
Overview

PHPoC (PHP on Chip) is a scripting language developed by Sollae Systems for programmable devices such as [PHPoC Black](#), [PHPoC Blue](#) and etc. PHPoC is, as the name suggests, based on the PHP (PHP: Hypertext Preprocessor) a widely-used as a general-purpose programming language. PHP is originally designed for web development on a server-side. And PHPoC also has this characteristic. But there are some differences between PHPoC and PHP, because PHPoC is used to program embedded systems.

PHPoC not only provides a powerful tool for making dynamic and interactive Web but also provides functions to access peripherals of embedded systems. Therefore, anyone can use it easily.

※ PHPoC is built similarly to PHP, but not all grammars are fully compatible. This document does not cover the differences between PHPoC and PHP, so please refer to [PHPoC vs PHP](#) for details.

Syntax

- [Basic Syntax](#)
- [Types](#)
- [Variable](#)
- [Constants](#)
- [Operators](#)
- [Control Structures](#)
- [Functions](#)
- [Classes and Objects](#)
- [Namespace](#)

Basic Syntax

PHPoC tags

PHPoC script consists of two tags: opening tag (<?php or <?) and closing tag (?>). Meaning of these tags is start and end of script. Everything outside of a pair of these tags is ignored by PHPoC parser and printed to standard output port. In case of webpage, the ignored text is sent to web browser.

```
<?php           // opening tag
echo "Hello PHPoC!"; // script
?>             // closing tag
```

Inserting PHPoC Script into Webpage (HTML)

Features of everything outside of a pair of opening and closing tags are ignored by the PHPoC parser which allows PHPoC to be embedded in HTML documents.

```
<p>This will be ignored by PHPoC and displayed by the browser. </p>
<?php echo "While this will be parsed."; ?>
<p>This will also be ignored by PHPoC and displayed by the browser. </p>
```

By using conditional statement, PHPoC can determine the outcome of text which is outside of PHPoC tags. See the example below.

```
<?php if(true){ ?>
This will show if the expression is true. <!-- This will be displayed -->
<?php } else { ?>
Otherwise this will show.           <!-- This will not be displayed -->
<?php } ?>
```

Instruction Separation

As in C, PHPoC requires instructions to be terminated with a semicolon at the end of each statement. The closing tag of a block of PHPoC statements automatically implies a semicolon; you do not need to type a semicolon.

```
<?php
echo "the first statement.WrWn";      // the first line, ';' is used
echo "the last statement.WrWn"      // the last line, ';' can be omitted
?>
<?php echo "single line statement.WrWn" ?> // single line, ';' can be omitted
```

※ Although you can omit a semicolon, we recommend using semicolon at all times because it is not correct syntax.

Comments

As 'C' and 'C++', PHPoC supports one-line and multiple-line comments.

```
<?php
echo "the first statement.WrWn";      // one-line comment
/* This
is
multiple-line comment. */
echo "the last statement.WrWn";
?>
```

※ UNIX shell-style comment '#' is not supported in PHPoC.

Types

PHPoC supports 5 types: Boolean, integer, floating-point number, string and arrays.

- [Boolean](#)
- [Integer](#)
- [Floating Point Numbers](#)
- [String](#)
- [Array](#)
- [Type Juggling](#)

※ Not support objects and NULL.

Boolean

This is the simplest type which can be either TRUE or FALSE. Both TRUE and FALSE are case-insensitive.

To explicitly convert a value to Boolean, use the (bool) or (boolean) casts and this is also case-insensitive.

```
<?php
$bool_true = TRUE;           // Boolean true
$int_test = 3;              // integer 3
$bool_test = (bool)$int_test; // convert integer to Boolean
?>
```

When converting to Boolean, the following values are considered as FALSE.

- the integer 0 (zero)
- the float 0.0 (zero)
- the empty string ("")

The others are considered as TRUE. (Including string "0")

Integer

Integers can be specified in decimal (base 10), hexadecimal (base 16) and octal (base 8), and binary (base 2) notation, optionally preceded by a sign (- or +).

To explicitly convert a value to integer, use either the (int) or (integer) casts and this is case-insensitive.

```
$octal = 010;           // 8 - base 8
$decimal = 10;          // 10 - base 10
$hexadecimal = 0x10;    // 16 - base 16
$binary = 0b10;         // 2 - base 2
$str_test = "10";       // string "10"
$int_test = (int)$str_test; // convert string to integer
```

Floating Point Numbers

Floating point numbers can be specified using any of the following syntaxes:

```
$float0 = 3.14;      // 3.14
$float1 = 3.14e3;    // 3140
$float2 = 3.14E-3;   // 0.00314
```

E3 (e3) above means multiplication by 1000 which is 10 cubed. E-3 (e-3) means multiplication by 1/1000 which is reciprocal number of 10 cubed.

To explicitly convert a value to floating point number, use the (float) casts and this is case-insensitive.

- Precision Floating point numbers have limited precision. Computer cannot show the correct value of rational number in base 10, like 0.1 or 0.3, because it calculates the number in base 2.

```
$a = 0.1 / 0.3;
printf("%.20eWrWn", $a); // print $a down to 20 places of decimals
```

```
[result]
3.333333333333333333333370341e-1
```

As you can see the result above, the value does not correct down to fifteen places of decimals. Because of these limitations, it is better not to use direct comparing operation of floating point numbers.

- NAN

Some numeric operations can result in a value represented by the constant NAN. This result represents an undefined or unrepresentable value in floating point calculations. Any loose or strict comparisons of this value against any other value, including itself, will have a result of FALSE.

```
$float0 = acos(2);  
if($float0 == $float0)  
    echo "True\n";           // result: FALSE  
else  
    echo "float0\n";       // output: NAN
```

- INF

Constant INF represents a number which is beyond the representable range in floating point calculations.

```
$float0 = 1.8E+309;  
echo "float0";           // output: INF
```

String

A string is a series of characters. This can be specified in both single quoted and double quoted ways.

To explicitly convert a value to string, use the (string) casts and this is case-insensitive.

```
<?php
$int_test = 10;           // 10
$str_test = (string)$int_test; // convert integer to string
?>
```

- Single quoted

The simplest way to specify a string is to enclose it in single quotes. To specify a literal single quote, escape it with a backslash (). To specify a literal backslash, double it (\\). All other instances of backslash will be treated as a literal backslash.

```
<?php
echo 'This is a simple string';
echo "r\n";
echo 'insert
newlines';
echo "r\n";
echo 'specify \' (single quotation)';
echo "r\n";
echo 'specify \\ (back slash)';
echo "r\n";
echo 'specify \ (back slash)';
echo "r\n";
echo 'nothing happened r\n';
echo "r\n";
echo 'nothing $a happened';
?>
```

```
[result]
This is a simple string
insert
newlines
specify ' (single quotation)
specify \ (back slash)
specify \ (back slash)
nothing happened r\n
nothing $a happened
```

- Double quoted

If the string is enclosed in double-quotes ("), PHPoC will interpret more escape sequences for special characters. The special characters below can be interpreted by double quoted string. All other instances of backslash will be treated as a literal backslash.

Sequence	Meaning
\n	linefeed (LF)
\r	carriage return (CR)
\t	horizontal tab (HT)
\\	backslash
\"	double-quote
\\$	dollar sign
\[0-7]{base 8}	character in octal notation
\x[0-9][A-F][a-f]{base 16}	character in hexadecimal notation

```
<?php
echo "This is a simple string";
echo "\r\n";
echo "insert \r\n newlines";
echo "\r\n";
echo "Specify \" (Double quotation)";
?>
```

```
[result]
This is a simple string
insert
newlines
Specify " (Double quotation)
```

※ PHPoC is not supporting \e, \v and \f sequences.

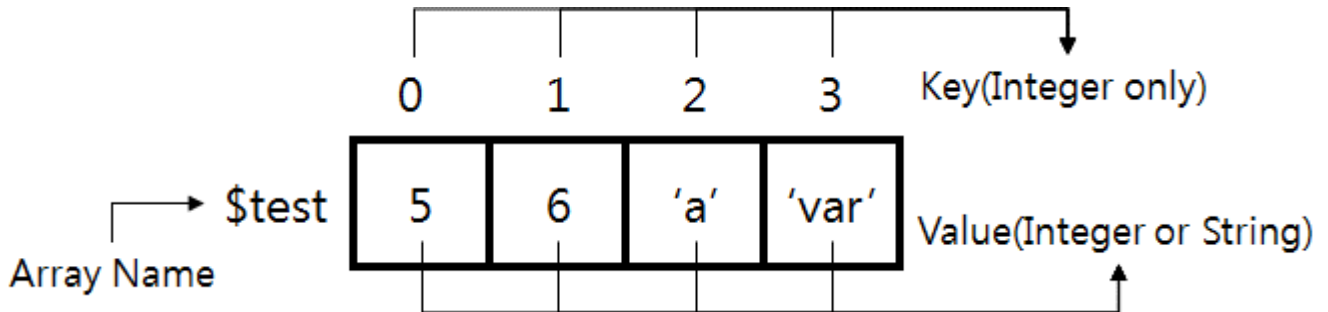
Also, PHPoC interpret variables in this manner.

```
<?php
$a = "a variable";
echo "Process $a";
?>
```

```
[result]
Process a variable
```

Array

Array is a gathering of character, arrays or integers in order. Array in PHPoC consists of value and key. Key is only numbers starting from 0.



- Array Definition

When define array in PHPoC, initial value should be given. You can give the initial values to array directly or indirectly by variables.

```
<?php
    $int1 = 1;
    $char1 = 's';
    $str1 = 'sollae';
    $array1 = array(1, 2, 3);           // array only with integer
    $array2 = array('a', 'b', 'c');   // array only with string
    $array3 = array($int1, $char1, $str1); // array with mix of integers and string
?>
```

- Array Use

Elements of array can be access by putting key value in square bracket.

```
<?php
    $array1 = array(1, 2, 3);           // define and initialize array
    $array1[0] = 5;                    // set the value of key 0 to 5
    echo $array1[0];                   // print the value of key 0
?>
```

```
[result]
5
```

- String and Array

String can be used as array like examples below.

```
<?php
    $str = "test";           // define a string variable
    $str[0] = 'T';          // set the first character to T
    echo $str;
?>
```

```
[result]
Test
```

- Multi-dimension Array

PHPoC supports multi-dimension array.

```
<?php
    $array0 = array(0, 1, 2);           // one-dimension
    $array1 = array(3, 4, 5);
    $array2 = array($array0, $array1, array(6, 7, 8)); // two-dimension
?>
```

Type Juggling

- Arithmetic Operator: addition(+), subtraction(-), multiplication(*), division(/)

Types	Boolean	Integer	Float Point	String
Boolean	X	X	X	X
Integer	X	O	O	X
Float Point	X	O	O	X
String	X	X	X	X

- Arithmetic Operator: the rest (%)

Types	Boolean	Integer	Float Point	String
Boolean	X	X	X	X
Integer	X	O	X	X
Float Point	X	X	X	X
String	X	X	X	X

- Bitwise Operator: AND(&), OR(|), XOR(^), left shift(<<), right shift(>>)

Types	Boolean	Integer	Float Point	String
Boolean	X	X	X	X
Integer	X	O	X	X
Float Point	X	X	X	X
String	X	X	X	X

- Bitwise Operator: compliment(~)

Boolean	Integer	Float Point	String
X	O	X	X

- Comparison Operator
less than(<), greater than(>), less or equal(<=), greater or equal(>=)

Types	Boolean	Integer	Float Point	String
Boolean	X	X	X	X
Integer	X	O	O	X
Float Point	X	O	O	X
String	X	X	X	O

- Comparison Operator: equal(==), not equal(!= and <>)

Types	Boolean	Integer	Float Point	String
Boolean	O	X	X	X
Integer	X	O	X	X
Float Point	X	X	O	X
String	X	X	X	O

- Increment/Decrement Operator: increment(++), decrement(--)

Boolean	Integer	Float Point	String
X	O	X	X

- Logical Operator: AND(&&), OR(||)

Types	Boolean	Integer	Float Point	String
Boolean	○	○	X	○
Integer	○	○	X	○
Float Point	X	X	X	X
String	○	○	X	○

- Logical Operator: NOT(!)

Boolean	Integer	Float Point	String
○	○	X	○

- Sign Operator: positive(+), negative(-)

Boolean	Integer	Float Point	String
X	○	○	X

- Expression in Control Structure: if, for, (do) while

Boolean	Integer	Float Point	String
○	○	X	○

- printf function output format

Types	Boolean	Integer	Float Point	String
%b, %o, %x	X	○	X	X
%d, %u	X	○	X	X
%c	X	○	X	X
%e, %f, %g	X	X	○	X
%s	X	X	X	○

Variables

Variables

A variable consists of variable name and mark in PHPoC.

Sign	Name	
	The first letter	The rest
\$	Alphabet or _ (underscore)	Alphabet, number or _ (underscore)

Examples are as follows:

Correct Example	<pre>\$_var = 0; \$var1 = 0; \$var_1 = 0;</pre>
Incorrect Example	<pre>\$123 = 0; // name begins with number \$var_#% = 0; // name with special characters (#, %)</pre>

When defining a variable in PHPoC, the initial value should be given. More than one variable cannot be defined in a single line.

Correct Example	<pre>\$var1 = 0; \$var2 = 1; \$var3 = 2;</pre>
Incorrect Example	<pre>\$var1; // no initial value \$var2 = 0, \$var3 = 1; // two values are defined in a line</pre>

※ The maximum size of variable name is 31 bytes. The rest will be ignored.

Predefined Variables

PHPoC does not offer the predefined variables.

Variable Scope

The scope of a variable is the context within which it is defined in PHPoC.

```
<?php
    $var1 = 0;    // $var1 is only available outside the function test
    function test()
    {
        $var2 = 1;    // $var2 is only available inside the function test
    }
?>
```

- The global Keyword
To expand the scope of variables, use the global keyword.

```
<?php
    $var1 = 0;
    function test()
    {
        global $var1; // $var1 is available inside the function test
    }
?>
```

Variable Variables

PHPoC is not supporting variable variables.

Constants

A constant is an identifier for a simple value which is not changed in running script. A constant can be specified with define keyword.

```
<?php
define("TEST_CONST", 16);           // integer constant
define("TEST_NAME", "constant");   // string constant
?>
```

- ※ PHPoC is not supporting to define constants by const keyword.
- ※ PHPoC is not supporting Magic constants.

Operators

An operator is something that takes one or more values and yields another value. PHPoC supports assignment, arithmetic, incrementing / decrementing, comparison, logical, string, bitwise and conditional operators.

- [Operator Precedence](#)
- [Arithmetic Operators](#)
- [Assignment Operators](#)
- [Bitwise Operators](#)
- [Comparison Operators](#)
- [Incrementing / Decrementing Operators](#)
- [Logical Operators](#)
- [String Operators](#)
- [Conditional Operator](#)

※ PHPoC does not support Error Control Operators, Execution Operators and Array Operators.

Operators

Operator Precedence

The precedence of an operator specifies how "tightly" it binds two expressions together. The operator precedence is as follows:

Precedence	Operator Mark	Operators
High	[(Parenthesis
	++ == ~ (int) (string) (bool)	Types/Increment/Decrement
	!	Logical
	* / %	Arithmetic
	+ - .	Arithmetic
	<>	Bitwise
	< <= > >=	Comparison
	== != === !== <>	Comparison
	&	Bitwise
	^	Bitwise
		Bitwise
	&&	Logical
		Logical
	? :	Ternary
	Low	= += -= *= /= .= %= &= = ^= <>=

※ When operators which have the same priority are used repeatedly, calculation is usually started from left. However, assignment, incrementing / decrementing, cast and logical operators are started from right.

- Example of Operator Precedence

```
<?php
    $var0 = 3 * 3 % 5;           // (3 * 3) % 5 = 4 (from left)

    $var1 = 1;
    $var2 = 2;
    $var1 = $var2 += 3;        // $var1 = ($var2 += 3), $var1, $var2 = 5 (from right)
?>
```

Operators

Arithmetic Operators

Operator	Sign	Syntax	Additional Information
addition	+	$\$var1 + \$var2$	-
subtraction	-	$\$var1 - \$var2$	-
multiplication	*	$\$var1 * \$var2$	-
division	/	$\$var1 / \$var2$	quotient of division in integer operation
the rest	%	$\$var1 \% \$var2$	remainder of division

Operators

Assignment Operators

Assignment operator assigns right value or result of expression to the left.

Operator	Sign	Syntax	Additional Information
assignment	=	<code>\$var = 1</code>	assign 1 to <code>\$var</code>
addition	<code>+=</code>	<code>\$var += 1</code>	assign result of <code>\$var + 1</code> to <code>\$var</code>
subtraction	<code>-=</code>	<code>\$var -= 1</code>	assign result of <code>\$var - 1</code> to <code>\$var</code>
multiplication	<code>*=</code>	<code>\$var *= 2</code>	assign result of <code>\$var * 2</code> to <code>\$var</code>
division	<code>/=</code>	<code>\$var /= 2</code>	assign result of <code>\$var / 2</code> to <code>\$var</code>
the rest	<code>%=</code>	<code>\$var %= 2</code>	assign result of <code>\$var % 2</code> to <code>\$var</code>
concatenate	<code>.=</code>	<code>\$var .= "string"</code>	assign result of <code>\$var . "string"</code> to <code>\$var</code>
bitwise AND	<code>&=</code>	<code>\$var &= 0x02</code>	assign result of <code>\$var & 0x02</code> to <code>\$var</code>
bitwise OR	<code> =</code>	<code>\$var = 0x02</code>	assign result of <code>\$var 0x02</code> to <code>\$var</code>
bitwise XOR	<code>^=</code>	<code>\$var ^= 0x02</code>	assign result of <code>\$var ^ 0x02</code> to <code>\$var</code>
left shift	<code><<=</code>	<code>\$var <<= 4</code>	assign result of <code>\$var << 4</code> to <code>\$var</code>
right shift	<code>>>=</code>	<code>\$var >>= 4</code>	assign result of <code>\$var >> 4</code> to <code>\$var</code>

Operators

Bitwise Operators

Operator	Sign	Syntax	Additional Information
bitwise AND	&	\$b1 & \$b2	bit AND \$b1 and \$b2
bitwise OR		\$b1 \$b2	bit OR \$b1 and \$b2
complement	~	~\$b1	invert \$b1 (0 to 1, 1 to 0)
bitwise XOR	^	\$b1 ^ \$b2	bit XOR \$b1 and \$b2
left shift	<<	\$b1 << 8	8 digits left shift \$b1
right shift	>>	\$b1 >> 8	8 digits right shift \$b1

- Example of Bitwise operators

```
<?php
  $b1 = 0x11;           // 0001 0001
  echo "$b1WrWn";
  $b2 = 0x23;           // 0010 0011
  echo "$b2WrWn";
  $b3 = $b1 & $b2;      // 0000 0001, bit AND
  echo "$b3WrWn";
  $b3 = $b1 | $b2;      // 0011 0011, bit OR
  echo "$b3WrWn";
  $b3 = ~$b1;           // 1110 1110, NOT
  echo "$b3WrWn";
  $b3 = $b1 << 1;       // 0010 0010, left shift 1 digit > double
  echo "$b3WrWn";
  $b3 = $b1 >> 1;       // 0000 1000, right shift 1 digit > half
  echo "$b3WrWn";
?>
```

```
[result]
17
35
1
51
-18
34
8
```

- Left Shift

Added bits by left shift operation are always 0.

```
<?php
$b1 = 0xFFFFFFFFFFFFFFFF; // -1
$b2 = $b1 << 1;           // 0xFFFFFFFFFFFFFFFE (added bit is 0)
echo "$b2";
?>
```

```
[result]
-2
```

- Right Shift

Added bits by right shift operation are always the same with sign bit.

```
<?php
$b1 = 0xFFFFFFFFFFFFFFFF; // -1
$b2 = $b1 >> 1;           // 0xFFFFFFFFFFFFFFF (added bit is 1)
echo "$b2";
?>
```

```
[result]
-1
```


Operators

Comparison Operators

The result of comparison operators is always Boolean type.

Operator	Sign	Syntax	Additional Information
Equal (value)	==	<code>\$var1 == \$var2</code>	TRUE if <code>\$var1</code> is equal to <code>\$var2</code>
Identical (value+type)	===	<code>\$var1 === \$var2</code>	TRUE if <code>\$var1</code> is equal to <code>\$var2</code> and they are of the same type
Not equal (value)	!=	<code>\$var1 != \$var2</code>	TRUE if <code>\$var1</code> is not equal to <code>\$var2</code>
Not identical (value+type)	!==	<code>\$var1 !== \$var2</code>	TRUE if <code>\$var1</code> is not equal to <code>\$var2</code> or they are not of the same type
Not equal (value)	<>	<code>\$var1 <> \$var2</code>	TRUE if <code>\$var1</code> is not equal to <code>\$var2</code>
Less than	<	<code>\$var1 < \$var2</code>	TRUE if <code>\$var1</code> is strictly less than <code>\$var2</code>
Greater than	>	<code>\$var1 > \$var2</code>	TRUE if <code>\$var1</code> is strictly greater than <code>\$var2</code>
Less than or equal to	<=	<code>\$var1 <= \$var2</code>	TRUE if <code>\$var1</code> is strictly less than or equal to <code>\$var2</code>
Greater than or equal to	>=	<code>\$var1 >= \$var2</code>	TRUE if <code>\$var1</code> is strictly greater than or equal to <code>\$var2</code>

- Example of Comparison Operator

```
<?php
    $var1 = 1;
    $var2 = 2;
    $var3 = $var1 == $var2; // $var3 = False (0 - False)
    $var4 = $var1 != $var2; // $var4 = True (1 - True)
    $var5 = $var1 <> $var2; // $var5 = True (1 - True)
    $var6 = $var1 < $var2; // $var6 = True (1 - True)
    $var7 = $var1 > $var2; // $var7 = False (0 - False)
    $var8 = $var1 <= $var2; // $var8 = True (1 - True)
    $var9 = $var1 >= $var2; // $var9 = False (0 - False)
?>
```

Operators

Incrementing / Decrementing Operators

Operator	Sign	Syntax	Additional Information
Increment	++	<code>\$var++</code>	Returns <code>\$var</code> , then increments <code>\$var</code> by one
		<code>++\$var</code>	Increments <code>\$var</code> by one, then returns <code>\$var</code>
Decrement	--	<code>\$var--</code>	Returns <code>\$var</code> , then decrements <code>\$var</code> by one
		<code>--\$var</code>	Decrement <code>\$var</code> by one, then returns <code>\$var</code>

- Example of Incrementing / Decrementing Operators

```
<?php
    $var = 3;

    echo $var++;    // echo $var, then increments $var by one
    echo $var;

    echo ++$var;    // increments $var by one, then echo $var
    echo $var;

    echo $var--;    // echo $var, then decrements $var by one
    echo $var;

    echo --$var;    // decrements $var by one, then echo $var
    echo $var;
?>
```

Operators

Logical Operators

Operator	Sign	Syntax	Additional Information
AND	&&	(expr1) && (expr2)	TRUE if both expr1 and expr2 are TRUE
OR		(expr1) (expr2)	TRUE if either expr1 or expr2 is TRUE
NOT	!	!(expr1)	TRUE if expr1 is not TRUE

- Example of Logical Operators

```
<?php
  $var1 = true;
  $var2 = false;

  $var3 = $var1 && $var2;
  $var4 = $var1 || $var2;
  $var5 = !$var1;

  echo (int)$var3, "rWn"; // 0 - FALSE
  echo (int)$var4, "rWn"; // 0 - TRUE
  echo (int)$var5, "rWn"; // 0 - FALSE
?>
```

[result]

1

Operators

String Operators

Operator	Sign	Syntax	Additional Information
Concatenation	.	<code>\$str1 . \$str2</code>	Concatenates <code>\$str1</code> and <code>\$str2</code>
	<code>.=</code>	<code>\$str1 .= \$str2</code>	Assigns result of concatenating <code>\$str1</code> and <code>\$str2</code> to <code>\$str1</code>

- Example of String Operators

```
<?php
    $str1 = "Hel";
    $str2 = "lo";

    $str3 = $str1 . $str2; // $str3 = "Hello"
    $str3 .= " PHPoC!";   // $str3 = "Hello PHPoC!"

    echo $str3;
?>
```

```
[result]
Hello PHPoC!
```

Operators

Conditional Operator

Operator	Sign	Syntax	Additional Information
Ternary	? :	(expr) ? \$a : \$b	evaluates to \$a if expr evaluates to TRUE, and \$b if expr evaluates to FALSE

- Example of Ternary Operator

```
<?php
  $var1 = $var2 = 1;
  $var3 = ($var1 == $var2) ? true : false;
  $var4 = ($var1 != $var2) ? true : false;
  echo (int)$var3, "WrWn";
  echo (int)$var4;
?>
```

```
[result]
1
```

※ It is recommended that you avoid "stacking" ternary expressions because PHPoC's behavior when using more than one ternary operator within a single statement is non-obvious.

Control Structures

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing (an empty statement). Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces.

All of statements and grouped statements are normally executed in order but you can skip or repeat statements by specifying condition.

Most control structures provided by PHPoC are very similar with other programming language including C.

- [if](#)
- [else](#)
- [elseif / else if](#)
- [while](#)
- [do-while](#)
- [for](#)
- [break](#)
- [continue](#)
- [switch](#)
- [return](#)
- [include](#)
- [include_once](#)

※ PHPoC does not support `foreach`, `declare`, `require`, `require_once` and `goto` in PHP.

Control Structures

if

if construct is one of the most important features of PHPoC. It allows for conditional execution of code fragments.

- Structure of if

Syntax	Description
if (expr) stmt;	Execute stmt if expr is TRUE, otherwise skip stmt.

- Example of Single if

```
<?php
  $var1 = $var2 = 1;
  if($var1 == $var2)           // expression is TRUE
    echo "var1 and var2 are equal"; // statement will be executed
?>
```

```
[result]
var1 and var2 are equal
```

- Example of Multi-line if

```
<?php
  $var1 = 1;
  $var2 = 2;
  if($var1 < $var2)
  {
    echo "var1 is smaller than var2"; // grouping by curly braces
    echo "Wnbye!";
  }
?>
```

```
[result]
var1 is smaller than var2
bye!
```

- Example of Recursive if

```
<?php
$var1 = $var2 = 1;
$var3 = 2;
if($var1 == $var2)           // expression is TRUE
{
    if($var1 < $var3)        // expression is TRUE
        echo "var1 and var2 are equal"; // statement will be executed
}
?>
```


Control Structures

else

else extends an if statement to execute a statement in case the expression in the if statement evaluates to FALSE. By using else, you can specify statements when the result of expression is both true and false. The else statement does not have expression and cannot be used without if statement.

- Structure of if-else

Syntax	Description
<pre>if(expr) stmt1; else stmt2;</pre>	<ol style="list-style-type: none">1) executes stmt1 if expr is TRUE2) executes stmt2 if expr is not TRUE

- Example of if-else

```
<?php
  $var1 = $var2 = 2;
  if($var1 == $var2)           // expression is FALSE
    echo "var1 and var2 are equal";
  else
    echo "var1 and var2 are not equal"; // statement will be executed
?>
```

```
[result]
var1 and var2 are equal
```

- Example of Recursive if-else

```
<?php
  $var1 = $var2 = 1;
  $var3 = 2;
  if($var1 > $var2)          // expression is FALSE
    echo "var1 and var2 are equal";
  else
  {
    if($var1 > $var3)        // expression is FALSE
      echo "good";
    else
      echo "bad";           // statement will be executed
  }
?>
```

```
[result]
bad
```

Control Structures

elseif / else if

elseif is a combination of if and else. Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

The elseif statement cannot be used without if statement. Multiple elseif statements can be used in a single if statement.

- Structure of elseif

Syntax	Description
<pre>if(expr1) stmt1; elseif(expr2) stmt2; elseif(expr3) stmt3; else stmt4;</pre>	<p>1)stmt1 will be executed if expr1 is TRUE 2)stmt2 will be executed if expr2 is TRUE 3)stmt3 will be executed if expr3 is TRUE 4) stmt4 will be executed if none of expr1, expr2 or expr3 is TRUE</p>

- Structure of elseif

```
<?php
  $var1 = 1;
  $var2 = 2;
  $var3 = 3;
  if($var1 == 0)           // expression is FALSE
    echo "var1 = 0";
  elseif($var2 == 0)      // expression is FALSE
    echo "var2 = 0";
  elseif($var3 == 0)      // expression is FALSE
    echo "var3 = 0";
  elseif($var3 == 3)      // expression is TRUE
    echo "var3 = 3";      // statement will be executed
  else
    echo "No Result";
?>
```

```
[result]
var3 = 3
```

Control Structures

while

while loops are the simplest type of loop. This executes the nested statements repeatedly, as long as the while expression evaluates to TRUE.

- Structure of while

Syntax	Description
while(expr) stmt;	stmt is repeatedly executed when expr is true

- Example of while

```
<?php
  $var = 0;
  while($var < 3) // expression will be TRUE till third comparison
  {
    echo "$var\n"; // statement will be executed three times
    $var++; // increase $var by one
    sleep(1); // 1 second delay
  }
?>
```

[result]

```
1
2
```

- Infinite Loop

You have to keep it in mind that repetitive statement can be repeated infinitely as if the result of expression is always TRUE. In this case, statement in the structure will be infinitely executed. This situation is called infinite loop.

```
<?php
  $var = 0;
  while(1)      // expression will always be TRUE
  {
    echo "$var\n";
    $var++;    // increase $var by one, $var = 1, 2, 3, ...
    sleep(1);  // 1 second delay
  }
?>
```

Control Structures

do-while

do-while loops are very similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning.

- Structure of do-while

Syntax	Description
<pre>do { stmt; } while(expr);</pre>	<p>1) execute stmt first, then check expr 2) execute stmt repeatedly as long as expr is TRUE</p>

- Example of do-while

```
<?php
  $var = 0;
  do
  {
    echo "$var\n"; // execute statement at least once
    $var++;
    sleep(1);
  }while($var < 3);
?>
```

```
[result]

1
2
```

Control Structures

for

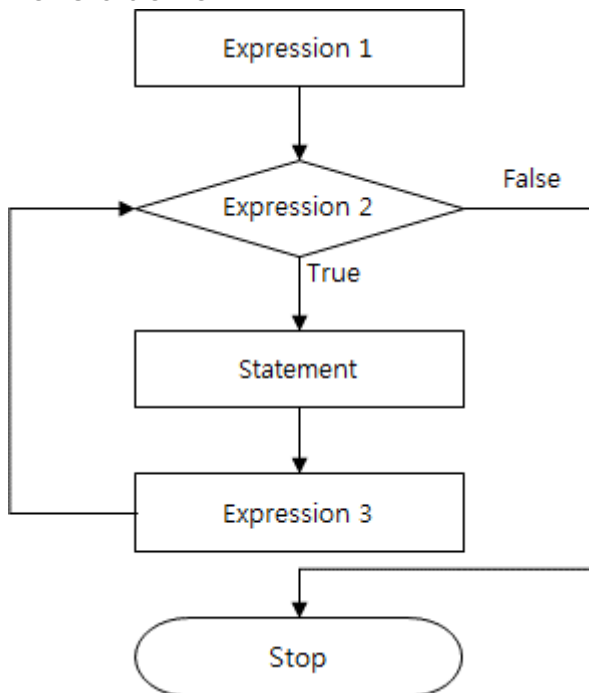
for loops are mainly used to iterate statement for specific number.

- Structure of for

Syntax	Description
<pre>for(expr1;expr2;expr3) { stmt; }</pre>	1) expr1 is evaluated once at the beginning of the loop 2) stmt will be executed if expr2 is TRUE 3) expr3 is evaluated at the end of iteration

In general, initializing a variable is given in expr1 and conditional expression is specified in expr2. In expr3, incrementing or decrementing operation is used to determine amount of iteration.

- Flowchart of for



- Example of for

```
<?php
  for($i = 0; $i < 5; $i++) // increase $i by one from 0 and compare with 5
  {
    echo $i;           // statement is executed if $i is less than 5
  }
?>
```

[result]

```
1
2
3
4
```

Each expression can be omitted.

- Example of Omitting Expression 1

```
<?php
  for($i = 1; ; $i++) // Omit the second expression
  {
    if($i > 10)
      break;           // Break the for loop
    echo $i;
  }
?>
```

[result]

```
12345678910
```


- Example of Omitting Expression 2

```
<?php
  $i = 0;
  for(;;) // Omit all of expressions
  {
    if($i > 10)
      break; // Break the for loop
    echo $i;
    $i++;
  }
?>
```

```
[result]
012345678910
```

- Combination of for and array
for loops are very suitable for proceeding elements of an array in order.

```
<?php
  $arr = array(1, 2, 3); // arr[0] = 1, arr[1] = 2, arr[2] = 3
  for($i = 0; $i < 3; $i++) // increase $i by one from 0 and compare with 3
  {
    echo $arr[$i]; // statement is executed if $i is less than 3
  }
?>
```

```
[result]
123
```

Control Structures

break

break ends execution of the current for, while, do-while or switch structure.

- Structure of break

Syntax	Description
<pre>for(;;) { if(expr) { stmt; break; } }</pre>	executes stmt and exit iteration and get out of for loop if expr is TRUE

- Example of break

```
<?php
for($i = 0; ; $i++) // infinite loop
{
  if($i > 10)
    break; // exit for loop
  echo $i;
}
?>
```

```
[result]
012345678910
```

- Option of break

break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

```
<?php
$j = 1;
for($i = 0; ; $i++) // infinite loop(level 1)
{
    while($j != 0) // infinite loop(level 2)
    {
        if($j > 10)
            break 2; // exit for loop as well as while loop
        echo $j;
        $j++;
    }
}
?>
```

```
[result]
12345678910
```

Control Structures

continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.

- Structure of continue

Syntax	Description
<pre>for(; ;) { if(expr) { stmt1; continue; } stmt2; }</pre>	<p>executes stmt1 and go to the beginning of for loop if expr is TRUE</p>

- Example of continue

```
<?php
for($i = 1; ; $i++) // infinite loop
{
  if($i % 5)
    continue;      // go to the beginning of for loop
  echo "$i\n";    // statement is executed if expression is FALSE
  sleep(1);
}
?>
```

```
[result]
5
10
15
... (repetition)
```

- Option of continue

continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

```
<?php
$j = 0;
for($i = 0; ; $i++)                // infinite loop(level 1)
{
    sleep(1);
    if($i)
        echo "This is for statement W$i = $iWrWn"; // repeated by continue 2
    while(1)                        // infinite loop(level 2)
    {
        $j++;
        if(($j % 5) == 0)
            continue 2;            // go to the beginning of for loop
        echo "$j, ";
        sleep(1);
    }
}
?>
```

```
[result]
1, 2, 3, 4, This is for statement $i = 1
6, 7, 8, 9, This is for statement $i = 2
11, 12, 13, 14, This is for statement $i = 3
... (repetition)
```

Control Structures

switch

The switch statement is similar to a series of if statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. A special case is the default case. This case matches anything that wasn't matched by the other cases.

- Structure of switch

Syntax	Description
<pre>switch(expr) { case val1: stmt1; break; case val2: stmt2; break; default: stmt3; break; }</pre>	<ol style="list-style-type: none"> 1) compare val1 and expr1 if they are the same 2) execute stmt1 and exit switch if val1 is the same with expr 3) compare val2 and expr1 if they are the same 4) execute stmt2 and exit switch if val2 is the same with expr 5) execute stmt3 if neither val1 nor val2 is the same with expr

- Example of switch

```
<?php
$var = 1;
switch($var)
{
  case 1:
    echo "var is 1";
    break;
  case 2:
    echo "var is 2";
    break;
  default:
    echo "Error";
    break;
}
?>
```

```
[result]
var is 1
```

- Example of default
default case can be omitted in switch statement.

```
<?php
$var = 1;
switch($var)
{
    case 1:
        echo "var is 1";
        break;
    case 2:
        echo "var is 2";
        break;
}
?>
```

```
[result]
var is 1
```

Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

- case statement without break

```
<?php
$var = 1;
switch($var)
{
    case 1:
        echo "3";
    case 2:
        echo "3";
    case 3:
        echo "3";
        break;
    case 4:
        echo "4";
}
?>
```

```
[result]
333
```

※ You cannot use semicolon (;) behind the case statement instead of colon (:).

Control Structures

return

If return statement is called from within a function, it immediately ends execution of the current function, and returns its argument as the value of the function call. If it is called from the global scope, then execution of the current script file is ended. If the current script file was included, then control is passed back to the calling file. Furthermore, if the current script file was included, then the value given to return will be returned as the value of the include call.

If return is called from within the init.php file, then script execution ends.

Syntax	Description
return argument;	ends execution of current function or script and returns argument. The argument can be omitted.

- Example of return in function call

```
<?php
function func()      // declare a user-define function fun()
{
    $var1 = 1;
    return $var1;    // return $var1 (1)
}

$var2 = 2;
$var3 = func();      // assign $var1 to $var3 by func()
$result = $var2 + $var3; // 2 + 1 = 3
echo $result;
?>
```

```
[result]
3
```


- Example of return in script file

```
test.php
```

```
<?php
$var2 = 2;
$var3 = 3;
return ($var2 + $var3); // return 5
?>
```

```
init.php
```

```
<?php
$var1 = include_once "test.php";
echo $var1;
?>
```

```
[result]
5
```

- Example of return in init.php

```
<?php
$var1 = 1;
echo ++$var1; // statement is executed
echo ++$var1; // statement is executed
return;      // ends script
echo ++$var1; // statement will be never executed
?>
```

```
[result]
23
```

Control Structures

include

The include statement includes and evaluates the specified file.

Syntax	Description
include filename;	includes the specified file into current script filename can be referenced by a variable filename is case-sensitive

- Example of include

```
test.php
```

```
<?php
$var1 = 1;
$var2 = 2;
?>
```

```
init.php
```

```
<?php
$var1 = $var2 = 0;
echo $var1 + $var2;
include "test.php"; // includes test.php
echo $var1 + $var2;
?>
```

```
[result]
03
```

- Example of include within functions

If the include occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function. So, they are declared to global variables if you want to use them in global scope.

```
test.php
```

```
<?php
$var1 = 1;
$var2 = 2;
?>
```

```
init.php
```

```
<?php
$var1 = $var2 = 0;
function func()
{
    global $var1;    // only $var1 is global
    include "test.php"; // includes test.php
    echo $var1 + $var2;
}
func();
echo $var1 + $var2;
?>
```

```
[result]
31
```

- Example of include with return

If included file has no return argument, include returns 1 on success. On failure, it causes PHPoC error.

```
test1.php
```

```
<?php
// return $var
$var = 3;
return $var;
?>
```

```
test2.php
```

```
<?php
// no return
$var = 3;
return;
?>
```

```
init.php
```

```
<?php
$var1 = include "test1.php";
echo $var1;
$var2 = include "test2.php";
echo $var2;
?>
```

```
[result]
31
```

Control Structures

include_once

The `include_once` statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the `include` statement, with the only difference being that if the code from a file has already been included, it will not be included again.

Syntax	Description
<code>include_once filename;</code>	includes the specified file into current script filename can be referenced by a variable filename is case-sensitive

- Example of `include_once`

test.php

```
<?php
echo "HelloWrWn";
?>
```

init.php

```
<?php
include "test.php"; // include test.php
include "test.php"; // include test.php again
include_once "test.php"; // not include test.php
?>
```

```
[result]
Hello
Hello
```

Functions

- [User-defined Functions](#)
- [Function Arguments](#)
- [Returning Values](#)
- [Internal Functions](#)

※ PHPoC does not support variable functions.

※ PHPoC does not support anonymous functions.

Functions

User-defined Functions

User-defined function can help to reduce size of source code and to give easy analyzation. You can define frequently used code as a function and then call only function name whenever you need. A function consists of name, argument, statement and return value. The naming rule is the same as variables.

User-defined function name	
The first letter	The rest
Alphabet or _(underscore)	Alphabet, number or _(underscore)

- Structure of Defining Function

Syntax	Description
<pre>function name(argument) { statement; return value; }</pre>	Create function with specified name Multiple or no argument can be allowed return or return value can be omitted

- Structure of Calling Function

Syntax	Description
<pre>name(argument1, argument2, ...);</pre>	argument can be referenced by variable function name is case-sensitive

- Example of Using Function
 User-defined function should be called after defining.

```
<?php

function func()    // define function func()
{
    echo "Hello PHPoC";
}
func();           // call function func()

?>
```

- Example of return value of Function

```
<?php
function func()    // define function func()
{
    return 5;
}
$var = func();    // call function func()
echo $var;

?>
```

```
[result]
5
```

- Example of Using Arguments

Information may be passed to functions via the argument list, which is a commadelimited list of expressions.

```
<?php
function func($arg) // define function func() with $arg
{
    return $arg + 1; // add one to $arg, then return it
}
$var = func(2);    // pass function func() to 2, then receive 3
echo $var;
$var = func($var); // pass function func() to $var(= 3)
echo $var;
$var = func($var+1); // pass function func() to $var+1(=5)
echo $var;

?>
```

```
[result]
346
```


- Example of Recursive Function Call
Functions can be called inside a function including itself.

```
<?php

function func($arg) // define function func() with $arg
{
    if($arg < 6)
    {
        echo "$arg\n"; // print value of $arg
        $arg++;        // increases $arg by one
        func($arg);    // call function func() and pass func() $arg
    }
}
func(1);              // call function func() and pass 1 for argument

?>
```

```
[result]
1
2
3
4
5
```

Functions

Function Arguments

PHPoC supports pass by value, pass by reference and default argument values.

- Pass by Value

By default, function arguments are passed by value. In this case if the value of the argument within the function is changed, it does not get changed outside of the function.

```
<?php

function func($arg1, $arg2) // pass by value
{
    $temp = $arg1;
    $arg1 = $arg2;
    $arg2 = $temp;
    return $arg1 + 1;
}
$var1 = 1;
$var2 = 2;
func($var1, $var2);      // function call
echo "$var1, $var2";    // $var1 and $var2 are not swapped

?>
```

```
[result]
1, 2
```

- Pass by Reference

If arguments are passed by reference, the memory address of argument is passed instead of value. Thus, if the value of the argument within the function is changed, it gets changed outside of the function. To have an argument to a function always passed by reference, prepend an ampersand (&) to the argument name in the function definition.

```
<?php

function func(&$arg1, &$arg2) // pass by reference
{
    $temp = $arg1;
    $arg1 = $arg2;
    $arg2 = $temp;
    return $arg1 + 1;
}
$var1 = 1;
$var2 = 2;
func($var1, $var2);          // function call
echo "$var1, $var2";        // $var1 and $var2 are swapped

?>
```

```
[result]
2, 1
```

- Default Argument Values

A function may define default values for scalar arguments as follows:

```
<?php

function print_str($str = "Hello PHPoC!\n") // set default argument value
{
    echo $str;
}
print_str();                               // call print_str() without argument

?>
```

```
[result]
Hello PHPoC!
```

Functions

Returning Values

Values are returned by using the optional return statement. In general, only one value can be returned except for array type. You should use array type if you want to return multiple values.

- Example of Returning an Array

```
<?php

function func()
{
    $var1 = 1;
    $var2 = 2;
    $var3 = 3;
    $arr = array($var1, $var2, $var3);
    return $arr;
}

$arr = func();
printf("%d, %d, %d\n", $arr[0], $arr[1], $arr[2]);

?>
```

```
[result]
1, 2, 3
```

※ PHPoC returns not NULL but 0 if there is no return statement or the argument of return statement is omitted.

※ PHPoC does not support returning a reference from a function.

Functions

Internal Functions

PHPoC supports various internal functions. Refer to the [PHPoC Internal Functions](#) document about detailed information.

Classes and Objects

※ PHPoC does not support classes and objects.

Namespaces

Overview

Namespaces in PHPoC are designed to solve name collision. Duplicate names are not allowed in the same namespace.

Sharing Namespace

PHPoC provides one namespace for keyword, function and constants. Thus, no duplicate name is allowed when you create one of them.

Appendix

- [Predefined Constants](#)
- [Keyword](#)
- [Restriction about Memory](#)
- [Error Messages](#)

Appendix

Predefined Constants

PHPoC provides predefined constants as following below.

Name	Description	Value
COUNT_NORMAL	Normal Counting for one-dimension array	0
COUNT_RECURSIVE	Recursive Counting for multi-dimension array	1
EPIPE	Broken Pipe, Returning as connection is broken while sending TCP data	32 (0x20)
EBUSY	Device or Resource Busy	16 (0x10)
ENOENT	No file entry	2
FALSE	False	-
M_PI	Pi	≐ 3.141592653589793
M_E	Euler's Constant	≐ 2.718281828459045
MAX_STRING_LEN	Maximum length of a string	1,534
O_NODIE	avoid to terminate script execution: file open	4
PHP_VERSION_ID	PHPoC Version	-
SEEK_SET	File pointer position: at the beginning of file	0
SEEK_CUR	File pointer position: current position of file	1
SEEK_END	File pointer position: at the end of file	2
SSL_CONNECTED	SSL status: connected	19 (0x13)
SSL_CLOSED	SSL status: not connected	0
SSL_LISTEN	SSL status: wait for connection	1
TRUE	True	-
TCP_CLOSED	TCP status: not connected	0
TCP_LISTEN	TCP status: wait for connection	1
TCP_CONNECTED	TCP status: connected	4

Appendix

Keyword

This table below shows predefined keywords in PHPoC. As keywords, functions and constants share namespace, try not to use duplicated names with those keywords when creating functions or constants.

Alphabet	Keywords
a	array
b	bool boolean break
c	case const continue
d	default define die do echo
e	else elseif exit
g	global goto
i	if int integer include include_once
p	print
r	return
s	static string switch
w	while

Appendix

Restriction about Memory

- Number of Variables
PHPoC has limited memory size. So both the number of variables and the length of string variables are limited. Maximum number for variable creation is in inverse proportion to the size of variables.

Appendix

Error Messages

PHPoC prints various error messages out for debugging by console.

Error Messages
address already in use
argument count mismatch
cannot break/continue N level(s)
'case' or 'default' expected
device or resource busy
divided by zero
duplicated name
expression syntax error
file name too long
file not found
function not implemented
integer number too large
invalid argument
invalid constant name
invalid device or address
maximum execution time exceeded
missing operator
missing terminating character ''' or ''''
modifiable value required
only variable can be passed by reference
operation not permitted
out of memory
string too long
syntax error
syntax error, unexpected array [, expecting 'token']
syntax error, unexpected character
syntax error, unexpected 'character' [, expecting 'character']
syntax error, unexpected end of file
syntax error, unexpected 'name' [, expecting 'character']
syntax error, unexpected number [, expecting 'character']
syntax error, unexpected 'operator' [, expecting 'token']
syntax error, unexpected string [, expecting 'character']
syntax error, unexpected 'token' [, expecting 'token']
syntax error, unexpected variable [, expecting 'character']
too many open files
undefined name
undefined offset
unsupported argument type
unsupported operand type
unsupported operator
unsupported pid
unsupported type juggling
'while' expected