USN | | | | | | | | | | |

Internal Assessment Test 2 – May 2022

Sub:	System Software and Compiler Design					Sub Code:	18CS61	Branch:	CSE
Date:	08/06/2022	Duration:	90 min's	Max Marks:	50	Sem/Sec:	6th / A,B,C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Given the grammar **1.declaration->type var_list** **type->int | float** **var_list ->identifier ,var_list | identifier** Check whether we require left recursion or left factoring 1. Construct FIRST and FOLLOW sets for the nonterminal A 2. Show this grammar is LL(1) 3. Construct parsing table for the grammar **4.** Show the action of the corresponding LL(1) parser given the input string int x,y,z **.Answer:** **Here D= declaration T=type V_L=var_list** **1.To find left recursion elimination is required** Not **2.To find Left factor required** Yes V_L ->id V_L' V_L' ->,V_L | ε So CFG is 1.D->T V_L 2.T -> int | float 3.V_L ->id V_L' 4.V_L' ->,V_L | ε **3.To find unambiguous grammar** **String int a,b** D->T V_L ->int V_L->int id V_L'->int id ,V_L-> int id,id Only one LMD so it is unambiguous grammar. **4.To find First and Fallow set** **1.first(D) = int,float Fallow(D)= $** **2.first(T) =int,float Fallow(T)=id** **3.first(V_L) =id Fallow(V_L)=$**	[10]	CO2	L3

4.first(V_L') = , ε Fallow(V_L')=$

5.To find LL(1) grammar or not

Answer :

V_L' ->,V_L | ε

First (,V_L) ∩ fallow(V_L')= , ∩ $ =ϕ

T ->int | float

First (int) ∩ first(float) = int ∩ float = ϕ

So it is LL(1) grammar

5. Construct Predictive table

CFG is

1.D->T V_L

2c

3.T-> float

4.V_L ->id V_L'

5.V_L' ->,V_L

6. V_L'-> ε

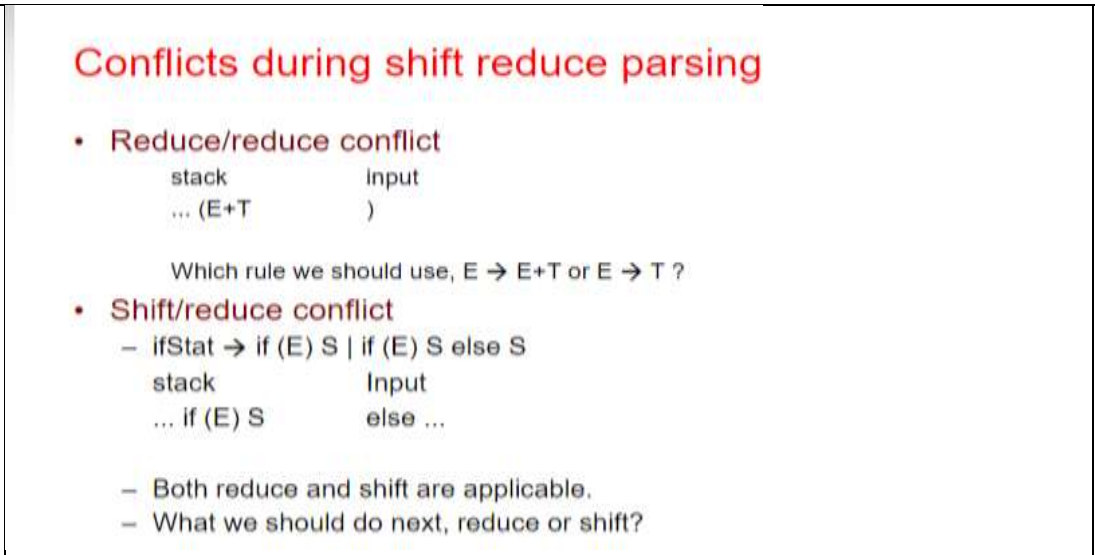
NT	,	int	float	id	$
D		.D->T V_L	.D->T V_L		
T		.T->int	.T->float		
V_l				V_L ->id V_L'	
V_l'	V_L' ->,V_L				V_L'-> ε

6. Tracing of string int x,y,z

Stack	input buffer	action
$D	**int x,y,z $**	.D->T V_L
$V_L T	**int x,y,z $**	.D->T V_L
$V_L int	**int x,y,z $**	.T->int match pop ip++
$V_L	**x,y,z $**	V_L ->id V_L'
$V_L' id	**id,y,z $**	**match pop ip++**
$V_L'	**,y,z $**	V_L' ->,V_L
$V_L,	**,y,z $**	**match pop ip++**
$V_L	**y,z $**	V_L ->id V_L'
$V_L' id	**id,z $**	**match pop ip++**
$V_L,	**,z $**	**match pop ip++**
$V_L'id	**z $**	V_L ->id V_L'
$V_L'id	**id $**	**match pop ip++**
$V_L'	**$**	V_L'-> ε
$	**$**	**accept**

So string is accepted

2 (a)	Explain the conflicts of shift reduce parser with example	[6]	CO2	L2

	 In the above example weather to consider E->E+T to reduce or E->T so this reduce reduce conflict In the above example consider either if (E) S or if (E) S else S So this is shift reduce conflict			
(b)	What is meant to handle processing(handle pruning)?how it helps shift-reduce parsing? list the actions of shift-reduce parser. Answer : **handle pruning** HANDLE PRUNING is the general approach used in shift-and-reduce parsing. A Handle is a substring that matches the body of production. Handle reduction is a step in the reverse of the rightmost derivation. **how it helps shift-reduce parsing** A rightmost derivation in reverse can be obtained by handle pruning **list the actions of the shift-reduce parser**. • **Shift:** This involves moving symbols from the input buffer onto the stack. • **Reduce:** If the handle appears on top of the stack then, its reduction by using the appropriate production rule is done i.e. RHS of a production rule is popped out of a stack and the LHS of a production rule is pushed onto the stack. • **Accept:** If only the start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accepted action is obtained, it is means successful parsing is done. • **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action	[4]	CO2	L1
3.	Consider the grammar S –> (L) | a L –> L , S | S Perform Shift Reduce parsing for input string (a, (a, a)) Answer : RMD : S ->(L) ->(L,S)->(L,(L))-> (l,(L,S))-> (L,(L,a))-> (L,(S,a))-> (L,(a,a))-> (S,(a,a))-> (a,(a,a)) Reverse RMD:	[10]	CO2	L3

(a,(a,a))-> (S,(a,a))-> (L,(a,a))-> (L,(S,a))-> (L,(L,a))-> (L,(L,S))-> (L,(L))-> (L,S)-> (L)->S

RSF Table :

RSF	Handle	Reduction Production
(a,(a,a))	a	S->a
(S,(a,a))	S	L->S
(L,(a,a))	a	S->a
(L,(S,a))	S	L->S
(L,(L,a))	a	S->a
(L,(L,S))	L,S	L->L,S
(L,(L))	(L)	S->(L)
(L,S)	L,S	L->L,S
(L)	(L)	S->(L)
S		

Tracing of string (a,(a,a)

Stack	Handle	action
$	(a,(a,a))$	shift
$(	a,(a,a))$	shift
$(a	,(a,a))$	S->a reduce
$(S	,(a,a))$	L->S reduce
$(L	,(a,a))$	shift
$(L,	(a,a))$	shift
$(L,(	a,a))$	shift
$(L,(a	,a))$	S->a reduce
$(L,(S	,a))$	L->S reduce
$(L,(L,	a))$	shift
$(L,(L,a	))$	shift
$(L,(L,a	))$	S->a reduce
$(L,(L,S	))$	L->L,S reduce
$(L,(L	))$	shift
$(L,(L)	)$	S->(L) reduce
$(L,S	)$	L->L,S reduce
$(L	)$	shift
$(L)	$	S->(L) reduce
$S	$	accept

Finally string is accepted

4. **Write the YACC program to evaluate the arithmetic expression including (*,/,+.-) operation..** [10] CO3 L3

Answer :

LEX Program

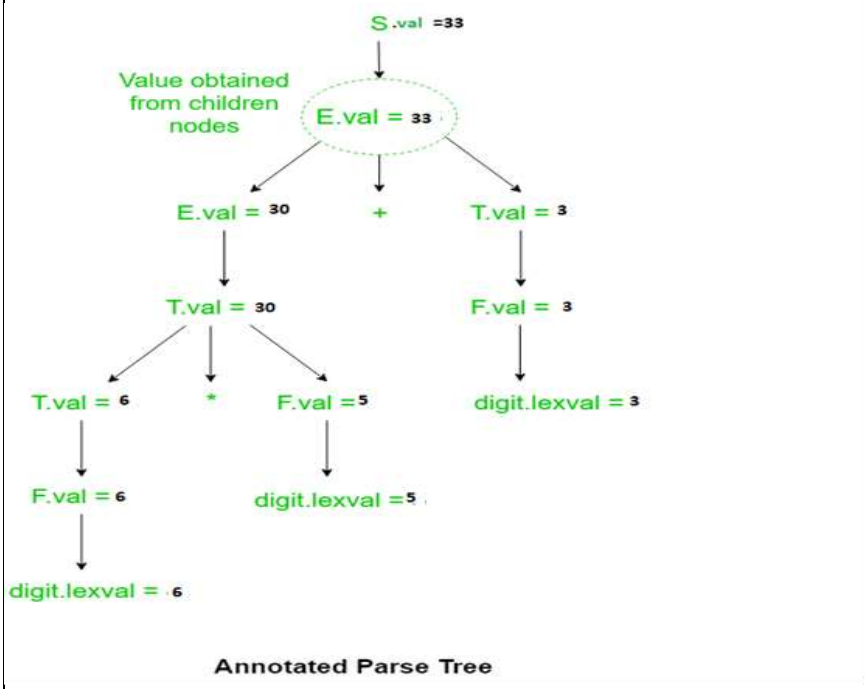
```
%{
    /* Definition section*/
    #include "y.tab.h"
    extern yylval;
}%

%%
[0-9]+     {
                yylval = atoi(yytext);
```

```
              return NUMBER;
            }

[a-zA-Z]+    { return ID; }
[ \t]+         ;  /*For skipping whitespaces*/

\n             { return 0; }
.             { return yytext[0]; }

%%

YACC Program
%{
    /* Definition section */
  #include <stdio.h>
%}

%token NUMBER ID
// setting the precedence
// and associativity of operators
%left '+' '-'
%left '*' '/'

/* Rule Section */
%%
S : E         {
                printf("Result = %d\n", $$);
                return 0;
            }

E :  E '+' T { $$ = $1 + $3; }
    | E '-' T { $$ = $1 - $3; }
    | E '*' T { $$ = $1 * $3; }
    | E '/' T { $$ = $1 / $3; }
    | '-' NUMBER { $$ = -$2; }
    | '-' ID { $$ = -$2; }
    | '(' E ')' { $$ = $2; }
    | NUMBER { $$ = $1; }
    | ID { $$ = $1; };
% %

int main() {
    printf("Enter the expression\n");
    yyparse();
}

/* For printing error messages */
int yyerror(char* s) {
    printf("\nExpression is invalid\n");
}
```

5	Obtain SDD for Bottom up parser and construct annotated parser tree for input string 6 * 5 +3 for the grammar below · S-> EN · E->E+T | E-T | T · T->T*F | T/F | F · F-> (E) | digit · N -> ; Answer : **SDD for arithmetic expression include (+* / - ())**	[10]	CO2	L3

Production	Semantic rules
S-> EN	S.val=E.val
E->E+T	E.val=E_1.val + T.val
E-> E-T	E.val=E_1.val - T.val
E-> T	E.val= T.val
T->T*F	T.val=T_1.val * F.val
T->T/F	T.val=T_1.val / F.val
T-> F	T.val=F.val
F-> (E)	F.val=E.val
F-> digit	F.val=digit.lexval
N -> ;	

Construct annotated parser tree for input string 6 * 5 +3n

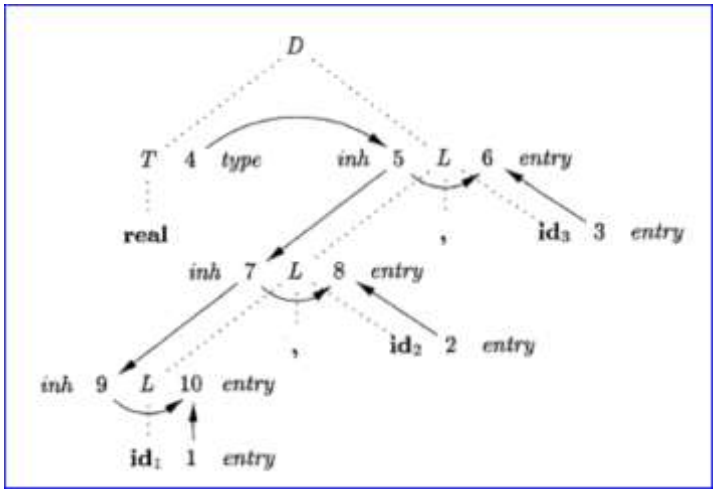


Annotated Parse Tree

6.	Write Syntax Directed Definition and Construct dependency graph for the declaration float id1,id2,id3 1) D -> T L 2) T -> int 3) T -> float 4) L -> L_1 , id 5) L -> id Answer : **The SDD for the above grammar can be written as follow**	[10]	CO2	L3

Production	Semantic Rules
1) D -> T L	L.inh = T.type
2) T ->int	T. type = integer
3) T -> float	T.type = float
4) L ->L_1 , id	L_1.inh=L.inh addType(id.entry, L.inh)
5) L -> id	add Type(id entry, L.inh)

Dependency graph for input **float x,y,z**



Course Outcomes		Modules covered	PO1	PO2	PO3	PO4	PO5	PO6		PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Explain system software	**1**	3	3	2	2	-	-	-	-	-	-	-	-	-	-	-	-	CO1
CO2	Design and develop lexical analyzers, parsers and code generators	**2,3,4,5**	3	3	3	3	3	2	-	-	-	-	-	-	-	-	2	-	CO2
CO3	Utilize lex and yacc tools for implementing different concepts of system software	**4**	3	3	3	3	3	2	-	-	-	-	-	-	-	-	3	-	CO3

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.

L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				