- Compilation of your compiler (ocamlc)
- Execution of your compiler, compilation of bird (hatch)
- Execution of bird program (foo.run)

Memory Management

1. Let user handle allocation/deallocation
2. Do allocation/deallocation work for programmer

Concerns:

- What invariants do we need/maintain on memory?
- Whose job is it to maintain them?
- How do we expect it to be used?
- What structures/algorithms will we use to track all of this?

Language similar to Eagle, but:

<expr> ::= ....
| free(<expr>)

Albatross

C-bird

```
let t1 = (1, 2) in
let t2 = (3, 4) in
let j = free(t1) in
(5, 6)
```

```
let t1 = (10, 11) in
let t2 = (12, 13, 14) in
let j = free(t1) in
(15, 16, 17, 18)
```

A.

1. Heap segmented into regions of varying block sizes (e.g. 2-tuples, 3-tuples, ...)
2. When a tuple is freed, use a queue of metadata of usable memory to track what's available so we can reuse that memory.
3. Statically analyze our program to see what segment sizes we need
?. How do we choose to size the segments?
   * Approach: during analysis, track lexical frequency of different sizes (may not be representative)

B.

1. Keep a dictionary mapping sizes of memory to a collection of memory regions of that size
2. Using heap_cursor.

? How do we reclaim memory that has been carved out in a different size?

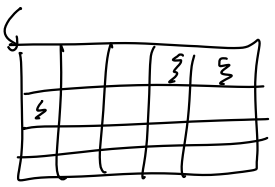
General Question:

* Where do these data structures live?

C.

1. Restructure tuples as linked lists. (avoid fragmentation)

? . How do we track which pieces are available



malloc

1. There are "free block records" describing where in the heap free memory can be found.
2. FBR tracks size of free block
3. FBR also contains ptr to next FBR.
4. Forms cyclic queue
5. Incrementally merges FBRs as they become available
6. Sorted by memory address
7. If we need more memory, call sysbrk
8. FBRs are stored in the memory that they describe

* No allocation is ever smaller than one FBR