BÁCH KHOA E-LEARNING

Đã bắt đầu vào lúc	Tuesday, 5 October 2021, 9:03 AM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Sunday, 10 October 2021, 7:27 PM
Thời gian thực hiện	5 ngày 10 giờ
Điểm	5,00/5,00
Điểm	**10,00** của 10,00 (**100**%)

Câu hỏi **1**

Chính xác

Điểm 1,00 của 1,00

Given the AST declarations as follows:

class Exp(ABC): #abstract class

class BinOp(Exp): #op:str,e1:Exp,e2:Exp #op is +,-,*,/,&&,||, >, <, ==, or !=

class UnOp(Exp): #op:str,e:Exp #op is -, !

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

and the Visitor class is declared as follows:

class StaticCheck(Visitor):

    def visitBinOp(self,ctx:BinOp,o): pass

    def visitUnOp(self,ctx:UnOp,o):pass

    def visitIntLit(self,ctx:IntLit,o): pass

    def visitFloatLit(self,ctx,o): pass

    def visitBoolLit(self,ctx,o): pass

Rewrite the body of the methods in class StaticCheck to check the following type constraints:

- + , - and * accept their operands in int or float type and return float type if at least one of their operands is in float type, otherwise, return int type
- / accepts their operands in int or float type and returns float type
- !, && and || accept their operands in bool type and return bool type
- >, <, == and != accept their operands in any type but must in the same type and return bool type

If the expression does not conform the type constraints, the StaticCheck will raise exception TypeMismatchInExpression with the innermost sub-expression that contains type mismatch.

Your code starts at line 55

For example:

Test	Result
`BinOp("+",IntLit(3),BoolLit(True))`	`Type Mismatch In Expression: BinOp("+",IntLit(3),BoolLit(True))`

Answer: (penalty regime: 0 %)

```
class StaticCheck(Visitor):

    def visitBinOp(self,ctx:BinOp,o):
        type1 = self.visit(ctx.e1,o)
        type2 = self.visit(ctx.e2,o)
        if ctx.op in ['+','-','*']:
            if type1 == 3 or type2 == 3:
                raise TypeMismatchInExpression(ctx)
            if type1 == 2 or type2 == 2:
                return 2
            else:
                return 1

        if ctx.op == '/':
            if type1 == 3 or type2 == 3:
                raise TypeMismatchInExpression(ctx)
            return 2

        if ctx.op == '&&' or ctx.op == '||':
            if type1 != 3 or type2 != 3:
                raise TypeMismatchInExpression(ctx)
            return 3

```

	Test	Expected
✔	`BinOp("+",IntLit(3),BoolLit(True))`	`Type Mismatch In Expression: BinOp("+",IntLit(3),BoolLit(`
✔	`BinOp("*",BinOp("+",IntLit(3),FloatLit(3.4)),BinOp(">",IntLit(3),FloatLit(2.1)))`	`Type Mismatch In Expression: BinOp(">",IntLit(3),FloatLit`
✔	`BinOp("&&",BinOp(">",BinOp("-",IntLit(3),FloatLit(3.4)),UnOp("-",FloatLit(2.1))),UnOp("-",BoolLit(True)))`	`Type Mismatch In Expression:`
✔	`UnOp("-",BinOp(">",BinOp("-",IntLit(3),FloatLit(3.4)),UnOp("-",FloatLit(2.1))))`	`Type Mismatch In Expression: ",BinOp(">",BinOp("-",IntLit(3),FloatLit(3.4)),U`
✔	`BinOp(">",BinOp("&&",BoolLit(True),BoolLit(False)),BinOp("||",BoolLit(True),UnOp("-",FloatLit(2.3))))`	`Type Mismatch In Expression: BinOp("||",BoolLit(True),UnO`
✔	`UnOp("!",BinOp("==",IntLit(3),BinOp("*",IntLit(5),IntLit(7))))`	
✔	`UnOp("!",BinOp("==",IntLit(3),BinOp("/",IntLit(5),IntLit(7))))`	`Type Mismatch In Expression: BinOp("==",IntLit(3),BinOp("/`
✔	`UnOp("!",BinOp("-",IntLit(3),BinOp("/",IntLit(5),IntLit(7))))`	`Type Mismatch In Expression: ",IntLit(3),BinOp("/",IntLit`
✔	`BinOp("/",IntLit(8),BinOp("<",IntLit(3),IntLit(8)))`	`Type Mismatch In Expression: BinOp("/",IntLit(8),BinOp("<`
✔	`BinOp("||",BoolLit(True),BinOp("<",IntLit(3),IntLit(8)))`	

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **2**

Chính xác

Điểm 1,00 của 1,00

Given the AST declarations as follows:

class Program: #decl:List[VarDecl],exp:Exp

class VarDecl: #name:str,typ:Type

class Type(ABC): #abstract class

class IntType(Type)

class FloatType(Type)

class BoolType(Type)

class Exp(ABC): #abstract class

class BinOp(Exp): #op:str,e1:Exp,e2:Exp #op is +,-,*,/,&&,||, >, <, ==, or !=

class UnOp(Exp): #op:str,e:Exp #op is -, !

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

class Id(Exp): #name:str

and the Visitor class is declared as follows:

```
class StaticCheck(Visitor):
    def visitProgram(self,ctx:Program,o):pass
    def visitVarDecl(self,ctx:VarDecl,o): pass
    def visitIntType(self,ctx:IntType,o):pass
    def visitFloatType(self,ctx:FloatType,o):pass
    def visitBoolType(self,ctx:BoolType,o):pass
    def visitBinOp(self,ctx:BinOp,o): pass
    def visitUnOp(self,ctx:UnOp,o):pass
    def visitIntLit(self,ctx:IntLit,o): pass
    def visitFloatLit(self,ctx,o): pass
    def visitBoolLit(self,ctx,o): pass
    def visitId(self,ctx,o): pass
```

Rewrite the body of the methods in class StaticCheck to check the following type constraints:

- + , - and * accept their operands in int or float type and return float type if at least one of their operands is in float type, otherwise, return int type
- / accepts their operands in int or float type and returns float type
- !, && and || accept their operands in bool type and return bool type
- >, <, == and != accept their operands in any type but must in the same type and return bool type
- the type of an Id is from the declarations, if the Id is not in the declarations, exception UndeclaredIdentifier should be raised with the name of the Id.

If the expression does not conform the type constraints, the StaticCheck will raise exception TypeMismatchInExpression with the innermost sub-expression that contains type mismatch.

Your code starts at line 90

For example:

Test	**Result**
`Program([VarDecl("x",IntType())],BinOp("*",BinOp("+",Id("x"),FloatLit(3.4)),BinOp(">",IntLit(3),FloatLit(2.1))))`	`Type M BinOp(`

Answer: (penalty regime: 0 %)

```
    def visitUnOp(self,ctx:UnOp,o):
        type0 = self.visit(ctx.e,o)
        if ctx.op == '-':
            if type0 == 3:
                raise TypeMismatchInExpression(ctx)
            return type0

        if ctx.op == '!':
            if type0 != 3:
                raise TypeMismatchInExpression(ctx)
            return type0

    def visitIntLit(self,ctx:IntLit,o):
        return 1

    def visitFloatLit(self,ctx,o):
        return 2

    def visitBoolLit(self,ctx,o):
        return 3

    def visitId(self,ctx,o):
        for var in o:
            if ctx.name == var.name:
```

	Test
✔	`Program([],BinOp("+",IntLit(3),BoolLit(True)))`
✔	`Program([VarDecl("x",IntType())],BinOp("*",BinOp("+",Id("x"),FloatLit(3.4)),BinOp(">",IntLit(3),FloatLit(2.1))))`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",BoolType())],BinOp("&&",BinOp(">",BinOp("-",IntLit(3),FloatLit(3.4)),",Id("y"))))`
✔	`Program([VarDecl("x",IntType())],UnOp("-",BinOp(">",BinOp("-",Id("x"),FloatLit(3.4)),UnOp("-",FloatLit(2.1)))))`
✔	`Program([VarDecl("x",BoolType()),VarDecl("y",BoolType()),VarDecl("z",FloatType())],BinOp(">",BinOp("&&",Id("x"),I",Id("z")))))`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",IntType()),VarDecl("z",IntType())],UnOp("!",BinOp("==",Id("z"),BinOp(`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",IntType()),VarDecl("z",IntType())],UnOp("!",BinOp("==",Id("x"),BinOp(`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",IntType()),VarDecl("z",IntType())],UnOp("!",BinOp("-",Id("z"),BinOp("`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",IntType()),VarDecl("z",IntType())],BinOp("/",Id("x"),BinOp("<",Id("y"`
✔	`Program([VarDecl("x",IntType()),VarDecl("y",IntType()),VarDecl("z",IntType())],BinOp("||",BoolLit(True),BinOp("<"`

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **3**

Chính xác

Điểm 1,00 của 1,00

Given the AST declarations as follows:

class Program: #decl:List[VarDecl],stmts:List[Assign]

class VarDecl: #name:str

class Assign: #lhs:Id,rhs:Exp

class Exp(ABC): #abstract class

class BinOp(Exp): #op:str,e1:Exp,e2:Exp #op is +,-,*,/,+.,-.,*.,/., &&,||, >, >., >b, =, =., =b

class UnOp(Exp): #op:str,e:Exp #op is -,-., !,i2f, floor

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

class Id(Exp): #name:str

and the Visitor class is declared as follows:

```
class StaticCheck(Visitor):

    def visitProgram(self,ctx:Program,o):pass

    def visitVarDecl(self,ctx:VarDecl,o): pass

    def visitAssign(self,ctx:Assign,o): pass

    def visitBinOp(self,ctx:BinOp,o): pass

    def visitUnOp(self,ctx:UnOp,o):pass

    def visitIntLit(self,ctx:IntLit,o): pass

    def visitFloatLit(self,ctx,o): pass

    def visitBoolLit(self,ctx,o): pass

    def visitId(self,ctx,o): pass
```

Rewrite the body of the methods in class StaticCheck to infer the type of identifiers and check the following type constraints:

- + , - , *, / accept their operands in int type and return int type
- +., -., *., /. accept their operands in float type and return float type
- > and = accept their operands in int type and return bool type
- >. and =. accept their operands in float type and return bool type
- !, &&, ||, >b and =b accept their operands in bool type and return bool type
- i2f accepts its operand in int type and return float type
- floor accept its operand in float type and return int type
- In an assignment statement, the type of lhs must be the same as that of rhs, otherwise, the exception TypeMismatchInStatement should be raised together with the assignment statement.
- the type of an Id is inferred from the above constraints in the first usage,
  - if the Id is not in the declarations, exception UndeclaredIdentifier should be raised together with the name of the Id, or
  - If the Id cannot be inferred in the first usage, exception TypeCannotBeInferred should be raised together with the name of the assignment statement.
- If an expression does not conform the type constraints, the StaticCheck will raise exception TypeMismatchInExpression with the expression.

Your code starts at line 95

Answer: (penalty regime: 0 %)

```
114                 return "float"
115             raise TypeMismatchInExpression(ctx)
116         if ctx.op == "floor":
117             if typ == "none":
118                 o[ctx.e.name] = "float"
119                 typ = "float"
120             if typ == "float":
121                 return "int"
122             raise TypeMismatchInExpression(ctx)
```

```
123
124     def visitIntLit(self,ctx:IntLit,o):
125         return "int"
126
127     def visitFloatLit(self,ctx,o):
128         return "float"
129
130     def visitBoolLit(self,ctx,o):
131         return "bool"
132
133     def visitId(self,ctx,o):
134         if ctx.name in o:
135             return o[ctx.name]
136         raise UndeclaredIdentifier(ctx.name)
137
```

	Test
✔	`Program([VarDecl("x")],[Assign(Id("x"),BinOp("+",IntLit(3),BoolLit(True)))])`
✔	`Program([VarDecl("x")],[Assign(Id("x"),BinOp("*",BinOp("+",Id("x"),IntLit(3.4)),BinOp("-",Id("x"),FloatLit(2.1)))`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],[Assign(Id("z"),BinOp("&&",BinOp(">",BinOp("-",Id("x"),IntLit(3)` `",Id("y"))),UnOp("!",Id("y")))])])`
✔	`Program([VarDecl("x")],[Assign(Id("x"),UnOp("-",BinOp(">.",BinOp("-.",Id("x"),FloatLit(3.4)),UnOp("-.",FloatLit(2`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],` `[Assign(Id("x"),BinOp(">b",BinOp("&&",Id("x"),Id("y")),BinOp("||",BoolLit(False),BinOp(">",Id("z"),IntLit(3)))),`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],[Assign(Id("x"),UnOp("!",BinOp("=",Id("z"),BinOp("*",Id("y"),Id(`
✔	`Program([VarDecl("x"),VarDecl("y")],[Assign(Id("x"),Id("y"))])`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],` `[Assign(Id("x"),UnOp("-.",BinOp("-.",Id("z"),BinOp("/.",UnOp("i2f",Id("y")),Id("x"))))),Assign(Id("y"),FloatLit(3`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],` `[Assign(Id("z"),IntLit(3)),Assign(Id("x"),Id("z")),Assign(Id("y"),BinOp("&&",Id("x"),BinOp("=b",Id("y"),BoolLit(T`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],[Assign(Id("t"),BinOp("||",BoolLit(True),BinOp(">",IntLit(3),Id(`
✔	`Program([VarDecl("x"),VarDecl("y"),VarDecl("z")],` `[Assign(Id("x"),FloatLit(3.0)),Assign(Id("x"),Id("y")),Assign(Id("z"),BinOp(">",IntLit(3),Id("y")))])`

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **4**

Chính xác

Điểm 1,00 của 1,00

Given the AST declarations as follows:

```
class Program: #decl:List[VarDecl],stmts:List[Stmt]
class VarDecl: #name:str
class Stmt(ABC): #abstract class
class Block(Stmt): #decl:List[VarDecl],stmts:List[Stmt]
class Assign(Stmt): #lhs:Id,rhs:Exp
class Exp(ABC): #abstract class
class BinOp(Exp): #op:str,e1:Exp,e2:Exp #op is +,-,*,/,+.,-.,*.,/., &&,||, >, >., >b, =, =., =b
class UnOp(Exp): #op:str,e:Exp #op is -,-., !,i2f, floor
class IntLit(Exp): #val:int
class FloatLit(Exp): #val:float
class BoolLit(Exp): #val:bool
class Id(Exp): #name:str
```

and the Visitor class is declared as follows:

```
class StaticCheck(Visitor):
    def visitProgram(self,ctx:Program,o):pass
    def visitVarDecl(self,ctx:VarDecl,o): pass
    def visitBlock(self,ctx:Block,o): pass
    def visitAssign(self,ctx:Assign,o): pass
    def visitBinOp(self,ctx:BinOp,o): pass
    def visitUnOp(self,ctx:UnOp,o):pass
    def visitIntLit(self,ctx:IntLit,o): pass
    def visitFloatLit(self,ctx,o): pass
    def visitBoolLit(self,ctx,o): pass
    def visitId(self,ctx,o): pass
```

Rewrite the body of the methods in class StaticCheck to infer the type of identifiers and check the following type constraints:

- + , - , *, / accept their operands in int type and return int type
- +., -., *., /. accept their operands in float type and return float type
- > and = accept their operands in int type and return bool type
- >. and =. accept their operands in float type and return bool type
- !, &&, ||, >b and =b accept their operands in bool type and return bool type
- i2f accepts its operand in int type and return float type
- floor accept its operand in float type and return int type
- In an assignment statement, the type of lhs must be the same as that of rhs, otherwise, the exception TypeMismatchInStatement should be raised together with the assignment statement.
- the type of an Id is inferred from the above constraints in the first usage,
  - if the Id is not in the declarations, exception UndeclaredIdentifier should be raised together with the name of the Id, or
  - If the Id cannot be inferred in the first usage, exception TypeCannotBeInferred should be raised together with the assignment statement which contains the type-unresolved identifier.
- For static referencing environment, this language applies the scope rules of block-structured programming language. When there is a declaration duplication of a name in a scope, exception Redeclared should be raised together with the second declaration.
- If an expression does not conform the type constraints, the StaticCheck will raise exception TypeMismatchInExpression with the expression.

Your code starts at line 110

For example:

Test	Result
`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],` `[Assign(Id("x"),Id("y")),Assign(Id("y"),BoolLit(True))])])`	`Type Mismatch In Statement:` `Assign(Id("y"),BoolLit(True))`

Answer: (penalty regime: 0 %)

```
class StaticCheck(Visitor):

    def visitProgram(self,ctx:Program,o):
        o = [{}]
        for decl in ctx.decl:
            self.visit(decl,o)
        for stmt in ctx.stmts:
            self.visit(stmt,o)

    def visitVarDecl(self,ctx:VarDecl,o):
        n=ctx.name
        if n in o[0]:
            raise Redeclared(ctx)
        else:
            o[0][ctx.name] = "none"
    def visitBlock(self,ctx:Block,o):
        block=[{}]
        listDecl=ctx.decl
        listStmts=ctx.stmts
        for nameDecl in listDecl:
           self.visit(nameDecl,block+o)
        for nameStmt in listStmts:
           self.visit(nameStmt,block+o)
```

	Test	Expected
✔	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],` `[Assign(Id("x"),Id("y")),Assign(Id("y"),BoolLit(True))])])`	`Type Mismatch In State` `Assign(Id("y"),BoolLit`

✔	`Program([VarDecl("x")],`
	`[Assign(Id("x"),IntLit(3)),Block([VarDecl("y"),VarDecl("x"),VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Assign(Id("y"),IntLit(3))])])`
		`Redeclared: VarDecl("y`

✔	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y"),VarDecl("x")],` `[Assign(Id("x"),Id("y")),Assign(Id("y"),FloatLit(3))])])`	`Type Cannot Be Inferre`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Block([],`
	`[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),Id("t"))])])])`
		`Undeclared Identifier:`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Block([VarDecl("z")],`
	`[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),UnOp("-",Id("t")))])])])`
		`Type Mismatch In Expre`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Block([VarDecl("z")],`
	`[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),BinOp("-",Id("t"),Id("x")))])])])`
		`Type Mismatch In Expre`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Block([VarDecl("z")],`
	`[Assign(Id("t"),FloatLit(3)),Assign(Id("y"),BinOp("-.",Id("t"),UnOp("i2f",Id("x")))))])])])`
		`Type Mismatch In State`
		`Assign(Id("y"),BinOp('`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],`
	`[Assign(Id("x"),Id("y")),Block([VarDecl("z")],`
	`[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),UnOp("floor",Id("y")))])])])`
		`Type Mismatch In Expre`

✔	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("x")],` `[Assign(Id("x"),FloatLit(3.0)),Assign(Id("t"),Id("x"))]),Assign(Id("x"),Id("t"))])`	`Type Mismatch In State`
✔	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("x")],` `[Assign(Id("x"),FloatLit(3.0))]),Assign(Id("x"),BoolLit(False))])`	`Type Mismatch In State` `Assign(Id("x"),BoolLit`

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi **5**

Chính xác

Điểm 1,00 của 1,00

Given the AST declarations as follows:

class Program: #decl:List[Decl],stmts:List[Stmt]

class Decl(ABC): #abstract class

class VarDecl(Decl): #name:str

class FuncDecl(Decl): #name:str,param:List[VarDecl],local:List[Decl],stmts:List[Stmt]

class Stmt(ABC): #abstract class

class Assign(Stmt): #lhs:Id,rhs:Exp

class CallStmt(Stmt): #name:str,args:List[Exp]

class Exp(ABC): #abstract class

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

class Id(Exp): #name:str

and the Visitor class is declared as follows:

```
class StaticCheck(Visitor):

    def visitProgram(self,ctx:Program,o):pass

    def visitVarDecl(self,ctx:VarDecl,o): pass

    def visitFuncDecl(self,ctx:FuncDecl,o): pass

    def visitCallStmt(self,ctx:CallStmt,o):pass

    def visitAssign(self,ctx:Assign,o): pass

    def visitIntLit(self,ctx:IntLit,o): pass

    def visitFloatLit(self,ctx,o): pass

    def visitBoolLit(self,ctx,o): pass

    def visitId(self,ctx,o): pass
```

Rewrite the body of the methods in class StaticCheck to infer the type of identifiers and check the following type constraints:

- In an Assign, the type of lhs must be the same as that of rhs, otherwise, the exception TypeMismatchInStatement should be raised together with the Assign
- the type of an Id is inferred from the above constraints in the first usage,
  - if the Id is not in the declarations, exception UndeclaredIdentifier should be raised together with the name of the Id, or
  - If the Id cannot be inferred in the first usage, exception TypeCannotBeInferred should be raised together with the statement
- For static referencing environment, this language applies the scope rules of block-structured programming language where a function is a block. When there is a declaration duplication of a name in a scope, exception Redeclared should be raised together with the second declaration.
- In a call statement, the argument type must be the same as the parameter type. If there is no function declaration in the static referencing environment, exception UndeclaredIdentifier should be raised together with the function call name. If the numbers of parameters and arguments are not the same or at least one argument type is not the same as the type of the corresponding parameter, exception TypeMismatchInStatement should be raise with the call statement. If there is at least one parameter type cannot be resolved, exception TypeCannotBeInferred should be raised together with the call statement.

Your code starts at line 120

For example:

Test	Result
`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("x")],[],[Assign(Id("x"),FloatLit(2))])],[Assign(Id("x"),IntLit(3)),CallStmt("foo",[Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`

Answer: (penalty regime: 0 %)

```
def infer(name,typ,o):
    for env in o:
        if name in env:
            env[name] = typ
            return

class StaticCheck(Visitor):

    def visitProgram(self,ctx:Program,o):
        o = [{}]
        [self.visit(decl,o) for decl in ctx.decl ]
        [self.visit(stmt,o) for stmt in ctx.stmts]

    def visitVarDecl(self,ctx:VarDecl,o):
        if ctx.name in o[0]:
            raise Redeclared(ctx)
        o[0][ctx.name] = 0

    def visitFuncDecl(self,ctx:FuncDecl,o):
        if ctx.name in o[0]:
            raise Redeclared(ctx)
        env = [{}] + o
        [self.visit(decl,env) for decl in ctx.param ]
```

	Test	Expected	Got	
✔	`Program([VarDecl("x"),FuncDecl("foo", [VarDecl("x")],[], [Assign(Id("x"),FloatLit(2))])], [Assign(Id("x"),IntLit(3)),CallStmt("foo", [Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	✔
✔	`Program([VarDecl("x"),FuncDecl("foo",[], [VarDecl("x")], [Assign(Id("x"),FloatLit(2))])], [Assign(Id("x"),IntLit(3)),CallStmt("foo", [Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	✔
✔	`Program([VarDecl("x"),FuncDecl("x", [VarDecl("y")],[],[])],[])`	`Redeclared: FuncDecl(x, [VarDecl("y")],[],[])`	`Redeclared: FuncDecl(x, [VarDecl("y")],[],[])`	✔
✔	`Program([VarDecl("x"),FuncDecl("foo",`	`Undeclared Identifier: x`	`Undeclared Identifier: x`	✔

	`[VarDecl("y")],[],[])],[CallStmt("foo", [IntLit(3)]),CallStmt("foo", [Id("x")]),Assign(Id("x"),FloatLit(0.0))])`	`Assign(Id("x"),FloatLit(0.0))`	`Assign(Id("x"),FloatLit(0.0))`	
✔	`Program([VarDecl("x"),FuncDecl("foo", [VarDecl("y"),VarDecl("z")],[], [Assign(Id("z"),FloatLit(0.0))])], [CallStmt("foo", [IntLit(3),Id("x")]),CallStmt("foo", [Id("x"),FloatLit(0.0)])])`	`Type Mismatch In Statement: CallStmt("foo", [Id("x"),FloatLit(0.0)])`	`Type Mismatch In Statement: CallStmt("foo", [Id("x"),FloatLit(0.0)])`	✔
✔	`Program([VarDecl("x"),FuncDecl("foo", [VarDecl("y"),VarDecl("z")],[],[])], [CallStmt("foo",[IntLit(3),Id("x")])])`	`Type Cannot Be Inferred: CallStmt("foo", [IntLit(3),Id("x")])`	`Type Cannot Be Inferred: CallStmt("foo", [IntLit(3),Id("x")])`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

◄ Type Quiz

Chuyển tới...

Link Video của buổi 5/10/2021 ►