```
expr := <number> | <name> | true | false
     |  (<name> <expr> <expr>)
     |  (if <expr> <expr> <expr>)
     |  (let (<name> <expr>) <expr> ...)
     |  (+ <expr> <expr>)
     |  (< <expr> <expr>)
     |  (dict <name> <expr> <name> <expr>)
     |  (get <expr> <name>)
     |  (update <expr> <name> <expr>)
     | (null <t>)
```

```
type expr =
  | ENum of int
  | EBool of bool
  | EId of string
  | EIf of expr * expr * expr
  | ELet of string * expr * expr
  | EPlus of expr * expr
  | ELess of expr * expr
  | EApp of string * expr * expr
  | EDict of string * expr * string * expr
  | EGet of expr * string
  | EUpdate of expr * string * expr
  | ENull of typ
```

```
def := (def <name> (<name> : <t> <name> : <t>) : <t>
         <expr>)
    | (type <name> <t>)
    | (data name <variant> .....)
```

better idea but not today

```
t := Num | Bool | (<name> : <t> <name> : <t>)
   | <name>
```

```
type typ = TNum | TBool | TDict of string * typ * string * typ
         | TName of string
```

```
type def =
  | DFun of string * string * typ * string * typ * typ * expr
  | DType of string * typ
```

Null-like thing
Need names for types
} Where do these go?

Write an example of a linked list using these two-element dictionaries:

(dict val 100 next (dict val 200 next (dict val 300 next false)))
(val : Num next: (val : Num next : (val : Num next : Bool)))

need a union type here!

(def sum (l : $\boxed{\text{Link}}$) : Num . . .)

(type Link (val : Num next : Link))

(dict val 100 next $\boxed{\text{(null Link)}}$)
(dict val 100 next (dict val 200 next $\boxed{\text{(null Link)}}$))

both have type Link

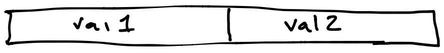
```
type ll =
  | Link of int * ll
  | Empty
```

Option 1: TNull plus "subtyping"
Option 2: Type variable
Option 3: (null T)

$$\frac{T \text{ isn't Num or Bool}}{\Gamma \vdash (\text{null } T) : T}$$

((Point)(null)).x "fun" Java program

```
int64_t print(int64_t val) {
  if (val == TRUE) {
    printf("true");
  } else if (val == FALSE) {
    printf("false");
  } else if ((val & 1L) == 1) {
    printf("%ld", val >> 1);
  } else if (val == 0) { // null
    printf("null");
  } else if ((val & 7L) == 0) { // 7 has 111 at the end
```



How to access val1 / val2   no warnings!

```
idea :  int64_t val1;
        memcpy(&val1, (void*) val, 8 );
```

→ treating val (long long) as ptr

```
idea:
int64_t *addr_val = &val;
```

```
idea:
int64_t *addr_val = (int64_t*) val;
int64_t  val1 = *addr_val;
int64_t  val2 = *(addr_val + 1);
```

TRUST ME

```
typedef struct {
  int64_t val1;
  int64_t val2;
} Dict ;

Dict* d = (Dict*) val;
int64_t val1 = d->val1;
int64_t val2 = d->val2;
```

```
  } else {
    printf("Unknown value: %#010lx", val);
  }
  return val;
}



int main(int argc, char** argv) {
  int64_t* THE_HEAP = calloc(10000, sizeof(int64_t));
  int64_t result = our_code_starts_here(THE_HEAP);
  print(result);
  return 0;
}
```

names of fields

```
(def Point (x : Num y : Num)
  (dict x x y y))
(let (p1 (Point 4 5))
  (let (p2 (Point 6 7))
    (Point (+ (get p1 x) (get p2 x)) (+ (get p1 y) (get p2 y)))))
```

: (x: Num y: Num)

values

What should this print?

A: 22
B: (x : 9 y : 13)
[✓ C: (x : 10 y: 12)]
D: A type error
E: A runtime error

how to write print()

```
(def Point (x : Num y : Num)
  (dict x x y y))
(def PairOfPoints (p1 : (x : Num y : Num)
                   p2 : (x : Num y : Num))
  (dict left p1 right p2))
(let (p1 (Point 4 5))
  (let (p2 (Point 6 7))
    (PairOfPoints p1 p2)))
```

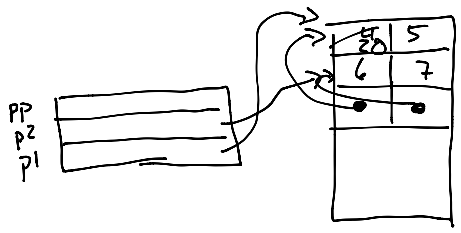
names

values

What should this print?

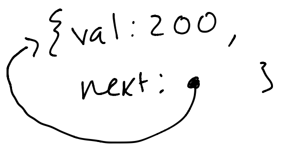
A: (x : 4 y : 5) (x : 6 y : 7)
B: (p1 : (x : 4 y : 5) p2 : (x : 6 y : 7))
C: (left : (x : 4 y : 5) right : (x : 6 y : 7))
D: A type error
E: A runtime error

```
(def Point (x : Num y : Num)
  (dict x x y y))
(def PairOfPoints (p1 : (x : Num y : Num)
                   p2 : (x : Num y : Num))
  (dict left p1 right p2))
(let (p1 (Point 4 5))
  (let (p2 (Point 6 7))
    (let (pp (PairOfPoints p1 p2))
      (update p1 x 20)
      (get pp left))))
```



What should this print?

A: (x : 4 y : 5)
[✓ B: (x : 20 y : 5)]
C: (left : (x : 20 y : 5) right : (x : 6 y : 7))
D: (left : (x : 4 y : 5) right : (x : 6 y : 7))
E: A type or runtime error


{val: 200,
next: • }