Mirage

A Graphical Sketching Language

Columbia University
COMS W4115 Programming Languages and Translators
Fall 2006

Prof. Stephen Edwards

Team Members:

Abhilash I	**ai2160@columbia.edu**
Ming Liao	**ml2288@columbia.edu**
Nalini Vasudevan	**nv2144@columbia.edu**
Peili Zhang	**pz2128@columbia.edu**

Table of Contents

Chapter 1

Introduction

1.1 Overview

Motivation:

The main objective of our project is to develop a simple, elegant and powerful graphical language that not only does assist the learning process for the beginners, but also help the skilled programmers to create preliminary and intermediate level graphics with an interpreter, which makes programming much efficient in certain graphical applications fields.

Mirage was motivated by the Logo Programming Language, and the idea was to merge Logo and C to provide a C like syntax with Logo drawing features.

Description:

The Mirage programming language is designed to be a graphic-specific, functional, and translated programming language, with the target language being Java. The graphic output of the Mirage is produced from the *Java.swing* package since it provides sufficient utilities to generate any 2D graphics.

Similar to Logo programming language, Mirage is easy to read, write and learn. It is used mainly for educational purpose, especially for teaching and inspiring programming beginners. It can also enable more advanced programmers to create fantastic “turtle curves” using recursive function calls. Moreover, Mirage graphics is useful for example in Lindenmayer system for generating fractals, for example, the dragon curve or hilbert curve. Not only can it create "Mirage graphics", but also it provides useful facilities for handling image I/O. Moreover, Mirage can be used to teach most basic computer science concepts.

Mirage also supports graphcial animations. For example, the motion of move forward 50 pixels or turn certain angle clockwise depend on the defined speed. In general, from the small building blocks programmers are able to build more complex geometries and any graphics.

1.2. Language Features

The “Pen” is a pre-defined object, which is used to draw geometries and graphics.

Primitive Data Types:

There is only one basic numerical data type in Mirage, namely the integer. It is used for specifying the moving distances, turning angles of the Pen and the curve's offset percentage, as well as for doing more advanced calculations to enable programmers to create complex geometries or graphics.

The "void" data type is used for specifying the type of the function when it is not returning anything.

Key words:

All the key words are written in the lower case. For example, "up", "down", "return", "break", and "while", etc… …

Comparative and Mathematical Operators:

Operators are necessary for condition checking and calculations. Mirage have all the essential comparative and mathematical operators, for examples, = =, ! =, >=, =<, +, -, *, /, <, >, =, ( ), &&, and so on.

Control Statements:

To control program's flow and carry out certain algorithms, we will implement and support the following control statements:

- while loop: for example, we can use a while loop to draw certain geometries repeatedly.
- for loop: for example, we can use a for loop to repeat a line certain times.
- if/else condition: for example, depends on different conditions, we can use if/else to draw different geometries.

Other Features:

- Each procedure or function encloses a group of statements by '{' and '}'. Each statement is separated by a ';'.
- Comments start with '/*' and ends with '*/'.
- Motion of PEN include "fwd", "bwd", "rgt", "lft", etc;
- Pen State include up, down, reset;
- Pen color can be any color, for example, one can use color code <R, G, B> to specify certain color.
- Turn eraser on and off can be set using the "eraseon" and "eraseoff" keywords.

Syntax:

The syntax of Mirage resembles Logo but with some modifications. For example, some basic syntax will be as follows:

```
int variable_name;      /* all variable are Integer type:
                        It is a local variable if defined in a procedure
                        or a global variable if passed to any procedure */


up; or down;                          /* indicate the state of PEN */

color <255, 0, 0>                     /* color code specifies the PEN color */

thick t;                              /* specify the thickness of PEN where t
                                      range in [1, 10] */

speed sp;                             /* specify the speed of PEN where sp
                                      range in [0, 1000] */

fwd 40; or bwd 30;                    /* specify the moving direction and
                                      distance of PEN */

rgt 20; or lft 5;                     /* specify turning direction and
                                      angles of PEN (right or left) */
```

Other key words includes curve, return, break, open, load, save, close, reset, etc…

```
while(condition){

      /* statement */;
      ... ...

}

for(i=0; i<10; i++){

      /* statement */;
      ... ...
}

if(condition){
      /* statement */;
      ... ...
}
else if(condition){
      /* statement */;
      ... ...
}
else{
      /* statement */;
      ... ...
}
```

```
void procedure_name(int a, int b){

      /* statement */;
      int c = a + b;
      ... ...
}

main(){

      /* statements… */;
      int a, b, c;
      a = 6;
      b = 7;
      procedure1 (a, b);
      c = 4;
      ... ...

}

/* overall structure in a program might look like: */

procedure1()
{
      ... ...
}

procedure2()
{
      ... ...
}

/* … statements … */;

main()
{
      ... ...

      /* statements */

      ... ...

}
```

1.3. Design Goals

Mirage: *A simple, intuitive, efficient, flexible, robust, high-performance, interpreted, architecture neutral, portable language.*

Simplicity and Intuitive:

Mirage uses simple, consistent and intuitive syntax that are easy for beginners to learn and

understand as well as allows experienced users to read and write quickly. Code definitions in Mirage are structured and blocked, which increases the readability of the language.

Efficiency and Flexibility:

Mirage can be implemented in a very efficient way with minimal system resources utilized. Since it is a simple graphic/geometry domain-specific language, it has a small set of commands and operators. Programs written in Mirage can be compiled very quickly. On the other hand, Mirage can also provide users flexibility for designing any kind of geometries or graphs. Moreover, Mirage employs flow control mechanisms such as if/else conditional checking and while and for loops.

Robust and High-performance

Not only does Mirage enable programmers to create arbitrary geometries or fantastic graphics, it can also be used to help programmers to simulate drawing in a Polar Coordinate System. Provided ability to perform various calculations, a programmer can create complicated graphics. Errors in the Mirage code are detected at compile time and guarantee the compiled Mirage code to be accurate.

Portability:

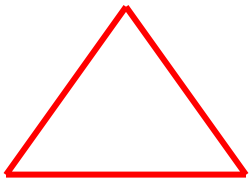
Since we use ANTLR as the syntax recognizer, interpreter will accept Mirage code as an input and generate the corresponding Java code. This source code operates on Java Virtual Machine, thus all Mirage code can be interpreted and executed on any machine that have a Java Runtime Environment. Users are able to create any geometry or graphics in the same way on a Windows platform as on a Unix platform. Thus, Mirage is a platform independent and portable language.

1.4. Sample Programs

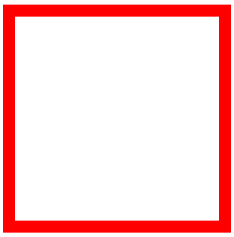
Three sample Mirage programs being traced shown below, which will draw a trianle, a square, and an arbitary shape. Althought Mirage code looks very simple, but one can group sets of instructions, employ recursive algorithms to create more complex graphics. Key words are usually to be written in upper case , while user-defined variables and function names are written in lower case.

```
/* Program to draw a triangle */

main()
{
      down;                   /*ready to draw*/
```

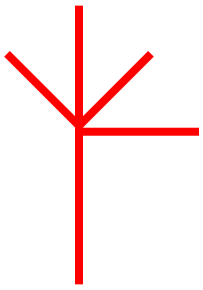
```
      color <255, 0, 0>;     /* set color to red */
      thick 2;               /* set pen thickness to 2 */
      speed 5;               /* set pen speed to 5 */

      rgt 30;                /* turn the pen by 30 degree */
      fwd 40;                /* draw a line by 40 pixel */

      rgt 120;               /* turn the pen by 120 degree */
      fwd 40;                /* draw a line by 40 pixel */

      rgt 120;               /* turn the pen by 120 degree */
      fwd 40;                /* draw a line by 40 pixel */


      up;                    /* stop to draw */

}
```



```
/*program to draw a square*/

main ()
{
      int i;      /* declare the variable i */

      down;       /* ready to draw */

      color <255, 0, 0>;     /* set color of the pen to red */
      thick 5;               /* set pen thickness to 5 */
      speed 3;               /* set pen speed to 3 */

      for (i=1; i<=4; i++)   /* repeat 4 times */
      {
            fwd 40;          /* draw a line by 40 pixel */
            rgt 90;          /* turn the pen by 90 degree */
      }

      up;               /* stop to draw */
}
```



```
/* Program to draw an arbitrary shape */
main()
{
      color <255, 0, 0>;  /* set color of the pen to red */
      thick 3;      /* set pen thickness to 3 */
      speed 3;      /* set pen speed to 3 */

      down;         /* ready to draw */
      fwd 40;       /* draw a line by 40 pixel */



      lft 45;       /* turn the pen to left by 45 degree  */
      fwd 20;     /* draw a line by 20 pixel */
      up;         /* stop to draw */

      rgt 45;     /* turn the pen to right by 45 degree  */
      fwd 20;     /* draw a line by 20 pixel */
      up;         /* stop to draw */

      rgt 90;      /* turn the pen to right by 90 degree  */
      fwd 30;      /* draw a line by 20 pixel */
      up;          /* stop to draw */

      fwd 30;      /* draw a line by 30 pixel */

      up;          /* stop to draw */

}
```



Chapter 2

Language Tutorial

In this section, we present three simple examples to demonstrate the basics of implementing a Mirage program. For a more detailed explanation regarding the language syntax, please consult chapter 3 --- the Mirage language reference manual.

2.1 Program Flow

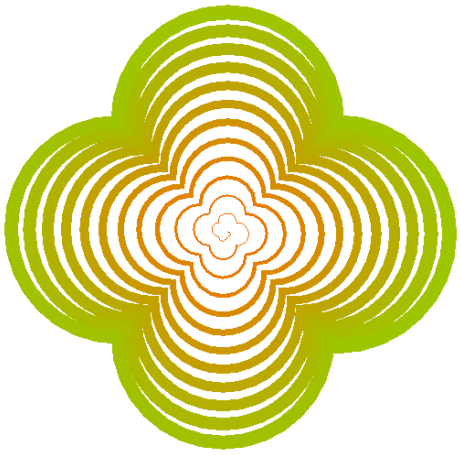
Mirage programs usually starts with several user-defined void function definitions to create the graphics. The main function definition is placed at the end of each program. Mirage does not support global declaration of variables. The scope of a variable is the context within which it is defined, thus any variable declared within a function is only local to that function. Variables can be declared only within functions and passed to other functions.

The programmer can start a new scope any time when they create a statement block within { }, such as in iterative statements or if/else statements. However, functions cannot be defined within functions.

2.2 Sample Programs

Here we show a simple Mirage program using curve, color and thickness statements, and a for loop to create a flower.

```
/* flower.mrg */

main()
{
      int a=5, c=90, i=1;

/* Define the shape of line: Half Cycle */

      curve <90,0>;

      for(i=0; i<50; i=i+1)
      {
            color <255-2*i,100+2*i,0>;
            thick i/3;
            rgt c;
            fwd a*i;
      }

      save "flower.jpg";

}
```



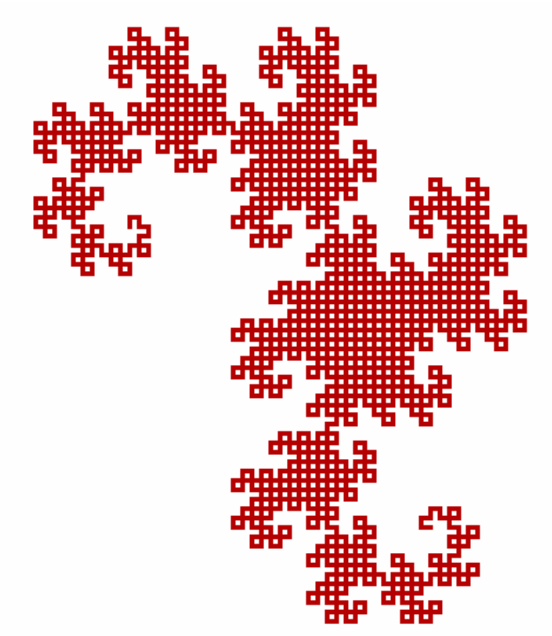
flower.jpg

Users can create Dragon curves using dragon algorithm. A dragon curve is a recursive nonintersecting curve whose name derives from its resemblance to a certain mythical creature.

In the main function, Dragon.mrg first define the dark red color <180, 0, 0>, and then calls dragon function with 3 values (11, 90, 6) to draw the graph. Finally, Dragon.mrg saves the graph as “dragon.jpg”.

In dragon function, depends on the value of variable ‘n’, dragon calls itself recursively:
if n<1, then forward PEN by distant ‘h’; otherwise it calls dragon function recursively and decrement value of ‘n’ by 1 and turn PEN clockwise by degree ‘a’ .

```
/* Dragon Curve */

void dragon(int n, int a, int h)
{
      if(n < 1)
      {
            fwd h;
            return;
      }
      dragon (n - 1, 90, h);
      rgt a;
      dragon (n - 1, -90, h);
}

main()
{
      color <180, 0, 0>;
      dragon( 11, 90, 6);
      save "dragon.jpg";
}
```



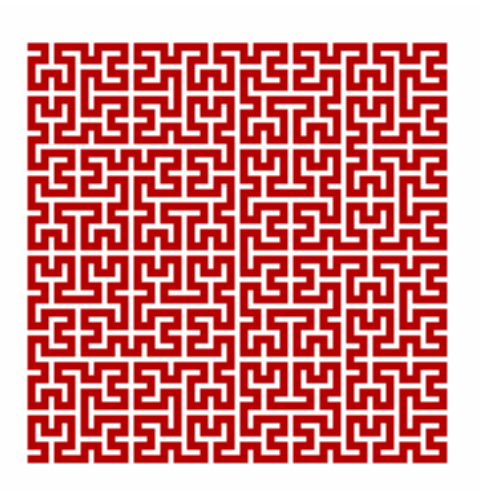
==========

Users can create Hilbert curves using hilbert algorithm. The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2×2, 4×4, 8×8, 16×16, or any other power of 2.

In the main function, hilbert.mrg first define the dark red color <180, 0, 0>, and then calls dragon function with 3 values (5, 90, 5) to draw the graph. Finally, hilbert.mrg saves the graph as “hilbert.jpg”.

The depth of recursion in this example is dependent on the value of ‘n’. In hilb function, depends on the value of variable ‘n’, hilb calls itself recursively:
if n is not equal to 0, then turn the PEN clockwise or counter clockwise by degree ‘a’, call hilb funcion recursively and then forward PEN by distant ‘h’.

```
/* hilbert Curve */

void hilb (int n, int a, int h)
{
      if (n == 0)
            return;

      rgt a;
      hilb  (n - 1, -a ,h);
      fwd h;
      lft a;
      hilb (n - 1, a, h);
      fwd h;
      hilb (n - 1, a, h);
      lft a;
      fwd h;
      hilb (n - 1, -a ,h);
      rgt a;
}

main()
{
      color <180, 0, 0>;
      hilb( 5, 90, 5);
      save "hilbert.jpg";
}
```



Please refer other program examples and detailed descriptions in Chapter 3 Language Reference Manual --- part 5.7 Graphics Drawing Statements and part 8 Sample Mirage Programs.

Chapter 3

Language Reference Manual

3.1 Lexical Conventions

Tokens

Mirage has six types of tokes, namely, identifiers, keywords, constants, strings, mathematical and comparative operators, and other separators. Blanks, tabs, newlines, and comments are ignored except as they are used to separate tokens. Some white space are required to separate otherwise adjacent identifiers, keywords, constants, strings, and operators.

If the input stream is parsed into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

Comments

Comments in Mirage, either a single line comment or multi-line comments, begin with "/*" and end with "*/".

Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore '_' counts as a letter. Mirage is case sensitive; the upper and lower case letters are treated differently, so the identifier 'aVar' and 'Avar' are considered to be distinct identifiers. There is no limit on the length of identifiers.

Keywords

The following identifiers are reserved as keywords within the Mirage language, and may not be used as any variable names:

if	else	while	for	return	break
main	void	int			
up	down	fwd	bwd	lft	rgt
color	thick	speed	reset		
load	save	eraseon	eraseoff		

Constants

Constants in the Mirage language are integers and strings.

Integer

An integer consists of a sequence of one or more consecutive digits.

Strings

A string literal is a sequence of characters surrounded by double quotes, for example, “flower.jpg” String literals do not contain newline or double-quote characters.

Other Kinds of Tokens
Each token will be described in detail in the following sections.

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

3.2 Types

Mirage supports three fundamental types: integer, string and void. Variables are bound to an int type during their declarations.

- int: is represented by 32-bit integers;
- string: image filenames are lists of characters.
- void: used for functions that do not return any values;

Mirage does not use any type casting since all variables are 16-bit integers.

3.3 Expressions

Expressions listed below are in the order of precedence from high to low. Within each subsection, the operators have the same precedence.

Primary Expressions

Primary expressions include identifiers, constants, integers, string, expressions contained within () group left to right, and function calls.

Identifiers

An identifier is a primary expression. Identifiers evaluate to the value with specified type at declaration.

Constant

An integer is a primary expression with the type int (an image filename within double quotes is a string).

Integers, Strings

As described above, they are evaluated to the result of expression with the type determined by the interpreter.

(expression)

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. Parentheses are used to ensure the proper precedence.

Primary-expression < expression-list >

The angle bracket groups a comma-separated list of parameters, which is used to specify RGB color values.

Primary-expression (expression-list)

Function calls are primary expressions. Functions are invoked by the function name followed by an optional comma-separated list of parameters contained within ( ). The parentheses are required. The type of the function is either void or the returned type by the function. All argument passed in Mirage are strictly by values, that is, a copy is made for each actual parameter in calling a function. Mirage allows recursive function calls.

Arithmetic Expressions

Mathematical and Comparative Operators are necessary for condition checking and calculations. Mirage supports most essential comparative and mathematical expressions. These expressions take primary expressions of type int operands.

Unary Operators

- expression

Unary operators - associate from right-to-left. The result is the negative of the operand and has the same type. The type of the operand must be int.

Binary Operators

- Multiplicative Operators

*expression * expression*
expression / expression
expression % expression

The multiplicative operators * (multiplication), / (division) and % associate from left-to-right. The operands of *, / and % (modulus) must be of the type int. / yields the quotient of the division of the first operand by the second; if the second operand is 0, the result is undefined.

- Additive Operators

expression + expression
expression - expression

The additive operators + (summation) and - (subtraction) associate from left-to-right. The operands of + and - must be of the type int.

- Relational Operators

expression < expression
expression > expression
expression <= expression
expression >= expression

The relational operators < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to) associate from left-to-right. The operators accept arithmetic expressions and primary expressions that evaluate to numeric values as operands.

All relational operators yield 0 if the specified relation is false and 1 if it is true. The operand of each operator must be of the type int.

- Equality Operators

expression == expression
expression != *expression*

The = = (equal to) and the != (not equal to) have lower precedence than the relational operators. For example, a<b = = a<c is 1 when a<b and c<d have the same truth-value. All yield 0 if the specified equality is false and 1 if it is true.

- Logical Operators

expression && expression

The && operator associate from left-to-right. It returns 1 if both its operands are not equal to zero, 0 otherwise. The second operands of && is not evaluated if the first operand is 0.

expression || expression

The || operator associate from left-to-right. It returns 1 if either of its operands is not equal to zero, 0 otherwise. The second operands of || is not evaluated if the first operand is non-zero.

- Assignment Operator

expression = expression

Assignment operator = associate from right-to-left. The left operand must be of type int. The right operand must be of type int, corresponding to the type of its left operand.

Other Operators

Expressions separated by a comma ‘,’ is evaluated left-to-right. Semicolon ‘;’ is used to terminate a statement. Curly brackets “{}” are used for grouping a block of statements in a function.

Operator Precedence and Associativity

All essential comparative and mathematical operators are listed below from the highest to the lowest precedence:

Operators	**Associativity**
( )	left to right
* / %	left to right
+ -	left to right
< <= > >=	left to right
= = !=	left to right
&&	left to right
||	left to right
=	right to left
,	left to right

3.4. Declarations

Declarations are used within function definitions to specify the interpretation that Mirage gives to each identifier. They do not reserve storage associated with the identifier. Declarations that reserve storage are definitions.

Variable Declaration:

vardeclaration:
typespecifier declarator-list ;

Type Specifiers

- *int*
- *void*
- *main*

Declarators

The declarator-list appearing in a variable declaration is a comma-separated sequence of declarators. The declarators in the declarator-list contain the identifiers being declared.

Declarator-list:
declarator
declarator declarator-list

declarator:
identifier

declarator declarator-list:
identifier

identifier:
int

Function Definitions

Functions must be defined when they are declared. A function definition must be an external declaration and cannot be inside any other function.

funcdefinition:
typespecifier identifier(parameter-list)
{
vardeclaration-list

statement-list
}

parameter-list:
typespecifier identifier
typespecifier identifier parameter-list

vardeclaration-list:
vardeclaration
vardeclaration vardeclaration-list

statement-list:
statement
statement statement-list

3.5 Statements

Each statement represents a command in Mirage and is always terminated with a semicolon ';'. Statements are either an expression or one or more statements separated by semicolons. Program flow can be modified with conditional and iterative statements or with function calls. Except as indicated, statements are executed in sequence.

Expression statements

Most statements are expression statements, which have the form

expression **;**

Usually expression statements are assignments or function calls.

assign-statement:
typespecifier identifier = expression;

Identifiers can be assigned a value using the assignment operator =. An identifier must be declared in a separate statement before it can be used in an assignment statement. The identifier appears on the left-hand side and an expression appears on the right-hand side. The type of the expression must match the type assigned to the identifier at declaration.

funccall-statement:
identifier (expression-list) ;

expression-list:
expression
expression expression-list

Compound statements

Several statements can be used where one is expected, the compound statement is provided within { }.

compond-statement:
{
statement-list
}

statement-list:
statement
statement statement-list

Conditional statements

The two forms of the conditional statement are

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

If statements allow the programmer to choose, at runtime, which set of commands will be execute. In both versions the expression is evaluated to be either true or false. If it is nonzero (true), the first substatement is executed. In the second case the second substatement is executed if the expression is 0. As usual the ‘‘else’’ ambiguity is resolved by connecting an else with the last encountered elseless if.

Iterative Statements

Iterative Statements include two kinds of statements: while and for statement.

while-statement:
while (expression) statement

The while statement will execute its substatements as long as the expression evaluates to 1 (true). The test takes place before each execution of the statement.

The for statement allows for iteration over a range of numbers.

for-statement:
for (expression1; expression2; expression3) statement

- expression1: initialization for the loop;
- expression2: a test made before each iteration, so that the loop is exited when the expression becomes 0;

- expression3: an increamentation is performed after each iteration.

Break statements

break-statement:
break ;

The break statement causes the termination of the enclosing while and for. The control passes to the statement following the terminated statement.

Return Statements

return-statement:
return ;
return (expression);

The return statement in a function will cause program control to return to the caller. An optional expression following the return keyword will cause the function to return the value of the expression to the caller. If required, the expression is converted, as if by assignment, to the type of the function in which it appears. Flowing off the end of a function is equivalent to a return with no returned value.

Graphics Drawing Statements

Mirage provides traditional turtle graphics. The graphic window is defined to be 800*600. Positive X is to the right; positive Y is down. Angles are measured in degrees counterclockwise from the positive X axis. The starting position is at the center of the window (at position (300, 300)). If the programmer doesn't move the pen from the starting position before his actual drawing, any graphics is drawn at position (300, 300).

- Colors

Mirage provides user-settable RGB colors. How many are available depends on the hardware and operating system environment. Mirage begins with a white background and black pen. Programmers can change the background and pen colors before drawing.

color <R, G, B>;

Color follows the RGB color model, which is enclosed by "< >" with three numbers listed inside ranging from 0 to 255 to represent Red, Green, and Blue. For example, color <255, 255, 0> represents color red. Color is used to color any line or arc. R, G, B can also be expressions.

- Graphics Motion

fwd dist;

moves the pen forward, in the direction that it's facing, by the specified distance (units are in pixels).

bwd dist;

moves the pen backward, for example, exactly opposite to the direction that it's facing, by the specified distance (units are in pixels).

lft degrees;

turns the pen counterclockwise by the specified angle, measured in degrees (1/360 of a circle).

rgt degrees;

turns the pen clockwise by the specified angle, measured in degrees (1/360 of a circle).

- Pen State

up;

lift the pen up, so the pen won't be able draw any graphics

down;

put the pen down, so the pen will be able to draw any graphics.

thick val;

defines the thickness of the pen in the integer range [1, 10]. val can be an expression.

speed val;

defines the speed of the drawing pen in the integer range [0, 1000]. Speed is between [0, 1000] (where 0 is the slowest, 1000 is fastest).

reset;

get the pen back to the center of the graphic window (300, 300).

eraseon;
eraseoff;

defines the ERASER object. eraseon turns the eraser on, for example, forward 30 with eraseon will erase a line of the length 30 with the background color, and with eraseoff will switch back to the PEN state.

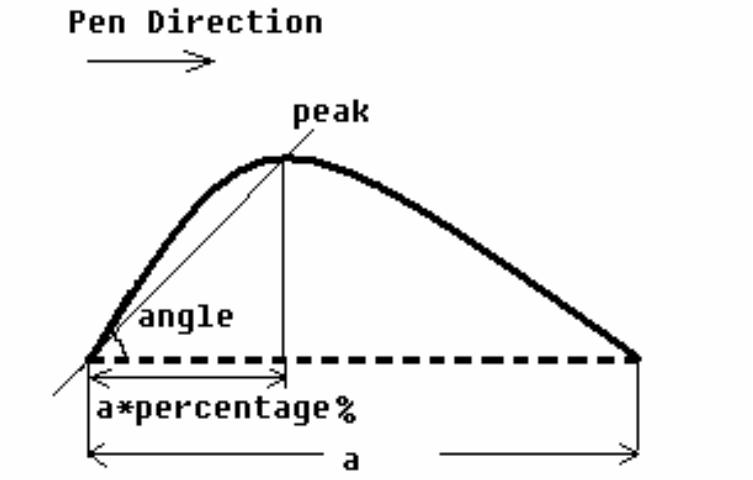
Changing the thickness, speed, or color in one function is reflected in another function. The pen movement is also globally maintained.

- Graphic Shape

curve <angle, offset in percentage>;

defines the line type: a straight line, a curve, or a half cycle. It is considered as one of the attributes of the pen. After defining the shape of the line (e.g. a straight line, a curve or a half cycle), each line will be drawn with the same shape, so we don't need to define it at each time.

As shown in the following figure Curve Definition, the degree of the angle for the curve is between [-90, 90]: a positive value indicates a counter-clockwise rotation, while a negative value indicates a clockwise rotation. The percentage represents the related location of the top point in the curve. The percentage is between [0, 100].



Curve Definition

In general, there are three types of shapes:

1. curve <0, any percentage number>: the shape is a straight line,.
2. curve <90, any percentage number> or curve <-90, any percentage number>: the shape is a semi-circle,
3. If the angle is any other value other than described in case 1 and case 2, the shape is an arbitrary curve.

3.6 Scope Rules

The scope of a variable is the context within which it is defined. Mirage does not support global declaration of variables. Variables can be declared only within functions. Any variable declared within a function is only local to that function. The scope of a parameter of a function definition begins at the start of the function block and persists through that function.

The programmer can start a new scope any time they create a statement block with { }, such as in iterative statements or if/else statements. Moreover, identifiers declared within curly brackets are not accessible outside of the brackets.

3.7 Miscellaneous

Mirage's provides basic functionalities for image file I/O. It allows the user to save the contents of the graphics buffer into an image file or load an image file into the graphics buffer.

save filename;

save an image as a file with specified filename. The type of filename is string. The filename should have the following extensions, such as .png and .jpg.

load filename;

Load an image file with specified filename. The type of filename is string. The user is able to draw additional image on the loaded image file and save the file.

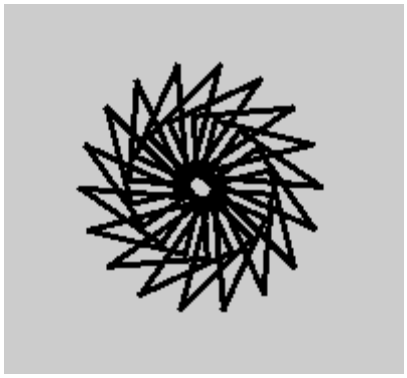
3.8 Sample Mirage Program

3.8.1

```
/* Function to draw a square */

void draw_square(int a)
{

      /* Variable declaration */
      int i;

      /* Loop four times */
      for(i=0; i<4; i++)
      {
            fwd a;
            rgt 90;
      }

}

/* Main function */

void main()
{
      /* Variable declaration */
      int a,b;
      a=10;

      /*Put the pen down*/
      down;

      /* Call the function to draw a square */
      draw_square(a);

      /* Lift the pen up */
      up;

      /* Move the pen to the required position*/
      if(a<20)
      fwd 3*a;
      else
      fwd 2*a;

      /* Put the pen down */
      down;
      b= 2*a+10;

      /* Draw another square */
      draw_square(b);
      save "mirage_image.jpeg"

 }
```

3.8.2

```
/* Code to draw a squaggle */

void main()
{
	int i;

	for(i=0;i<20;i++)
	{
		fwd 50;
		rgt 150;
		fwd 60;
		rgt 100;
		fwd 30;
		rgt 90;

	}

}
```



3.9 References

[1] *C Reference Manual*, Dennis M. Ritchie.

[2] *Computer Science Logo Style volume 1: Symbolic Computing,* University of California, Berkeley.

Chapter 4

Project Plan

Throughout this semester, our group held weekly meetings with Professor Stephen Edwards to discuss and evaluate the progress of our project.

At the beginning of the semester, our group scheduled meetings much frequently to brainstorm and design the project. As the project moved progressively and getting more clearly about how Antlr and each part of the project relate and work together, we began to work independently and relied heavily on email correspondence to keep each other informed of the status of our assigned tasks. We also held a regular weekly meeting on Friday to update and exam individual responsibilities and works that had been done.

4.1 Timeline

The deadlines for some of the major components of the projects are listed below:

Planning	
9/08, 9/12	Brainstorming about the type of language, language target users, goals and functionalities. Proposing several languages.
9/15	Finalizing the type of language and refine design decisions.
9/22	Deciding the detailed language structure and basic requirements. Dividing project work responsibilities and starting to learn and getting familiar with Antlr.
9/26	Project Proposal due
Specifications	
9/29	Writing a simple Lexer and Parser
10/06	A simple Walker
10/13	Trying Hello world program in Mirage
10/19	Language Reference Manual due
Implementation	
10/20	Finishing Pen functions: forward, backward…
10/27	Finishing Data types
11/03	Finishing Scoping
11/07	Finishing Functions
11/10	Finishing For/if/while loop
11/17	Finishing Return and break statements
11/21	Finishing Erase on/off functions
11/31	Finishing Other functions curve, reset, etc

12/05	Code testing and sample programs in Mirage
Testing	
12/08	Testing and debugging
12/15	Finishing documentation and prepare for presentation
12/19	Project Report due

4.2 Team Responsibilities

The project responsibilities were divided among the team members as follows: (please see the details in chapter 5 Architecture design and Appendix source code and testing code.)

Front End	Peili Zhang, Abhilash I
Back End	Nalini Vasudevan, Ming Liao
Documentation	Ming Liao, Nalini Vasudevan
Testing	Nalini Vasudevan, Peili Zhang, Abhilash I

4.3 Project Log

September 8	Brainstorm about type of language, language target users, goals and functionalities. Propose several languages
September 12	First draft of proposal completed
September 15	Finish learning basics of ANTLR and writing small expression parsers
September 16	Finalize language and refine design decisions
September 20	Second draft of proposal completed
September 22	Finalize the language implementation details
September 25	Write a simple Lexer and Parser
September 26	Final project proposal completed
September 28	Finish a simple demo for Lexer, Parser, Walker
September 30	Implement forward, backward, rgt and left, up, down in Antler
October 1	Integrate and test Walker, Parser, Lexer again
October 3	Begin working on the grammar
October 5	First meeting with Professor
October 5	Begin working on the LRM
October 6	Finish expressions (ANTLR)
October 8	Finish If /Else (Control flow)
October 12	Finish Identifier (ANTLR)
October 15	Finalize grammar
October 19	Finalize LRM
October 20	For and While loop implementation

October 26	Fix some bugs that inhibit in progress
October 25	Major changes to Parser and Walker to meet the complete specification in LRM. Changes to get the desired AST structures
October 27	Begin working on symbol table
November 1	Function implementation, activation records, recursion handling
November 5	Start working on data type support and Symbol tables
November 7	Finish Return statement
November 11	Revise the LRM
November 14	Finish data type support and implement scoping rules.
November 15	Test programs
November 14	Finish Save/Load Image(ANTLR&JAVA)
November 16	Finish proper scoping
November 17	Revise For,While Loop(ANTLR)
November 20	Argument handling
November 21	Bug fixing
November 22	Finish break, reset;
November 23	Support for nested for, while, working with functions
November 28	Implemented semantic analysis
November 29	Finish Curve (ANTLR&JAVA)
November 30	Testing and fixing bugs.
November 31	Finish thickness, speed and color functionalities (ANTLR & JAVA)
December 5	Giving proper error messages for scope
December 7	Revise grammar
December 8	Testing code, indent and clean commented code
December 10	Begin working on documentation
December 11	Return with a value return flag. Throwing appropriate error in function handling.
December 12	Finish eraseon and erasoff
December 13	Fix the bugs
December 14	Testing code again
December 15	Cleaning & wrapping up the code
December 16	Complete project documentation
December 17	Begin working on presentation and organize presentation
December 18	Presentation
December 19	Report due

4.4 Programming Guide and Software Development Environment

The back end codes are written entirely in Java, while the front end codes are written in ANTLR. In general, there were often overlaps of participation between the front-end parsing and back-end coding. Not only did every team member work on the specific assigned piece of the source code but also each one participated in several works simultaneously to ensure the whole project is agreed upon. For example, as the team leader, Nalini had to participate in developing both the front-end and the back end coding to make sure that there are no inconsistencies in the developing progress.

Software Development Environment Table

Operating System	Windows XP, Unix
Java Runtime Environment	Java 1.4.2
Development Platform	Eclipse 3.1, NetBeans IDE 5.0
ANTLR	ANTLR 2.7.5
Version control system	CVS

Java

With the exception of the Lexer, Parser, and Walker, which were written in ANTLR, all of the other components of the project were written in Java. We utilized Java object oriented philosophy to create well-defined classes and methods. The version of the Jave Runtime Environment used for this project is Java 1.4.2. We also followed the java programming convention at http://java.sun.com/docs/codeconv/, and constantly referred Java 2D API documentation at http://java.sun.com/j2se/1.4.2/docs/guide/2d/spec.html.

Operating Systems

Our team members mainly used Windows XP system with the certain assistants from Unix system for project development.

ANTLR (ANother Tool for Language Recognition)

ANTLR is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions. We used ANLTR to implement our Lexer and Parser, and used ANLTR-generated codes to construct the AST (abstract syntax tree) and to create a tree walker to traverse through it.

IDE

Eclipse SDK 3.1 and NetBeans 5.0 were used to facilitate the development process. The ANLTR plug-in for Eclipse that we used is ANTLR 2.7.5.

CVS

CVS is a revision control system that was used to organize our project. The CVS Repository was hosted on a CRF Unix machine and was accessed both remotely and from the CLIC lab.

Chapter 5

Architectural Design

Mirage translator consists seven inter-related components: Lexer, Parser, Walker, Interpreter, Exception, Symbol Table and Painter.

- The Lexer that reads Mirage program *.mrg and translates a stream of characters into a stream of tokens and then outputs tokens to the Parser.
- The Parser analyzes the syntactic structure of the program and translates the stream of tokens into an AST (abstract syntax tree).
- Walker reads the AST and checks whether the context constraints are obeyed, such as checking errors and the language semantics.
- Interpreter reads the AST and executes the program. When the interpreter (at the back-end) executes the program, there are three things to do:

  (1) Looks up the symbol table to find the corresponding functions and variables.
  (2) Executes the graphics functions and paints the graphics on the Frame by the painter.
  (3) Error handling and exception catching.

The following diagram illustrates the interaction between above stated elements:

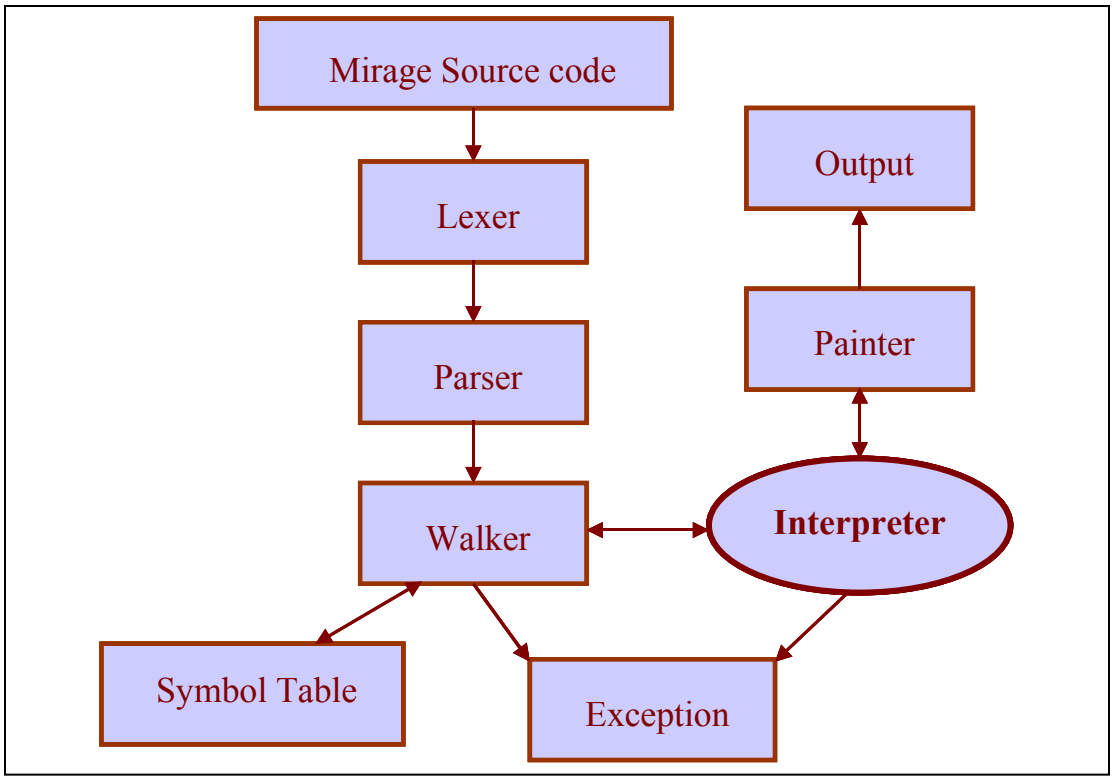


Figure 5.1 The Overall Mirage Architectural Design

The Lexer (MirageLexer.g), the Parser (MirageParser.g) and the tree Walker (MirageWalker.g) are implemented by ANTLR. The source file MirageWalker.g contains actions calling the methods defined in the class Interpreter. The Lexer, Parser and Walker are implemented by Abhilash I and Peili Zhang.

The Interpreter is implemented by JAVA. The source files for the Interpreter include MirageMain.java (main function), Env.java (symbol table, scope), Hstack.java (control flow), ActivationRecord.java (activation record for functions). The MirageMain.java reads the content of any Mirage file *.mrg as the input and then passes it to the Lexer; and the Parser and the Walker will do the subsequent tasks. These java files are implemented Nalini Vasudevan and Abhilash I.

The Painter uses BufferedImage to store graphics. The Painter first paints the components into BufferedImage, and subsquently calls the paintComponent() method whenever the graphics are updated. The Painter supports most graphic functionalities, for example, to change the color, thickness, speed and so on, plus perform the Image I/O. The source files for the Painter include MirageApp.java and MirageDisplay.java. These java files are implemented by Ming Liao and Nalini Vasudevan.

(Please refer the authors with each file in the Appendix source code)

Lexer:

There are four parts in the Lexer: expression operators, identifier definitions, key words and others (including comments, semicolon, and so on).

(1) Expression operator: we defined the operators for expressions:

```
LPAREN     : '(';          NEQUAL     : "!=";
RPAREN     : ')';          NOT        : '!';
PLUS       : '+';          LBRACE     : '{';
MINUS      : '-';          RBRACE     : '}';
STAR       : '*';          DOT        : '.';
DIV        : '/';          LTE        : "<=";
ASSIGN     : '=';          GTE        : ">=";
COMMA      : ',';          LT         : '<';
EQUAL      : "==";         GT         : '>';
```

(2) Identifier definitions: identifiers consist of letters, digits and underline "_" and the first character must be a letter, for example:

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

(3) Key words: we have many key words for Mirage programs. For example, "fwd","bwd","up", and so on. We placed them in Tokens section as predefined tokens in order to avoid the conflicts between the key words and the identifiers. For example,

```
tokens {

      /* Key words  */

      FD = "fwd";
      BD = "bwd" ;
      LEFT = "lft";
      RIGHT = "rgt";
      PENUP = "up";
      PENDOWN = "down";
      RESET = "reset";
      ...

}
```

(4) Comments: we use the option "Greedy = false" for comments. For example,

```
COMMENT : "/*" (options{greedy=false;}:(.))* "*/" {$setType(Token.SKIP);};
```

Parser:

The MirageParser uses the results obtained from the MirageLexer to create an abstract syntax tree that represents the grammatical phrases in the source program. The user will be notified during this phase if there are any syntax errors present in the code.

Mentioning the parser for ‘for’ loop is worthwhile. ‘for’ is made the parent node. The condition for the for loop includes three expressions separated by two semicolons within a pair of parenthesis followed by the body of the ‘for’ loop. All the three expressions are optional and since the Walker has a look-ahead value of ‘1’, we cannot have optional expressions in the middle of the AST sibling sequence. So, we have added three dummy nodes called ‘FOR EXP’, which are placeholders for the three expressions.

AST Walker:

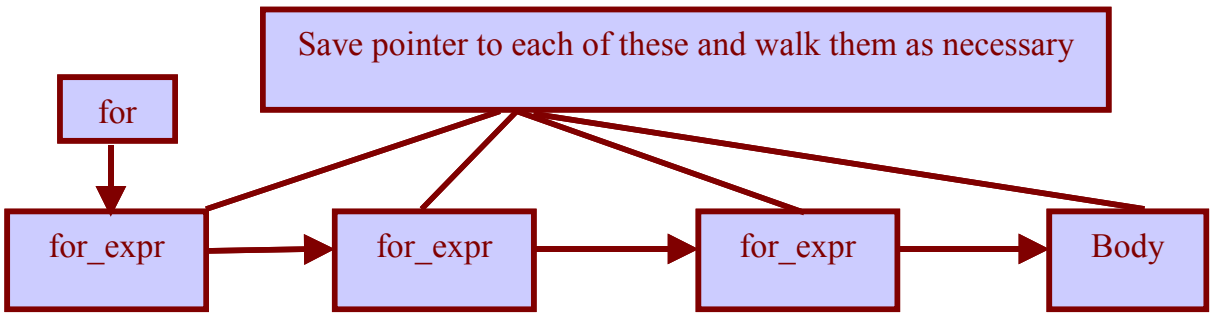
Parsing nodes related to graphics statements in AST is simple since the graphics statements include the keywords “fwd”, “bwd”, “rgt”, “lft”, “up”, “down”, “color”, “thick”, “speed”, “curve”, “save”, and “load”. The AST represents the graphics statements as a branch with root as keyword and children as arguments. When walking in the AST, if there is a root matching the keyword, the walker will call the related function from the back-end and pass the values from children nodes to the corresponding function.

- Control flow

The most interesting part of building our interpreter is to implement control flow. We know that a compiler producing Java code simply outputs code for the "if/else", 'for', 'while' and function calls and java compiler takes care of the execution of the loops and function calls. While, an interpreter has to actually handle the control flow.

For example, here is how we handle the 'for' and 'while' loops. First of all, the tree structure of for loop as discussed in Parser section has a root node "for" and it has four children 'for_exp', 'for_exp', 'for_exp' and the body of for loop. 'for_exp's are three different expressions separated by semicolons in for loop. We first save the pointer to the AST of the for body, then walk the first expression, walk the second expression and evaluate the value it returns. If it is '0', then start walking the AST that is sibling of 'for'. If it is not '0', then walk the pointer of the body saved earlier and then third expression. Now walk expression two, evaluate it and repeat the processes of walking if it is not '0'. The while loop works in a similar way.

The following figure shows how the for loop is implemented:



- Function Handling:

ActivationRecord.java and MirageFunction.java are the two classes that implement functions and support recursion. The structure of user-define functions has void function type, a function name, a parameter list, and the corresponding symbol table. Whenever a new function is called, a new activation record is created and pushed onto a stack and the control enters into a new scope. An activation record is in turn a stack of scopes. When the function finishes, the activation record is deleted from the stack, and the control returns to the previous activation record.

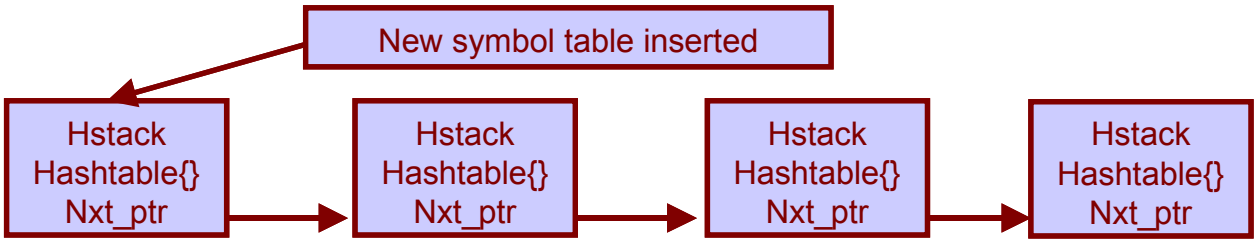
Since we are walking only once, all functions have to be defined before they are called. Also note that the functions and variables have different namespaces, that is, different symbol tables are maintained for them accordingly. There is one global symbol table for functions and many local symbol tables for variables depending on the scope and which function the variable occurs in.

- Scoping and Symbol Tables:

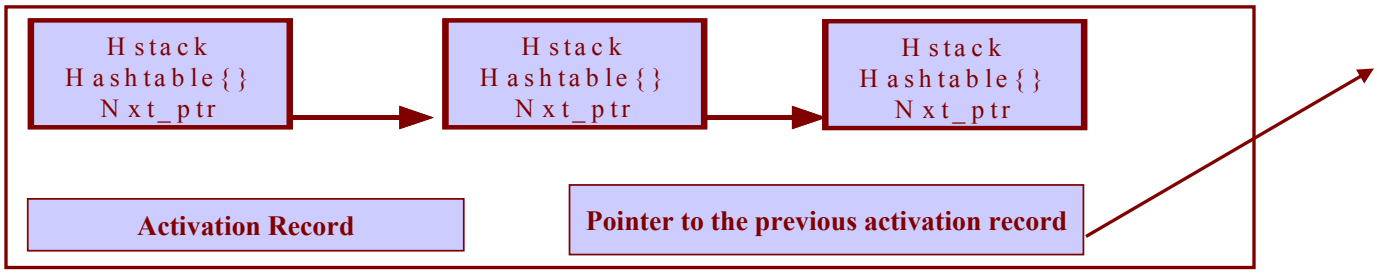
The symbol table keeps track of the name, the type and the scope of all the identifiers used in the Mirage source program as well as the name, the number and the type of the arguments passed to the function. The symbol table is implemented as a Hash stack data structure for easy lookup.

Hstack is a wrapper class for Hashtable class provided by java. It is basically used to support a linked list of hash tables. Within an activation record, the structure of the linked list of symbol tables is shown in the following figure.

If the program enters a new scope, a new hash table (a symbol table) is created and added to the linked list as shown in the following figure.



When reading the function definition, the interpreter puts the function name, type, the parameters, and the root of the subtree of the function body into the current symbol table. When a function call occurs, it takes the parameters from the function call statements and tries to match the function definition in the current symbol table. If matched, a child of the current symbol table is created.



Declaring a function:

Whenever a function is declared, the AST of the function with the identified argument types and the return value is saved in a hash table which is different from the hash table used in the current scope. The binding of function name and the function is global. So the AST of function as

mentioned above is stored in a hash table with the identifier as index. Whenever a function call is encountered, we do a lookup in the hash table by the identifier and thus get the pointer to the AST. At this point of time, a new activation record is also created. Each activation record consists of linked list of hash tables mentioned in the previous section.

Calling a function:

When a function call is made, the data structure shown above will have just one scope and there is a pointer to the activation record of the calling function. As we enter and leave scopes in the same function, the linked list grows and shrinks. When leaving the function, present activation record pointer is made equal to the previous activation record. Thus scoping rules are obeyed and recursion is also taken care of. Therefore, at any point of time there is a stack of such activation records and the height of stack depends on how deep the function calls are nested.

Declaring a Variable:

Whenever a new variable is going to be added to the symbol table, only the first element of the linked list in present activation record (only in the current scope) is checked to see if this variable has been already declared. It can be present in a different scope in the same activation record. However, if it is already declared in the same current scope an error message is generated and the program exits. If it is not declared, the new variable will be added to the current scope.

Accessing a variable:

Whenever a variable is going to be accessed, in the present activation record (the current scope) the first hash table is checked. If it cannot be found, the linked list recursively searches in the previous scope (the parent of the current symbol table) for the presence of the variable until it finds the variable. If the variable cannot be found in any of the hash tables of the present activation record, then an error message is generated and program exits.

- Error Handling

Semantic error handling is done with the help of symbol tables. Whenever a variable or function is used, it is first checked in the symbol table. If it is not present, an error is thrown. Similarly if a variable is already declared, an error is thrown. Similar checking is made on functions as well. For example, checking for the right number of arguments, right return type and so on.

Chapter 6

Testing Plan

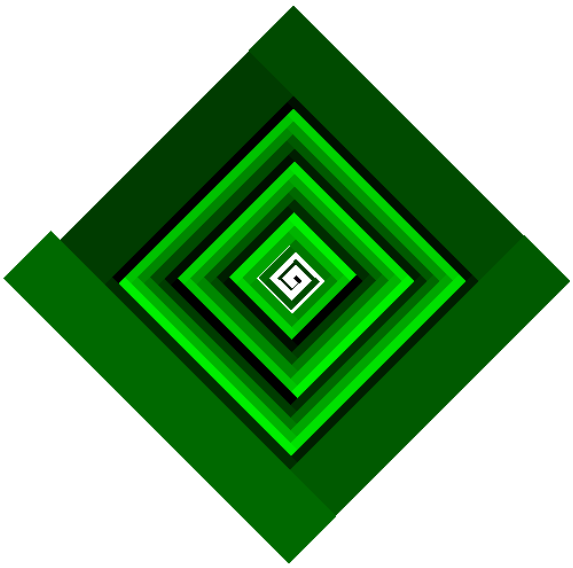
Since the main goal of our project is to design a simple graphic programming language that can creates pleasant graphics and encourage the beginners to learn, it is important that our interpreter is thoroughly tested so that there are no difficulties experienced by the users at runtime.

6.1 Unit testing

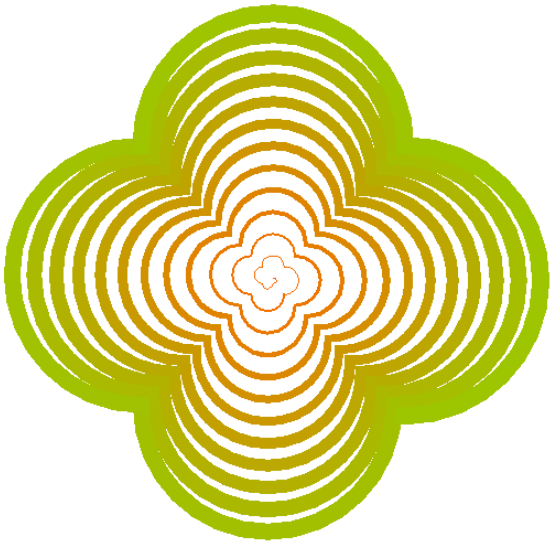
The Lexer, the Parser and the AST tree were tested at the early developing stage. Each java file has been test individually during the coding progress. Every team member was responsible for testing the part of code that was written by him or her. Here are some example programs for testing control flow statements--- while and for loops.

/* testing with while loop */

```
main()
{
	thick 3;
	int a=5;
	int c=90;
	int i=1;
	up;
	rgt 45;
	fwd 350;
	down;

	while (1)
	{
		if(i<60)
		{
			speed 20*i;
			rgt c;
			fwd a*i;
			color <0,i*15,0>;
			thick i;

			i=i+1;
		}
		else
		{
			break;
		}
	}

	save "1.jpg";
}
```

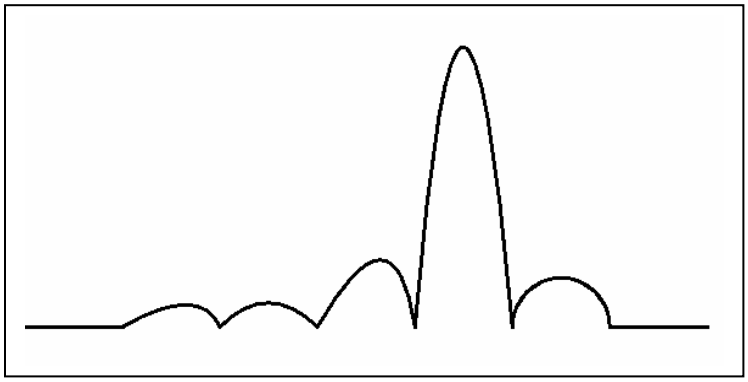


```
/* testing with for loop */

main()
{

int a=5, c=90, i=1;

/*Define the shape of line: Half
Cycle*/

curve <90,0>;

for(i=0; i<50; i=i+1)
{
      color <255-2*i,100+2*i,0>;
      thick i/3;
      rgt c;
      fwd a*i;
}

save "flower.jpg";
}
```



6.2 Basic graphic operations testing

Every basic graphical operation has been also tested. Either the resulting graphic will be shown on the screen or the error will be printed on the command line. Below is a curve statement sample test:

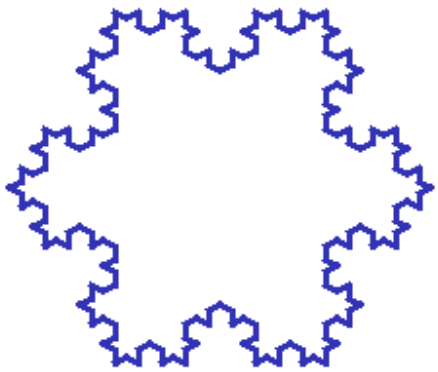


```
/* Testing to draw a Curve */

main()
{
      int a=80;
      up;

      rgt 90;
      fwd 500;
      down;

      lft 90;

      /*Default: Draw a line*/
      forward a;
      rgt 90;

      /*Draw a curve with offset 80%, 30 degrees angles*/
      curve <30,80>;
      fwd a;

      /*Draw a curve with offset 50%, 45 degrees angles*/
      curve <45,50>;
      fwd a;

      /*Draw a curve with offset 50%, 45 degrees angles*/
      curve <60,80>;
      fwd a;

      /*Draw a curve with offset 50%, 85 degrees angles*/
      curve <85,50>;
      fwd a;

      /*Draw a half cycle*/
      curve <90,0>;
      fwd a;

      /*Draw a line, return default mode*/
      curve <0,0>;
      fwd a;

      save "curve.jpg";
`
```

6.3 Integrated testing

We wrote many testing programs to test the error handling of back-end code, for example, calling functions with wrong types of parameters. Moreover, advanced tests such as nested loops, recursion, nested if/else, and scoping have been integrated in the Mirage programs and tested, so we can ensure our basic graphical statements can achieve certain specific features of the language.

```
/* recursion testing with user-
defined function */

void snowflake (int side, int depth)
  {
     if (depth ==0)
     {
      fwd side;
      return;
     }

     snowflake(side/3, depth-1);
     lft 60;

     snowflake(side/3, depth -1);
     rgt 120;

       snowflake(side/3, depth-1);

     lft 60;
     snowflake(side/3, depth-1);

  }

 void flake (int s, int d)
 {
      int i=0;

      for(i=0;i<3;i=i+1)
      {
            snowflake(s, d);
            rgt 120;
      }
 }

 main()
 {
      int i,n;
      color <50,50,180>;
      flake(200, 3);
      save "koch.jpg";

 }
```

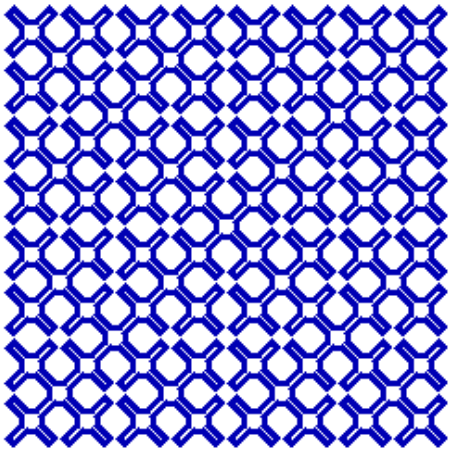


```
/* testing integrated functions */

void sier (int n, int a, int h, int k)
{
	if(n==0)
	{
		fwd k;
		return;
	}

	rgt a;
	sier( n-1, -a, h, k);
	lft a;
	fwd h;
	lft a;
	sier(n-1, -a, h,  k);
	rgt a;
}

void one_square()
{
	int i, h=5;
	for(i=0; i<4; i=i+1)
	{
		sier( 7, 45, h, 10);
		rgt 45;
		fwd h;
		rgt 45;
	}
}

int eight_squares(int flag)
{
	int  j;

	for(j=0;j<8; j=j+1)
	{
		rgt 45;
		if(flag==1)
		rgt 45;
		one_square();
	}
}

void hilb (int n, int a, int h)
{
	if (n == 0)
		return;

	rgt a;
	hilb  (n - 1, -a ,h);
	fwd h;
	lft a;
	hilb (n - 1, a, h);
	fwd h;
	hilb (n - 1, a, h);
	lft a;
	fwd h;
	hilb (n - 1, -a ,h);
	rgt a;
}
```

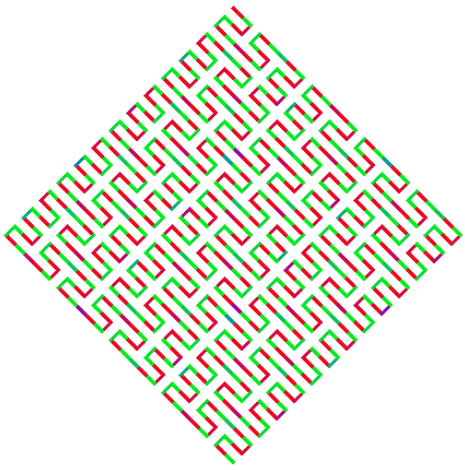
```
void hilb_mix()
{
	up;
	lft 90;
	fwd 130;
	rgt 90;
	fwd 130;
	down;
	int i=0;

	for(i=0;i<4;i=i+1)
	{
		rgt 90;
		hilb( 5, 90, 8);
	}
}
```

/* testing animation with eraseon & eraseoff */

```
main()
{
	color <180,20,0>;
	eight_squares(0);
	eraseon;
	eight_squares(1);
	eraseoff;
	color <11,125,18>;
	eight_squares(0);
	eraseon;
	eight_squares(0);
	eraseoff;
	color <20,0,255>;
	hilb_mix();
	save "sierp.jpg";

}
```

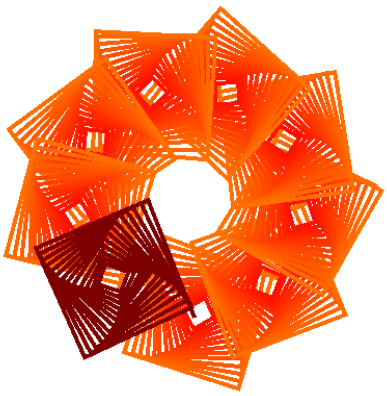


```
/*
      peano.mrg:
      testing for recursion

*/

void pean(int n, int a,  int h)
{
      if (n == 0)
            return;
      rgt a;
      pean(n - 1, -a, h);
      color<10, 255- n*20, n*30>;
      fwd h;
      pean (n - 1, a, h);
      color<255- n*20, 10, n*30>;
      fwd h;
      pean (n - 1, -a, h);
      lft a;
}

main()
{
      up;
      bwd 200;
      lft 45;
      down;
      pean( 6, 90, 12);
      save "peano.jpg";
}
```



Peano.jpg

```
/*
      tick.mrg:
      testing for while / animation

*/

void spiral(int size, int angle, int my_color)
      {
      if (size > 100)
      return;
    if(my_color)
    color <125, 0, 0>;
    else
    color <255, 14+size, my_color> ;
    fwd size+20;
    rgt angle;
    spiral (size + 2, angle, my_color);
    }

 main()
      {

      int i, j=0;

      thick 4;
      while(j<99)
      {
            int mod= j%9;
            j= j+1;
            reset;
            up;
            bwd 100;
            lft 120;
            down;
            for(i=0; i<9; i=i+1)
            {
                  if(i==mod)
                        spiral(0, 91, 1);
                  else
                        spiral(0, 91, 0);
            }
      }
      save "tick.jpg";
}
```



tick.jpg

Chapter 7

Lessons Learned

7.1 Abhilash I

According to me, the best decision we made for the final project was to implement an interpreter other than a compiler. This was an exciting programming experience that deals with the management of control flow. It has also made me more comfortable with building an AST and walking and traversing the tree as dictated by the control flow. Control flow wouldn't have been so enjoyable had it been a compiler.

Moreover, implementing the symbol tables and deciding the structure of AST were challenging to me. During the initial stage of the project implementation, it requires a great deal of thinking to figure out how to make the precedence and associativity correct. However, by the time when we have reached the end of project, more complex things also turned to be feasible to be done. I believe that working on the final project has significantly increased my comfort levels to understand the concepts of compilers and how it works in a real world.

What's more, discussing with teammates, exchanging ideas and meeting with Prof. Edwards and TAs have been an efficient learning process, which helped us to quickly solve some problems we are stucking on. In general, for me implementing the PLT project was a really fun learning experience.

7.2 Ming Liao

It has been a great experience working with my excellent and hard working teammates. The following lessons all came from what we have planed, organized and managed to do during this semester. I believe these good approaches have helped us to implement our project efficiently.

During the early design stage, our group wasted quite a lot of time in discussion due to the lack of a team leader. Once we elected a good team leader, we found everything progressed much faster and efficiently. Hence, finding a good team leader as early as possible can speed up and smooth the developing progress of the project.

Good communication is essential to achieving success with a teamwork-oriented project. Especially for both the documentation and the connection between back-end and front-end works, the project required frequent communications with all of the team members. Our teammates are very responsible, which made the communication straightforward and smoothly.

Everyone should ask Professor, TA or discuss with other team members with any concerns or questions regarding to the project without hesitation. It was not only an efficient learning procedure, but also helped us to save time to solve hard problems.

A team should always setup reasonable goals in advance and give enough time to work between consecutive deadlines. For example, during exam period, since team members have their own exams to prepare for, it is very difficult to arrange weekly meetings and discuss the work together. For us, we found it was very efficient to divide the jobs early and planed carefully according each member's own schedule and cooperated together while let each member focus on own separated job. After exams, we came back as soon as possible to plan and finish our next step for the project.

7.3 Nalini Vasudevan

It has been a wonderful experience working with others in this project. I learnt a lot from this project especially the compiler concepts; there are other important things that I would like to share about:

- We used an incremental approach that turned out to be a great success during the course of the project. We started off with simple versions of Lexer, Parser and Walker and we built on it. It may not be the best approach, but it has turned out to be successful for the development of our project.
- The project was about to develop a graphical language interpreter; therefore, the outputs were visual and colorful, which attracted the eyes of both the team members and others and we enjoyed a lot during the implementation.
- The project was started quite early and it was almost well completed in advance before the deadline. Hence it did not leave us with any last minute work. So starting as early as possible will help each team to be flexible and relaxed at the end of the semester.

7.4 Peili Zhang

The most two significant skills I have learned during the class are how to become a quick learner and how to make a good plan. I think these skills will benefit me a lot in the future study and work.

Before doing the project, I was not familiar with any ANTLR and JAVA graphic programming. At the beginning, I searched a lot of sample codes for some functions online, and kept trying them and finding how they work. However, it didn't speed my learning process up, since I only focused on some specific functions, such as the expressions in ANTLR or the image storage in JAVA. I couldn't catch the whole picture of either two languages and even couldn't figure out some unexpected problems. Therefore, I began to read the whole ANTLR manual and the documentation for the JAVA graphic part. The good thing was that it didn't take me much time but benefit me a lot. I realized that the quick learning does not mean one can ignore the basics.

Another lesson I have learned from the project is that the planning of the project is very important and should be supervised by someone as early as possible. For example, when all team

members do not have a coordinate schedule to meet or work during special time, it would be better to plan ahead. Moreover, due to the lack of a team leader and the unclear task assignments at the early developing stage, teammates might waste time to repeat the same jobs. Finally, I have learned it was a more flexible and efficient way for implementing the teamwork through good communication. In addition, a team can use any resources to communicate with each other. For instance, teammates can meet in class and discuss more details via emails or Skype instead of fixed meeting if they cannot make it.

From the project, I did learn a lot of things. ANTLR is really interesting and powerful language and it helps me understand more about the mechanisms of a compiler. I was very much enjoyed working with such an organized team!

Appendix A

Source Code

Statistics

```
MirageMain.java: 27 lines
MirageApp.java: 121 lines
MirageDisplay.java: 156 lines
MirageFunction.java: 91 lines
ActivationRecord.java: 19 lines
Env.java: 68 lines
Hstack.java: 37 lines

MirageParserLexer.g :156 lines
Walker.g: 110 lines

Total: 785 line
```

A.1 Front End

```
/*
   MirageParserLexer.g
   Peili Zhang (pz2128@columbia.edu)
   Abhilash I  (ai2160@columbia.edu)

*/

class MirageLexer extends Lexer;

options {

   testLiterals =false;
   k=2;
   charVocabulary = '\3'..'\377';
   }

tokens {
   /* Key words  */

   FD = "fwd";
   BD = "bwd" ;
   LEFT = "lft";
   RIGHT = "rgt";
   PENUP = "up";
   PENDOWN = "down";
   RESET = "reset";
   INT= "int";
   END = "end";
   IF = "if";
```

```
   ELSE = "else";
   RETURN = "return";
   FOR = "for";
   WHILE = "while";
   LOAD = "load";
   SAVE = "save";
   MAIN = "main";
   BREAK = "break";
   VOID = "void";
   AND = "&&";
   OR = "||";
   COLOR ="color";
   THICK ="thick";
   SPEED ="speed";
   CURVE ="curve";
   ERASEON="eraseon";
   ERASEOFF="eraseoff";
}

/*  Expression */

LPAREN   : '(' ;
RPAREN   : ')' ;
PLUS     : '+' ;
MINUS    : '-' ;
STAR     : '*' ;
DIV            : '/';
ASSIGN   : '=';
COMMA   : ',';
EQUAL   : "==";
NEQUAL  : "!=";
NOT     : '!';
LBRACE  : '{';
RBRACE  : '}';
DOT            : '.';
LTE            : "<=";
GTE            : ">=";
LT       : '<';
GT       : '>';


/* Identifier  */

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

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

NUMBER : (DIGIT)+;

EXP : DOT (NUMBER|) 'e'(PLUS|MINUS|/*NOTHING*/)NUMBER;

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

STRING options { testLiterals =false;}: '"'!(~('"'|'\n'))*'"'!;

/* Others  */
```

```
COMMENT : "/*" (options{greedy=false;}:(.))* "*/" {$setType(Token.SKIP);};

SC : ';';

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


class MirageParser extends Parser;
options {
   buildAST=true;
   k=2;

}
tokens {
NEGATE; DECLS; FUNCCALL; SUBPROG; FUNCDEF;FOREXPR;
}

//We don't have global variables, our program has functions always
program:  functions;

//functions is one or more functions
functions: (function)* main;
main: MAIN^ LPAREN! RPAREN! subprogram;

//function is something like
//int foo(int a, int b ...)
//{...}

function: ret ID  LPAREN! decls RPAREN! subprogram
 {#function = #([FUNCDEF,"FUNCDEF"], function);};
decls : (decl  (COMMA! decl )*
      | /*NOTHING*/){ #decls = #([DECLS, "DECLS"], #decls); } ;


ret: INT| VOID;

subprogram : LBRACE! (stmt)* RBRACE!
         {#subprogram = #([SUBPROG,"SUBPROG"], subprogram);};

stmt     : (movestmt
         | penstmt
         | varstmt
         | bool
         | returnstmt
         | lsstmt
         | breakstmt
         | colorstmt
         | fillstmt
         | speedstmt
         | curvestmt
         | RESET
```

```
         | thickstmt) SC! | subprogram | forstmt | whilestmt | ifstmt
         ;
fcstmt : ID LPAREN! varlist RPAREN!
              {#fcstmt = #([FUNCCALL,"FUNCCALL"], fcstmt); };

varlist: (((bool) (COMMA! (bool))*)|/*nothing*/);

colorstmt : COLOR^ LT! expr COMMA! expr COMMA! expr GT!;

thickstmt : THICK^ expr;

speedstmt : SPEED^ expr;

curvestmt : CURVE^   LT! expr COMMA! expr GT!;

fillstmt : FILL^ LT! NUMBER COMMA! NUMBER GT!;

breakstmt : BREAK;

lsstmt : (LOAD^|SAVE^) STRING;
forstmt : FOR^ LPAREN! forexpr SC!
            forexpr SC!
            forexpr RPAREN! stmt;

forexpr : (bool)? {#forexpr = #([FOREXPR,"FOREXPR"], forexpr); };

whilestmt
: WHILE^  LPAREN! bool RPAREN! subprogram
;

returnstmt : RETURN^ (bool)?;

ifstmt
      : IF^ LPAREN! bool RPAREN! stmt
        (options{greedy = true;}: ELSE! stmt)?;
varstmt
   : (INT^) args
   ;

decl : (INT^)  ID;

args
   : arg (COMMA! arg )*
   ;

arg      : ID (ASSIGN^ bool)?
   ;

movestmt
   :  (FD^|BD^|LEFT^|RIGHT^) expr
   ;

penstmt
   : PENUP|PENDOWN|ERASEON|ERASEOFF
   ;

bool : fbool ((AND^ | OR^) fbool)*;
```

```
   fbool : gbool ((EQUAL^ | NEQUAL^ | LT^ | GT^ | LTE^ | GTE^) gbool)*;

   gbool : (ID ASSIGN^ expr) | expr;

   expr     :  mexpr ((PLUS^|MINUS^) mexpr)*
      ;

   mexpr    :  unary ((STAR^|DIV^) unary)*
      ;

   unary : (MINUS^ atom) {#unary.setType(NEGATE);}
         | (NOT^ atom)
         | atom ;

   atom    :  NUMBER
      |   LPAREN! expr RPAREN!
      |   ID
      | fcstmt
      ;
```

```
   /*
      MirageWalker.g
      Abhilash I  (ai2160@columbia.edu)
      Peili Zhang (pz2128@columbia.edu)
      Nalini Vasudevan (nv2144@columbia.edu)

   */

   class MirageTreeParser extends TreeParser;

   options {
      importVocab=MirageParser;
   }
   {
      MirageApp appobj;
      java.util.Hashtable dict = new java.util.Hashtable();
      Env env ;
      static int RETURN_TRUE=1;
      static int BREAK_TRUE=2;
      String s=null;
   }

   program
   {
      appobj= new MirageApp();
   }
      : (function)* main;


   function {int return_type=0;}: #(FUNCDEF return_type = ret ID decls
subprogram:.) {MirageFunction.funcRegister( #ID.getText(), #subprogram,
return_type );};

   ret returns [int r=0]: (VOID {r=0;}) | (INT {r=1;});
```

```
   decls : #(DECLS args);

   args : arg args| ;


   arg : #(INT ID) {MirageFunction.registerOneArgument(#ID.getText());};

   main{env = ActivationRecord.create_activation_record(null);}:      #(MAIN
subprogram);

   funexecute returns [int r=0]: #(SUBPROG  r=stmts){if(r==RETURN_TRUE)
return RETURN_TRUE;};

   subprogram  returns [int r=0]: #(SUBPROG {env.enter_scope();}r=stmts) {if
(r==RETURN_TRUE) return RETURN_TRUE; env.leave_scope();};

   stmts returns [int r= 0 ]: (r=stmt { if(r==RETURN_TRUE) return
RETURN_TRUE; if(r==BREAK_TRUE) return BREAK_TRUE;})*;


   stmt     returns [int r=0]
   {int a=0,b=0,c=0;}

      :   r= subprogram {if(r==RETURN_TRUE) return
RETURN_TRUE;if(r==BREAK_TRUE) return BREAK_TRUE;}
      |     #(FD  a=expr)  {appobj.forward(a);}
      |   #(BD  a=expr)  {appobj.backward(a);}
      |   #(RIGHT  a=expr)  {appobj.clock(a);}
      |   #(LEFT  a=expr)  {appobj.aclock(a);}
      |   #(INT (initialize)*)
       |   #(ASSIGN ID

      {if(env.get_variable(#ID.getText())==null)

      {System.out.println("Variable "+ #ID.getText()+" not
found");System.exit(0);}}
                                                          a=expr
{env.add_or_modify(#ID.getText(),a);}
            )
      |   #(IF a=expr temp1:. (temp2:.)?)
            {
             if(a!=0)
                         {r=stmt(#temp1); if(r==RETURN_TRUE) return
RETURN_TRUE; if(r==BREAK_TRUE) return BREAK_TRUE;}
            else  if(#temp2!=null)
                         {r=stmt(#temp2); if(r==RETURN_TRUE) return
RETURN_TRUE; if(r==BREAK_TRUE) return BREAK_TRUE; }
            }
    |    #(WHILE while_expr:. loop_body:.)
            {
                  while (expr(#while_expr)!=0)
                  {
                         r= stmt(#loop_body); if(r==RETURN_TRUE) return
RETURN_TRUE; if(r==BREAK_TRUE) break;
                   }
            }
      |    #(FOR  {a=1;}
```

```
                            #(FOREXPR expr1:.
{if(#expr1!=null){this.stmt((#expr1));}})
                            #(FOREXPR expr2:.
{if(#expr2!=null){a=this.stmt((#expr2));}})
                            #(FOREXPR expr3:.)  body:.)
            {

                    while(a!=0)
                    {
                        if((#body)!=null)
                        {
                        r=stmt((#body));if(r==RETURN_TRUE) return
RETURN_TRUE; if(r==BREAK_TRUE) break;
                        }
                        if((#expr3)!=null)
                        {
                              stmt((#expr3));
                        }

                        if((#expr2)!= null)
                        {
                        a=expr((#expr2));

                        }
                    }
              }
      |   BREAK   { r=BREAK_TRUE;}
      |   PENUP {appobj.penup();}
      |   PENDOWN {appobj.pendown();}
      |   ERASEON {appobj.eraseon();}
      |   ERASEOFF {appobj.eraseoff();}
      |   #(COLOR a=expr b=expr c=expr) { appobj.color(a,b,c); }
      |   #(THICK a=expr) { appobj.thickness(a);}
      |   #(SPEED a=expr) {appobj.speed(a);}
      |   #(CURVE a=expr  b=expr) { appobj.setcurve(a,b);}
      |   #(SAVE savefile:STRING){appobj.save(savefile.getText());}
      |   #(LOAD loadfile:STRING){appobj.load(loadfile.getText());}
      |   #(RETURN   (exp:.)? {if(#exp== null)
MirageFunction.addReturnValue(null); else MirageFunction.addReturnValue(new
Integer(expr(#exp))); if(true) return RETURN_TRUE;})
      |     RESET {appobj.reset();}
      |   r=expr
      ;


   initialize{int a;}: #(ASSIGN ID

      {if(env.check_in_present_scope(#ID.getText())==true)

      {System.out.println("Variable " +#ID.getText()+ " already
defined");System.exit(0);}}
                                                      a=expr
{env.add_in_present_scope(#ID.getText(),a);}
                                                      )

                  |ID {if(env.check_in_present_scope(#ID.getText())==true)
                                                            {
```

```
   System.out.println("Variable " +#ID.getText()+ " already defined");
   System.exit(0);
   }

env.add_in_present_scope(#ID.getText(),0);};

   expr returns [int r=0;]
   { int a,b; }
      :   #(PLUS  a=expr b=expr)  {
                                                    r=a+b;
                                                 }
      |   #(MINUS a=expr b=expr)  {
                                                    r=a-b;
                                                 }
      |   #(STAR  a=expr b=expr)  {
                                                    r=a*b;
                                                 }
      |   #(DIV   a=expr b=expr)  {
                                                    r=a/b;
                                                 }
      |   #(EQUAL a=expr b=expr) { r=0;if(a==b)r=1; }
      |   #(NEQUAL a=expr b=expr){ r=0;if(a!=b)r=1; }
      |     #(LT a=expr b=expr){r=0;if(a<b)r=1;}
      |     #(LTE a=expr b=expr){r=0;if(a<=b)r=1;}
      |     #(GT a=expr b=expr){r=0;if(a>b)r=1;}
      |     #(GTE a=expr b=expr){r=0;if(a>=b)r=1;}
      |   i:NUMBER               {r = (int)Integer.parseInt(i.getText());}
      |   #(NEGATE a = expr) { r = -a;}
      |   #(FUNCCALL ID  pass_args ) {r=0;
                                                                  env =
ActivationRecord.create_activation_record(env);

      env.enter_scope();

      r=MirageFunction.funcCall(this, #ID.getText(), env);

env.leave_scope();
                                                                  env
=ActivationRecord.remove_activation_record(env);}
      |   ID {Integer t=env.get_variable(#ID.getText());
            if(t==null)
            {
                  System.out.println("Variable "+#ID.getText()+" not
found");System.exit(1);
            }
            else
            {
                  r=t.intValue();
            }
      }

      ;
      pass_args: (value)*;
      value {int r;}:(r=expr){MirageFunction.setValueOfOneArgument(r);};
```

A.2 Back End

```
   /*
      MirageMain.java
      Peili Zhang (pz2128@columbia.edu)

   */

import antlr.*;
import antlr.collections.*;
import java.io.*;
import java.util.Vector;

public class MirageMain {
      public static void main(String[] args) throws Exception
      {
            if(args.length==0) { error(); }
                  FileInputStream fileInput = null;

            try {

                  fileInput = new FileInputStream(args[0]);
            } catch(Exception e) { error(); }


            try {

                  DataInputStream input = new DataInputStream(fileInput);
                  MirageLexer lexer = new MirageLexer(input);
                  MirageParser parser = new MirageParser(lexer);
                  parser.program();
                  CommonAST tree = (CommonAST)parser.getAST();
                  /*System.out.println("=============AST
Structure=============");
                  System.out.println( tree.toStringList() );
                  System.out.println("=============     END
=============");*/
                  MirageTreeParser treeParser = new MirageTreeParser();
                  treeParser.program(tree);

            } catch(Exception e) { System.err.println(e.getMessage()); }
      }

      private static void error() {

        System.out.println("*----------------------------*");
      System.out.println("| USAGE:                     |");
      System.out.println("|   java MirageMain inputfile |");
      System.out.println("*----------------------------*");
      System.exit(0);
   }


}
```

```
   /*
      Env.java
      Abhilash I (ai2160@columbia.edu)

   */

import java.util.Hashtable;

public class Env
{
      Hstack scope=null;
      Env parent_env = null; //for activation records

      public Env()
      {
            scope = null;
            parent_env= null;

      }
      public void enter_scope()
      {
            Hstack temp=scope;
            scope=new Hstack();
            scope.outer=temp;
      }
      public void leave_scope()
      {
            if(scope.get_outer()!=null)
                  scope.set_p_scope(scope.get_outer().get_p_scope());
      }
      public void add_or_modify(String s, int i)
      {
            Integer t;
            Hstack temp=scope;
            t=(Integer)temp.get_variable(s);
            while(t==null && temp.outer != null)
            {
                  temp=temp.get_outer();
                  t=temp.get_variable(s);
            }
            if(t==null && temp.outer == null)
            {
                  t=new Integer(i);

                  scope.add_or_modify_variable(s, t);
            }
            t=new Integer(i);

            temp.add_or_modify_variable(s, t);
      }
      public void add_in_present_scope(String s, int i)
      {
            Integer t=new Integer(i);
```

```
            scope.add_or_modify_variable(s, t);
      }
      public boolean check_in_present_scope(String s)
      {
            if(scope.get_variable(s)!=null)
                  return true;
            else
                  return false;
      }
      public Integer get_variable(String s)
      {
            Integer t;
            t=(Integer)scope.get_variable(s);
            Hstack temp=scope.get_outer();
            while(t==null && temp != null)
            {
                  t=temp.get_variable(s);
                  temp=temp.get_outer();
            }
            return t;
      }
}
```

```
   /*
      Hstack.java
      Abhilash I (ai2160@columbia.edu)
   */

   import java.util.Hashtable;

   public class Hstack
   {
      Hashtable p_scope=null;
      Hstack outer=null;
      public Hstack()
      {
            p_scope=new Hashtable();
            outer=null;
      }
      public void add_or_modify_variable(String s, Integer T)
      {
            p_scope.put(s, T);
      }
      public Integer get_variable(String s)
      {
            Integer t;
            t=(Integer)p_scope.get(s);
            return t;
      }
      public Hashtable get_p_scope()
      {
            return p_scope;
      }
      public Hstack get_outer()
      {
```

```
        return outer;
    }
    public void set_p_scope(Hashtable h)
    {
        p_scope=h;
    }
    public void set_outer(Hstack h)
    {
        outer=h;
    }
}
```

```
/*
    ActivationRecord.java
    Nalini Vasudevan (nv2144@columbia.edu)

*/

public class ActivationRecord
{

    public ActivationRecord()
    {

    }
    public static Env create_activation_record(Env caller)
    {
                Env new_env = new Env();
                new_env.parent_env = caller;
                return new_env;
    }

    public static Env remove_activation_record(Env present)
    {
        Env temp = present.parent_env;
        present = null;
        return temp;
    }

}
```

```
/*
    MirageDisplay.java
    Ming Liao  (ml2288@columbia.edu)
    Nalini Vasudevan (nv2144@columbia.edu)

*/

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.lang.Object;
import javax.swing.*;
```

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.QuadCurve2D;

   public class MirageDisplay extends JPanel {
   static Graphics2D g2i;

   final static int WIDTH = 600;
   final static int HEIGHT = 600;

   int imgWidth,imgHeight;
   int state;
   int erase;
   int stroke;
   int speed;
   Color color;
   int angle;
   int percentage;

   public static BufferedImage bufferimage;

   protected int xglobal, yglobal, thglobal;
   //private Graphics2D g2d;

  //panel constructor
   public MirageDisplay(){
       state=0; //Default pen down
       erase=0; //Default no erase

       bufferimage = new BufferedImage(WIDTH, HEIGHT,
               BufferedImage.TYPE_INT_RGB);


       g2i = bufferimage.createGraphics();
       color = Color.black;
       stroke=3;
       speed=0;
       angle=0;
       percentage=50;
       g2i.setColor(color);
       g2i.setBackground(Color.WHITE);
       g2i.clearRect(0,0,WIDTH,HEIGHT);
       g2i.setStroke(new BasicStroke(stroke));
       setPreferredSize(new Dimension(WIDTH, HEIGHT));
      //setBackground(Color.GRAY);
   }

    public void move_forward (Graphics2D g2d, int val) {
        int x0, y0;
        int x1, y1;
        int ctrlx,ctrly;
        int ctrlthglobal=(thglobal+angle)%360;
        int deltax = 0, deltay = 0;

        x0 = xglobal;
        y0 = yglobal;
```

```
            deltax = (int)(val* Math.cos((thglobal)*Math.PI/180));
            deltay = (int)(val* Math.sin((thglobal)*Math.PI/180));

            x1 = x0 + deltax;
            y1 = y0 - deltay;

            //g2d.setPaint(Color.MAGENTA);
            g2d.setColor(color);
            g2d.setStroke(new BasicStroke(stroke));
            g2i.setColor(color);
            g2i.setStroke(new BasicStroke(stroke));


             if(erase==1)
             {
                   g2d.setColor(color.WHITE);

             }

            if(state==0)
            {
             if(angle==0)
             {
                   g2d.drawLine(x0, y0, x1, y1);
                   g2i.drawLine(x0, y0, x1, y1);
             }
             else if(angle==90)
             {

                   g2d.drawArc(((x1+x0)/2-val/2),((y1+y0)/2-
val/2),val,val,thglobal,180);
                   g2i.drawArc(((x1+x0)/2-val/2),((y1+y0)/2-
val/2),val,val,thglobal,180);
                   //System.out.println("The cycle is :"+((x1+x0)/2-
val/2)+","+((y1+y0)/2-val/2)+","+val+","+val+","+((thglobal)*Math.PI/180));

             }
             else if(angle==-90)
             {
                   g2d.drawArc(((x1+x0)/2-val/2),((y1+y0)/2-
val/2),val,val,thglobal+180,180);
                   g2i.drawArc(((x1+x0)/2-val/2),((y1+y0)/2-
val/2),val,val,thglobal+180,180);

             }
             else
             {
                   int ctrlval = (int)(
((float)percentage*(float)val/100)/(Math.cos((angle)*Math.PI/180)));
                    ctrlx =x0+ (int)(ctrlval*
Math.cos((ctrlthglobal)*Math.PI/180));
                    ctrly =y0- (int)(ctrlval*
Math.sin((ctrlthglobal)*Math.PI/180));


                   g2d.draw(new QuadCurve2D.Float((float)x0,
(float)y0,(float)ctrlx,(float)ctrly,  (float)x1, (float)y1));
```

```
                g2i.draw(new QuadCurve2D.Float((float)x0,
(float)y0,(float)ctrlx,(float)ctrly,  (float)x1, (float)y1));
            }
           }
           xglobal = x1;
           yglobal = y1;
           try
           {

           Thread.currentThread().sleep(speed);


      }
      catch(Exception E)
      {

      }
    }//move_forward

    public void move_backward (Graphics2D g2d,  int val)
    {

       move_forward(g2d,  -val);

   }//move_backward


    public void turn_clockwise (Graphics2D g2d, int val)
    {
       thglobal = (thglobal - val) % 360;
       //System.out.println(thglobal);

   }//turn_clockwise


   public void turn_anticlockwise (Graphics2D g2d,  int val)
   {
       thglobal = (thglobal + val) % 360;
       //System.out.println(thglobal);
   }//turn_anticlockwise




       public void paintComponent (Graphics g) {

           super.paintComponent(g);

           xglobal = 300;    //starting point (x, y)
           yglobal = 300;

           thglobal = 0;

           Graphics2D g2d = (Graphics2D)g;

           turn_anticlockwise(g2d,  90);
```

```
        g.drawImage(bufferimage,0,0,WIDTH,HEIGHT,this);


     }// end of paintComponent

    }
```

```
/*
   MirageApp.java
   Ming Liao  (ml2288@columbia.edu)
   Nalini Vasudevan (nv2144@columbia.edu)

*/

import java.io.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

import javax.swing.*;

public class MirageApp extends MirageDisplay
{
    //final static int WIDTH = 600;
    //final static int HEIGHT = 600;

    JFrame theFrame;
    MirageDisplay panel;


    public MirageApp( )
    {

        theFrame = new JFrame("Mirage Application");

        panel = new MirageDisplay();
        theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        theFrame.setResizable(false);
        theFrame.setBackground(Color.WHITE);
        theFrame.getContentPane().add(panel);



         //g2i.drawString("Hello BufferedImage", 50, 50);
         //System.out.println("hahaha");



        theFrame.pack();
        theFrame.setSize(new Dimension(WIDTH, HEIGHT));

        theFrame.show();
```

```
        //set look and feel
        try
        {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }


    }

    void forward (int val)
    {
        Graphics g = panel.getGraphics();

        Graphics2D g2d = (Graphics2D) g;


        panel.move_forward(g2d,  val);
   }


    void backward (int val)
    {
        Graphics g = panel.getGraphics();

        Graphics2D g2d = (Graphics2D) g;


        panel.move_backward(g2d,  val);
   }

    void clock (int val)
    {
        Graphics g = panel.getGraphics();

        Graphics2D g2d = (Graphics2D) g;


        panel.turn_clockwise(g2d,  val);
   }
    void aclock (int val)
    {
        Graphics g = panel.getGraphics();

        Graphics2D g2d = (Graphics2D) g;


        panel.turn_anticlockwise(g2d,val);
   }
```

```
    void penup ()
    {
     panel.state =1; //up means 1
   }

    void pendown ()
    {
     panel.state =0; //down means 0
   }

    void eraseon ()
    {
     panel.erase =1; //on means 1
   }

    void eraseoff ()
    {
     panel.erase =0; //off means 0
   }



    public void save(String file)
    {
               try{

                     ImageIO.write(bufferimage , "PNG", new File(file));

                     //g2i.setBackground(Color.BLACK);


                     //panel.paintComponent(g2i);
               }
               catch(Exception E)
               {

               }
    }

    public void reset()
    {
         panel.xglobal=300;
         panel.yglobal=300;
    }

    public void load(String file)
    {
         File path=new File("");
         String filepath =new String(path.getAbsolutePath()+"\\"+file);
        Graphics g = panel.getGraphics();

        Graphics2D g2d = (Graphics2D) g;
         try{

               g2d.drawImage(ImageIO.read(new File(filepath)), 0, 0,
this);
```

```
                g2i.drawImage(ImageIO.read(new File(filepath)), 0, 0,
this);

          }
          catch(Exception E)
          {

          }


       System.out.println("Loading"+filepath);

    }

    public void color(int R,int G, int B)
    {

        panel.color = new Color(R%256, G%256, B%256);

    }

    public void thickness(int thickness)
    {

        panel.stroke = thickness;

    }

    public void speed(int speed)
    {
       speed = speed*20;
         if(speed>1000)
         {      panel.speed =0;}
         else if(speed<=0)
         {
               panel.speed=1000;

         }
         else
         {
        panel.speed = 1000-speed;
         }
    }

    public void setcurve(int angle, int percentage)
    {
        if(angle==0)
        {
         panel.angle=0;
         return;
        }
        if(angle%90==0)
        {
         if(angle>0)
         {
         panel.angle=90;
         }
```

```
            else
            {
                  panel.angle=-90;
            }
           }
           else
           {
            panel.angle=angle%90;
           }
            panel.percentage=percentage;
       }

   }



   /*
      MirageFunction.java
      Nalini Vasudevan (nv2144@columbia.edu)

   */

import antlr.collections.AST;
import java.util.*;

public class MirageFunction
{

      String args[];
      AST body;
      int ret_type;
      String name;
      static Hashtable functions=new Hashtable();;
      static Vector temp_vector=new Vector();

      public MirageFunction(String name,  AST body, int return_type)
      {
       this.name= name;
       this.body = body;
       this.ret_type= return_type;
      }

      public static void registerOneArgument(String arg)
      {

            temp_vector.addElement(arg);

      }

      public static void addReturnValue(Integer return_val)
      {

            temp_vector.addElement(return_val);

      }
```

```
	public void registerArguments()
	{
		args = new String[temp_vector.size()];
		for(int i=0;i<temp_vector.size();i++)
		{

		args[i]= (String)temp_vector.elementAt(i);
		}

		//Clear the vector for next use
				temp_vector.removeAllElements();
	}

	public static void setValueOfOneArgument(int value)
	{
		temp_vector.addElement(new Integer(value));
	}

	public void allocateAndSetArguments(Env env)
	{
		if(temp_vector.size()!= args.length)
		{
			System.out.println("Wrong no. of arguments");
			System.exit(0);
		}

		for(int i=0;i<args.length; i++)
		{

			//should not allow something like fun(int a, int a);
			if(env.check_in_present_scope(args[i])==true)
				{
				System.out.println("Variable "+ args[i]+" already in
scope");
				System.exit(0);
				}
			  int value =
((Integer)temp_vector.elementAt(i)).intValue();

				env.add_in_present_scope(args[i], value);
		}
		//		Clear the vector for next use
		temp_vector.removeAllElements();

	}

	/*Return type is 0 for void and 1 for int*/
	public static void funcRegister(String name,  AST body, int return_type
) throws antlr.RecognitionException
{
		if(functions.get(name)!= null)
		{System.out.println("Function "+ name+ " aleady defined");
		 System.exit(1);
		}
		MirageFunction fun = new MirageFunction( name, body,
return_type);
		fun.registerArguments();
```

```
            functions.put(name, fun);
      }

      public static int funcCall( MirageTreeParser walker, String name, Env
env)
      {
            int ret_value=0;
            try
            {

                  MirageFunction fun = (MirageFunction)functions.get(name);

                  if(fun == null)
                  {
                        System.out.println("Function "+name+" not defined.");
                        System.exit(1);
                  }

                  fun.allocateAndSetArguments(env);
                  walker.subprogram( fun.getBody() ) ;

                  Integer temp= (Integer)temp_vector.elementAt(0);
                  //Get the return value from temp_vector
                  if(fun.ret_type==0 && temp!=null || fun.ret_type==1 &&
temp==null)
                  {
                        System.out.println("Return type of the function
"+name+ " does not match");
                        System.exit(1);
                  }
                  ret_value= (temp!=null)? temp.intValue():0;
                  temp_vector.removeAllElements();

            }
            catch(Exception e){}
            return ret_value;
      }

      public AST getBody()
      {
            return body;
      }
}
```

A.3 Testing Cases

```
/*
   animation.mrg: testing graphical animation with control flow and
   recursive function calls
   Nalini Vasudevan (nv2144@columbia.edu)

*/

void sier (int n, int a, int h, int k)
{

   if(n==0)
   {
         fwd k;
         return;
   }

   rgt a;
   sier( n-1, -a, h, k);
   lft a;
   fwd h;
   lft a;
   sier(n-1, -a, h,  k);
   rgt a;
}

void one_square()
{
   int i, h=5;
   for(i=0; i<4; i=i+1)
   {
         sier( 7, 45, h, 10);
         rgt 45;
         fwd h;
         rgt 45;
   }
}

int eight_squares(int flag)
{
   int  j;


   for(j=0;j<8; j=j+1)
   {
         rgt 45;
         if(flag==1)
         rgt 45;
         one_square();

   }

}
```

```
main()
{
   color <180,20,0>;
   eight_squares(0);
   eraseon;
   eight_squares(1);
   eraseoff;
   color <20,180,180>;
   eight_squares(0);
   eraseon;
   eight_squares(0);
   eraseoff;
   save "sierp.jpg";
}
```

```
/*
   colorfulball.mrg: testing curve, color, turn clockwise, forward,
   backward graphic statements
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{

/* Move to the center of the Frame*/
down;
rgt 180;

int a=1;
int i;



/* Draw the curves*/
for (i=30;i<=180;i=i+1 )
{
   color <15*i,i,50*i>;
   curve <90,50>;
   rgt 10;
   fwd a*i;
   rgt 180;
   backward a*i;
}

save "ColorfulBall.jpg";

}
```

```
/*
   curve.mrg: testing curve statement
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{

int a=80;

up;

rgt 90;
fwd 500;
down;

lft 90;

/*Default: Draw a line*/
fwd a;

rgt 90;
/*Draw a curve with offset 80%, 30 degrees angles*/
curve <30,80>;
fwd a;

/*Draw a curve with offset 50%, 45 degrees angles*/
curve <45,50>;
fwd a;

/*Draw a curve with offset 50%, 45 degrees angles*/
curve <60,80>;
fwd a;

/*Draw a curve with offset 50%, 85 degrees angles*/
curve <85,50>;
fwd a;

/*Draw a half cycle*/
curve <90,0>;
fwd a;

/*Draw a line, return default mode*/
curve <0,0>;
fwd a;

save "curve.jpg";
}
```

```
/*
   flower.mrg: testing mathematical operators and thickness statements
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{

   int a=5, c=90, i=1;

   /*Define the shape of line: Half Cycle*/

   curve <90,0>;

   for(i=0; i<50; i=i+1)
   {
         color <255-2*i,100+2*i,0>;
         thick i/3;
         rgt c;
         fwd a*i;
   }

   save "flower.jpg";
}
```

```
/*
   dragon.mrg: testing recursive function calls
   Nalini Vasudevan (nv2144@columbia.edu)

*/

void dragon(int n, int a, int h)
{
   if(n < 1)
   {
         fwd h;
         return;
   }
   dragon (n - 1, 90, h);
   rgt a;
   dragon (n - 1, -90, h);
}

main()
{
color <180, 0, 0>;
dragon( 11, 90, 6);
save "dragon.jpg";
}
```

```
/*
   for.mrg: testing for loop with other graphical statements
   Abhilash I (ai2160@columbia.edu)

*/

main()
{

int a=250;
int c=90;
int i;

up;

fwd 300;

down;
for (i=1;i<=4;i=i+1 )
{
   rgt c;
   fwd a;

}

}
```

```
/*
   hilbert.mrg: testing user-defined function and recursive function
   Nalini Vasudevan (nv2144@columbia.edu)

*/
void hilb (int n, int a, int h)
{
   if (n == 0)
         return;

   rgt a;
   hilb  (n - 1, -a ,h);
   fwd h;
   lft a;
   hilb (n - 1, a, h);
   fwd h;
   hilb (n - 1, a, h);
   lft a;
   fwd h;
   hilb (n - 1, -a ,h);
   rgt a;
}

main()
{
color <180, 0, 0>;
hilb( 5, 90, 5);
save "hilbert.jpg";
}
```

```
/*
   koch.mrg: testing user-defined function and recursive function calls
   Nalini Vasudevan (nv2144@columbia.edu)

*/

  void snowflake (int side, int depth)
  {
     if (depth ==0)
     {
         fwd side;
         return;
     }

     snowflake(side/3, depth-1);
     lft 60;

     snowflake(side/3, depth -1);
     rgt 120;

     snowflake(side/3, depth-1);

     lft 60;
     snowflake(side/3, depth-1);

  }

 void flake (int s, int d)
 {
   int i=0;

         for(i=0;i<3;i=i+1)
         {
               snowflake(s, d);
               rgt 120;
         }
 }

 main()
 {
   int i,n;
   color <50,50,180>;
   flake(200, 3);
   save "koch.jpg";

 }
```

```
/*
   sierp.mrg: testing user-defined function and return statement
   Nalini Vasudevan (nv2144@columbia.edu)

*/

void sier (int n, int a, int h, int k)
{

   if(n==0)
   {
         fwd k;
         return;
   }

   rgt a;
   sier( n-1, -a, h, k);
   lft a;
   fwd h;
   lft a;
   sier(n-1, -a, h,  k);
   rgt a;
}


main()
{
   int i;
   color <0,0,180>;
   int h=5;
   for(i=0; i<4; i=i+1)
   {
               sier( 7, 45, h, 10);
               rgt 45;
               fwd h;
               rgt 45;
   }

   save "sierp.jpg";
}
```

```
/*
   simple.mrg
   Ming Liao  (ml2288@columbia.edu)

*/

var c=20,d;
d=90;
fwd c;
rgt d;
fwd c;
rgt d;
fwd c;
```

```
rgt d;
fwd c;
rgt d;
fwd c;
end;
```

```
/*
   test1.mrg: testing break, save, speed statement
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{

thick 3;
int a=5;
int c=90;
int i=1;
up;
rgt 45;
fwd 350;
down;


while (1)
{
  if(i<60)
  {
    speed 20*i;
   rgt c;
   fwd a*i;
   color <0,i*15,0>;
   thick i;

   i=i+1;
  }
  else
  {
     break;
  }

}

save "1.jpg";
}
```

```
/*
   test2.mrg: testing load and save statement
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{
load "1.jpg";
int a=5;
int c=1;
int i=1;
up;
fwd 250;
down;
while (i<500)
{
   rgt c;
   fwd a;

        i=i+1;
}

save "2.jpg";
}
```

```
/*
   test3.mrg: testing forward statement and function main
   Ming Liao  (ml2288@columbia.edu)
*/

main()
{
   fwd 30;
}
```

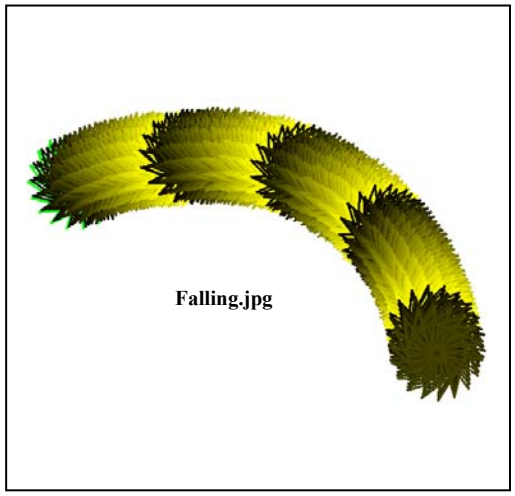
```
/*
   while1.mrg: tesing while loop
   Abhilash I (ai2160@columbia.edu)

*/

main()
{
   int a=5;
   int c=1;
   int i=1;

   up;

   fwd 250;
   down;
   while (i<500)
   {
```

```
        rgt c;
        fwd a;

        i=i+1;
   }

}
```

```
/*
   while2.mrg: testing while and if/else flow control
   Peili Zhang (pz2128@columbia.edu)

*/

main()
{
   int a=5;
   int c=90;
   int i=1;
   up;
   rgt 45;
   fwd 350;
   down;
   while (1)
   {
        if(i<50)
        {
        rgt c;
        fwd a*i;
        i=i+1;
        }
        else
        {
              break;
        }

   }

}
```

```
/*
   falling.mrg: testing for for / animation
   Nalini Vasudevan (nv2144@columbia.edu)
*/

void squaggle()
{
   int i;
   for(i=0;i<18;i=i+1)
   {
        fwd 50;
        rgt 150;
        fwd 60;
```

```
        rgt 100;
        fwd 30;
        rgt 90;
    }
}

main()
{
up;
lft 65;
fwd 280;
rgt 130;
down;
int i;
    for(i=0;i<110;i=i+1)
    {
        color <10*i, 255+10*i, 0>;
     squaggle();
     up;
     fwd 7;
     rgt 1;
     down;

    }

    save "falling.jpg";

}
```



Falling.jpg

```
/*
   peano.mrg: testing for recursion
   Nalini Vasudevan (nv2144@columbia.edu)
*/

void pean(int n, int a,  int h)
{
   if (n == 0)
        return;
   rgt a;
   pean(n - 1, -a, h);
   color<10, 255- n*20, n*30>;
   fwd h;
   pean (n - 1, a, h);
   color<255- n*20, 10, n*30>;
   fwd h;
   pean (n - 1, -a, h);
   lft a;
}

main()
{
   up;
```

```
   bwd 200;
   lft 45;
   down;
   pean( 6, 90, 12);
   save "peano.jpg";
}
```

```
/*
   tick.mrg: testing for while / animation
   Nalini Vasudevan (nv2144@columbia.edu)
*/

void spiral(int size, int angle, int my_color)
   {
   if (size > 100)
         return;
    if(my_color)
    color <125, 0, 0>;
    else
    color <255, 14+size, my_color> ;
    fwd size+20;
    rgt angle;
    spiral (size + 2, angle, my_color);
    }

 main()
   {

   int i, j=0;

   thick 4;
   while(j<99)
   {
         int mod= j%9;
         j= j+1;
         reset;
         up;
         bwd 100;
         lft 120;
         down;
         for(i=0; i<9; i=i+1)
         {
               if(i==mod)
                     spiral(0, 91, 1);
               else
                     spiral(0, 91, 0);
         }
   }
   save "tick.jpg";
}
```