TYPECHECKING

This worksheet describes a simple programming language called P (it is a simplified version of Pascal). Here is an example program.

```
main():int
{
  a:int;

  a := f(1, 2) + 1;
  if (a > 5) return 0;
  else return 1;
}

f(x:int, y:int):int
{
  return x+y+100;
}
```

We provide a grammar for this language.

Statements

$$\begin{aligned}\langle \text{stmt} \rangle ::= \; & \langle \text{id} \rangle := \langle \text{expr} \rangle; \\ | \; & \textbf{if } (\langle \text{expr} \rangle) \; \langle \text{stmt} \rangle \; \textbf{else } \langle \text{stmt} \rangle \\ | \; & \textbf{return } \langle \text{expr} \rangle; \\ | \; & \{\langle \text{stmt} \rangle \ldots \langle \text{stmt} \rangle\}\end{aligned}$$

Types

$$\langle \text{type} \rangle ::= \textbf{int} \mid \textbf{bool} \mid \textbf{fun } (\langle \text{type} \rangle, \ldots, \langle \text{type} \rangle) : \langle \text{type} \rangle$$

Expressions

$$\begin{aligned}\langle \text{expr} \rangle ::= \; & \langle \text{id} \rangle \\ | \; & \langle \text{expr} \rangle + \langle \text{expr} \rangle \\ | \; & \langle \text{expr} \rangle > \langle \text{expr} \rangle \\ | \; & \langle \text{id} \rangle (\langle \text{expr} \rangle, \ldots, \langle \text{expr} \rangle) \\ | \; & \langle \text{int-literal} \rangle \\ | \; & \textbf{true} \mid \textbf{false}\end{aligned}$$

Functions

$$\begin{aligned}\langle \text{func} \rangle ::= \; & \langle \text{id} \rangle (\langle \text{decl} \rangle, \ldots) : \langle \text{type} \rangle \; \{\langle \text{decl} \rangle \ldots \langle \text{decl} \rangle \; \langle \text{stmt} \rangle\} \\ \langle \text{decl} \rangle ::= \; & \langle \text{id} \rangle : \langle \text{type} \rangle\end{aligned}$$

A program is just a sequence of functions (⟨func⟩s).

Your job is to write a typechecker for this language. You will be writing two functions.

- Calling `t = typeExpr(e, env)` returns the type `t` of the expression `e` in environment `env`.
- Calling `typeStmt(stmt, env)` reports an error if `stmt` has type errors in environment `env`. If there are no errors it returns nothing.

Assume an environment is an object that has already been supplied. You can look up a name by doing "`env[name]`" and you'll get a type back.

The environment contains the types of all the functions as well as the types of the parameters and local variables of the functions being checked. For example, the environment when checking `f` in the example contains the type of `main`, `f`, `x`, and `y`. It also contains the return type under the special name "`$ret`."

Write pseudocode for the functions `typeExpr` and `typeStmt`. We have gotten it started for you.

```
def typeExpr(e, env):
  if e has the form "var":
    return env[var]
  elif e has the form "n":
    return int
  elif e has the form "e1 + e2":
    check typeExpr(e1, env) == int
    check typeExpr(e2, env) == int
    return int
  elif e has the form "e1 > e2":
    check typeExpr(e1, env) == int
    check typeExpr(e1, env) == int
    return bool
  elif e has the form "func(e1, ..., en)":
    t = env[func]
    check t has the form "fun (t1, ..., tn):rt"
    for i = 1..n: check typeExpr(ei, env) == ti
    return rt

def typeStmt(s, env):
  if s has the form "return e;":
    check env["$ret"] == typeExpr(e, env)
  elif s has the form "var := e":
    check env[var] == typeExpr(e, env)
  elif s has the form "if (e) s1 else s2":
    check typeExpr(e, env) == bool
    typeStmt(s1, env); typeStmt(s2, env)
  elif s has the form "{ s1 ... sn }":
    for i in 1..n: typeStmt(si, env)
```