            tree = (EzgAST)parser.getAST();
            setTreeInfo(tree, filename);
            if (debug) {
                System.out.println("Tree:" + tree.toStringList());
                System.out.println();
                ASTFrame frame = new ASTFrame("Tree", tree);
```



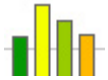
```
            frame.setVisible(true);
        }
        walker = new EzgAntlrWalker();
        walker.extern_def(tree);
        walker.main_func(tree);
    } catch (Exception e) {
        String message = "Error: ";
        if (!parserError && null != walker) {
            message += walker.getFilename() + ":" +
                       walker.getLine() + ":" +
                       walker.getColumn() + ": ";
            message += e.getMessage();
        } else {
            message += errorMessage(e);
        }
        System.out.flush();
        System.err.println(message);
        if (debug)
            e.printStackTrace();
    }
}

private static void parseArguments(String[] args) {
    String arg;

    tabSize = 4;
    verbose = test = debug = help = version = false;
    filename = null;

    for (int i = 0; i < args.length; i++) {
        arg = args[i];
        if (arg.equals("--help")) {
            help = true;
            return;
        }
        if (arg.equals("--version")) {
            version = true;
            return;
        }
        if (arg.startsWith("--tab-size")) {
            try {
                String size = arg.substring(arg.indexOf("=") + 1);
                tabSize = Integer.parseInt(size);
            } catch (Exception e) {
                throw new IllegalArgumentException("illegal argument: " +
                                                   arg);
            }
        } else if (arg.equals("--verbose")) {
            verbose = true;
        } else if (arg.equals("--test")) {
            test = true;
        } else if (arg.equals("--debug")) {
            debug = true;
        } else if (arg.matches("-[vtd]+")) {
            if (arg.indexOf('v') != -1)
                verbose = true;
            if (arg.indexOf('t') != -1)
```



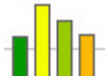
```
                test = true;
            if (arg.indexOf('d') != -1)
                debug = true;
        } else {
            if (null != filename || !arg.endsWith(".ezg"))
                throw new IllegalArgumentException("illegal argument: " +
                                                   arg);
            filename = arg;
        }
        if (test)
            debug = false;
        if (test || debug)
            verbose = true;
    }
}

/**
 * Sets filename for each AST node and fixes line and column values for
 * some nodes that were "artificially" added during parsing (nodes
 * without corresponding tokens).
 */
public static void setTreeInfo(EzgAST tree, String filename) {
    if (null == tree)
        return;

    /* Set filename. */
    tree.setFilename(filename);

    /* Fix line and column values. */
    EzgAST t;
    int l, c;
    switch (tree.getType()) {
    /* program. */
    case EzgAntlrWalker.PROGRAM:
        tree.setLine(1);
        tree.setColumn(1);

        break;
    case EzgAntlrWalker.FUNC_DEF:
        /* func_def. */
        t = (EzgAST)tree.getFirstChild().getNextSibling(); /* name. */
        tree.setLine(l = t.getLine());
        tree.setColumn(c = t.getColumn());

        /* arg_list. */
        t = (EzgAST)t.getNextSibling();
        t.setLine(l);
        t.setColumn(c);

        /* func_body. */
        t = (EzgAST)t.getNextSibling();
        t.setLine(l);
        t.setColumn(c);

        break;
    case EzgAntlrWalker.LITERAL_for:
        /* expr_list in for loop initializer. */
```



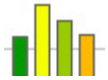
```
            t = (EzgAST)tree.getFirstChild();
            l = tree.getLine();
            c = tree.getColumn();
            if (t.getType() == EzgAntlrWalker.EXPR_LIST) {
                t.setLine(l);
                t.setColumn(c);
            }

            /* for_cond. */
            t = (EzgAST)t.getNextSibling();
            t.setLine(l);
            t.setColumn(c);

            /* expr_list in for loop iterator. */
            t = (EzgAST)t.getNextSibling();
            t.setLine(l);
            t.setColumn(c);

            break;
        case EzgAntlrWalker.DECLARATION:
            /* declaration. */
            t = (EzgAST)tree.getFirstChild().getFirstChild(); /* type_spec.
*/
            tree.setLine(t.getLine());
            tree.setColumn(t.getColumn());

            break;
        case EzgAntlrWalker.TYPE:
            /* type or base_type. */
            t = (EzgAST)tree.getFirstChild(); /* type_spec. */
            tree.setLine(t.getLine());
            tree.setColumn(t.getColumn());

            break;
        case EzgAntlrWalker.LITERAL_new:
            /* index_list. */
            t = (EzgAST)tree.getFirstChild().getNextSibling();
            t.setLine(tree.getLine());
            t.setColumn(tree.getColumn());

            break;
        case EzgAntlrWalker.INDEX:
        case EzgAntlrWalker.FUNC_CALL:
            /* index or func_call. */
            t = (EzgAST)tree.getFirstChild(); /* ID (var or func name). */
            tree.setLine(l = t.getLine());
            tree.setColumn(c = t.getColumn());

            /* index_list or expr_list. */
            t = (EzgAST)t.getNextSibling();
            t.setLine(l);
            t.setColumn(c);

            break;
        }

        /* Recursively walk the tree. */
```



```
        setTreeInfo((EzgAST)tree.getFirstChild(), filename);
        setTreeInfo((EzgAST)tree.getNextSibling(), filename);
    }

    public static void printTree(EzgAST tree) {
        if (null == tree)
            return;
        System.out.print(tree.getText() + "\t");
        if (tree.getText().length() < 8)
            System.out.print("\t");
        System.out.println(tree.getLine() + ":" + tree.getColumn());
        printTree((EzgAST)tree.getFirstChild());
        printTree((EzgAST)tree.getNextSibling());
    }

    public static String errorMessage(Exception e) {
        String regex = ".+\\.ezg:\\d+:\\d+:.+";
        if (e.getMessage().matches(regex))
            return e.getMessage();
        if (e.toString().matches(regex))
            return e.toString();
        return e.getMessage();
    }

    /**
     * Validates a path by analyzing each directory and file name it contains
     * based on the current platform (Windows or Unix/Linux).
     */
    public static void validatePath(String path) {
        String[] tokens;
        if (OS.indexOf("windows") != -1) { /* DOS filename conventions. */
            if (path.startsWith("\\\\"))
                path = path.substring(2);
            else if (path.length() >= 3 &&
                     path.substring(0,3).matches("[A-Za-z]:\\\\"))
                path = path.substring(3);
            tokens = path.split(FS + FS);
        } else { /* Unix/Linux filename conventions. */
            if (path.startsWith("/"))
                path = path.substring(1);
            tokens = path.split(FS);
        }
        for (int i = 0; i < tokens.length; i++)
            if (!validFilename(tokens[i]))
                throw new RuntimeException("\"" + path + "\" " +
                                           "(invalid filename)");
    }

    /** Validates a file or directory name. */
    private static boolean validFilename(String filename) {
        String regex;
        if (OS.indexOf("windows") != -1)       /* DOS filename conventions.
*/
            regex = "[^\\\\/:\\*\\?\"<>\\|]*"; /* Invalid: \/:*?"<>|
*/
        else                               /* Unix/Linux filename conventions.
*/
```



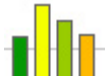
```
            regex = "(\\w|\\.|\\-|_)*"; /* Invalid: all except A-Za-z0-9_.-
*/
        if (!filename.matches(regex))
            return false;
        return true;
    }

    public static void createPath(File file) throws Exception  {
        /* Create path to file if it does not exist. */
        File path = file.getParentFile();
        if (path != null && !path.exists())
            path.mkdirs();
    }

    public static void createPath(String filename) throws Exception {
        File file = new File(filename);
        createPath(file);
    }

    private static void printHelp() {
        System.out.println("NAME");
        System.out.println("        " + EzgMain.class.getName() +
                           " - interpret source file");
        System.out.println();
        System.out.println("SYNOPSIS");
        System.out.println("        java " + EzgMain.class.getName() +
                           " [OPTION]... FILE");
        System.out.println();
        System.out.println("DESCRIPTION");
        System.out.println("        This application interprets the given " +
                           "FILE.");
        System.out.println();
        System.out.println("OPTIONS");
        System.out.println("        --tab-size=SIZE");
        System.out.println("                set one tab character to equal "
+
                           "SIZE space characters (default");
        System.out.println("                size is 4)");
        System.out.println();
        System.out.println("        -v, --verbose");
        System.out.println("                explain what is being done");
        System.out.println();
        System.out.println("        -t, --test");
        System.out.println("                explain what is being done " +
                           "without drawing anything");
        System.out.println();
        System.out.println("        -d, --debug");
        System.out.println("                explain what is being done and "
+
                           "display debugging information");
        System.out.println();
        System.out.println("        --help  display this help and exit");
        System.out.println();
        System.out.println("        --version");
        System.out.println();
        System.out.println("FILE");
```



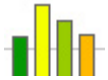
```
        System.out.println("        Specifies a particular file you want to "
+
                           "interpret.");
    }

    private static void printVersion() {
        System.out.println(EzgMain.class.getName() + " 1.0");
        System.out.println("Written by Vincent Dobrev and Edlira Kumbarce.");
    }
}
```

B.2.2 src/ezg/EzgInterpreter.java

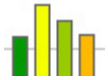
```
/**
 * Title:       EzgInterpreter.java
 * Description: Interacts directly with the tree walker. Keeps track of
symbol
 *              tables and activation records.
 * Modified:    2007-05-07
 */

package ezg;

import java.io.FileReader;
import java.util.Hashtable;
import java.util.Vector;
import antlr.RecognitionException;
import antlr.collections.AST;
import ezg.type.EzgBool;
import ezg.type.EzgInt;
import ezg.type.EzgFloat;
import ezg.type.EzgString;
import ezg.type.EzgType;

public class EzgInterpreter {
    private static final int CTRL_TO_NONE  = 0;
    private static final int CTRL_TO_BREAK = 1;
    private static final int CTRL_TO_CONT  = 2;
    private static final int CTRL_TO_RET   = 3;

    public static String pref;

    private EzgSymbolTable symTbl; /* Variables in different scopes. */
    private Hashtable funcs;       /* Functions. */
    private EzgPainter painter;
    private int ctrl;

    EzgInterpreter() {
        symTbl  = new EzgSymbolTable(null, null);
        funcs   = new Hashtable();
        painter = new EzgPainter();
        ctrl    = CTRL_TO_NONE;
        if (EzgMain.debug)
            pref = "";
        EzgInternalFunctions.register(funcs);
```



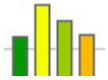
```
        registerVariable("E", new EzgFloat(), new EzgFloat((float)Math.E));
        registerVariable("PI", new EzgFloat(), new EzgFloat((float)Math.PI));
    }

    public static EzgType[] convertList(Vector v) {
        EzgType[] list = new EzgType[v.size()];
        for (int i = 0; i < list.length; i++)
            list[i] = (EzgType)v.remove(0);
        return list;
    }

    public static int[] convertIntList(Vector v) {
        int[] list = new int[v.size()];
        for (int i = 0; i < list.length; i++)
            list[i] = EzgInt.intValue((EzgType)v.remove(0));
        return list;
    }

    public static void mustBeExpr(AST node) {
        switch (node.getType()) {
        case EzgAntlrWalker.ASSIGN:
        case EzgAntlrWalker.MULASGN:
        case EzgAntlrWalker.DIVASGN:
        case EzgAntlrWalker.PLASGN:
        case EzgAntlrWalker.MINASGN:
        case EzgAntlrWalker.PREF_INC:
        case EzgAntlrWalker.PREF_DEC:
        case EzgAntlrWalker.POST_INC:
        case EzgAntlrWalker.POST_DEC:
        case EzgAntlrWalker.FUNC_CALL:
            return;
        }
        throw new RuntimeException("not a statement");
    }

    public static void mustBeStmt(AST node) {
        switch (node.getType()) {
        case EzgAntlrWalker.SEMI:
        case EzgAntlrWalker.LITERAL_if:
        case EzgAntlrWalker.LITERAL_while:
        case EzgAntlrWalker.LITERAL_do:
        case EzgAntlrWalker.LITERAL_for:
        case EzgAntlrWalker.LITERAL_break:
        case EzgAntlrWalker.LITERAL_continue:
        case EzgAntlrWalker.LITERAL_return:
        case EzgAntlrWalker.STATEMENT:
        case EzgAntlrWalker.BLOCK:
            return;
        }
        mustBeExpr(node);
    }

    public static void mustBeStmtOrDecl(AST node) {
        if (node.getType() == EzgAntlrWalker.DECLARATION)
            return;
        mustBeStmt(node);
    }
```



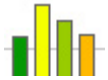
```
    public static EzgType rValue(EzgType a) {
        if (null == a.getName())
            return a;
        return a.copy();
    }

    public void includeFile(EzgAntlrWalker walker, EzgType a)
            throws RecognitionException {
        if (!(a instanceof EzgString))
            throw new RuntimeException("include parameter must be string");

        String filename = a.toString();
        if (!filename.endsWith(".ezg"))
            throw new RuntimeException(filename + " (include parameter not "
+
                                       "EZGraphs source file)");

        EzgAntlrLexer lexer;
        EzgAntlrParser parser;
        try {
            lexer = new EzgAntlrLexer(new FileReader(filename));
            lexer.setFilename(filename);
            lexer.setTabSize(EzgMain.tabSize);
            parser = new EzgAntlrParser(lexer);
            parser.setFilename(filename);
            parser.setASTNodeClass(EzgAST.class.getName());
            parser.program();
        } catch (Exception e) {
            EzgMain.parserError = true;
            throw new RuntimeException(EzgMain.errorMessage(e));
        }

        EzgAST tree = (EzgAST)parser.getAST();
        EzgMain.setTreeInfo(tree, filename);
        walker.extern_def(tree);
    }

    public void registerFunction(String name, EzgType rtype, EzgType[] args,
                                 AST body) {
        if (null != funcs.get(name + "(...)"))
            throw new RuntimeException("duplicate function: " + name +
                                       "(...)");
        EzgFunction func = new EzgFunction(name, rtype, args, body, symTbl);
        if (null != funcs.get(func.getSignature()))
            throw new RuntimeException("duplicate function: " +
                                       func.getSignature());
        funcs.put(func.getSignature(), func);
        if (EzgMain.debug)
            System.out.println(pref + "reg func: " + func.toString());
    }

    public void registerVariable(String name, EzgType type, EzgType val) {
        if (null != symTbl.get(name))
            throw new RuntimeException("duplicate variable: " + name);
        EzgType var = type.type();
        var.setName(name);
```



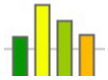
```
        if (null != val)
            var.assign(val);
        symTbl.put(name, var);
        if (EzgMain.debug)
            System.out.println(pref + "reg var: " + var.what());
    }

    private EzgFunction getFunction(String name, String[] typeNames,
                                    int start) {
        EzgFunction func;
        for (int i = start; i < typeNames.length; i++) {
            if (typeNames[i].equals("int")) {
                String[] tn = (String[])typeNames.clone();
                tn[i] = "float";
                String sig = EzgFunction.getSignature(name, tn);
                func = (EzgFunction)funcs.get(sig);
                if (null != func)
                    return func;
                func = getFunction(name, tn, i + 1);
                if (null != func)
                    return func;
            }
        }
        return null;
    }

    public EzgType getVariable(String name) {
        EzgType var;
        for (EzgSymbolTable st = symTbl; null != st; st =
st.getStaticParent())
            if (null != (var = (EzgType)st.get(name)))
                return var;
        throw new RuntimeException("variable " + name + " not declared");
    }

    private void matchResult(EzgType result, EzgFunction func) {
        String error = "function " + func.getSignature() +
                       " expected to return " +
                       func.getReturnType().typeName();
        if (null == result) {
            if (func.returnsVoid())
                return;
            throw new RuntimeException(error);
        }
        if (result.typeName().equals(func.getReturnType().typeName()))
            return;
        throw new RuntimeException(error + ", not " + result.typeName());
    }

    public EzgType invokeFunction(EzgAntlrWalker walker, String name,
                                  EzgType[] params)
            throws RecognitionException {
        EzgFunction func;

        /*
         * Check for internal function with variable nr of args.
         * If found, call it and return.
```



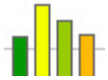
```
         */
        func = (EzgFunction)funcs.get(name + "(...)");
        if (null != func)
            return EzgInternalFunctions.call(painter, func, params);

        /* Try getting the called function using signatures. */
        String funcSig = EzgFunction.getSignature(name, params);
        func = (EzgFunction)funcs.get(funcSig);

        /*
         * Try getting the called function using all possible signatures
         * derived after promoting ints to floats.
         */
        if (null == func) {
            String[] typeNames = new String[params.length];
            for (int i = 0; i < typeNames.length; i++)
                typeNames[i] = params[i].typeName();
            func = getFunction(name, typeNames, 0);
        }

        /* Function not found. */
        if (null == func)
            throw new RuntimeException("function " + funcSig + " not
defined");

        /*
         * Check for internal function with fixed nr of args.
         * If found, call it and return.
         */
        if (func.isInternal())
            return EzgInternalFunctions.call(painter, func, params);

        /* Enter new scope. */
        symTbl = new EzgSymbolTable(func.getStaticParent(), symTbl);
        if (EzgMain.debug) {
            System.out.println(pref + "func " + funcSig + " {");
            pref += "  ";
        }

        /* Match and assign parameters to arguments. */
        EzgType[] args = func.getArguments();
        String argName;
        for (int i = 0; i < params.length; i++){
            argName = args[i].getName();
            args[i].assign(params[i]);
            args[i].setName(argName);
            symTbl.put(argName, args[i]);
        }

        /* Execute the function body. */
        EzgType result = walker.statement(func.getBody());

        /* Handle break. */
        if (CTRL_TO_BREAK == ctrl)
            throw new RuntimeException("unexpected break");

        /* Handle continue. */
```



```
        if (CTRL_TO_CONT == ctrl)
            throw new RuntimeException("unexpected continue");

        /* Handle return. */
        handleReturn();

        /* Match result with return type. */
        if (result instanceof EzgInt &&
            func.getReturnType() instanceof EzgFloat) {
            result = new EzgFloat(((EzgInt)result).getVar());
        } else {
            matchResult(result, func);
        }

        /* Exit scope. */
        symTbl = symTbl.getDynamicParent();
        if (EzgMain.debug) {
            pref = pref.substring(0, pref.length() - 2);
            System.out.println(pref + "}");
        }

        return result;
    }

    public EzgType executeBlock(EzgAntlrWalker walker, AST block)
            throws RecognitionException {
        /* Enter new scope. */
        symTbl = new EzgSymbolTable(symTbl, symTbl);
        if (EzgMain.debug) {
            System.out.println(pref + "block {");
            pref += "  ";
        }

        /* Execute the block. */
        EzgType result = walker.statement(block);

        /* Exit scope. */
        symTbl = symTbl.getDynamicParent();
        if (EzgMain.debug) {
            pref = pref.substring(0, pref.length() - 2);
            System.out.println(pref + "}");
        }

        return result;
    }

    public boolean canProceed() {
        return CTRL_TO_NONE == ctrl;
    }

    public boolean loopCanProceed(EzgAntlrWalker walker, AST condition)
            throws RecognitionException {
        if (CTRL_TO_NONE != ctrl)
            return false;
        if (condition.getType() == EzgAntlrWalker.FOR_COND) {
            condition = condition.getFirstChild();
            if (null == condition)
```



```
            return true;
        }
        EzgType result = walker.expression(condition);
        if (!(result instanceof EzgBool))
            throw new RuntimeException("loop condition must evaluate to a " +
                                       "bool");
        return ((EzgBool)result).getVar();
    }

    public void executeForLoop(EzgAntlrWalker walker, AST initializer,
                               AST condition, AST iterator, AST body)
            throws RecognitionException {
        /* Enter new scope. */
        symTbl = new EzgSymbolTable(symTbl, symTbl);
        if (EzgMain.debug) {
            System.out.println(pref + "for {");
            pref += "  ";
        }

        /* Execute loop. */
        mustBeStmt(body);
        walker.statement(initializer);
        while (loopCanProceed(walker, condition)) {
            walker.statement(body);
            handleContinue();
            walker.statement(iterator);
        }
        handleBreak();

        /* Exit scope. */
        symTbl = symTbl.getDynamicParent();
        if (EzgMain.debug) {
            pref = pref.substring(0, pref.length()-2);
            System.out.println(pref + "}");
        }
    }

    public void setBreak() {
        ctrl = CTRL_TO_BREAK;
    }

    public void setContinue() {
        ctrl = CTRL_TO_CONT;
    }

    public void setReturn() {
        ctrl = CTRL_TO_RET;
    }

    public void handleBreak() {
        if (CTRL_TO_BREAK == ctrl)
            ctrl = CTRL_TO_NONE;
    }

    public void handleContinue() {
        if (CTRL_TO_CONT == ctrl)
            ctrl = CTRL_TO_NONE;
```



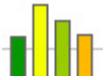
```
    }

    private void handleReturn() {
        if (CTRL_TO_RET == ctrl)
            ctrl = CTRL_TO_NONE;
    }
}
```

B.2.3 src/ezg/EzgSymbolTable.java

```
/**
 * Title:       EzgSymbolTable.java
 * Description: Keeps track of identifiers, their values, and the scope in
 *              which they're visible.
 * Modified:    2007-04-26
 */

package ezg;

import java.util.Hashtable;

public class EzgSymbolTable extends Hashtable {
    private static final long serialVersionUID = 1L;

    private EzgSymbolTable statPrnt;
    private EzgSymbolTable dynmcPrnt;

    EzgSymbolTable(EzgSymbolTable statPrnt, EzgSymbolTable dynmcPrnt) {
        this.statPrnt = statPrnt;
        this.dynmcPrnt = dynmcPrnt;
    }

    public EzgSymbolTable getStaticParent() {
        return statPrnt;
    }

    public EzgSymbolTable getDynamicParent() {
        return dynmcPrnt;
    }
}
```

B.2.4 src/ezg/EzgAST.java

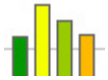
```
/**
 * Title:       EzgAST.java
 * Description: Common AST node with added line, column, and filename.
 * Modified:    2007-05-02
 */

package ezg;

import antlr.CommonAST;
import antlr.Token;
```



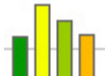
```
public class EzgAST extends CommonAST {
    private static final long serialVersionUID = 1L;

    private int line;
    private int column;
    private String filename;

    public void initialize(Token tok) {
        super.initialize(tok);
        line = tok.getLine();
        column = tok.getColumn();
    }

    public int getLine() {
        return line;
    }

    public int getColumn() {
        return column;
    }

    public String getFilename() {
        return filename;
    }

    public void setLine(int line) {
        this.line = line;
    }

    public void setColumn(int column) {
        this.column = column;
    }

    public void setFilename(String filename) {
        this.filename = filename;
    }
}
```

B.2.5 src/ezg/EzgFunction.java

```
/**
 * Title:       EzgFunction.java
 * Description: Class to represent functions.
 * Modified:    2007-05-07
 */

package ezg;

import antlr.collections.AST;
import ezg.type.EzgType;
import ezg.type.EzgVoid;

public class EzgFunction {
    private String name;
```



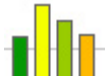
```
    private String funcSig;  /* main(), drawPoint(int,int), print(...). */
    private EzgType retType;
    private EzgType[] args;  /* null for variable number of arguments. */
    private AST body;        /* null for internal functions. */
    private EzgSymbolTable statPrnt;  /* Static parent. */
    private int id;          /* Only for internal functions. */

    EzgFunction(String name, EzgType retType, EzgType[] args, AST body,
                EzgSymbolTable statPrnt) {
        this.name = name;
        this.retType = retType;
        this.args = args;
        this.body = body;
        this.statPrnt = statPrnt;
        funcSig = getSignature(name, args);
    }

    EzgFunction(String name, EzgType[] args, int id) {
        this.name = name;
        this.args = args;
        this.id = id;
        funcSig = getSignature(name, args);
    }

    public static String getSignature(String name, EzgType[] args) {
        if (null == args)
            return name + "(...)";
        String funcSig = name + "(";
        for (int i = 0; i < args.length; i++) {
            if (!(args[i] instanceof EzgType))
                throw new RuntimeException("unknown argument type passed to "
+
                                           "function " + name + "()");
            funcSig += args[i].typeName();
            if (i < args.length - 1)
                funcSig += ",";
        }
        funcSig += ")";
        return funcSig;
    }

    public static String getSignature(String name, String[] typeNames) {
        if (null == typeNames)
            return name + "(...)";
        String funcSig = name + "(";
        for (int i = 0; i < typeNames.length; i++) {
            funcSig += typeNames[i];
            if (i < typeNames.length - 1)
                funcSig += ",";
        }
        funcSig += ")";
        return funcSig;
    }

    public String getName() {
        return name;
    }
```



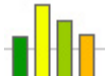
```
    public String getSignature() {
        return funcSig;
    }

    public EzgType getReturnType() {
        return retType;
    }

    public EzgType[] getArguments() {
        EzgType[] args = new EzgType[this.args.length];
        for (int i = 0; i < args.length; i++) {
            args[i] = this.args[i].type();
            args[i].setName(this.args[i].getName());
        }
        return args;
    }

    public AST getBody() {
        return body;
    }

    public EzgSymbolTable getStaticParent() {
        return statPrnt;
    }

    public int getId() {
        return id;
    }

    public boolean isInternal() {
        return null == body;
    }

    public boolean returnsVoid() {
        return retType instanceof EzgVoid;
    }

    public String toString() {
        String str = retType.typeName() + " " + name + "(";
        for (int i = 0; i < args.length; i++) {
            str += args[i].typeName() + " " + args[i].getName();
            if (i < args.length - 1)
                str += ", ";
        }
        str += ")";
        return str;
    }
}
```

B.2.6 src/ezg/EzgInternalFunctions.java

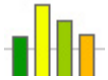
```
/**
 * Title:       EzgInternalFunctions.java
 * Description: Class to represent all the predefined functions.
```



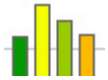
```
 * Modified:    2007-05-06
 */

package ezg;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;
import ezg.type.*;

public class EzgInternalFunctions {
    /** Printing. */
    private static final int FN_PRINT        = 0;
    private static final int FN_PRINTLN      = 1;

    /** Strings. */
    private static final int FN_STR_TO_INT   = 2;
    private static final int FN_STR_TO_FLOAT = 3;
    private static final int FN_INDEX        = 4;
    private static final int FN_LAST_INDEX   = 5;
    private static final int FN_SUBSTRING    = 6;

    /** String or array size. */
    private static final int FN_SIZE         = 7;

    /** Math. */
    private static final int FN_RANDOM       = 8;
    private static final int FN_ROUND        = 9;
    private static final int FN_FLOOR        = 10;
    private static final int FN_CEIL         = 11;
    private static final int FN_ABS_INT      = 12;
    private static final int FN_ABS_FLOAT    = 13;
    private static final int FN_SQRT         = 14;
    private static final int FN_POW          = 15;
    private static final int FN_EXP          = 16;
    private static final int FN_LOG          = 17;
    private static final int FN_SIN          = 18;
    private static final int FN_COS          = 19;
    private static final int FN_TAN          = 20;
    private static final int FN_ASIN         = 21;
    private static final int FN_ACOS         = 22;
    private static final int FN_ATAN         = 23;

    /** Data. */
    private static final int FN_DATA         = 24;

    /** Canvas. */
    private static final int FN_CANVAS       = 25;
    private static final int FN_SHOW         = 26;
    private static final int FN_SAVE         = 27;

    /** Environment. */
    private static final int FN_BACKGROUND   = 28;
    private static final int FN_COLOR        = 29;
```



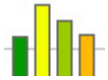
```
    private static final int FN_STROKE       = 30;
    private static final int FN_FONT         = 31;

    /** Transformations. */
    private static final int FN_SCALE        = 32;
    private static final int FN_SHEAR        = 33;
    private static final int FN_ROTATE       = 34;
    private static final int FN_TRANSLATE    = 35;
    private static final int FN_RESET        = 36;
    private static final int FN_PUSH         = 37;
    private static final int FN_POP          = 38;

    /** Drawing. */
    private static final int FN_POINT        = 39;
    private static final int FN_LINE         = 40;
    private static final int FN_RECT         = 41;
    private static final int FN_FILL_RECT    = 42;
    private static final int FN_POLYGON      = 43;
    private static final int FN_FILL_POLYGON = 44;
    private static final int FN_ARC          = 45;
    private static final int FN_FILL_ARC     = 46;
    private static final int FN_OVAL         = 47;
    private static final int FN_FILL_OVAL    = 48;
    private static final int FN_TEXT         = 49;
    private static final int FN_WIDTH        = 50;
    private static final int FN_HEIGHT       = 51;

    public static void register(Hashtable funcs) {
        registerFunction(funcs, "print",       FN_PRINT);
        registerFunction(funcs, "println",     FN_PRINTLN);
        registerFunction(funcs, "strToInt",    FN_STR_TO_INT);
        registerFunction(funcs, "strToFloat",  FN_STR_TO_FLOAT);
        registerFunction(funcs, "index",       FN_INDEX);
        registerFunction(funcs, "lastIndex",   FN_LAST_INDEX);
        registerFunction(funcs, "substring",   FN_SUBSTRING);
        registerFunction(funcs, "size",        FN_SIZE);
        registerFunction(funcs, "random",      FN_RANDOM);
        registerFunction(funcs, "round",       FN_ROUND);
        registerFunction(funcs, "floor",       FN_FLOOR);
        registerFunction(funcs, "ceil",        FN_CEIL);
        registerFunction(funcs, "abs",         FN_ABS_INT);
        registerFunction(funcs, "abs",         FN_ABS_FLOAT);
        registerFunction(funcs, "sqrt",        FN_SQRT);
        registerFunction(funcs, "pow",         FN_POW);
        registerFunction(funcs, "exp",         FN_EXP);
        registerFunction(funcs, "log",         FN_LOG);
        registerFunction(funcs, "sin",         FN_SIN);
        registerFunction(funcs, "cos",         FN_COS);
        registerFunction(funcs, "tan",         FN_TAN);
        registerFunction(funcs, "asin",        FN_ASIN);
        registerFunction(funcs, "acos",        FN_ACOS);
        registerFunction(funcs, "atan",        FN_ATAN);
        registerFunction(funcs, "data",        FN_DATA);
        registerFunction(funcs, "canvas",      FN_CANVAS);
        registerFunction(funcs, "show",        FN_SHOW);
        registerFunction(funcs, "save",        FN_SAVE);
        registerFunction(funcs, "background",  FN_BACKGROUND);
```



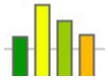
```
        registerFunction(funcs, "color",       FN_COLOR);
        registerFunction(funcs, "stroke",      FN_STROKE);
        registerFunction(funcs, "font",        FN_FONT);
        registerFunction(funcs, "scale",       FN_SCALE);
        registerFunction(funcs, "shear",       FN_SHEAR);
        registerFunction(funcs, "rotate",      FN_ROTATE);
        registerFunction(funcs, "translate",   FN_TRANSLATE);
        registerFunction(funcs, "reset",       FN_RESET);
        registerFunction(funcs, "push",        FN_PUSH);
        registerFunction(funcs, "pop",         FN_POP);
        registerFunction(funcs, "point",       FN_POINT);
        registerFunction(funcs, "line",        FN_LINE);
        registerFunction(funcs, "rect",        FN_RECT);
        registerFunction(funcs, "fillRect",    FN_FILL_RECT);
        registerFunction(funcs, "polygon",     FN_POLYGON);
        registerFunction(funcs, "fillPolygon", FN_FILL_POLYGON);
        registerFunction(funcs, "arc",         FN_ARC);
        registerFunction(funcs, "fillArc",     FN_FILL_ARC);
        registerFunction(funcs, "oval",        FN_OVAL);
        registerFunction(funcs, "fillOval",    FN_FILL_OVAL);
        registerFunction(funcs, "text",        FN_TEXT);
        registerFunction(funcs, "width",       FN_WIDTH);
        registerFunction(funcs, "height",      FN_HEIGHT);
    }

    private static void registerFunction(Hashtable funcs,
                                         String name, int id) {
        EzgFunction func = new EzgFunction(name, getArguments(id), id);
        funcs.put(func.getSignature(), func);
        if (EzgMain.debug)
            System.out.println(EzgInterpreter.pref + "reg int func: " +
                               func.getSignature());
    }

    private static EzgType[] getArguments(int id) {
        EzgType[] args;

        switch (id) {
        case FN_PRINT:
        case FN_PRINTLN:
        case FN_SIZE:
        case FN_DATA:
            /* (...). */
            args = null;
            break;
        case FN_STR_TO_INT:
        case FN_STR_TO_FLOAT:
        case FN_SAVE:
        case FN_WIDTH:
            /* (string). */
            args = new EzgType[1];
            args[0] = new EzgString();
            break;
        case FN_INDEX:
        case FN_LAST_INDEX:
            /* (string,string). */
            args = new EzgType[2];
```



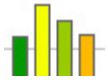
```
            args[0] = new EzgString();
            args[1] = new EzgString();
            break;
        case FN_SUBSTRING:
        case FN_FONT:
        case FN_TEXT:
            /* (string,int,int). */
            args = new EzgType[3];
            args[0] = new EzgString();
            args[1] = new EzgInt();
            args[2] = new EzgInt();
            break;
        case FN_RANDOM:
        case FN_SHOW:
        case FN_RESET:
        case FN_PUSH:
        case FN_POP:
        case FN_HEIGHT:
            /* (). */
            args = new EzgType[0];
            break;
        case FN_ROUND:
        case FN_FLOOR:
        case FN_CEIL:
        case FN_ABS_FLOAT:
        case FN_SQRT:
        case FN_EXP:
        case FN_LOG:
        case FN_SIN:
        case FN_COS:
        case FN_TAN:
        case FN_ASIN:
        case FN_ACOS:
        case FN_ATAN:
        case FN_ROTATE:
            /* (float). */
            args = new EzgType[1];
            args[0] = new EzgFloat();
            break;
        case FN_ABS_INT:
        case FN_STROKE:
            /* (int). */
            args = new EzgType[1];
            args[0] = new EzgInt();
            break;
        case FN_POW:
        case FN_SCALE:
        case FN_SHEAR:
        case FN_TRANSLATE:
            /* (float,float). */
            args = new EzgType[2];
            args[0] = new EzgFloat();
            args[1] = new EzgFloat();
            break;
        case FN_CANVAS:
        case FN_POINT:
            /* (int,int). */
```



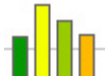
```
            args = new EzgType[2];
            args[0] = new EzgInt();
            args[1] = new EzgInt();
            break;
        case FN_BACKGROUND:
        case FN_COLOR:
            /* (int,int,int). */
            args = new EzgType[3];
            for (int i = 0; i < 3; i++)
                args[i] = new EzgInt();
            break;
        case FN_LINE:
        case FN_RECT:
        case FN_FILL_RECT:
        case FN_OVAL:
        case FN_FILL_OVAL:
            /* (int,int,int,int). */
            args = new EzgType[4];
            for (int i = 0; i < 4; i++)
                args[i] = new EzgInt();
            break;
        case FN_POLYGON:
        case FN_FILL_POLYGON:
            /* (int[],int[],int). */
            args = new EzgType[3];
            args[0] = new EzgArray(new EzgInt());
            args[1] = new EzgArray(new EzgInt());
            args[2] = new EzgInt();
            break;
        case FN_ARC:
        case FN_FILL_ARC:
            /* (int,int,int,int,int,int). */
            args = new EzgType[6];
            for (int i = 0; i < 6; i++)
                args[i] = new EzgInt();
            break;
        default:
            throw new RuntimeException("unknown internal function");
        }

        return args;
    }

    private static void checkParameters(EzgFunction func, EzgType[] params) {
        String error = null;
        int id = func.getId();
        if (FN_PRINT == id) {
            if (0 == params.length)
                error = func.getName() + "() accepts 1 or more parameters";
        } else if (FN_SIZE == id) {
            if (1 != params.length)
                error = func.getName() + "() accepts exactly 1 parameter";
            else if (!(params[0] instanceof EzgString) &&
                     !(params[0] instanceof EzgArray))
                error = "the argument to " + func.getName() +
                        "() must be a string or an array";
        }
```



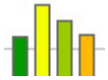
```
        if (null != error)
            throw new RuntimeException(error);
    }

    public static EzgType call(EzgPainter painter, EzgFunction func,
                               EzgType[] params) {
        checkParameters(func, params);

        String msg = func.getName() + "(";
        for (int i = 0; i < params.length; i++) {
            if (params[i] instanceof EzgString)
                msg += "\"" + params[i] + "\"";
            else
                msg += params[i];
            if (i < params.length - 1)
                msg += ",";
        }
        msg += ")";

        int     i1, i2, i3, i4, i5, i6;
        float   f1, f2;
        String  s1, s2;
        int[]   a1, a2;
        switch (func.getId()) {
        case FN_PRINT:
            msg = func.getName() + "(): ";
            if (EzgMain.debug)
                System.out.print(EzgInterpreter.pref + msg);
            for (int i = 0; i < params.length; i++)
                System.out.print(params[i]);
            if (EzgMain.debug)
                System.out.println();
            return new EzgVoid();
        case FN_PRINTLN:
            msg = func.getName() + "()";
            if (0 == params.length)
                System.out.println(EzgMain.debug ?
                                   EzgInterpreter.pref + msg : "");
            msg += ": ";
            for (int i = 0; i < params.length; i++) {
                if (EzgMain.debug)
                    System.out.print(EzgInterpreter.pref + msg);
                System.out.println(params[i]);
            }
            return new EzgVoid();
        case FN_STR_TO_INT:
            debug(msg);
            return ((EzgString)params[0]).toEzgInt();
        case FN_STR_TO_FLOAT:
            debug(msg);
            return ((EzgString)params[0]).toEzgFloat();
        case FN_INDEX:
            debug(msg);
            s1 = ((EzgString)params[0]).getVar();
            s2 = ((EzgString)params[1]).getVar();
            return new EzgInt(s1.indexOf(s2));
        case FN_LAST_INDEX:
```



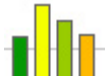
```
            debug(msg);
            s1 = ((EzgString)params[0]).getVar();
            s2 = ((EzgString)params[1]).getVar();
            return new EzgInt(s1.lastIndexOf(s2));
        case FN_SUBSTRING:
            debug(msg);
            s1 = ((EzgString)params[0]).getVar();
            i1 = ((EzgInt)params[1]).getVar();
            i2 = ((EzgInt)params[2]).getVar();
            return new EzgString(s1.substring(i1, i2));
        case FN_SIZE:
            debug(msg);
            if (params[0] instanceof EzgString)
                return new EzgInt(((EzgString)params[0]).getSize());
            else /* EzgArray. */
                return new EzgInt(((EzgArray)params[0]).getSize());
        case FN_RANDOM:
            debug(msg);
            return new EzgFloat((float)Math.random());
        case FN_ROUND:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgInt(Math.round(f1));
        case FN_FLOOR:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgInt((int)Math.floor(f1));
        case FN_CEIL:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgInt((int)Math.ceil(f1));
        case FN_ABS_INT:
            debug(msg);
            i1 = ((EzgInt)params[0]).getVar();
            return new EzgInt(Math.abs(i1));
        case FN_ABS_FLOAT:
            debug(msg);
            f1 = ((EzgFloat)params[0]).getVar();
            return new EzgFloat(Math.abs(f1));
        case FN_SQRT:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.sqrt(f1));
        case FN_POW:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            f2 = EzgFloat.floatValue(params[1]);
            return new EzgFloat((float)Math.pow(f1,f2));
        case FN_EXP:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.exp(f1));
        case FN_LOG:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.log(f1));
        case FN_SIN:
```



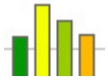
```
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.sin(Math.toRadians(f1)));
        case FN_COS:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.cos(Math.toRadians(f1)));
        case FN_TAN:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.tan(Math.toRadians(f1)));
        case FN_ASIN:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.toDegrees(Math.asin(f1)));
        case FN_ACOS:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.toDegrees(Math.acos(f1)));
        case FN_ATAN:
            debug(msg);
            f1 = EzgFloat.floatValue(params[0]);
            return new EzgFloat((float)Math.toDegrees(Math.atan(f1)));
        case FN_DATA:
            String filename;
            String delims = " \t\n\r\f";
            boolean header = false;
            boolean quoted = false;
            msg = func.getName();
            if (params.length < 1 || params.length > 4)
                throw new RuntimeException(msg + "() accepts one to four " +
                                           "arguments");
            if (!(params[0] instanceof EzgString))
                throw new RuntimeException("the first argument to " + msg +
                                           "() must be a string");
            filename = ((EzgString)params[0]).getVar();
            if (params.length > 1) {
                if (params[1] instanceof EzgString)
                    delims = ((EzgString)params[1]).getVar();
                else if (params[1] instanceof EzgBool)
                    header = ((EzgBool)params[1]).getVar();
                else
                    throw new RuntimeException("the second argument to " +
                                               msg + "() must be a string " +
                                               "or a bool");
            }
            if (params.length > 2) {
                if (!(params[2] instanceof EzgBool))
                    throw new RuntimeException("the third argument to " + msg
+
                                               "() must be a bool");
                if (params[1] instanceof EzgString)
                    header = ((EzgBool)params[2]).getVar();
                else
                    quoted = ((EzgBool)params[2]).getVar();
            }
            if (params.length > 3) {
```



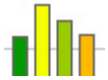
```
            if (!(params[3] instanceof EzgBool))
                throw new RuntimeException("the fourth argument to " +
                                           msg + "() must be a bool");
            quoted = ((EzgBool)params[3]).getVar();
        }
        debug(msg + "(\"" + filename + "\",\"" + delims + "\"," +
              header + "," + quoted + ")");
        return getData(filename, delims, header, quoted);
    case FN_CANVAS:
        verbose(msg);
        painter.newCanvas(((EzgInt)params[0]).getVar(),
                          ((EzgInt)params[1]).getVar());
        return new EzgVoid();
    case FN_SHOW:
        verbose(msg);
        if (!EzgMain.test)
            painter.showCanvas();
        return new EzgVoid();
    case FN_SAVE:
        verbose(msg);
        if (!EzgMain.test)
            painter.saveCanvas(((EzgString)params[0]).getVar());
        return new EzgVoid();
    case FN_BACKGROUND:
        verbose(msg);
        i1 = ((EzgInt)params[0]).getVar();
        i2 = ((EzgInt)params[1]).getVar();
        i3 = ((EzgInt)params[2]).getVar();
        painter.setBackground(i1, i2, i3);
        return new EzgVoid();
    case FN_COLOR:
        verbose(msg);
        i1 = ((EzgInt)params[0]).getVar();
        i2 = ((EzgInt)params[1]).getVar();
        i3 = ((EzgInt)params[2]).getVar();
        painter.setColor(i1, i2, i3);
        return new EzgVoid();
    case FN_STROKE:
        verbose(msg);
        i1 = ((EzgInt)params[0]).getVar();
        painter.setStroke(i1);
        return new EzgVoid();
    case FN_FONT:
        verbose(msg);
        s1 = ((EzgString)params[0]).getVar();
        i1 = ((EzgInt)params[1]).getVar();
        i2 = ((EzgInt)params[2]).getVar();
        painter.setFont(s1, i1, i2);
        return new EzgVoid();
    case FN_SCALE:
        verbose(msg);
        f1 = EzgFloat.floatValue(params[0]);
        f2 = EzgFloat.floatValue(params[1]);
        painter.scale(f1, f2);
        return new EzgVoid();
    case FN_SHEAR:
        verbose(msg);
```



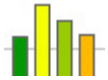
```
            f1 = EzgFloat.floatValue(params[0]);
            f2 = EzgFloat.floatValue(params[1]);
            painter.shear(f1, f2);
            return new EzgVoid();
        case FN_ROTATE:
            verbose(msg);
            f1 = EzgFloat.floatValue(params[0]);
            painter.rotate(f1);
            return new EzgVoid();
        case FN_TRANSLATE:
            verbose(msg);
            f1 = EzgFloat.floatValue(params[0]);
            f2 = EzgFloat.floatValue(params[1]);
            painter.translate(f1, f2);
            return new EzgVoid();
        case FN_RESET:
            verbose(msg);
            painter.resetTransform();
            return new EzgVoid();
        case FN_PUSH:
            verbose(msg);
            painter.pushTransform();
            return new EzgVoid();
        case FN_POP:
            verbose(msg);
            painter.popTransform();
            return new EzgVoid();
        case FN_POINT:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            painter.drawPoint(i1, i2);
            return new EzgVoid();
        case FN_LINE:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            painter.drawLine(i1, i2, i3, i4);
            return new EzgVoid();
        case FN_RECT:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            painter.drawRect(i1, i2, i3, i4);
            return new EzgVoid();
        case FN_FILL_RECT:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            painter.fillRect(i1, i2, i3, i4);
            return new EzgVoid();
```



```
        case FN_POLYGON:
            verbose(msg);
            a1 = EzgInt.convertArray(((EzgArray)params[0]).getVar());
            a2 = EzgInt.convertArray(((EzgArray)params[1]).getVar());
            i1 = ((EzgInt)params[2]).getVar();
            painter.drawPolygon(a1, a2, i1);
            return new EzgVoid();
        case FN_FILL_POLYGON:
            verbose(msg);
            a1 = EzgInt.convertArray(((EzgArray)params[0]).getVar());
            a2 = EzgInt.convertArray(((EzgArray)params[1]).getVar());
            i1 = ((EzgInt)params[2]).getVar();
            painter.fillPolygon(a1, a2, i1);
            return new EzgVoid();
        case FN_ARC:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            i5 = ((EzgInt)params[4]).getVar();
            i6 = ((EzgInt)params[5]).getVar();
            painter.drawArc(i1, i2, i3, i4, i5, i6);
            return new EzgVoid();
        case FN_FILL_ARC:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            i5 = ((EzgInt)params[4]).getVar();
            i6 = ((EzgInt)params[5]).getVar();
            painter.fillArc(i1, i2, i3, i4, i5, i6);
            return new EzgVoid();
        case FN_OVAL:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            painter.drawOval(i1, i2, i3, i4);
            return new EzgVoid();
        case FN_FILL_OVAL:
            verbose(msg);
            i1 = ((EzgInt)params[0]).getVar();
            i2 = ((EzgInt)params[1]).getVar();
            i3 = ((EzgInt)params[2]).getVar();
            i4 = ((EzgInt)params[3]).getVar();
            painter.fillOval(i1, i2, i3, i4);
            return new EzgVoid();
        case FN_TEXT:
            verbose(msg);
            s1 = ((EzgString)params[0]).getVar();
            i1 = ((EzgInt)params[1]).getVar();
            i2 = ((EzgInt)params[2]).getVar();
            painter.drawString(s1, i1, i2);
            return new EzgVoid();
```



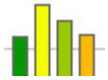
```
        case FN_WIDTH:
            debug(msg);
            s1 = ((EzgString)params[0]).getVar();
            return new EzgInt(painter.stringWidth(s1));
        case FN_HEIGHT:
            debug(msg);
            return new EzgInt(painter.stringHeight());
        default:
            throw new RuntimeException("unknown internal function");
        }
    }

    private static EzgArray getData(String filename, String delims,
                                    boolean header, boolean quoted) {
        try {
            BufferedReader in = new BufferedReader(new FileReader(filename));
            Vector fields;
            Vector records = new Vector();

            if (header)
                in.readLine(); /* Skip header row. */

            /* Read data from file. */
            String line;
            EzgStringTokenizer est;
            while (null != (line = in.readLine())) {
                est = new EzgStringTokenizer(line, delims, false, true);
                if (quoted)
                    est.setQuotes(EzgStringTokenizer.SOMETIMES);
                fields = new Vector();
                while (est.hasMoreTokens())
                    fields.add(est.nextToken());
                records.add(fields);
            }
            in.close();
            if (records.size() == 0)
                throw new RuntimeException("no data in file " + filename);

            /* Store data in an array. */
            EzgArray[] rows = new EzgArray[records.size()];
            EzgString[] cols;
            for (int i = 0; i < rows.length; i++) {
                fields = (Vector)records.remove(0);
                cols = new EzgString[fields.size()];
                for (int j = 0; j < cols.length; j++)
                    cols[j] = new EzgString((String)fields.remove(0));
                rows[i] = new EzgArray(cols[0].type(), cols);
            }

            return new EzgArray(rows[0].type(), rows);
        } catch (FileNotFoundException e) {
            throw new RuntimeException(filename + " (file not found)");
        } catch (IOException e) {
            throw new RuntimeException(filename + " (I/O error)");
        }
    }
```



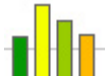
```
    private static void verbose(String msg) {
        if (EzgMain.verbose) {
            if (EzgMain.debug)
                System.out.print(EzgInterpreter.pref);
            System.out.println(msg);
        }
    }

    private static void debug(String msg) {
        if (EzgMain.debug)
            System.out.println(EzgInterpreter.pref + msg);
    }
}
```

B.2.7 src/ezg/EzgPainter.java

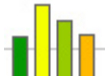
```
/**
 * Title:       EzgPainter.java
 * Description: Class with all the graph drawing functionality.
 * Modified:    2007-05-06
 */

package ezg;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.Stack;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class EzgPainter extends JPanel {
    private static final long serialVersionUID = 1L;

    private JFrame frame;
    private Color bgColor;
    private Color penColor;
    private BasicStroke stroke;
    private Font font;
    private BufferedImage canvas;
    private Graphics2D g2d; /* Graphics object to draw on the canvas. */
    private Stack transforms;

    /** Graph panel dimensions and title. */
```



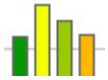
```
    private int width;
    private int height;
    private String title = "EZGraphs";

    public EzgPainter() {
        frame    = null;
        bgColor  = new Color(255, 255, 255); /* Default bg color: white. */
        penColor = new Color(0, 0, 0);       /* Default pen color: black. */
        stroke   = new BasicStroke(1);       /* Default pen stroke width: 1.
*/
        font     = new Font(null, Font.PLAIN, 10);
    }

    private void setBackground() {
        pushTransform();
        resetTransform();
        g2d.setColor(bgColor);
        g2d.fillRect(0, 0, width, height);
        g2d.setColor(penColor);
        popTransform();
    }

    public void newCanvas(int width, int height) {
        canvas = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        transforms = new Stack();
        g2d = canvas.createGraphics();
        g2d.setColor(penColor);
        g2d.setStroke(stroke);
        g2d.setFont(font);
        this.width = width;
        this.height = height;
        setBackground();
    }

    public void showCanvas() {
        /* Set the size of the panel. */
        setPreferredSize(new Dimension(width, height));

        /* Make sure that there is a canvas to show. */
        if (null == canvas)
            throw new RuntimeException("no canvas has been specified");

        /* Set up the frame and add the panel to it. */
        if (null == frame)
            frame = new JFrame(title);
        frame.getContentPane().add(this, BorderLayout.CENTER);
        frame.setResizable(false);
        frame.pack();
        frame.setLocationRelativeTo(null); /* Centered on screen */
        frame.setVisible(true);

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                Frame f = (Frame) e.getSource();
                f.setVisible(false);
                f.dispose();
```



```
                g2d.dispose();
            }
        });
    }

    public void saveCanvas(String filepath) {
        EzgMain.validatePath(filepath);
        File file = new File(filepath + ".png"); /* Only PNG files created.
*/
        try {
            EzgMain.createPath(file);
            ImageIO.write(canvas, "png", file);
        } catch (Exception e) {
            throw new RuntimeException("\"" + filepath + "\" (" +
                                       e.getMessage() + ")");
        }
    }

    public void setBackground(int red, int green, int blue) {
        bgColor = new Color(red, green, blue);
        setBackground();
    }

    public void setColor(int red, int green, int blue) {
        penColor = new Color(red, green, blue);
        g2d.setColor(penColor);
    }

    public void setStroke(int width) {
        stroke = new BasicStroke(width);
        g2d.setStroke(stroke);
    }

    public void setFont(String name, int style, int size) {
        font = new Font(name, style, size);
        g2d.setFont(font);
    }

    public void scale(float sx, float sy) {
        g2d.scale(sx, sy);
    }

    public void shear(float shx, float shy) {
        g2d.shear(shx, shy);
    }

    public void rotate(float theta) {
        g2d.rotate(Math.toRadians(theta));
    }

    public void translate(float tx, float ty) {
        g2d.translate(tx, ty);
    }

    public void resetTransform() {
        g2d.setTransform(new AffineTransform());
    }
```



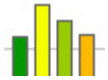
```
    public void pushTransform() {
        transforms.push(g2d.getTransform());
    }

    public void popTransform() {
        g2d.setTransform((AffineTransform)transforms.pop());
    }

    public void drawPoint(int x, int y) {
        int r = (int)stroke.getLineWidth();
        g2d.fillOval(x - r/2, y - r/2, r, r);
    }

    public void drawLine(int x1, int y1, int x2, int y2) {
        g2d.drawLine(x1, y1, x2, y2);
    }

    public void drawRect(int x, int y, int width, int height) {
        g2d.drawRect(x, y, width, height);
    }

    public void fillRect(int x, int y, int width, int height) {
        g2d.fillRect(x, y, width, height);
    }

    public void drawPolygon(int[] xPoints, int[] yPoints, int nPoints) {
        g2d.drawPolygon(xPoints, yPoints, nPoints);
    }

    public void fillPolygon(int[] xPoints, int[] yPoints, int nPoints) {
        g2d.fillPolygon(xPoints, yPoints, nPoints);
    }

    public void drawArc(int x, int y, int width, int height,
                        int startAngle, int arcAngle) {
        g2d.drawArc(x, y, width, height, startAngle, arcAngle);
    }

    public void fillArc(int x, int y, int width, int height,
                        int startAngle, int arcAngle) {
        g2d.fillArc(x, y, width, height, startAngle, arcAngle);
    }

    public void drawOval(int x, int y, int width, int height) {
        g2d.drawOval(x, y, width, height);
    }

    public void fillOval(int x, int y, int width, int height) {
        g2d.fillOval(x, y, width, height);
    }

    public void drawString(String str, int x, int y) {
        g2d.drawString(str, x, y);
    }

    public int stringWidth(String str) {
```



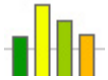
```
        return g2d.getFontMetrics().stringWidth(str);
    }

    public int stringHeight() {
        return g2d.getFontMetrics().getHeight();
    }

    /**
     * Overrides parent class's paint() method. This method gets called
     * automatically everytime the panel needs to be displayed on screen. The
     * Graphics object used here draws on the panel.
     */
    public void paint(Graphics g) {
        Graphics2D panelG2D = (Graphics2D)g;
        panelG2D.drawImage(canvas, 0, 0, this);
        panelG2D.dispose();
    }
}
```

B.2.8 src/ezg/EzgStringTokenizer.java

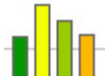
```
/**
 * Title:       EzgStringTokenizer.java
 * Description: Similar to java.util.StringTokenizer with added features.
 * Company:     InCHOIR, Columbia University
 * Author:      Vincent Dobrev
 * Modified:    2007-04-26
 */

package ezg;

import java.util.NoSuchElementException;

public class EzgStringTokenizer {
    public static final int NEVER     = 0;
    public static final int SOMETIMES = 1;
    public static final int NONEMPTY  = 2;
    public static final int ALWAYS    = 3;

    private static final int NA        = 0;
    private static final int EMPTY     = 1;
    private static final int TOKEN     = 2;
    private static final int DELIMITER = 3;

    private String str;
    private String delims;

    public boolean returnDelims;
    public boolean returnEmptyTokens;

    private int quotes;
    private char quoteChar;

    private int currPos;
    private int newPos;
```



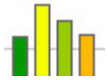
```
    private int maxPos;
    private int lastToken;
    private boolean quotedToken;
    private boolean delimsChanged;

    /**
     * minDelimChar stores the value of the delimiter character with the
     * lowest value. It is used to optimize the detection of delimiter
     * characters.
     */
    private char minDelimChar;

    /**
     * maxDelimChar stores the value of the delimiter character with the
     * highest value. It is used to optimize the detection of delimiter
     * characters.
     */
    private char maxDelimChar;

    /**
     * Sets minDelimChar to the lowest char in the delimiter set and
     * maxDelimChar to the highest char in the delimiter set.
     */
    private void setMinMaxDelimChars() {
        if (delims == null) {
            minDelimChar = maxDelimChar = 0;
            return;
        }

        char min, max;
        min = max = delims.charAt(0);
        for (int i = 1; i < delims.length(); i++) {
            char c = delims.charAt(i);
            if (min > c)
                min = c;
            if (max < c)
                max = c;
        }
        minDelimChar = min;
        maxDelimChar = max;
    }

    /**
     * Determines if the specified character is a delimiter.
     *
     * @param    c   the character to be tested.
     * @return   <code>true</code> if the character is a delimiter;
     *           <code>false</code> otherwise.
     */
    private boolean isDelimiter(char c) {
        if (c >= minDelimChar && c <= maxDelimChar && delims.indexOf(c) >= 0)
            return true;
        return false;
    }

    /**
     * Makes sure the value of the <code>quotes</code> flag is one of the
```



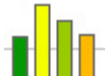
```
     * fields <code>NEVER</code>, <code>SOMETIMES</code>,
     * <code>NONEMPTY</code>, or <code>ALWAYS</code>.
     *
     * @exception   IllegalArgumentException   if the value of the
     *                  <code>quotes</code> flag is not one of the fields
     *                  <code>NEVER</code>, <code>SOMETIMES</code>,
     *                  <code>NONEMPTY</code>, or <code>ALWAYS</code>.
     */
    private void validateQuotes() {
        if (quotes != NEVER && quotes != SOMETIMES &&
            quotes != NONEMPTY && quotes != ALWAYS)
            throw new IllegalArgumentException(String.valueOf(quotes) +
                                               " (Illegal quotes argument)");
    }

    /**
     * Makes sure the value of the <code>quoteChar</code> field is not one of
     * the delimiter characters while the value of the <code>quotes</code>
flag
     * is not <code>NEVER</code>.
     *
     * @exception   IllegalArgumentException   if the value of the
     *                  <code>quoteChar</code> field is one of the delimiter
     *                  characters while the value of the <code>quotes</code>
     *                  flag is not <code>NEVER</code>.
     */
    private void validateQuoteChar(){
        if (quotes != NEVER && isDelimiter(quoteChar))
            throw new IllegalArgumentException(quoteChar +
                " (quoteChar cannot be in the delimiter set)");
    }

    /**
     * Constructs a string tokenizer for the specified string. All
     * characters in the <code>delims</code> argument are the delimiters
     * for separating tokens.
     * <p>
     * If the <code>returnDelims</code> flag is <code>true</code>, then
     * the delimiter characters are also returned as tokens. Each
     * delimiter is returned as a string of length one. If the flag is
     * <code>false</code>, the delimiter characters are skipped and only
     * serve as separators between tokens.
     * <p>
     * If the <code>returnEmptyTokens</code> flag is <code>true</code>, then
     * each token between two consecutive delimiters is returned as a string
of
     * length zero. If quotes are used to surround the tokens, then a token
     * consisting of two consecutive quote characters is also considered an
     * empty token.
     * <p>
     * The <code>quotes</code> flag indicates whether tokens are enclosed
     * by quotes. Possible values for it are the fields <code>NEVER</code>,
     * <code>SOMETIMES</code>, <code>NONEMPTY</code>, or <code>ALWAYS</code>:
     * <ul>
     * <li><code>NEVER</code>
     * <li><code>SOMETIMES</code>
     * <li><code>NONEMPTY</code>
```



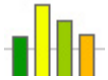
```
     * <li><code>ALWAYS</code>
     * </ul>
     * Any other value will result in <code>IllegalArgumentException</code>.
     * <p>
     * The <code>quoteChar</code> argument sets the quote character to be
used
     * when returning tokens enclosed by quotes. This character cannot be one
     * of the delimiter characters and if it is and the value of the
     * <code>quotes</code> flag is not <code>NEVER</code>,
     * <code>IllegalArgumentException</code> will be thrown.
     *
     * @param   str                 a string to be parsed.
     * @param   delims              the delimiters.
     * @param   returnDelims        flag indicating whether to return the
     *                              delimiters as tokens.
     * @param   returnEmptyTokens   flag indicating whether empty tokens
should
     *                              be returned.
     * @param   quotes              flag indicating whether tokens are
enclosed
     *                              by quotes.
     * @param   quoteChar           the quote character.
     * @exception   IllegalArgumentException   if the value of the
     *                  <code>quotes</code> flag is not one of the fields
     *                  <code>NEVER</code>, <code>SOMETIMES</code>,
     *                  <code>NONEMPTY</code>, or <code>ALWAYS</code>. Or if
     *                  the value of the <code>quoteChar</code> argument is
     *                  one of the delimiter characters while the value of
the
     *                  <code>quotes</code> flag is not <code>NEVER</code>.
     */
    public EzgStringTokenizer(String str, String delims,
                              boolean returnDelims,
                              boolean returnEmptyTokens,
                              int quotes, char quoteChar) {
        this.str = str;
        this.delims = delims;
        setMinMaxDelimChars();
        this.returnDelims = returnDelims;
        this.returnEmptyTokens = returnEmptyTokens;
        this.quotes = quotes;
        validateQuotes();
        this.quoteChar = quoteChar;
        validateQuoteChar();
        currPos = 0;
        newPos = -1;
        maxPos = str.length();
        lastToken = NA;
        quotedToken = false;
        delimsChanged = false;
    }

    /**
     * Constructs a string tokenizer for the specified string. All
     * characters in the <code>delims</code> argument are the delimiters
     * for separating tokens.
     * <p>
```



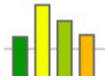
```
     * If the <code>returnDelims</code> flag is <code>true</code>, then
     * the delimiter characters are also returned as tokens. Each
     * delimiter is returned as a string of length one. If the flag is
     * <code>false</code>, the delimiter characters are skipped and only
     * serve as separators between tokens.
     * <p>
     * If the <code>returnEmptyTokens</code> flag is <code>true</code>, then
     * each token between two consecutive delimiters is returned as a string
of
     * length zero. If quotes are used to surround the tokens, then a token
     * consisting of two consecutive quote characters is also considered an
     * empty token.
     * <p>
     * The tokenizer will use the default <code>quotes</code> flag, which is
     * <code>NEVER</code>. The <code>quoteChar</code> field is set to '"'
(the
     * double quote character), even though that is irrelevant at this time,
     * since the <code>quotes</code> flag is set to <code>NEVER</code>.
     *
     * @param   str                 a string to be parsed.
     * @param   delims              the delimiters.
     * @param   returnDelims        flag indicating whether to return the
     *                              delimiters as tokens.
     * @param   returnEmptyTokens   flag indicating whether empty tokens
should
     *                              be returned.
     */
    public EzgStringTokenizer(String str, String delims,
                              boolean returnDelims,
                              boolean returnEmptyTokens) {
        this(str, delims, returnDelims, returnEmptyTokens, NEVER, '"');
    }

    /**
     * Constructs a string tokenizer for the specified string. All
     * characters in the <code>delims</code> argument are the delimiters
     * for separating tokens.
     * <p>
     * If the <code>returnDelims</code> flag is <code>true</code>, then
     * the delimiter characters are also returned as tokens. Each
     * delimiter is returned as a string of length one. If the flag is
     * <code>false</code>, the delimiter characters are skipped and only
     * serve as separators between tokens.
     * <p>
     * The tokenizer will not return empty tokens and will use the default
     * <code>quotes</code> flag, which is <code>NEVER</code>. The
     * <code>quoteChar</code> field is set to '"' (the double quote
character),
     * even though that is irrelevant at this time, since the
     * <code>quotes</code> flag is set to <code>NEVER</code>.
     *
     * @param   str            a string to be parsed.
     * @param   delims         the delimiters.
     * @param   returnDelims   flag indicating whether to return the
     *                         delimiters as tokens.
     */
    public EzgStringTokenizer(String str, String delims,
```



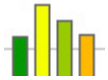
```
                                boolean returnDelims) {
        this(str, delims, returnDelims, false, NEVER, '"');
    }

    /**
     * Constructs a string tokenizer for the specified string. All
     * characters in the <code>delims</code> argument are the delimiters
     * for separating tokens.
     * <p>
     * The tokenizer will not return delimiters and empty tokens, and will
use
     * the default <code>quotes</code> flag, which is <code>NEVER</code>. The
     * <code>quoteChar</code> field is set to '"' (the double quote
character),
     * even though that is irrelevant at this time, since the
     * <code>quotes</code> flag is set to <code>NEVER</code>.
     *
     * @param   str      a string to be parsed.
     * @param   delims   the delimiters.
     */
    public EzgStringTokenizer(String str, String delims) {
        this(str, delims, false, false, NEVER, '"');
    }

    /**
     * Constructs a string tokenizer for the specified string. The tokenizer
     * uses the default delimiter set, which is " \t\n\r\f": the space
     * character, the tab character, the newline character, the carriage-
return
     * character, and the form-feed character.
     * <p>
     * The tokenizer will not return delimiters and empty tokens, and will
use
     * the default <code>quotes</code> flag, which is <code>NEVER</code>. The
     * <code>quoteChar</code> field is set to '"' (the double quote
character),
     * even though that is irrelevant at this time, since the
     * <code>quotes</code> flag is set to <code>NEVER</code>.
     *
     * @param   str   a string to be parsed.
     */
    public EzgStringTokenizer(String str) {
        this(str, " \t\n\r\f", false, false, NEVER, '"');
    }

    /**
     * Gets the value of the <code>quotes</code> flag.
     *
     * @return   the value of the <code>quotes</code> flag.
     */
    public int getQuotes() {
        return quotes;
    }

    /**
     * Sets the value of the <code>quotes</code> flag.
     *
```



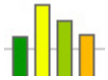
```
     * @param   quotes   flag indicating whether tokens are enclosed by
quotes.
     *                   Possible values are the fields <code>NEVER</code>,
     *                   <code>SOMETIMES</code>, <code>NONEMPTY</code>, or
     *                   <code>ALWAYS</code>.
     * @exception   IllegalArgumentException   if the value of the
     *                  <code>quotes</code> flag is not one of the fields
     *                  <code>NEVER</code>, <code>SOMETIMES</code>,
     *                  <code>NONEMPTY</code>, or <code>ALWAYS</code>. Or if
     *                  the value of the <code>quoteChar</code> field is one
of
     *                  the delimiter characters while the value of the
     *                  <code>quotes</code> flag is not <code>NEVER</code>.
     */
    public void setQuotes(int quotes) {
        this.quotes = quotes;
        validateQuotes();
    }

    /**
     * Gets the value of the <code>quoteChar</code> field.
     *
     * @return   the value of the <code>quoteChar</code> field.
     */
    public char getQuoteChar() {
        return quoteChar;
    }

    /**
     * Sets the value of the quote character.
     *
     * @param   quoteChar   the quote character.
     * @exception   IllegalArgumentException   if the value of the
     *                  <code>quoteChar</code> argument is one of the
delimiter
     *                  characters while the value of the <code>quotes</code>
     *                  flag is not <code>NEVER</code>.
     */
    public void setQuoteChar(char quoteChar) {
        this.quoteChar = quoteChar;
        validateQuoteChar();
    }

    private int getNextTokenPossition(int startPos) {
        if (delims == null)
            throw new NullPointerException();

        int pos = startPos;
        while (pos < maxPos) {
            char c = str.charAt(pos);
            if (isDelimiter(c)) {
                if (quotes == ALWAYS &&
                    (lastToken == NA || lastToken == DELIMITER))
                    throw new NoSuchElementException("Unquoted empty token");
                if (returnDelims)
                    break;
                if (returnEmptyTokens &&
```



```
                    (lastToken == NA || lastToken == DELIMITER))
                    break;
                lastToken = DELIMITER;
                pos++;
            } else if (c == quoteChar) {
                if (quotes == NEVER)
                    break;
                if (pos+1 == maxPos)
                    throw new NoSuchElementException("Unmatched quoteChar");
                if (returnEmptyTokens || str.charAt(pos+1) != quoteChar)
                    break;
                if (pos+2 < maxPos && !isDelimiter(str.charAt(pos+2)))
                    break;
                lastToken = EMPTY;
                pos += 2;
            } else {
                if (quotes == NONEMPTY || quotes == ALWAYS)
                    throw new NoSuchElementException("Unquoted token");
                break;
            }
        }
        return pos;
    }

    private int scanToken(int startPos) {
        int pos = startPos;
        char c = str.charAt(startPos);
        if (quotes != NEVER && c == quoteChar) {
            quotedToken = true;
            pos++;
        }
        while (pos < maxPos) {
            c = str.charAt(pos);
            if (isDelimiter(c) && !quotedToken)
                break;
            if (c == quoteChar && quotedToken) {
                pos++;
                if (pos == maxPos) {
                    if (pos == startPos+2)
                        lastToken = EMPTY;
                    break;
                }
                c = str.charAt(pos);
                if (isDelimiter(c)) {
                    if (pos == startPos+2)
                        lastToken = EMPTY;
                    break;
                }
                if (c != quoteChar)
                    throw new NoSuchElementException("Single quoteChar in " +
                                                     "quoted token");
            }
            lastToken = TOKEN;
            pos++;
        }
        if (startPos == pos) {
            if (returnEmptyTokens &&
```



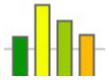
```
                (lastToken == NA || lastToken == DELIMITER)) {
                lastToken = EMPTY;
            } else {
                lastToken = DELIMITER;
                pos++;
            }
        }
        if (quotedToken && str.charAt(pos-1) != quoteChar)
            throw new NoSuchElementException("Unmatched quoteChar");
        return pos;
    }

    public boolean hasMoreTokens() {
        newPos = getNextTokenPossition(currPos);
        if (newPos == maxPos && quotes == ALWAYS &&
            (lastToken == NA || lastToken == DELIMITER))
            throw new NoSuchElementException("Unquoted empty token");
        if (newPos == maxPos && returnEmptyTokens &&
            (lastToken == NA || lastToken == DELIMITER))
            return true;
        return newPos < maxPos;
    }

    public String nextToken() {
        currPos = (newPos >= 0 && !delimsChanged) ?
            newPos : getNextTokenPossition(currPos);

        /* Reset these anyway. */
        delimsChanged = false;
        quotedToken = false;
        newPos = -1;

        if (currPos == maxPos && returnEmptyTokens &&
            (lastToken == NA || lastToken == DELIMITER)) {
            lastToken = EMPTY;
            return "";
        }
        if (currPos >= maxPos)
            throw new NoSuchElementException();

        int startPos = currPos;
        currPos = scanToken(currPos);
        String token = str.substring(startPos, currPos);
        if (quotedToken) {
            token = token.substring(1, token.length()-1);
            token = token.replaceAll(String.valueOf(quoteChar) +
                                     String.valueOf(quoteChar),
                                     String.valueOf(quoteChar));
        }
        return token;
    }

    public String nextToken(String delims) {
        this.delims = delims;

        /* Delimiter string specified, so set the appropriate flag. */
        delimsChanged = true;
```



```
        setMinMaxDelimChars();
        return nextToken();
    }

    /**
     * Calculates the number of times that this tokenizer's
     * <code>nextToken</code> method can be called before it generates an
     * exception. The current position is not advanced.
     *
     * @return  the number of tokens remaining in the string using the
current
     *          delimiter set.
     * @see     EzgStringTokenizer#nextToken()
     */
    public int countTokens() {
        /* Save original values. */
        int pos = currPos;
        int last = lastToken;

        /* Count tokens. */
        int count = 0;
        while (hasMoreTokens()) {
            nextToken();
            count++;
        }

        /* Restore original values. */
        currPos = pos;
        lastToken = last;
        newPos = -1;

        return count;
    }
}
```

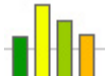
B.2.9 build.xml

```
<?xml version="1.0"?>
<project name="EZGraphs" default="compile">

  <property name="project"    location="."/>
  <property name="src"        location="${project}\src\ezg"/>
  <property name="classes"    location="${project}\classes"/>
  <property name="lib"        location="${project}\lib"/>
  <property name="tests-dir"  location="${project}\tests"/>
  <property name="tests-exe"  location="${tests-dir}\tests-exe.bat"/>
  <property name="tests-out"  location="${tests-dir}\tests-out"/>
  <property name="tests-ref"  location="${tests-dir}\tests-ref"/>
  <property name="tests-diff" location="${tests-dir}\tests-diff.txt"/>

  <path id="classpath">
    <pathelement location="${lib}\antlr.jar"/>
  </path>
```



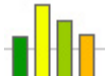
```
  <target name="init">
    <mkdir dir="${classes}"/>
  </target>

  <target name="compile" depends="init">
    <antlr target="${src}\grammar.g">
      <classpath refid="classpath"/>
    </antlr>
    <antlr target="${src}\walker.g">
      <classpath refid="classpath"/>
    </antlr>
    <javac srcdir="${src}" destdir="${classes}" debug="yes">
      <include name="**/*.java"/>
      <classpath refid="classpath"/>
    </javac>
  </target>

  <target name="clean">
    <delete includeEmptyDirs="true">
      <fileset dir="${classes}" includes="**/*"/>
      <fileset dir="${src}" includes="EzgAntlr*.*"/>
    </delete>
  </target>

  <target name="tests-init">
    <mkdir dir="${tests-out}"/>
  </target>

  <target name="tests-run" depends="tests-init">
    <exec executable="${tests-exe}" dir="${tests-dir}" failonerror="true"/>
    <antcall target="tests-diff"/>
    <antcall target="tests-clean"/>
    <echo message="Tests passed!"/>
  </target>

  <target name="tests-diff">
    <exec executable="diff" output="${tests-diff}" failonerror="true">
      <arg line="-r -x .svn"/>
      <arg value="${tests-out}"/>
      <arg value="${tests-ref}"/>
    </exec>
  </target>

  <target name="tests-clean">
    <delete includeEmptyDirs="true">
      <fileset dir="${tests-out}" includes="**/*"/>
      <fileset file="${tests-diff}"/>
    </delete>
  </target>

</project>
```

B.3 Types

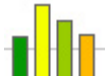


B.3.1 EzgType.java

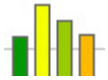
```
/**
 * Title:       EzgType.java
 * Description: Base class for data types. All other type classes inherit
from
 *              this one.
 * Modified:    2007-04-26
 */

package ezg.type;

public class EzgType {
    protected String name;
    protected boolean initialized;

    public EzgType() {
        name = null;
        initialized = false;
    }

    protected void mustBeType() {
        if (null == name && !initialized)
            return;
        throw new RuntimeException("object must be type");
    }

    protected void mustBeVariable() {
        if (null != name)
            return;
        throw new RuntimeException("object must be variable");
    }

    protected void mustHaveValue() {
        if (initialized)
            return;
        throw new RuntimeException("object must have value");
    }

    public EzgType type() {
        return new EzgType();
    }

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

    public String getName() {
        return name;
    }

    public EzgType[] getArray(int size) {
        throw new RuntimeException("unexpected call");
    }

    public void setName(String name) {
```



```
        this.name = name;
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Assignment after addition/concatenation. */
    public EzgType plasgn(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Assignment after subtraction. */
    public EzgType minasgn(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Assignment after multiplication. */
    public EzgType mulasgn(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Assignment after division. */
    public EzgType divasgn(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Assignment after modulo. */
    public EzgType modasgn(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Logical or. */
    public EzgType or(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Logical and. */
    public EzgType and(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Equality. */
    public EzgType eq(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Inequality. */
    public EzgType neq(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Relational less than. */
    public EzgType lt(EzgType b) {
        throw new RuntimeException("unexpected call");
    }
```



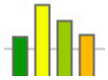
```
    /** Relational greater than. */
    public EzgType gt(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Relational less than or equal to. */
    public EzgType lteq(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Relational greater than or equal to. */
    public EzgType gteq(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Addition/concatenation. */
    public EzgType plus(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Subtraction. */
    public EzgType minus(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Multiplication. */
    public EzgType mult(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Division. */
    public EzgType div(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Modulo. */
    public EzgType mod(EzgType b) {
        throw new RuntimeException("unexpected call");
    }

    /** Prefix increment. */
    public EzgType prefinc() {
        throw new RuntimeException("unexpected call");
    }

    /** Postfix increment. */
    public EzgType postinc() {
        throw new RuntimeException("unexpected call");
    }

    /** Prefix decrement. */
    public EzgType prefdec() {
        throw new RuntimeException("unexpected call");
    }

    /** Postfix decrement. */
```



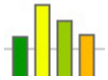
```
    public EzgType postdec() {
        throw new RuntimeException("unexpected call");
    }

    /** Unary plus. */
    public EzgType uplus() {
        throw new RuntimeException("unexpected call");
    }

    /** Unary minus. */
    public EzgType uminus() {
        throw new RuntimeException("unexpected call");
    }

    /** Logical not. */
    public EzgType not() {
        throw new RuntimeException("unexpected call");
    }

    public EzgType index(int[] indexes) {
        throw new RuntimeException("unexpected call");
    }

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

    public String toString() {
        throw new RuntimeException("unexpected call");
    }

    public String what() {
        String str = typeName();
        if (null != name)
            str += " " + name;
        if (initialized)
            str += " = " + toString();
        return str;
    }
}
```

B.3.2 EzgBool.java

```
/**
 * Title:       EzgBool.java
 * Description: Class to represent the boolean data type.
 * Modified:    2007-04-26
 */

package ezg.type;

public class EzgBool extends EzgType {
    private boolean var;

    public EzgBool(boolean var) {
```



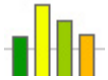
```
        super();
        this.var = var;
        initialized = true;
    }

    public EzgBool() {
        this(false);
        initialized = false;
    }

    public static boolean boolValue(EzgType b) {
        b.mustHaveValue();
        if (b instanceof EzgBool)
            return ((EzgBool)b).var;
        throw new RuntimeException("unexpected data type being cast to
bool");
    }

    /** Converts EzgBool[] to boolean[]. */
    public static boolean[] convertArray(EzgType[] src) {
        boolean[] dest = new boolean[src.length];
        for (int i = 0; i < dest.length; i++)
            dest[i] = boolValue(src[i]);
        return dest;
    }

    public EzgType type() {
        return new EzgBool();
    }

    public EzgType copy() {
        mustHaveValue();
        return new EzgBool(var);
    }

    public boolean getVar() {
        mustHaveValue();
        return var;
    }

    public EzgType[] getArray(int size) {
        return new EzgBool[size];
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        mustBeVariable();
        b.mustHaveValue();
        var = boolValue(b);
        initialized = true;
        return this;
    }

    /** Logical or. */
    public EzgType or(EzgType b) {
        mustHaveValue();
        return new EzgBool(var || boolValue(b));
```



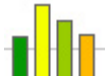
```
    }

    /** Logical and. */
    public EzgType and(EzgType b) {
        mustHaveValue();
        return new EzgBool(var && boolValue(b));
    }

    /** Equality. */
    public EzgType eq(EzgType b) {
        mustHaveValue();
        return new EzgBool(var == boolValue(b));
    }

    /** Inequality. */
    public EzgType neq(EzgType b) {
        mustHaveValue();
        return new EzgBool(var != boolValue(b));
    }

    /** Logical not. */
    public EzgType not() {
        mustHaveValue();
        return new EzgBool(!var);
    }

    public String typeName() {
        return "bool";
    }

    public String toString() {
        mustHaveValue();
        return Boolean.toString(var);
    }
}
```

B.3.3 EzgInt.java

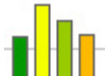
```
/**
 * Title:       EzgInt.java
 * Description: Class to represent the int data type.
 * Modified:    2007-05-06
 */

package ezg.type;

public class EzgInt extends EzgType {
    private int var;

    public EzgInt(int var) {
        super();
        this.var = var;
        initialized = true;
    }
```



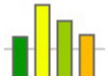
```
    public EzgInt() {
        this(0);
        initialized = false;
    }

    public static int intValue(EzgType b) {
        b.mustHaveValue();
        if (b instanceof EzgInt)
            return ((EzgInt)b).var;
        if (b instanceof EzgFloat)
            return (int)((EzgFloat)b).getVar();
        throw new RuntimeException("unexpected data type being cast to int");
    }

    /** Converts EzgInt[] to int[]. */
    public static int[] convertArray(EzgType[] src) {
        int[] dest = new int[src.length];
        for (int i = 0; i < dest.length; i++)
            dest[i] = ((EzgInt)src[i]).getVar();
        return dest;
    }

    public EzgType type() {
        return new EzgInt();
    }

    public EzgType copy() {
        mustHaveValue();
        return new EzgInt(var);
    }

    public int getVar() {
        mustHaveValue();
        return var;
    }

    public EzgType[] getArray(int size) {
        return new EzgInt[size];
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        mustBeVariable();
        b.mustHaveValue();
        if (b instanceof EzgFloat)
            throw new RuntimeException("possible loss of precision");
        var = intValue(b);
        initialized = true;
        return this;
    }

    /** Assignment after addition. */
    public EzgType plasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var += intValue(b);
        return this;
```



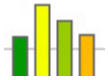
```
    }

    /** Assignment after subtraction. */
    public EzgType minasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var -= intValue(b);
        return this;
    }

    /** Assignment after multiplication. */
    public EzgType mulasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var *= intValue(b);
        return this;
    }

    /** Assignment after division. */
    public EzgType divasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var /= intValue(b);
        return this;
    }

    /** Assignment after modulo. */
    public EzgType modasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var %= intValue(b);
        return this;
    }

    /** Equality. */
    public EzgType eq(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(intValue(b) == var);
        return b.eq(this);
    }

    /** Inequality. */
    public EzgType neq(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(intValue(b) != var);
        return b.neq(this);
    }

    /** Relational less than. */
    public EzgType lt(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(var < intValue(b));
        return b.gt(this);
    }
```



```
    /** Relational greater than. */
    public EzgType gt(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(var > intValue(b));
        return b.lt(this);
    }

    /** Relational less than or equal to. */
    public EzgType lteq(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(var <= intValue(b));
        return b.gteq(this);
    }

    /** Relational greater than or equal to. */
    public EzgType gteq(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgBool(var >= intValue(b));
        return b.lteq(this);
    }

    /** Addition. */
    public EzgType plus(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgInt(var + intValue(b));
        return new EzgFloat(var + EzgFloat.floatValue(b));
    }

    /** Subtraction. */
    public EzgType minus(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgInt(var - intValue(b));
        return new EzgFloat(var - EzgFloat.floatValue(b));
    }

    /** Multiplication. */
    public EzgType mult(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgInt(var * intValue(b));
        return new EzgFloat(var * EzgFloat.floatValue(b));
    }

    /** Division. */
    public EzgType div(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgInt(var / intValue(b));
        return new EzgFloat(var / EzgFloat.floatValue(b));
    }
```

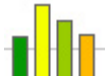


```
    /** Modulo. */
    public EzgType mod(EzgType b) {
        mustHaveValue();
        if (b instanceof EzgInt)
            return new EzgInt(var % intValue(b));
        return new EzgFloat(var % EzgFloat.floatValue(b));
    }

    /** Prefix increment. */
    public EzgType prefinc() {
        mustBeVariable();
        mustHaveValue();
        return new EzgInt(++var);
    }

    /** Prefix decrement. */
    public EzgType prefdec() {
        mustBeVariable();
        mustHaveValue();
        return new EzgInt(--var);
    }

    /** Postfix increment. */
    public EzgType postinc() {
        mustBeVariable();
        mustHaveValue();
        return new EzgInt(var++);
    }

    /** Postfix decrement. */
    public EzgType postdec() {
        mustBeVariable();
        mustHaveValue();
        return new EzgInt(var--);
    }

    /** Unary plus. */
    public EzgType uplus() {
        mustHaveValue();
        return new EzgInt(+var);
    }

    /** Unary minus. */
    public EzgType uminus() {
        mustHaveValue();
        return new EzgInt(-var);
    }

    public String typeName() {
        return "int";
    }

    public String toString() {
        mustHaveValue();
        return Integer.toString(var);
    }
}
```

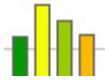


B.3.4 EzgFloat.java

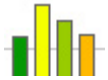
```
/**
 * Title:       EzgFloat.java
 * Description: Class to represent the float data type.
 * Modified:    2007-04-26
 */

package ezg.type;

public class EzgFloat extends EzgType {
    private float var;

    public EzgFloat(float var) {
        super();
        this.var = var;
        initialized = true;
    }

    public EzgFloat() {
        this(0);
        initialized = false;
    }

    public static float floatValue(EzgType b) {
        b.mustHaveValue();
        if (b instanceof EzgFloat)
            return ((EzgFloat)b).var;
        if (b instanceof EzgInt)
            return (float)((EzgInt)b).getVar();
        throw new RuntimeException("unexpected data type being cast to
float");
    }

    /** Converts EzgFloat[] to float[]. */
    public static float[] convertArray(EzgType[] src) {
        float[] dest = new float[src.length];
        for (int i = 0; i < dest.length; i++)
            dest[i] = ((EzgFloat)src[i]).getVar();
        return dest;
    }

    public EzgType type() {
        return new EzgFloat();
    }

    public EzgType copy() {
        mustHaveValue();
        return new EzgFloat(var);
    }

    public float getVar() {
        mustHaveValue();
        return var;
```



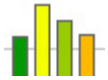
```
    }

    public EzgType[] getArray(int size) {
        return new EzgFloat[size];
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        mustBeVariable();
        var = floatValue(b);
        initialized = true;
        return this;
    }

    /** Assignment after addition. */
    public EzgType plasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var += floatValue(b);
        return this;
    }

    /** Assignment after subtraction. */
    public EzgType minasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var -= floatValue(b);
        return this;
    }

    /** Assignment after multiplication. */
    public EzgType mulasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var *= floatValue(b);
        return this;
    }

    /** Assignment after division. */
    public EzgType divasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var /= floatValue(b);
        return this;
    }

    /** Assignment after modulo. */
    public EzgType modasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        var %= floatValue(b);
        return this;
    }

    /** Equality. */
    public EzgType eq(EzgType b) {
        mustHaveValue();
```



```
        return new EzgBool(floatValue(b) == var);
    }

    /** Inequality. */
    public EzgType neq(EzgType b) {
        mustHaveValue();
        return new EzgBool(floatValue(b) != var);
    }

    /** Relational less than. */
    public EzgType lt(EzgType b) {
        mustHaveValue();
        return new EzgBool(var < floatValue(b));
    }

    /** Relational greater than. */
    public EzgType gt(EzgType b) {
        mustHaveValue();
        return new EzgBool(var > floatValue(b));
    }

    /** Relational less than or equal to. */
    public EzgType lteq(EzgType b) {
        mustHaveValue();
        return new EzgBool(var <= floatValue(b));
    }

    /** Relational greater than or equal to. */
    public EzgType gteq(EzgType b) {
        mustHaveValue();
        return new EzgBool(var >= floatValue(b));
    }

    /** Addition. */
    public EzgType plus(EzgType b) {
        mustHaveValue();
        return new EzgFloat(var + floatValue(b));
    }

    /** Subtraction. */
    public EzgType minus(EzgType b) {
        mustHaveValue();
        return new EzgFloat(var - floatValue(b));
    }

    /** Multiplication. */
    public EzgType mult(EzgType b) {
        mustHaveValue();
        return new EzgFloat(var * floatValue(b));
    }

    /** Division. */
    public EzgType div(EzgType b) {
        mustHaveValue();
        return new EzgFloat(var / floatValue(b));
    }
```

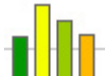


```
    /** Modulo. */
    public EzgType mod(EzgType b) {
        mustHaveValue();
        return new EzgFloat(var % floatValue(b));
    }

    /** Prefix increment. */
    public EzgType prefinc() {
        mustBeVariable();
        mustHaveValue();
        return new EzgFloat(++var);
    }

    /** Prefix decrement. */
    public EzgType prefdec() {
        mustBeVariable();
        mustHaveValue();
        return new EzgFloat(--var);
    }

    /** Postfix increment. */
    public EzgType postinc() {
        mustBeVariable();
        mustHaveValue();
        return new EzgFloat(var++);
    }

    /** Postfix decrement. */
    public EzgType postdec() {
        mustBeVariable();
        mustHaveValue();
        return new EzgFloat(var--);
    }

    /** Unary minus. */
    public EzgType uplus() {
        mustHaveValue();
        return new EzgFloat(+var);
    }

    /** Unary minus. */
    public EzgType uminus() {
        mustHaveValue();
        return new EzgFloat(-var);
    }

    public String typeName() {
        return "float";
    }

    public String toString() {
        mustHaveValue();
        return Float.toString(var);
    }
}
```

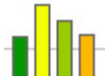


B.3.5 EzgString.java

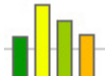
```
/**
 * Title:       EzgString.java
 * Description: Class to represent the string data type.
 * Modified:    2007-05-05
 */

package ezg.type;

public class EzgString extends EzgType {
    /** The system-dependent default line-separator character(s). */
    private static final String LS = System.getProperty("line.separator");

    private String var;

    private String scanString(String str) {
        String res = "", s;
        char ch;
        for (int i = 0; i < str.length(); i++) {
            ch = str.charAt(i);
            s = "" + ch;
            if ('\\' == ch) {
                try {
                    ch = str.charAt(++i);
                } catch (ArrayIndexOutOfBoundsException e) {
                    ch = 0;
                }
                if ('n' == ch)
                    s = LS;
                else if ('t' == ch)
                    s = "\t";
                else if ('"' == ch)
                    s = "\"";
                else if ('\\' == ch)
                    s = "\\";
                else
                    throw new RuntimeException("illegal escape character");
            }
            res += s;
        }
        return res;
    }

    public EzgString(String var) {
        super();
        this.var = null != var ? scanString(var) : null;
        initialized = true;
    }

    public EzgString() {
        this(null);
        initialized = false;
    }

    /** Converts EzgString[] to String[]. */
```



```
    public static String[] convertArray(EzgType[] src) {
        String[] dest = new String[src.length];
        for (int i = 0; i < dest.length; i++)
            dest[i] = ((EzgString)src[i]).getVar();
        return dest;
    }

    public EzgType type() {
        return new EzgString();
    }

    public EzgType copy() {
        mustHaveValue();
        return new EzgString(var);
    }

    public int getSize() {
        mustHaveValue();
        return var.length();
    }

    public String getVar() {
        mustHaveValue();
        return var;
    }

    public EzgType[] getArray(int size) {
        return new EzgString[size];
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        mustBeVariable();
        b.mustHaveValue();
        if (b instanceof EzgString) {
            var = b.toString();
            initialized = true;
            return this;
        }
        throw new RuntimeException("unexpected data type being assigned to "
+
                                   "string");
    }

    /** Assignment after concatenation. */
    public EzgType plasgn(EzgType b) {
        mustBeVariable();
        mustHaveValue();
        b.mustHaveValue();
        var += b.toString();
        initialized = true;
        return this;
    }

    /** Equality. */
    public EzgType eq(EzgType b) {
        mustHaveValue();
```



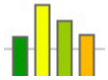
```
        b.mustHaveValue();
        if (b instanceof EzgString)
            return new EzgBool(var.equals(((EzgString)b).var));
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Inequality. */
    public EzgType neq(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        if (b instanceof EzgString)
            return new EzgBool(!var.equals(((EzgString)b).var));
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Relational less than. */
    public EzgType lt(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        if (b instanceof EzgString)
            return new EzgBool(var.compareTo(((EzgString)b).var) < 0);
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Relational greater than. */
    public EzgType gt(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        if (b instanceof EzgString)
            return new EzgBool(var.compareTo(((EzgString)b).var) > 0);
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Relational less than or equal to. */
    public EzgType lteq(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        if (b instanceof EzgString)
            return new EzgBool(var.compareTo(((EzgString)b).var) <= 0);
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Relational greater than or equal to. */
    public EzgType gteq(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        if (b instanceof EzgString)
```



```
            return new EzgBool(var.compareTo(((EzgString)b).var) >= 0);
        throw new RuntimeException("unexpected data type being compared to "
+
                                   "string");
    }

    /** Concatenation. */
    public EzgType plus(EzgType b) {
        mustHaveValue();
        b.mustHaveValue();
        return new EzgString(var + b.toString());
    }

    public EzgInt toEzgInt() {
        mustHaveValue();
        try {
            return new EzgInt(Integer.valueOf(var).intValue());
        } catch (NumberFormatException e) {
            throw new RuntimeException("cannot convert string to int");
        }
    }

    public EzgFloat toEzgFloat() {
        mustHaveValue();
        try {
            return new EzgFloat(Float.valueOf(var).floatValue());
        } catch (NumberFormatException e) {
            throw new RuntimeException("cannot convert string to float");
        }
    }

    public String typeName() {
        return "string";
    }

    public String toString() {
        mustHaveValue();
        return var;
    }
}
```

B.3.6 EzgVoid.java

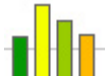
```
/**
 * Title:       EzgVoid.java
 * Description: Class to represent the void data type.
 * Modified:    2007-04-26
 */

package ezg.type;

public class EzgVoid extends EzgType {
    public EzgVoid() {
        super();
    }
```



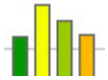
```
    public EzgType type() {
        return new EzgVoid();
    }

    public String typeName() {
        return "void";
    }
}
```

B.3.7 EzgArray.java

```
/**
 * Title:       EzgArray.java
 * Description: Class to represent the array data type.
 * Modified:    2007-05-06
 */

package ezg.type;

public class EzgArray extends EzgType {
    private EzgType baseType;
    private EzgType[] var;

    private EzgArray(EzgType baseType, int[] sizes, int level) {
        super();
        baseType.mustBeType();
        this.baseType = baseType;
        if (null != sizes) {
            var = baseType.getArray(sizes[level]);
            for (int i = 0; i < var.length; i++) {
                if (baseType instanceof EzgArray) {
                    EzgType bType = ((EzgArray)baseType).baseType.type();
                    var[i] = new EzgArray(bType, sizes, level + 1);
                } else {
                    var[i] = baseType.type();
                }
            }
            initialized = true;
        }
    }

    public EzgArray(EzgType baseType, int[] sizes) {
        this(baseType, sizes, 0);
    }

    public EzgArray(EzgType baseType) {
        this(baseType, null, 0);
    }

    public EzgArray(EzgType baseType, EzgType[] var) {
        this(baseType);
        this.var = var;
        initialized = true;
    }
```



```
    public EzgType type() {
        return new EzgArray(baseType.type());
    }

    public EzgType copy() {
        mustHaveValue();
        return this;
    }

    public int getSize() {
        mustHaveValue();
        return var.length;
    }

    public EzgType[] getVar() {
        mustHaveValue();
        return var;
    }

    public EzgType[] getArray(int size) {
        return new EzgArray[size];
    }

    public void setName(String name) {
        this.name = name;
        if (null != var) {
            for (int i = 0; i < var.length; i++)
                var[i].setName(name + "[" + i + "]");
        }
    }

    /** Assignment. */
    public EzgType assign(EzgType b) {
        mustBeVariable();
        b.mustHaveValue();
        if (b instanceof EzgArray &&
            baseType.typeName().equals(((EzgArray)b).baseType.typeName())) {
            var = ((EzgArray)b).var;
            setName(name);
            initialized = true;
            return this;
        }
        throw new RuntimeException("unexpected data type being assigned to "
+
                                   "array");
    }

    private EzgType index(int[] indexes, int level) {
        mustHaveValue();
        try {
            if (indexes.length - 1 == level)
                return var[indexes[level]];
            if (baseType instanceof EzgArray)
                return ((EzgArray)var[indexes[level]]).index(indexes,
level+1);
            throw new RuntimeException("array index list too big");
```



```
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new RuntimeException("array index out of bounds: " +
                                       indexes[level]);
        }
    }

    public EzgType index(int[] indexes) {
        return index(indexes, 0);
    }

    public String typeName() {
        return baseType.typeName() + "[]";
    }

    public String toString() {
        mustHaveValue();
        String str = "{";
        for (int i = 0; i < var.length; i++) {
            str += var[i].initialized ? var[i].toString() : "<null>";
            if (i < var.length - 1)
                str += ",";
        }
        str += "}";
        return str;
    }
}
```

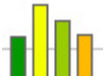
B.4 Testing Files

B.4.1 test-exe.bat

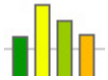
```
@echo off
setlocal
set CLASSPATH=..\classes;..\lib\antlr.jar
for %%i in (tests-in\*.ezg) do (
  for /F "tokens=2 delims=\." %%j in ("%%i") do (
    java -cp %CLASSPATH% ezg.EzgMain -t %%i > tests-out\%%j.txt
  )
)
endlocal
```

B.4.2 arithm.ezg

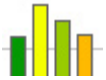
```
void main() {
    numbers();
    println();
    ints();
    println();
    floats();
    println();
    strings();
    println();
```



```
    mixed();
}

void numbers() {
    int i;
    float f;

    i = 1;      println(i);
    f = 1.;     println(f);
    f = 1.1;    println(f);
    f = 1.e1;   println(f);
    f = 1.e+1;  println(f);
    f = 1.e-1;  println(f);
    f = 1.E1;   println(f);
    f = 1.E+1;  println(f);
    f = 1.E-1;  println(f);
    f = 1.1e1;  println(f);
    f = 1.1e+1; println(f);
    f = 1.1e-1; println(f);
    f = 1.1E1;  println(f);
    f = 1.1E+1; println(f);
    f = 1.1E-1; println(f);
    f = 1e1;    println(f);
    f = 1e+1;   println(f);
    f = 1e-1;   println(f);
    f = 1E1;    println(f);
    f = 1E+1;   println(f);
    f = 1E-1;   println(f);
    f = .1;     println(f);
    f = .1e1;   println(f);
    f = .1e+1;  println(f);
    f = .1e-1;  println(f);
    f = .1E1;   println(f);
    f = .1E+1;  println(f);
    f = .1E-1;  println(f);
}

void ints() {
    int a = 56, b = 6, c = 1000;

    println(a + a);
    println(a + b + c);

    println(b - a);
    println(c - a - b);

    println(b * c);
    println(a * b * c);

    println(b / a);
    println(c / a / b);

    println(b % a);
    println(c % a % b);
}

void floats() {
```



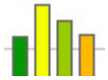
```
    float a = 5.3, b = 5e2, c = 0.2;

    println(a + b);
    println(a + b + c);

    println(a - b);
    println(a - b - c);

    println(a * b);
    println(a * b * c);

    println(a / b);
    println(b / a / c);

    println(a % b);
    println(a % b % c);
}

void strings() {
    string a = "data", b = "base";

    println(a + b);
    println(a + b + "s");
}

void mixed() {
    float f;
    string s;

    f = 1.2;
    println(6 + 1.2);
    println(1.2 + 6);
    println(f += 6);

    f = 1.2;
    println(6 - 1.2);
    println(1.2 - 6);
    println(f -= 6);

    f = 1.2;
    println(6 * 1.2);
    println(1.2 * 6);
    println(f *= 6);

    f = 1.2;
    println(6 / 1.2);
    println(1.2 / 6);
    println(f /= 6);

    f = 1.2;
    println(6 % 1.2);
    println(1.2 % 6);
    println(f %= 6);

    s = "str";
    println("str" + 6);
    println(s += 6);
```



```
    s = "str";
    println("str" + 1.2);
    println(s += 1.2);

    s = "str";
    println("str" + true);
    println(s += true);

    s = "str";
    println("str" + 6 + 1.2);
    println(s += "" + 6 + 1.2);

    s = "str";
    println("str" + 6 + 1.2 + false);
    println(s += "" + 6 + 1.2 + false);
}
```

B.4.3 arithm.txt

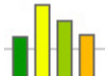
```
1
1.0
1.1
10.0
10.0
0.1
10.0
10.0
0.1
11.0
11.0
0.11
11.0
11.0
0.11
10.0
10.0
0.1
10.0
10.0
0.1
0.1
1.0
1.0
0.01
1.0
1.0
0.01

112
1062
-50
938
6000
336000
```



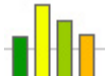
```
0
2
6
0

505.3
505.5
-494.7
-494.90002
2650.0
530.0
0.010600001
471.69812
5.3
0.10000011

database
databases

7.2
7.2
7.2
4.8
-4.8
-4.8
7.2000003
7.2000003
7.2000003
5.0
0.2
0.2
1.1999998
1.2
1.2
str6
str6
str1.2
str1.2
strtrue
strtrue
str61.2
str61.2
str61.2false
str61.2false
```

B.4.4 arrays.ezg

```
void main() {
    /* Passing an array. */
    int[] a = new int[6];
    for (int i = 0; i < size(a); i++)
        a[i] = i;
    println(a);
    foo(a);
    println(a);
```



```
    println();

    /* Returning an array. */
    float[] b = bar();
    println(b);
    println();

    fun1();
    fun2();
}

void foo(int[] a) {
    a = new int[2];
    a[0] = 8;
    a[1] = 9;
    println(a);
}

float[] bar() {
    float[] b = new float[5];
    for (int i = 0; i < size(b); i++)
        b[i] = i;
    return b;
}

void fun1() {
    int[][] a = new int[3][0];
    for (int i = 0; i < 3; i++) {
        a[i] = new int[i+1];
        for (int j = 0; j <= i; j++)
            a[i][j] = j;
    }
    int[][] b = new int[4][0];
    for (int i = 0; i < 4; i++) {
        b[i] = new int[i+1];
        for (int j = 0; j <= i; j++)
            b[i][j] = j;
    }
    println("a = " + a, "b = " + b);
    b = a;
    println("a = " + a, "b = " + b);
}

void fun2() {
    string[][][] c = new string[2][2][2];
    c[0][0][0] = "a";
    c[0][0][1] = "b";
    c[0][1][0] = "c";
    c[0][1][1] = "d";
    c[1][0][0] = "e";
    c[1][0][1] = "f";
    c[1][1][0] = "g";
    c[1][1][1] = "h";
    println("c = " + c);
    for (int i = 0; i < size(c); i++)
        for (int j = 0; j < size(c[i]); j++)
            for (int k = 0; k < size(c[i][j]); k++)
```



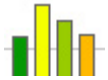
```
                print(" ", c[i][j][k]);
}
```

B.4.5 arrays.txt

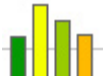
```
{0,1,2,3,4,5}
{8,9}
{0,1,2,3,4,5}

{0.0,1.0,2.0,3.0,4.0}

a = {{0},{0,1},{0,1,2}}
b = {{0},{0,1},{0,1,2},{0,1,2,3}}
a = {{0},{0,1},{0,1,2}}
b = {{0},{0,1},{0,1,2}}
c = {{{a,b},{c,d}},{{e,f},{g,h}}}
 a b c d e f g h
```

B.4.6 draw.ezg

```
void main() {
    canvas(700, 700);
    points();
    lines();
    rectangles();
    polygons();
    arcs();
    ovals();
    text();
    show();
}

void points() {
    for (int i = 1; i <= 25; i++) {
        color(i*5, 10, 200+i*2);
        stroke(i);
        point(i*20, i*2);
    }
}

void lines() {
    stroke(1);
    for (int i = 0; i < 25; i++) {
        color(200+i/2, 0, 0);
        line(700, i*20, 400, 700);
    }

    color(0,0,0);
    for (int i = 10; i > 0; i--) {
        stroke(10-i);
        line(600, 700-i*10, 700, 700-i*10);
    }
}
```



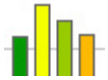
```
void rectangles() {
    color(0, 150, 80);
    stroke(2);
    for (int i = 0; i < 50; i=i+10)
        rect(i+50, i+50, 100-i*2, 100-i*2);

    for (int i = 0; i < 40; i=i+6) {
        color(100+i*3, 37+i*5, 71+i*4);
        fillRect(i+200, i+50, 120, 60);
    }
}

void polygons() {
    int n = 6;
    int[] x = new int[n];
    int[] y = new int[n];
    x[0] = 190; y[0] = 200;
    x[1] = 220; y[1] = 200;
    x[2] = 240; y[2] = 230;
    x[3] = 220; y[3] = 260;
    x[4] = 190; y[4] = 260;
    x[5] = 170; y[5] = 230;
    color(255, 0, 0);
    polygon(x, y, n);

    n = 4;
    x = new int[n];
    y = new int[n];
    x[0] = 185; y[0] = 205;
    x[1] = 200; y[1] = 245;
    x[2] = 225; y[2] = 180;
    x[3] = 200; y[3] = 225;
    color(45, 145, 245);
    fillPolygon(x, y, n);
}

void arcs() {
    color(0, 187, 187);
    stroke(1);
    arc(150, 350, 200, 200, 0, 135);

    fillArc(150, 350, 200, 200, 135, 135);

    color(0, 0, 248);
    stroke(5);
    arc(150, 600, 100, 200, 90, 75);

    color(120, 120, 120);
    stroke(1);
    fillArc(200, 600, 200, 50, 0, 145);
}

void ovals() {
    color(13, 77, 18);
    stroke(3);
    for (int i = 0; i < 50; i=i+10)
```



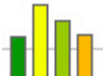
```
        oval(i+450, i+150, 100-i*2, 100-i*2);

    for (int i = 0; i < 40; i=i+6) {
        color(100+i*4, 37+i*3, 71+i*2);
        fillOval(i+350, i+260, 120, 60);
    }
}

void text() {
    string str = "Arial - 10";
    string fontName = "Arial";
    font(fontName, 0, 10);
    text(str, 0, 250);

    str = "Courier - 12";
    fontName = "Courier";
    font(fontName, 1, 12);
    text(str, 0, 300);

    str = "Georgia - 14";
    fontName = "Georgia";
    font(fontName, 2, 14);
    text(str, 0, 350);

    str = "Tahoma - 16";
    fontName = "Tahoma";
    font(fontName, 3, 16);
    text(str, 0, 400);

    str = "Times - 18";
    fontName = "Times New Roman";
    font(fontName, -1, 18);
    text(str, 0, 450);

    str = "My Font - 20";
    fontName = "My Font";
    font(fontName, 4, 20);
    text(str, 0, 500);

    color(100, 100, 100);
    string title = "Centered Text";
    int width = width(title);
    text(title, 700/2 - width/2, 200);
}
```

B.4.7 draw.txt

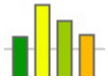
```
canvas(700,700)
color(5,10,202)
stroke(1)
point(20,2)
color(10,10,204)
stroke(2)
point(40,4)
color(15,10,206)
```



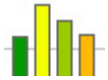
```
stroke(3)
point(60,6)
color(20,10,208)
stroke(4)
point(80,8)
color(25,10,210)
stroke(5)
point(100,10)
color(30,10,212)
stroke(6)
point(120,12)
color(35,10,214)
stroke(7)
point(140,14)
color(40,10,216)
stroke(8)
point(160,16)
color(45,10,218)
stroke(9)
point(180,18)
color(50,10,220)
stroke(10)
point(200,20)
color(55,10,222)
stroke(11)
point(220,22)
color(60,10,224)
stroke(12)
point(240,24)
color(65,10,226)
stroke(13)
point(260,26)
color(70,10,228)
stroke(14)
point(280,28)
color(75,10,230)
stroke(15)
point(300,30)
color(80,10,232)
stroke(16)
point(320,32)
color(85,10,234)
stroke(17)
point(340,34)
color(90,10,236)
stroke(18)
point(360,36)
color(95,10,238)
stroke(19)
point(380,38)
color(100,10,240)
stroke(20)
point(400,40)
color(105,10,242)
stroke(21)
point(420,42)
color(110,10,244)
```



```
stroke(22)
point(440,44)
color(115,10,246)
stroke(23)
point(460,46)
color(120,10,248)
stroke(24)
point(480,48)
color(125,10,250)
stroke(25)
point(500,50)
stroke(1)
color(200,0,0)
line(700,0,400,700)
color(200,0,0)
line(700,20,400,700)
color(201,0,0)
line(700,40,400,700)
color(201,0,0)
line(700,60,400,700)
color(202,0,0)
line(700,80,400,700)
color(202,0,0)
line(700,100,400,700)
color(203,0,0)
line(700,120,400,700)
color(203,0,0)
line(700,140,400,700)
color(204,0,0)
line(700,160,400,700)
color(204,0,0)
line(700,180,400,700)
color(205,0,0)
line(700,200,400,700)
color(205,0,0)
line(700,220,400,700)
color(206,0,0)
line(700,240,400,700)
color(206,0,0)
line(700,260,400,700)
color(207,0,0)
line(700,280,400,700)
color(207,0,0)
line(700,300,400,700)
color(208,0,0)
line(700,320,400,700)
color(208,0,0)
line(700,340,400,700)
color(209,0,0)
line(700,360,400,700)
color(209,0,0)
line(700,380,400,700)
color(210,0,0)
line(700,400,400,700)
color(210,0,0)
line(700,420,400,700)
color(211,0,0)
```



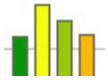
```
line(700,440,400,700)
color(211,0,0)
line(700,460,400,700)
color(212,0,0)
line(700,480,400,700)
color(0,0,0)
stroke(0)
line(600,600,700,600)
stroke(1)
line(600,610,700,610)
stroke(2)
line(600,620,700,620)
stroke(3)
line(600,630,700,630)
stroke(4)
line(600,640,700,640)
stroke(5)
line(600,650,700,650)
stroke(6)
line(600,660,700,660)
stroke(7)
line(600,670,700,670)
stroke(8)
line(600,680,700,680)
stroke(9)
line(600,690,700,690)
color(0,150,80)
stroke(2)
rect(50,50,100,100)
rect(60,60,80,80)
rect(70,70,60,60)
rect(80,80,40,40)
rect(90,90,20,20)
color(100,37,71)
fillRect(200,50,120,60)
color(118,67,95)
fillRect(206,56,120,60)
color(136,97,119)
fillRect(212,62,120,60)
color(154,127,143)
fillRect(218,68,120,60)
color(172,157,167)
fillRect(224,74,120,60)
color(190,187,191)
fillRect(230,80,120,60)
color(208,217,215)
fillRect(236,86,120,60)
color(255,0,0)
polygon({190,220,240,220,190,170},{200,200,230,260,260,230},6)
color(45,145,245)
fillPolygon({185,200,225,200},{205,245,180,225},4)
color(0,187,187)
stroke(1)
arc(150,350,200,200,0,135)
fillArc(150,350,200,200,135,135)
color(0,0,248)
stroke(5)
```



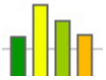
```
arc(150,600,100,200,90,75)
color(120,120,120)
stroke(1)
fillArc(200,600,200,50,0,145)
color(13,77,18)
stroke(3)
oval(450,150,100,100)
oval(460,160,80,80)
oval(470,170,60,60)
oval(480,180,40,40)
oval(490,190,20,20)
color(100,37,71)
fillOval(350,260,120,60)
color(124,55,83)
fillOval(356,266,120,60)
color(148,73,95)
fillOval(362,272,120,60)
color(172,91,107)
fillOval(368,278,120,60)
color(196,109,119)
fillOval(374,284,120,60)
color(220,127,131)
fillOval(380,290,120,60)
color(244,145,143)
fillOval(386,296,120,60)
font("Arial",0,10)
text("Arial - 10",0,250)
font("Courier",1,12)
text("Courier - 12",0,300)
font("Georgia",2,14)
text("Georgia - 14",0,350)
font("Tahoma",3,16)
text("Tahoma - 16",0,400)
font("Times New Roman",-1,18)
text("Times - 18",0,450)
font("My Font",4,20)
text("My Font - 20",0,500)
color(100,100,100)
text("Centered Text",288,200)
show()
```

B.4.8 equality.ezg

```
void main() {
    bools();
    println();
    ints();
    println();
    floats();
    println();
    strings();
    println();
    mixed();
}
```



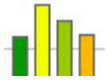
```
void bools() {
    bool a = true, b = false;

    println(a == a);
    println(b == b);
    println(a == b);
    println(b == a);

    println(a != a);
    println(b != b);
    println(a != b);
    println(b != a);
}

void ints() {
    int a = 256, b = -2;

    println(a == a);
    println(b == b);
    println(a == b);
    println(b == a);

    println(a != a);
    println(b != b);
    println(a != b);
    println(b != a);
}

void floats() {
    float a = 2.35, b = 23.5e-1, c = 4.6E-1, d = 4.6;

    println(a == a);
    println(a == b);
    println(b == d);
    println(c == d);
    println(d == c);

    println(a != a);
    println(a != b);
    println(b != d);
    println(c != d);
    println(d != c);
}

void strings() {
    string a = "a", b = "b";

    println(a == a);
    println(b == b);
    println(a == b);
    println(b == a);

    println(a != a);
    println(b != b);
    println(a != b);
    println(b != a);
}
```



```
void mixed() {
    println(7 == 7.0);
    println(7.0 == 7);
    println(7 == 8.3);
    println(8.3 == 7);

    println(7 != 7.0);
    println(7.0 != 7);
    println(7 != 8.3);
    println(8.3 != 7);
}
```

B.4.9 equality.txt

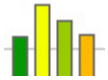
```
true
true
false
false
false
false
true
true

true
true
false
false
false
false
true
true

true
true
false
false
false
false
false
true
true
true

true
true
false
false
false
false
true
true

true
true
```



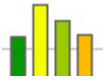
```
false
false
false
false
true
true
```

B.4.10 flow.ezg

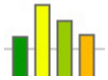
```
void main() {
    // Test if-else statements
    ctrl_if();
    println();

    // Test while-do loops
    ctrl_while();
    println();

    // Test for loops
    ctrl_for();
    println();

    // Test return statements
    println("ctrl_ret returned: " + ctrl_ret());
    println("ret_block returned: " + ret_block());
    println();

    // Test break statements
    println("ctrl_break returned: " + ctrl_break());
    println("break_block returned: " + break_block());
    println();

    //Test continue statements
    println("ctrl_cont returned: " + ctrl_cont());
    println("cont_block returned: " + cont_block());
}

/* If-Else */
void ctrl_if() {
    int a = 5, b = 6;

    if (a < b)
        println("a = " + a);

    if (a > b && 6 == b)
        println("a = " + a);
    else
        println("b = " + b);

    if (a < b || true)
        if (a != 5)
            println("b = " + b);
        else
            println("a = " + a);
```



```
    if (b == 0)
        println(0);
    else if (b == 1)
        println(1);
    else
        println("b = " + b);
}

/* While-Do */
void ctrl_while() {
    int i = 3, j = 2;

    while (i > 0) {
        println(i);
        i--;
    }
    while (++i < 3)
        println(i);
    while (i < 3)
        println(i);
    println("i = " + i);

    do {
        println(j);
        j++;
    } while (j < 2);
    do
        println(j);
    while (j-- > 0);
    do {
        println(j);
        j++;
    }
    while (j > 0);
    println("j = " + j);

}

/* For */
void ctrl_for() {
    int i;
    for (i = 3; i > 0;) {
        println(i);
        i--;
    }
    println("i = " + i);

    for (; i < 2; i++)
        println(i);
    println("i = " + i);

    for (; i < 4 ;) {
        println(i);
        i++;
    }
    println("i = " + i);
```



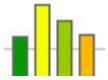
```
    for (;;) {
        if (i < 2)
            break;
        println(i);
        i--;
    }
    println("i = " + i);

    for (int i = 0; i < 3; i++)
        println(i);

    for (float i = 0, j = 1; i < 3; i++, j--)
        println("i = " + i,"j = " + j);
}

/* Return */
int ctrl_ret() {
    int a = 5;
    while (a > 0) {
        println("a = " + a);
        if (a == 3) {
            return a;
        }
        a--;
    }
    return a;
}

float ret_block() {
    float a = 10;
    return a + foo();
    println("Should never get here!");
}

int foo() {
    int b = 100;
    {
        int b = 0;
        return b;
    }
}

/* Break */
int ctrl_break() {
    int a = 4, b;
    while (a > 0) {
        println("a = " + a);
        b = 0;
        while (b < 5) {
            println("b = " + b);
            if (b == 1)
                break;
            b++;
        }
        a--;
    }
    return a;
```



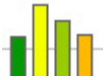
```
}

int break_block() {
    int a = 5;
    while (a > 0) {
        println("a = " + a);
        {
            if (a == 2)
                break;
        }
        a--;
    }
    return a;
}

/* Continue */
int ctrl_cont() {
    int a = 6, b = 0;
    while (a > 0) {
        println("a = " + --a);
        if (a >= 4)
            continue;
        b += 5;
        println("b = " + b);
    }
    return b;
}

int cont_block() {
    int a = 4, b = 0;
    while (a > 0) {
        println("a = " + --a);
        {
            if (a >= 2)
                continue;
            b += 5;
        }
        println("b = " + b);
    }
    return b;
}
```

B.4.11 flow.txt

```
a = 5
b = 6
a = 5
b = 6

3
2
1
1
2
i = 3
```



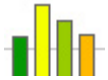
```
2
3
2
1
0
-1
j = 0

3
2
1
i = 0
0
1
i = 2
2
3
i = 4
4
3
2
i = 1
0
1
2
i = 0.0
j = 1.0
i = 1.0
j = 0.0
i = 2.0
j = -1.0

a = 5
a = 4
a = 3
ctrl_ret returned: 3
ret_block returned: 10.0

a = 4
b = 0
b = 1
a = 3
b = 0
b = 1
a = 2
b = 0
b = 1
a = 1
b = 0
b = 1
ctrl_break returned: 0
a = 5
a = 4
a = 3
a = 2
break_block returned: 2
```



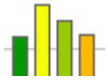
```
a = 5
a = 4
a = 3
b = 5
a = 2
b = 10
a = 1
b = 15
a = 0
b = 20
ctrl_cont returned: 20
a = 3
a = 2
a = 1
b = 5
a = 0
b = 10
cont_block returned: 10
```

B.4.12 internal.ezg

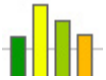
```
int a = 2;
float b = 2.2E-4;
bool c = true;
string d = "This is EZGraphs";
string[] e = new string[3];
int[][] f = new int[24][12];

void main() {
    prints();
    println();
    strToNr();
    println();
    datas();
    println();
    sizes();
    println();
    stringUtils();
    println();
    math();
}

void prints() {
    e[0] = "E";
    e[1] = "Z";
    e[2] = "G";

    println(a);
    println(b);
    println(c);
    println(d);
    println(e);
    println();
    println(a, b, c, d, e);
    println();
```



```
    print(a);
    print(b);
    print(c);
    print(d);
    print(e);
    println();
    print(a, b, c, d, e);
}

void strToNr() {
    string i = "1",
           f = "1.5";

    println(strToInt(i));
    println(strToFloat(f));
}

void datas() {
    string [][] data;

    data = data("input\\varsize.txt");
    println(data[0]);
    println(data[1]);
    println(data[2]);
    println(data[3]);
    println("data: " + data);

    println();

    // default delims, no header, not quoted
    data = data("input\\options.txt");
    println(data);

    // default delims, has header, not quoted
    data = data("input\\options.txt", true);
    println(data);

    // default delims, has header, quoted
    data = data("input\\options.txt", true, true);
    println(data);

    // pipe-delimited, no header, quoted
    data = data("input\\options.txt", "|", false, true);
    println(data);

    // star-delimited, has header, not quoted
    data = data("input\\options.txt", "*", true);
    println(data);
}

void sizes() {
    println(size(""));
    println(size(d));
    println(size(d + " v1.0"));
    println(size(e));
    println(size(f));
    println(size(f[0]));
```



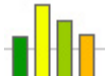
```
}

void stringUtils() {
    println(index(d, "r"));
    println(index(d, "s"));
    println(lastIndex(d, "f"));
    println(lastIndex(d, "s"));
    println(substring(d, 0, 6));
    println(substring(d, index(d, "p"), size(d)));
}

void math() {
    println("round(2) = " + round(2));
    println("round(2.3) = " + round(2.3));
    println("round(2.5) = " + round(2.5));
    println("round(2.7) = " + round(2.7));

    println("floor(2) = " + floor(2));
    println("floor(2.3) = " + floor(2.3));
    println("floor(2.5) = " + floor(2.5));
    println("floor(2.7) = " + floor(2.7));

    println("ceil(2) = " + ceil(2));
    println("ceil(2.3) = " + ceil(2.3));
    println("ceil(2.5) = " + ceil(2.5));
    println("ceil(2.7) = " + ceil(2.7));

    println("abc(2) = " + abs(2));
    println("abc(-2) = " + abs(2));
    println("abc(2.3) = " + abs(2.3));
    println("abc(-2.3) = " + abs(2.3));

    println("sqrt(25) = " + sqrt(25));
    println("sqrt(25.0) = " + sqrt(25.0));
    println("sqrt(100) = " + sqrt(100));
    println("sqrt(100.0) = " + sqrt(100.0));
    println("sqrt(2) = " + sqrt(2));
    println("sqrt(2.0) = " + sqrt(2.0));
    println("sqrt(10) = " + sqrt(10));
    println("sqrt(10.0) = " + sqrt(10.0));

    println("pow(2, 10) = " + pow(2, 10));
    println("pow(2.0, 10.0) = " + pow(2.0, 10.0));
    println("pow(2401, 1/4) = " + pow(2401, 1/4));
    println("pow(2401, 1.0/4) = " + pow(2401, 1.0/4));
    println("pow(2401.0, 1.0/4) = " + pow(2401.0, 1.0/4));

    println("exp(0) = " + exp(0));
    println("exp(0.0) = " + exp(0.0));
    println("exp(1) = " + exp(1));
    println("exp(1.0) = " + exp(1.0));

    println("log(1) = " + log(1));
    println("log(1.0) = " + log(1.0));
    println("log(E) = " + log(E));

    println("sin(0) = " + sin(0));
```



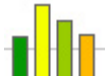
```
    println("sin(0.0) = " + sin(0.0));
    println("sin(90) = " + sin(90));
    println("sin(90.0) = " + sin(90.0));

    println("cos(0) = " + cos(0));
    println("cos(0.0) = " + cos(0.0));
    println("cos(90) = " + cos(90));
    println("cos(90.0) = " + cos(90.0));

    println("tan(0) = " + tan(0));
    println("tan(0.0) = " + tan(0.0));
    println("tan(45) = " + tan(45));
    println("tan(45.0) = " + tan(45.0));

    println("asin(0) = " + asin(0));
    println("asin(0.0) = " + asin(0.0));
    println("asin(1) = " + asin(1));
    println("asin(1.0) = " + asin(1.0));

    println("acos(0) = " + acos(0));
    println("acos(0.0) = " + acos(0.0));
    println("acos(1) = " + acos(1));
    println("acos(1.0) = " + acos(1.0));

    println("atan(0) = " + atan(0));
    println("atan(0.0) = " + atan(0.0));
    println("atan(1) = " + atan(1));
    println("atan(1.0) = " + atan(1.0));
}
```

B.4.13 internal.txt

```
2
2.2E-4
true
This is EZGraphs
{E,Z,G}

2
2.2E-4
true
This is EZGraphs
{E,Z,G}

22.2E-4trueThis is EZGraphs{E,Z,G}
22.2E-4trueThis is EZGraphs{E,Z,G}
1
1.5

{1,2,3,4}
{5,6}
{7,8,9}
{10,11,12,13,14}
data: {{1,2,3,4},{5,6},{7,8,9},{10,11,12,13,14}}
```



```
{{Nr,Name,Age,Height,DOB},{1|,"Jane,Doe",|,5.3|12/31/52},{2|"John,Smith"|,01/
01/42},{3|"Adam,|Brown"|23|6.0|02/01/84},{4|Mary,Stone|,40}}
{{1|,"Jane,Doe",|,5.3|12/31/52},{2|"John,Smith"|,01/01/42},{3|"Adam,|Brown"|2
3|6.0|02/01/84},{4|Mary,Stone|,40}}
{{1|,Jane
Doe,|,5.3|12/31/52},{2|"John,Smith"|,01/01/42},{3|"Adam,|Brown"|23|6.0|02/01/
84},{4|Mary,Stone|,40}}
{{Nr  Name  Age   Height      DOB},{1, "Jane Doe" , 5.3,12/31/52},{2,John
Smith, 01/01/42},{3,Adam |Brown,23,6.0,02/01/84},{4,Mary Stone, 40}}
{{1| "Jane Doe" | 5.3|12/31/52},{2|"John Smith"| 01/01/42},{3|"Adam
|Brown"|23|6.0|02/01/84},{4|Mary Stone| 40}}

0
16
21
3
24
12

11
3
-1
15
This i
phs

round(2) = 2
round(2.3) = 2
round(2.5) = 3
round(2.7) = 3
floor(2) = 2
floor(2.3) = 2
floor(2.5) = 2
floor(2.7) = 2
ceil(2) = 2
ceil(2.3) = 3
ceil(2.5) = 3
ceil(2.7) = 3
abc(2) = 2
abc(-2) = 2
abc(2.3) = 2.3
abc(-2.3) = 2.3
sqrt(25) = 5.0
sqrt(25.0) = 5.0
sqrt(100) = 10.0
sqrt(100.0) = 10.0
sqrt(2) = 1.4142135
sqrt(2.0) = 1.4142135
sqrt(10) = 3.1622777
sqrt(10.0) = 3.1622777
pow(2, 10) = 1024.0
pow(2.0, 10.0) = 1024.0
pow(2401, 1/4) = 1.0
pow(2401, 1.0/4) = 7.0
pow(2401.0, 1.0/4) = 7.0
exp(0) = 1.0
exp(0.0) = 1.0
```



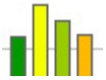
```
exp(1) = 2.7182817
exp(1.0) = 2.7182817
log(1) = 0.0
log(1.0) = 0.0
log(E) = 0.99999994
sin(0) = 0.0
sin(0.0) = 0.0
sin(90) = 1.0
sin(90.0) = 1.0
cos(0) = 1.0
cos(0.0) = 1.0
cos(90) = 6.123234E-17
cos(90.0) = 6.123234E-17
tan(0) = 0.0
tan(0.0) = 0.0
tan(45) = 1.0
tan(45.0) = 1.0
asin(0) = 0.0
asin(0.0) = 0.0
asin(1) = 90.0
asin(1.0) = 90.0
acos(0) = 90.0
acos(0.0) = 90.0
acos(1) = 0.0
acos(1.0) = 0.0
atan(0) = 0.0
atan(0.0) = 0.0
atan(1) = 45.0
atan(1.0) = 45.0
```

B.4.14 logical.ezg

```
void main() {
    bool a = true, b = false;

    println(a && a);
    println(b && b);
    println(a && b);
    println(b && a);

    println(a || a);
    println(b || b);
    println(a || b);
    println(b || a);

    println(!a);
    println(!b);
}
```

B.4.15 logical.txt

```
true
false
```



```
false
false
true
false
true
true
false
true
```

B.4.16 mixexpr.ezg

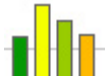
```
void main() {
    bool  a = true,
          b = false;
    float c = 5.3,
          d = 23.5e-1;
    int   e = 2,
          f = 156,
          g = 0;

    println(!b || a && (b && a));
    println((a || b) || b || a);
    println(!(b || a) && a);
    println((!b || a) && a);
    println((c != 53e-1) || (e <= e) && (g != g));

    println(f - e + g * c - (c + d) / 2);
    println(c * f - -c + g);
    println(f - c % e);
    println(++e / 3);
    println(e);
    println(--g + 1);
}
```

B.4.17 mixexpr.txt

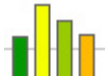
```
true
true
false
true
false
150.175
832.10004
154.7
1
3
0
```

B.4.18 relational.ezg

```
void main() {
    ints();
```



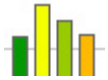
```
    println();
    floats();
    println();
    strings();
    println();
    mixed();
}

void ints() {
    int a = 5, b = 88;

    println(a < a);
    println(b < b);
    println(a < b);
    println(b < a);

    println(a > a);
    println(b > b);
    println(a > b);
    println(b > a);

    println(a <= a);
    println(b <= b);
    println(a <= b);
    println(b <= a);

    println(a >= a);
    println(b >= b);
    println(a >= b);
    println(b >= a);
}

void floats() {
    float a = 1099E-5, b = 2.3;

    println(a < a);
    println(b < b);
    println(a < b);
    println(b < a);

    println(a > a);
    println(b > b);
    println(a > b);
    println(b > a);

    println(a <= a);
    println(b <= b);
    println(a <= b);
    println(b <= a);

    println(a >= a);
    println(b >= b);
    println(a >= b);
    println(b >= a);
}

void strings() {
```



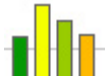
```
    string a = "abc", b = "bcd";

    println(a < a);
    println(b < b);
    println(a < b);
    println(b < a);

    println(a > a);
    println(b > b);
    println(a > b);
    println(b > a);

    println(a <= a);
    println(b <= b);
    println(a <= b);
    println(b <= a);

    println(a >= a);
    println(b >= b);
    println(a >= b);
    println(b >= a);
}

void mixed() {
    println(43 < 43.0);
    println(43.0 < 43);
    println(43 < 12.02);
    println(12.02 < 43);

    println(43 > 43.0);
    println(43.0 > 43);
    println(43 > 12.02);
    println(12.02 > 43);

    println(43 <= 43.0);
    println(43.0 <= 43);
    println(43 <= 12.02);
    println(12.02 <= 43);

    println(43 >= 43.0);
    println(43.0 >= 43);
    println(43 >= 12.02);
    println(12.02 >= 43);
}
```

B.4.19 relational.txt

```
false
false
true
false
false
false
false
true
```



```
true
true
true
false
true
true
false
true

false
false
true
false
false
false
false
true
true
true
true
false
true
true
false
true

false
false
true
false
false
false
false
true
true
true
true
false
true
true
false
true

false
false
false
true
false
false
true
false
true
true
false
true
true
true
```



```
true
false
```

B.4.20 scope.ezg

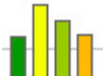
```
int i = 1;

void main() {
    println(i);
    int i = 2;
    println(i);
    {
        int i = 3;
        println(i);
        i++;
    }
    println(i);
    fun();

    for (int i = 0; i <= 0; i++)
        println(i);
}

void fun() {
    println(i);
}
```

B.4.21 scope.txt

```
1
2
3
2
1
0
```

B.4.22 string.ezg

```
void main() {
    allChars();
    println();
    escChars();
}

void allChars() {
    string a = "abcdefghiklmnopqrstuvwxyz";
    string b = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    string c = "`1234567890-=";
    string d = "~!@#$%^&*()_+";
    string e = "[]\\;',./";
    string f = "{}|:\"<>?";
```



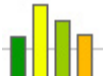
```
    println(a, b, c, d, e, f);
}

void escChars() {
    string a = "***\n***\t***\"***\\***";

    println(a);
}
```

B.4.23 string.txt

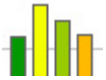
```
abcdefghiklmnopqrstuvwxyz
ABCDEFGHIKLMNOPQRSTUVWXYZ
`1234567890-=
~!@#$%^&*()_+
[]\;',./
{}|:"<>?

***
***   ***"***\***
```

B.4.24 transform.ezg

```
void main() {
    canvas(600, 600);
    stroke(2);

    rotate();
    reset();
    scale();
    reset();
    shear();
    reset();
    translate();

    show();
}

void rotate() {
    color(0, 0, 0);
    rect(100, 100, 100, 50);
    color(255, 0, 0);
    rotate(20.0);
    rect(100, 100, 100, 50);
    color(0, 255, 0);
    rotate(-35.0);
    rect(100, 100, 100, 50);
    push();
    reset();
    color(0, 0, 255);
    rotate(-25.0);
    rect(100, 100, 100, 50);
    pop();
```



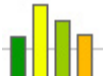
```
    color(255, 255, 0);
    rotate(45.0);
    oval(100, 100, 100, 50);
}

void scale() {
    color(0, 0, 0);
    rect(100, 500, 25, 75);
    color(255, 0, 0);
    scale(2.0 ,1.0);
    rect(100, 500, 25, 75);
    color(0, 255, 0);
    scale(1.0, 0.8);
    rect(100, 500, 25, 75);
    push();
    reset();
    color(0, 0, 255);
    scale(-1.0, 0.8);
    rect(-100, 500, 25, 75);
    pop();
    color(255, 255, 0);
    scale(1.0, 0.8);
    oval(100, 500, 25, 75);
}

void shear() {
    color(0, 0, 0);
    rect(500, 500, 20, 20);
    color(255, 0, 0);
    shear(-0.2, 0.0);
    rect(500, 500, 20, 20);
    color(0, 255, 0);
    shear(0.0, -0.2);
    rect(500, 500, 20, 20);
    push();
    reset();
    color(0, 0, 255);
    shear(-0.3, -0.3);
    rect(500, 500, 20, 20);
    pop();
    color(255, 255, 0);
    shear(0.1, 0.3);
    oval(500, 400, 20, 20);
}

void translate() {
    color(0, 0, 0);
    rect(300, 50, 50, 50);
    color(255, 0, 0);
    translate(80.0, 80.0);
    rect(300, 50, 50, 50);
    color(0, 255, 0);
    translate(-60.0, -40.0);
    rect(300, 50, 50, 50);
    push();
    reset();
    color(0, 0, 255);
```



```
    translate(100.0, 0.0);
    rect(300, 50, 50, 50);
    pop();
    color(255, 255, 0);
    oval(300, 50, 50, 50);
}
```

B.4.25 transform.txt

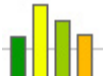
```
canvas(600,600)
stroke(2)
color(0,0,0)
rect(100,100,100,50)
color(255,0,0)
rotate(20.0)
rect(100,100,100,50)
color(0,255,0)
rotate(-35.0)
rect(100,100,100,50)
push()
reset()
color(0,0,255)
rotate(-25.0)
rect(100,100,100,50)
pop()
color(255,255,0)
rotate(45.0)
oval(100,100,100,50)
reset()
color(0,0,0)
rect(100,500,25,75)
color(255,0,0)
scale(2.0,1.0)
rect(100,500,25,75)
color(0,255,0)
scale(1.0,0.8)
rect(100,500,25,75)
push()
reset()
color(0,0,255)
scale(-1.0,0.8)
rect(-100,500,25,75)
pop()
color(255,255,0)
scale(1.0,0.8)
oval(100,500,25,75)
reset()
color(0,0,0)
rect(500,500,20,20)
color(255,0,0)
shear(-0.2,0.0)
rect(500,500,20,20)
color(0,255,0)
shear(0.0,-0.2)
rect(500,500,20,20)
```



```
push()
reset()
color(0,0,255)
shear(-0.3,-0.3)
rect(500,500,20,20)
pop()
color(255,255,0)
shear(0.1,0.3)
oval(500,400,20,20)
reset()
color(0,0,0)
rect(300,50,50,50)
color(255,0,0)
translate(80.0,80.0)
rect(300,50,50,50)
color(0,255,0)
translate(-60.0,-40.0)
rect(300,50,50,50)
push()
reset()
color(0,0,255)
translate(100.0,0.0)
rect(300,50,50,50)
pop()
color(255,255,0)
oval(300,50,50,50)
show()
```

B.4.26 unary.ezg

```
void main() {
    ints();
    println();
    floats();
}

void ints() {
    int a = 10;

    println(-a);
    println(-(-a));
    println(--a);
    println(++a);
    println(a--);
    println(+a);
    println(a++);
    println(+(+a));
}

void floats() {
    float a = 3.6;

    println(-a);
    println(-(-a));
    println(++a);
```



```
    println(--a);
    println(a++);
    println(+a);
    println(a--);
    println(+(+a));
}
```

B.4.27 unary.txt

```
-10
10
9
10
10
9
9
10

-3.6
3.6
4.6
3.6
3.6
4.6
4.6
3.6
```