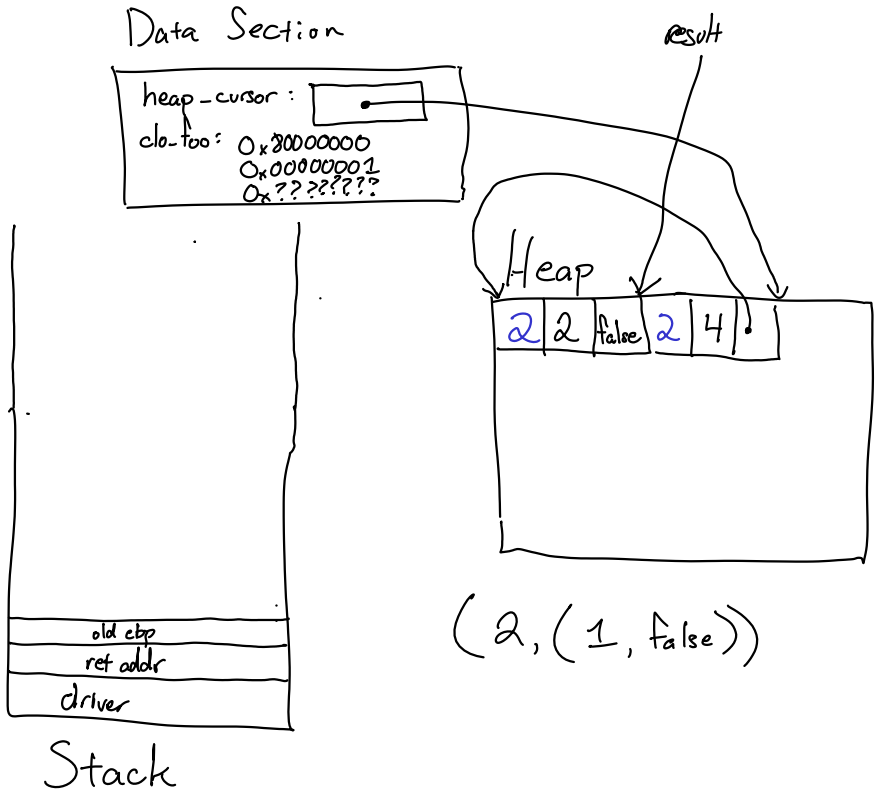
Memory

```
def foo n =
  if n = 0 then false else
    let x = foo (n-1) in
    (n, x)
end
foo 2
```



Set

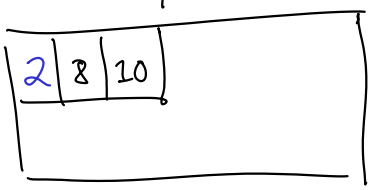
<expr> ::= ... | <expr> [<expr>] := <expr>

```
let x = (2, 5) in
x[0] := 4
```

C

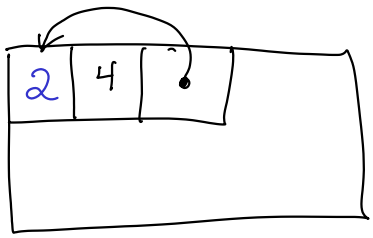
assignment :

```
int x;
x = 4;      <-- "expression stmt"
y = (x = 4);
```



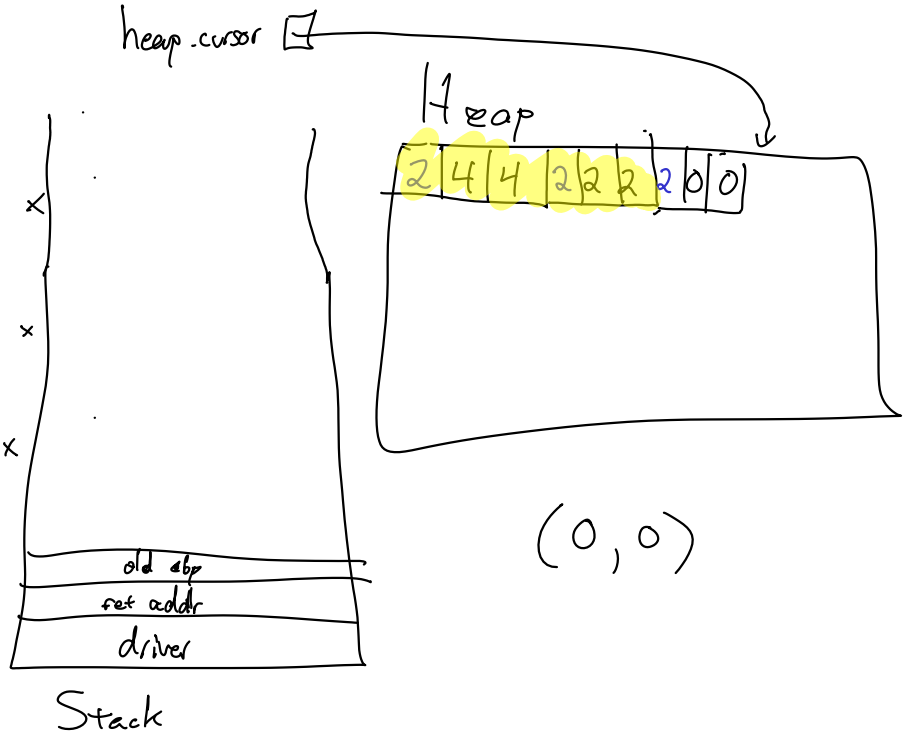
$e_1[e_2] := e_3$ ⟶ e_1 is a tuple
e_2 is an int (valid index)
store result of e_3 in e_1 at e_2, replacing old contents
returns result of e_3

```
let x = (2, 5) in
x[1] := x
```

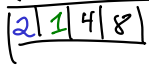


More Memory

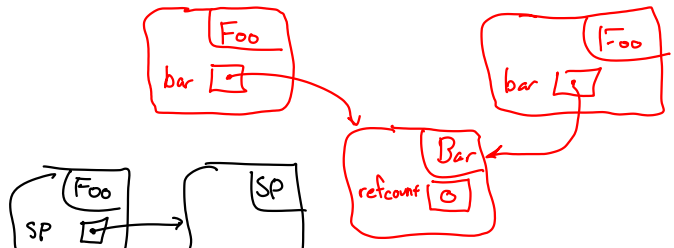
```
def foo n =
    let x = (n, n) in
    if n > 0 then foo(n-1) else x
end

foo 2
```

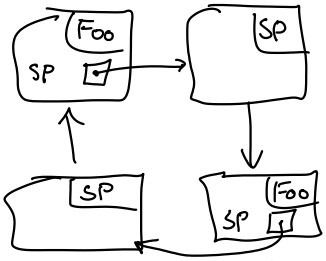


Memory Management Strategies

* Manual management — programmer explicitly frees memory
  * (2, 3) — implicit allocation
  * free(<expr>) — explicit deallocation
  * "Not my problem"
* Reference counting: every heap allocation remembers how many pointers there are to it
  * (2, 4) ⟹ 
  * C++11 : shared_ptr<T>

```
shared_ptr<Foo> p = ....;
shared_ptr<Foo> q = p;
```





* Garbage Collection
  * Program allocates when it wants
  * GC looks for inaccessible memory automatically & frees it