```
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>)
```

```
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
```

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

```
type typ = TNum | Tbool | TDict of string * typ * string * typ
t := Num | Bool | (<name> : <t> <name> : <t>)
```

```
type def =
  | DFun of string * string * typ * string * typ * typ * expr
prog := def ... <expr>
```

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

```
section .text
global our_code_starts_here




our_code_starts_here:









ret
```

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




  int64_t result = our_code_starts_here(                        );


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

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