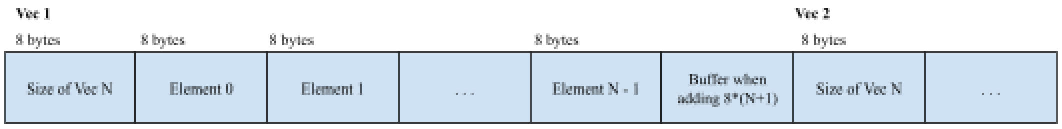
1. Concrete Grammar Additions

```
Vec(Vec<Expr>),              // Vec implementation of any size ("vec 2 3 true")
Get(Box<Expr>, Box<Expr>),   // Get to retrieve data from index ("vec-get (vec 1) 0")
Length(Box<Expr>),           // Length to retrieve size of the vec ("vec-len (vec 1 true 3 4)")
```

2. Diagram of heap

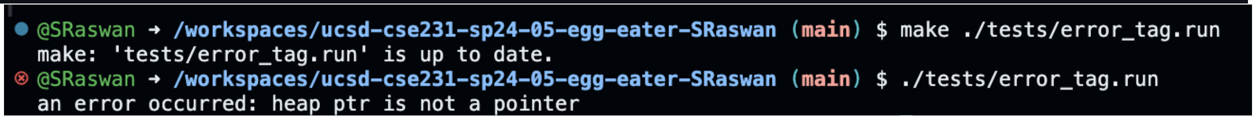


3. Tests

https://github.com/ucsd-cse231/ucsd-cse231-sp24-05-egg-eater-SRaswan/tree/main/input
`/input/` directory has required test files, which is also included in `/tests/` directory. `all_tests.rs` runs the `tests/bst.snek` and `tests/points.snek` multiple times with different inputs to test all aspects of them.

input/error-tag.snek

```
(vec-get false 3)
```

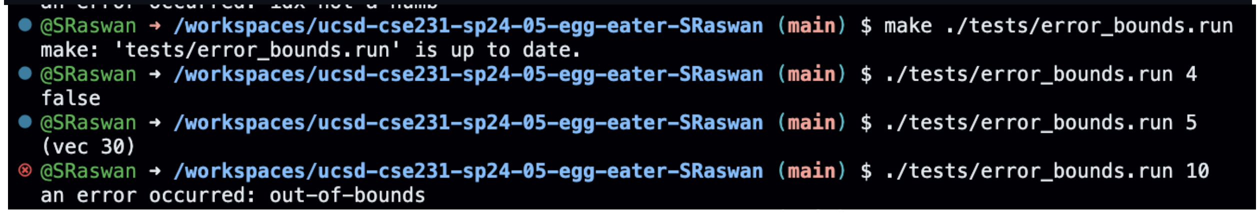


```
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ make ./tests/error_tag.run
make: 'tests/error_tag.run' is up to date.
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_tag.run
an error occurred: heap ptr is not a pointer
```

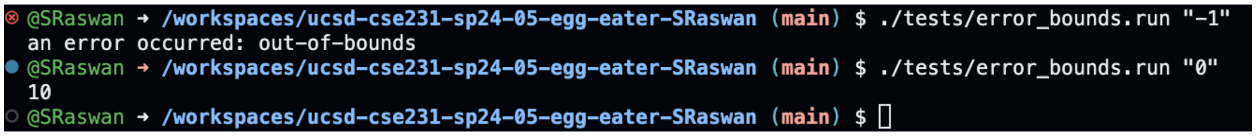
Runtime error: here the tag checking returns an error telling the user that the vector inputted in vec-get is not actually a vector because it is not a valid pointer (it is a boolean "false" which is not ending in 001 tag).

input/error-bounds.snek

```
(vec-get (vec 10 20 30 true false (vec 30) 10 20 3 12) input)
```



```
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ make ./tests/error_bounds.run
make: 'tests/error_bounds.run' is up to date.
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_bounds.run 4
false
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_bounds.run 5
(vec 30)
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_bounds.run 10
an error occurred: out-of-bounds
```

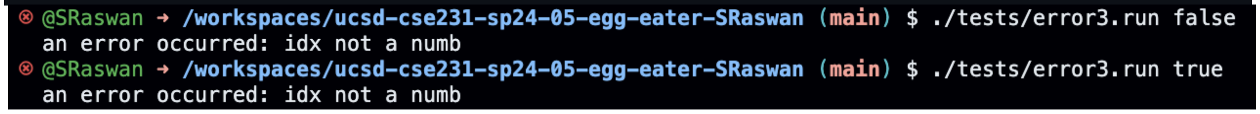


```
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_bounds.run "-1"
an error occurred: out-of-bounds
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error_bounds.run "0"
10
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ 
```

Runtime error: here the input is the index that determines which value is returned. If we vec-get outside of the bounds of the vector, it returns a runtime error saying that the index is out of bounds.

input/error3.snek

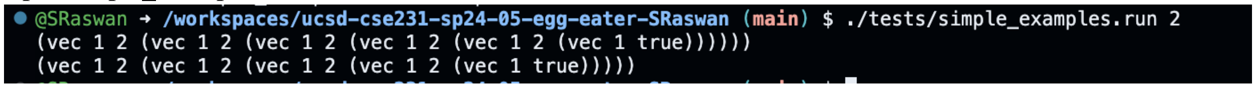
```
(vec-get (vec 3 12 3 false true) input)
```



```
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error3.run false
an error occurred: idx not a numb
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/error3.run true
an error occurred: idx not a numb
```

Runtime error: here the input determines the input like before. When we input a non-number value, it creates a runtime error as the expression does not reduce to a number and therefore does not represent an index. If we make the index as a vector (vec 3 for example), it will also create a runtime error.

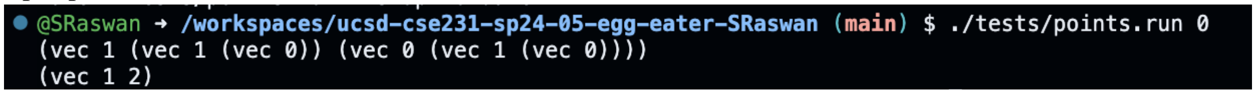
input/simple_examples.snek



```
@SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/simple_examples.run 2
(vec 1 2 (vec 1 2 (vec 1 2 (vec 1 2 (vec 1 2 (vec 1 true))))))
(vec 1 2 (vec 1 2 (vec 1 2 (vec 1 2 (vec 1 true)))))
```

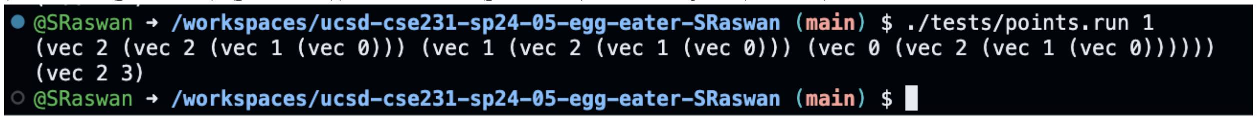
This example will recursivley wrap vectors around the lst variable which holds an inital vector. It then modifies lst to its latest value. I inputted the number 2 which made 5 recursively nested vectors around the initial value of lst. It also indexed and got the second value of the vector, which was a nested vector (the third value of the outside vector).

input/points.snek



```
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/points.run 0
  (vec 1 (vec 1 (vec 0)) (vec 0 (vec 1 (vec 0))))
  (vec 1 2)
```

It creates a structure of a 2 x 2 coordinate system, with the second value representing the y axis and the third value representing the next value of the x axis. The value below is when we have a point function that takes in parameters x and y. This outputs a (vec x y) which represents a point. We use the combine function to combine 2 points. (combine (point 0 2) (point 1 0)) results in a (point 1 2) which is just (vec 1 2).

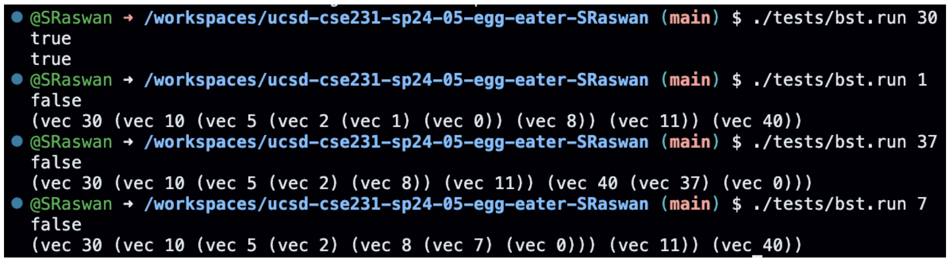


```
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/points.run 1
  (vec 2 (vec 2 (vec 1 (vec 0))) (vec 1 (vec 2 (vec 1 (vec 0))) (vec 0 (vec 2 (vec 1 (vec 0))))))
  (vec 2 3)
○ @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ 
```

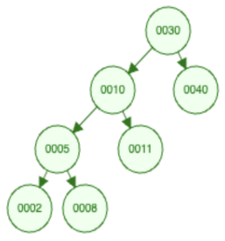
This is a 3 x 3 coordinate system structure, where we have a nested vector with a recursive (vec (N-1) (nested y axis) (nested x axis - 1)). (combine (point 1 2) (point 1 1)) results in a (point 2 3) which is just (vec 2 3).

input/bst.snek

This creates a bst (vec 30 (vec 10 (vec 5 (vec 2) (vec 8)) (vec 11)) (vec 40)) where the root is 30 and the leaves are 2, 8, 11, 40. The structure is (vec key (vec leftTree) (vec rightTree)). We can add to the bst with our input, and it only adds it if we do not find the value in our tree already. It returns true if the value is already in it. Note that in order to show whether it is a left or right leaf when only one of them are created, we use the number 0 as kind of the "null" value. This is a problem however as we cannot try to insert 0 as it acts as a null value. The lower key value is on the left and the higher value key is on the right.



```
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/bst.run 30
  true
  true
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/bst.run 1
  false
  (vec 30 (vec 10 (vec 5 (vec 2 (vec 1) (vec 0)) (vec 8)) (vec 11)) (vec 40))
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/bst.run 37
  false
  (vec 30 (vec 10 (vec 5 (vec 2) (vec 8)) (vec 11)) (vec 40 (vec 37) (vec 0)))
● @SRaswan → /workspaces/ucsd-cse231-sp24-05-egg-eater-SRaswan (main) $ ./tests/bst.run 7
  false
  (vec 30 (vec 10 (vec 5 (vec 2) (vec 8 (vec 7) (vec 0))) (vec 11)) (vec 40))
```



4. Other Programming Languages

Pick two other programming languages you know that support heap-allocated data, and describe why your language's design is more like one than the other.
Python and C both support heap-allocated data. I believe that my program is much more like C in this case as Python has private heap management for all Python objects and data structures. The Python memory manager automatically knows how to manage this data and even garbage collect. In C, we need to use malloc to allocate memory from the heap for us to use, which also requires a size to be specifically specified, just like our Vec Expr+ has the size of Expr+ +1 to be allocated and we do not have automatic allocation of heap data unless we specifically use Vec. C can free memory with free(), which needs to be specifically called to for garbage collection. We will probably implement something like this later in our language, which is still not as automatic as Python does it.

5. Sources

Class Lecture Notes
BST visualization and testing: https://www.cs.usfca.edu/~galles/visualization/BST.html
Here I learned the idea of saving the heap object size in the first 8 bytes of memory:
https://maxsnew.com/teaching/eecs-483-fa21/lec_tuples_notes.html
Python heap: https://docs.python.org/3/c-api/memory.html
C heap: https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/