```
; just-read.snake
(def (our_main _)
(read_line))
```

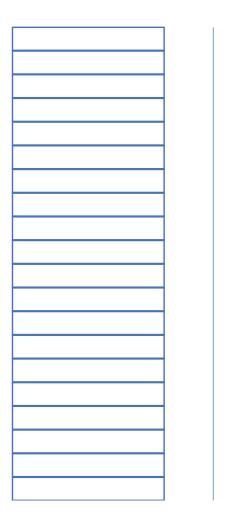
```
$ ./just-read.run
10 9 7
(10,(9,(7,false)))
```

```
; simple-read.snake
(def (our_main _)
  (let (line (read_line))
    (fst (snd line))))
```

```
$ ./just-read.run
10 9 7
???
```

```
/* Reads a line of space-separated numbers from the user. Allocates
a pair-based list containing those numbers, and returns a reference
to it, updating the heap pointer appropriately. Pairs are allocated
in order with the list.*/
int* read_line_c(int* heap_start) {
   char* line = NULL;
   size_t size = 0;
   getline(&line, &size, stdin);
   char* tok = strtok(line, " ");
   while(tok != NULL) { ... }
}
```

```
$ ./min-max.run
10 90 80
(10,90)
??? ; first updated print
200 300 40
(40,300)
??? ; second updated print
```







```
; just-read.snake
(def (our_main _)
    (read_line))

$ ./just-read.run
10 9 7
(10,(9,(7,false)))
```

```
; simple-read.snake
  (def (our_main _)
     (let (line (read_line))
        (fst (snd line))))
```

```
$ ./just-read.run
10 9 7
???
```

```
/* Reads a line of space-separated numbers from the user. Allocates
a pair-based list containing those numbers, and returns a reference
to it, updating the heap pointer appropriately. Pairs are allocated
in order with the list.*/
int* read_line_c(int* heap_start) {
  char* line = NULL;
  size_t size = 0;
  getline(&line, &size, stdin);
  char* tok = strtok(line, " ");
  while(tok != NULL) { ... }
}
```

```
; min-max.snake
(def (max lst) \dots); return largest num in list
(def (min lst) ...) ; return smallest num in list
(def (maxmin 1)
  (pair (min 1) (max 1)))
(def (read_maxmin)
  (let (line (read_line))
   (if (== line false) false
       (maxmin line))))
(def (our_main so_far)
  (let (next (print (read_maxmin)))
   (if (== next false)
      so_far
      (let (updated (print (pair next so_far)))
        (our_main updated)))))
$ ./min-max.run
10 90 80
(10,90)
???
        ; first updated print
```

; second updated print

200 300 40 (40,300)