# Backport191 - Bug #2737

## StringConstant +"string literal" (unspaced) raises exception

02/12/2010 03:54 AM - jtlapp (Joe Lapp)

| | |
|---|---|
| **Status:** | Rejected |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |
| **ruby -v:** | 1.9.1p376 |

**Description**

=begin
(I may have originally posted this in the wrong place. It also appears at http://redmine.ruby-lang.org/issues/show/2736)

When concatenating to a string constant, if a space does not follow '+' operator, Ruby raises an exception complaining that the unary '+' operator on strings is not defined.

The following code:

Konst = 'A'
Konst +'B'

...raises the following exception in IRB:

NoMethodError: undefined method +@' for "B":String
from (irb):3
from /usr/local/bin/irb:12:in'

Note that the following code -- adding a space after the '+' -- works as expected:

Konst = 'A'
Konst + 'B'
=> "AB"

The behavior appears to be a bug because the following does work as expected:

v = 'A'
v +'B'
=> "AB"

This also works:

'A' +'B'
=> "AB"

I discovered the problem when I changed working code by replacing an r-value variable with a constant. Replacing a literal or an r-value variable with a constant shouldn't break anything if the types match.

I found he problem in Ruby 1.9.1p376, but also reproduced it in 1.9.1p243. Friends are reporting that Ruby 1.8.7 (MRI) and JRuby 1.4 also have the problem.
=end

### History

**#1 - 02/12/2010 07:57 AM - nobu (Nobuyoshi Nakada)**

*- Status changed from Open to Rejected*

=begin
Use -w option.
=end

**#2 - 02/12/2010 08:21 AM - soverdor (Sam Overdorf)**

=begin
None of the tar balls for the year 2010 will pass the "make test".

tar xvzf stable-snapshot.tar.gz
cd ruby
./configure
make
make test

The tar balls dated before 2010 seem to work fine.

Is there something wrong with all of the 2010 or am I just doing something incorrectly.

I am using Linux RedHat 4 Update 3 on my Intel computer.

Here is the output of "ruby -v" and "make test".

u0140:[11]/tmp/ruby> ./ruby -v
ruby 1.8.8dev (2010-01-01) [x86_64-linux]

u0140:[11]/tmp/ruby> make test
./sample/test.rb:142: warning: multiple values for a block parameter (0 for 1)
from ./sample/test.rb:142
not ok assignment 95 -- ./sample/test.rb:142
not ok assignment 131 -- ./sample/test.rb:182
not ok assignment 169 -- ./sample/test.rb:224
not ok assignment 198 -- ./sample/test.rb:256
not ok assignment 271 -- ./sample/test.rb:351
not ok assignment 291 -- ./sample/test.rb:373
not ok assignment 342 -- ./sample/test.rb:430
not ok assignment 362 -- ./sample/test.rb:453
test failed
make: *** [test] Error 1

Thanks for the help,
Sam

=end

#### #3 - 02/17/2010 02:21 PM - jtlapp (Joe Lapp)

=begin
Okay, I applied -w on the first of the above examples and got this warning:
"warning: ambiguous first argument; put parentheses or even spaces"

Can you tell me what the ambiguity is?  The only valid interpretation of this that anybody in our local Ruby user group could see was string concatenation.

I'm running Rails, which must not have -w set.  I'll have to look for that and hope Rails doesn't spit out warnings.

Thanks!
~joe
=end

#### #4 - 02/17/2010 02:47 PM - coatl (caleb clausen)

=begin
This line:
Konst +'B'
could be interpreted 2 ways. You expect + to be binary:
(Konst + 'B')
but ruby always chooses this way:
Konst(+'B')
In other words, it sees a method call. Local variables are kind of a special case...
=end

#### #5 - 02/17/2010 02:54 PM - murphy (Kornelius Kalnbach)

=begin
On 17.02.10 06:21, Joe Lapp wrote:

> Can you tell me what the ambiguity is?  The only valid interpretation
> of this that anybody in our local Ruby user group could see was
> string concatenation.
> a +b could also be understood as a(+b) or a(b.+@()), where +@ is the

unary plus operator. of course, it's not defined for Strings, but the
parser can't determine that statically.

So, the warning is absolutely to the point.

[murphy]

=end

#### #6 - 02/20/2010 09:38 AM - jtlapp (Joe Lapp)

=begin
Wow, great responses!  After the terse rejection, I wasn't expecting anything.

If I define a method I also get that +@ exception:
method +'b'

But as Caleb suggested, the following works:
local +'b'

So then the question is, why *should* this work for locals but not for constants?

Also, it would seem that +'s' never has two functionally valid interpretations - it's ambiguous to the implementation, not to end users.  A choice has been made to impose an implementation limitation on the end user.  I understand the need to do this sometimes to keep implementation complexity under control, but if this is the real reason for keeping this behavior, it might be better to pile this issue on the might-fix-someday list.

I have a lot of Java, C, and C++ under my belt, but I'm coming from a long, painful stint with PHP.  I'm really tired of needless idiosyncrasies.  This is my first one in Ruby, so I'm not complaining, but the Ruby community ought to at least hold the ideal that these might someday be ironed out.

Thanks for taking the time to answer my question!

=end

#### #7 - 02/20/2010 10:57 AM - murphy (Kornelius Kalnbach)

=begin
I'd argue that this is ambiguous to the reader, too. We should not make a special case for the + operator combined with a literal; the parser has to decide the ambiguity before even getting to the literal.

locals are a special case, because Ruby determines them statically (everything that has been assigned to in the current scope). constants, on the other hand, cannot be determined that way, so Ruby has to decide whether it treats an upper-case identifier as a constant or a method. (There are upper-case methods like Kernel#Array.)
=end