YAIL LRM

Andrew Kisch (aik2113)
Aniket Phatak (avp2110)
Pranay Prabhakar (ppp2113)
Uday Chandrasen (uc2124)

Instructor –
Prof. Stephen A Edwards

COMS W4115 – Fall 2010

Contents –

1. Introduction

Image processing has been an integral part of computer science ever since signal processing and digitizing analog data became possible. Digitized images lend themselves perfectly to classical applications of Mathematical Transformations like Fourier, Laplace et al.

The applications of this fascinating field include Medical Imaging, Computer Vision, Biometrics, Computer Animation, Digital Cameras, Remote Sensing, Entertainment etc. Also, the fields of Artificial Intelligence like pattern matching etc. have found a new application with images. Moreover, we now live in a world where new art seems to be judged less and less on content and increasingly on groundbreaking originality. The ability to easily customize photo filters would appear to be an extremely powerful tool.

The entire process of Image Processing involves many stages, right from image acquisition, sampling, storing to processing and rendering. We are motivated to work on that part of Image Processing which involves the application of various transforms and computations in order to have the desired effect on a digital image.

2. Lexical Conventions

The various kinds of tokens in YAIL are as mentioned below. The tokens are separated by one or more of the following: comments, spaces, tabs or newlines. The separators are otherwise ignored. The tokens do not form unique prefixes. This means that a token is the longest substring of characters that could possibly be legal in the grammar defined.

2.1 Comments

Multi-line comments only. Comments start with /* and end with */. Everything else within t he comments is ignored.

2.2 Identifiers

An identifier is a name given for a type or function. It can be a sequence of one or more letters and/or digits. Identifiers must start with a letter, not a digit. Underscores and special characters are not permitted. Identifiers are case sensitive.

2.3 Keywords

YAIL reserves the following keywords for itself. They may not be used as identifier in any YAIL program:
int break continue float if else struct for return image color

2.4 Constants

2.4.1 Integer
Integer constants consist of decimal integers as a signed sequence of one or more digits.
2.4.2 Float
Float constants consist of signed decimal real numbers with one and only one decimalpoint. The integral part is optional but the fractional part is not.

2.4.3 String
String constants will be any sequence of characters within double quotes. To have a double quote within a string use the escape sequence ‘\’ followed by the double quote.

3 Data Types

Types define the “kind” of data that can be understood by YAIL. YAIL bases the interpretation of the identifiers based on their types. The following types are supported

3.1 int

32-bit signed integers are denoted by type int. Range of integer constants is -2147483648 to 2147483647, defaulting to 0.

3.2 float

64-bit signed decimal point numbers are denoted by type float. Default is 0.0.

3.3 string

Character strings are denoted by type string. Default is empty string “”.

3.4 image

2-dimensional matrix of colors. Its functions include: load, save, display, getWidth and getHeight. Access the individual pixels by: image_name[horizontal_value, vertical_value]. Be careful of ArrayIndexOutOfBounds errors.

3.5 color

4-tuple of integer values from 0-255, of the format [red, green, blue, alpha]. The value of alpha may be omitted during instantiation, defaulting to 0.

3.6 filter

NxM matrix of non-negative floats, representing an image filter. Declared by:
filter f = [[1.0,1.0,1.0],[1.0,1.0,1.0]];
where the innermost values ([1.0,1.0,1.0]) represent the values within a row, where each number is in a different column, and the inner brackets ([[],[], []]) represent the different rows, where each set of inner brackets is its own row.

3.7 User defined

Besides the basic types the user can also define the following: structs, arrays, functions.

3.7.1 Arrays

Users can define a collection of elements of the same type by declaring arrays. Arrays ensure contiguous memory allocation for the elements within the array. A general way to declare an array in YAIL is:
`type variablename[size].`
This will allocate memory for `size` number of variables of type `type.` `variablename` is any valid identifier.

3.7.2 Functions

All functions in YAIL will have the following declarations pattern:
`returntype functionname (args-list) {statements;}`
Here, `returntype` should be a valid YAIL type, `functionname` is a valid identifier, and `args list` should be a valid comma separated list of YAIL types. The `args-list` can be empty. The function body should contain a valid YAIL statement or a group of statements.

4 Conversions

YAIL is strictly typed and does not support any explicit type casting. However lower numerical types can be expanded into higher ones. Example: A type int can be assigned to a type float but not vice-versa.

5 Expressions

5.1 Primary expressions

A primary expression can be an identifier, any of the constants defined above, an expression contained in parentheses.

5.2 Unary operators

YAIL supports one unary operator for negation. It is denoted by a ‘-’ sign before an expression which negates the expression. Only int and float, image and color values can be negated. Negation of numeric values means multiplying the value by -1. Negation of an image means replacing the value of each pixel of the image with max(ColorSpace value) - pixel value. Negation of a color would be similar to negation of an image.

5.3 Multiplicative operators

The multiplicative operators *, /, and % group left to right.

5.3.1 Multiplication

expression * expression

The binary * operator indicates multiplication. An int can be multiplied with an int or a float. A float can be multiplied with a float or an int. A int and float multiplication results in a float value. No other combinations are allowed.

5.3.2 Division

The binary / operator indicates division. The same type considerations as for multiplication apply.

5.3.3 Modulus

The binary % operator yields the remainder from the division of the first expression by the second. Both operands must be an int and the result is int. The remainder has the same sign as the dividend.

5.4 Additive operators

The additive operators + and - group left to right.

5.4.1 Addition/Subtraction

An int can be added to an int or float. A float can be added to a float and int. The result of addition/subtraction involving float and int is a float.

5.5 Logical Operators

‘&&’ corresponds to logical AND. ‘||’ corresponds to logical OR. ’!’ corresponds to logical NOT. ‘&&’ and ‘||’ are binary operators whereas ‘!‘ is a unary operator. None of the logical operators are short circuiting.

5.6 Relational Operators

==, !=, < , <= , >, >= are all binary operators.

5.7 Image Operators

There are several Image operators defined in YAIL for quick and easy modification of images. These operators are listed below-

5.7.1 Predefined Filters

`image_identifier.grayScale()` – to convert a colored picture into a black and white (gray scale) image.

`image_identifier.medianFilter()` - to apply a median filter to an image using the default 3x3 neighborhood.

`image_identifier.contrastChange()` – enhances the contrast of an image by a default historgram.

`image_identifier.edgeDetection()` – to detect edges of an image

`image_identifier.threshold()` – All objects in the image will be denoted by blackand all the background in the image will be denoted as white.

5.7.2 Operators

`Image_name.flipImageVertical()` – to flip an image on the vertical axis.
`Image_name.flipImageHorizontal()` – to flip an image on the horizontal axis.
`Image_name @ (angle_value)` – to rotate the image by a given positive angle in degrees.
`Image_name # filter_name` – to find the convolution of an image and a filter.
`Image_name1 = Image_name2` – returns 1 if the images are same and 0 if they are not the same.

6. Function Definitions

A function in YAIL can be defined as: TYPE ID (args list){ statement }. The args list can be optionally empty.

7. Statements

Expressions followed by semi colons are statements in YAIL. They are executed in sequence.

7.1 Selection Statements

Selection statements evaluate conditions and direct control flow appropriately.

```
if ( expression ) statement-block
if ( expression ) statement-block else statement-block
```

7.2 For Loops

A valid for statement form is:

```
for ( expression-statement; expression-statement;expression-
statement ) statement-block
```

The first statement is evaluated before the loop begins, the second expression is evaluated at the beginning of each iteration and, if false, ends loop execution. The third statement is evaluated at the end of each iteration. Each expression can be multiple expressions separated by commas.

7.3. Break

The `break` statement allows the termination of the current `for` loop and takes execution to the statement immediately after the `for` loop.

7.4 Continue Statement

The `continue` statement can be used only within a `for` loop. When encountered, the remaining part of the for loop is ignored and the iteration execution goes to the condition evaluation of the for loop, possibly for the next iteration.

7.5 Compound Statements

Nested statements are permitted, such that selection and iteration statements can appear inside of a statement block. All statement blocks must begin with an open bracket and end with a close bracket.

8. Scope

8.1 Static Scoping

YAIL uses static scoping. That is, the scope of a variable is a function of the program text and is unrelated to the runtime call stack. In YAIL, the scope of a variable is the most immediately enclosing block, excluding any enclosed blocks where the variable has been re-declared.

8.2 Global vs. Local

Global variable: The variables declared outside of the function are global variables, which will be applied in the whole program except the function where there is a local variable with the same name as that of the global variable. Global variables will exist until the program terminates.
Local variable: The variables declared inside of the function are local variables, which will exist and be applied only inside that function.
Scope conflicts: If there is a global variable whose name is the same with that of the local variable, then the value of the local variable will be applied inside the function while the value of the global variable will be applied in all the other part of the program except that function.

8.3 Forward Declarations

YAIL requires forward declarations for variables and functions. That is, a variable needs to be declared before it can be referenced, and any function needs to be defined before it can be invoked.
For example, YAIL generally prohibits the following and will throw an error:

```
float a;
float b;
float mean;
mean = func(a, b);
...
```

In this case, the function func() needs to be defined before it is called.

8.4 Arithmetic Operator Overloading

Arithmetic operators (+, -, *, /) are overloaded in YAIL. They can be used in expressions where integers and floats are mixed, and where an image/filter is mixed with a scalar value.
The convolution operator (#), however, is not overloaded. It takes exactly an image on the left and a filter on the right, nothing else.

8.5 Function Name Overloading

YAIL does not allow function name overloading. That is, each function should have a unique function name, or YAIL compiler will complain. This helps make a YAIL program more readable and easier to understand, which are two important goals of the language.

8.6 Namespaces

YAIL has only one namespace. Functions, variables, types, and record names, all share the same namespace. For example, a variable foo and a function foo() cannot coexist in a YAIL program. This helps make a YAIL program more readable and easier to understand, which are two important goals of the language.