- PICEL -

Language Reference Manual

Manager | Chia-Hao Hsu | ch3141

System Architect | Chih-Sheng Wang | cw2952

Language Guru |Ruijie Zhang | rz2337

Language Guru | Chang Liu | cl3418

Testing | Rui Lu | rl2784

Content

1 Introduction

Nowaday, there are many photo editing software around the market, which provides friendly user interface to manipulate and edit the pictures easily. However, there are very few handy picture editing library or programming language. When software engineers want to develop the a photo editing related software, they often need to include several third party library into their project. We want to make a language that make photo editing software development more easier. Our Language --- PICEL provides user some easy way to store the picture. Also, our language has the built-in matrix operation, which make developer easy to build up the image processing function for the desired features. Moreover, PICEL also have the built-in convolution library, which produce some fantastic photo editing effect, for example, sharpen, blur, edge enhance, etc. For this language manual we will introduce the grammar and syntax for this language.

2 Lexical Convention

This chapter talks about the lexical conventions in PICEL including comments and tokens. Those tokens are a collection of 4 types: identifiers, keywords, literals and operators. Punctuators, such as white space, is also described in this chapter.

2.1 Comments

PICEL supports the comment starting with /* and terminating with */. Once the comment starting character /* is seen, all the other characters will be ignored until the comment ending character */ is seen. Comments do not nest. The characters /* and */ in a string literal are not considered as comments. The content in /* and */ can be multiple lines.

```
/* This is a comment.

And because it's long,

it spans three lines. */
```

2.2 Identifiers

Identifiers are sequences of characters to describe the names of variables and functions. An identifier can include letters, digits and underscores(_). The first character of an identifier cannot be a digit. An identifier cannot be the same as a keyword , a boolean literal , an int literal or a null literal. The length of an identifier is limited to 256 characters. Uppercase and lowercase letters are distinct.

identifier_name = “[‘a-z’|’A-Z’|’_’]([‘a-z’|’A-Z’|’_’|’0-9’])*”

2.3 Keywords

The keywords listed in the chart below are reserved and cannot be used for any other purpose.

if	else	for	while	break	continue

int	bool	char	sizeof	pic	void
true	false	#include	main	this	and
or	not	return	copy	delete	

2.4 Types

2.4.1 Primitive Data Type

- **Int**

An integer can hold a number in 32 bit. The value of an integer can range from -2,147,483,648(-2^{31}) to 2,147,483,647($2^{31}-1$). For a number used as a string, you should use type char.

- **bool**

Bool type is a binary logical value which can be either true or false. A bool can also be null. A bool takes up 1 bit.

- **char**

The type char contains a single character enclosed with quotation marks, e.g. ‘a’. A char takes 8 bits. Some characters may not be represented using one char.

- **string**

A string is represented by two quotes, example: “hello world”

2.4.2 Non-primitive Data Type

- **array**

An array is a data structure that holds one or more literals of the same type. The elements in an array are stored consecutively in the memory. An array can have one or more dimensions. The identifying number of an array begins at 0, not 1.

```
int NumArray[5] = [1, 2, 3, 4, 5];
char CharArray[2][3] = [‘a’, ‘b’, ‘c’; ‘d’, ‘e’, ‘f’];
```

You can also define an array and implement it later.

```
int NumArray[5];
NumArray[3] = 4;
```

You can retrieve the element in an array using its identifying number.

```
NumArray[0];                /* 1 */
CharArray[1][0];            /* 'd' */
```

But if you store more elements than the array size you declared, you will receive an overflow exception. For example, "int NumArray[5] = {1, 2, 3, 4, 5, 6};" is forbidden. Retrieving the element which exceeds the array size is forbidden as well.

```
a = NumArray[5];    /* invalid */
```

- **pic**

A pic is a data structure we use in PICEL to store an image. The picture is stored in three matrices(arrays) indicating the RGB values of this image. And syntax ".r[x][y]" can be used to refer to the R(ed) value of pixel in position (x,y). In the same way, there are ".g[x][y]" and ".b[x][y]" to refer to the G(reen) and B(lue) value of pixels. Notice that coordinates start from 0. There are some examples:

```
/* Suppose there is a pic A with size 100*100 */
int x=A.r[0][0];        /* get the Red value of point (0,0), the first pixel */
A.g[1][1]=255;          /* set the Gree value of point(1,1) to 255 */
int y=A.b[100][0];      /* Invalid since A is 100*100. So the boundary is 99*99 */
```

Picture type supports a series of specific functions such as lengthOf(), widthOf(), etc. Picture type must be implemented when defined.

And due to the fact that we implement pic in a special way, the normal assignment "=" works different here. Consider a the statement "pic A=B" where "B" is some expressions which returns a pic. And this assignment, instead of copying the content of B to a new space for A, will let the pointer of A points to B's content. If B is also a pic variable, then change A will also change B, vice verse. Thus the assignment of pic must be used carefully.

To support normal "copy" assignment, we have special function "copy(A,B)" which will copy the content of pic B to A.

2.5 Operators & Expression

Operator is the special token that perform a special operation in the program. In this

chapter we will introduce our operator in PICEL.

- **Arithmetic Operator:**

PICEL provides some basic operators for arithmetic operation: addition, subtraction, multiplication, and division. Also, there are division and negation in PICEL too. Following are some examples to show you the functionality of the operators:

Operator	Description	Example Code
+	addition	int x = 4 + 3; /* x = 7 */ int y = 4 + 5.0; /* Error! There is no float type */
-	subtraction	int y = 5 - 2; /* y = 3 */
*	multiplication	int z = 12 * 10; /* z = 120 */
/	division	int w1 = 12 / 3; /* w1 = 4 */ int w2 = 100 / 3; /* w2 = 33 */
%	modular	int m = 10 % 3; /* m = 1 */
-(int var)	negation	int a = -5; /* a = -5 */ int b = -a; /* b = 5 */

- **Array Access Operator:**

In PICEL, array can be accessed with integer index starting from 0 base, the following sample code can show the how to access array in PICEL:

int arr[3] = [1, 2, 3];

int a = arr[1]; /* a = 2 */

- **Logical Operators:**

Operator	Description	Example Code
and	conjunction	true and false; /* false */
or	disjunction	true or false; /* true */

not	negation	bool a = false; not a; /* true */

- **Comparison Operators:**

Operator	Description	Example Code
>	greater than	5 > 3; /* return true*/ ‘a’ > ‘b’; /* return false */
<	smaller than	4 < 3; /* return false*/ ‘a’ < ‘z’; /* return true */
>=	greater than or equal to	4 >= 4 / *return true */ 5 >= 4 / *return true */
<=	smaller than or equal to	4 <= 4 / * return true */ 4 <= 10 / * return false */
==	equal to	5 == 5 /* return true */ ‘a’ == ‘z’ /* return false */
!=	not equal to	‘a’ != ‘z’ /* return true */

- **Assignment Operators:**

Operator	Description	Example Code
=	assign value	int a = 5;

- **Matrix Manipulation Operators:**

Operator	Description	Example Code
#	convolution	pic_return = pic # kernel

.+	matrix addition	int matrix1[3][3] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
		int matrix2[3][3] = [[1, 1, 1], [1, 1, 1], [1, 1, 1]];
		matrix1 .+ matrix2
		/* return [[2, 3, 4], [5, 6, 7], [8, 9, 10]] */

.-	matrix subtraction	int matrix1[3][3] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; int matrix2[3][3] = [[1, 1, 1], [1, 1, 1], [1, 1, 1]];

		matrix1 .- matrix2 /* return [[0, 1, 2], [3, 4, 5], [6, 7, 8]] */

.*	matrix multiplication	int matrix1[2][2] = [[1, 2], [3, 4]];
		int matrix2[2][2] = [[1, 1], [1, 1]];
		matrix1 .+ matrix2
		/* return [[3, 3], [7, 7]] */

- **Operator Precedence and Associative Property:**

When the program contains multiple operators , then expression will follow the operator precedence and associative property to read our code. For example, if we have a code in our program like “a + b * foo()”, which will follow the rules below, in this part we follow a lot from C Language manual:

1. Function call and membership access operator expression
2. Unary Operator
3. Arithmetic Operator: Multiplication(*) & Division(/)
4. Matrix Manipulation Operator: #, .+, .-, .*
5. Arithmetic: Addition(+) & Subtraction(-)
6. Logical Operator: Greater-than(>), less-than(<), greater-than-or-equal-to(>=), and less-than-or-equal-to(<=)
7. Logical Operator: Equal-to(==) and not-equal-to(!=) expressions.
8. Logical AND expressions (and)
9. Logical OR expressions (or)
10. Conditional expressions, evaluated from left to right
11. All assignment expressions, evaluated right to left.
12. Comma Expression

3 Control Flow

In PICEL, there are two types of control flow syntax, namely Condition and Loop, which make program run into different branches according to the corresponding conditions. With the help of control flow, we can build a complex program with a clear structure.

3.1 Block

In PICEL, if-block, for /while loop block and function block all begin with "{ " and end with "}". This design is for the unification.

3.2 Condition

Conditional control flow is the basic type. In conditional control flow, program will choose among two branches, or choose whether execute a branch or not, according to the condition. Here is the syntax of conditional control flow.

if <condition>{ <branch1> } [else { <branch2> }]

All conditional control flow must begin with "if <condition> {" and end with "}". After "if", there is <condition>, the condition of this control flow. All statements in <branch1> will be executed if and only if the condition is true.

And there is also an optional syntax "else". It can be used to set the second branch of this conditional control flow. If the condition of this "if" control flow is false, then <branch1> will be ignored and statements in <branch2> will be executed.

<condition>

The <condition> could be any expression which returns a result in bool type, including a simply bool variable and complex boolean expressions with lots of boolean operations.

else

The "else" is an optional syntax of conditional control flow. And one "if" conditional control flow could have at most one "else" syntax. Since a "if" conditional control flow begins with "if" and "else" should follow the first branch which end with "}", a "else" control flow will match with the "if" which owns the closet "}" . And if this "if" already

has a “else”, it is an error. The following code is an example of this error.

```
if (b == 1){
        if (a > 0){
                if (c > 0){
                        c = 1;
                }
                a=0;
        }
        else /* match to the “if a>0” */
        {
                a=-1;
        }
        else /* match to the “if a>0” too, but that “if” already has a “else”,
        {
                so it leads to a compile error*/
                a = -2;
        }
}
```

Again, the matching cannot cross functions.

<branch>

The <branch> is a series of statements. Actually <branch1> includes all statements between <condition> and “else” (or “en” if there is no “else”) and <branch2> includes all statements between “else {” and “}”. As a result, a <branch> could just have a single statement or hundreds of statements with complex structure. Notice that, <branch> could contain all kinds of statements, including all control flow statements. Here is an example.

```
if (a>0) {
        if (c>10) {
                a=a+1;
                 c=c-10;
        }
        else{
                a=0;
                c=c*c;
        }
                b=a+c;
}
```

And inside <branch>, user could also define local variables. Those variables would only be available inside the corresponding branch.

3.3 Loop

In PICEL, we have two kinds of syntax for loop, “for” and “while”. Basically, they work in the similar way, but “for” provides a more convenient way to initialize a variable, set a stop condition and change some variable every iteration, just like how C++ does.

```
for (<initialization>;<condition>;<modification>) {
        <statement>
}
```

A “for” loop begins with a token “for <cond.> {” and end with “}”. After “for” there is a pair of parentheses which contains <initialization>, <condition> and <modification> separated by semicolon. Between the left parenthesis and right parenthesis, there is the <statement> that loop will execute every iteration. Since the three section in the parentheses is separated by semicolon, each section could only contain one statement. Here is an example code of “for” loop.

```
s=0;
for (int i=1;i<11;i=i+1){
        s=s+i; /*accumulate from 1 to 10*/
}
```

<initialization>

The <initialization> section can be regarded as an extra component that will be executed before the program runs into the loop. As a result, we can do some initialization for the loop, like set a variable "i" to 1 to do a loop from 1 to 10. In <initialization>, we can also define a new local variable which is only available in the "for" loop.

<condition>

The <condition> should be an expression which returns a boolean variable. Each time before the program enters the loop, it will check the <condition>. If the <condition> is true, it will begin the new iteration, otherwise it will quit the loop immediately. As a result, if the <condition> is false before the first iteration, the program will just pass the loop but the <initialization> will be executed since it happens at the beginning. For example

```
a=1;
b=0;
for(b=1;a<1;a=a+1){          /* Here b is set to 1 during <initialization> */
                             /* But <condition> is false, so program will not enter the loop and set b to 2*/
        b=2;
}
/*after those code, b=1 and a=1.  */
```

<modification>

The <modification> is the code that will be executed after each iteration, namely after

each time <statement> is finished. Basically it is used to modify the variable related to <condition>. And it's equivalent to set <modification> empty and put the code at the end of <statement>. So the following two examples are equivalent.

```
for (int i=1;i<11;i=i+1){
        s=s+i; /*accumulate from 1 to 10*/
    }
for (int i=1;i<11;) {
        s=s+i; /*accumulate from 1 to 10*/
        i=i+1;
}
```

while (<condition>) { <statement> }

The "while" is a simple version of "for". As long as <condition> holds true, a "while" loop will repeat and execute <statement> again and again. It's easy to change a "for" loop:

```
        for (<initialization>;<condition>;<modification>) {<statement> }
```

into a "while" loop:

```
        {
            <initialization>
            while (<condition>){
                <statement>
                <modification>
            }
        }
```

break

The “break” is a special token for loop control flow. It is used to jump out of the deepest loop it is. As a result, it could only appear inside a loop.

continue

The “continue” is another token for loop control flow. It can jump to the end of the current loop statement it’s in. It could only appear inside a loop as well.

4 GRAPHIC LIBRARY

4.1 Accessing Method

In order to access PICEL's powerful graphic library, programmer should type '#include graphics.pic' at the top of the source code, which is similar with C manner. An example is as follows.

```
#include graphics.pic

int main() {

    /* code here */

    /* Once including our graphic library */

    /* pre-defined functions are available */

    ...

}
```

The graphic library should be only included once, it be error if the library is included again. So do not include the graphic library a second time.

In the graphic library, PICEL provides three main kinds of picture-editing functions, namely Property, Enhance and Edit. Property-related functions allow programmer to obtain basic properties of single picture. Enhance-related functions allow programmer to do some self-defined color operations or enhancing operations toward the target pictures. Edit-related functions allows programmer to do other operations regarding size and direction.

4.2 File Operations

Functions

pic load(char src_filename[])

Returns a picture read from path of src_filename.

void save(char dist_filename[], pic picture)

Save a picture to a given path, which is defined in dist_filename.

void copy(pic dist_picture, pic src_picture)

Copy a picture and assign it to another identifier.

4.3 Property

PICEL provides two functions to assist programmer to obtain the width and height of a certain picture.

Functions

int sizeOf(Array array)

Returns the size of the input array.

int widthOf(pic picture)

Returns the width of the input picture, the unit is pixel.

int heightOf(pic picture)

Returns the height of the input picture, the unit is pixel.

4.4 Enhance

PICEL provides functions for programmers to modify R,G,B values of a picture. Correspondingly, PICEL supports operations toward Hue. Besides, operations about saturation and brightness are also supported by PICEL's graphic library.

Functions

void setR(pic picture, int percentage)

Sets the R value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted R value to original R value. For example, if programmer wants to double the R value, this function should be used in the following manner:

```
setR(original_pic, 200);
```

void setG(pic picture, int percentage)

Sets the G value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted G value to original G value. The usage is the same as setR().

void setB(pic picture, int percentage)

Sets the B value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted B value to original B value. The usage is the same as setR().

void setSaturation(pic picture, int percentage)

Sets the saturation value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted saturation value to original saturation value.

void setHue(pic picture, int percentage)

Sets the hue value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted hue value to original hue value.

void setValue(pic picture, int percentage)

Sets the brightness value of the given picture and directly modifies the input picture. The second parameter indicates the ratio of targeted brightness value to original brightness value.

4.5 Edit

PICEL supports resize, rotate and crop operations.

Functions

pic resize(pic picture, int new_width, int new_height)

Resizes the given picture with given width and height, and returns a new picture.

pic rotate(pic picture)

Rotates the given picture 90 degrees clockwise and returns a new picture, the degree here is default.

pic crop(pic picture, int start_x, int start_y, int new_width, int new_height)

Crops the given picture from pixel point (x, y) and returns a new picture whose size is new_width x new_height.

4.6 Built-in functions

1. **File operations:**
    a. **open**
    b. **read**
    c. **write**
2. **Screen output:**
    a. **printf**
3. **Memory operations:**
    a. **malloc**
    b. **free**
    c. **memcpy**

5 Program Structure

Layout of .pic file

1. All #include [filename] should be placed on top of source file
2. A single file can only be included once

#include stdlib.pic
Global variable declarations
Function Declarations
Function Definitions

Function and Variable Scoping

1. All variables should be declared before being referenced.
    a. Global variables being declared without initial value will have a default value 0.
2. All functions should be declared(or defined) before being called.
3. The entry point of a PICEL program is the function declared as “main”
4. The variables declared in inner blocks can NOT be referenced by from outer blocks.
5. Variables of the inner block have higher precedence than the outer block.

Function Declaration Syntax

<return type> <function name>(<parameter list>)

{

    <function content>

}

Example program: hello.pic

```
#include graphics.pic

int g_int_hello;

char* hello =”hello world ”

void foo( int a)

{

        a = a+1

}

int main()

{

        int b;

        foo(g_int_hello);

        b = g_int_hello;

        printf(“%s, %d”, b) /* the output should be “hello world, 1“ */

}
```