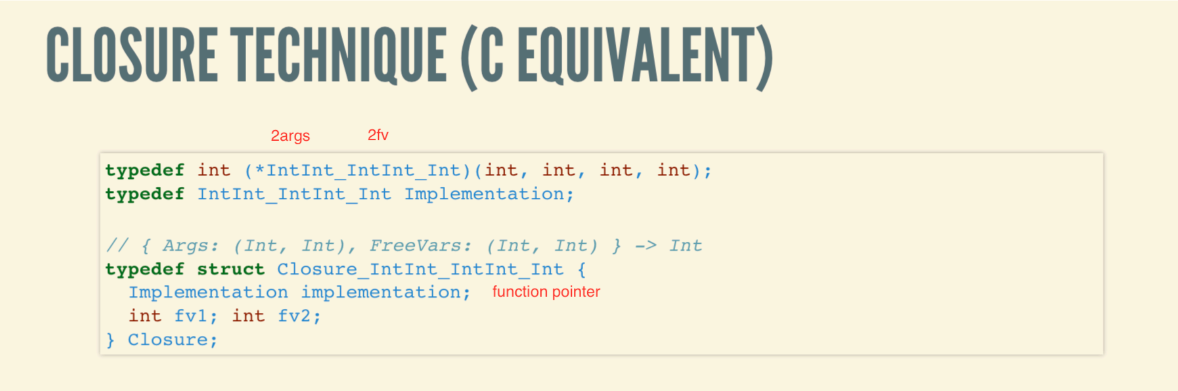
DAY 10

Closure is a data structure
Closure is in pattern matching in the function_implementation
closure is a function which does not contain free variables

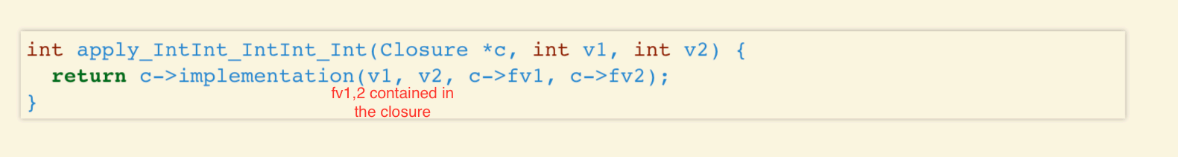


CLOSURE TECHNIQUE (C EQUIVALENT)

2args 2fv

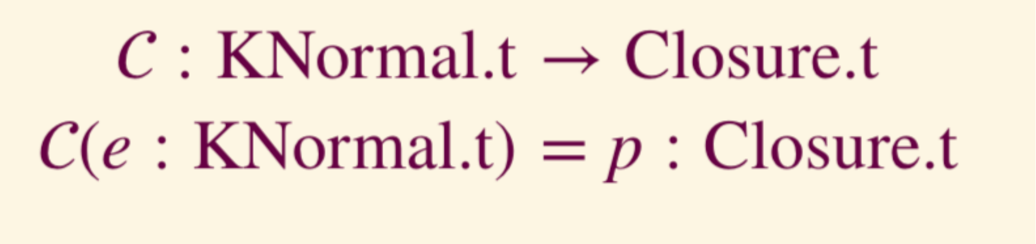
```c
typedef int (*IntInt_IntInt_Int)(int, int, int, int);
typedef IntInt_IntInt_Int Implementation;

// { Args: (Int, Int), FreeVars: (Int, Int) } -> Int
typedef struct Closure_IntInt_IntInt_Int {
  Implementation implementation;
  int fv1; int fv2;
} Closure;
```

function pointer



```c
int apply_IntInt_IntInt_Int(Closure *c, int v1, int v2) {
  return c->implementation(v1, v2, c->fv1, c->fv2);
}
```

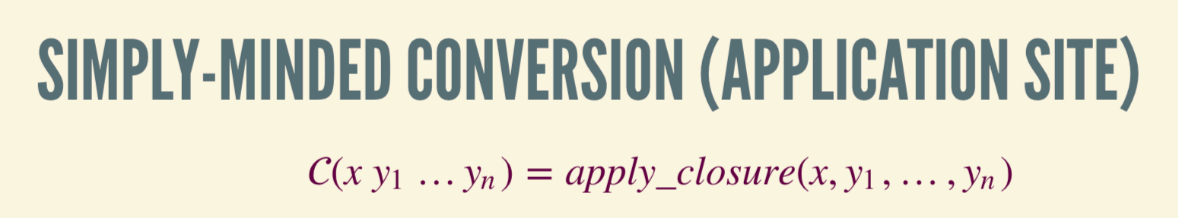
fv1,2 contained in the closure



$$\mathcal{C} : \text{KNormal.t} \to \text{Closure.t}$$

$$\mathcal{C}(e : \text{KNormal.t}) = p : \text{Closure.t}$$

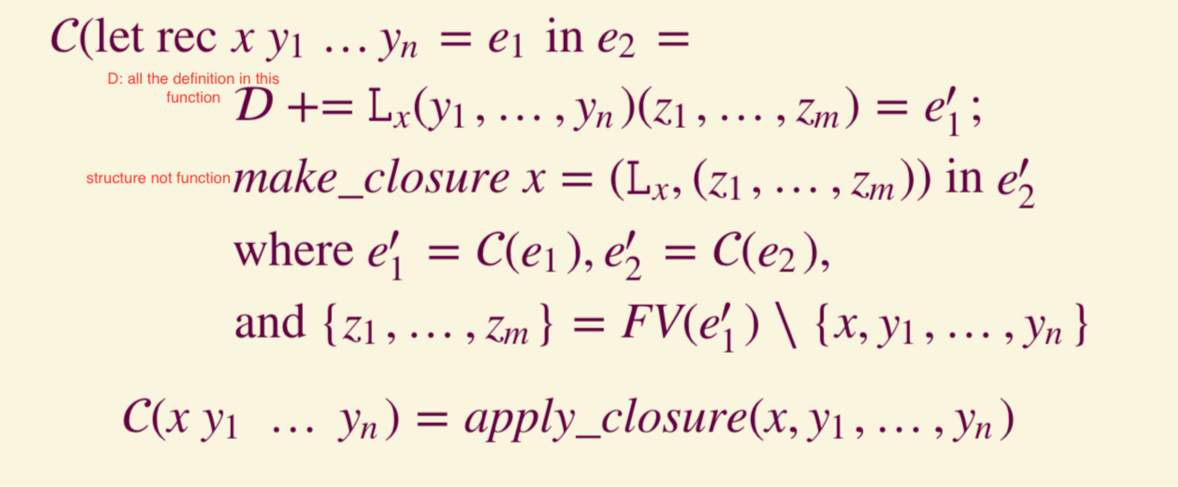
Not using any env so this is a pure syntax structure conversion.



SIMPLY-MINDED CONVERSION (APPLICATION SITE)

$$\mathcal{C}(x\ y_1\ \ldots\ y_n) = apply\_closure(x, y_1, \ldots, y_n)$$

Function application convert to closure application
\



$$\mathcal{C}(\text{let rec } x\ y_1\ \ldots\ y_n = e_1 \text{ in } e_2 =$$

D: all the definition in this function

$$\mathcal{D} \mathrel{+}= \mathtt{L}_x(y_1, \ldots, y_n)(z_1, \ldots, z_m) = e'_1\,;$$

structure not function

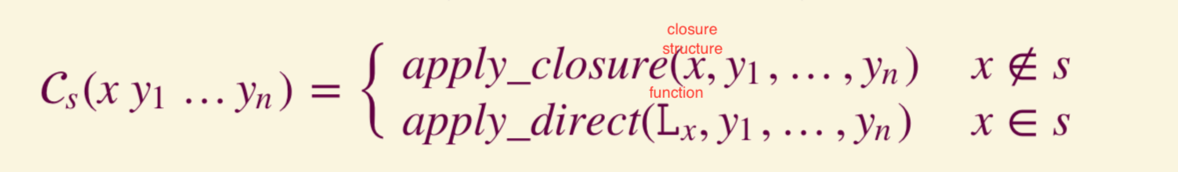
$$\mathit{make\_closure}\ x = (\mathtt{L}_x, (z_1, \ldots, z_m)) \text{ in } e'_2$$

$$\text{where } e'_1 = \mathcal{C}(e_1), e'_2 = \mathcal{C}(e_2),$$

$$\text{and } \{z_1, \ldots, z_m\} = FV(e'_1) \setminus \{x, y_1, \ldots, y_n\}$$

$$\mathcal{C}(x\ y_1\ \ldots\ y_n) = \mathit{apply\_closure}(x, y_1, \ldots, y_n)$$

Slower to access than registers. Inefficient!



closure structure

$$\mathcal{C}_s(x\ y_1\ \ldots\ y_n) = \begin{cases} \mathit{apply\_closure}(x, y_1, \ldots, y_n) & x \notin s \\ \mathit{apply\_direct}(\mathtt{L}_x, y_1, \ldots, y_n) & x \in s \end{cases}$$

function