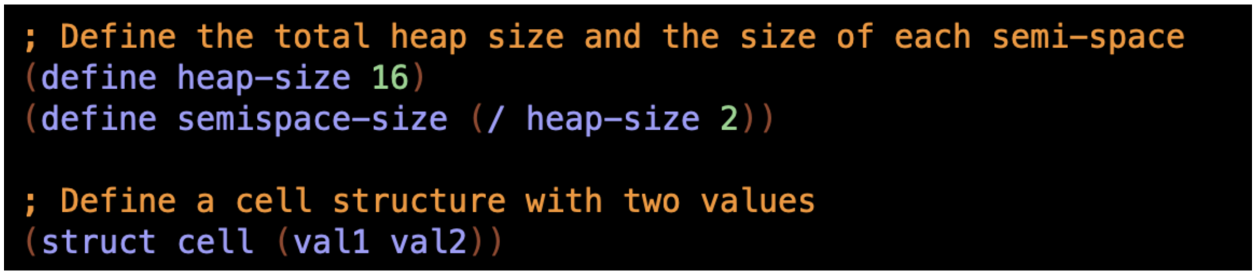
Stop-and-copy Garbage Collection

Lasha Kaliashvili and Mikheil Uglava

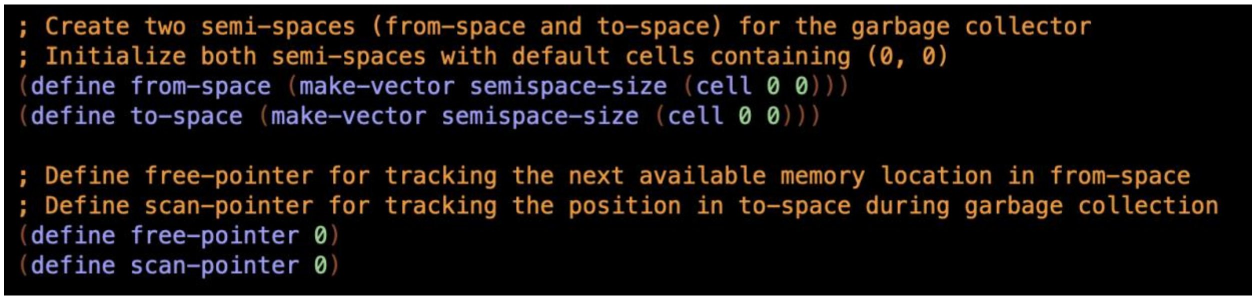
The program starts with defining two variables: heap-size and semispace-size. Heap-size stands for the total heap size, which is being used as the main memory space in this simulation, and semispace-size is equal to the heap size divided by 2 – since the whole space is divided into two equal parts (from-space and to-space).

The program also defines a cell structure with two values. The cell structure represents the basic unit of memory allocation, and for the purpose of this simulation, it just stores two values (in the case of an example later it stores two integers).



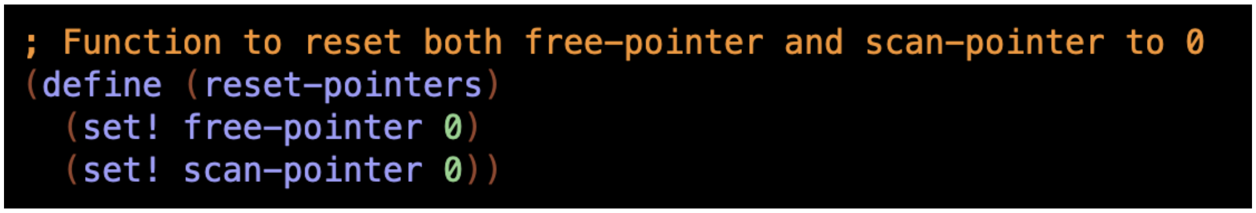
```
; Define the total heap size and the size of each semi-space
(define heap-size 16)
(define semispace-size (/ heap-size 2))

; Define a cell structure with two values
(struct cell (val1 val2))
```

The program initializes both semi-spaces, from-space and to-space, with default cells containing (0,0). Vectors are used to represent the spaces. We also define two pointers, “free-pointer” and “scan-pointer”, to track the next available memory location in from-space and the position in to-space during garbage collection, respectively.



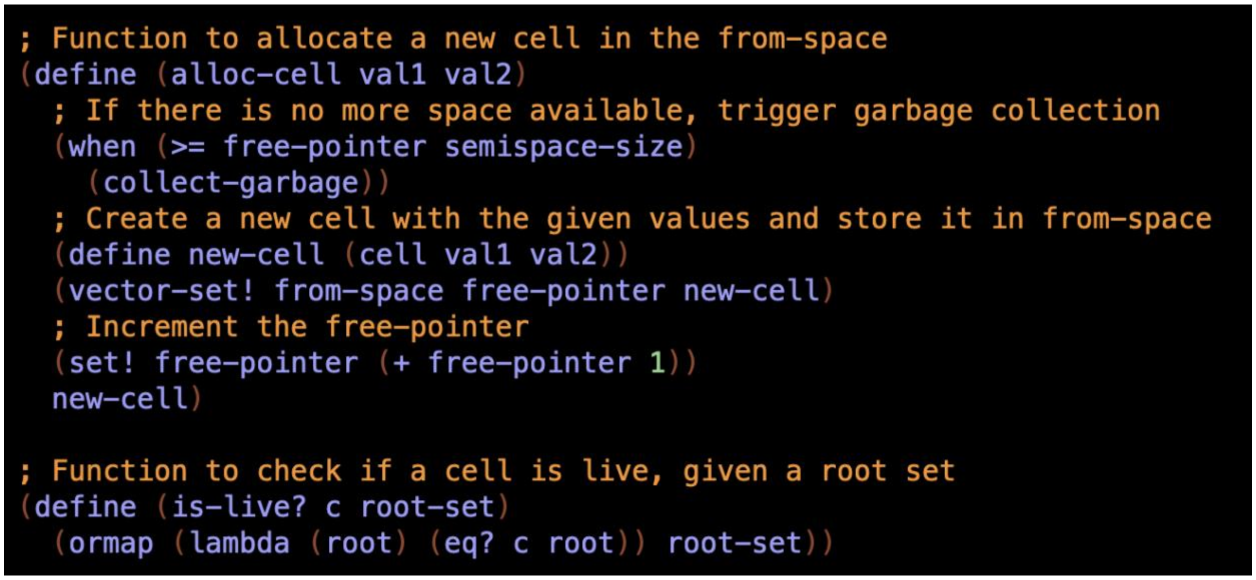
```
; Create two semi-spaces (from-space and to-space) for the garbage collector
; Initialize both semi-spaces with default cells containing (0, 0)
(define from-space (make-vector semispace-size (cell 0 0)))
(define to-space (make-vector semispace-size (cell 0 0)))

; Define free-pointer for tracking the next available memory location in from-space
; Define scan-pointer for tracking the position in to-space during garbage collection
(define free-pointer 0)
(define scan-pointer 0)
```

Next, The program defines a “reset-pointers” function that resets both “free-pointer” and “scan-pointer” to 0. These pointers basically count the number of used memory cells in each semi-space. This function is used in the collect-garbage function, as well as in the testing process to reset the memory.



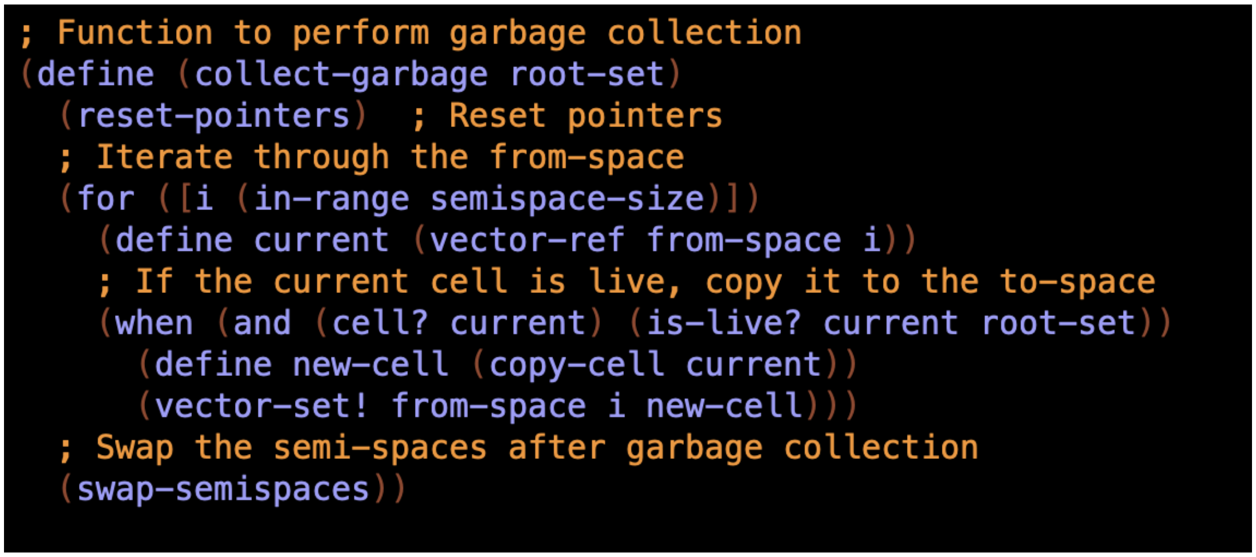
```
; Function to reset both free-pointer and scan-pointer to 0
(define (reset-pointers)
  (set! free-pointer 0)
  (set! scan-pointer 0))
```

The “Alloc-cell” function allocates a new cell in the from-space. If there is no more space available, the function triggers garbage collection. The function creates a new cell with the given values and stores it in from-space and then the “free-pointer” variable is incremented. The “Is-live?” function just checks if a cell is reachable. The operation of this function is very simple – the program just goes through every root cell in the list called root-set and compares it to the cell we are checking, if at least one match is made, the function would return true.



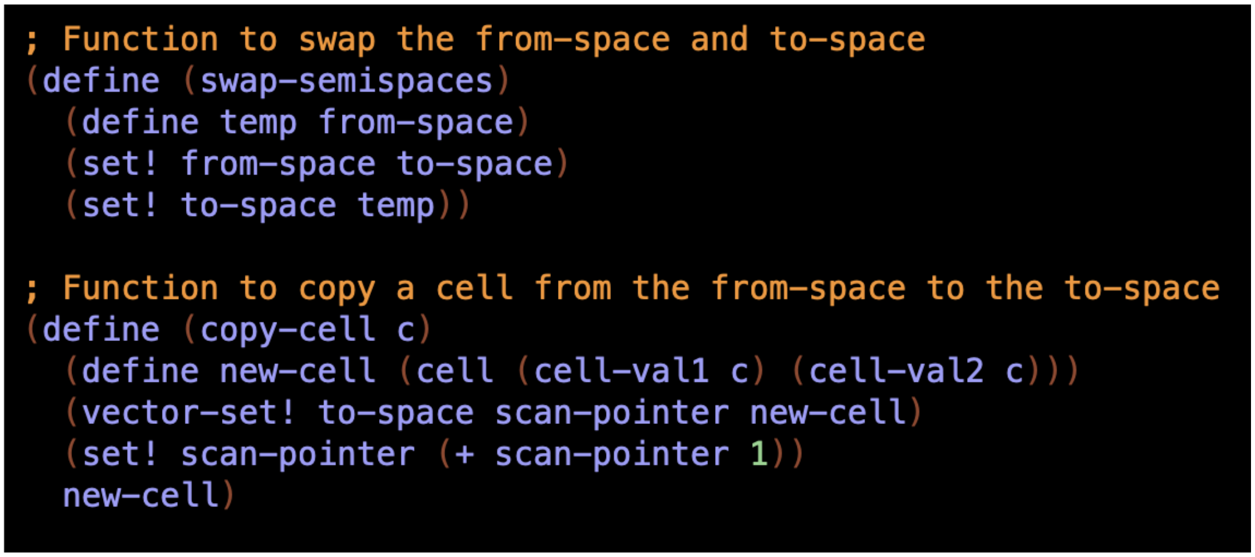
```
; Function to allocate a new cell in the from-space
(define (alloc-cell val1 val2)
  ; If there is no more space available, trigger garbage collection
  (when (>= free-pointer semispace-size)
    (collect-garbage))
  ; Create a new cell with the given values and store it in from-space
  (define new-cell (cell val1 val2))
  (vector-set! from-space free-pointer new-cell)
  ; Increment the free-pointer
  (set! free-pointer (+ free-pointer 1))
  new-cell)

; Function to check if a cell is live, given a root set
(define (is-live? c root-set)
  (ormap (lambda (root) (eq? c root)) root-set))
```

The program defines a “collect-garbage” function to perform garbage collection. The function takes a root set as an input and resets the pointers. After that, the function goes through each cell in the from-space and checks for live objects. If the cell is live, then it's copied to the to-space, and its place in from-space is replaced by a new, empty cell with values 0 and 0. After the garbage collection is complete, the function swaps the semi-spaces.



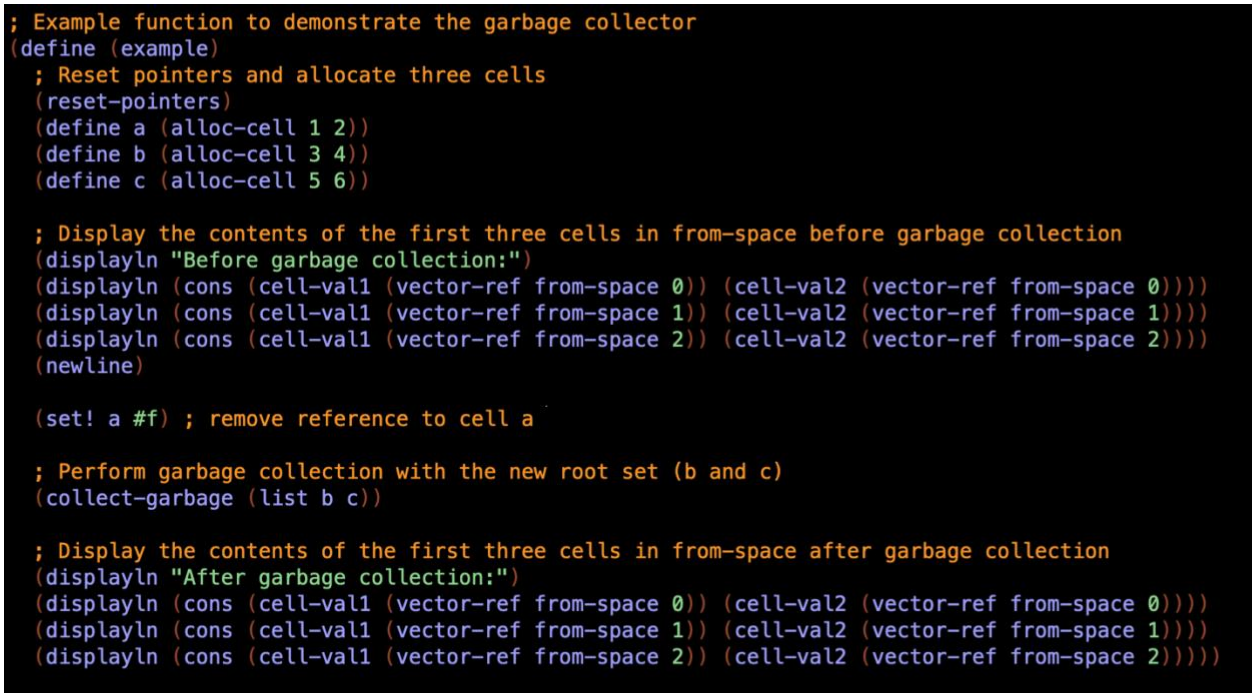
```
; Function to perform garbage collection
(define (collect-garbage root-set)
  (reset-pointers)  ; Reset pointers
  ; Iterate through the from-space
  (for ([i (in-range semispace-size)])
    (define current (vector-ref from-space i))
    ; If the current cell is live, copy it to the to-space
    (when (and (cell? current) (is-live? current root-set))
      (define new-cell (copy-cell current))
      (vector-set! from-space i new-cell)))
  ; Swap the semi-spaces after garbage collection
  (swap-semispaces))
```

The "swap-semispaces" function simply swaps the two semi-spaces and the "copy-cell" function copies a cell between spaces. The function creates a new cell with the same values as the input cell and stores it in the "to-space". Finally, the "scan-pointer" is incremented.

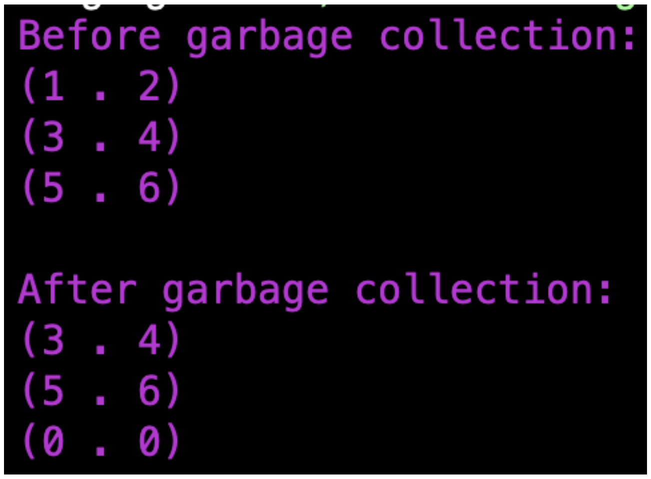


```
; Function to swap the from-space and to-space
(define (swap-semispaces)
  (define temp from-space)
  (set! from-space to-space)
  (set! to-space temp))

; Function to copy a cell from the from-space to the to-space
(define (copy-cell c)
  (define new-cell (cell (cell-val1 c) (cell-val2 c)))
  (vector-set! to-space scan-pointer new-cell)
  (set! scan-pointer (+ scan-pointer 1))
  new-cell)
```

For demonstration purposes, we wrote this simple function called example, which creates three different objects and allocates a memory cell for each of them. Then a reference is removed for one of the cells, which means that that particular cell is no longer reachable, which should lead to the object being collected by the collect-garbage function. After the collect-garbage function is called, we printed out the results for comparison.



```
; Example function to demonstrate the garbage collector
(define (example)
  ; Reset pointers and allocate three cells
  (reset-pointers)
  (define a (alloc-cell 1 2))
  (define b (alloc-cell 3 4))
  (define c (alloc-cell 5 6))

  ; Display the contents of the first three cells in from-space before garbage collection
  (displayln "Before garbage collection:")
  (displayln (cons (cell-val1 (vector-ref from-space 0)) (cell-val2 (vector-ref from-space 0))))
  (displayln (cons (cell-val1 (vector-ref from-space 1)) (cell-val2 (vector-ref from-space 1))))
  (displayln (cons (cell-val1 (vector-ref from-space 2)) (cell-val2 (vector-ref from-space 2))))
  (newline)

  (set! a #f) ; remove reference to cell a

  ; Perform garbage collection with the new root set (b and c)
  (collect-garbage (list b c))

  ; Display the contents of the first three cells in from-space after garbage collection
  (displayln "After garbage collection:")
  (displayln (cons (cell-val1 (vector-ref from-space 0)) (cell-val2 (vector-ref from-space 0))))
  (displayln (cons (cell-val1 (vector-ref from-space 1)) (cell-val2 (vector-ref from-space 1))))
  (displayln (cons (cell-val1 (vector-ref from-space 2)) (cell-val2 (vector-ref from-space 2)))))
```



```
Before garbage collection:
(1 . 2)
(3 . 4)
(5 . 6)

After garbage collection:
(3 . 4)
(5 . 6)
(0 . 0)
```

As you can see above, before running the collect-garbage function, we had three different objects. But after we removed the reference to the first object and then called the garbage collector, in the process the first object would not be copied to the to-space, only the second and the third objects would, and after that, they would both return to the from-space and take the first 2 available positions (the second object now holds cell 0 and the third object – cell 1).