(Refer to the reference page for this question)

Assume that any errors in the program have been fixed, and all variables are now of type int. Draw the 3AC code that would be generated for stmtA and stmtB. If you need to make any additional assumptions about the program, explain your assumptions. You may use the following 3AC instructions, plus any others you need (explain the functionality of the instruction if you add it, it must obey the rules of 3AC).

3AC reference:

```
x = y op z
x = y
ifz ( x ) goto L
goto L
call p
retrieve x
return x
enter p
get_arg k, x (where k is the arg position)
set_arg k, x (where k is the arg position)
leave
label L
```

```
if (c == 4) {
    int c
    c = 7
}
addOne (addOne( c ));
```

```
t1 := c == 4
ifz t1 Goto Label1
    c := 7
Label1: set_arg 1 c
call addOne
retrieve t2
set_arg 1 t2
call addOne
retrieve t3
```

-0.01 var renaming doesn't line up

QUESTION 4 (2 POINTS)

Check out this simple language of declarations (decls) and uses (statements):

```
program -> LCURLY declList stmtList RCURLY #1
declList -> declList decl                  #2
     | epsilon                             #3
stmtList -> stmtList stmt                  #4
     | epsilon                             #5
decl -> BOOL ID SEMICOLON                  #6
stmt -> ID ASSIGN exp SEMICOLON            #7
exp -> exp EQUALS exp                      #8
     | ID                                  #9
     | BOOLLITERAL                         #10
```

Let the SDT goal be to interpret the program as **the set of identifier names that are used without being declared.** Fill out the SDT rules to meet this goal.

#1: program.trans = declList.trans ∪ stmtList.trans ⇒ stmtList.trans − declList.trans

#2: declList.trans = declList.trans ∪ decl.trans

#3: declList.trans = {}

#4: stmtList.trans → stmtList.trans ∪ stmt.trans

#5: stmtList.trans = {}

#6: decl.trans = {ID.trans}. -0.5 incorrect flag/set

#7: stmt.trans = {ID.trans} ∪ exp.trans

#8: exp1.trans = exp2.trans ∪ exp3.trans

#9: exp.trans = {ID.trans}

#10: exp.trans = {}

QUESTION 5 (2 POINTS)

Assume the following LL(1) parser enhanced with SDT actions:

SDT ACTIONS

#1: tTrans = pop
eTrans = pop
push(tTrans + eTrans)

#2: push(intlit.value + 7)

	intlit	+	eof
E	T E'		
E'		+ T #1 E'	ε
T	#2 intlit		

INPUT TOKENS

intlit 1	plus	intlit 2	intlit 3

SEMANTIC STACK: empty

SYNTACTIC STACK:

E
eof

This parser will eventually encounter an error. Draw the configuration of the semantic stack and syntactic stack at that point

E' cannot read intlit; Error!

T intlit intlit eof

#2 T intlit intlit eof

+ T T #2 intlit intlit eof

#1 #1 #1 #1 #1 E' E' E' E' E' E' E' E' E' E' eof eof eof eof eof eof eof eof eof eof

17

semantic stack