Question 1 of 9 1.0

Given the following AST structure in Scala,

```
trait Exp
case class BinExp(op:String,e1:Exp,e2:Exp) extends Exp
case class UnaExp(op:String,e:Exp) extends Exp
case class Lit(i:Integer) extends Exp
case class Id(i:String) extends Exp
```

which is the valid AST of the following expression? (The association and precedence of operators as defined in BKOOL)

23 - (12 + 6) * 4 / 5

- ○ A. BinExp("/", BinExp("-",Lit(23), BinExp("*", BinExp("+", Lit(12), Lit(6)), Lit(4))))
- ○ B. BinExp("-",Lit(23), BinExp("*", BinExp("+", Lit(12),Lit(6)), Lit(4)),BinExp("/",Lit(5)))
- ○ C. BinExp("-",Lit(23),BinExp("*",BinExp("+",Lit(12),Lit(6)),BinExp("/",Lit(4),Lit(5))))
- [✓ D. BinExp("-",Lit(23), BinExp("/", BinExp("*", BinExp("+", Lit(12),Lit(6)),Lit(4)),Lit(5)))]

Reset Selection

Question 2 of 9 1.0 Points

Assume that "- + ! 4" is the valid unary expression and the operators in unary expressions are right-association, i.e., the last operator "!" in the above expression is calculated first and then operator "+" and the first operator "-" is calculated last. What is the AST of the above expression?

- ○ A. UnaExp("+",UnaExp("!",UnaExp("-",Lit(4))))
- [✓ B. UnaExp("-",UnaExp("+",UnaExp("!",Lit(4))))]
- ○ C. UnaExp("!",UnaExp("+",UnaExp("-",Lit(4))))
- ○ D. UnaExp("-",UnaExp("!",UnaExp("+",Lit(4))))

Reset Selection

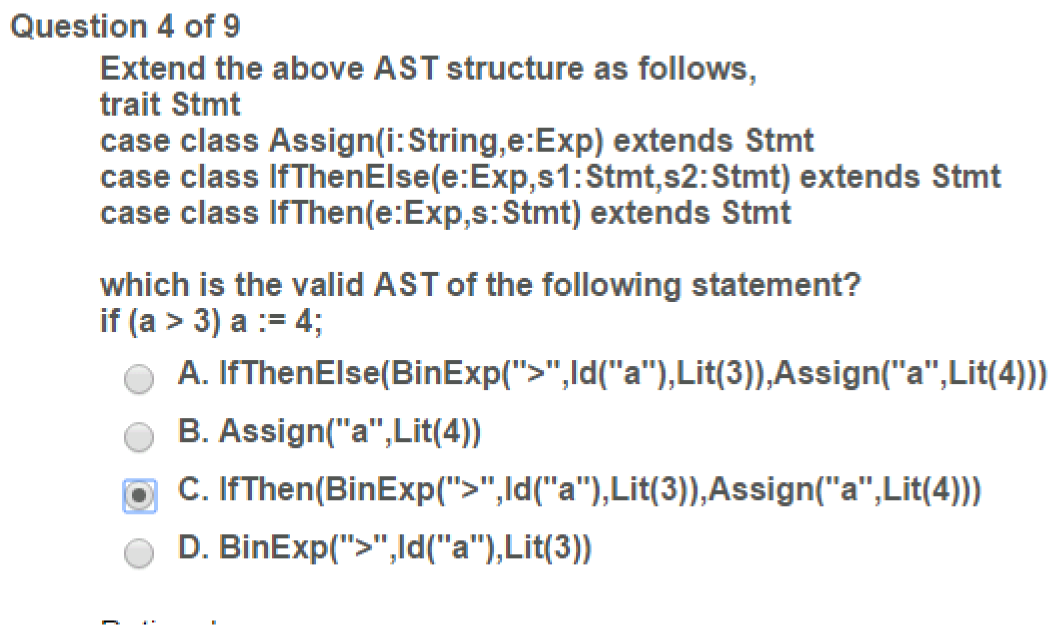
Question 3 of 9 1.0 Points

As "- + ! 4" is a valid unary expression, the grammar written in ANTLR4 for a unary expression is given as follows,

```
fact: ('+'|'-'|'!') fact
    | factor
```

As concerned in the previous question, the operators in unary expressions are right-association. Select the right code for the visitor-subclass to generate AST for a unary expression

- [✓] A. override def visitFact(ctx:ExpParser.FactContext) = if (ctx.fact != null) UnaExp(ctx.getChild(0).getText,visit(ctx.fact)) else visit(ctx.factor)
- [ ] B. override def visitFact(ctx:ExpParser.FactContext) = if (ctx.fact != null) UnaExp(ctx.getChild(0).getText,ctx.fact) else ctx.factor
- [ ] C. override def visitFact(ctx:ExpParser.FactContext) = if (ctx.fact.size > 1) UnaExp(ctx.getChild(0).getText,visit(ctx.fact)) else visit(ctx.factor)
- [✓] D. override def visitFact(ctx:ExpParser.FactContext) = if (ctx.children.size > 1) UnaExp(ctx.getChild(0).getText,visit(ctx.fact)) else visit(ctx.factor)



Question 4 of 9

Extend the above AST structure as follows,
trait Stmt
case class Assign(i:String,e:Exp) extends Stmt
case class IfThenElse(e:Exp,s1:Stmt,s2:Stmt) extends Stmt
case class IfThen(e:Exp,s:Stmt) extends Stmt

which is the valid AST of the following statement?
if (a > 3) a := 4;

- A. IfThenElse(BinExp(">",Id("a"),Lit(3)),Assign("a",Lit(4)))
- B. Assign("a",Lit(4))
- [✓ C. IfThen(BinExp(">",Id("a"),Lit(3)),Assign("a",Lit(4)))]
- D. BinExp(">",Id("a"),Lit(3))

Question 5 of 9

Which is the valid AST of the following statement?

if (x >= 5) then a = 5; else a = 7;

○ A. IfThenElse("x >= 5", "a = 5", "a = 7")

○ B. IfThen(BinExpr(">=",Id(x),Lit(5)),Assign(Id(a),Lit(5)),Assign(Id(a),Lit(7)))

○ C. [BinExpr(">=",Id(x),Lit(5)),Assign(Id(a),Lit(5)),Assign(Id(a),Lit(7))]

● D. IfThenElse(BinExpr(">=",Id(x),Lit(5)),Assign(Id(a),Lit(5)),Assign(Id(a),Lit(7)))

Reset Selection

Question 6 of 9 1.0 Points

Given the rule written in ANTLR4 for recognizing an assignment statement as follows,

```
assign: ident '=' exp ';'  ;
```

Write the corresponding method to generate an AST for an assignment statement.

☐ A. override def visitAssign(ctx:ExParser.AssignContext) = Assign(visit(ctx.ident),visit(ctx.exp))

☐ B. override def visitAssign(ctx:ExParser.AssignContext) = Assign(ctx.ident.asInstanceOf[String],ctx.exp.asInstanceOf[Exp])

☑ C. override def visitAssign(ctx:ExParser.AssignContext) =
Assign(visit(ctx.ident).asInstancOf[String],visit(ctx.exp).asInstanceOf[Exp])

☑ D. override def visitAssign(ctx:ExParser.AssignContext) =
Assign(visit(ctx.getChild(0)).asInstancOf[String],visit(ctx.getChild(2)).asInstanceOf[Exp])

Question 7 of 9 1.0 Points

Given the rule written in ANTLR4 for recognizing an if statement as follows,

```
ifStmt: 'if'  exp  'then'  stmt  ('else'  stmt)?
```

Write the corresponding method to generate an AST for an if statement.

☐ A. override def visitIfStmt(ctx:ExpParser.IfStmtContext) =
```
if (ctx.children.size > 4)
   IfThenElse(visit(ctx.exp).asInstanceOf[Exp],visit(ctx.stmt(0)).asInstanceOf[Exp], visit(ctx.stmt(1)).asInstanceOf[Stmt])
else
  IfThen(visit(ctx.exp).asInstanceOf[Exp],visit(cx.stmt).asInstanceOf[Stmt])
```

☐ B. visitIfStmt(ctx:ExpParser.IfStmtContext) =
```
if (ctx.stmt.size > 1)
  IfThenElse(ctx.exp.asInstanceOf[Exp],ctx.stmt(0).asInstanceOf[Exp], visit(ctx.stmt(1)).asInstanceOf[Stmt])
else
  IfThen(ctx.exp.asInstanceOf[Exp],ctx.stmt(0).asInstanceOf[Stmt])
```

☑ C. visitIfStmt(ctx:ExpParser.IfStmtContext) =
```
if (ctx.stmt.size > 1)
  IfThenElse(visit(ctx.exp).asInstanceOf[Exp],visit(ctx.stmt(0)).asInstanceOf[Exp], visit(ctx.stmt(1)).asInstanceOf[Stmt])
else
  IfThen(visit(ctx.exp).asInstanceOf[Exp],visit(ctx.stmt(0)).asInstanceOf[Stmt])
```

☑ D. visitIfStmt(ctx:ExpParser.IfStmtContext) =
```
if (ctx.children.size > 4)
  IfThenElse(visit(ctx.exp).asInstanceOf[Exp],visit(ctx.stmt(0)).asInstanceOf[Exp], visit(ctx.stmt(1)).asInstanceOf[Stmt])
else
  IfThen(visit(ctx.exp).asInstanceOf[Exp],visit(cx.stmt(0)).asInstanceOf[Stmt])
```

Question 8 of 9 1.0 Points

Extend the above AST structure as follows,

case class CallStmt(name:String,explst:List[Exp]) extends Stmt

Match the statement and the corresponding AST.

A. CallStmt(List(Id('a'),Lit(1),Lit(2))
B. CallStmt('foo',List())
C. CallStmt('foo',List(BinExp('+',Id(a),Lit(1)),Lit(2)))
D. CallStmt('foo',List(Lit(1),Lit(2)))
E. CallStmt('foo',List(Id('a'),Lit(1)))

[B ▾] 1. foo()
[D ▾] 2. foo(1,2)
[E ▾] 3. foo(a,1)
[A ▾] 4. foo(a,1,2)
[C ▾] 5. foo(a+1,2)

Question 9 of 9 2.0 Points

Given the rule written in ANTLR4 for the call statement as follows,

call: ident '(' param? ')' ;
param: exp (',' exp)* ;

Select the valid method for call and param to generate AST for call statement.

[✓] A. override def visitCall(ctx:ExpParser.CallContext) = CallStmt(visit(ctx.ident).asInstanceOf[String],if (ctx.getChild(2) == null) List() else visit(ctx.param).asInstanceOf[List[Exp]])

[✓] B. override def visitParam(ctx:ExpParser.ParamContext) = ctx.exp.asScala.map(visit).toList

[ ] C. override def visitParam(ctx:ExpParser.ParamContext) = if (ctx.exp != null) ctx.exp.asScala.map(visit).toList

[ ] D. override def visitParam(ctx:ExpParser.ParamContext) = ctx.exp.asScala.foldLeft(List())((a,b)=>visit(a)::b)

[ ] E. override def visitCall(ctx:ExpParser.CallContext) = CallStmt(visit(ctx.ident).asInstanceOf[String],visit(ctx.param).asInstanceOf[List[Exp]])

[✓] F. override def visitCall(ctx:ExpParser.CallContext) = CallStmt(visit(ctx.ident).asInstanceOf[String],if (ctx.param != null) visit(ctx.param).asInstanceOf[List[Exp]] else List())

[ ] G. override def visitCall(ctx:ExpParser.CallContext) = CallStmt(visit(ctx.ident),if (ctx.param != null) visit(ctx.param) else List())

[✓] H. override def visitParam(ctx:ExpParser.ParamContext) = ctx.exp.asScala.foldRight(List())((a,b)=>visit(a)::b)