A Correct TinyJ Input File and the Corresponding Parse Tree That is Written to the Output File by a Solution to TinyJ Assignment 1

Input File (a TinyJ program):

```
import java.util.Scanner;

class Simple2 {

 static Scanner input = new Scanner(System.in);

 public static void main(String args[])
 {
   int x = input.nextInt();
   x = x % 3;
   System.out.println(x + 2);
 }
}
```

Output File (the above program's parse tree, in sideways representation):

```
<program>
 <importStmt>
  Reserved Word: import
  Reserved Word: java
  .
  Reserved Word: util
  .
  Reserved Word: Scanner
  ;
  ... node has no more children
 Reserved Word: class
 IDENTIFIER: Simple2
 {
 <dataFieldDecl>
  Reserved Word: static
  <varDecl>
   Reserved Word: Scanner
   IDENTIFIER: input
   =
   Reserved Word: new
   Reserved Word: Scanner
   (
   Reserved Word: System
   .
   Reserved Word: in
   )
   ;
   ... node has no more children
  ... node has no more children
 <mainDecl>
  Reserved Word: public
  Reserved Word: static
  Reserved Word: void
  Reserved Word: main
  (
  Reserved Word: String
  IDENTIFIER: args
  [
  ]
  )
  <compoundStmt>
   {
   <statement>
    <varDecl>
     Reserved Word: int
     <singleVarDecl>
      IDENTIFIER: x
      =
      <expr3>
       <expr2>
        <expr1>
         IDENTIFIER: input
         .
         Reserved Word: nextInt
         (
         )
         ... node has no more children
        ... node has no more children
       ... node has no more children
      ... node has no more children
     ;
     ... node has no more children
    ... node has no more children
   <statement>
    <assignmentOrInvoc>
     IDENTIFIER: x
     =
     <expr3>
      <expr2>
       <expr1>
        IDENTIFIER: x
        ... node has no more children
       %
       <expr1>
        UNSIGNED INTEGER LITERAL: 3
        ... node has no more children
       ... node has no more children
      ... node has no more children
     ;
     ... node has no more children
    ... node has no more children
   <statement>
    <outputStmt>
     Reserved Word: System
     .
     Reserved Word: out
     .
     Reserved Word: println
     (
     <printArgument>
      <expr3>
       <expr2>
        <expr1>
         IDENTIFIER: x
         ... node has no more children
        ... node has no more children
       +
       <expr2>
        <expr1>
         UNSIGNED INTEGER LITERAL: 2
         ... node has no more children
        ... node has no more children
       ... node has no more children
      ... node has no more children
     )
     ;
     ... node has no more children
    ... node has no more children
   }
   ... node has no more children
  ... node has no more children
}
... node has no more children
```

A TinyJ Input File with a Syntax Error, and the Output That is Generated by a Solution to TinyJ Assignment 1

Input File (the error is that `public` should be preceded by a semicolon):

```
import java.util.Scanner;

class Simple2 {

 static Scanner input = new Scanner(System.in)

 public static void main(String args[])
 {
   int x = input.nextInt();
   x = x % 3;
   System.out.println(x + 2);
 }
}
```

Output on the Screen (shows the tokens that are read before the error is detected):

```
1:    import java.util.Scanner;
2:
3:    class Simple2 {
4:
5:     static Scanner input = new Scanner(System.in)
6:
7:     public


ERROR!  Something's wrong--maybe the following token is missing: ;
input.currentChar = ' '
LexicalAnalyzer.currentToken = Reserved Word: public
```

Output File (an incomplete parse tree whose leaves are the tokens that appear in the input file before the syntax error):

```
<program>
 <importStmt>
  Reserved Word: import
  Reserved Word: java
  .
  Reserved Word: util
  .
  Reserved Word: Scanner
  ;
  ... node has no more children
 Reserved Word: class
 IDENTIFIER: Simple2
 {
 <dataFieldDecl>
  Reserved Word: static
  <varDecl>
   Reserved Word: Scanner
   IDENTIFIER: input
   =
   Reserved Word: new
   Reserved Word: Scanner
   (
   Reserved Word: System
   .
   Reserved Word: in
   )
```

An Old Exam Question

A student is debugging his current version of Parser.java for TinyJ Assignment 1. He compiles his file and then runs his program as follows:

```
java –cp .   TJ1asn.TJ  X.java  X.out
```

He also runs the solution that was provided, as follows:

```
java -cp TJ1solclasses:.   TJ1asn.TJ  X.java  X.sol
```

The first difference between the output files `X.out` and `X.sol` is that `X.sol` has a comma on **line 567**, but this is missing in `X.out`. Lines 556 – 568 of `X.sol` and `X.out` are reproduced below with line numbers. (Lines 556 – 566 are the same in both output files.)

Lines 556 – 568 of X.sol [Output produced by **java -cp TJ1solclasses:. TJ1asn.TJ ...**]:

```
556           <expr1>
557            IDENTIFIER: leq
558            <argumentList>
559             (
560             <expr3>
561              <expr2>
562               <expr1>
563                IDENTIFIER: size
564                ... node has no more children
565               ... node has no more children
566              ... node has no more children
567             ,
568             <expr3>
```

Lines 556 – 568 of **X.out** [Output produced by **java –cp . TJ1asn.TJ ...**]:

```
556           <expr1>
557            IDENTIFIER: leq
558            <argumentList>
559             (
560             <expr3>
561              <expr2>
562               <expr1>
563                IDENTIFIER: size
564                ... node has no more children
565               ... node has no more children
566              ... node has no more children
567             <expr3>
568              <expr2>
```

Hint: In reading this output, recall that the indentation levels of consecutive lines are either the same or differ by just 1; thus line 567 has the same indentation as line 559.

Now answer the following two questions. In each case, ***circle the correct choice*.** **[**The answers are given on the next page.]

(i) The output files show there is probably an error in the student's version of the method
(a) `expr1()` (b) `expr2()` (c) `expr3()` (d) `argumentList()` (e) `ifStmt()`
[1 pt.]

(ii) Which one of the following changes might well fix this error?
(a) Insert a missing call of `accept(COMMA)` or `nextToken()` in the student's Parser.java.
(b) Delete a call of `accept(COMMA)` from the student's Parser.java.
(c) Delete a call of `nextToken()` from the student's Parser.java.
(d) Insert a missing call of `expr3()` in the student's Parser.java.
(e) Delete a call of `expr2()` from the student's Parser.java. **[1 pt.]**

Debugging Hints for TinyJ Assignment 1

1. It is a very common mistake to omit a call of `accept()` or `nextToken()`. For ***each*** token on the right side of the EBNF rule that defines a non-terminal <*N*>, there should be a call of `accept()` or `nextToken()` in the body of the corresponding parsing method *N*( ). Another common mistake is to call `nextToken()` when `accept()` should be called. This often produces the following error message: `Internal error in parser: Token discarded without being inspected` Yet another common mistake is to pass a `Symbols` object that represents a ***non***-terminal as an argument to `accept()`— for example, `accept(NTexpr7)` must be a mistake.

2. The sideways parse tree in the output file can be regarded as an ***execution trace*** of your program, and can be useful when debugging your code! Assuming you have produced both *k*.`sol` and *k*.`out` for some *k* (as described on page 4 of the assignment document), each line in *k*.`sol` shows something my solution did. If your program is not working correctly, then the first line in *k*.`sol` that is ***not*** in *k*.`out` shows the first thing my solution did that your program did ***not*** do! (You can easily find that line from the output of `diff -c` [on `euclid` or `venus`] or `fc.exe /n` [on a PC].) When reading the output file for debugging purposes, bear the following in mind:
   A. In a sideways parse tree, the parent of a node appears on the most recent previous line that has lower indentation. (Note that adjacent lines of the tree either have the same indentation or have indentation levels that differ by just 1.) For example, in the Old Exam Question, the parent of the comma on line 567 of `X.sol`, and of `<expr3>` on line 568, is `<argumentList>` on line 558.
   B. Each non-terminal <*N*> in the output file is written when the corresponding parsing method *N*`()` is called. The value of `getCurrentToken()` at that time is shown by the first token in the output file ***after*** <*N*>'s line. <*N*>'s parent in the parse tree shows the caller of *N*`()`. For example, in the Old Exam Question, `<expr3>` on line 560 of `X.sol` was written when `expr3()` was called. The value of `getCurrentToken()` was `IDENT` at the time of the call (as shown by line 563); `expr3()` was called by the method corresponding to the parent of the `<expr3>` node on line 560—i.e., by `argumentList()`, as we see from line 558.
   C. Each token in the output file is written during execution of a call of `accept(T)` or `nextToken()` in some non-terminal's parsing method, at a time when the value of `getCurrentToken()` is `T`; here `T` is the `Symbols` object that represents the token. The parsing method in question is shown by the token's parent in the parse tree. For example, in the Old Exam Question, the comma on line 567 of `X.sol` was written during execution of a call of `accept(COMMA)` or `nextToken()`in a non-terminal's parsing method; the value of `getCurrentToken()` was `COMMA` at the time of the call, and we see from line 558 that the parsing method in question was `argumentList()`.
   D. The `... node has no more children` line that is a child of a node <*N*> of the tree is written ***just before*** the corresponding call of method *N*`()` returns control to its caller. The value of `getCurrentToken()` at that time is shown by the first token in the output file ***after*** the line `... node has no more children` For example, in the Old Exam Question, the line `... node has no more children` on line 565 of `X.sol` is a child of the node `<expr2>` on line 561, and was therefore written just before the corresponding call of `expr2()` returned control to its caller. The caller was `expr3()`, since the parent of `<expr2>` is the node `<expr3>` on line 560. Line 567 of `X.sol` shows that the value of `getCurrentToken()` was `COMMA` when `expr2()` returned control to `expr3()`.

The correct answers to the Old Exam Question are `(i)`—`(d)` and `(ii)`—`(a)`. This follows from hints 2A, 2B, and 2C above.