1 Object Layout

Consider the following class (in primes.cool)

class Sieve(var prime : Int) extends IO() {
    var next : Sieve = null;
    ...
}

1. How much space does each Cool Int object take (including the eye-catcher)?

2. How much space does a Sieve object take? Include its eye-catcher, all its attributes and also including all the words making up the integer object it refers to, but not including the words for the next Sieve object.

3. There are also the prototype objects and dispatch tables. How many of each of these are there in a running Cool program? If we have 1000 integer objects, how much space is taken up by integer dispatch tables? By memory words that have integer dispatch table pointers in them?

4. On more sophisticated systems, no garbage collector (eye-catcher) tag is needed and the class tag and object size can be put into the (shared) dispatch table. Furthermore, an integer can be stored “unboxed”, that is, instead of a pointer to an integer object, we simply store the bare integer. How much space would a Sieve object take if this method was used?

5. If you had 64K to use for Sieve objects, how many could you store (1) assuming the existing Cool layout (2) assuming the optimized layout?

6. Compile primes.cl using the -g option. What happens different when you run it? Explain!

2 Literals

Copy over primes.cool into a temporary directory and compile it there. Then examine the generated assembly file, looking for literals. The boolean literals and the unit literal are not particularly interesting, but you will notice eleven integer literals (int_lit0 through int_lit10) and twenty string literals (string_lit0 through string_lit19).

For each one of these literals, indicate what the value is (the value of int_lit5 is not five!) and list everywhere this literal is used. Refer to “real” labels, not temporary labels. Thus, don’t say that a literal is used on the sixth instruction after L24. Instead say (for instance) that it is used inside Sieve.test. A few of the literals (such as string_lit2) are used everywhere a specific situation occurs. You don’t need to list every occurrence but do describe the specific situation.

3 Testing

Using the test6 script to evaluate it, write a fresh test case example.cool that would be a good test case for PA6. Do not start with an existing test file; write one from scratch. Make sure that the compiler detects no error while compiling, but the code need not run.
CS 754: Homework # 5  
**Due: 2020/4/15**

Instead of the above assignment, investigate space overhead in objects in Java implementations. Explain why there is overhead for the garbage collector, for the hash code and for synchronization. Anything else that leads to overhead?

Find (and cite) academic papers that propose ways to reduce the space overhead. One work is that on “thin locks.” Try to find experience papers: whether these ideas made it into Sun’s actual implementation. What is the meta-overhead of these schemes? That is, if one uses thin locks and has to convert to “fat locks,” how much worse is that than having “fat locks” all the time?

Please also discuss “bidirectional layout.” This is used for multiple inheritance but also in some Java virtual machines. Why? (Again, cite papers.)