

Operators and precedence (O&P) in programming language (PL) design

Najwani Razali, James Noble & Stuart Marshall
Victoria University of Wellington

I want to play a game!



$$6 \div 2 (1 + 2) = ?$$

1

9

Examples of operator precedence bugs in Linux kernel

```
> - *delay = (u16)(val & I40E_PRTDCB_GENC_PFCLDA_MASK >>  
> - I40E_PRTDCB_GENC_PFCLDA_SHIFT);
```

Examples of operator precedence bugs in Linux kernel

```
> - *delay = (u16)(val & I40E_PRTDCB_GENC_PFCLDA_MASK >>  
> - I40E_PRTDCB_GENC_PFCLDA_SHIFT);
```

```
> + *delay = (val & I40E_PRTDCB_GENC_PFCLDA_MASK) >>  
> + I40E_PRTDCB_GENC_PFCLDA_SHIFT;
```

Wrongly used parentheses!!!!

Continue....

- `if (!max77802->opmode[id] == MAX77802_OFF_PWRREQ) {`

Continue....

- `if (! max77802->opmode[id] == MAX77802_OFF_PWRREQ) {`
- + `if (max77802->opmode[id] != MAX77802_OFF_PWRREQ) {`
`dev_warn(&rdev->dev, "%s: is disabled, mode: 0x%x not set\n",`
`rdev->desc->name, mode);`
`return 0;`

CWE-783: Operator Precedence Logic Error

Operator Precedence Logic Error

Weakness ID: 783 (Weakness Variant)

Status: Draft

▼ Description

Description Summary

The program uses an expression in which operator precedence causes incorrect logic to be used.

Extended Description

While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.

```
public double calculateReturnOnInvestment(double currentValue, double initialInvestment) {
    double returnROI = 0.0;

    // calculate return on investment
    returnROI = currentValue - initialInvestment / initialInvestment;

    return returnROI;
}
```

```
returnROI = (currentValue - initialInvestment) / initialInvestment;
```

```
#define FAIL 0
#define SUCCESS 1

...

int validateUser(char *username, char *password) {
    int isUser = FAIL;

    // call method to authenticate username and password
    // if authentication fails then return failure otherwise return success
    if (isUser = AuthenticateUser(username, password) == FAIL) {
        return isUser;
    }
    else {
        isUser = SUCCESS;
    }

    return isUser;
}
```

```
if ((isUser = AuthenticateUser(username, password)) == FAIL) {
```

- Consequences:**
- Confidentiality
 - Integrity
 - Availability

Is Knight's \$440 million glitch the costliest computer bug ever?

By Brian Patrick Eha @CNNTech August 9, 2012: 10:22 AM ET

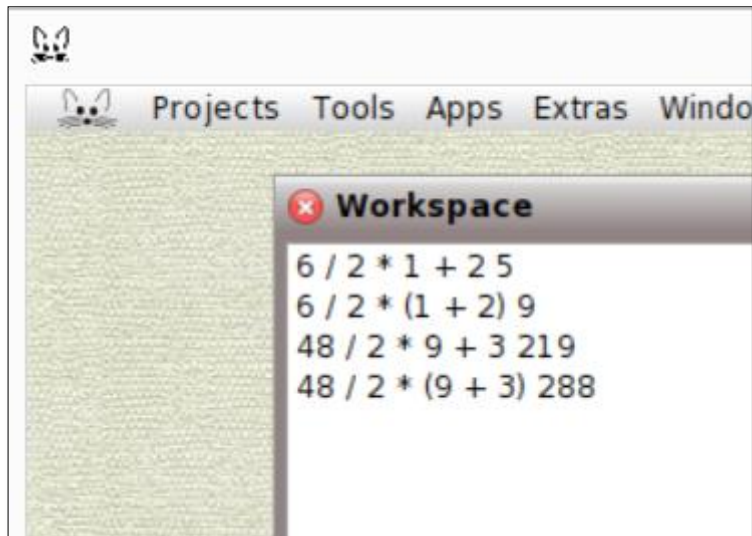
Recommend 154



SPENCER PLATT/GETTY IMAGES

Knight Capital's computer bug cost the firm \$440 million, making it one of history's most expensive software glitches.

Smalltalk



Order of evaluation:

1. Parentheses
2. Unary message sends (left to right)
3. Binary message sends (left to right)
4. Keyword message sends
5. Assignments
6. Returns

LISP

```
LispWorks Personal Edition 6.1.1 - [Liste
File Edit Tools Works Debug History Wind
5
CL-USER 17 : 2 > (+ (* 1 (/ 6 2)))
3
CL-USER 18 : 2 > (+ 2 (* 1 (/ 6 2)))
5
CL-USER 19 : 2 > (* (/ 6 2) (+ 1 2))
9
CL-USER 20 : 2 >

CL-USER 20 : 2 > (/ 6 2 (+ 1 2))
1
```

1. Required parentheses
2. Prefix expression

Pyret

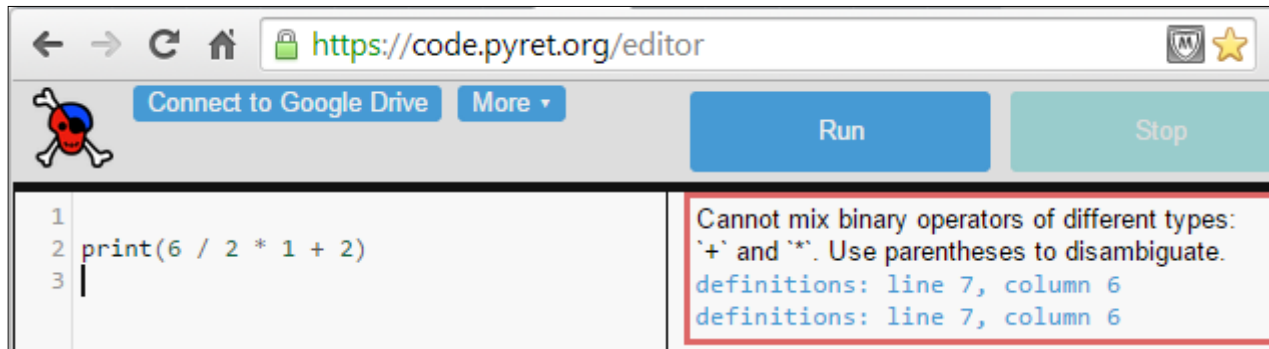


A screenshot of the Pyret online editor interface. The browser address bar shows `https://code.pyret.org/editor`. The interface includes a "Connect to Google Drive" button, a "More" dropdown, and a "Run" button. The code editor contains three lines of Python code:

```
1 print((6 / 2) * (1 + 2))
2 print((6 / (2 * 1)) + 2)
3 print(6 / ((2 * 1) + 2))
4
```

The output area on the right shows the results of the execution:

```
9
5
3/2
```



A screenshot of the Pyret online editor interface showing a syntax error. The browser address bar shows `https://code.pyret.org/editor`. The interface includes a "Connect to Google Drive" button, a "More" dropdown, and "Run" and "Stop" buttons. The code editor contains the following code:

```
1
2 print(6 / 2 * 1 + 2)
3
```

The error message displayed in a red-bordered box is:

```
Cannot mix binary operators of different types:
`+` and `*`. Use parentheses to disambiguate.
definitions: line 7, column 6
definitions: line 7, column 6
```

Pascal - $6 / 2 * (1 + 2)$

```
Free Pascal
File Edit Search Run Compile Debug Test
var
a,b,c,d,e : real;
begin
a:=6;
b:=2;
d:=1;
e:=2;
c:= a / b * (d + e);
writeln(' Line 1 - Value of c is',c);
end.
```

```
Free Pascal
File Edit Search Run Compile Debug Test
User screen
Free Pascal IDE Version 1.0.12 [2014/03/
Compiler Version 2.6.4
GDB Version GDB 7.4
Using configuration files from: C:\FPC\2
Running "c:\fpc\2.6.4\bin\i386-win32\test.
Line 1 - Value of c is 5.000000000000000E+
Running "c:\fpc\2.6.4\bin\i386-win32\test.
Line 1 - Value of c is 9.000000000000000E+
```

Operator	Precedence
~, not,	Highest
*, /, div, mod, and, &	
!, +, -, or,	
=, <>, <, <=, >, >=, in	
or else, and then	Lowest

Python

```
C:\Pytho
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct
064)] on win32
Type "help", "copyright", "credits" or
>>>
>>> a=0
>>> a=6/2*1+2
>>> print("a is",a)
a is 5.0
>>> a=0
>>> a=6/2*(1+2)
>>> print("a is",a)
a is 9.0
>>>
```

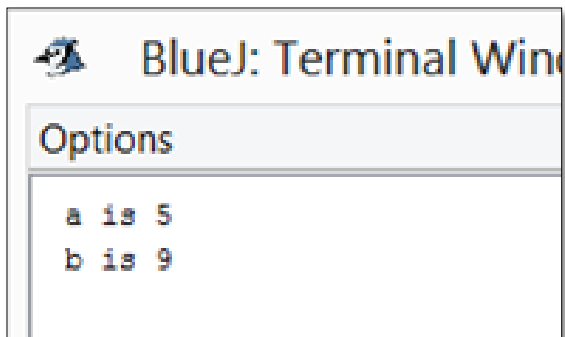
Operator	Description
lambda	Lambda expression
if - else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, including membership tests and identity tests
	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, //, %	Multiplication, division, remainder [8]
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation [9]
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...}, `expressions...`	Binding or tuple display, list display, dictionary display, string conversion

JAVA

```
public class test2
{
    // instance variables - replace the example
    public static void main(){
        int a, b = 0;

        a = 6 / 2 * 1 + 2;
        System.out.println(" a is " + a);

        b = 6 / 2 * (1 + 2);
        System.out.println(" b is " + b);
    }
}
```



BlueJ: Terminal Window

Options

```
a is 5
b is 9
```

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ -- ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

C

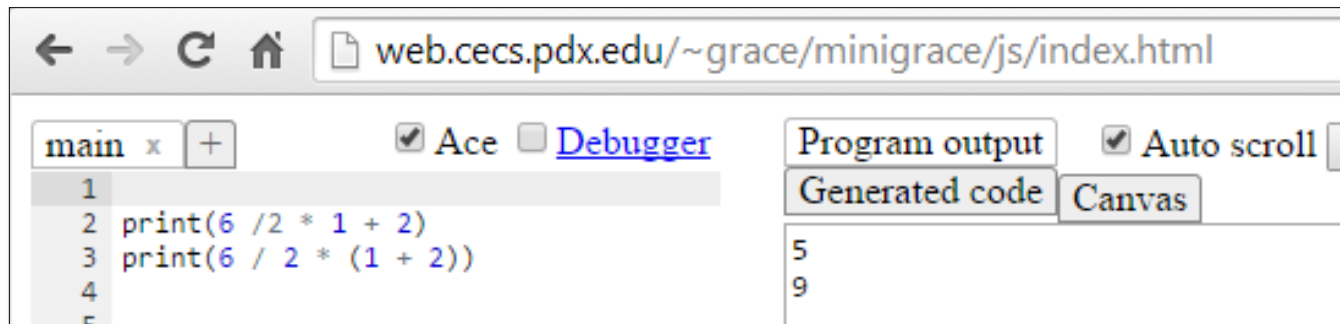
```
Compile & Execute main.c input.txt
1 #include <stdio.h>
2 #include <string.h>
3
4 main()
5 {
6     int a, b = 0;
7
8     a = 6 / 2 * 1 + 2;
9     printf("a is %d\n", a);
10
11    b = 6 / 2 * (1 + 2);
12    printf("b is %d\n", b);
13
14 }
15
```

```
Result
Compiling the source code....
$gcc main.c -o demo -lm -pthread

Executing the program....
$demo
a is 5
b is 9
```

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

GRACE



The screenshot shows a web browser window with the address bar containing the URL `web.cecs.pdx.edu/~grace/minigrace/js/index.html`. The browser has a tab titled "main" and a toolbar with navigation icons. Below the address bar, there is a code editor with the following code:

```
1  
2 print(6 / 2 * 1 + 2)  
3 print(6 / 2 * (1 + 2))  
4  
5
```

To the right of the code editor, there are two panels. The top panel is titled "Program output" and has a checked "Auto scroll" checkbox. The bottom panel is titled "Generated code" and has a "Canvas" button. The "Program output" panel displays the following output:

```
5  
9
```

Summary

Programming languages	Expression	Output	Rules
Pascal, Python, Java, C, C++ & Grace	$6 / 2 * (1 + 2)$	9	Precedence
	$6 / 2 * 1 + 2$	5	
	$2 + 6 / 2 + 1$	5	
Smalltalk	$6 / 2 * (1 + 2)$	9	Left-to-right
	$6 / 2 * 1 + 2$	5	
	$2 + 6 / 2 * 1$	4	
LISP	$(* (/ 6 2) (+ 1 2))$	9	Parenthesis & prefix
	$(/ 6 2 (+ 1 2))$	1	
Pyret	$((6 / 2) * (1 + 2))$	9	Parenthesis

Pilot study

- Participants: 1st year student in computer science, currently taking Java course from University Utara Malaysia (UUM)
- There are 29 participants
- Goals:
 - To understand and evaluate students interpretation on expression of operators and precedence

Initial findings

Question 1

Add parentheses for the expression based on their precedence in order to remove ambiguity and give the output:

$$1 + 2 * 3 - 4$$

No	Pattern of expression	answers	Total no of pattern
1	$1 + (2 * 3) - 4$	3	12
2	$1 + (2 * 3) - 4$ $1 + 6 - 4$ $7 - 4$	3	13
3	$(1 + (2 * 3)) - 4$	3	3

Initial findings

Question 2

Add parentheses for the expression based on their precedence in order to remove ambiguity and give the output.

```
int a = 1;  
int b = 2;  
int c = 3;
```

$a + b++ * c / a * b$



$a + (((b++) * c) / a) * b = 19$

No	Pattern of expression	answers	Total no of pattern
1	$a + ((b++ * c) / (a * b))$	3	6
2	$a + (b++ * c) / (a * b)$	3	4
3	$a + ((b++) * c) / (a * b)$	3	3
4	$(a + ((b++ * c) / (a * b)))$	3	1
5	$a + (((b++) * c) / (a * b))$	3	2
6	$a + (((b++) * c) / a) * b$	19	2
7	$a + (((b++ * c) / a) * b)$	19	1
8	$((a + b++) * c) / a * b$	27	1
9	$a + (b++) * c / a * b$	19	1

Initial findings

Question 2

Add parentheses for the expression based on their precedence in order to remove ambiguity and give the output.

```
int a = 1;  
int b = 2;  
int c = 3;
```

$a + b++ * c / a * b$



$a + (((b++) * c) / a) * b = 19$

No	Pattern of expression	answers	Total no of pattern
1	$a + ((b++ * c) / (a * b))$	3	6
2	$a + (b++ * c) / (a * b)$	3	4
3	$a + ((b++) * c) / (a * b)$	3	3
4	$(a + ((b++ * c) / (a * b)))$	3	1
5	$a + (((b++) * c) / (a * b))$	3	2
6	$a + (((b++) * c) / a) * b$	19	2
7	$a + (((b++ * c) / a) * b)$	19	1
8	$((a + b++) * c) / a * b$	27	1
9	$a + (b++) * c / a * b$	19	1

Objectives

- To understand student conceptual understanding on O&P
- To design an empirical framework based on users interpretation on operators & precedence
- To design a tools based on the empirical framework

Goals

- Help novices (who has no background in programming) and computer science students to understand operators & precedence
- Help language designers by providing a framework as a guideline during their design stage

Thank you