Photoshop--

Language Reference Manual

Programming Languages & Translators
Columbia University, Fall 2014
Professor Edwards

Gilbert Feig (grf2108)
David Figueroa (df2442)
Alana Ramjit (amr2235)

Table of Contents

Language Reference Manual

1. **Lexical Conventions**
    1.1. **Tokens**

    There are three main categories of tokens not mentioned in the other lexical conventions: whitespace, block separators, and semicolons. Whitespace includes the tab, newline, and space characters. Block separators are the ‘{‘ and ‘}’ symbols that enclose the component statements of a block. Semicolons indicate the end of an expression, and also indicate that the expression is a statement.

    Whitespace is used to separate tokens which can be identifiers, keywords, constants, operators, and comments.

    1.2. **Comments**

    Comments are strings that are ignored by the compiler. Indicate the start with a single ‘~’ character. Comments may be several lines in length, and are terminated by another single ‘~’ character.

    1.3. **Identifiers**

    Identifiers are a series of letters and/or digits, always beginning with a letter. The maximum length is 20 characters.

    1.4. **Keywords**

    Keywords are reserved for special use cases, and may not be used as identifiers or anything else unintended. These consist of:

`at`	`green`	`rect`
`block`	`if`	`red`
`blue`	`int`	`right`
`bool`	`left`	`run`
`down`	`loop`	`true`
`ellipse`	`main`	`up`
`else`	`move`	
`false`	`put`	

    1.5. **Constants**

    The only constants supported are base decimal integer constants. All integers are signed and may be stored in variables of type int only.

2. **Syntax**
    1.1. **Basic Types**

    There are four fundamental types: `bool`, `int`, `rect`, and `ellipse`.
    The `bool` type may only take values `true` and `false`.
    The `int` type may take any signed integer value.
    The `rect` and `ellipse` types have the following properties:

- `x` - the `x` coordinate of the top left corner of the containing frame (defaults to 0)
- `y` - the `y` coordinate of the top left corner of the containing frame (defaults to 0)
- `width` - the width of the shape
- `height` - the height of the shape
- `color` - the color of the shape

3. Expressions

1.1. Operators

i. Multiplicative Operators

The multiplicative operators are * and / and group from left-to-right.
multiplicative-expression:
*multiplicative-expression * int*
multiplicative-expression / int
The operands of * and / must be of type `int`.
The * operator denotes multiplication and returns a product of the operands as an `int`.
The / operator denotes division. If the divisor does not equally divide the dividend, then the integer quotient is returned.

ii. Additive Operators

The additive operators are `+` and `-` and group from left-to-right.
additive-expression:
multiplicative-expression
additive-expression + multiplicative-expression
additive-expression - multiplicative-expression
The + operator denotes addition and returns the sum of the operands.
The - operator denotes subtraction and returns the difference of the operands.

iii. Relational Operators

The relational operators <, <=, >, >= evaluate to either true or false and group left-to-right such that `x>y>z` is parsed as `(x>y)>z`.
relational-expression:
additive-expression
relational-expression < additive-expression
relational-expression <= additive-expression
relational-expression > additive-expression
relational-expression >= additive-expression
The operators `<` (less), <= (less or equal), > (greater), and >= (greater or equal) return true if the relation is true and false otherwise. The return type is of type bool.

iv. Equality Operators

The equality operators == and != evaluate to either true or false
equality-expression:

relational-expression
equality-expression == relational-expression
equality-expression != relational-expression

The operators == (equal) and != (not equal) return true if the equality comparison is true and false otherwise. The return type is of type bool.

v. Animation Operator

The animation operators `move` (`left`, `right`, `up`, and `down`) and `put at` modify the `x` or `y` position of objects.

animation-expression:
move identifier left additive-expression
move identifier right additive-expression
move identifier up additive-expression
move identifier down additive-expression
put identifier at additive-expression additive-expression

The animation operators change the location of objects. The `move` operator changes the `x` position of an object with the keywords `left` and `right` and changes the `y` position of an object with the keywords `up` and `down`. The `put` operator changes both the `x` and `y` position of an object to the position x,y after `at`.

1.2. Function Calls

A function is called by calling `run` followed by the name of the function.
ex. *run myFunction; ~function named myFunction is being called~*

4. Declarations

Declarations of an identifier can be associated with one of the four basic types or a function. All declarations may be accompanied by an a definition of an initial value; if no initial value is provided, a default value will be provided. All variables and functions must be declared before they are referenced.

1.1. Declarations of int and bool

Identifiers of type int are declared as int <identifier> with an optional assignment to a constant integer value. **bools** are declared in the same manner but are optionally assigned to either false, true, or the value of some boolean expression. If the optional assignment is neglected, the default constant for an int is **0** and the default value for a bool is **false**.
ex: int x = 10; ~declares an int of value 10~
bool isBool; ~declares a boolean with an initial value of false.

1.2. Declarations of shape objects

Shape objects such as rect and ellipse are declared in a similar format of <type> <identifier>. They may optionally be assigned, in order, a height, a width, and a color. If the optional assignment is neglected, the default values are **10, 10 and blue.** These properties are separated by white space:
rect r1 = 10 15 green; ~ declares a green rectangle of height 10, width 15~

1.3. Declarations of functions are specified by the keyword “block” followed by an identifier and curly braces containing the group of statements associated with that block. Statements are discussed in section 5.1 of this reference manual.

5. Statements and Execution

1.1. Statements and Expressions

An expression is a syntactically valid variable declaration, boolean or relational evaluation, arithmetic expression, function call, or animator operation on a shape as discussed in section 3.

A statement is any expression that is terminated with a semi-colon. Expressions such as variable declarations, function calls, and animator operations must always end with a semi-colon and are always statements. Relational evaluations, or arithmetic expressions may evaluated as part of a declaration or as a condition within an **if** block. **if, block,** and **loop,** are all followed by braces that must group together a set of statements.

1.2. Execution

Execution begins at the top of the **main** block. The **main** block may also include the special **loop** block. All statements within the **loop** block will be executed continuously at a rate of 60 frames per second, enabling animation simulation.

All variables and blocks must be declared before they are referenced in a non-declarative statement following it in the execution path.

6. Scope

Any variable declared within a block, understood to mean a group of statements between braces, is only visible within those braces. If a block is nested within another block, and a variable is declared with the same name in the inner block as a variable in the outer block, then the inner block copy takes precedence and the outer block copy is rendered invisible.

Global variables are visible within any block but must be declared at the beginning of the file before any block declarations.