Q3

1.

	SLR	LALR	CLR
(1)	SLR parser is the smallest in size.	LALR & SLR have the small size, as they have less no. of states.	CLR parser is the largest. As the number of states is very large.
(2)	Error detection is not immediate in SLR.	Error detection is not immediate in LALR.	Error detection can be done immediately in CLR parser.
(3)	SLR fails to produce a parsing table for a certain class of grammars.	It is intermediate in power between SLR and CLR. i.e, SLR ≤ LALR ≤ CLR	It is very powerful and works on a large class of grammar.
(4)	It requires less time and space complexity.	It requires more time and space complexity.	It also requires more time and space complexity.
(5)	It is very easy and cheap to implement.	It is also easy and cheap to implement.	It is expensive and difficult to implement.

2. For target machines with several CPU registers, the code generator is responsible for register allocation. This means that the compiler must be aware of which register are being used for particular purposes in the generated program, and which become available as code is generated.

The instruction-set architecture of the target machine has a significant impact on the difficulty of constructing a good code generator that produces high-quality machine code.

The most common target-machine architectures are

(i) RISC (Reduced instruction set computer)
(ii) CISC (Complex instruction set computer)
(iii) Stack based

3. Given. (a,(a,a))

Stack	Input	Action
$	(a,(a,a)) $	$ < (, shift
$(	a,(a,a)) $	(< a , shift
$(a	,(a,a)) $	a > , , Reduce
$(	,(a,a)) $	(> , , Reduce
$	,(a,a)) $	$ < , , shift
$,	(a,a)) $	, < (, shift
$,(	a,a)) $	(< a , shift
$,(a	,a)) $	a > , , Reduce
$,(	,a)) $	(> , , Reduce
$,	,a)) $	, > , , Reduce
$	,a)) $	$ < , , shift
$,	a)) $	, < a , shift
$,a	)) $	a >) , reduce
$,	)) $	, >) , reduce
$	)) $	$ <) , shift
$)	) $	) >) , reduce
$	)$	$ <) shift
$)	$	) > $ reduce
$	$	accept

4. Given, $S \to E$

$E \to E_1 + T$

$E \to T$

$T \to T_1 * F$

$T \to F$

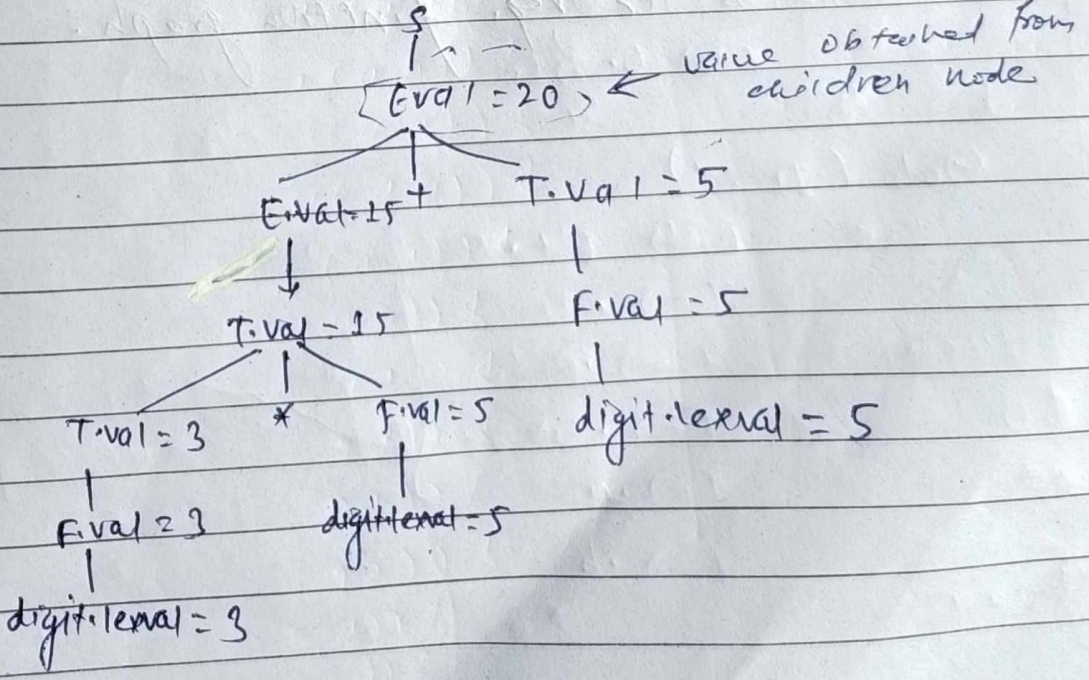
$F \to digit$

The SDD for the given grammer can be written as:

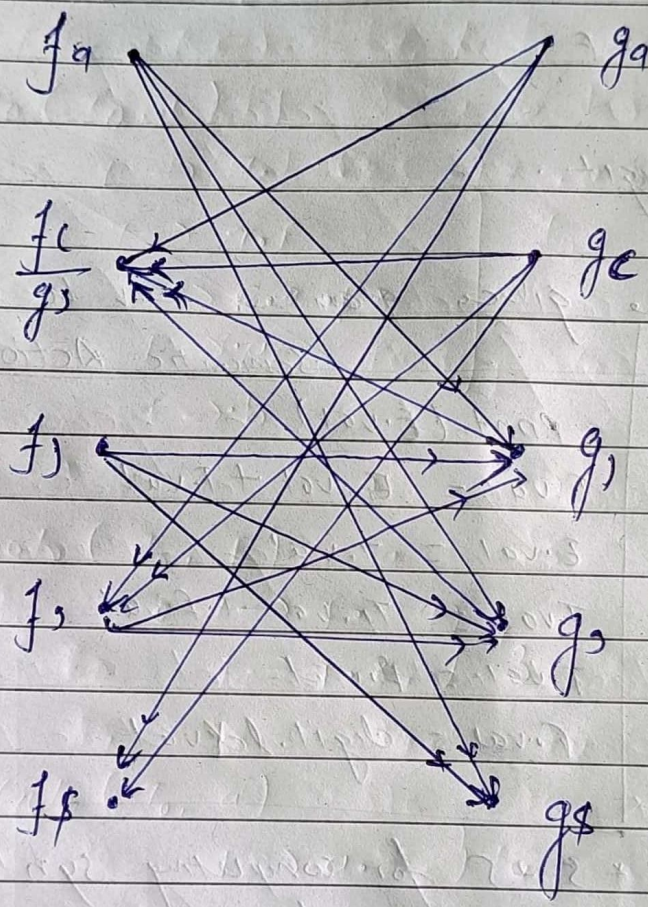
production	Semantic Actions
$S \to E$	print (E.val)
$E \to E_1 + T$	E.val = E_1.val + T.val
$E \to T$	E.val = T.val
$T \to T_1 * F$	T.val = T_1.val + f.val
$T \to F$	T.val = F.val
$F \to digit$	F.val = digit.lexval

input string : 3 * 5 + 5 for computing synthesized attribute.

The annotated parse tree for the input string is



precedence function graph:



This is the req. function graph.

SLR

6. $E \to E+T \mid T$
$T \to TF \mid F$
$F \to F* \mid a \mid b$

Input : a*b+a

Sol^n

Step 1. Convert the grammar into augment grammar

0	$E' \to E$
1	$E \to E+T$
2	$E \to T$
3	$T \to TF$
4	$T \to F$
5	$F \to F*$
6	$F \to a$
7	$F \to b$

Step 2 : Construct LR(0) items.

I_0

$E' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot TF$
$T \to \cdot F$
$F \to \cdot F*$
$F \to \cdot a$
$F \to \cdot b$

Goto (I_0, E)

$E' \to E.$
$E \to E. + T$ } I_1

Goto (I_0, T)

$E \to T.$ R_2
$T \to T. F$
$F \to .F*$
$F \to .a$
$F \to .b$ } I_2

Goto (I_0, F)

$T \to F.$ R_4
$F \to F. *$ } I_3

Goto (I_0, a)

$F \to a.$ R_6 } I_4

Goto (I_0, b)

$F \to b.$ R_7 } I_5

Goto (I_1, +)

$E \to E + .T$
$T \to .TF$
$T \to .F$
$F \to .F*$
$F \to .a$
$F \to .b$ } I_6

Goto (I_2, F)

$T \rightarrow TF.$ R_5 } I_7

$F \rightarrow F.*$

Goto (I_2, a)

$F \rightarrow a.$ R_3 } I_4

Goto (I_2, b)

$F \rightarrow b.$ R_4 } I_5

Goto (I_3, *)

$F \rightarrow F*.$ R_6 } I_8

Goto (I_6, T)

$E \rightarrow E+T.$

$T \rightarrow T.F$

$F \rightarrow .F*$

$F \rightarrow .a$

$F \rightarrow .b$ } I_9

Goto (I_6, F)

$T \rightarrow F.$ R_2 } I_3

$F \rightarrow F.*$

Goto (I_6, a)

$F \rightarrow a.$ R_3 } I_4

Goto (I_6, b)

$F \rightarrow b.$ R_4 } I_5

Goto (I_7, *)

$F \rightarrow F*.$ R_6 } I_8.

Goto (I_9, F)

$T \rightarrow TF.$ R_5 } I_7

$F \rightarrow F.*$

Goto (I_9, a)

$F \rightarrow a.$ } I_4

Goto (I_9, b)

$F \rightarrow b.$ } I_5

computation of follow
we can find out

Follow(E) = {+, $}
follow (T] = {+, a, b, $}
follow (F) = {+, *, a, b, $}.

SLR Parsing Table

State	Action					goto		
	+	*	a	b	$	E	T	F
0			S_4	S_5		1	2	3
1	S_6				accept			
2	r_2		S_4	S_5	r_2			7
3	r_4	S_8	r_4	r_4	r_4			
4	r_6	r_6	r_6	r_6	r_6			
5	r_7	r_7	r_7	r_7	r_7			
6			S_4	S_5			9	3
7	r_3	S_8	r_3	r_3	r_3			
8	r_5	r_5	r_5	r_5	r_5			
9	r_1		S_4	S_5	r_1			7

Given input : a*b+a

Stack	input string	Action
$0	a*b+a$	action[0, a] = S_4. shift, push a, 4
$0a4	*b+a$	action[4, *] = r_6

		reduce r_6 $F \to a$ then pop $2 \times |a| = 2$ push F and Goto goto(0, F) = 3 push 3
$0F3	*b+a$	action [3, *] = s8 shift, push (*, 8)
$0F3*8	b+a$	action [8, b] = r_5 $F \to F*$ $2 \times |2| = 4$ push F and goto [0, F] = 3 push 3.
$0F3	b+a$	action [3, b] = r_4 $T \to F$ $2 \times |F| = 2$ push T and goto [0, T] = 2 push 2
~~$0F3*8~~	*	
$0T2	b+a$	action [2, b] = s5 shift, push b, 5
$0T2b5	+a$	action [5, +] = r_7 $F \to b$ $2 \times 1 = 2$ push F and goto [2, F] = 7 push 7.

$0T2F7	+a$	action [7, +] = r_3 $T \rightarrow TF$ 2x|TF| = 4 push T and goto[0,T] = 2 push 2
$0T2	+a$	action [2, +] = r_2 $E \rightarrow T$ pop 2x|T| = 2 push E and goto[0,E] = 1 push 1
$0E1	+a$	action [1, +] = S_6 push +, 6
$0E1+6	a$	action [6, a] = S_4 push a, 4
$0E1+6a4	$	action [4, $] = r_6 reduce, $F \rightarrow a$ 2x|a| = 2 push F and goto [6, F] = 3 push 3
$0E1+6F3	$	action [3, $] = r_4 Reduce, $T \rightarrow F$, 2x|1| = 2 push T and goto [6, T] = 9 push 9.

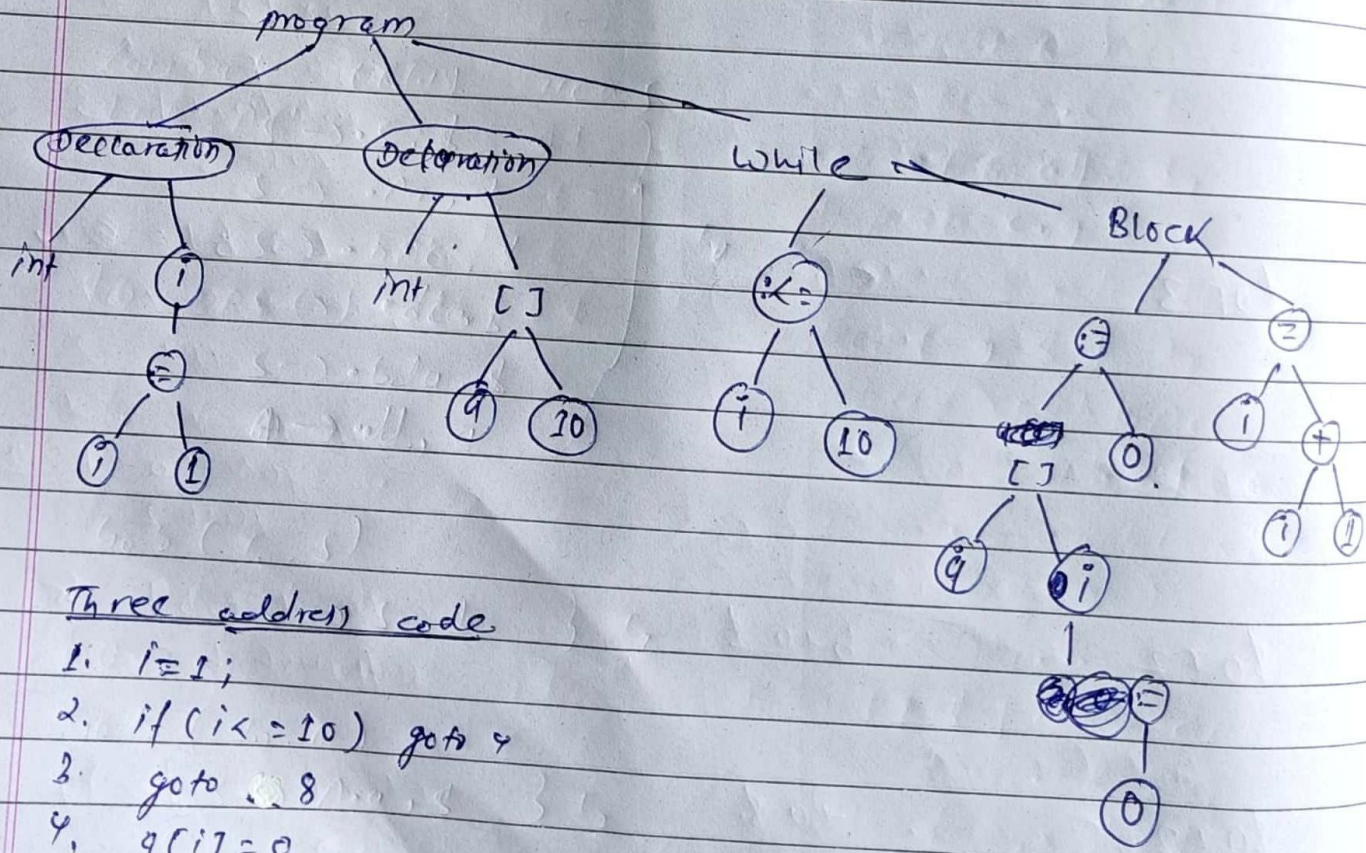
$0E1+6T9	$	action [9, $] = r_1 reduce, $E \to E+T$ $2 \times 3 = 6$ push E and goto [0, E] = 1 Push 1
$0E1	$	action [1, $] = Accept.

Hence the given string is accepted.

7.

```
1   main()
2   {
3       int i;
4       int a[10];
5       i = 1;
6       while (i<=10)
7       {
8           a[i] = 0;
9           i = i+1;
10      }
11  }
12.
```

Syntax tree

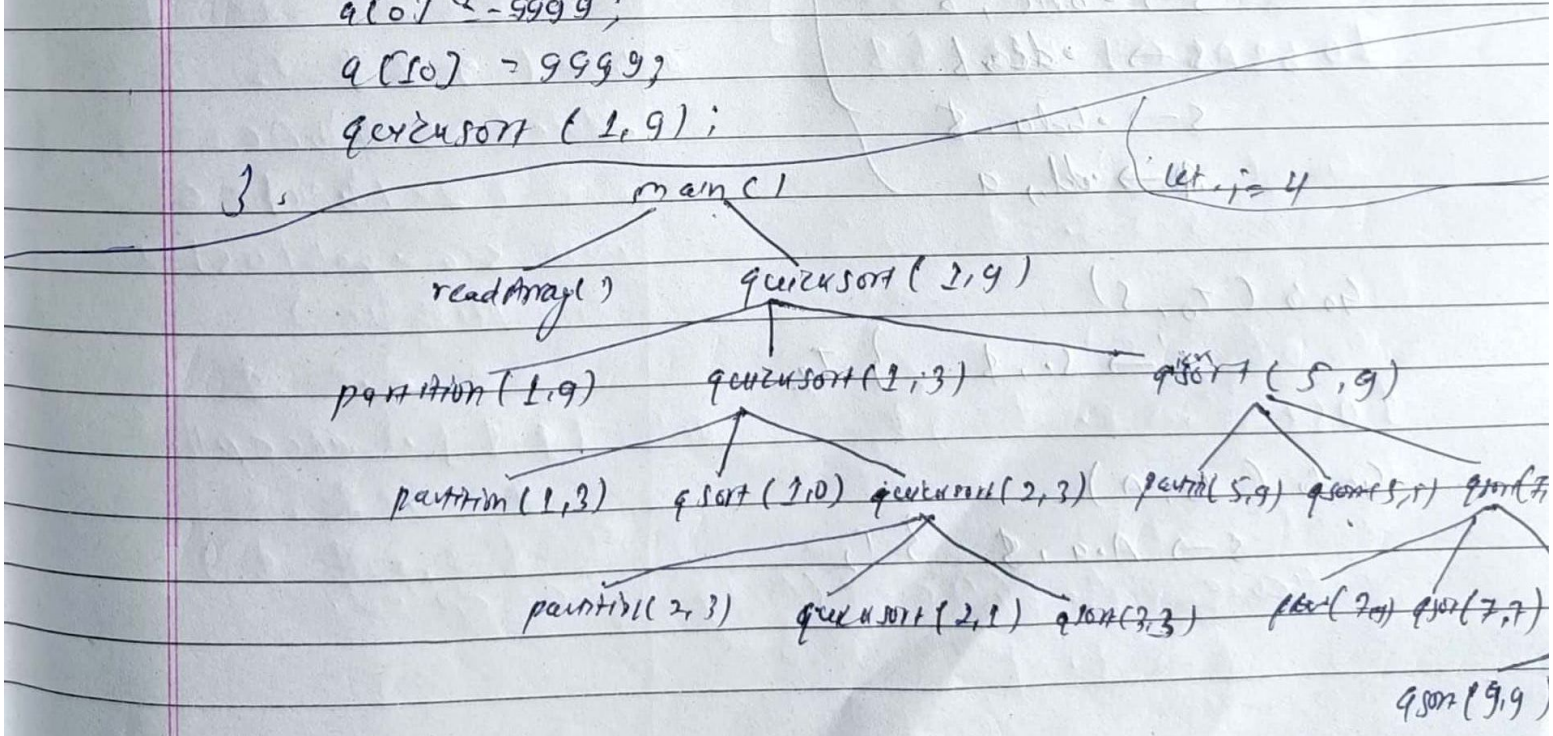


Three address code

```
1.  i = 1;
2.  if (i<=10) goto 4
3.  goto 8
4.  a[i] = 0
5.  t1 = i+1;
6.  i = t1;
7.  goto 2
8.
```

8.

```
int a[9];
void readArray (){
    // read 9 integers into array a[]
}
int partition (int m, int n){
    // returns partition
}
void quicksort (int m, int n){
    int i;
    if (n > m){
        i = partition (m, n);
        quicksort (m, i-1);
        quicksort (i+1, n);
    }
}
int main()
    readArray ();
    a[0] = -9999;
    a[10] = 9999;
    quicksort (1, 9);
}
```



9.

1. $S \to Aa$
2. $S \to bAc$
3. $S \to dc$
4. $S \to bda$
5. $A \to d$
6. So in 2

Construct Augmented Grammar.

(0) $S' \to S$
(1) $S \to Aa$
(2) $S \to bAc$
(3) $S \to dc$
(4) $S \to bda$
(5) $A \to d$

Construct CLR(1) items

I_0:
- $S' \to .S, \$$
- $S \to .Aa, \$$
- $S \to .bAc, \$$
- $S \to .dc, \$$
- $S \to .bda, \$$
- $A \to .d, a$

Goto (I_0, S)

I_1:
- $S' \to S., \$$

Goto (I_0, A)

I_2:
- $S \to A.a, \$$

Goto(I_0, b)

$S \to b \cdot AC, \$$
$S \to b \cdot dq, \$$
$A \to \cdot d, c$
} I_3

Goto(I_0, d)

$S \to d \cdot c, \$$
$A \to d \cdot, a$
} I_4

Goto(I_2, a)

$S \to Aq \cdot, \$$ } I_5

Goto(I_3, A)

$S \to bA \cdot c, \$$ } I_6

Goto(I_3, d)

$S \to bd \cdot q, \$$
$A \to d \cdot, c$
} I_7

Goto(I_4, c)

$S \to dc \cdot, \$$ } I_8

Goto(I_6, c)

$S \to bAc \cdot, \$$ } I_9

Goto(I_7, q)

$S \to bdq \cdot, \$$ } I_{10}

In the states I_0 to I_{10}, no states have a similar first element or core. So we cannot merge the states.

states	Action					goto	
	a	b	c	d	$	S	A
0		s_3		s_4		1	2
1					accept		
2	s_5						
3				s_7			6
4	r_5		s_8				
5					r_1		
6			s_9				
7	s_{10}		r_5				
8					r_3		
9					r_2		
10					r_4		

parsing of string "bdc".

stacks	input string	Action
$0	bdc$	action[0, b] = s_3 shift, push b, 3
$0b3	dc$	action[3, d] = s_7 shift, push d, 7
$0b3d7	c$	action(7, c] = r_5 reduce, A → d 2x|d| = 2, pop push A and goto[3, A] = 6 push 6.
$0b3A6	c$	action[6, c] = s_9 shift push c, 9
$0b3A6c9	$	action[9, $] = r_2 reduce, S → bAc 2x|bAc| = 6 push S and goto[0, S] = 1 push 1.
$0S1	$	action[1, $] = accept.

Hence, the given string is accepted.

11. $t_1 := -r$

$t_2 := q * t_1$

$t_3 := -r$

$t_4 := s * t_3$

$t_5 := t_2 + t_4$

$p := t_5$

Soln:

Quadruple Representation.

Location (#)	operator	Arg1	Arg2	Result
(0)	uminus	r		t_1
(1)	*	q	t_1	t_2
(2)	uminus	r		t_3
(3)	*	s	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	=	t_5		p

12. 1. $S \to L=R$
2. $S \to R$
3. $L \to *R$
4. $L \to id$
5. $R \to L$

Sol:-

Construct Augmented Grammar

(0) $S' \to S$
(1) $S \to L=R$
(2) $L \to *R$
(3) $L \to id$
(4) $R \to L$

Construct LR(1) items:-

I_0:
$S' \to .S, \$$
$S \to .R, \$$
$S \to .L=R, \$$
$L \to .*R, \$/=$
$L \to .id, \$/=$
$R \to .L, \$$

Goto (I_0, S)
$S' \to S., \$$ } I_1

Goto (I_0, R)
$S \to R., \$$ } I_2

Goto (I_0, L)
$S \to L.=R, \$$
$R \to L., \$$ } I_3

Goto (I_0, *)
$L \to *.R, \$/=$
$R \to .L, \$/=$
$L \to .*R, \$/=$
$L \to .id, \$/=$ } I_4

Goto (I_0, id)
$L \to id., \$/=$ } I_5

Goto (I_3, =)
$S \to L=.R, \$$
$R \to .L, \$$
$L \to .*R, \$$
$L \to .id, \$$ } I_6

Goto (I_4 , R)

$L \to *R \cdot , \$ =$ } I_7

Goto (I_4 , L)

$R \to L \cdot , \$ =$ } I_8

Goto (I_4 , *)

$L \to * \cdot R , \$ =$

$R \to \cdot L , \$ =$

$L \to \cdot * R , \$ =$

$L \to \cdot id , \$ =$ } I_4

Goto (I_4 , id)

$L \to id \cdot , \$ =$ } I_5

Goto (I_6 , R)

$S \to L = R \cdot , \$$ } I_9

Goto (I_6 , L)

$R \to L \cdot , \$$ } I_{10}

Goto (I_6 , *)

$L \to * \cdot R , \$$

$R \to \cdot L , \$$

$L \to \cdot * R , \$$

$L \to \cdot id , \$$ } I_{11}

Goto (I_6 , id)

$L \to id \cdot , \$$ } I_{12}

Goto

Goto (I_{11} , R)

$L \to *R \cdot , \$$ } I_{13}

Goto (I_{11} , L)

$R \to L \cdot , \$$ } I_{10}.

Goto (I_{11} , *)

$L \to * \cdot R , \$$

$R \to \cdot L , \$$

$L \to \cdot * R , \$$

$L \to \cdot id , \$$ } I_{11}

Goto (I_{11} , id)

$L \to id \cdot , \$$ } I_{12}

CALR

	Action				Goto		
states	id	=	*	$	S	L	R
0	S_5		S_4		1	3	2
1				accept			
2				R_2			
3		S_6		R_5			
4,11	S_5		S_4			8	7
5,12		R_4		R_4			
6	S_{12}		S_{11}			10	11
7,13		R_3		R_3			
8,10		R_5		R_5			
9				R_1			

Stack	input	Actions
$0	id = id $	action [0, id] = $5 shift, push id, 5
$0 id 5	= id $	action [5, =] = R_4 Reduce, L → id, 2×|id| = 2 pop 2 and goto [0, L] = 3 push 3
$0 L 3	= id $	action [3, =] = S_6 push =, 6
$0 L 3 = 6	id $	action [6, id] = S_{12} push id, 12

$0L3=6id12	$	action [12, $] = R4 Reduce L → id, 2×1=2 push L and goto [6, 2] = 10 push 10.
$0L3=6L10	$	action [10, $] = R5 Reduce, R → L, 2×1=2, pop push R and goto [6, R] = 9 push 9
$0L3=6R9	$	action [9, $] = R1 S → L=R, 2×3 = 6 push S and goto [0, S] = 1 push 1.
$0S1	$	action [1, $] = Accept.

Hence the given string is accepted.

13. In static allocation scheme, the compilation data is bound to a fixed location in the memory and it does not change when the program executes.

As the memory requirement and storage locations are known in advance, runtime support package for memory allocation and de-allocation is not required.

In a static storage-allocation strategy, it is necessary to be able to decide at compile time exactly where data object will reside at run time. In order to make such a decision, at least two criteria must be met:

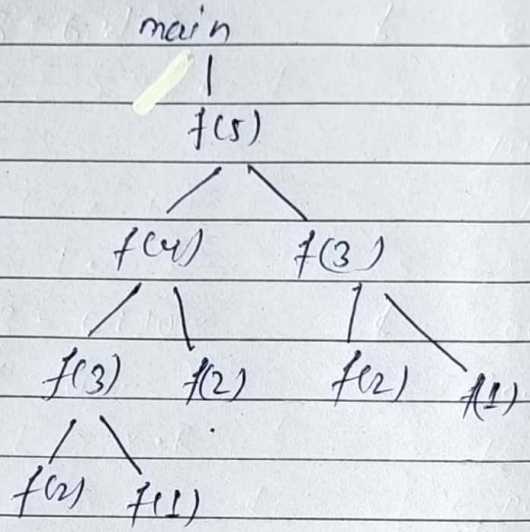
(i) The size of each object must be known at compile time.

(ii) Only one occurrence of each object is allowable at a given moment during program execution.

imp

Because of second ~~(Execution)~~ criterion, nested procedures are not possible in a static storage-allocation scheme. This is the case because it is not known at compile time which or how many nested procedures, and hence their local variables, will be active at execution time.

14.



Result = 5 Ans//

Recall the fibonacci sequence 1, 1, 2, 3, 5, 8, 13, 21 ---- defined by $f(1) = f(2) = 1$ and for $n > 2$, $f(n) = f(n-1) + f(n-2)$. Consider the main function calls that result from a main program calling $f(5)$.

15. Same as Q.N. 4 Repeat

16. program not given to optimize.

17. Question bank for unit 4 & unit 5

18. question bank for unit 4 & unit 5

20. question bank for unit 4 & unit 5.