Egg Eater Report

My implementation:

Parse:

Parsing for tuple expression, it also panics if the expression only contains the tuple keyword. Otherwise, parse each element and push into array then return the expression of tuple with these elements.

```
let op: &String = match &vec[0] {
    Sexp::Atom(S(name: &String)) => name,
    _ => panic!("Invalid expression"),
};
match op.as_str() {
    "tuple" => {
        if vec.len() == 1 {
            panic!("Invalid tuple");
        }
        let mut arr: Vec<Expr> = Vec::new();
        for ele: &Sexp in &vec[1..] {
            arr.push(parse_expr(ele));
        }
        Expr::Tuple(arr)
    },
```

Simply parse index with actual two expressions.

```
[Sexp::Atom(S(op: &String)), e1: &Sexp, e2: &Sexp] if op == "index" => {
    Expr::Index(Box::new(parse_expr(e1)), Box::new(parse_expr(e2)))
},
```

Tuple:

I implement tuple features with the first element should be declared the size of the tuple and the tuple is 0-index based. As for instructions, it's just simply stack above r15.

```
Expr::Tuple(es: &Vec<Expr>) => {
    let mut instrs: String = String::new();
    let arr_len: i32 = es.len() as i32;
    for (i: usize,e: &Expr) in es.iter().enumerate() {
        let e_is: String = compile_to_instrs(e, si: si + i as i32, env, ctx);
        let offset: usize = i * 8;
        instrs = instrs + "\n" + &e_is + "\n" + &format!("mov [r15+{offset}], rax");
    }
    instrs = instrs + &format!("
    mov rax, r15
    add rax, 1
    add r15, {}
    ", arr_len*8);
    instrs
},
```

Index:

First of all, I compute the value of the index and store it into rsp+offset, then I load the tuple into rax. Second, I check if the value in rax is actually a tuple by checking its value and check if the index is out of bound by comparing the size of tuple with the index.

```
    Expr::Index(e1: &Box<Expr>, e2: &Box<Expr>) => {
        let e1_instrs: String = compile_to_instrs(e: e1, si: si+1, env, ctx);
        let e2_instrs: String = compile_to_instrs(e: e2, si, env, ctx);
        let offset: i32 = si * 8;
        format!("
{e2_instrs}
mov [rsp + {offset}], rax
{e1_instrs}
mov rbx, rax
cmp rbx, 7
mov rdx, 4
jle throw_error
and rbx, 1
cmp rbx, 1
mov rdx, 4
jne throw_error
mov rbx, [rax-1]
cmp rbx, [rsp + {offset}]
mov rdx, 3
jle throw_error
mov rbx, [rsp+{offset}]
imul rbx, 4
add rbx, 7
mov rax, [rax+rbx]
        ")
    },
```

I will use point.snek as an example to show how the values are arranged on the heap. First we construct x so the heap will arrange 3 words which are size first element and second element respectively. Then we construct points y and z.

size(x)	x[0]	x[1]	size(y)	y[0]	y[1]	size(z)	z[0]	z[1]
4	6	8	4	2	4	4	8	12

Required Tests:

tuple.snek

This is a simple test case for tuple features, the program constructs a tuple with 3 elements [10, 20, 30]. Then print the second element and return the sum of second and third elements.

```
(let ((x (tuple 3 10 20 30)))
    (block
        (print (index x 1))
        (+ (index x 1) (index x 2))
    )
)
20
50
```

point.snek

This is a test case for application of tuple features, the program defines a function point to construct a point structure based on the argument of x and y coordinate. And another function that adds two points and returns the result of a new point. The program will print x point(3,4) and y point(1,2) and the result point z(4,6) which is add x and y.

```
(fun (point x y)
  (tuple 2 x y)
)

(fun (add px py)
    (tuple
        2
        (+ (index px 0) (index py 0))
        (+ (index px 1) (index py 1))
    )
)

(let ((x (point 3 4)) (y (point 1 2)) (z (add x y)))
    (block
        (print x)
        (print y)
        (print z)
    )
)
[3, 4]
[1, 2]
[4, 6]
[4, 6]
```

bst.snek

```
(fun (node ele left right)
    (tuple 3 ele left right)
)

(fun (insert bst ele)
    (block
        (if (= bst nil)
            (node ele nil nil)
            (if (> ele (index bst 0))
                (node (index bst 0) (index bst 1) (insert (index bst 2) ele))
                (if (< ele (index bst 0))
                    (node (index bst 0) (insert (index bst 1) ele) (index bst 2))
                    bst
                )
            )
        )
    )
)


(let ((root (node 5 nil nil)) (i 1))
    (loop
        (if (> i 10)
            (break root)
            (block
                (set! root (insert root i))
                (set! i (+ i 1))
            )
        )
    )
)
[5, [1, nil, [2, nil, [3, nil, [4, nil, nil]]]], [6, nil, [7, nil, [8, nil, [9, nil, [10, nil, nil]]]]]]
```

tuple_error_bounds.snek

This is a test case for testing indexes out of bounds of tuple case. The tuple has only three elements so when I index 3 in report index out of range error in the runtime phase.

```
(let ((x (tuple 3 10 20 30)))
    (block
        (print (index x 2))
        (+ (index x 2) (index x 3))
    )
)
30
index out of range
```

tuple_error_tag.snek

This is a test case for testing when I index something not tuple. y variable is assigned number 20 not a tuple, so when I (index y 0) it reports index to not array error in the runtime phase.

```
(let ((x (tuple 3 10 20 30)) (y 20))
    (block
        (print (index x 1))
        (print (+ (index x 0) y))
        (print (index y 0))
    )
)
20
30
index to not array
```

tuple_error_3.snek

This is an extra test case for testing we only give one argument to index, the situation will not be parsed into regular expression so the program will panic invalid bind.

```
(let ((x (tuple 3 10 20 30) (y 20)))
    (block
        (print (index x 1))
        (print (+ (index x 0) y))
        (print (index x))
    )
)
```

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.10s
     Running `target/debug/diamondback ./inputs/tuple_error3.snek ./inputs/tuple_error3.s`
thread 'main' panicked at 'Invalid bind', src/main.rs:70:18
```

Compare with other Languages:

C++ and Python:

Every object in Python is managed by the heap data structure. In C++, when we new memory to a pointer it is managed by the heap. In my opinion, I think our language is more similar to C++, since we have to declare the size of the tuple which is similar to C++ using new to allocate memory.

Reference:

https://www.javatpoint.com/python-memory-management
https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/
https://edstem.org/us/courses/38748/discussion/3145567