Câu hỏi **1**
Hoàn thành
Điểm 3,00 của 4,00

Cờ câu hỏi

Nội dung câu hỏi

Given grammar MP written in ANTLR as follows:

program: vardecls ;

vardecls: vardecl vardecls | vardecl ;

vardecl: type ids ;

type: INTTYPE | FLOATTYPE ;

ids: ID (COMMA ID)* ;

And AST corresponding to the above grammar is defined briefly in Python as follows:

```
class AST(ABC)

class Program(AST):
   def __init__(self,decls:List[VarDecl]):

class VarDecl(AST):
  def __init__(self,typ:Type,id:List[String]):

class Type(AST)

class IntType(Type)

class FloatType(Type)
```

Fill in the following blanks to complete the visitor to generate AST from a parse tree? To match with the solution, please:
- use space only when necessary

- write in one line

- Variable names are x, y, z (if there is only one variable, its name must be x; if there are two variables, the first appeared variable must be x and the second must be y, etc)

- if there is an integer literal in an expression, it must be appeared in the left, for example 3+self.visit()

- follow the comment after each blank

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

Program(self.visit(ctx.vardecls()))

```
def visitVardecls(self,ctx:MPParser.VardeclsContext):
    return Trả lời[self.visit(ctx.vardecl())]+self.visit(ctx.vardecls())
 if ctx.getChildCount() == 2 else [self.visit(ctx.vardecl())] # append 2 lists by +, no space
  def visitVardecl(self,ctx:MPParser.VardeclContext):
    return Trả lời VarDecl(self.visit(ctx.type())+self.visit(ctx.ids()))
  def visitType(self,ctx:MPParser.TypeContext):
    return IntType() if ctx.INTTYPE() else FloatType()
  def visitIds(self,ctx:MPParser.IdsContext):
    return Trả lời[x.getText() for x in ctx.ID()]
 #there is 1 for in solution
```

Câu hỏi **2**

Hoàn thành

Điểm 10,00 của 15,00

Cờ câu hỏi

Nội dung câu hỏi

Given grammar MP written in ANTLR as follows:

exp: term COMPARE term | term ; # COMPARE is none-association

term: factor EXPONENT term | factor ;

factor: operand (ANDOR operand)* ; # ANDOR is left-association

operand: INTLIT | BOOLIT | LB exp RB ;

And AST corresponding to the above grammar is defined briefly in Python as follows:

```
class Exp(ABC)
class Binary(Exp):
   def __init__(self,op:String,left:Exp,right:Exp): #dùng getText() để lấy String ứng với op
class IntLit(Exp):
  def __init__(self,val:int):
class BoolLit(Exp):
  def __init__(self,val:boolean):
```

Fill in the following blanks to complete the visitor to generate AST from a parse tree? To match with the solution, please:
- use space only when necessary

- write in one line

- Variable names are x, y, z (if there is only one variable, its name must be x; if there are two variables, the first appeared variable must be x and the second must be y, etc)

- if there is an integer literal in an expression, it must be appeared in the left, for example 3+self.visit()

- follow the comment after each blank

- **x[::-1] returns the reversed list of x, x[1:] returns the list x without the first element, zip(l1,l2) returns the list of pairs from the corresponding l1 and l2.**

- Assume that function toBool(String) converses a String into boolean value.

```
from functools import reduce
class ASTGeneration(MPVisitor):
 def visitExp(self,ctx:MPParser.ExpContext):
    return Trả lời[Binary(ctx.COMPARE().getText(),self.visit(ctx.term(0)),self.visi]
 if ctx.COMPARE() else Trả lời[self.visit(ctx.term(0))]
 #no getChild
 def visitTerm(self,ctx:MPParser.TermContext):
    return Trả lời[Binary(ctx.EXPONENT().getText(),self.visit(ctx.factor()),self.vis]
 if ctx.term() else Trả lời[self.visit(ctx.factor())]
 # no getChild
  def visitFactor(self,ctx:MPParser.FactorContext):
    rl = ctx.operand()[::-1]
    cl = zip(ctx.ANDOR()[::-1],rl[1:])
    dl = zip(ctx.ANDOR(),ctx.operand()[1:])
    return Trả lời[bda x,y:Binary(y[0],x,self.visit(y[1])),dl,self.visit(ctx.operand(0)))]
 # there is 1 reduce in the solution, neither for nor map
  def visitOperand(self,ctx:MPParser.OperandContext):
    return Trả lời[self.visit(ctx.exp())]
 if ctx.getChildCount() == 3 else Trả lời[IntLit(int(ctx.INTLIT().getText()))]
 if ctx.INTLIT() else Trả lời[Boolit(toBool(ctx.BOOLIT().getText()))]
```

Câu hỏi **3**
Hoàn thành
Điểm 2,00 của 4,00

Cờ câu hỏi

Nội dung câu hỏi

Given grammar MP written in ANTLR as follows:

program: vardecl+ EOF;

vardecl: type ids SEMI ;

type: INTTYPE | FLOATTYPE | ARRAY LB INTLIT RB OF type ;

ids: ID COMMA ids | ID ;

Fill in the following blanks to complete the visitor to count the number of LEAF nodes of a parse tree? To match with the solution, please:

- use space only when necessary

- write in one line

- Variable names are x, y, z (if there is only one variable, its name must be x; if there are two variables, the first appeared variable must be x and the second must be y, etc)

- if there is an integer literal in an expression, it must be appeared in the left, for example 3+self.visit()

- follow the comment after each blank

```
class Count(MPVisitor):
  def visitProgram(self,ctx:MPParser.ProgramContext):
     return Trả lời[1+[self.visit(x) for x in ctx.vardecl()]]
 #  there is for in solution
  def visitVardecl(self,ctx:MPParser.VardeclContext):
    return Trả lời[1+self.visit(ctx.type())+self.visit(ctx.ids())]
  def visitType(self,ctx:MPParser.TypeContext):
    return Trả lời[5+self.visit(ctx.type())]
 if ctx.type() else 1
  def visitIds(self,ctx:MPParser.IdsContext):
    return Trả lời[ctx.ID()+ctx.COMMA()+self.visit(ctx.ids())]
 if ctx.ids() else 1
```