Saahas Reddy
AP18110010220
CSE-D

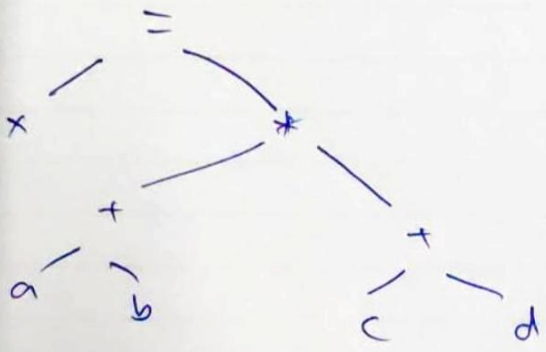
1. a. lexical Analysis :- Given code snippet : x = (a+b) * (c+d)

Identifiers — x, a, b, C, d

operators — +, *, =

Delimiters — (,)

Syntax tree :



Intermediate code Generation :

$t_1 = c+d;$

$t_2 = a+b;$

$t_3 = t_1 + t_2;$

$x = t_3;$

Code optimization:

```
t1 = c+d;
t2 = a+b;
x = t1 * t2;
```

Code Generation:

```
LD R1, d    // loading d in R1
LD R2, c    // loading c in R2
ADD R1, R1, R2  // Adding R1, R2
LD R2, b    // loading b in R2
LD R3, a    // loading a in R3
// Adding R2, R3    ADD R2, R2, R3
// multiplying R1, R2    MUL R1, R1, R2
// Storing in x    ST x, R1
```

2) Define lexeme, token and pattern. Identify lexemes that makes up the tokens in the following Program Segment and indicate the corresponding Pattern.

```
Int sum (int i, int j)
  {
      int temp;
      temp = i+j;
      return temp;
  }
```

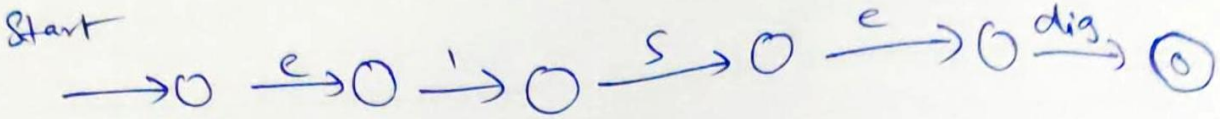
A) lexemes :- Sequence of characters in Source Program matching a pattern.

Tokens :- A Pair consisting of tokens names, attributes values.

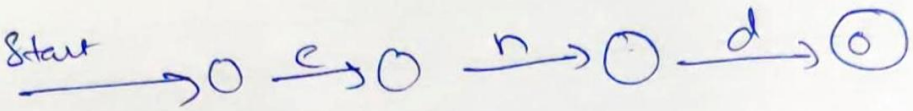
Pattern :- A description of the form that different lexeme tokens.

Tokens	lexemes
Identifiers	sum i, j, temp.
Keywords	int, return.
Parenthesis	{, }, (,)
Plus (Arithmetic OP)	+
Semicolon	;
Comma	,

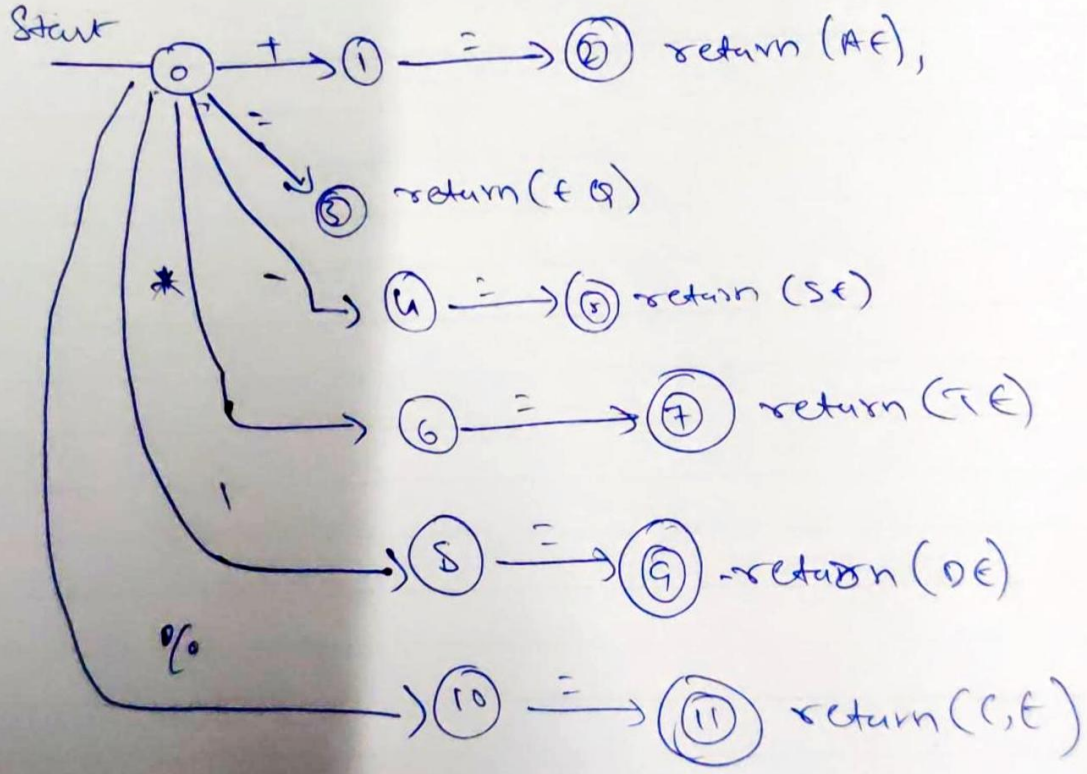
3) a. else :-



end ;



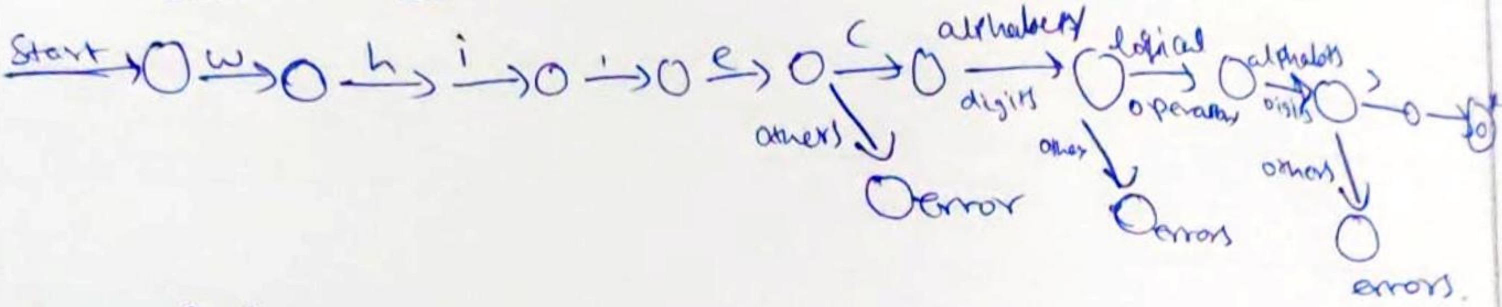
b) Assignment operators : , + =, - =, * =, / =, % =



C while (condition) Regular expression: "while +1(* (a-z)/w*)*

([=<>]=?)* ([a-z]\w*]\d+*])*

Transition diagram :



Q.4, %{ int first-line = 1;

%}

line .*\n

%%

{line} {printf ("%10d %s", first-line++, yytext);}

%%

int yywrap(){}

int main (int argc, char * argv[])

{ extern FILE *yyin;

yyin = fopen ("file.c", "r");

yylex();

return0;

}