Java

new Object
new int[10]

Python (built-in syntax)

[1, 2, 3] {} (...)

o.x = ____;
a[i] = ____;

* ptr

```
expr := <number> | <name> | true | false
     |  (<name> <expr> <expr>) // two arg functions
     |  (if <expr> <expr> <expr>)
     |  (let (<name> <expr>) <expr>)
     |  (+ <expr> <expr>)
     |  (< <expr> <expr>)
     | (dict (<name> <expr>) (<name> <expr>))
     | (get <expr> <name>)
```

```
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
```

typ

typ

```
def := (def <name> (<name> : <t> <name> : <t>) : <t>
        <expr>)
```

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

```
prog := def ... <expr>
```

τ := Num | Bool | (x:τ y:τ)

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

```
type prog = def list * expr
```

Write some example programs!

(dict (a 5) (b 7)) —parse→ EDict(..)

output → ADDRESS {a:5, b:7}

(let (d (dict (a 5) (b 7)))
  (get d a))

output → 5

(let (d (dict (a 5) (b 7)))
  (get d notthere))

output → A: WF
B: TC
C: Runtime
D: "works"

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{dict } (x_1\ e_1)\ (x_2\ e_2)) : (x_1:\tau_1\ x_2:\tau_2)}$$

$$\frac{\Gamma \vdash e : (x:\tau\ x_2:\tau_2)}{\Gamma \vdash (\text{get } e\ x) : \tau}$$

$$\frac{\Gamma \vdash e : (x_1:\tau_1\ x:\tau)}{\Gamma \vdash (\text{get } e\ x) : \tau}$$

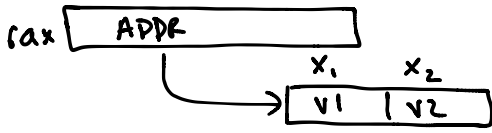
```
let rec tc_e (defs : def list) (env : (string * typ) list) (e : expr) : typ =
  match e with
    | EGet(e,x) → ...

    | EDict(x1,e1,x2,e2) → ...
```

```
let rec e_to_is (e : expr) (si : int) (env : tenv) (defs : def list) =
  match e with
```

```
| EGet(e, x) →
  let e_is = e_to_is e1 si env defs in
  ( match get_typ e with
      | TDict(x1, _, x2, _) →
        if x = x1 then ["mov rax, [rax]"]
        else ["mov rax, [rax+8]"]
      | _ → failwith ...
  )
| EDict(x1, e1, x2, e2) →
  let e1_is = e_to_is e1 si env defs in
  let e2_is = e_to_is e2 (si+1) env defs in
  Call alloc_dict
  address in rax
```

how to store types after tc



num 63 bit 1
true 62 0's 10
false 640's
addresses 61 bits 000

Python!

char* x1 char* x2

```
section .text
global our_code_starts_here
```

```
int64_t alloc_dict(int64_t v1, int64_t v2) { (* ...
do malloc, store v1, v2
return address;
}
```

```
our_code_starts_here:
```

```
int main(int argc, char** argv) {
```

```
  int64_t result = our_code_starts_here(                    );

  print(result);
  return 0;
}
```

```
ret
```