Ruby Float() Bug

Converting an invalid string into a float

Sam Napolitano

Float examples

```
irb(main):0:0> Float("3.14")
=> 3.14
irb(main):0:0> Float("3.14a")
ArgumentError: invalid value for Float(): "3.14a"
irb(main):0:0> Float("0xFF")
=> 255.0
```

ruby_bug.rb

```ruby
# ...
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456

number = ""
bad_char = 'a'

def string_to_float(arg)
  Float(arg) rescue nil
end

1.upto(72).each do |j|
  number     = number + (j % 10).to_s
  bad_number = number + bad_char
```

initialize

ruby_bug.rb

```ruby
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456
# 123456789012345678901234567890123456789012345678901234567890123456

number = ""
bad_char = 'a'

def string_to_float(arg)
  Float(arg) rescue nil
end

1.upto(72).each do |j|
  number     = number + (j % 10).to_s
  bad_number = number + bad_char

  if flt = string_to_float(bad_number)
    puts "#{bad_number} len=#{number.size} val=#{flt}"
  end
end
```

convert string to float

ruby_bug.rb

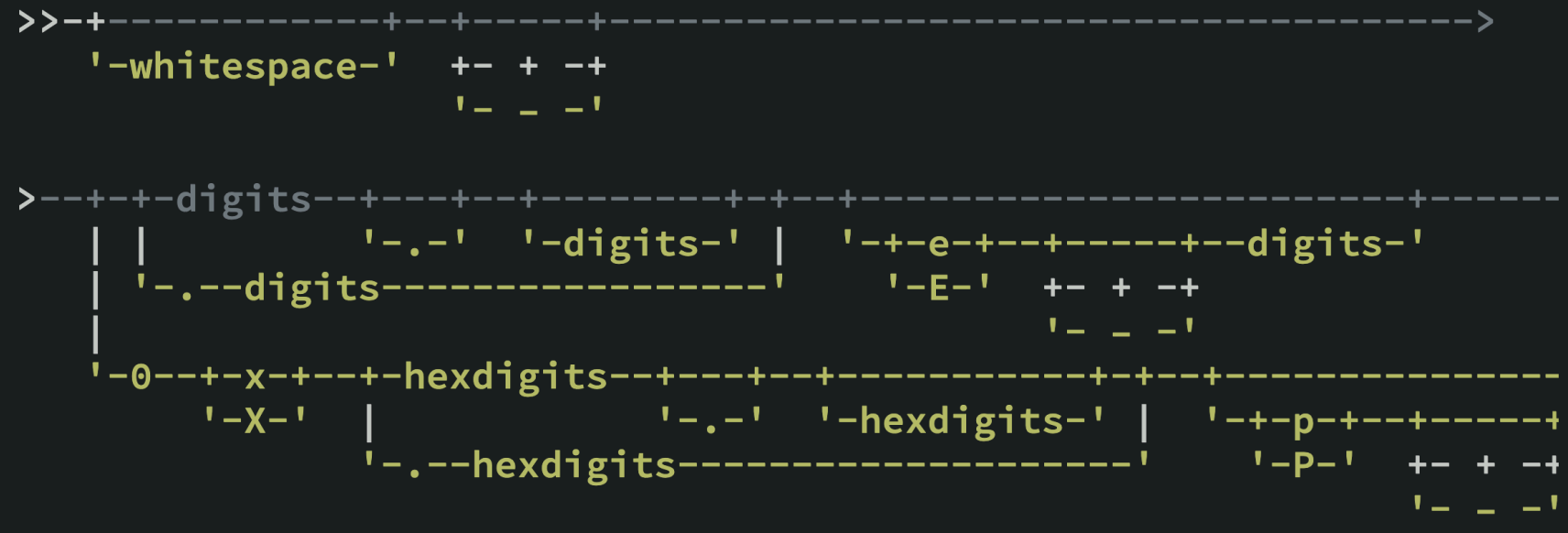
```ruby
number = ""
bad_char = 'a'

def string_to_float(arg)
  Float(arg) rescue nil
end

1.upto(72).each do |j|
  number     = number + (j % 10).to_s
  bad_number = number + bad_char

  if flt = string_to_float(bad_number)
    puts "#{bad_number} len=#{number.size} val=#{flt}"
  else
    puts "#{bad_number} len=#{number.size} val=error"
  end
end
```

loop growing string one digit at a time

ruby_bug.rb output in ruby 2.2.7

```
1a len=1 val=error
12a len=2 val=error
123a len=3 val=error
...
1234567890123...01234567a len=67 val=error
1234567890123...012345678a len=68 val=error
1234567890123...0123456789a len=69 val=1.234567890123457e+68
1234567890123...01234567890a len=70 val=1.234567890123457e+68
```

strtod() specification



https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rtref/strtod.htm

Float uses strtod() - more examples

```
01  irb(main):0:0> Float("1.2345e+3")
02  => 1234.5
03  irb(main):0:0> Float("1.2345e-3")
04  => 0.0012345
05  irb(main):0:0> Float("  0x1p3")
06  => 8.0
07  irb(main):0:0> Float("  0x2p3")
08  => 16.0
09  irb(main):0:0> "%b" % Float("  0x1p3")
10  => "1000"
```

```
01  double
02    strtod(const char *restrict nptr, char **restrict endptr);
```

If endptr is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

If no conversion is performed, zero is returned and the value of nptr is stored in the location referenced by endptr.

object-v63130.c

```
01  /**********************************************************************
02
03    object.c -
04
05    $Author: nobu $
06    created at: Thu Jul 15 12:01:24 JST 1993
07
08    Copyright (C) 1993-2007 Yukihiro Matsumoto
09    Copyright (C) 2000  Network Applied Communication Laboratory, Inc.
10    Copyright (C) 2000  Information-technology Promotion Agency, Japar
11
12  **********************************************************************
13
14  #include "ruby/encoding.h"
15  #include "ruby/st.h"
16  #include "ruby/util.h"
17  #include "internal.h"
18  #include <stdio.h>
```

object-v63130.c

```
3223            break;
3224          default:
3225            UNREACHABLE;
3226        }
3227
3228        return rb_convert_to_integer(arg, base, opts_exception_p(opts)
3229    }
3230
3231    static double
3232    rb_cstr_to_dbl_raise(const char *p, int badcheck, int raise, int *
3233    {
3234        const char *q;
3235        char *end;
3236        double d;
3237        const char *ellipsis = "";
3238        int w;
3239        enum {max_width = 20};
3240    #define OutOfRange() ((end - p > max_width) ? \
```

p=string, badcheck=true, raise=true, error=NULL

object-v63130.c

```
3244        if (!p) return 0.0;
3245        q = p;
3246        while (ISSPACE(*p)) p++;
3247
3248        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X'))
3249            return 0.0;
3250        }
3251
3252        d = strtod(p, &end);
3253        if (errno == ERANGE) {
3254            OutOfRange();
3255            rb_warning("Float %.*s%s out of range", w, p, ellipsis);
3256            errno = 0;
3257        }
3258        if (p == end) {
3259            if (badcheck) {
3260              bad:
```

first attempt to convert string

object-v63130.c

```
3262                rb_invalid_str(q, "Float()");
3263            else {
3264                if (error) *error = 1;
3265                return 0.0;
3266            }
3267        }
3268        return d;
3269    }
3270    if (*end) {
3271        char buf[DBL_DIG * 4 + 10];
3272        char *n = buf;
3273        char *e = buf + sizeof(buf) - 1;
3274        char prev = 0;
3275
3276        while (p < end && n < e) prev = *n++ = *p++;
3277        while (*p) {
3278            if (*p == '_') {
```

'end' is pointing to bad char 'a'

object-v63130.c

```
3264                if (error) *error = 1;
3265                return 0.0;
3266            }
3267        }
3268        return d;
3269    }
3270    if (*end) {
3271        char buf[DBL_DIG * 4 + 10];
3272        char *n = buf;
3273        char *e = buf + sizeof(buf) - 1;
3274        char prev = 0;
3275
3276        while (p < end && n < e) prev = *n++ = *p++;
3277        while (*p) {
3278            if (*p == '_') {
3279                /* remove an underscore between digits */
3280                if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3281                    if (badcheck) goto bad;
```

fixed buffer? Hmm... what size?

object-v63130.c

```
3268            return d;
3269        }
3270        if (*end) {
3271            char buf[DBL_DIG * 4 + 10];
3272            char *n = buf;
3273            char *e = buf + sizeof(buf) - 1;
3274            char prev = 0;
3275
3276            while (p < end && n < e) prev = *n++ = *p++;
3277            while (*p) {
3278                if (*p == '_') {
3279                    /* remove an underscore between digits */
3280                    if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3281                        if (badcheck) goto bad;
3282                        break;
3283                    }
3284                }
```

load buf with chars stopping at 'a'; keep track of prev char

object-v63130.c

```
3272        char *n = buf;
3273        char *e = buf + sizeof(buf) - 1;
3274        char prev = 0;
3275
3276        while (p < end && n < e) prev = *n++ = *p++;
3277        while (*p) {
3278            if (*p == '_') {
3279                /* remove an underscore between digits */
3280                if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3281                    if (badcheck) goto bad;
3282                    break;
3283                }
3284            }
3285            prev = *p++;
3286            if (n < e) *n++ = prev;
3287        }
3288        *n = '\0';
3289        p = buf;
```

remove underscore; 8_00_0 == 8_000

object-v63130.c

```
3278            if (*p == '_') {
3279                /* remove an underscore between digits */
3280                if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3281                    if (badcheck) goto bad;
3282                    break;
3283                }
3284            }
3285            prev = *p++;
3286            if (n < e) *n++ = prev;
3287        }
3288        *n = '\0';
3289        p = buf;
3290
3291        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X
3292            return 0.0;
3293        }
3294
```

stop when n is at end of buf - the bug!

object-v63130.c

```
3280                if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3281                    if (badcheck) goto bad;
3282                    break;
3283                }
3284            }
3285            prev = *p++;
3286            if (n < e) *n++ = prev;
3287        }
3288        *n = '\0';
3289        p = buf;
3290
3291        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X
3292            return 0.0;
3293        }
3294
3295        d = strtod(p, &end);
3296        if (errno == ERANGE) {
3297            OutOfRange():
```

terminate string and assign p

object-v63130.c

```
3287        }
3288        *n = '\0';
3289        p = buf;
3290
3291        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X
3292            return 0.0;
3293        }
3294
3295        d = strtod(p, &end);
3296        if (errno == ERANGE) {
3297            OutOfRange();
3298            rb_warning("Float %.*s%s out of range", w, p, ellipsis
3299            errno = 0;
3300        }
3301        if (badcheck) {
3302            if (!end || p == end) goto bad;
3303            while (*end && ISSPACE(*end)) end++;
```

convert string using normalized buffer

object.c on trunk

```
01  /**********************************************************************
02
03    object.c -
04
05    $Author: ko1 $
06    created at: Thu Jul 15 12:01:24 JST 1993
07
08    Copyright (C) 1993-2007 Yukihiro Matsumoto
09    Copyright (C) 2000  Network Applied Communication Laboratory, Inc.
10    Copyright (C) 2000  Information-technology Promotion Agency, Japar
11
12  **********************************************************************
13
14  #include "ruby/encoding.h"
15  #include "ruby/st.h"
16  #include "ruby/util.h"
17  #include "internal.h"
18  #include <stdio.h>
```

object.c on trunk

```
3264                return 0.0;
3265            }
3266        }
3267        return d;
3268    }
3269    if (*end) {
3270        char buf[DBL_DIG * 4 + 10];
3271        char *n = buf;
3272        char *const init_e = buf + DBL_DIG * 4;
3273        char *e = init_e;
3274        char prev = 0;
3275        int dot_seen = FALSE;
3276
3277        switch (*p) {case '+': case '-': prev = *n++ = *p++;}
3278        if (*p == '0') {
3279            prev = *n++ = '0';
3280            while (*++p == '0');
```

additional new locals

object.c on trunk

```
3271        char *n = buf;
3272        char *const init_e = buf + DBL_DIG * 4;
3273        char *e = init_e;
3274        char prev = 0;
3275        int dot_seen = FALSE;
3276
3277        switch (*p) {case '+': case '-': prev = *n++ = *p++;}
3278        if (*p == '0') {
3279            prev = *n++ = '0';
3280            while (*++p == '0');
3281        }
3282        while (p < end && n < e) prev = *n++ = *p++;
3283        while (*p) {
3284            if (*p == '_') {
3285                /* remove an underscore between digits */
3286                if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3287                    if (badcheck) goto bad;
```

handle +/- and leading zeros

object.c on trunk

```
3283            while (*p) {
3284                if (*p == '_') {
3285                    /* remove an underscore between digits */
3286                    if (n == buf || !ISDIGIT(prev) || (++p, !ISDIGIT(*
3287                        if (badcheck) goto bad;
3288                        break;
3289                    }
3290                }
3291                prev = *p++;
3292                if (e == init_e && (prev == 'e' || prev == 'E' || prev
3293                    e = buf + sizeof(buf) - 1;
3294                    *n++ = prev;
3295                    switch (*p) {case '+': case '-': prev = *n++ = *p+
3296                    if (*p == '0') {
3297                        prev = *n++ = '0';
3298                        while (*++p == '0');
3299                    }
```

assign prev to current char

object.c on trunk

```
3288                break;
3289            }
3290        }
3291        prev = *p++;
3292        if (e == init_e && (prev == 'e' || prev == 'E' || prev
3293            e = buf + sizeof(buf) - 1;
3294            *n++ = prev;
3295            switch (*p) {case '+': case '-': prev = *n++ = *p+
3296            if (*p == '0') {
3297                prev = *n++ = '0';
3298                while (*++p == '0');
3299            }
3300            continue;
3301        }
3302        else if (ISSPACE(prev)) {
3303            while (ISSPACE(*p)) ++p;
3304            if (*p) {
3305                if (badcheck) goto bad;
```

handle exponentiation validation

object.c on trunk

```
3297                prev = *n++ = '0';
3298                while (*++p == '0');
3299            }
3300            continue;
3301        }
3302        else if (ISSPACE(prev)) {
3303            while (ISSPACE(*p)) ++p;
3304            if (*p) {
3305                if (badcheck) goto bad;
3306                break;
3307            }
3308        }
3309        else if (prev == '.' ? dot_seen++ : !ISDIGIT(prev)) {
3310            if (badcheck) goto bad;
3311            break;
3312        }
3313        if (n < e) *n++ = prev;
```

trailing whitespace

object.c on trunk

```
3302            else if (ISSPACE(prev)) {
3303                while (ISSPACE(*p)) ++p;
3304                if (*p) {
3305                    if (badcheck) goto bad;
3306                    break;
3307                }
3308            }
3309            else if (prev == '.' ? dot_seen++ : !ISDIGIT(prev)) {
3310                if (badcheck) goto bad;
3311                break;
3312            }
3313            if (n < e) *n++ = prev;
3314        }
3315        *n = '\0';
3316        p = buf;
3317
3318        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X
3319            return 0.0;
```

validate one dot & validate digits

object.c on trunk

```
3309            else if (prev == '.' ? dot_seen++ : !ISDIGIT(prev)) {
3310                if (badcheck) goto bad;
3311                break;
3312            }
3313            if (n < e) *n++ = prev;
3314        }
3315        *n = '\0';
3316        p = buf;
3317
3318        if (!badcheck && p[0] == '0' && (p[1] == 'x' || p[1] == 'X
3319            return 0.0;
3320        }
3321
3322        d = strtod(p, &end);
3323        if (errno == ERANGE) {
3324            OutOfRange();
3325            rb_warning("Float %.*s%s out of range", w, p, ellipsis
3326            errno = 0;
```

same as before

ruby_bug.rb output in ruby 2.6preview2

```
1a len=1 val=error
12a len=2 val=error
123a len=3 val=error
...
1234567890123...01234567a len=67 val=error
1234567890123...012345678a len=68 val=error
1234567890123...0123456789a len=69 val=error
1234567890123...01234567890a len=70 val=error
```

Thanks!

Questions?

https://github.com/samiam/ruby_bug_14729

https://gitpitch.com/samiam/ruby_bug_14729