Project 1: Inline Caching

Goal

Add monomorphic inline caching to SDyn.

Time

This project is due by 12PM (**noon**, not midnight), Friday, February 8th, 2019.

Requirements

- Your implementation must be based on SDyn 1.2 or newer.
- You must not break any existing functionality of SDyn. In particular, field accesses to objects of shapes which have not been cached should work.
- Your code should compile and run on a standard 64-bit x86_64 Linux environment. It will be tested on `linux.student.cs.uwaterloo.ca`.
- Accesses—both reads and writes—to fields of an SDyn object must use inline caching. That is, after some number of executions of any such function, if objects are provided with the same shape as used in previous executions, their fields should be accessed in constant time, without performing any hash-table lookup.

Options

- It is presumed that you will use the existing inline cache scaffolding provided on the web site, but not required.
- If you use the patch: For each cacheable access site, the JIT creates an SDyn_InlineCache object. An SDyn_InlineCache* is passed to each function which accesses members of objects. You should define SDyn_InlineCache in h/sdyn/value.h , and use it in the relevant functions in value.c to implement caching of object shapes.
- You must implement at least a monomorphic inline cache, but are free to implement a polymorphic inline cache if you prefer.
- Any caching strategy is acceptable: You may update the cache every time a new shape is seen, record commonly-seen shapes and choose the most common, cache only the first shape seen, etc.
- It is not necessary to modify the JIT per se to implement inline caching: The provided template allows for an implementation of inline caching purely in C.

Marks

Your submission will be graded partially on "black-box" tests, which test its correctness with no examination of the code, and partially on "white-box" tests, which involve direct inspection:

- 33.34%: All tests included in the SDyn template, plus private grading tests, run correctly, give correct output, and correctly use inline caching. If the submitted SDyn does not compile, none of these points are available. Although no performance requirements are set in this project, they are graded by a human (me), so execution times which test human patience will be given no points.
- 66.66%: Inline caching is implemented correctly. This is purely by human inspection, so these points are technically available even if your code doesn't compile.

Notes

- SDyn is released under the ISC license. Your changes are your own.
- If your code exposes any existing bugs in SDyn (quite likely, frankly), that's fine. Please tell me so, so I know to expect it and can fix it.
- You may implement inline caching in any way you wish. That being said, while I will certainly smile upon an implementation that actually modifies the JIT to do inline caching at a low level—this is the "right" way to do it—there are no bonus points for such an implementation.
- Future projects will reuse the code created for this project.
- There are very few software components more difficult to debug than just-in-time compilers. `gdb` will catch bugs long after the root cause. Print statements are crucial but not sufficient. Accesses to GC-freed objects don't raise segmentation faults. If you stick to the runtime code, and not the JIT per se, `gdb` should mostly work as expected, but don't expect an easy time of debugging.
- Do not be afraid of rewriting. Rewriting a portion of this project, or even the entire project, may take considerably fewer hours than tracking down an obscure bug.
- A correct implementation of inline caching can be done in about twenty lines of code. The trickiness is not in the size of the code, but in getting the *right* twenty lines of code. If you've written thousands of lines and it's not working, you've probably overengineered the problem.

Submission

Project code must be submitted by email to the instructor. I have many email addresses, all of which go to the same inbox, but they get tagged differently; if you send your submissions to `cs842-2019@gregor.im`, I will appreciate it. As email isn't always the most reliable mechanism, I will respond to verify that I have received each submission. If you don't see a response from me, it's possible that something has gone wrong, so please tell me.

Late submissions are not accepted without prior agreement from the instructor. Extensions will rarely be granted. Requests for extensions sent within the 24 hours prior to the due date, or sent after the due date, will be ignored, except where existing university regulations on extensions, e.g. due to illness, are relevant.

Exceptions and help

For reasons of fairness, uniform projects are required. However, if you need to deviate from this document for exceptional reasons, or if you want any clarification or help, feel free to email the instructor.