1. JFlex **[10 Marks]**

Indicate at least 10 different kinds of errors in the following fragment of JFlex code. ("..." just means omitted code). Assume line breaks **are** syntactically important, but spaces **are not**. Assume comments **can** be nested.

lineCount should be initialised to 1.

Should not have "|" for [\ |\t]

Should be * instead of + in [A-Za-z][A-Za-z0-9]+

Number should be 0|[1-9][0-9]*

{ident}, not ident.

{ident} should be moved to after the keywords.

. should be ".".

Should not have return for start of a comment.

Default action should be at end, and return error.

yybegin(NORMAL), not yyend(COMMENT).

Should have lineCount++ for newline in comment.

Should not return token for . in comment.

(12 errors)

2. Bottom Up LALR(1) Parsing [20 Marks]

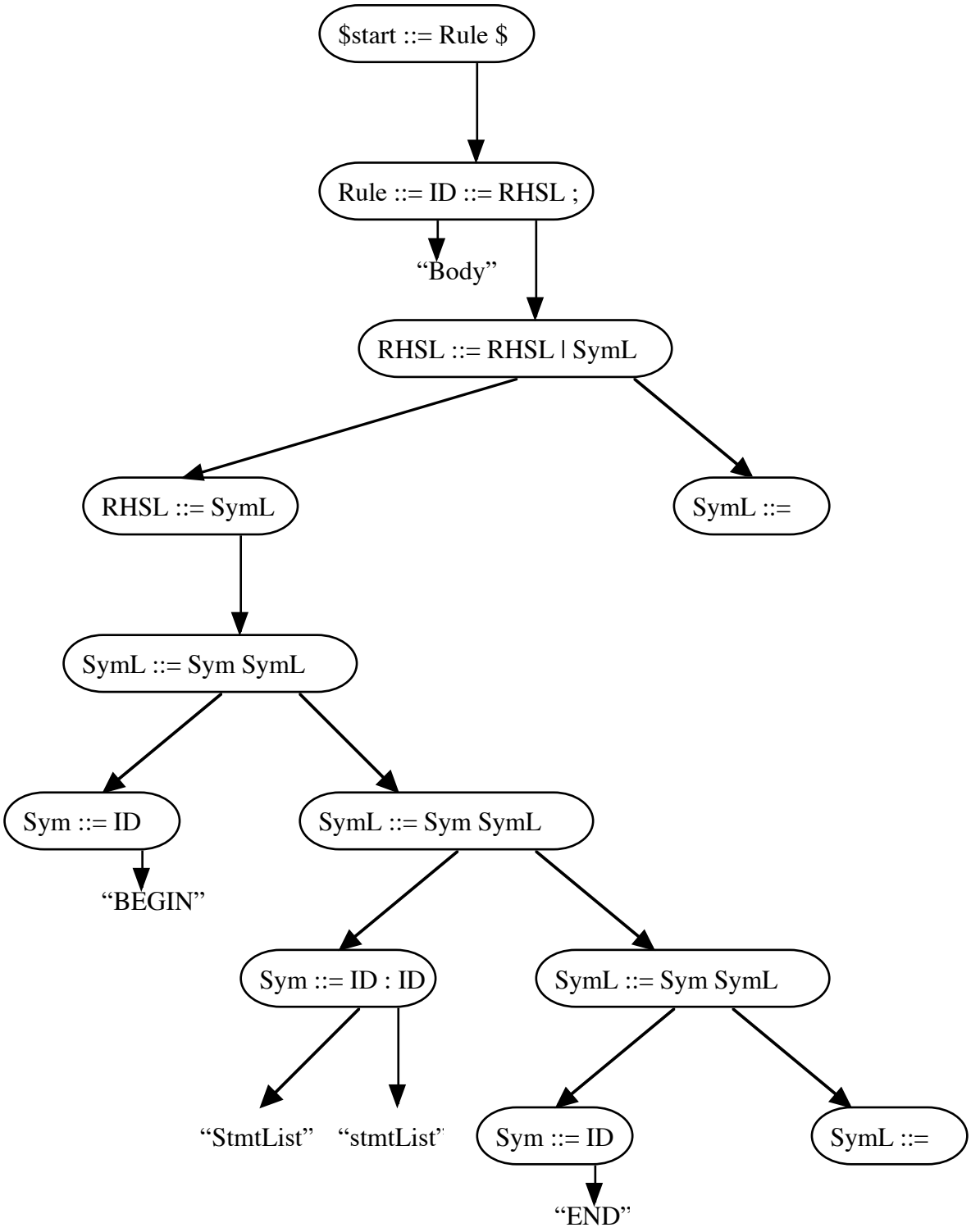
Consider the CUP grammar in the **Appendix For Question 2**. Note that the rules for `RHSList` are left recursive, while the rules for `SymbolList` are right recursive. Also, `SymbolList` can expand to empty.

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
Body ::= BEGIN StmtList:stmtList END | ;
```

Assume "`Body`", "`BEGIN`", "`StmtList`", "`stmtList`", and "`END`" match `IDENT`, and "`::=`", "`:`", "`|`" and "`;`" match `EXPANDSTO`, `COLON`, `OR` and `SEMICOLON`, respectively.

Stack							Token	Reduce	Shift
$0							ID Body		Shift 2
$0	ID 2						::=		Shift 3
$0	ID 2	::= 3					ID BEGIN		Shift 7
$0	ID 2	::= 3	ID 7				ID StmtList	Sym ::= ID	Shift 6
$0	ID 2	::= 3	Sym 6						Shift 7
$0	ID 2	::= 3	Sym 6	ID 7			:		Shift 8
$0	ID 2	::= 3	Sym 6	ID 7	: 8		ID stmtList		Shift 9
$0	ID 2	::= 3	Sym 6	ID 7	: 8	ID 9	ID END	Sym ::= ID : ID	Shift 6
$0	ID 2	::= 3	Sym 6	Sym 6					Shift 7
$0	ID 2	::= 3	Sym 6	Sym 6	ID 7		|	Sym ::= ID	Shift 6
$0	ID 2	::= 3	Sym 6	Sym 6	Sym 6			SymL ::=	Shift 10
$0	ID 2	::= 3	Sym 6	Sym 6	Sym 6	SymL 10		SymL ::= Sym SymL	Shift 10
$0	ID 2	::= 3	Sym 6	Sym 6	SymL 10			SymL ::= Sym SymL	Shift 10
$0	ID 2	::= 3	Sym 6	SymL 10				SymL ::= Sym SymL	Shift 5
$0	ID 2	::= 3	SymL 5					RHSL ::= SymL	Shift 4
$0	ID 2	::= 3	RHSL 4						Shift 12
$0	ID 2	::= 3	RHSL 4	| 12			;	SymL ::=	Shift 13
$0	ID 2	::= 3	RHSL 4	| 12	SymL 13			RHSL ::= RHSL | SymL	Shift 4
$0	ID 2	::= 3	RHSL 4						Shift 11
$0	ID 2	::= 3	RHSL 4	; 11			$	Rule ::= ID ::= RHSL ;	Shift 1
$0	Rule1								Shift 14
$0	Rule1	$14					$	$start ::= Rule $	Shift -1
$0	$strt -1							Accept	


$start ::= Rule $
Rule ::= ID ::= RHSL ;
“Body”
RHSL ::= RHSL | SymL
RHSL ::= SymL
SymL ::=
SymL ::= Sym SymL
Sym ::= ID
SymL ::= Sym SymL
“BEGIN”
Sym ::= ID : ID
SymL ::= Sym SymL
“StmtList”
“stmtList”
Sym ::= ID
SymL ::=
“END”

3. Write a grammar for component invocation statements[20 Marks]

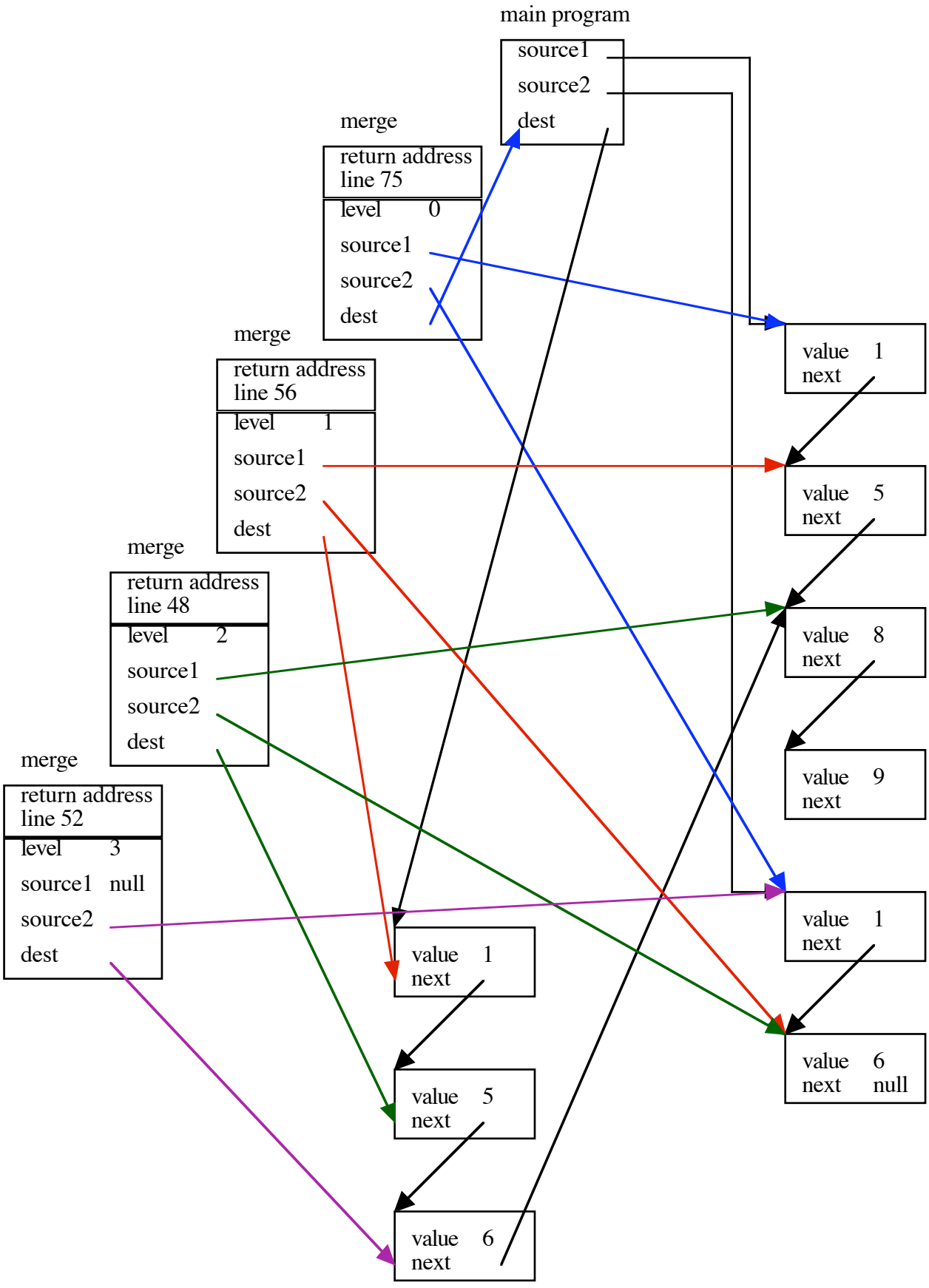
```
InvocationStmt::=
          InputParams
          InvocList
          OutputParams
          SEMICOLON
     ;

InvocList::=
          Invoc
     |
          InvocList DOT Invoc
     ;

Invoc::=
          IDENT
          ValueParams
     ;

InputParams::=
          LEFTBRACE IN PathArrayList RIGHTBRACE
     |
          /* Empty */
     ;

ValueParams::=
          LEFT ExprList RIGHT
     |
          /* Empty */
     ;

OutputParams::=
          LEFTBRACE OUT PathArrayList RIGHTBRACE
     |
          /* Empty */
     ;

PathArrayList::=
          PathArray
     |
          PathArrayList COMMA PathArray
     ;

PathArray::=
          PathNameList
     ;

PathNameList::=
          /* Empty */
     |
          PathNameList PathName
     ;

PathName::=
          IDENT
     |
          IDENT LEFTSQ Expr RIGHTSQ
     |
          IDENT LEFTSQ Expr AT Expr RIGHTSQ
     ;
```

(b) Write a component declaration

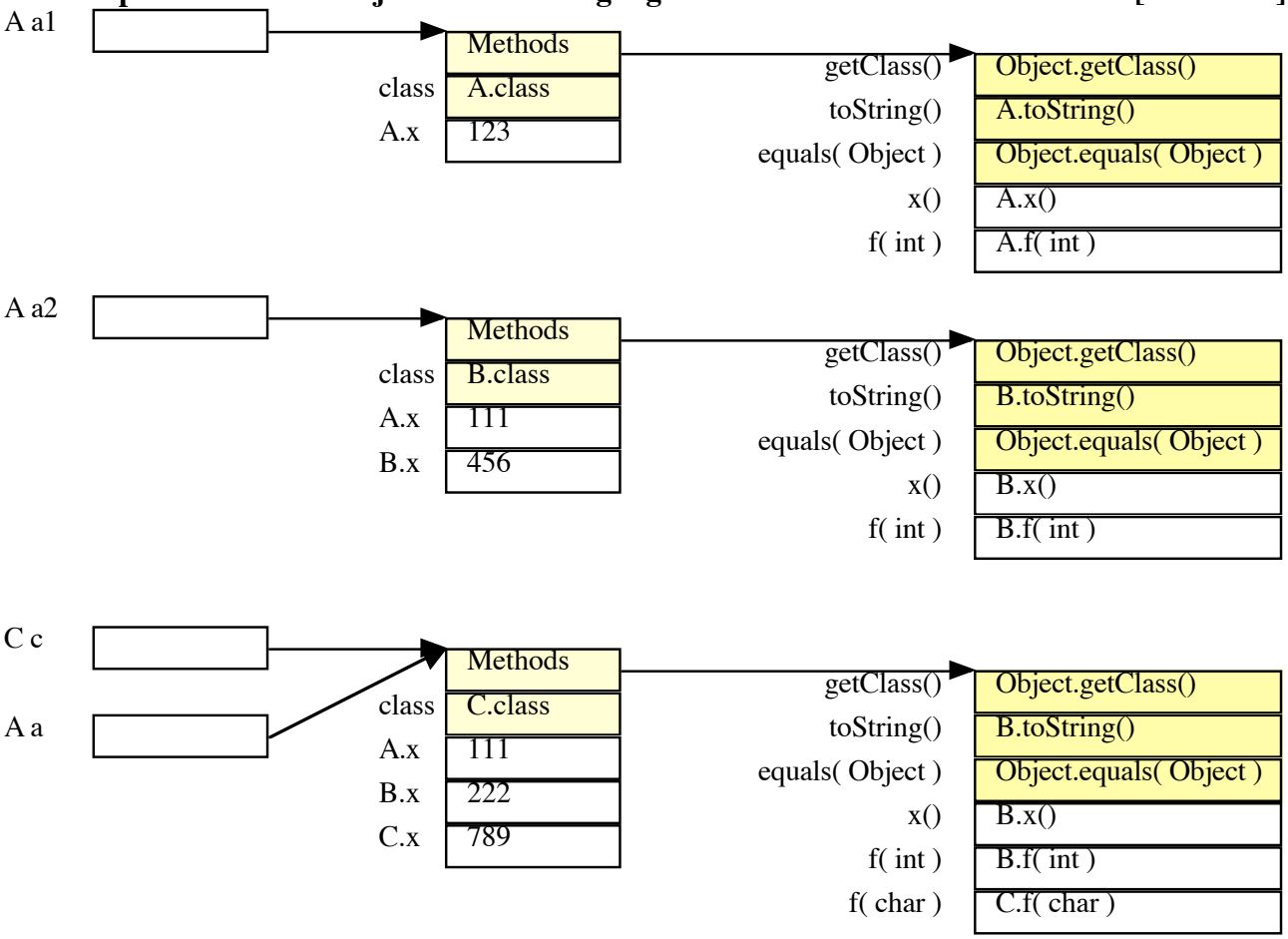
```
component { in opd1, opd2 } compareBit { out less, equal, greater }
     begin
          path notOpd1, notOpd2;
          { in opd1 } not( 1 ) { out notOpd1 };
          { in opd2 } not( 1 ) { out notOpd2 };
          { in notOpd1 opd2 } and( 2 ) { out less };
          { in opd1 opd2 } xor( 2 ).not( 1 ) { out equal };
          { in notOpd2 opd1 } and( 2 ) { out greater };
     end
```

4. Show the run time stack for a B-- program **[17 Marks]**

```
{ 1, 5, 8, 9 }
{ 1, 6 }
{ 1, 5, 6, 8, 9 }
```



5. Implementation of object oriented languages **[18 Marks]**




```
Constructor A( 123 )
Constructor A()
Constructor B( 456 )
Constructor A()
Constructor B()
Constructor C( 789 )
a1.getClass() = class A
a2.getClass() = class B
a.getClass() = class C
c.getClass() = class C
a1 = A.toString()
a2 = B.toString()
a = B.toString()
c = B.toString()
a1.x = 123
a2.x = 111
a.x = 111
c.x = 789
a1.x() = 123
a2.x() = 456
a.x() = 222
c.x() = 222
a1.f( 'A' ) = A.f( 65 )
a.f( 'A' ) = B.f( 65 )
c.f( 'A' ) = C.f( 'A' )
```

6. Code generation **[15 Marks]**

(a) Consider the the B-- language.

What action has to be performed at run time to initialise an object?

At least set up the pointer to the method table.

(2 marks)

What are the $pv, $ra, $fp, $ip, and $nip registers used for, and when are they used?

$pv

Procedure value register, used to contain the address of the method, when performing function entry.

$ra

Return address register, used to save the pc, when when performing function entry/exit.

$fp

Frame pointer register, used to point to the current activation record, when invoking function.

$ip

Instance pointer register, used to point to the current instance, when invoking instance function, or performing initialisation code.

$nip

New Instance pointer register, used to point to the new instance, when performing function entry.

(5 marks)

(b) Consider the following program written in the B-- language.

Indicate the Alpha assembly language likely to be generated for the lines

```
dest^ = dest^ + change;
```

```
{
    ldq $t0, dest_0($fp);
    ldq $t0, ($t0);
    ldq $t1, change_8($fp);
    addq $t0, $t1, $t0;
    ldq $t1, dest_0($fp);
    stq $t0, ($t1);
}
```

```
inc( &b, c );
```

```
{
    lda $sp, -invoc.act2($sp);
    ldiq $t0, main.b_10;
    lda $t0, ($t0);   //   could delete
    stq $t0, invoc.act0($sp);
    ldiq $t0, main.c_18;
    ldq $t0, ($t0);
    stq $t0, invoc.act1($sp);
    mov $zero, $nip;
    ldiq $pv, main.methodCode.inc_0.enter;
    jsr;
    lda $sp, +invoc.act2($sp);
}
```