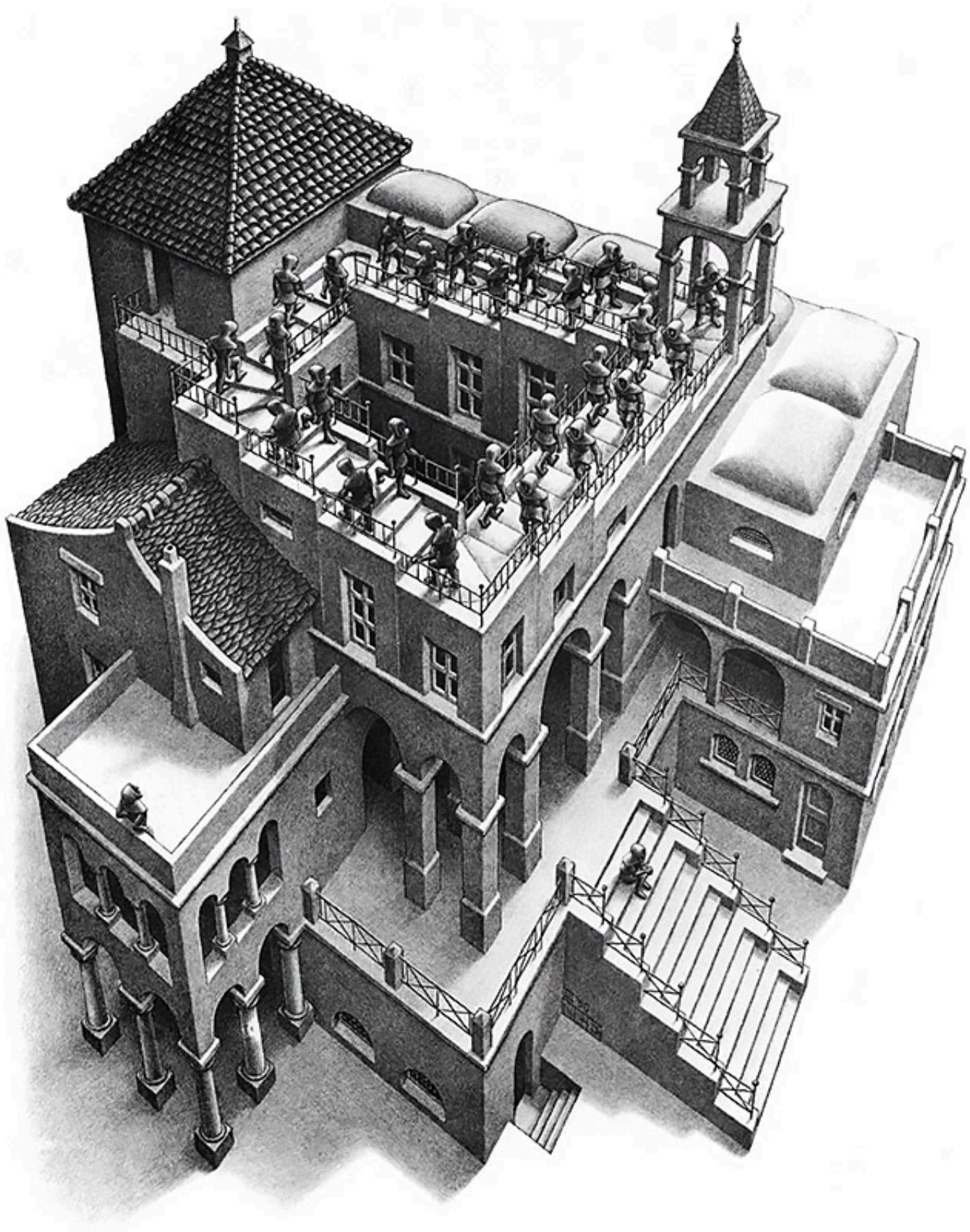
Article
Timur Safin · Apr 1, 2019 3m read

Closures in ObjectScript



After many sleepless nights it's a pleasure to announce the newer, better, moderner ObjectScript compiler which implemented pretty much everything you ever wanted to have in modern ObjectScript:

- Design objective of this new compiler is to parse reasonable subset of current ObjectScript syntax which will look readable for stranger, and not scare them with 1 letter syntax. The good start for compiler was the old-good COS Guidelines from here https://github.com/intersystems-ru/cos-guidelines

- For reasons we mentioned above we do not parse 1 letter syntax. It's declared evil;
- We do not handle dotted syntax for the same reason - modern syntax with {} is proper replacement for dotted syntax blocks;

But we not only parse the modern ObjectScript syntax, we have implemented finally the long-standing request which we always dreamed about. Closures!

Closures in ObjectScript

Lambdas, functors and closures are everything today. Ask every MIT graduate what they are thinking about functional-programming and Haskell - you will soon learn that it's hot. Every modern (or not so modern, but decent enough) language has their own way to implement closures: JavaScript/TypeScript, Haskell, even Perl or C++. We want to be modern and hot, that's why we implement closures in our ObjectScript parser.

```
==test.mac==
 set closure = &function { return "hello world!" }

 write $$&closure(),!

 set makeCounter = &function(init) {return &function() { $Increment(init) } }

 set cnt1 = $$&makeCounter(1)

 set cnt2 = $$&makeCounter(3)


 write makeCounter,!

 write $$&cnt1(),!

 write $$&cnt2(),!

 write $$&cnt1(),!


=============


USER>do ^test

hello world!

1@%Library.Closure

2

4

3
```

i.e. &function(args) { statement; } creates closure with arguments which will be used inside of braces. This is actually creating anonymous function object, which in turn could return closure (another anonymous function object), and if this inner function object accesses local variables passed then they will be captured for later invocations. Yes, this intentionally resembles JavaScript.

$$&id(args) calls closure commands via value stored to id local variable;

```
===resultset.mac==

 set rs = ##class(%ResultSet).%New()

 do rs.Prepare("select ID,Age from Sample.Person where :fn(Age) = 1")

  set adult = &function() { return age > 21 }

 do rs.%Execute(.adult)

 while rs.Next() {

   write rs.GetData(1), " ", rs.GetData(2)," ", rs.GetData(3),!

 }

=====

USER>do ^resultset

1 Gallant,Yan N. 62

2 Waal,Umberto G. 54

3 Jenkins,Sam A. 75

4 Marks,Milhouse B. 35

6 Peterson,Zeke U. 63

....
```

This syntax is preliminary and is subject to modifications, but hope you already got an idea, and want to begin to use it asap. The future with this new modern language implementing closures is bright, and it (hopefully) it will come soon!

The next steps we foresee:

- We are working hard on transpiler for version 1.1 to make it eventually possible to use operators with reasonable associativity. Yes, we want to make operators to have priorities we learnt at the school. It will break some backward compatibility, and that's why converter is being prepared;
- In version 1.2 we will fix ugly ##class() syntax because it hurts our eyes, the newer syntax is yet to be announced, but please not worry much - our converter will take care of that also;
- Stay tuned for more announcements - https://github.com/tsafin/ShinyClosures/blob/master/README.md

Update #1:

We have received some initial feedback, and decided to change syntax to be more JS-ish, than rather C++-ish. Because it's looking more compatible with current ObjectScript design look.

#ObjectScript #Caché