SSN COLLEGE OF ENGINEERING, KALAVAKKAM – 603 110
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
B.E. Computer Science and Engineering
CS6660 COMPILER DESIGN
UNIT TEST – 2 Answer Key

Qn. No	Part - A	Marks	(KL,COn)
1	How precedence and associativity are handled by yacc compiler? **Ans:** In the definition section of yacc, associativity of a set of operators is given using %left. If there are two sets of operators, The latest order in which the definition is written gets the higher priority	2	(K2,CO3)
2	Whether the following grammar is ambiguous or not? G: S → i E t S | i E t S e S | a E → b **Ans:** It is Ambiguous. Any sentence particular to this grammar generates more than one parse trees.	2	(K2,CO3)
3	Briefly explain handle and handle pruning with suitable example. **Ans:** A substring that matches the right hand side of the production is handle. The process of locating a handle in the string and replace that handle by the production in such a way that it reaches to the previous sentential form in the right most derivation is known as handle pruning.	2	(K1,CO3)
4	Which type of grammar leads to backtracking? How it is handled? Give example. **Ans:** Non-deterministic grammar leads to backtracking. The grammar can be converted into non-deterministic grammar using left factoring method	2	(K2,CO3)
5	What are the possibilities of errors in top down parsing and how they are handled? **Ans:** • All empty entries in the parsing table are errors. • If X is a terminal symbol different from a, this is also an error case	2	(K2,CO3)

Part – B Answer all questions (8+16+16)

6.	What is LL(1) grammar? Whether the following grammar is LL(1) grammar or not? G: S → i E t S S' | a S' → e S' | ε E → b **Ans:**	8	(K3,CO3)

Ambiguous grammar

$S \rightarrow \mathbf{i}\,E\,\mathbf{t}\,S\,S' \mid \mathbf{a}$

$S' \rightarrow \mathbf{e}\,S \mid \varepsilon$

$E \rightarrow \mathbf{b}$

$A \rightarrow \alpha$	FIRST(α)	FOLLOW(A)
$S \rightarrow \mathbf{i}\,E\,\mathbf{t}\,S\,S'$	i	e $
$S \rightarrow \mathbf{a}$	a	e $
$S' \rightarrow \mathbf{e}\,S$	e	e $
$S' \rightarrow \varepsilon$	ε	e $
$E \rightarrow \mathbf{b}$	b	t

Error: duplicate table entry

	a	b	e	i	t	$
S	$S \rightarrow \mathbf{a}$			$S \rightarrow \mathbf{i}\,E\,\mathbf{t}\,S\,S'$		
S'			$S' \rightarrow \varepsilon$ $S' \rightarrow \mathbf{e}\,S$			$S' \rightarrow \varepsilon$
E		$E \rightarrow \mathbf{b}$				

OR

7 How the error during parsing the strings **aab** and **ceadb** will be handled in top down parser for the following grammar? 8 (K3,CO3)

G: S →AbS | e | ε

A → a | cAd

Ans:

Example

S → AbS | e | ε

A → a | cAd

FOLLOW(S)={$}

FOLLOW(A)={b,d}

	a	b	c	d	e	$
S	S → AbS	*sync*	S → AbS	*sync*	S → e	S → ε
A	A → a	*sync*	A → cAd	*sync*	*sync*	*sync*

stack	input	output
$S	aab$	S → AbS
$SbA	aab$	A → a
$Sba	aab$	
$Sbab	$	Error: missing b, inserted
$S	ab$	S → AbS
$SbA	ab$	A → a
$Sba	ab$	
$Sbb	$	
$S	$	S → ε
$	$	accept

stack	input	output
$S	ceadb$	S → AbS
$SbA	ceadb$	A → cAd
$SbdAc	ceadb$	
$SbdA	eadb$	unexpected e (illegal A)
(Remove all input tokens until first b or d, pop A)		
$Sbd	db$	
$Sb	b$	
$S	$	S → ε
$	$	accept

8 a) Write a Yacc desk calculator program that will evaluate Boolean expression. 4 (K3,CO3)

Ans:

Lex Program

```
%{
#include “y.tab.h”
%}
d [0-1]
%%
{d} {return ID;}
[&&] {return *yytext;}
```

```
[||] {return *yytext;}
[!] {return *yytext;}
['\n\} {return *yytext;}
%%
```

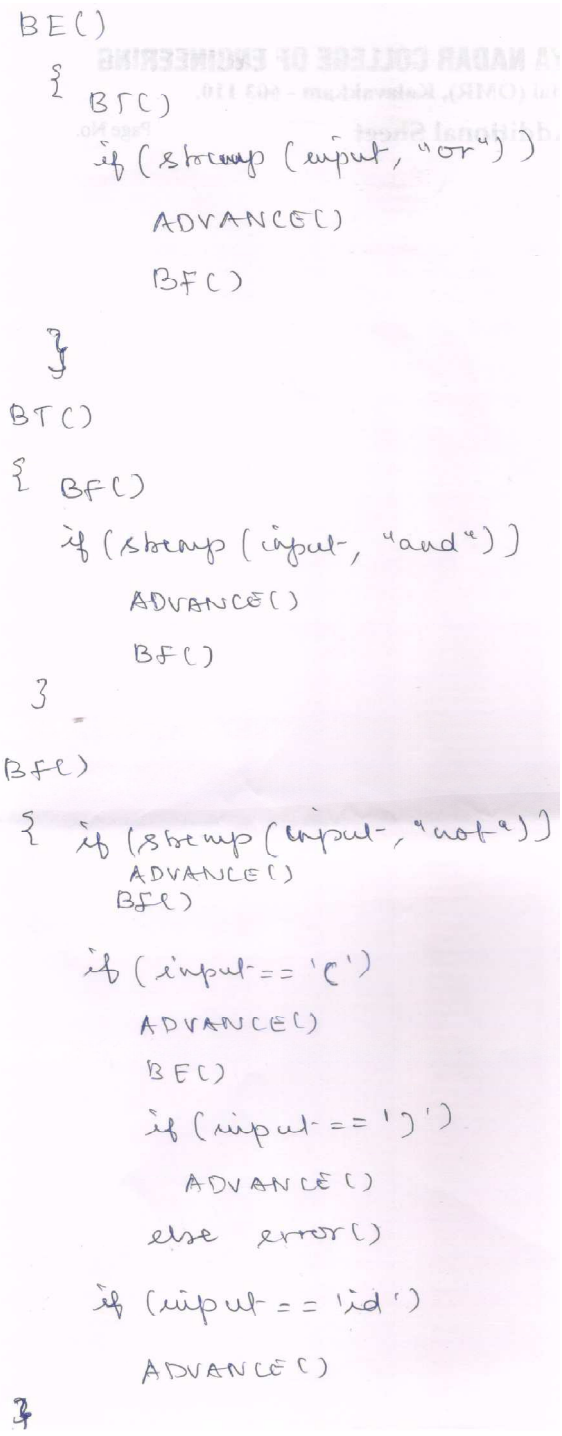
Yacc Program

```
%{
#include "y.tab.h"
%}
%token ID
%right '!'
% left '&&'
%left '||'
%%
P: P E '\n' {printf("Value=%d",$2);}
   |
   ;
E:
  '!' E  {$$=$1;}
   | E '&&' E {$$=$1&&$3;}
   | E '||' E {$$=$1||$3;}
   |
   ;
%%
```

(b) Construct recursive descent parser for the grammar for Boolean expression and parse the string **w=id and id or (id or id)** 12 (K3,CO3)

G: BE → BT or BF| BT
BT → BT and BF | BF
BF → not BF | (BE) | id

Ans:



```
BE()
{
    BT()
    if (strcmp (input, "or"))
        ADVANCE()
        BT()
}
BT()
{
    BF()
    if (strcmp (input, "and"))
        ADVANCE()
        BF()
}
BF()
{
    if (strcmp (input, "not"))
        ADVANCE()
        BF()
    if (input == '(')
        ADVANCE()
        BE()
        if (input == ')')
            ADVANCE()
        else error()
    if (input == 'id')
        ADVANCE()
}
```

OR

9 (a) Construct the predictive parsing table and parse the string w= cad for the following grammar (K3,CO3) 10

G: S → cAd

A → ab|a

Ans:

Grammar given is non-deterministic. Remove the non-determinism by applying left factoring.

G':

S → cAd

A → aA'

A' → b | ε

	FIRST	FOLLOW
S	{c}	{$}
A	{a}	{d}
A'	{b,ε}	{d}

Predictive Parsing Table

	a	b	c	d
S			S → cAd	
A	A → aA'			
A'		A' → b		A' → ε

Parsing the string

Stack	Input Buffer	Output
$S	cad$	S → cAd
$dAc	cad$	
$dA	ad$	A → aA'
$dA'a	ad$	
$dA'	d$	A' → ε
$d	d$	
$	$	

(b) Write the algorithms for FIRST, FOLLOW, construction of predictive parsing table and parsing the string

10 (a) Check whether the following grammar is SLR(1) or not. 10 (K3,CO3)

G: S → L=R

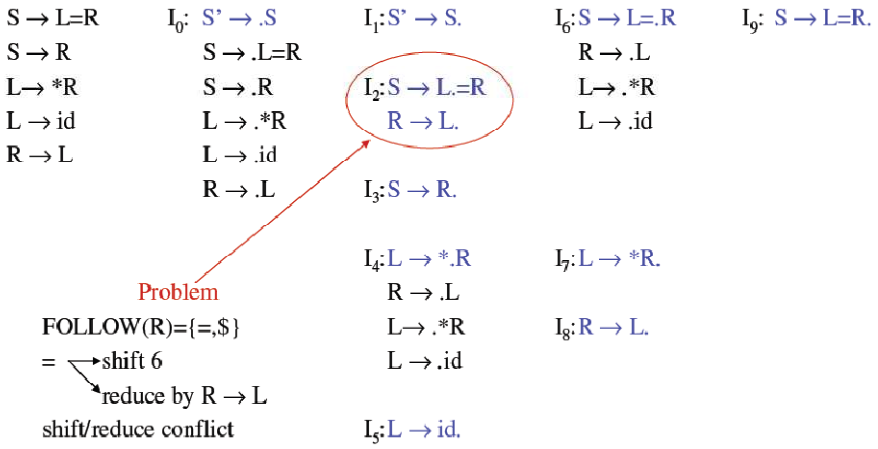
S → R

L → *R

L → id

R → L

Ans:



(b) For the given grammar draw the parse tree of the sentence given below 6 (K2,CO3)

Grammar:

S → if E then S | if E then S else S | while E do S | begin L end | AS

L → L S | S

E → E R E |E A E | id

R → < | <= | > | >= | != | = =

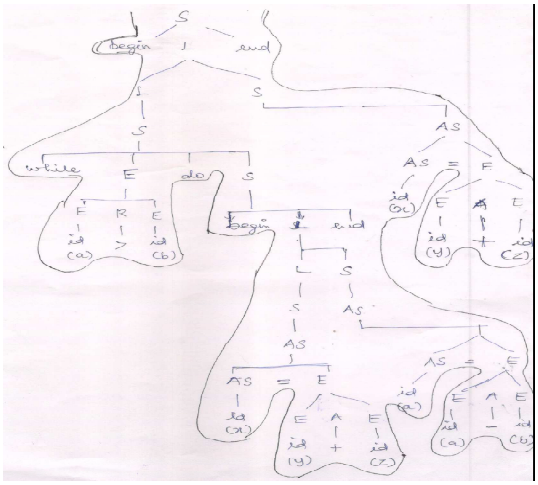
A → + | - | * | / | %

AS → AS=E | id

Sentence:

```
begin
  while a > b do
    begin
      x = y + z
      a = a - b
    end
  x = y - z
end
```

Ans:



OR

11 Find the SLR parsing table for the given grammar and parse the sentence (a,(a,a)) for the following grammar 16 (K3,CO3)

G: S → a | ↑ | (T)

T → T , S | S

G':

S' → S

S → a

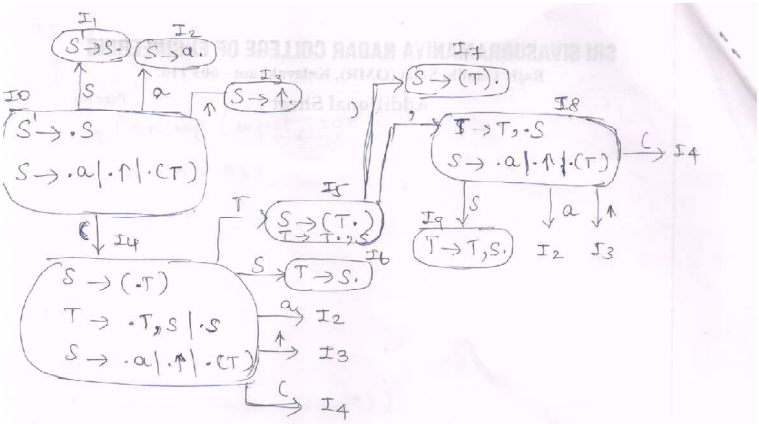
S → ↑

S → (T)

T → T , S

T → S

Ans:



SLR Parsing Table

	Action						Goto	
	a	↑	(	)	,	$	S	T
0	S2	S3	S4				1	
1						Acc		
2				r1	r1	r1		
3				r2	r2	r2		
4	S2	S3	S4				6	5
5			S7	S8				
6				r5	r5			
7				r3	r3	r3		
8	S2	S3	S4	S			9	
9				r4	r4			

① S → a
② S → ↑
③ S → (T)
④ T → T, S
⑤ T → S

Follow(S) = {$,), ,}
Follow(T) = {), ,}