Register Number

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Computer Science and Engineering

Continuous Assessment Test – I

Answer key

Degree & Branch	**BE (CSE)**				**Semester**	VI
Subject Code & Name	**UCS1602 – Compiler Design**				**Regulation:**	**2018**
Academic Year	2021-2022	**Batch**	2019-2023	**Date**	**31-03-2022**	**FN**
Time: 90 Minutes **8.30 – 10.00 am**	**Answer All Questions**				**Maximum: 50 Marks**	

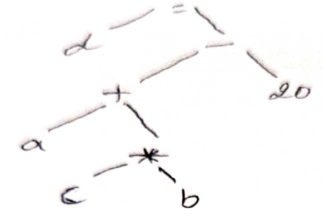
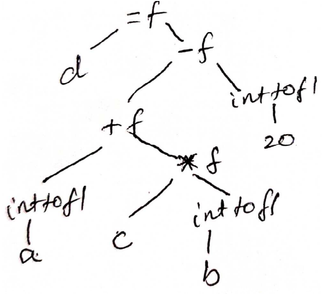
Part – A (6×2 = 12 Marks)

<KL3>	1.Estimate the correct number of LOC(lines of code) after applying appropriate code optimization techniques for the given three address code.
	t1=t1*30
	t2=t1+0
	t3=t2+c
	t4=t3
	
	t1=t1*30
	t4=t1+c
	LOC = 2
		<CO1>

<KL1>	2. What is the correct sequence of processes involved in program execution? Preprocessor -> compiler -> Assembler → loader/linker → target code	<CO1>
<KL2>	3. Illustrate the use of the global variables yytext, yyleng and yylval used in LEX with examples. yytext → lexeme value yyleng → length of the lexeme yylval → used to pass the semantic value associated with a token from the lexer to the parser	<CO1>
<KL3>	4. Consider a language L generates the following: It starts with $ followed by float values with both whole number and fractional part. eg. $1234.56 It can start with $ followed by integer values. e.g $56 It can start with $ followed by float values with only fractional part e.g $.45 Construct a regular expression to generate L. $(digit*)(.digit+)?	<CO1>
<KL3>	5. Consider a regular expression (a/b)*abb(a/b)*. Let the follow position table be Node followpos 1 1,2,3 2 1,2,3 3 4	<CO1>

	4 5 5 6,7,8 6 6,7,8 7 6,7,8 8 – Apply DFA construction algorithm to find the next state for the input symbol ‘a’ from a state {1,2,3,5}. {1,2,3,4}	
<KL2>	6. Explain ambiguous grammar with an example. A CFG is said to ambiguous if there exists more than one derivation tree for the given input string E→E+E |E-E|E*E|E/E|id e.g input string id+id*id	<CO2>

Part – B (3×6 = 18 Marks)

<KL2>	7. Explain the phases of a compiler. Illustrate the output of each phase for the following code segment.
	int a,b;
	float c,d;
	d=a+c*b-20;
	Lexical analysis
	KW ID SP ID SP
	KW ID SP ID SP
	ID ASSIGN ID ARITH+ ID ARITH* ID ARITH- NUM SP
	Syntax analysis
	
	
	Semantic analysis
	
	
	Intermediate code generation
	t1= inttofl (b)
	t2=c*t1
	t3=inttofl (a)
	t4= a+t2
	t5=inttofl (20)
	t6=t4-t5
	d=t6
		<CO1>

<table>
<tr><td></td><td>Code optimization
t1= inttofl (b)
t2=c*t1
t3=inttofl (a)
t4= a+t2
d= t4-20.0

Code generation
MOVF R1, c s
MULF R1, t1
MOVF t2,R1
MOVF R2,t3
ADDF R2, t2
MOVF t4,R2
MOVF R3, t4
SUBF R3, #20.0
MOVF d,R3

Symbol table
<b>a int</b>
<b>b int</b>
<b>c float</b>
<b>d float</b></td><td></td></tr>
<tr><td><KL2></td><td>8. Write a LEX specification to recognize the identifier, numeric constants including fraction and exponentiation, keywords and operators
% {
#include <stdio.h>
    %
}
  digit [0-9]

    / rule section % %

auto|double|int|struct|break|else|long|switch|case|enum|register|typedef|char|extern|return|union|continue|for|signed|void|do|if|static | while |default |goto| sizeof|volatile|const|float |short
{ECHO; printf("\nKEYWORD\n");}

 [{};,()]   {ECHO; printf("\tSEPERATOR\t");}

 [+-/=*%]   {ECHO; printf("\tOPERATOR\t");}

[+ -]?{digit}+(\.{digit}+)? (e[+ -]?{digit}+)?   { ECHO; printf("\tconstant\t");}

 ([a-zA-Z][0-9])+|[a-zA-Z]* {ECHO; printf("\tIdentifier\t");}

/*No action*/
.|\n ;
% %

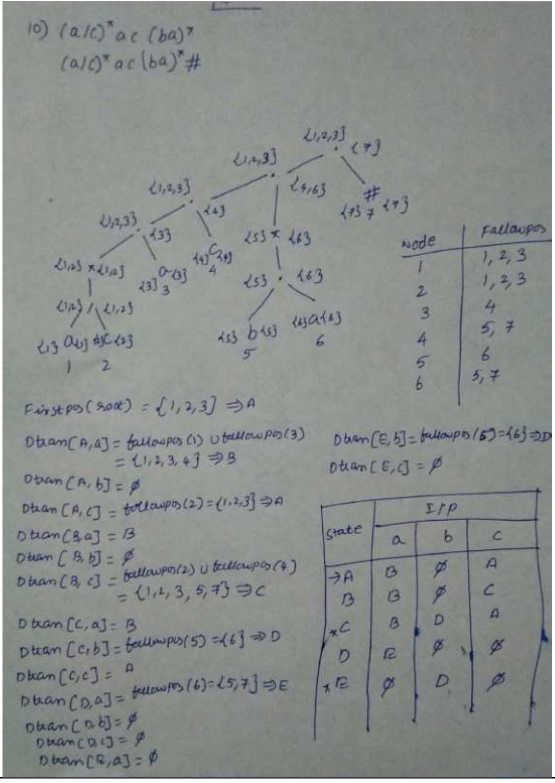
    main()
{
    yylex();
}</td><td><CO1></td></tr>
</table>

<KL2>	9. Show that the grammar G1 is not suitable for implementing top down parser. Rewrite the grammar to overcome this problem. G1: A → AB1 | B0 | 1 B → B1 | A0 | 0 Eliminating immediate left recursion for A A → B0A’ / 1A’ A’ → B1A’ / ∈ Substituting the productions of A in B → A0 A → B0A’ / 1A’ A’ → B1A’ / ∈ B → B1 / B0A’0 /1A’0 / 0 Eliminating left recursion from the productions of B A → B0A’ / 1A’ A’ → B1A’ / ∈ B → 1A’0B’ / 0B’ B’ → 1B’ / 0A’0B’ / ∈	<CO2>

Part – C (2×10 = 20 Marks)

<KL3>

10. Apply direct method to construct DFA for the regular expression ((a|c)*)ac(ba)*



10) (a|c)* ac (ba)*
(a|c)* ac (ba)* #

Node	Followpos
1	1, 2, 3
2	1, 2, 3
3	4
4	5, 7
5	6
6	5, 7

Firstpos(root) = {1, 2, 3} ⇒ A

Dtran[A, a] = followpos(1) ∪ followpos(3)
= {1, 2, 3, 4} ⇒ B

Dtran[A, b] = ∅

Dtran[A, c] = followpos(2) = {1, 2, 3} ⇒ A

Dtran[B, a] = B

Dtran[B, b] = ∅

Dtran[B, c] = followpos(2) ∪ followpos(4)
= {1, 2, 3, 5, 7} ⇒ C

Dtran[C, a] = B

Dtran[C, b] = followpos(5) = {6} ⇒ D

Dtran[C, c] = A

Dtran[D, a] = followpos(6) = {5, 7} ⇒ E

Dtran[D, b] = ∅

Dtran[D, c] = ∅

Dtran[E, a] = ∅

Dtran[E, b] = followpos(5) = {6} ⇒ D

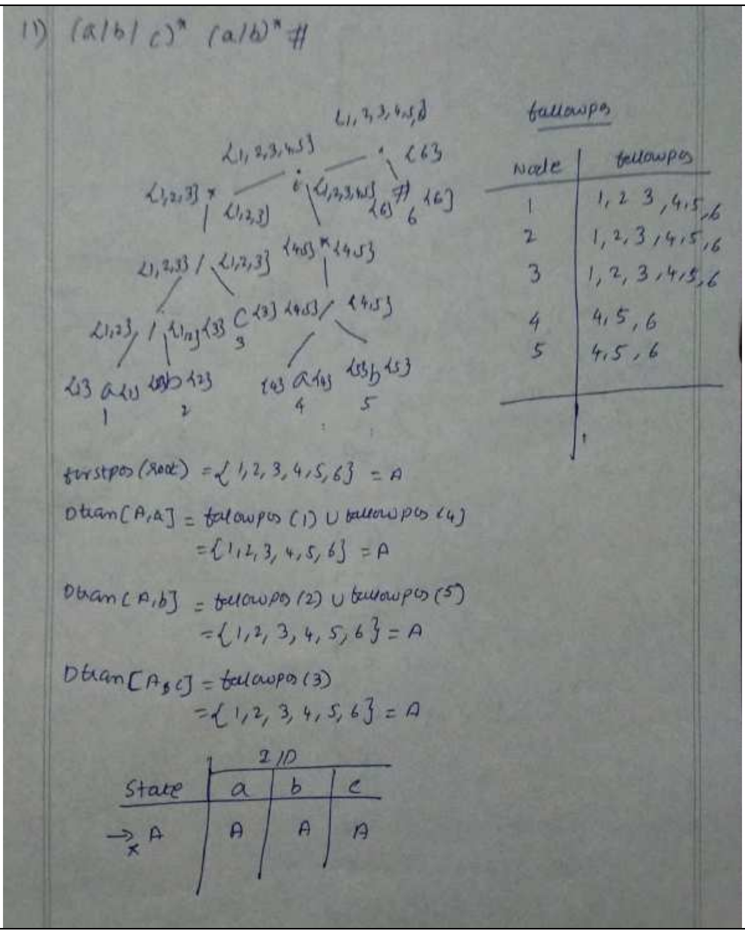
Dtran[E, c] = ∅

State	I/P		
	a	b	c
→A	B	∅	A
B	B	∅	C
*C	B	D	A
D	E	∅	∅
*E	∅	D	∅

<CO1>

(OR)

<KL3>	11. Apply direct method to construct DFA for the regular expression (a|b|c)*(a|b)*.	<CO1>



11) (a/b/ c)* (a/b)* #

{1,2,3,4,5,6}
{1,2,3,4,5}
{6}
{1,2,3} *
{1,2,3}
{1,2,3,4,5}
# {6}
{6}
6
{1,2,3} / {1,2,3}
{4,5} * {4,5}
{1,2} / {1,2}
{3} c {3}
3
{4,5} / {4,5}
{1} a {1}
1
{2} b {2}
2
{4} a {4}
4
{5} b {5}
5

followpos

Node	followpos
1	1, 2 3 ,4,5,6
2	1, 2, 3 ,4,5,6
3	1, 2, 3 ,4,5,6
4	4, 5 , 6
5	4,5 , 6

firstpos (root) = {1,2,3,4,5,6} = A

Dtran[A,a] = followpos (1) ∪ followpos (4)
= {1,2,3,4,5,6} = A

Dtran[A,b] = followpos (2) ∪ followpos (5)
= {1,2, 3, 4, 5, 6} = A

Dtran[A,c] = followpos (3)
= {1,2, 3, 4, 5, 6} = A

	2/D		
State	a	b	c
→ * A	A	A	A

<KL3>

12. Construct parse tree for the sentence S using grammar G.

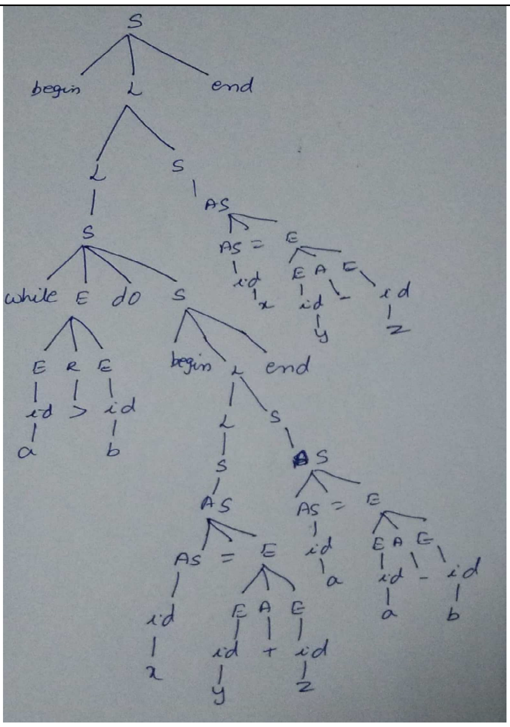
Grammar G:

```
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 S:

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

<CO2>

	 S begin L end L S S AS AS = E while E do S id E A E id x id - z y E R E begin L end id > id a b L S S AS AS AS = E AS = E id E A E a id - id a b id E A E x id + id y z	
	(OR)	

<KL3>	13. Construct recursive descent parser for the grammar G.
	Parse the string **id / id - (id - id)**
	G: E $\rightarrow$E – T | T
	T $\rightarrow$ T / F | F
	F $\rightarrow$ (E) | id
	
	Left recursion eliminated grammar
	E$\rightarrow$TE^1
	$E^1$$\rightarrow$-T$E^1$ \| ε
	T$\rightarrow$FT^1
	$T^1$$\rightarrow$/F$T^1$ | ε
	F $\rightarrow$ (E) | id
	Procedure E()
	{
	T();E^1();
	}
	Procedure E^1()
	{
	If input symbol = ‘-‘ then
	Advance(); T();E^1();
	}
	Procedure T()
	{
	F();T^1();
	}
	Procedure T^1()
	{
	If input symbol = ‘/‘ then
	Advance(); F();T^1();
	}
	
	Procedure F()
	{
	If input symbol = ‘id’ then
		<CO2>

	Advance(); E();
	If input symbol = ‘(‘ then
	Advance();
	If input symbol = ‘)‘ then
	Advance();
	Else error();
	Else error();
	}
	
	Funtion call Input
	                                                        id/id-(id-id)
	E()
	T()
	F()
	Advance()
	T^1()
	Advance()
	F()
	Advance()
	T^1()
	E^1()
	Advance()
	T()
	F()
	Advance()
	E()
	T()
	F()
	Advance()
	T^1()
	E^1()
	Advance()
	T()
	F()
	Advance()
	T^1()
	E^1()
	Advance()
	T^1()
	E^1()	

Prepared By	Reviewed By	Approved By
Course Coordinator	PAC Team	HOD