1 Lab 5: Mark-Sweep Garbage Collector

Week 6 Marks: 10

1.1 Problem Statement

Implement a stop-the-world mark-sweep garbage collector integrated into the VM from Lab 4, managing dynamically allocated objects based on reachability.

1.2 Functional Requirements

1. Heap allocator
2. Root discovery
3. Mark phase
4. Sweep phase

1.3 Non-Functional Requirements

Memory safety, correctness under stress.

1.4 Optional Extensions

Generational GC, tri-color marking.

1.5 Deliverables

GC-enabled VM, performance evaluation, report.

1.6 Garbage Collector Test Cases

We give a suite of test cases used to validate the correctness of the stop-the-world mark–sweep garbage collector.

Conventions

- `new_pair(a, b)` allocates a heap object containing two references
- `push(vm, VAL_OBJ(o))` pushes an object reference onto the VM stack
- `pop(vm)` removes the top stack element

- `gc(vm)` explicitly triggers garbage collection

The VM stack is assumed to be part of the root set.

1.6.1 Basic Reachability

Purpose: Verify that objects directly reachable from the stack are preserved.

```
Obj* a = new_pair(NULL, NULL);
push(vm, VAL_OBJ(a));
gc(vm);
```

Expected Outcome:

- Object `a` survives
- Heap remains unchanged

1.6.2 Unreachable Object Collection

Purpose: Verify that unreachable objects are reclaimed.

```
Obj* a = new_pair(NULL, NULL);
gc(vm);
```

Expected Outcome:

- Object `a` is freed
- Heap is empty

1.6.3 Transitive Reachability

Purpose: Verify recursive marking of referenced objects.

```
Obj* a = new_pair(NULL, NULL);
Obj* b = new_pair(a, NULL);
push(vm, VAL_OBJ(b));
gc(vm);
```

Expected Outcome:

- Both `a` and `b` survive

1.6.4 Cyclic References

Purpose: Ensure that cycles are correctly handled.

```
Obj* a = new_pair(NULL, NULL);
Obj* b = new_pair(a, NULL);
a->right = b;

push(vm, VAL_OBJ(a));
gc(vm);
```

Expected Outcome:

- Both objects survive

1.6.5 Deep Object Graph

Purpose: Stress-test recursive marking.

```
Obj* root = new_pair(NULL, NULL);
Obj* cur = root;

for (int i = 0; i < 10000; i++) {
        Obj* next = new_pair(NULL, NULL);
        cur->right = next;
        cur = next;
}

push(vm, VAL_OBJ(root));
gc(vm);
```

Expected Outcome:

- All objects survive
- No stack overflow occurs

1.6.6 Closure Capture

Purpose: Ensure closures extend lifetimes of captured objects.

```
Obj* env = new_pair(NULL, NULL);
```

```
Obj* fn = new_function();
Obj* cl = new_closure(fn, env);

push(vm, VAL_OBJ(cl));
gc(vm);
```

Expected Outcome:

- Closure, function, and environment survive

1.6.7 Stress Allocation

Purpose: Validate heap integrity under heavy allocation.

```
for (int i = 0; i < 100000; i++) {
        new_pair(NULL, NULL);
}

gc(vm);
```

Expected Outcome:

- Heap is empty after GC
- No memory leaks or crashes