CPSC 326 Final Project:

Mark and Sweep Garbage Collector for MyPL

Spring 2024

Colin McClelland

Mark and Sweep Algorithm

- Definition: Mark and Sweep is a garbage collection algorithm used to reclaim memory in programming languages. It consists of two main phases: **marking** and **sweeping**
- Mark Phase:
  - Begins with a set of root objects (global variables, local variables, etc)
  - Traverses all reachable objects from the “roots” and marks them as "reachable“
- Sweep Phase:
  - Scans through the entire memory space.
  - Reclaims memory occupied by unmarked (unreachable) objects.
  - Unmarked objects are considered “garbage” and are eligible for deallocation.
- Advantage: Straightforward and easier than other algorithms
- Disadvantage: Program execution has to wait on garbage collection

Adapted to MyPL

- Nearly the same as the previous description with some slight differences (no global variables)
- Minimally Invasive to base MyPL (iust the VM changed)
- Maintains an object graph
  - A dictionary of HeapObjects

```
class HeapObject:
    def __init__(self, oid):
        self.oid = oid
        self.parents = []
        self.references = []
```

- Also maintains a live root set over the course of the program
  - Updated accordingly as references to structs/arrays go out of scope
- Collector is invoked when a function returns
  - Traverse the roots to mark all of the directly and indirectly referenced objects
  - Iterate heaps, it object isn't marked, it is deleted

Unit Tests

- Approach: Examine the struct & array heap just before main returns
- Many example cases covering the simpler cases to the more advanced ones
  - Easy: Deallocating an int array that goes out of scope
  - Difficult: Deallocating an array of structs with the exception of the index that gets returned

```
def test_return_array_in_while_loop(capsys):
    program = (
        'void main() {\n'
        '    array int xs = new int[5];  // 2024\n'
        '    array int zs = my_fun();    // 2028\n'
        '    array int ys = new int[5];  // 2029\n'
        '}\n'
        '\n'
        'array int my_fun() {\n'
        '    int i = 0;\n'
        '    while (i < 5) {\n'
        '        array int xs = new int[2];  //2025, 2026, 2027, 2028\n'
        '        if (i == 3) {\n'
        '            return xs;  // 2028\n'
        '        }\n'
        '        i = i + 1;\n'
        '    }\n'
        '}\n'
    )
    build(program).run()
    captured = capsys.readouterr()
    print(captured.out)
    assert captured.out == 'struct: [] , array: [2024, 2028, 2029]\n'
```

Future Work & Takeaways

- Figure out the optimal time to run the garbage collector
  - Heap reaches certain size, performance slowdowns, etc
- Run the garbage collector concurrently as program continues to execute
- A good intro/overview of one of the more popular garbage collections algorithms
- A better understanding of memory management
- A nice sendoff for MyPL