Câu hỏi **1**

Chính xác

Điểm 0,80 của 1,00

Given the grammar of MP as follows:

program: vardecls EOF;

vardecls: vardecl vardecltail;

vardecltail: vardecl vardecltail | ;

vardecl: mptype ids ';' ;

mptype: INTTYPE | FLOATTYPE;

ids: ID ',' ids | ID;

INTTYPE: 'int';

FLOATTYPE: 'float';

ID: [a-z]+ ;

Please copy the following class into your answer and modify the bodies of its methods to count the terminal nodes in the parse tree?

```
class ASTGeneration(MPVisitor):
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return None
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return None
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        return None
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        return None
```

def visitMptype(self,ctx:MPParser.MptypeContext):

return None

def visitIds(self,ctx:MPParser.IdsContext):

return None

For example:

Test	*Result*
`"int a;"`	`4`

Answer: *(penalty regime: 10, 20, ... %)*

```
class ASTGeneration(MPVisitor):
    # program: vardecls EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return self.visit(ctx.vardecls()) + 1
    # vardecls: vardecl vardecltail;
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail())
    # vardecltail: vardecl vardecltail | ;
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        if ctx.getChildCount() == 0: return 0
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail())
    # vardecl: mptype ids ';' ;
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        typ = self.visit(ctx.mptype())
        ids = self.visit(ctx.ids())
        return typ + ids + 1
    # mptype: INTTYPE | FLOATTYPE;
    def visitMptype(self,ctx:MPParser.MptypeContext):
        return 1
    # ids: ID ',' ids | ID;
    def visitIds(self,ctx:MPParser.IdsContext):
        if ctx.getChildCount() == 1: return 1
        return self.visit(ctx.ids()) + 2
```

	Test	Expected	Got	
✓	`"int a;"`	4	4	✓
✓	`"""int a,b;"""`	6	6	✓
✓	`"int a;float b;"`	7	7	✓
✓	`"int a,b;float c;"`	9	9	✓
✓	`"int a,b;float c,d,e;"`	13	13	✓

Passed all tests! ✓

Chính xác

*Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm **0,80/1,00**.*

Câu hỏi 2

Chính xác

Điểm 0,50 của 1,00

Given the grammar of MP as follows:

program: vardecls EOF;

vardecls: vardecl vardecltail;

vardecltail: vardecl vardecltail | ;

vardecl: mptype ids ';' ;

mptype: INTTYPE | FLOATTYPE;

ids: ID ',' ids | ID;

INTTYPE: 'int';

FLOATTYPE: 'float';

ID: [a-z]+ ;

Please copy the following class into your answer and modify the bodies of its methods to count the non-terminal nodes in the parse tree?

```
class ASTGeneration(MPVisitor):
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return None
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return None
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        return None
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        return None
```

def visitMptype(self,ctx:MPParser.MptypeContext):

return None

def visitIds(self,ctx:MPParser.IdsContext):

return None

For example:

Test	*Result*
`"int a;"`	`6`

Answer: *(penalty regime: 10, 20, ... %)*

```
class ASTGeneration(MPVisitor):
    # program: vardecls EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return self.visit(ctx.vardecls()) + 1
    # vardecls: vardecl vardecltail;
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail()) + 1
    # vardecltail: vardecl vardecltail | ;
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        if ctx.getChildCount() == 0: return 1
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail()) + 1
    # vardecl: mptype ids ';' ;
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        typ = self.visit(ctx.mptype())
        ids = self.visit(ctx.ids())
        return typ + ids + 1
    # mptype: INTTYPE | FLOATTYPE;
    def visitMptype(self,ctx:MPParser.MptypeContext):
        return 1
    # ids: ID ',' ids | ID;
    def visitIds(self,ctx:MPParser.IdsContext):
        if ctx.getChildCount() == 1: return 1
        return self.visit(ctx.ids()) + 1
```

	Test	Expected	Got	
✓	`"int a;"`	6	6	✓
✓	`"""int a,b;"""`	7	7	✓
✓	`"int a;float b;"`	10	10	✓
✓	`"int a,b;float c;"`	11	11	✓
✓	`"int a,b;float c,d,e;"`	13	13	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm ***0,50/1,00***.

Câu hỏi ***3***

Chính xác

Điểm 0,20 của 1,00

Given the grammar of MP as follows:

program: vardecls EOF;

vardecls: vardecl vardecltail;

vardecltail: vardecl vardecltail | ;

vardecl: mptype ids ';' ;

mptype: INTTYPE | FLOATTYPE;

ids: ID ',' ids | ID;

INTTYPE: 'int';

FLOATTYPE: 'float';

ID: [a-z]+ ;

and AST classes as follows:

```
class AST(ABC):
    def __eq__(self, other):
        return self.__dict__ == other.__dict__

    @abstractmethod
    def accept(self, v, param):
        return v.visit(self, param)

class Type(AST):
    __metaclass__ = ABCMeta
    pass

class IntType(Type):
    def __str__(self):
        return "IntType"

    def accept(self, v, param):
        return v.visitIntType(self, param)

class FloatType(Type):
    def __str__(self):
        return "FloatType"

    def accept(self, v, param):
        return v.visitFloatType(self, param)



class Program(AST):
    #decl:list(Decl)
    def __init__(self, decl):
        self.decl = decl

    def __str__(self):
        return "Program([" + ','.join(str(i) for i in self.decl) + "])"

    def accept(self, v: Visitor, param):
        return v.visitProgram(self, param)

class Decl(AST):
    __metaclass__ = ABCMeta
    pass

class VarDecl(Decl):
    #variable:Id
    #varType: Type
    def __init__(self, variable, varType):
        self.variable = variable
        self.varType = varType

    def __str__(self):
        return "VarDecl(" + str(self.variable) + "," + str(self.varType) + ")"

    def accept(self, v, param):
        return v.visitVarDecl(self, param)


class Id(AST):
    #name:string
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return "Id(" + self.name + ")"

    def accept(self, v, param):
        return v.visitId(self, param)
```

Please copy the following class into your answer and modify the bodies of its methods to generate the AST of a MP input?

```
class ASTGeneration(MPVisitor):
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return None
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return None
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        return None
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        return None
    def visitMptype(self,ctx:MPParser.MptypeContext):
        return None
    def visitIds(self,ctx:MPParser.IdsContext):
        return None
```

For example:

Test	*Result*
`"int a;"`	`Program([VarDecl(Id(a),IntType)])`

Answer: *(penalty regime: 10, 20, ... %)*

```
class ASTGeneration(MPVisitor):
    # program: vardecls EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return Program(self.visit(ctx.vardecls()))

    # vardecls: vardecl vardecltail;
    def visitVardecls(self,ctx:MPParser.VardeclsContext):
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail())

    # vardecltail: vardecl vardecltail | ;
    def visitVardecltail(self,ctx:MPParser.VardecltailContext):
        if ctx.getChildCount() == 0:
            return []
        return self.visit(ctx.vardecl()) + self.visit(ctx.vardecltail())

    # vardecl: mptype ids ';' ;
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        mptype = self.visit(ctx.mptype())
        ids = self.visit(ctx.ids())
        return [VarDecl(x, mptype) for x in ids]

    # mptype: INTTYPE | FLOATTYPE;
    def visitMptype(self,ctx:MPParser.MptypeContext):
        if ctx.INTTYPE():
            return IntType()
        return FloatType()

    # ids: ID ',' ids | ID;
    def visitIds(self,ctx:MPParser.IdsContext):
        if ctx.ids():
            return [Id(ctx.ID().getText())] + self.visit(ctx.ids())
        return [Id(ctx.ID().getText())]

```

	Test	*Expected*
✓	"int a;"	Program([VarDecl(Id(a),IntType)])
✓	"""int a,b;"""	Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType)])
✓	"int a;float b;"	Program([VarDecl(Id(a),IntType),VarDecl(Id(b),FloatType)])

	Test	*Expected*
✓	`"int a,b;float c;"`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType),VarDecl(Id(c),FloatType)])`
✓	`"int a,b;float c,d,e;"`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType),VarDecl(Id(c),FloatType),VarDecl(Id(d),FloatType),VarDecl(`

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm ***0,20/1,00****.*

Câu hỏi **4**

Chính xác

Điểm 0,60 của 1,00

Given the grammar of MP as follows:

program: exp EOF;

exp: term ASSIGN exp | term;

term: factor COMPARE factor | factor;

factor: factor ANDOR operand | operand;

operand: ID | INTLIT | BOOLIT | '(' exp ')';

INTLIT: [0-9]+ ;

BOOLIT: 'True' | 'False' ;

ANDOR: 'and' | 'or' ;

ASSIGN: '+=' | '-=' | '&=' | '|=' | ':=' ;

COMPARE: '=' | '<>' | '>=' | '<=' | '<' | '>' ;

ID: [a-z]+ ;

and AST classes as follows:

```
class AST(ABC):
    def __eq__(self, other):
        return self.__dict__ == other.__dict__

    @abstractmethod
    def accept(self, v, param):
        return v.visit(self, param)

class Expr(AST):
    __metaclass__ = ABCMeta
    pass

class Binary(Expr):
    #op:string:
    #left:Expr
    #right:Expr
    def __init__(self, op, left, right):
        self.op = op
        self.left = left
        self.right = right

    def __str__(self):
        return "Binary(" + self.op + "," + str(self.left) + "," + str(self.right) + ")"

    def accept(self, v, param):
        return v.visitBinaryOp(self, param)

class Id(Expr):
    #value:string
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return "Id(" + self.value + ")"

    def accept(self, v, param):
        return v.visitId(self, param)

class IntLiteral(Expr):
    #value:int
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return "IntLiteral(" + str(self.value) + ")"

    def accept(self, v, param):
        return v.visitIntLiteral(self, param)

class BooleanLiteral(Expr):
    #value:boolean
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return "BooleanLiteral(" + str(self.value) + ")"

    def accept(self, v, param):
        return v.visitBooleanLiteral(self, param)
```

Please copy the following class into your answer and modify the bodies of its methods to generate the AST of a MP input?

class ASTGeneration(MPVisitor):

def visitProgram(self,ctx:MPParser.ProgramContext):

```
        return None
    def visitExp(self,ctx:MPParser.ExpContext):
        return None
    def visitTerm(self,ctx:MPParser.TermContext):
        return None
    def visitFactor(self,ctx:MPParser.FactorContext):
        return None
    def visitOperand(self,ctx:MPParser.OperandContext):
        return None
```

For example:

Test	*Result*
`"a := b := 4"`	`Program(Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(4))))`

Answer: *(penalty regime: 10, 20, ... %)*

```
class ASTGeneration(MPVisitor):
    # program: exp EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return Program(self.visit(ctx.exp()))
    # exp: term ASSIGN exp | term;
    def visitExp(self,ctx:MPParser.ExpContext):
        if ctx.ASSIGN():
            return Binary(ctx.ASSIGN().getText(), self.visit(ctx.term()), self.visit(ctx.exp()))
        return self.visit(ctx.term())
    # term: factor COMPARE factor | factor;
    def visitTerm(self,ctx:MPParser.TermContext):
        if ctx.COMPARE():
            return Binary(ctx.COMPARE().getText(), self.visit(ctx.factor(0)), self.visit(ctx.factor(1)))
        return self.visit(ctx.factor(0))
    # factor: factor ANDOR operand | operand;
    def visitFactor(self,ctx:MPParser.FactorContext):
        if ctx.ANDOR():
            return Binary(ctx.ANDOR().getText(), self.visit(ctx.factor()), self.visit(ctx.operand()))
        return self.visit(ctx.operand())
    # operand: ID | INTLIT | BOOLIT | '(' exp ')';
    def visitOperand(self,ctx:MPParser.OperandContext):
        if ctx.ID():
            return Id(ctx.ID().getText())
        elif ctx.INTLIT():
            return IntLiteral(int(ctx.INTLIT().getText()))
        elif ctx.BOOLIT():
            return BooleanLiteral(ctx.BOOLIT().getText() == "True")
        return self.visit(ctx.exp())
```

	Test	Expected	Got
✓	"a := b := 4"	Program(Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(4))))	Program(Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(
✓	"""a += b -= a and (b > 3)"""	Program(Binary(+=,Id(a),Binary(-=,Id(b),Binary(and,Id(a),Binary(>,Id(b),IntLiteral(3))))))	Program(Binary(+=,Id(a),Binary(-=,Id(b),Binary(and,Id(a),Binary(>,Id(b),IntLiteral(
✓	"a or b and True"	Program(Binary(and,Binary(or,Id(a),Id(b)),BooleanLiteral(True)))	Program(Binary(and,Binary(or,Id(a),Id(b)),BooleanLi

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm **0,60/1,00**.

Câu hỏi *5*

Chính xác

Điểm 0,40 của 1,00

Given the grammar of MP as follows:

program: vardecl+ EOF;

vardecl: mptype ids ';' ;

mptype: INTTYPE | FLOATTYPE;

ids: ID (',' ID);*

INTTYPE: 'int';

FLOATTYPE: 'float';

ID: [a-z]+ ;

and AST classes as follows:

class Program:#decl:list(VarDecl)

class Type(ABC): pass

class IntType(Type): pass

class FloatType(Type): pass

class VarDecl: #variable:Id; varType: Type

class Id: #name:str

Please copy the following class into your answer and modify the bodies of its methods to generate the AST of a MP input?

```
class ASTGeneration(MPVisitor):
    def visitProgram(self,ctx:MPParser.ProgramContext):
```

return None

def visitVardecl(self,ctx:MPParser.VardeclContext):

return None

def visitMptype(self,ctx:MPParser.MptypeContext):

return None

def visitIds(self,ctx:MPParser.IdsContext):

return None

For example:

Test	*Result*
`"int a;"`	`Program([VarDecl(Id(a),IntType)])`

Answer: *(penalty regime: 10, 20, ... %)*

```python
class ASTGeneration(MPVisitor):
    # program: vardecl+ EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        vardecl = [self.visit(x) for x in ctx.vardecl()]
        return Program([x for sublist in vardecl for x in sublist])
    # vardecl: mptype ids ';' ;
    def visitVardecl(self,ctx:MPParser.VardeclContext):
        return list(map(lambda x: VarDecl(x, self.visit(ctx.mptype())), self.visit(ctx.ids())))
    # mptype: INTTYPE | FLOATTYPE;
    def visitMptype(self,ctx:MPParser.MptypeContext):
        if ctx.INTTYPE():
            return IntType()
        return FloatType()
    # ids: ID (',' ID)*;
    def visitIds(self,ctx:MPParser.IdsContext):
        return [Id(x.getText()) for x in ctx.ID()]
```

	Test	*Expected*
✓	`"int a;"`	`Program([VarDecl(Id(a),IntType)])`
✓	`"""int a,b;"""`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType)])`
✓	`"int a;float b;"`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),FloatType)])`
✓	`"int a,b;float c;"`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType),VarDecl(Id(c),FloatType)])`
✓	`"int a,b;float c,d,e;"`	`Program([VarDecl(Id(a),IntType),VarDecl(Id(b),IntType),VarDecl(Id(c),FloatType),VarDecl(Id(d),FloatType),VarDecl(`

Passed all tests! ✓

Chính xác

*Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm **0,40/1,00**.*

Câu hỏi 6

Chính xác

Điểm 0,50 của 1,00

Given the grammar of MP as follows:

program: exp EOF;

exp: (term ASSIGN) term;*

term: factor COMPARE factor | factor;

factor: operand (ANDOR operand);*

operand: ID | INTLIT | BOOLIT | '(' exp ')';

INTLIT: [0-9]+ ;

BOOLIT: 'True' | 'False' ;

ANDOR: 'and' | 'or' ;

ASSIGN: '+=' | '-=' | '&=' | '|=' | ':=' ;

COMPARE: '=' | '<>' | '>=' | '<=' | '<' | '>' ;

ID: [a-z]+ ;

and AST classes as follows:

class Expr(ABC):

class Binary(Expr): #op:string;left:Expr;right:Expr

class Id(Expr): #value:string

class IntLiteral(Expr): #value:int

class BooleanLiteral(Expr): #value:boolean

Please copy the following class into your answer and modify the bodies of its methods to generate the AST of a MP input?

class ASTGeneration(MPVisitor):

def visitProgram(self,ctx:MPParser.ProgramContext):

return None

def visitExp(self,ctx:MPParser.ExpContext):

return None

def visitTerm(self,ctx:MPParser.TermContext):

return None

def visitFactor(self,ctx:MPParser.FactorContext):

return None

def visitOperand(self,ctx:MPParser.OperandContext):

return None

For example:

Test	*Result*
`"a := b := 4"`	`Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(4)))`

Answer: *(penalty regime: 10, 20, ... %)*

```
class ASTGeneration(MPVisitor):
    # program: exp EOF;
    def visitProgram(self,ctx:MPParser.ProgramContext):
        return self.visit(ctx.exp())
    # exp: (term ASSIGN)* term;
    def visitExp(self,ctx:MPParser.ExpContext):
        terms = [self.visit(x) for x in ctx.term()] # [Id(e), Id(f), Id(g)]
        assigns = [x.getText() for x in ctx.ASSIGN()] # [+=, -=]
        right = terms[-1]
        for i in range(len(assigns)):
            op = assigns[len(assigns)-i-1]
```

```
                left = terms[len(assigns)-i-1]
                right = Binary(op, left, right)
            return right
        # term: factor COMPARE factor | factor;
        def visitTerm(self,ctx:MPParser.TermContext):
            if ctx.COMPARE():
                return Binary(ctx.COMPARE().getText(), self.visit(ctx.factor(0)), self.visit(ctx.factor(1)))
            return self.visit(ctx.factor(0))
        # factor: operand (ANDOR operand)*;
        def visitFactor(self,ctx:MPParser.FactorContext):
            operands = [self.visit(x) for x in ctx.operand()] # [Id(e),Id(f), Id(g)]
            andors = [x.getText() for x in ctx.ANDOR()] # [or, and]
```

	Test	Expected	Got
✓	"a := b := 4"	Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(4)))	Binary(:=,Id(a),Binary(:=,Id(b),IntLiteral(4)))
✓	"""a += b -= a and (b > 3)"""	Binary(+=,Id(a),Binary(-=,Id(b),Binary(and,Id(a),Binary(>,Id(b),IntLiteral(3)))))	Binary(+=,Id(a),Binary(-=,Id(b),Binary(and,Id(a),Binary(>,Id(b),IntLiteral(3)))))
✓	"a or b and True"	Binary(and,Binary(or,Id(a),Id(b)),BooleanLiteral(True))	Binary(and,Binary(or,Id(a),Id(b)),BooleanLiteral(True))

Passed all tests! ✓

Chính xác

*Điểm cho bài nộp này: 1,00/1,00. Tính toán cho lần làm bài trước đó, điểm **0,50/1,00**.*