Ruby - Bug #16841

Some syntax errors are thrown from compile.c

05/08/2020 11:30 AM - ibylich (Ilya Bylich)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.7.1p83 (2020-03-31 revision a0c7c23c9c) [x86_64-darwin19]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

compile.c has a few places where it raises SyntaxError. Because of that ruby -c, Ripper and RubyVM::AbstractSyntaxTree don't catch them:

```
> ruby -vce 'class X; break; end'
ruby 2.7.1p83 (2020-03-31 revision a0c7c23c9c) [x86_64-darwin19]
Syntax OK

2.7.1 :001 > require 'ripper'
 => false
2.7.1 :002 > Ripper.sexp('class X; break; end')
 => [:program, [[:class, [:const_ref, [:@const, "X", [1, 6]]], nil, [:bodystmt, [[:void_stmt], [
:break, []]], nil, nil, nil]]]]
2.7.1 :003 > RubyVM::AbstractSyntaxTree.parse('class X; break; end')
 => #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:0-1:19>
```

I've changed locally assert_valid_syntax to use RubyVM::AbstractSyntaxTree for parsing and got ~5 failing tests (like Invalid next/break/redo and one more related to pattern matching).

I started playing with parse.y yesterday but then I quickly realized that to reject such code we need some information about scopes (basically something like a stack of scopes).
This way we could reject break if we are not directly in block/lambda/loop.
But then I realized that we can't properly collect stack elements (by doing something like scopes.push(<scope name>)) for post-loops:

```
break while true
```

because the rule is

```
| stmt modifier_while expr_value
```

and adding something like { push_context(p, IN_LOOP) } in front of it causes a ton of shift/reduce conflicts (which makes a lot of sense). Is it the reason why these cases are rejected during compilation?

If so, is there any simple way to reject it in the grammar? Maybe some kind of the AST post-processor? But then I guess we need a separate version for Ripper, right?

History

#1 - 05/27/2020 08:43 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Feedback

This doesn't seem to be a bug, it is by design. Trying to move all syntax errors into the parser is probably too difficult to justify the effort even if it is possible.

If you want to more fully check for valid syntax without having to execute code, you can probably use the same approach used in rdoc:

```
  check = lambda do |code|
    begin
      eval "BEGIN {return true}\n#{code}"
    rescue SyntaxError
      false
    end
  end
  check.("class X; end") # => true
  check.("class X; break; end") # => false
```

Do you think that will work for your purposes?

#2 - 06/10/2020 05:28 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Feedback to Closed

#3 - 06/10/2020 07:03 PM - ibylich (Ilya Bylich)

Thanks for looking into this issue.

> Do you think that will work for your purposes?

Not really. I'm working on the 3d party parser and I was thinking about whether I need to backport these errors. If I'd need it in some other project (that is not about parsing) I'd definitely do it as you suggested (maybe I'd use eval("throw :tag; #{code.dump}") or something similar, the idea would be the same). I guess I'll simply ignore errors that come after parsing, thank you!

#4 - 06/10/2020 07:13 PM - jeremyevans0 (Jeremy Evans)

ibylich (Ilya Bylich) wrote in #note-3:

> Thanks for looking into this issue.
>
> > Do you think that will work for your purposes?
>
> Not really. I'm working on the 3d party parser and I was thinking about whether I need to backport these errors. If I'd need it in some other project (that is not about parsing) I'd definitely do it as you suggested (maybe I'd use eval("throw :tag; #{code.dump}") or something similar, the idea would be the same). I guess I'll simply ignore errors that come after parsing, thank you!

Just FYI, there are problems with eval("throw :tag; #{code.dump}"):

1. code.dump is the same as code.inspect, so it evals a string literal, and will never catch a syntax error.
2. Assuming you switch to #{code}, it is unsafe if code is not in your control, as code can contain BEGIN blocks evaluated before the throw :tag (resulting in an RCE vulnerability).