Mirage

Language Reference Manual

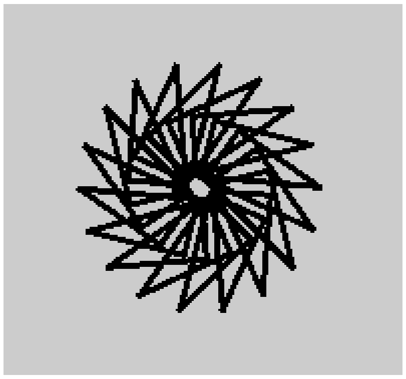
Image drawn using Mirage 1.1

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**

1.Lexical Conventions

1.1 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.

1.2 Comments

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

1.3 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.

1.4 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	float	string	const
up	down	fwd	bwd	lft	rgt
color	fill	thick	speed		
open	close	load	save		

1.5 Constants

Constants in the Mirage language are integer, float, and string.

1.5.1 Integer Constants

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

1.5.2 Floating Constants

A floating constant consists of an integer part, a decimal point, a fraction part, an e, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing.

1.6 Strings

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

1.7 Other Kinds of Tokens

Each token will be described in detail in the following sections.

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

2. Types

Mirage supports five fundamental types: integers, floating point numbers and strings. Variables are bound to a type during their declarations.

- int: is represented by 16-bit integers;
- float: has magnitude in the range approximately e^10±38 or 0; the precision is 24 bits or about seven decimal digits;
- string : a list of characters
- void: used for functions that do not return any values;

Mirage does default typecasting to the higher order type. For example, suppose an integer number and a float number are added, the result is of type float.

3. Expressions

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

3.1 Primary Expressions

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

3.1.1 Identifiers

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

3.1.2 Constant

An integer, a float or a string is a primary expression with the type int, float, or string.

3.1.3 Integers, Floats, Strings

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

3.1.4 (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.

3.1.5 Primary-expression < expression-list >

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

3.1.6 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.

3.2 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 and float as operands.

3.2.1 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 or float.

3.2.2 Binary Operators

3.2.2.1 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 or float. / yields the quotient of the division of the first operand by the second; if the second operand is 0, the result is undefined.

3.2.2.2 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 or float.

3.2.2.3 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 var or float.

3.2.2.4 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.

3.2.2.5 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.

3.2.2.6 Assignment Operator

expression = expression

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

3.3 Other Operators

Dot '.' is used for floating point numbers. 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.

3.4 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

4. Declarations

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

4.1 Variable Declaration:

vardeclaration:
    typespecifier declarator-list ;

4.1.1 Type Specifiers

- *var*
- *float*
- *string*
- *const*
- *void*

4.1.2 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:
    var | float | string

4.2 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
```

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.

5.1 Expression statement

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

5.2 Compound statement

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

5.3 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.

5.4 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.

5.5 Break statement

> *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.

5.6 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.

5.7. 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 lower left corner of the window (at position [0, 600]). If the programmer doesn't move the pen from the starting position before his actual drawing, any graphics is drawn at position [0, 560], so that the display of the base of the graphics is not being obscured by the window (such as it would happen after drawing a square, the base line of the square will be shown).

5.7.1 Colors

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

> *color <R, G, B>*
> *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.

> *fill <R, G, B>*
> *FILL <R, G, B>*

Same as Color, Fill follows the RGB color model and is used to fill the color of any shape.

5.7.2 Graphics Motion

> *fwd dist*
> *FWD dist*

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

> *bwd*
> *BWD dist*

moves the pen backward, i.e., exactly opposite to the direction that it's facing, by the specified distance (Units are in pixels).

> *lft degrees*
> *LFT degrees*

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

> *rgt degrees*
> *RGT degrees*

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

5.7.3 Pen State

> *up*
> *UP*

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

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

thick val
THICK val
define the thickness of the pen in the integer range [1, 10].

speed val
SPEED val
define the speed of the pen in the integer range [1, 10]

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.

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 a file.

save   filename;
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 .gif, .png, .jpg and .jpeg.

8. Sample Mirage Program

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++)
     {
          forward 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 requied position*/
     if(a<20)
     forward 3*a;
     else
     forward 2*a;

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

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

 }
```

8.2

```
/* Code to draw a squaggle (the figure on the first page of the
manual)*/

void main()
{
     int i;

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

     }

}
```

9. References

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

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