Parsing a string as a 64-bit integer, somehow

devblogs.microsoft.com/oldnewthing/20150629-00

June 29, 2015

Raymond Chen

Today's Little Program takes a string and tries to parse it as a 64-bit integer in formats that a programmer would likely encounter.

Here's a first stab:

```
using System;
using System.Globalization;

class Program
{
 static long ParseLongSomehow(string s)
 {
  if (s.StartsWith("0x", StringComparison.OrdinalIgnoreCase)) {
   return long.Parse(s.Substring(2), NumberStyles.HexNumber);
  } else {
   return long.Parse(s);
  }
 }

 public static void Main(string[] args)
 {
  long value = ParseLongSomehow(args[0]);
  Console.WriteLine(value);
  Console.WriteLine("0x{0:X}", value);
 }
}
```

If the string begins with `0x`, then we treat the rest of the argument as a hex value; otherwise, we treat it as a decimal value.

Unfortunately, this doesn't work if the input is `9223372036854775808`, which is the value of `1 << 63`, a value that is representable as a 64-bit unsigned value but not a 64-bit signed value.

Our problem statement was pretty vague, so let's write a functional specification. It helps to know what problem you're solving before you start to solve it. Otherwise, you're just flailing around writing code before you have a plan. When I tried to solve this problem, I flailed around a bit until I realized that I didn't have a spec.

What formats would a programmer be likely to encounter as the string representation of a 64-bit integer?

- `0x1234` : 64-bit number in hex format, case-insensitive. The value can range from 0 to `UInt64.MaxValue` .
- `12345` : 64-bit unsigned number in decimal format. The value can range from 0 to `UInt64.MaxValue` .
- `-12345` : 64-bit signed number in decimal format. The value can range from `Int64.MinValue` to `Int64.MaxValue` .
- Other formats may be permitted, but you need to support at least the above.

Writing down exactly what I was doing and what I wasn't doing was the part that solved my flailing. I had been worrying about things like `-0x12345` and `-9223372036854775809` and `99999999999999999999` , even though those numbers would not be something a programmer would be likely to encounter.

From the specification we can develop our algorithm.

- If the string begins with `0x` , then parse what's left as an unsigned 64-bit hexadecimal number.
- If the string begins with a minus sign, then parse it as a 64-bit signed number in decimal format.
- If the string does not begin with a minus sign, then parse it as a 64-bit unsigned number in decimal format.

And that is pretty easy to implement.

```
static long ParseLongSomehow(string s)
{
 if (s.StartsWith("0x", StringComparison.OrdinalIgnoreCase)) {
  return long.Parse(s.Substring(2), NumberStyles.HexNumber);
 } else if (s[0] == '-') {
  return long.Parse(s);
 } else {
  return (long)ulong.Parse(s);
 }
}
```

Note that we are a little sloppy with our treatment of whitespace. We accept leading and trailing spaces on decimal values, and allow trailing spaces on hex values (and even allow spaces between the `0x` and the first hex digit). That's okay, because the spec allows us to accept formats beyond the ones listed.

Now, for bonus points, let's revise the functional specification a little bit, specifically by adding another case:

> `0x12`3456789A`: 64-bit number in hex format, case-insensitive, with backtick separating the upper 32 bits from the lower 32 bits.

This is the format used by the Windows debugger engine.

```
static long ParseLongSomehow(string s)
{
 if (s.StartsWith("0x", StringComparison.OrdinalIgnoreCase)) {
  return long.Parse(s.Substring(2).Replace("`", ""), NumberStyles.HexNumber);
 } else if (s[0] == '-') {
  return long.Parse(s);
 } else {
  return (long)ulong.Parse(s);
 }
}
```

We'll leave it here for now. Next time, we'll start putting some blocks together.

Raymond Chen

Follow