Question 6

$E' \to E$

$E \to E \text{ sub } E \text{ sup } E$ ①

$E \to E \text{ sub } E$ ②

$E \to \{ E \}$ ③

$E \to c$ ④

FOLLOW (E) = {$, sub, sup, }}

state	sub	sup	c	{	}	$	E
0			s_3	s_2			1
1	s_4					accept	
2			s_3	s_2			6
3	r_4		r_4		r_4	r_4	
4			s_3	s_2			5
5	s_4/r_2	s_7/r_2			r_2	r_2	
6	s_4				s_9		
7			s_3	s_2			8
8	s_4/r_1	r_1			r_1	r_1	
9	r_3	r_3			r_3	r_3	

2 A compiler that recognises only the parts of a program that have changed since the last compile

- Faster and efficient

Question 8

1. Improving quality, reducing redundancy, optimising performance

2. $E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow id$

	id	*	+	$
id	—	·>	·>	·>
*	<·	·>	·>	·>
+	<·	<·	·>	·>
$	<·	<·	<·	—

Question 9

1. Supplying data, memory management, register allocation

2. Input - 3 address statement (a := b op c)

3. E → iE'

E' → +iE' | ε

Recursive Descent Parser:

```
i E() {
    if (lookahead == 'i')
        match('i')
        E'()
}

E'() {
    if (lookahead == '+')
        match('+')
        match('i')
        E'()
    else return
}
```

Question 10

1. (> @ % > # > $

2. 90

Question 11

1. Finite automata is used to maintain control flow in compilers

2. LEX should keep track of keywords itself

3. The input buffer contains the input and leaks one character at a time to LEX

4. $S \rightarrow PQR$

$P \rightarrow a \mid Rb \mid \varepsilon$

$Q \rightarrow c \mid dp \mid \varepsilon$

$R \rightarrow e \mid f$

FIRST	FOLLOW
S = a, e, f, c, d	S = ~~c, d, ε, e, f~~ $
P = a, e, f, , ε	P = ~~anf~~ c, d, e, f
Q = c, d, ε	Q = e, f
R = e, f	R = $

Question 12

1. Finite automata is used to maintain control flow in compilers

2. LEX should keep track of keywords itself

3. The input buffer contains the input and leaks one character at a time to the lexical analyser

4.

FIRIT	FOLLOW
S = a, e, f, c, d	S = $
P = a, e, f, $\overset{\varepsilon}{\text{~~[illegible]~~}}$	P = c, d, e, f
Q = c, d, ε	Q = e, f
R = e, f	R = $

Question 13

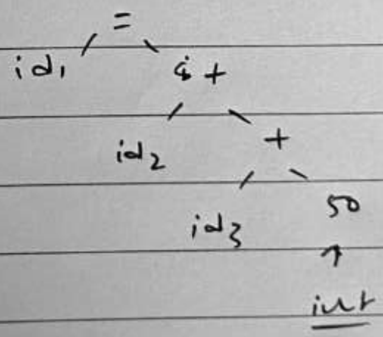
1. Lexical analyser a = b+c * 50

→ $id_1 = id_2 + id_3 * 50$

↓

Syntactic analyser $id_1 = id_2 + id_3 * 50$

→



↓

Semantic analyser

↓

Immediate code Gen

↓

code optimisation

↓

code Generation