SIP Reference Manual

Emad Barsoum
COMS W4115
eb2871@columbia.edu

1 CONTENTS

2 INTRODUCTION

SIP which stand for “Simple Image Processing Language” is a programing language designed to simplify writing image processing algorithm, by removing the plumbing work that is usually associated with dealing with images in most programming language.

To achieve that, SIP borrowed some concepts and ideas from functional language in order to let developers focus on the mathematic aspect of image processing instead of its presentation in computer language.

3 LEXICAL CONVENTIONS

We have the following tokens in SIP: identifiers, keywords, expression operators, separators, constant and string literals. Blanks, tabs, newlines, and comments are ignored except as they serve to separate tokens.

3.1 COMMENTS

SIP will support block comment and one line comment. We will borrow C\C++ block comments. So comment in SIP will begin with /* and end with */, anything in between will be ignored. SIP won’t support nested comments. One line comment will be also similar to C\C++ using // until the end of the line.

3.2 IDENTIFIERS

An identifier is a sequence of alphabetic letters in which first letter cannot be a number. Upper and lower case are considered different. If their ASCII code is different than they are different letter.

3.3 KEYWORDS

Here the list of reserved keywords:

image	kernel
histogram	float
fun	int
in	bool
for	uint
if	true
else	false
while	Pixel
return	break

We are not planning to use kernel and pixel keywords in the first version of the compiler; however, we reserved them for future use.

3.4 EXPRESSION OPERATORS

SIP support the following operators:

^	=<
=	?
==	->
>	!
<	&&
>=	&
||	|
*	+
-	/
%	~
>>	<<
[	]
(	)
:	.

3.5 SEPARATORS

Token separator are blank spaces, tab, and newline. Expression separator are comma “,” and semicolon “;”.

3.6 CONSTANT

SIP support constant for each of the basic type and initialization constant.

3.6.1 Basic type constant

- For uint (unsigned integer), constant is simply a sequence of numerical characters next to each other, each characters can be between ‘0’ and ‘9’ inclusively.
- int (integer) is similar to uint except it can be prepended with the unary ‘–‘ operator to indicate negative number.
- Float constant will support the same floating point format supported by the C language. Which is a sequence of numeric characters, then a ‘.’ Symbol to indicate fraction, then the fraction part with optional exponent “e” component.

3.7 STRING LITERALS

String literals constant will be presented, similar to C, between double quotes.

4 MEANING OF IDENTIFIERS

4.1 TYPES

We will support two category of types:

4.1.1 Object Types

Such as image and histogram. By object we mean that those types will have some extra properties and metadata associated with them.

4.1.2 Basic Types

Basic types are the common types available in most programming language such as: bool, float, int, and uint.

5 OBJECTS

The two built in objects for image manipulation are “image” and “histogram”, “image” abstract the concept of 2D image with 1 or more channels, and “histogram” abstract the concept of 1D gray level image histogram.

5.1 IMAGE

In SIP language, image object represent a 2 dimensional image, with one or more named channels, in memory. Image object also can act as a kernel filter for the convolution operator, in such a case, the image needs to be 3x3 and matches the same number of channels as the source image.

Here some usage for image object:

```
//
// Convert from HSV to RGB, using the ‘in’ operator. Red, green, blue, hue, saturation, value are
//  named channel that SIP library will treat them as RGB and HSV color space, however, there is no
// restriction on them. Developers can freely add or remove named channels or put whatever value
// in the already predefined names.
//
image rgbImage = hsvImage in (red, green, blue);

//
// Manually convert each pixel from rgb to a single channel named ‘l’, and store the result in
// custImage. “rgbImage in rgb” means map the input image to RGB color space, which will be nop
// if rgbImage is already in RGB color space,  “rgb for (l:2*r + 4*g + b)” means that each mapped
// pixel in RGB space is provided as argument to the ‘l’ expression and the result for such expression
// will be stored in “custImage”.
//
image custImage = rgbImage in (red, green, blue) for {l:2*red + 4*green + blue};
```

Because image is an object it has some properties that can be used for inspection or in an imperative algorithm. SIP image object will expose height, width, channelCount, and <channel name> properties.

<channel name> property will be added dynamically based on the list of channels in the image, if it conflicts with an existing property, we will raise a runtime error.

```
//
// Reading an image from a local file.
//
image srcImage << "c:\\images\\test.jpg";

//
// Read some properties from the just read image.
//
uint width = srcImage.width;
uint height = srcImage.height;
uint channelCount = srcImage.channelCount;
image blueImage = srcImage.blue; // Read blue channel from property
```

5.2 Histogram

Histogram object abstract image based histogram computation and results. Histogram will contains 1 dimensional histogram per each named channel, which mean that histogram will have one or more named channels to match the corresponding source image.

```
//
// 1D histogram with one channel from 3 channels image source.
//
histogram hist = rgbImage in (red); // hist now has the histogram of the red channel.

//
// 1D histogram with 3 channels from RGB image source.
//
histogram hist = rgbImage; // hist now has the histogram of the red, green and blue channels.
```

Similar to image, histogram has some properties that can be used for inspection or in an imperative algorithm. Histogram object will expose channelCount and <channel name> property.

<channel name> property will be added dynamically based on the list of channels in the histogram, if it conflicts with an existing property, we will raise a runtime error.

6 Expressions Operators

In this section, we will describe expression operators supported by SIP.

6.1 Logical not ! and binary not ~

Logical Not '!': simply negate the Boolean evaluation of a condition expression, a Boolean expression is an expression in which all its terms return "true" or "false", such as the result of <, >, <=, >=.

!logical-expression

Binary Not '~': can be applied on int or uint type only, and the result is simply flipping each bit from 1 to 0, and vice versa.

~expression

6.2 Convolution operator ^

Convolution operator is a binary operator that takes two images, or one image and another kernel function. Kernel function is a function that takes one parameter that represent a specific pixel and return the resulted channels. The evaluation is from left to right.

image-object ^ image-kernel

6.3 Multiplication and Division operators * /

Multiplication and division operators are a binary operator that takes two basic types (int, uint, or float) and return the result of the operation. If one of the types is float, then the result will be float. Evaluation is from left to right.

*expression * expression*

expression / expression

6.4 Addition and subtraction operators + -

Addition and subtraction operators are a binary operator that takes two basic types (int, uint, or float) and return the result of the operation. If one of the types is float, then the result will be float. Evaluation is from left to right.

expression + expression

expression- expression

6.5 Unary operator –

The unary operator – can be applied to int or float to negate the given number.

-constant

-variable

6.6 Mod operator %

Mod operator are a binary operator that takes int or uint type and return the result of the mod operation. Evaluation is from left to right.

expression % expression

6.7 Assignment operator =

The result of the expression on the right of the assignment operator will be copied to the variable on the left of the assignment operator.

variable-name = expression

6.8 Equality operator ==

== is a binary operator which check if the result of the expression on the left side equal to the result of the expression on the right side. The result of the == operator is a Boolean which can't be casted to other basic type (different from C and C++).

expression == expression

6.9 Logical operators && and ||

&& and || are binary operators that takes two Boolean expressions, and return true or false based on the result of those two expressions.

logical-expression && logical-expression

logical-expression || logical-expression

6.10 Binary operators & and |

& and | are binary operators that takes two expressions that evaluate to a basic types only, and return a new basic type in which each bit is the result of the & or | operator corresponding to the same bit in the two expression results.

expression & expression

expression | expression

6.11 Condition operators <, >, <=, >=

Those are binary operators that takes two expression, and the return of the result of: smaller than <, greater than >, smaller than or equal<=, and greater than or equal operations.

expression< expression

expression > expression

expression <= expression

expression >= expression

6.12 READ AND WRITE OPERATORS << AND >>

SIP will use << operator to read from a file and >> operator to write to a file. The right side of << or >> is a string literal that contains the file path, and the left side is an image object.

image-object << string-literal;

image-object >> string-literal;

6.13 ACCESSOR OPERATOR [ ] ,

[and] operators can be applied only to image object or histogram object, for histogram [ ] operator takes an integer from 0 to 255 (the range of pixel per channel will be from 0 to 255) that represent the index of the bin that need access.

histogram-object[expression]

For image, the operator is in the form of [<row>, <col>] where <row> is the index of the row that we need to access, and <col> is the index of the column that we need to access.

image-object[expression, expression]

For histogram [ ] operator will be read only.

7 OPERATOR PRECEDENCE

Operator	Associativity
[] () .	Left to right
! ~ - (unary)	Right to left
^	Left to right
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
& |	Left to right
&& ||	Left to right
expression ? true_expression : false_expression	Left to right
,	Left to right

8 DECLARATION

8.1 VARIABLES

Variable declaration in SIP will be begin with the type, followed by the variable name, and ended by a semi colon. As follow:

type-specifier instance-name;

SIP support initialization during declaration, as follow:

type-specifier instance-name = expression;

8.2 FUNCTIONS

Here how to declare a function in SIP language.

fun function-name (comma-separated-parameter-list) -> return-type

{

statements

}

Parameter-list are a sequence of type name pairs separated by comma. Recursive and nested functions aren't supported.

9 STATEMENTS

Statements are a sequence of statements separated by a semi colon ';' and executed from left to right.

9.1 CONDITIONAL STATEMENTS

SIP support if\else statement and ternary operator, as shown below:

If (condition-expression) then statement

If (condition-expression) then statement else statement

(condition-expression) ? success- statement : failed-statement

9.2 WHILE STATEMENTS

SIP support loop on a condition:

while (condition-expression) statement

9.3 FOR STATEMENTS

For statement is as follow:

for (expression; conditional-expression; expression) statement

9.4 IN STATEMENTS

In statement map the source image to the specified domain (list of named channels).

Image-object in (list-of-channel-names);

9.5 IN FOR STATEMENTS

In For statement map the source image to the specified domain (list of named channels) using the provided expression.

Image-object in (list-of-channel-names) for expression;

9.6 RETURN STATEMENT

Return exit the execution of the current function, there are two form of return:

return

return expression

9.7 BREAK STATEMENT

Break break from the inner loop.

break;

10 PROGRAM ENTRY

In SIP, the entry point to the application is a function called main similar to C. The main function takes no argument and return nothing.

```
fun main()
{
        statements
}
```

11 REFERENCES

"The C Programming Language, Second Edition. Prentice-Hall, 1988", B. W. Kernighan and D. Ritchie.