

# COBOL\_LEX

BHAVYA JAIN (12240420) , KRITI ARORA (12240880)

January 2024

## 1 HOW TO RUN THE PROGRAM

- **Using Makefile:**

**Command 1: make**

(Compiles the lex file cobol\_proc.l to give lex.c, compiles cobol\_proc.y and generates y.tab.c and then generates executable a.out using gcc compiler on lex.c and using y.tab.c)

Makefile content:

```
# Variables for input files
LEX_FILE := cobol_proc.l
YACC_FILE := cobol_proc.y

# Define targets
all: lex.c y.tab.c a.out

a.out: y.tab.c lex.c
gcc lex.c y.tab.c

lex.c: $(LEX_FILE)
lex -o lex.c $(LEX_FILE)

y.tab.c: $(YACC_FILE)
yacc -d $(YACC_FILE)

clean:
rm -f a.out lex.c y.tab.c y.tab.h

.PHONY: clean
```

**Command 2:**

./a.out file\_proc

(Runs the executable a.out)

- **Using Commands:**

**Command 1:**

lex cobol\_proc.l

**Command 2:**

yacc -d cobol\_proc.y

**Command 3:**

```
gcc lex.yy.c y.tab.c
```

**Command 4:** `./a.out file_proc`

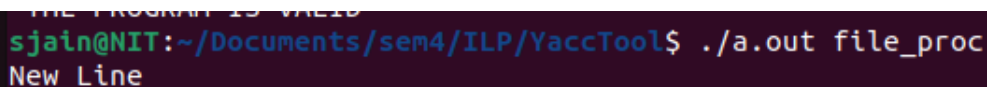
## 2 SAMPLE COBOL CODE

It runs on cobol compiler and our compiler too. For running on cobol compiler, save the file as `file_proc.cob` and run the command

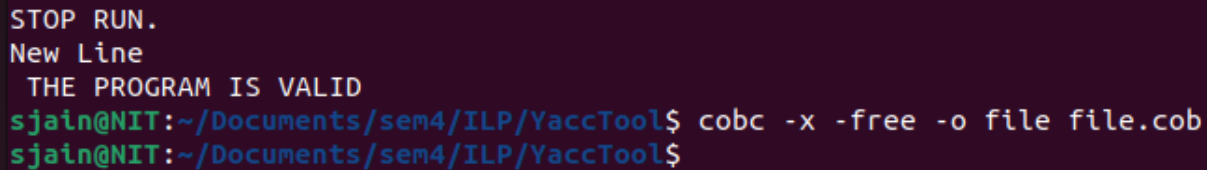
```
cobc -x -free -o file file_proc.cob
```

,Then for running the executable run

```
./file
```



```
THE PROGRAM IS VALID
sjain@NIT:~/Documents/sem4/ILP/YaccTool$ ./a.out file_proc
New Line
```



```
STOP RUN.
New Line
THE PROGRAM IS VALID
sjain@NIT:~/Documents/sem4/ILP/YaccTool$ cobc -x -free -o file file.cob
sjain@NIT:~/Documents/sem4/ILP/YaccTool$
```

We tried to make our syntax analyzer as close to real cobol compiler.

And this code is running on both.

Also we tried to incorporate any type of possible statement combinations. Only for user defined functions, we could not find any sample code that use some user defined function inside other function. Maybe cobol only has intrinsic functions.

There are four divisions in a cobol program. Identification Division , Environment division, Data division and Procedure division.

For given constructs only last two divisions are required but we made grammar rules for all divisions so as to make it close to the original compiler. So, inside Procedure division there are different blocks and for a block it can have simple statements each followed by a period or loops or switch case or if else statements. Also inside a loop or if else there can be again blocks too but statements without period. Also comments are added anywhere in the program after any statement. All keywords are in capital letters.

A tried program is in **file\_proc**.

## 3 SIMPLE STATEMENTS USING OPERATORS

**CODE AND OUTPUT** These are the grammar rules for the simple operation statements in procedure division.

### 3.1 Grammar

```
/* SIMPLE STATEMENTS WHICH INCLUDE PERIOD */
statements
: statement tok_Period comments statements
|
```

```

|{printf("invalid statement line: %d\n", yylineno); exit(1);}
;

statement
: tok_Add operand tok_To identifier_list with
| tok_Add operand tok_To identifier_list with tok_Giving tok_Identifier rounded
| tok_Add identifier_list with tok_Giving tok_Identifier rounded

| tok_Subtract operand tok_From tok_Identifier with rounded
| tok_Subtract operand tok_From tok_Identifier with tok_Giving tok_Identifier rounded

| tok_Multiply operand tok_By operand with
| tok_Multiply operand tok_By operand with tok_Giving tok_Identifier rounded
| tok_Multiply operand tok_By operand

| tok_Divide operand tok_By operand with
| tok_Divide operand tok_By operand with tok_Giving tok_Identifier rounded
| tok_Divide operand tok_Into operand tok_Giving tok_Identifier tok_Remainder tok_Identifier

| tok_Compute arithmetic_stmt

| tok_Move iden_or_int tok_To tok_Identifier
| tok_Move tok_Identifier tok_Thru tok_Identifier tok_To tok_Identifier
| tok_Accept tok_Identifier
| display_statement
| error {printf("invalid statement line: %d\n", yylineno); exit(1);}
;

```

We have statements variable which captures statement after statement.

For a statement there can be Add statements, Subtract statements, Multiply statements, Divide statements, compute statements, Move statements, Display(print) statements. Arithmetic statements can be computed It is like assignment operations with computation. Move statement is only assignment not computation. Assignment operations are like Variable = arithmetic expression. Arithmetic expression is combination of variables, integers and operators. Integers and variables are operands. Also multiplication and division happen first then the addition and subtraction. operand can also be enclosed in brackets.

## 3.2 Sample input

And this is the sample code on which it runs. can run by using

```

./a.out simple_statements

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
ADD WS-NUMBER1, WS-NUMBER2 GIVING WS-SUM.

SUBTRACT WS-NUMBER1 FROM WS-NUMBER2 GIVING WS-TOTAL ROUNDED.

MULTIPLY WS-NUMBER1 BY WS-NUMBER2 GIVING WS-RESULT.
MULTIPLY WS-NUMBER BY 10 GIVING WS-RESULT.
MULTIPLY WS-NUMBER1 BY WS-NUMBER2 GIVING WS-TOTAL ROUNDED.

```

```
DIVIDE WS-NUM1 INTO WS-NUM2 GIVING WS-RESULT REMAINDER WS-REMAINDER.
DIVIDE WS-NUMBER BY 10 GIVING WS-RESULT.
DIVIDE WS-NUMBER BY 5 GIVING WS-RESULT.
```

```
COMPUTE WS-SUM = WS-NUMBER1 + WS-NUMBER2.
COMPUTE WS-QUOTIENT = WS-NUMBER1 / WS-NUMBER2.
COMPUTE WS-PRODUCT = WS-NUMBER1 * WS-NUMBER2.
COMPUTE WS-DIFFERENCE = WS-NUMBER1 - WS-NUMBER2.
COMPUTE WS-TOTAL = 100 + WS-NUMBER * 10.
COMPUTE WS-TOTAL = WS-AMOUNT1 * WS-QUANTITY + WS-AMOUNT2.
COMPUTE WS-TOTAL = WS-AMOUNT1 * WS-QUANTITY + WS-AMOUNT2.
COMPUTE WS-RESULT = (WS-NUMBER1 + WS-NUMBER2) / WS-DIVISOR.
```

```
DISPLAY 'kriti'.
STOP RUN.
```

## 4 CONSTANTS AND IDENTIFIERS DATA TYPES DECLARATION

**CODE AND OUTPUT** Inside Data division we have structs and simple record entries that declare a variable or a constant. Inside a struct there should be record entries. Data type is written after PIC and for constant we give a value to it using the VALUE keyword. we can also declare an array.

And this is the sample code on which it runs. can run by using

```
./a.out data_types
```

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```
02 Struc.
```

### 4.1 Constants

```
01 WS-NUMBER PIC 9(5) VALUE 0.
01 WS-NUM1 PIC 9(5) VALUE 0.
01 WS-NUM2 PIC 9(5) VALUE 0.
01 WS-REMAINDER PIC 9(5) VALUE 0.
01 WS-QUOTIENT PIC 9(5) VALUE 0.
```

### 4.2 Variables

```
01 WS-PRODUCT PIC 9(5) .
01 WS-DIFFERENCE PIC 9(5) .
01 WS-AMOUNT1 PIC 9(5) .
01 WS-QUANTITY PIC 9(5) .
01 WS-AMOUNT2 PIC 9(5) .
PROCEDURE DIVISION.
```

## 5 FOR LOOP (PERFORM)

### CODE AND OUTPUT

Loop is implemented using Perform statement.

PERFORM 10 TIMES. Perform some number of times  
or PERFORM UNTIL COUNTER >5. Perform Until meet some condition  
or PERFORM VARYING WS-I FROM 1 BY 1 UNTIL WS-I >10

Perform Varying some variable from an operand by some operand until you meet some condition. Between Perform and End-Perform we can write our code but in that code we do not take a Period after any statement. Also Inside this we can also include if else or loop or switch case or simple statements.

### 5.1 Grammar

```
perform
: tok_Perform iden_or_int tok_Times no_period_blocks tok_EndPerform end_block comments
| tok_Perform tok_Until condition comments no_period_blocks tok_EndPerform end_block comments
| tok_Perform tok_Varying tok_Identifier tok_From iden_or_int tok_By iden_or_int tok_Until condition no_p
| tok_Perform tok_Identifier params
| error {printf("syntax error in perform statement line: %d\n", yylineno); exit(1);}
;
```

And this is the sample code on which it runs. can run by using

```
./a.out loop
```

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
```

### 5.2 LOOP using PERFORM and UNTIL and multiple simple statements inside

```
PERFORM UNTIL COUNTER > 5
DISPLAY 'Iteration ' COUNTER
ADD 1 TO COUNTER
END-PERFORM.
```

### 5.3 LOOP using TIMES and another loop inside

```
PERFORM 10 TIMES
    PERFORM UNTIL COUNTER > 5
    DISPLAY 'Iteration ' COUNTER
    ADD 1 TO COUNTER
    END-PERFORM.
END-PERFORM.
```

## 5.4 LOOP using VARYING FROM BY UNTIL and IF inside

```
PERFORM VARYING WS-I FROM 1 BY 1 UNTIL WS-I > 10
  IF WS-NUMBER = KRITI - WS-NUMBER
    PERFORM 5 TIMES
    DISPLAY 'Counter value is ' WS-COUNTER
  END-PERFORM
END-IF
END-PERFORM.
STOP RUN.
```

## 6 Switch Case (Evaluate When)

**CODE AND OUTPUT** We implement switch case using evaluate when statements. So it takes an identifier which it has to evaluate. Then it takes cases with when statements and then statements to perform if the case satisfies the when-condition. For default case it uses when other then it ends the switch case by End Evaluate.

### 6.1 Grammar

Not putting all the rules here as they can be seen in the cobol.y

```
evaluate
: tok_Evaluate tok_Identifier switch_cases tok_EndEvaluate tok_Period comments

switch_cases
: when_clauses ;
when_clauses
: when_clause when_clauses
| /*null*/
;

when_clause
: comments tok_When when_condition comments no_period_statements comments;

when_condition
: tok_Integer tok_Thru tok_Integer
| value_list
| tok_Identifier operator constant
| tok_Other

value_list
: constant
| value_list tok_Comma constant
| /*null*/
;
constant
: tok_Integer
| tok_Float
| tok_Character
| tok_BString
| tok_SString
;
```

## 6.2 Sample input

And this is the sample code on which it runs. can run by using

```
./a.out switch_case

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
EVALUATE B
    WHEN 50 THRU 60
        ADD 5 TO B
    WHEN 100
        ADD 5 TO B
        DIVIDE WS-NUMBER1 BY WS-NUMBER2 GIVING WS-TOTAL ROUNDED
    WHEN B <70
        DISPLAY "kriti"
    WHEN OTHER
        DISPLAY 'WS-NUMBER is less than 50'
END-EVALUATE.
STOP RUN.
```

## 7 IF-ELSE

**CODE AND OUTPUT** This is the sample code on which it runs. can run by using

```
./a.out if_else
```

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.

PROCEDURE DIVISION.
```

### 7.1 Grammar

```
/* IF ELSE STATEMENTS */
```

```
if_else
```

```
: tok_If condition comments no_period_statements comments tok_Else comments no_period_statements comments
| tok_If condition comments no_period_statements comments tok_EndIf end_block

| tok_If condition comments no_period_statements comments else_ifs tok_Else no_period_statements tok_EndIf

| tok_If condition no_period_statements tok_EndIf end_block
| tok_If condition no_period_statements tok_Else no_period_statements tok_EndIf end_block
| tok_If condition no_period_statements else_ifs tok_Else no_period_statements tok_EndIf end_block
```

```

| tok_If condition no_period_blocks tok_EndIf end_block
| error {printf("syntax error in if statement line: %d\n", yylineno); exit(1);}
;

else_ifs
: else_if else_ifs
|
;

else_if
: tok_Elseif condition no_period_statements
;

/* CONDITIONS FOR IF OR TERMINATING LOOP*/
condition
: string operator string
| arithmetic_expr operator string
| arithmetic_expr operator arithmetic_expr
| string operator arithmetic_expr
| error {printf("syntax error in condition line: %d\n", yylineno); exit(1);}
;

end_block : tok_Period comments
| comments
| /*null*/

```

## 7.2 Multiple statements inside if block

```

IF WS-NUMBER = 0
    DISPLAY 'WS-NUMBER is positive.'
    DISPLAY 'This is the positive branch.'
    MOVE WS-NUMBER TO WS-TEMP
    COMPUTE WS-TEMP = WS-TEMP * 2
    DISPLAY 'Double of WS-NUMBER is ' WS-TEMP
ELSE
    DISPLAY 'WS-NUMBER is not positive.'
    DISPLAY 'This is the negative branch.'
    MOVE WS-NUMBER TO WS-TEMP
    COMPUTE WS-TEMP = WS-TEMP * 1
    DISPLAY 'Absolute value of WS-NUMBER is ' WS-TEMP
END-IF

```

## 7.3 Perform loop inside if block

```

IF WS-NUMBER > 0
    PERFORM 5 TIMES
    DISPLAY 'Counter value is ' WS-COUNTER
    END-PERFORM
END-IF

IF 0 = WS-NUMBER

```



```

        PERFORM 5 TIMES
        DISPLAY 'Counter value is ' WS-COUNTER
    END-PERFORM
END-IF

```

## 7.4 Else If conditions (also with multiple statements)

```

IF "Bhavya" < "KRITI"
    DISPLAY 'WS-NUMBER is positive.'
    DISPLAY 'This is the positive branch.'
ELSE IF WS-NUMBER < 0
    DISPLAY 'WS-NUMBER is negative.'
    DISPLAY 'This is the negative branch.'
    MOVE WS-NUMBER TO WS-TEMP
    COMPUTE WS-TEMP = WS-TEMP * 1
    DISPLAY 'Absolute value of WS-NUMBER is ' WS-TEMP
ELSE
    DISPLAY 'WS-NUMBER is zero.'
    DISPLAY 'This is the zero branch.'
END-IF

```

## 7.5 If Else inside If Else block

```

IF WS-NUMBER > 0
IF 0 = WS-NUMBER
    PERFORM 5 TIMES
    DISPLAY 'Counter value is ' WS-COUNTER
END-PERFORM
END-IF
END-IF

```

## 7.6 If block without Else

```

IF WS-NUMBER = KRITI - WS-NUMBER
    PERFORM 5 TIMES
    DISPLAY 'Counter value is ' WS-COUNTER
END-PERFORM.
END-IF.

```

## 7.7 Different types of conditions possible

### 7.7.1 Condition Grammar

Here our arithmetic expr is also containing the cases where it is mixture of only integers and operators and also where it is mixture of identifiers, integers and operators. But in actual cobol, it cannot be a mixture of only integers and operators and has to include some identifier(variable) in an arithmetic expr.

```

condition
: string operator string
| arithmetic_expr operator string
| arithmetic_expr operator arithmetic_expr
| string operator arithmetic_expr
;

```

### 7.7.2 Sample code for showing different conditions it is accepting

```
IF WS-NUMBER = KRITI - WS-NUMBER
  IF KRITI - WS-NUMBER = "KRITI"
    IF "WS-NUMBER" = UDF
      PERFORM 5 TIMES
      DISPLAY 'Counter value is ' WS-COUNTER
    END-PERFORM
  END-IF.
END-IF

STOP RUN.
```

## 8 FUNCTIONS(user defined)

**CODE AND OUTPUT** For functions we have assumed that that they are declared and defined using Function keyword inside the procedure division. And they are called using Perform. For declaring and defining a function we give its parameters using USING keyword.

### 8.1 Grammar

```
perform
: tok_Perform iden_or_int tok_Times no_period_blocks tok_EndPerform end_block comments
| tok_Perform tok_Until condition comments no_period_blocks tok_EndPerform end_block comments
| tok_Perform tok_Varying tok_Identifier tok_From iden_or_int tok_By iden_or_int tok_Until condition no_p

| tok_Perform tok_Identifier params

| error {printf("syntax error in perform statement line: %d\n", yylineno); exit(1);}
;

/* function*/
params : tok_Using params_list
| /*null*/
;

params_list : tok_Identifier
| tok_Identifier params_list
;

function : tok_Function tok_Identifier params tok_Period comments statements exit
;

exit : tok_Exit tok_Program tok_Returning tok_Identifier
| /*null*/
;

iden_or_int
: tok_Identifier
| tok_Integer
```

;

And this is the sample code on which it runs. can run by using

./a.out function

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
PERFORM FUNCTION-CALL USING PARAM1 PARAM2
FUNCTION kriti USING bhavya bhoomi.
ADD KRITI TO BHAVYA.
STOP RUN.
```

## 9 COMMENTS

**CODE and OUTPUT** In the sample cobol code we can see that we have included comments many places between statements and between divisions. We have added comments in many places like this. After every tok\_Period we have added comments using end\_block variable so that comment can be added after any statement.

```
pgm: program {printf(" THE PROGRAM IS VALID \n");exit(0);}
program
: comments identification_division comments environment_division  comments data_division comments procedur
| comments

;

comments : comment comments
|/*null*/
;

comment : tok_SingleLineComment
| /*null*/

procedure_division
: tok_Procedure tok_Division tok_Period comments  blocks tok_StopRun
| error {printf("syntax error in procedure division declaration line: %d\n", yylineno); exit(1);}
| tok_Procedure tok_Division tok_Period comments
;
end_block : tok_Period comments
| comments
| /*null*/
no_period_statements
: statement comments
| statement no_period_statements ;
```

### 9.1 Sample commented code

```
IF WS-NUMBER < 50
    *> vawrgqer5bh
```

```

    DISPLAY 'WS-NUMBER is less than 50.'
    *> vawrgqer5bh
ELSE
    *> vawrgqer5bh
    DISPLAY 'WS-NUMBER is greater than or equal to 50.'
    *> vawrgqer5bh
END-IF
    *> vawrgqer5bh

```

## 10 Errors

### 10.1 Error in data type

|   |  |
|---|--|
| <pre> 1 IDENTIFICATION DIVISION. 2 PROGRAM-ID. MYPROGRAM. 3 AUTHOR. kriti arora. 4 DATE-WRITTEN. 98/56/5234. 5 6 ENVIRONMENT DIVISION. 7 8 9 10 DATA DIVISION. 11 WORKING-STORAGE SECTION. 12 13     01 WS-NUMBER PIC 9(5) VALUE 0. 14     01 WS-NUM1 PIC 9(5) VALUE 0. 15     01 WS-NUM2 PIC 9(5) VALUE 0. 16     01 WS-REMAINDER PIC 9(5) . 17     01 WS-QUOTIENT PIC 9(5 18     01 WS-PRODUCT PIC 9(5). 19     01 WS-DIFFERENCE PIC 9(5) VALUE 0. 20 21 PROCEDURE DIVISION. </pre> | <pre> PERIOD New Line Integer Identifier PIC Integer Declaration PERIOD New Line Integer Identifier PIC Integer syntax error in line 17 Invalid data type in line: 17 sjain@NIT:~/Documents/sem4/ILP/YaccTool\$ </pre> |
|---|--|

## 10.2 Error in statement

| data_types ×  | loop × | switch_case × | if_else × | sjain@NIT: ~/Documents/sem4/ILP/Yacc...   |
|---|--------|---------------|-----------|---|
| 1 IDENTIFICATION DIVISION.                                    |        |               |           | SECTION                                   |
| 2 PROGRAM-ID. MYPROGRAM.                                      |        |               |           | PERIOD                                    |
| 3 AUTHOR. kriti arora.  |        |               |           | New Line                                  |
| 4 DATE-WRITTEN. 98/56/5234.                                   |        |               |           | New Line                                  |
| 5   |        |               |           | PROCEDURE                                 |
| 6 ENVIRONMENT DIVISION.                                       |        |               |           | DIVISION                                  |
| 7   |        |               |           | PERIOD                                    |
| 8   |        |               |           | New Line                                  |
| 9   |        |               |           | New Line                                  |
| 10 DATA DIVISION.   |        |               |           | New Line                                  |
| 11 WORKING-STORAGE SECTION.                                   |        |               |           | ADD keyword                               |
| 12  |        |               |           | Identifier                                |
| 13 PROCEDURE DIVISION.  |        |               |           | Comma                                     |
| 14  |        |               |           | Identifier                                |
| 15  |        |               |           | GIVING keyword                            |
| 16 ADD WS-NUMBER1, WS-NUMBER2 GIVING WS-SUM.                  |        |               |           | Identifier                                |
| 17  |        |               |           | PERIOD                                    |
| 18 SUBTRACT FROM WS-NUMBER2 GIVING WS-TOTAL ROUNDED.          |        |               |           | New Line                                  |
| 19  |        |               |           | New Line                                  |
| 20 MULTIPLY WS-NUMBER1 BY WS-NUMBER2 GIVING WS-RESULT.        |        |               |           | SUBTRACT keyword                          |
| 21 MULTIPLY WS-NUMBER BY 10 GIVING WS-RESULT.                 |        |               |           | FROM                                      |
| 22 MULTIPLY WS-NUMBER1 BY WS-NUMBER2 GIVING WS-TOTAL ROUNDED. |        |               |           |   |
| 23  |        |               |           | syntax error in line 18                   |
| 24 STOP RUN.  |        |               |           |   |
| 25  |        |               |           | invalid operand line: 18                  |
| 26  |        |               |           | sjain@NIT:~/Documents/sem4/ILP/YaccTool\$ |
| 27  |        |               |           |   |

### 10.3 Error in Evaluate

```
5
6 ENVIRONMENT DIVISION.
7
8
9
10 DATA DIVISION.
11 WORKING-STORAGE SECTION.
12
13 PROCEDURE DIVISION.
14
15
16 ADD WS-NUMBER1, WS-NUMBER2 GIVING WS-SUM.
17
18 COMPUTE WS-TOTAL = WS-AMOUNT1 * WS-QUANTITY + WS-AMOUNT2.
19 COMPUTE WS-RESULT = (WS-NUMBER1 + WS-NUMBER2) / WS-DIVISOR.
20
21 DISPLAY 'kriti'.
22
23 EVALUATE B
24   *> vawrgqer5bh
25 WHEN 50 THRU
26     ADD 5 TO B
27 WHEN 100
28     ADD TO B
29     DIVIDE WS-NUMBER1 BY WS-NUMBER2 GIVING WS-TOTAL ROUNDED
30     *> vawrgqer5bh
31 WHEN B <70
32     DISPLAY "kriti"
33 WHEN OTHER
34     DISPLAY 'WS-NUMBER is less than 50'
35 END-EVALUATE.
36
```



sjain@NIT: ~/Docu...

sjain@NIT: ~/Documents/sem4/ILP/Yacc...

```
)
/
Identifier
PERIOD
New Line
New Line
DISPLAY
Smallq_Strings
PERIOD
New Line
New Line
Evaluate
Identifier
New Line
Single Line Comment
New Line
When
Integer
Thru
New Line
ADD keyword

syntax error in line 26

invalid statement line: 26
sjain@NIT:~/Documents/sem4/ILP/Yacc...
```

## 10.4 Error in If

```
7
8
9
10 DATA DIVISION.
11 WORKING-STORAGE SECTION.
12
13 PROCEDURE DIVISION.
14
15
16 ADD WS-NUMBER1, WS-NUMBER2 GIVING WS-SUM.
17
18 COMPUTE WS-TOTAL = WS-AMOUNT1 * WS-QUANTITY + WS-AMOUNT2.
19 COMPUTE WS-RESULT = (WS-NUMBER1 + WS-NUMBER2) / WS-DIVISOR.
20
21 DISPLAY 'kriti'.
22
23 IF name = Bhavya'
24 DISPLAY 'WS-NUMBER is positive.'
25 ELSE IF WS-NUMBER < 0.
26 DISPLAY 'WS-NUMBER is negative.'
27 ELSE
28 DISPLAY 'WS-NUMBER is zero.'
29 END-IF.
30
31
32
```

=  
(  
Identifier  
+  
Identifier  
)  
/  
Identifier  
PERIOD  
New Line  
New Line  
DISPLAY  
Smallq\_Strings  
PERIOD  
New Line  
New Line  
If  
Identifier  
=  
Identifier  
Smallq\_Strings  
syntax error in line 23  
invalid statement line: 23  
sjain@NIT:~/Documents/sem4/ILP/YaccTool\$

## 10.5 Error in Perform

```
52 END-IF
53
54 PERFORM UNTIL WS-NUMBER 0
55 DISPLAY 'WS-NUMBER is ' WS-NUMBER
56 IF WS-NUMBER = 50
57     DISPLAY 'Breaking loop at WS-NUMBER = 50'
58 END-IF
59 SUBTRACT 10 FROM WS-NUMBER
60 END-PERFORM
61
```

PERFORM  
Until  
Identifier  
Integer  
syntax error in line 54  
syntax error in arithmetic expression line: 54  
sjain@NIT:~/Documents/sem4/ILP/YaccTool\$

NOTE: ALL GRAMMAR RULES INCLUDED HERE ARE ONLY PART OF ACTUAL CODE. THE ACTUAL CODE HAS TO BE RUN TO RUN ANY FILE. ASSUMPTIONS HAVE BEEN MADE FOR DATA DIVISION AND FUNCTIONS. REST EVERYTHING WORKS LIKE REAL COBOL COMPILER. TRIED AND TESTED.

## REFERENCES

1) IBM COBOL Documentation 2) GITHUB