# PRACTICAL 2

## Treadmill Garbage Collector

---

*Matric. No.*
*140021043*
*26 November 2014*

*Introduction*

As instructed by the practical specifications, this is a treadmill garbage collector modelled on Baker's implementation. The following are some notes about this particular GC implementation.

The "memory" is represented by a doubly-linked list. Each node in the list has an address, type, two values, color, next, and prev. The next and prev are of course pointers to the next and previous nodes in the linked list. The rest of the node attributes relate to the heap objects. In a real garbage collector, the heap object and its attributes would not be a part of the memory itself, and I could have moved all of them to a separate data structure. However, this would have just added a level of indirection to the node structure, and does not have any real benefits in this implementation. As such, all information about the heap objects is stored in the node struct.

Upon initialization, an empty linked list is created with a specified number of nodes – throughout writing and testing, I primarily used a linked list of 15 nodes. Heap objects, specified in main, are then added to the linked list. A word about the heap objects – all of the functionality laid out in the CF object representation is not fully implemented. Here is what is implemented:

- `INT i`
- `BOOL b`
- `IND p`
- `WEAK p`
- `SOFT p`

The following CF objects are either not implemented, or implemented with some restrictions:

- `FLOAT`: it's implemented, in a sense. The val1 parameter of a `FLOAT` node corresponds to the digits of the float to the left of the decimal, and the val2 parameter corresponds to the digits to the right. For example, the float 2.34 would have node->val1 = 2 and node->val2 = 34.
- `TUP`: it works the same as `CONS` in previous examples, and is limited to two pointers. For example, `TUP 3 6` points to addresses 3 and 6.
- `VECTOR`: same as `TUP`
- `CALL`: can only have one argument. For example, `CALL 3 6` points to object at 3 representing a function, and object at 6 representing the function argument.

Despite the limitations of some of the CF objects, you will find that the program runs as a proper treadmill garbage collector. Heap objects begin in the list as white, unchecked objects. The roots are then moved to grey, checked but not scanned. The roots are then scanned, in which they are moved to black and parsed. Any objects pointed to by the roots are moved to grey. Once all the roots have been scanned, all grey objects are scanned until there are no more grey objects (only black and white). Once this point has been reached, the marking is done. All white objects are released, and all black objects are switched to white. New heap objects are added, and the process begins again.

*Test Cases*

The following test cases where done with a doubly-linked list of 15 nodes. Ecru is free, white is unchecked, grey is visited but not scanned, black is live.

*Test 1 – Single GC collection (no mutation)*

Initial Heap (roots 21,0,3,12)

```
0     | SOFT     | 21 | 0   | white
3     | VAR      | 4  | 0   | white
6     | IND      | 0  | 0   | white
9     | TUP      | 18 | 21  | white
12    | TUP      | 6  | 24  | white
15    | CALL     | 9  | 3   | white
18    | SOFT     | 0  | 0   | white
21    | TUP      | 9  | 0   | white
24    | WEAK     | 27 | 0   | white
27    | NULL     | 0  | 0   | white
30    | IND      | 15 | 0   | white
33    | (null)   | 0  | 0   | ecru
36    | (null)   | 0  | 0   | ecru
39    | (null)   | 0  | 0   | ecru
42    | (null)   | 0  | 0   | ecru
```

Before collecting trash:

```
0     | SOFT     | 21 | 0   | black
3     | VAR      | 4  | 0   | black
6     | IND      | 0  | 0   | black
12    | TUP      | 6  | 24  | black
18    | SOFT     | 0  | 0   | black
9     | TUP      | 18 | 21  | black
21    | TUP      | 9  | 0   | black
15    | CALL     | 9  | 3   | white
24    | WEAK     | 27 | 0   | white
27    | NULL     | 0  | 0   | white
```

```
30    | IND      | 15  | 0   | white
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

After Collection trash:

```
Removing 15 from heap
Removing 24 from heap
Removing 27 from heap
Removing 30 from heap
0     | SOFT     | 21  | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
12    | TUP      | 6   | 9   | white
18    | SOFT     | 0   | 0   | white
9     | TUP      | 18  | 21  | white
21    | TUP      | 9   | 0   | white
15    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

*Test Case 2 - Two Rounds of GC (one mutation)*

Initial Heap (roots 21,0,3,12)

```
0     | SOFT     | 21  | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 18  | 21  | white
12    | TUP      | 6   | 24  | white
15    | CALL     | 9   | 3   | white
18    | SOFT     | 0   | 0   | white
21    | TUP      | 9   | 0   | white
24    | WEAK     | 27  | 0   | white
27    | NULL     | 0   | 0   | white
30    | IND      | 15  | 0   | white
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

Objects added in mutation (roots 6,18)

```
15    | INT      | 4   | 0   | white
24    | WEAK     | 3   | 0   | white
27    | TUP      | 6   | 24  | white
```

After mutation and trash collecting:

```
Removing 3 from heap
Removing 12 from heap
Removing 15 from heap
Removing 24 from heap
Removing 27 from heap
0     | SOFT     | 21  | 0   | white
6     | IND      | 0   | 0   | white
3     | (null)   | 0   | 0   | ecru
12    | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
18    | SOFT     | 0   | 0   | white
9     | TUP      | 18  | 21  | white
21    | TUP      | 9   | 0   | white
```

*Test Case 3 - Four Rounds of GC (three mutations), while running out of memory space on the fourth mutation*

Initial Heap (roots 21,0,3,12)

```
0     | SOFT     | 21  | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 18  | 21  | white
12    | TUP      | 6   | 24  | white
15    | CALL     | 9   | 3   | white
18    | SOFT     | 0   | 0   | white
21    | TUP      | 9   | 0   | white
24    | WEAK     | 27  | 0   | white
27    | NULL     | 0   | 0   | white
30    | IND      | 15  | 0   | white
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

Objects added in first mutation (roots 6,18)

```
15    | INT      | 4   | 0   | white
24    | WEAK     | 3   | 0   | white
27    | TUP      | 6   | 24  | white
```

After first mutation and trash collecting:

```
Removing 3 from heap
Removing 12 from heap
Removing 15 from heap
Removing 24 from heap
Removing 27 from heap
0     | SOFT     | 21  | 0   | white
6     | IND      | 0   | 0   | white
3     | (null)   | 0   | 0   | ecru
12    | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
18    | SOFT     | 0   | 0   | white
9     | TUP      | 18  | 21  | white
21    | TUP      | 9   | 0   | white
```

Objects added in second mutation (roots 15)

```
3     | IND      | 24  | 0   | white
12    | WEAK     | 18  | 0   | white
15    | TUP      | 3   | 12  | white
```

After second mutation and trash collecting:

```
Removing 6 from heap
Removing 15 from heap
0     | SOFT     | 21  | 0   | white
18    | SOFT     | 0   | 0   | white
12    | WEAK     | 18  | 0   | white
3     | TUP      | 9   | 12  | white
6     | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

```
9     | TUP      | 18  | 21  | white
21    | TUP      | 9   | 0   | white
```

Objects added in third mutation (roots 27)

```
3     | TUP      | 9   | 12  | white
6     | BOOL     | 1   | 0   | white
15    | IND      | 18  | 0   | white
24    | TUP      | 3   | 12  | white
27    | IND      | 0   | 0   | white
30    | IND      | 0   | 0   | white
33    | TUP      | 12  | 12  | white
36    | NULL     | 0   | 0   | white
39    | CALL     | 3   | 15  | white
42    | INT      | 13  | 0   | white
45    | CALL     | 36  | 12  | white
Not enough space! Only 9 spaces free
```

As a result, the last object (address 45) is left out, since there is no space for it in memory. Another approach could have been to expand the memory size as needed.

After third mutation and trash collecting:

```
Removing 12 from heap
Removing 3 from heap
Removing 6 from heap
Removing 15 from heap
Removing 24 from heap
Removing 30 from heap
Removing 33 from heap
Removing 36 from heap
Removing 39 from heap
Removing 42 from heap
0     | SOFT     | 21  | 0   | white
27    | IND      | 0   | 0   | white
12    | (null)   | 0   | 0   | ecru
3     | (null)   | 0   | 0   | ecru
6     | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
18    | SOFT     | 0   | 0   | white
9     | TUP      | 18  | 21  | white
21    | TUP      | 9   | 0   | white
```

*Timing Tests & Analysis*

Test 1 - 1,000,000 iterations of the following heap (root 6)

```
0     | SOFT     | 3   | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 18  | 21  | white
12    | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
18    | (null)   | 0   | 0   | ecru
21    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

**Average time: 1 second**

Test 2 - How does more roots affect runtime? (roots 6,3)

```
0     | SOFT     | 3   | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 3   | 0   | white
12    | (null)   | 0   | 0   | ecru
15    | (null)   | 0   | 0   | ecru
18    | (null)   | 0   | 0   | ecru
21    | (null)   | 0   | 0   | ecru
24    | (null)   | 0   | 0   | ecru
27    | (null)   | 0   | 0   | ecru
30    | (null)   | 0   | 0   | ecru
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

**Average time: 1**

Test 3 - How does number of objects affect runtime?

```
0     | SOFT     | 21  | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 18  | 21  | white
12    | TUP      | 6   | 24  | white
15    | CALL     | 9   | 3   | white
18    | SOFT     | 0   | 0   | white
21    | TUP      | 9   | 0   | white
24    | WEAK     | 27  | 0   | white
27    | NULL     | 0   | 0   | white
30    | IND      | 15  | 0   | white
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
```

**Average time: 1.4 seconds**

Test 4 - How does size of linked list affect runtime?

```
0     | SOFT     | 21  | 0   | white
3     | VAR      | 4   | 0   | white
6     | IND      | 0   | 0   | white
9     | TUP      | 18  | 21  | white
12    | TUP      | 6   | 24  | white
15    | CALL     | 9   | 3   | white
18    | SOFT     | 0   | 0   | white
21    | TUP      | 9   | 0   | white
24    | WEAK     | 27  | 0   | white
27    | NULL     | 0   | 0   | white
30    | IND      | 15  | 0   | white
33    | (null)   | 0   | 0   | ecru
36    | (null)   | 0   | 0   | ecru
39    | (null)   | 0   | 0   | ecru
42    | (null)   | 0   | 0   | ecru
…           …           …     …
(n-1)*3    | (null)   | 0   | 0   | ecru
```

**Average time for:**

10 nodes: 1.7 seconds

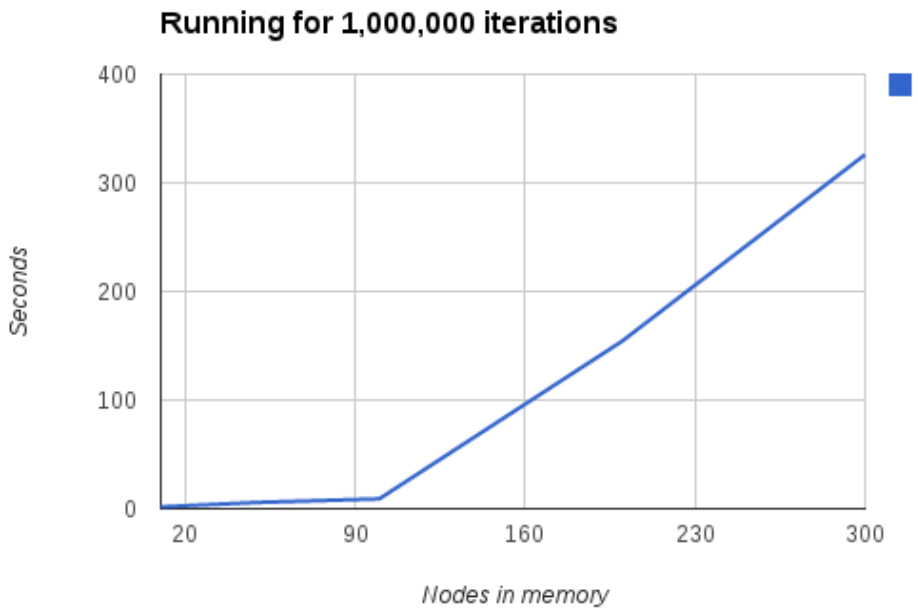
50 nodes: 5.8 seconds

75 nodes: 8 seconds

100 nodes: 9.1 seconds

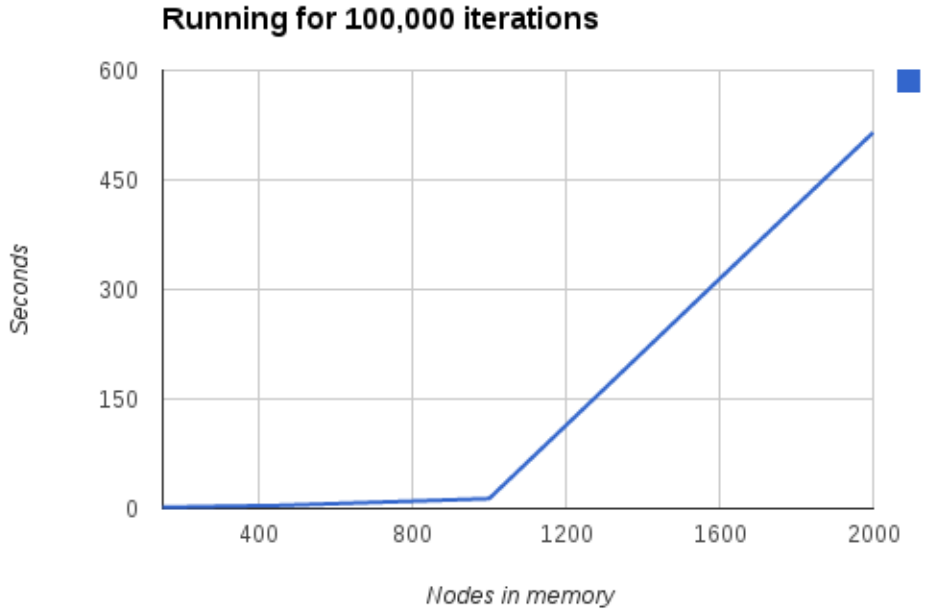
200 nodes: 154.33 seconds

500 nodes: computer crashed




**Average time for 100,000 iterations**

150 nodes: 1.7 seconds
300 nodes: 3.1 seconds
450 nodes: 4.2 seconds
1000 nodes: 13.6 seconds
2000 nodes: 515 seconds
5000 nodes: timed out



As you can see, the number of nodes in the memory linked list has an exponential effect on the runtime of the garbage collection. This is mainly due to inefficient methods in my program. Many programs, such as find_node(), iterate through the list to find the node it's looking for, and find_node() is called very frequently. Methods such as take_out_trash() and reset_heap() also contain for loops that iterate through the entire HEAP_SIZE variable (which I now realize is misnamed and should be called MEMORY_SIZE), which can be thousands of nodes for some of the higher memory allocations.