COLUMBIA UNIVERSITY
The Fu-Foundation School of Engineering and Applied Science

COMS W4115: Programming Languages and Translators
By Professor Stephen Edwards

Simplified Image Processing(sIP)

Language Reference Manual

Vaibhav Jagannathan (vj2192@columbia.edu)

Shubanshu Yadav (sy2511@columbia.edu)

Bhargav Sethuraman (bs2814@columbia.edu)

October 28, 2013

Contents –

1. Introduction

In recent times Image processing has become a very popular topic. A number of amateurs and dilettantes wish to try their hand at image processing. Some discouraging factors in the currently available image processing applications are the extremely expensive licenses and counter-intuitiveness of the programming languages.

Pictures are increasingly being used as an essential part of digital diaries and blogs, thanks to the widely prevalent digital cameras that started gaining popularity since the year 2000.

Furthermore, images need to be modified, rotated, tilted, cropped, dimmed as a part of computer animation. For certain applications such as remote sensing and medical imaging, an image needs to be sharpened, needs it's contrast increased, color filtered or size modified. Mobile cameras use perspective correction to correct human errors while capturing images and to stabilize videos. Image comparison applications are also widely used.

We aim to create a simple, easy to use, language that will provide people with the means to process images and videos. This involves modifying images/videos, storing, etc.

2. Lexical Conventions

The members of this group are familiar with the C++ language. Hence, for , we propose to use a lexical convention that is similar to that of C++, but adapted to suit our needs to modify and process images. The following lexical elements are part of our compiler :

- Tokens
- Comments

Identifiers
Keywords
Punctuators
Operators
Literals

Tokens are the smallest elements of our language that the compiler recognizes. They are separated using one or a combination of blank spaces, horizontal tabs, new lines or comments.

Tokens are separated by one or more of the following: comments, spaces, tabs or newlines. A token is also the longest sub-string of characters that is accepted as legal by our compiler. The different type of tokens in our language are mentioned below in a little more detail.

2.1 Comments

Any comment needs to start with "//" and end with the same "//". For example, this would be considered a comment in our language:

```
// example comment //
```

A comment can also span on multiline. For example, if one of the line has "//" in it, it will comment out everything until it encounters the next "//".

```
//example
Multiline
Comment
//
```

2.2 Identifiers

An identifier is a combination of one or more letters and/or digits and the underscore character (_). It is used to represent an object of a particular data type or is the name of a function. No other special character can be a part of the identifier. They must start with a letter and are case sensitive.

The following characters are legal as the first character of an identifier,

or any subsequent character:

```
_ a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
```

The following characters are legal as any character in an identifier except for the character which has to be one among those listed above above.

```
0 1 2 3 4 5 6 7 8 9
```

2.3 Keywords

Keywords are predefined reserved identifiers that have special meanings. They cannot be used as identifiers in your program. The following keywords are reserved for **sIP** :-

```
int float pix img break continue if else for return show
```

Here, int, float, char, pix, img and color represent data types and the rest are keywords that function similar to their counterparts in C++.\

2.4 Constants

sIP allows constants corresponding to any of the data types discussed above.

2.4.1 Integer(int)

Integer constants consist of decimal integers as a signed sequence of one or more digits.

2.4.2 Float(float)

Float constants consist of signed decimal real numbers with one and only one decimal point, which separates the integer and fractional parts. The integral part is optional but the fractional part is not.

2.4.3 Pixel(pix)

We have decided to extend the concept of constant to all of our data types. A constant pix is associated with a particular color and a format which is one among YUV420 or RGB888. Hence a constant pixel has a particular color and format defined.

2.4.4 Image(img)

A constant image follows the same concept. An image is associated with a file and is group of pixels.

3 Data Types

As mentioned earlier, the identifiers are either function names or related to one of the data types defined in sIP. The interpretation of an identifier by the compiler is thus dependent on the data type. The following types are supported:-

3.1 int

Any integer within the range – (2^31) to ((2^31)--1) or 2147483648 to 2147483647 can be stored in an int type. The size associated with an int attribute is 2^32. If not initialized, the default value considered is 0.

3.2 float

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

3.3 pix

This data type is the basic unit of an image – a pixel. It consists of four components - C1, C2, C3 and format. The first three the values (between 0 and 255) that would be sufficient to discern the color. The final component can be either 0 or 1. A value of '0' indicates RGB888 format(C1, C2 and C3 represent the R, G and B components of color in this case) while a '1' indicates YUV420 format(and C1, C2 and C3 would represent Y, U and V components respectively).

The functions getC1(), getC2(), getC3() and getF() return the values of the four components. The components and functions can be accessed by the name of the operator followed by a '.'(dot) followed by the component or function. All values are instantiated to 0 by default.

3.4 img

This is essentially a 2-dimensional matrix pixels. This data type makes it easy to perform functions such as adding a pixel or group of pixels to a pre-existing image.

It has two components : w(width) and h(height) and four functions associated with an attribute of this data type : load, display, getWidth() and getHeight(). Individual pixels by: image_name[horizontal_value, vertical_value]. The components and functions can be accessed by the name of the operator followed by a '.'(dot) followed by the component or function. By default all values are 0.

3.5 string

This is essentially the same as an array of characters in c++. It consists of characters enclosed between two double quotations.

Example : “#Example String 1” is a string.

3.6 User defined

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

3.6.1 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. The general syntax for declaring arrays is given below:
type variablename[size].
This statement allocates memory of size equal to n times the size of data_type name[n].
This statement allocates memory of size equal to n times the size of 'data_type' and uses 'name' to refer to that memory location.

3.6.2 Functions

Functions in sIP have the following syntax for definition:
returntype functionname (args-list) {statements;}
Here, returntype is a valid sIP data type, functionname is a valid identifier according to the sIP identifier rules, and args-list should be a valid comma separated list of identifiers. The args-list may be left empty. The function body may be empty or can contain sIP statements and expressions.

4. Conversions

Lower numerical types can be expanded into higher ones. Example: A type int can be assigned to a type float but not vice-versa. When the int is assigned to float, the int is converted to float data format.

Similarly a pix attribute can be assigned to one of type img. Here, the pix(pixel) is converted to image specific format with which it is associated.

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

sIP 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 pix 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 pix 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*

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. With respect to images, multiplication comes in two forms. The first form takes two input images and produces an output image in which the pixel values are just those of the first image, multiplied by the values of the corresponding values in the second image. The second form takes a single input image and produces output in which each pixel value is multiplied by a specified constant. This latter form is probably the more widely used and is generally called *scaling*.

5.3.2 Division

The binary / operator indicates division. The same type considerations as for multiplication apply. The image division normally takes two images as input and produces a third whose pixel values are just the pixel values of the first image divided by the corresponding pixel values of the second image. Many implementations can also be used with just a single input image, in which case every pixel value in that image is divided by a specified constant. One of the most important uses of division is in change detection.

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. Additionally, we can also perform addition operations on the image along with the pixels. So either we can do image+image or image+pixel or pixel+ pixel. Additionally, image subtraction can also be done. Image subtractionor subtractionis a process whereby the digital numeric value of one pixel or whole image is subtracted from another image. This is primarily done for one of two reasons – leveling uneven sections of an image such as half an image having a shadow on it, or detecting changes between two images

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.

6. Function Definitions

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

7. Statements

Expressions followed by semi colons are statements in sIP. 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

sIP 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 sIP, 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

sIP 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, sIP 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 sIP. They can be used in expressions where integers and floats are mixed, and where an image/filter is mixed with a scalar value.

8.5 Function Name Overloading

sIP does not allow function name overloading. That is, each function should have a unique function name, or sIP compiler will complain.