Martina Atabong | maa2247
Charvinia Neblett | cdn2118
Samuel Nnodim | son2105
Catherine Wes | ciw2109
Sarina Xie | sx2166

The Warhol Language Reference Manual

Introduction

Warhol is a functional and imperative programming computer language based on both Java and Matlab. Warhol's types, syntax, and semantics are meant to help the user easily manipulate images. Images are uploaded from files and translated into our main primitive type, matrices. Our primitive types are designed to store pixel data. Warhol allows frequent writes, reads, and computes of pixels. Built-in functions are provided to compute commonly used image algorithms while also giving users freedom to implement functions.

The Warhol Library contains methods to facilitate declarative programming. Library functions provide standard implementations of image editing task that can be performed on matrix types.

Lexical Conventions

Tokens consists of identifiers, constants, separators, keywords, strings, and expression operators. Comments and tabs are completely ignored. Blanks are new lines are used to separate tokens.

Comments

- Comments are denoted by opening and closing dollar signs.
    - Example 1: $ this is my code $
    - Example 2: $ this is also my code
      $ and this as well $

Identifiers

- Identifiers are a sequence of at least one character and a number. The first character of an identifier must be alphabetic. Identifiers are case sensitive and cannot begin with an uppercase letter. Longest string search is used to determine if an identifier is identical to a key word.

Keywords

int	mat

bool	if
else	for
while	break
continue	sizeof
void	null
fun	char

Constants

- Integers
  - Integers constants are sequence of at least one digit.
- Characters
  - Character constants are one character enclosed with single quotes. Two character character constants are allowed only if the first character is a backslash. Characters are treated as integers.
    - Example: '\n', 'a', '\t', '0'
- Strings
  - Strings are implemented as a matrix of characters. The matrix is always the exact length of the string. Termination of string is determined by bounds of the matrix containing a string's characters.

Types

Matrix

- Matrices are 1, 2 and 3 dimensional arrays that can contain other objects defined in the Warhol language.

Functions

- Functions are procedures that return void or methods that return other objects.

Integers

- Integer types are declared using keyword *int*

Characters

- Characters are chosen from ASCII set and declared using keyword char. Characters can also be read as a number.

Boolean

- Booleans are declared using keyword bool. Booleans are stored and equivalent to integers equal to 0 for false and greater than 0 for true.

Scope

- No variables of the same name, Ocaml style in the sense of immediate globalization (by C definition, automatic)

Conversions

Characters and Integers

- Certain operators perform implicit conversions from one type to another such as characters. Characters can be used the same as integers.

Expressions

Primary Expressions

Primary expressions are literals and names. Primary expressions involving subscripting and function calls group left to right.

- An identifier is a primary expression. Its type is specified by its declaration (ex: matrix, function)
- A decimal or character constant is a primary expression. Its type is int.
- A string is a primary expression. Its type is matrix of chars.
- (*expression)*

A parenthesized expression is a primary expression. Its type and value are identical to the expression inside the parentheses.

- *primary-expression [expression]*

A primary expression followed by an expression in square brackets is a subscript and is a primary expression.

- *primary-expression (expression-list* (optional) *)*

A function call is a primary expression that is immediately followed by parentheses containing the list of its arguments. It is of type “function returning ...” and the result of the function call is of type “...”.
Any arguments of type char are converted to type int before the call.
Parameters are passed by reference.

Equality Operators

- Is-equal-to
  - *expression_a == expression_b*
- Is-not-equal-to
  - *expression_a != expression_b*

Relational Operators

The operators < , > , <=, and >= all yield 0 if the specified relation is false, and 1 if it is true.

- Is-less-than
  - *expression_a < expression_b*
- Is-greater-than

    - *expression_a > expression_b*
- Is-less-than-or-equal-to
    - *expression_a <= expression_b*
- Is-greater-than-or-equal-to
    - *expression_a >= expression_b*

Math Operators

If both operands are int or char, the result is int. If one is int or char and the other is mat, then the result is mat. If both operands are mat, then the result is mat. No other combinations are allowed.

- Addition
    - *expression_a + expression_b*
        - The result is the sum of the expressions.
- Subtraction
    - *expression_a - expression_b*
        - The result is the difference of the expressions.
- Division
    - *expression_a / expression_b*
        - The binary / operator signifies division.
- Multiplication
    - *expression_a * expression_b*
        - The binary * operator signifies multiplication.

Unary Operators

Any expression with a unary operator is read from left to right.

- *! expression*
    - This negation operator returns 1 if the value of the expression is 0, 0 if the value of the expression is non-zero. The type of the result is bool. This operator is only applicable to bools.
- *- expression*
    - The result is the negative of the expression. This operator is only applicable to ints and chars.
- *sizeof expression*
    - Returns the size (bytes) of the operand.

Logical Operators

- *expression && expression*
    - The && (“and”) operator returns 1 if both its operands are non-zero. Otherwise, it returns 0.
- *expression || expression*
    - The || (“or”) operator returns 1 if either of its operands is non-zero, and 0 otherwise.

Assignment Operator

- *value = expression*

- The value of the expression replaces that of the object referred to by the value.

Declaration

Declarations in **Warhol** are used to establish particular types.

Declaration

- In Warhol there is one declaration max, per semicolon.

mat name = [x1 x2 x3; y1 y2 y3; z1 z2 z3];

- The scope of declaration binds the declaration to some declarator for later function application.

Declarators

- **declarator()** is a function declaration that applies a builtin function to the expression declaration statement

For example, **mat** in Warhol, is an example of how type declaration in Warhol is done via declarators in the declaration statement.

Statements

Expressions

- In Warhol there is one expression statement max, per semicolon.

Expression statements look like:

form *expression ;*

Compound Statement

- In Warhol, a compound statement consists of a list of statements where each statement ends in a semicolon.

Compound statements look like:

expression1 ; expression2 ; expression3 ;

Conditional Statement

- In Warhol, a conditional statement consists of a conditional keyword, such as **if**, followed by an expression, **in parentheses** and an expression statement.

Conditional statements look like:

keyword *(expression) statement;*

Break

- The **break** identifier terminates loops so it must be used in the context of a loop.

Break statements look like:

- *break;*

Continue

- The **continue** identifier skips one iteration of a loop so it must be used in the context of a loop.

Continue statements look like:

- *continue;*

Return

- The **return** statement returns the result of a function to the caller of the function.

Return statements look like:

- *return;*

Function

Declaration

```
type fun nameOfFunction (Declarator1, Declarator2, …) {
    <statement1>
    <statement 2>
    …
    return result;
}
```

Parameters

Default scope for any variable is local. If declared in the parameters of a function, the scope is the length of the function. All parameters are passed by value, leaving the original object passed into the function unchanged.

Built-in Functions

Upload

Takes in a image file format and returns an image object

Input

- string - name of the image file

Return Type

- Image - the image uploaded

Write

Takes an image object and writes it to a file

Input

- Image - file to be written to file
- String - name of file

Return Type

- Void - the function returns void

Print

Takes an image object and displays

Input

- Image - file to be written to file

Return Type

- Void - the function returns void

Standard Library

RevX()

Takes a matrix and reverses the given matrix along the columns

Input

- Mat - the original matrix

Return Type

- Mat - the same matrix with the columns reversed

RevY()

Takes a matrix and reverses the given matrix along the rows

Input

- Mat - the original matrix

Return Type

- Mat - the same matrix with the rows reversed

Concat()

Takes a matrix or image and concatenates the second matrix to the intended row/column. Types concatenating must match

Input

- Mat/image - the original matrix or image
- Mat/image - the matrix or image to be concatenated
- Direction - enumeration type of *LEFT, BOTTOM, RIGHT, TOP,* to choose where to concatenate

Return Type

- Mat - new concatenated matrices in one

Filter()

Takes an image and filters the image using a convolution matrix

Input

- Image - the image to be filtered
- Mat - the convolution matrix

Return Type

- Image - the filtered copy of the image

Blur()

Takes an image and blurs the image using basic sharpen matrix multiplication

Input

- Image - the image to be sharpened

Return Type

- Image - the blurred copy of the image

Sharpen()

Takes an image and sharpens the image using basic sharpen matrix multiplication

Input

- Image - the image to be sharpened

Return Type

- Image - the sharpened copy of the image