Egg Eater Report

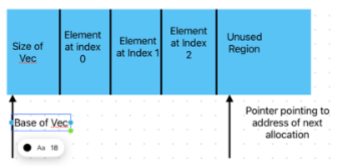
Concrete Grammar:

```
<expr> :=
 |<Everything in diamondback>
 | (vec <expr>*)                    (new!)
 | (vec-get <expr> <expr>)          (new!)
```

Heap Allocated Values

A vec is a list of expr values which are stored on the heap. In our design, each member of the vec occupies 8 bytes. To access a member of a vec, we simply need to load the value that is <index + 1> number of words away from the base address of the vec. We need to use <index+1> since the 0th offset of any vec holds its size. The vec-get operation takes a vec and an index and returns the value stored at that index in the specified vec. If the index is out of bounds of the vec size, an error is reported.

There are no limitations on the type of members of the vecs. All vec addresses are byte aligned.

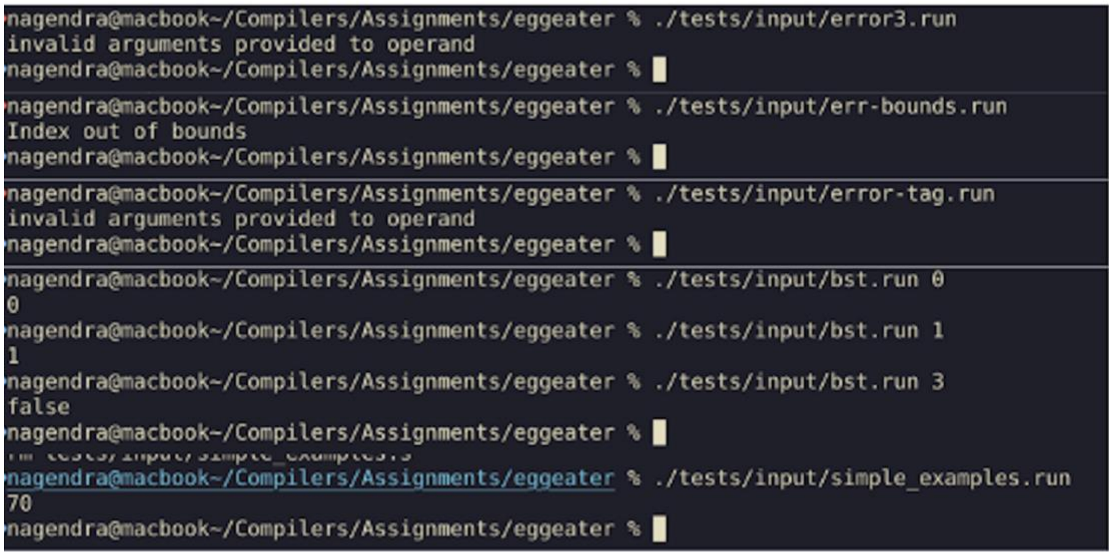


Sample Programs which make use of Vecs (Please compile programs from inside tests/input folder because of the way my Makefile is setup)

1. tests/input/simple_examples.snek:
   a. (let ((v1 (vec 10 20 30)) (v2 (vec 40 50 60))) (+ (vec-get v1 0) (vec-get v2 2))) – Creates 2 vecs v1 and v2. Adds the 0th element of v1 and 2nd element of v2 and returns the sum.
2. tests/input/error-tag.snek:
   a. (vec-get 4 0) - Tries to access 0th index of a non vec expr. Since numbers are stored with last bit compulsorily set to 0, when we check for the tag bit of a vector, 001, this fails, and we print an invalid type error message.
3. tests/input/error_bounds.snek:
   a. (let ((lst (vec 10 20 30))) (vec-get lst 5)) - Creates a 3 element vec 10, 20 and 30 and tries to access element at index 5. Since the size of the vector is stored at the base address, we compare against this size before making any access.
4. tests/input/error3.snek:
   a. (+ false 40) – Tries to add a number and a Boolean. Since Booleans have the tag 011 or 111 and numbers always end in 0, we check these types before adding and it leads to an invalid type error.
5. tests/input/points.snek
   Defines 2 functions, make_point and add_point. make_point takes the x and y values of a point and returns a vec with x as 0th member and y as 1st member. add_point takes two points p1 and p2, and creates a new vec with x1+ x2 and y1 + y2 as the two co-ordinates.
6. tests/input/bst.snek
   Defines 2 functions, input_bst and search_bst to input values into the bst and to search values in the bst.
   We create a binary search tree, where 0th element is the value of the node, 1st element is left subtree and 2nd elemet is right subtree. The null values are represented by false, and we recursively search the tree until value is found or false is encountered. To input a value into the tree, we take the node and check if it is false, if it is false, we return a new node with root

node equal to number and left and right children set to false. If the number is smaller than node, we return a tree with the same root node, but left child equal to that tree created on recursively calling input with left child as root and number as number to be inputted and same right child as before. If number is greater than root node we repeat the same operation as before with left and right children interchanged.

Outputs of the above Programs:



```
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/error3.run
invalid arguments provided to operand
nagendra@macbook~/Compilers/Assignments/eggeater % 
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/err-bounds.run
Index out of bounds
nagendra@macbook~/Compilers/Assignments/eggeater % 
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/error-tag.run
invalid arguments provided to operand
nagendra@macbook~/Compilers/Assignments/eggeater % 
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/bst.run 0
0
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/bst.run 1
1
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/bst.run 3
false
nagendra@macbook~/Compilers/Assignments/eggeater % 
nagendra@macbook~/Compilers/Assignments/eggeater % ./tests/input/simple_examples.run
70
nagendra@macbook~/Compilers/Assignments/eggeater % 
```

Heap Allocated Values in Other Languages

C: In C, malloc/calloc/realloc are used to allocate a chunk of memory on the heap. Once allocated, the user is also responsible for freeing the memory once he is done using it. To access members of the heap, we use indexing operator [] or member dereference operator -> in case of structs and unions. The language itself does not do any bounds checking to ensure that all accesses are within the allocated region for our heap. If an invalid access is made, the OS intervenes and raises a SEGFAULT error. [1]

Java: In Java, all objects are created on the heap, and memory is allocated on the heap when we use the new operator. The values are initialized to 0, false or null depending on the type of the object. If a constructor is invoked, the constructor initializes the memory. To access members of heap allocated values we use the indexing operator [] or the member access operator ‘ . ‘. The Java runtime does bounds checking and raises an out of bounds exception in the event of an out of bounds access. The allocated memory is freed by the garbage collector. [2]

My implementation is like Java because we already have a pre-allocated amount of memory provided to us on the runtime within which we store our values. Runtime bounds checking in done like that of Java. However, unlike Java, we do not have a garbage collector mechanism to dispose of values we no longer need. Also unlike Java we do not do compile time type checking.

Resources:

1. C Programming Language 2nd Edition – Kernighan, Ritchie
2. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/ArrayIndexOutOfBoundsException.html