IML Reference Manual

Travis J. Galoppo
tjg2107@columbia.edu

1. Introduction

IML is a computer language, based loosely on the C and Pascal languages, designed for easy manipulation of RGB raster images. IML is an interpreted language, built on top of the Java platform, and is therefore platform independent.

At the time of this writing, absolutely no software is written in IML, as the interpreter is being developed in parallel with this reference manual. As such, the final language is subject to vary slightly from this version of the manual.

2. Lexical conventions

IML distinguishes between five kinds of tokens: identifiers, keywords, numeric constants, expression operators, and separators. Blanks, tabs, newlines, and comments (as to be described) are ignored, other than to separate tokens.

2.1 Comments

IML employs C style comments, starting with the characters /* and terminating with the characters */.

2.2 Identifiers

An identifier is a sequence of letters, digits, and underscores; identifiers must start with a letter (underscore is not allowed), and can be any length. Upper and lower case letters are considered different.

2.3 Keywords

The following identifiers are reserved as keywords, and are not valid user defined identifiers:

```
integer        if
real           else
string         for
null           while
return         program
end            do
image          pixel
var            function
main           print
```

2.4 Constants

There are both integer and real (floating point) constants:

2.4.1 Integer Constants

An integer constant is a sequence of digits. Integer constants are always interpreted as base 10. Future versions of the language will include hexadecimal notation, similar to that of C.

2.4.2 Real Constants

Real, or floating point, constants consist of a series of digits followed by a . followed, possibly, by more digits. For example: 1.24 1234.5 1. are all Real numbers. Note that a number of the form .25 is not. Also, there is no provision for C style "scientific notation".

2.5 Strings

A string is a sequence of characters surrounded by double quotes; strings containing double quotes must represent the double quote using two consecutive double quotes. For example "IML is ""fun"" to learn!" represents the string —> IML is "fun" to learn.

3. Types

IML has five declarable types: integer, real, string, pixel, and image.

Integer	- 32 bit signed integer value
real	- 64 bit signed floating point value
string	- Printable sequence of characters
pixel	- 32 bit value representing red, blue, and green color components; each channel uses only 8 bits, so 8 bits of this type are wasted.
image	- Essentially an array of pixels; however, this type is more of a class, with some methods associated to it.

4. Conversions

The following conversion are valid between types.

4.1 Integer and Real

Integers may be converted to reals without loss of accuracy; reals may be converted to integers, any fractional part will be lost. Behavior is undefined if the value of the real exceeds 2^31-1.

4.2 Integer and Pixel

Integers and pixels may be freely converted, as may be useful for manual masking and manipulation of color channels.

5. Expressions

Expressions consist of identifiers (including function calls) and operators. The following list of operators are defined in IML, from highest precedence to lowest:

() - ID	Grouping Unary minus Indentifier/Function call
! ~	Logical NOT Bitwise NOT
* / %	Multiplication Division Modulus

+ -	Addition Subtraction
<< >>	Bitwise shift left Bitwise shift right
> >= < <=	Greater than Greater than or equal Less than Less than or equal
== !=	Logical equality Logical inequality
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise OR
&&	Logical AND
||	Logical OR
=	Assignment

6. Declarations

Declarations of variables and functions are described in this section in a pseudo-ANTLR type notation; both are derived from Pascal syntax.

6.1 Variable Declarations

Variable declarations take the form of

var ID [**,** ID]* **:** type **;**

where ID is any valid identifier as described in section 2.2, and 'type' is any of the five declarable types described in section 3.

6.2 Function Declarations

Function declarations take the form of

function ID **(** (PARAM_LIST)? **)** **:** type

where PARAM_LIST takes the form of

PARAM (**,** PARAM)*

where PARAM is further defined as

ID **:** type

where ID is any valid identifier as described in section 2.2, and 'type' is any valid IML type as defined in section 3.

7. Statements

This section described the various statements permissible in IML.

7.1 Expressions

Expression statements, such as assignments or function calls, take the form of

expression **;**

7.2 Compound Statements

Sequences of statements can be grouped together to form a single logical statement by enclosing them in braces:

{ (statement **;**)* **}**

7.3 Conditional Statement

There are two forms of the conditional statement:

if (expression **) then** statement1

and

if (expression **) then** statement1 **else** statement2

In either case, 'expression' is evaluated, and if it is non-zero then 'statement1' is executed; in the latter case, if 'expression' evaluates to zero, then 'statement2' is executed. "Else ambiguity" is resolved by connecting the else with the nearest "elseless if".

7.4 While Statement

The while statement takes the form of

while (expression **)** statement

'expression' is evaluated at the beginning of each iteration, and statement is executed if the result is non-zero; iteration terminates when 'expression' evaluates to zero.

7.5 Do Statement

The do statement takes the form

do statement **while (** expression **) ;**

'expression' is evaluated at the end of each iteration, and statement is executed if the result is non-zero; iteration terminates when 'expression' evaluates to zero. 'statement' is guaranteed to execute at least one time.

7.6 For Statement

The for statement takes the form of

```
for ( expr1 ; expr2 ; expr3 ) statement
```

and is equivalent to

```
expr1;
while(expr2){
        statement;
        expr3;
}
```

7.7 Return statement

Functions return to their caller via the return statement, which takes the form

return (expression)? **;**

The value of expression will be returned to the caller, assuming the function is declared to return a value of matching type. It is an error for a function not declared as null to omit 'expression', as well as it is an error for a function declared as null to include 'expression'.

7.8 Print Statement

The print statement takes the form of

print expression (**,** expression)* **;**

Each expression will be evaluated and output on the same line.

8. Scope rules

IML allows for global variable declarations between the program statement and the first function declaration; these globals will be available to every function so long as such function does not declare a local variable by the same name. Variables declared within a function, including parameters, are local to that function and can not be accessed via any other function; such variables can have the same name as global variables, thereby making the global inaccessible within that function.

9. Program structure

The overall structure of an IML program is as follows:

program ID **;**
<global variable block>
<function definition block>
main()
end.

Program execution always begins at the first statement in the **main** function.

9.1 Example

The following sample program demonstrates how to invert an image

```
program InvertImage:

function InvertPixel(p : pixel) : pixel
{
        var np : pixel;
        np.red = 255 - p.red;
        np.green = 255 - p.green;
        np.blue = 255 - p.blue;
        return np;
}

main()
{
        var img : image;
        var j, k : integer;
        img = image::open("myimage.jpg");

        for(j=0; j<img.height; j = j + 1){
                for(k=0; k<img.width; k = k + 1){
                        img[k,j] = InvertPixel(img[k,j]);
                }
        }

        img.save("myimage_neg.jpg");
}

end.
```