Q.1.

Ans :-

1. <u>Lexical Analysis</u>:

a -	Identifier.
=	Assignment operator.
10	Number
;	delimeter.
b	Identifier.
=	Assignment operator.
(	delimeter.
a	Assignment operator.
*	Multiplication operator.
100	Number
)	delimeter.
+	addition. operator.
a.	Identifier.
;	delimeter.
c	Identifier.
=	assignment operator.
a	Identifier.
*	multiplication. operator.
100	Number.
;	delimeter.

As output It gives stream of token.

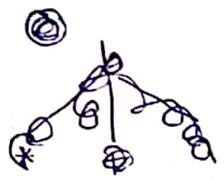
Scanned by CamScanner

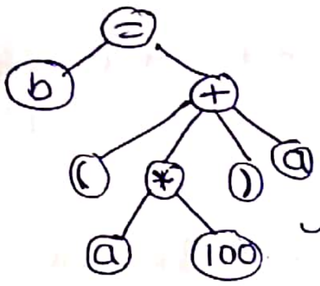
Name-Ashutosh Soni Id-2018UCP1505

2. Syntax analysis :-

Parse tree for Expression :

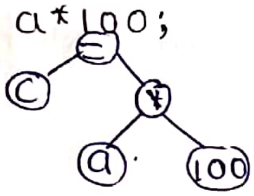
b = (a * 100) + a;





→ parse tree for (a * 100) + a

c = a * 100;



as output it gives parse tree.

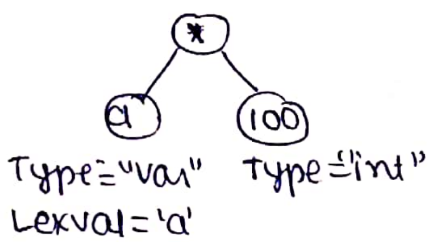
3. Semantic Analysis:

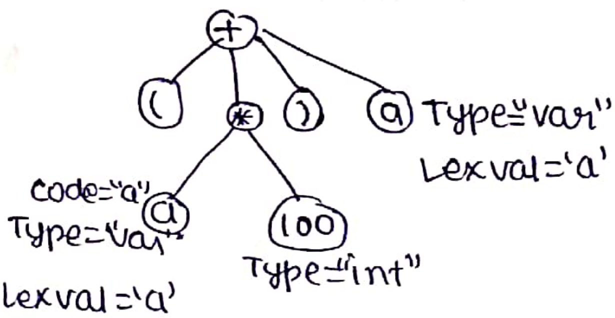
It takes parse tree & symbol table & verify the given code is semantic correct or Not.

Symbol Table.

a	variable	int
b	variable	int
+	operator	
*	operator.	
(	delimeter	
)	delimeter	

→ According to this table it checks wheather the code is containing same type or not.





Name-Ashutosh Soni Id-2018UCP1505

(4) Intermediate code Generation.

a=10; ↳ change to 3 Address code.

b=(a*100)+a;.

t1:= a*100.
t2:= t1+a
b:= t2.

→ three address code for above Exp.

c= a*100;.
ok according to 3 address code

This step give a program for ~~target~~ abstract machine.

(5). Code optimization.

two ways

① Store value of a*100 on another variable then. let x is that variable

a=10;
x=a*100;.
b= x+a;
c=x;

by this we remove multiple calculation of a*100.

②

a=10;.

b= (a*100)+a;
↓
change to
b= a*101;

c= a*100;

another way.

In this step we optimize the code.

Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505

a=10;.

b=(a*100)+a;.

3 address code we get

B1

GEN(a,t1,t2,
b, c).

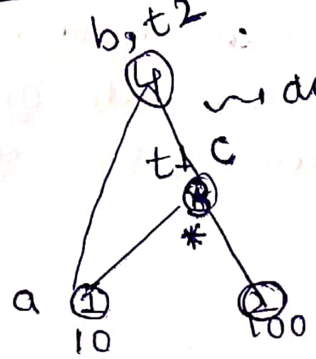
IN → Φ.

OUT → (a,t1,t2,
b,c).

$\boxed{\begin{array}{l} a=10;. \\ t1=a*100 \\ t2=t1+a \\ b=t2. \\ c=a*100. \end{array}}$

leader B1.

linearflow so no.



~ dag.

DAG for above code
to analyse flow.

⑥. Code Generation.

It is machine dependent phase. According to Machine this step takes place.

LOAD R0, a

MUL R0, 100

Store. C, R0

ADD R0, a

Store b, R0.

final machine code.

Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505

Q.2.

Ans ⇒

Lex code for counting frequency of keywords:

```
%{.
    int countauto=0; countdouble=0, countif=0,
     countstatic=0, countbreak=0, countelse=0,
     countint=0, countstruct=0, countcase=0,
     countenum=0, countlong=0 , countswitch=0,
     countchar=0, countextern=0, counttypedef=0,
     countconst=0, countfloat=0, countcontinue=0,
     countregister=0, countunion=0, countunsigned=0,
     countvoid=0, countwhile=0, countdefault=0,
     countdo=0, countgoto=0, countsigned=0,
%}.

%%.

[a-z A-Z]+ {
        if (strcmp(yytext,"auto")==0){
                countauto++;
        }.
        if(strcmp(yytext,"double")==0){
                countdouble++;
        }
        if (strcmp(yytext,"if") ==0){
                countif++;
        }
        if (strcmp(yytex,"static")==0){
                countstatic++;
        }.
```

Scanned by CamScanner

Name- Ashutosh Soni ID-2018UCP1505

```
if(strcmp(yytext,"break")==0){
        countbreak++;
}
if(strcmp(yytext,"else")==0){
        countbreak++;
}
if(strcmp(yytext,"int")==0){
        countint++;
}
if(strcmp(yytext,"struct")==0){
        countstruct++;
}
if(strcmp(yytext,"case")==0){
        countstruct++;
}
if(strcmp(yytext,"long")==0){
        countnum++;
}
if(strcmp(yytext,"switch")==0){
        countswitch++;
}
if(strcmp(yytext,"char")==0){
        countchar++;
}
if(strcmp(yytext,"typedef")==0){
        counttypedef++;
}
if(strcmp(yytext,"float")==0){
        countfloat++;
}
if(strcmp(yytext,"continue")==0){
        countcontinue++;
}.
```

Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505

```
if(!strcmp(yytext, "register")){
      count register++;
}
if(!strcmp(yytext, "unsigned"){
      countunsigned++;
}
if ( strcmp(yytext, "do") ){
      countdo++;
}
if (strcmp(yy text, "default)"{
      countdefault++;
}
if ( strcmp(yytext, "do")){
      countdo++;
}
if (strcmp(yytext, "goto")){
      countgoto++;
}
if( strcmp(yy text, "signed") ){
      countsigned++;
}
}
%%
int yywrap()
{ return1;
}
```

Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505.

```
int main(){.
    FILE *file;.
    file. = fopen("input.txt","r")}
    yyin=file;.
    if(countauto!=0){
        printf( "auto: freq= %d", countauto);
    }
    if (countdouble!=0){
        printf("double: freq= %d",countdouble);
    }
    if (countif!=0){
        printf(" if :freq= %d", countif);
    }
    if(countstatic!=0){
        printf("static :freq=%d",static);
    |
    |
    // as for rest we write).
    return 1;.
}.
```

Name - Ashutosh soni   Id-2018UCP1505

Que 3:

Ans →.

$S \rightarrow . SA | A.$

$A \rightarrow (S) | ().$

After adding new one. to remove left recursion

~~S → A~~

$S \rightarrow AS'$

$S' \rightarrow AS' | \varepsilon.$

$A \rightarrow (S) | ().$

Now grammer is free from of left recursion

$S \rightarrow . AS'$

$S' \rightarrow AS' | \varepsilon$

$A \rightarrow (Y.$

$Y \rightarrow S) |).$

~~&~~ now it is free from common prefixes.

First(S) = { (}.   Follow(S) = {), \$ }

First(S') = { (, ε).   Follow(S') = { \$ }.

First(A) = { (}.   Follow(A) = { (, \$ }.

First(Y) = { (,) }   Follow(Y) = { (, \$ }.

Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505.

Parsing table for above Grammer.

	(	)	$
S	S→AS'		
A	A→(Y		
Y	Y→S)	Y→)	
S'	S→AS'		S'→ε.

LL(1) Parsing table for the given grammer.

Ques 4:→

Ans ⇒ (b) Id→2018UCP1505

Part (C)

S → print (E);

| if (C) then S else S.

| while (C) S.

E → id | num.

trace of the parse tree.

if (C) then while (C) print(id);.

else print (num);

S' → S.

S → print (E).

S → if (C) then else(S).

S →. while(C) S.

E → id | ~~num~~.

E → num.

Scanned by CamScanner

Name - Ashutosh Soni - Id - 2018UCP1505

I0

$S' \to \cdot S$
$S \to \cdot print(E)$
$S \to \cdot if(C) then else (S)$
$S \to \cdot while(C) S$
$E \to \cdot id$
$E \to \cdot num$

$I0 \xrightarrow{S} I1$: $S' \to S \cdot$

$I0 \xrightarrow{print} I2$: $S \to print \cdot (S)$

$I0 \xrightarrow{if} I3$ $S \to if \cdot (C)$ then. S else S.

$I0 \xrightarrow{while} I4$. $S \to while \cdot (C) S$.

$I2 \xrightarrow{(} I7$. $S \to print (\cdot E)$. $E \to id / \cdot num$.

$I3 \xrightarrow{S} I8$ if ($\cdot$ C) then else S

$I4 \xrightarrow{(} I9$ while ($\cdot$ C) S.

$I7 \xrightarrow{E} I10$ $S \to print (E \cdot)$.

$I0 \xrightarrow{id} I11$ $E \to id \cdot$

$I0 \xrightarrow{num} I12$ $E \to num \cdot$

$I8 \xrightarrow{C} I13$ $S \to if (C \cdot)$ then. S else S.

$I9 \xrightarrow{C} I14$ $S \to while (C \cdot) S$.

Scanned by CamScanner

Name-Ashutosh Soni   Id-2018UCP1505

$I_{10} \xrightarrow{)} I_{15}$ S→print(E)·

$I_{13} \xrightarrow{)} I_{16}$ S→if (l)·then Selse S.

$I_{41} \xrightarrow{)}$

I_{17}

S→while ()·S.	→while I_4
S→·print(E).	→print I_2
S→·if(()then Selse S.	→if I_7
S→·while (l)·S	

$I_{16} \xleftarrow{then} I_{18}$

$I_4 \xleftarrow{while}$

$I_2 \xleftarrow{print}$

$I_3 \xleftarrow{if}$

S→if (l) then ·Selse S

S→ while(E)·S

S→·print(E).

S→·if(() then Selse S

S→while(()S

Name - Ashutosh Soni Id - 2018UCP1505

Que 5

Ans ⇒

E ⇒ (+ E E)

{ E.val = (E.val + E.val) }.

E ⇒ (* E E)

{ E.val = (E.val * E.val) }.

E ⇒ (- E E)

{ E.val = (E.val - E.val) }

E ⇒ { / E E }

{ E.val = E.val / E.val) }.

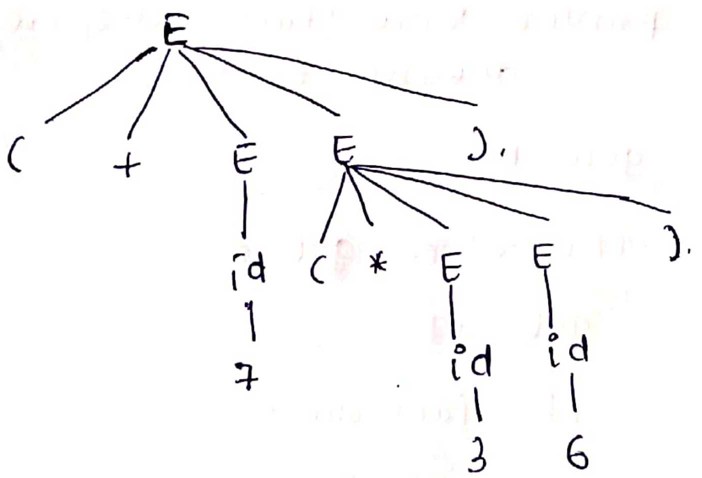
E → id

id → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

this is the S - attributed definition

Now, we construct Parse tree for given prefix expression :

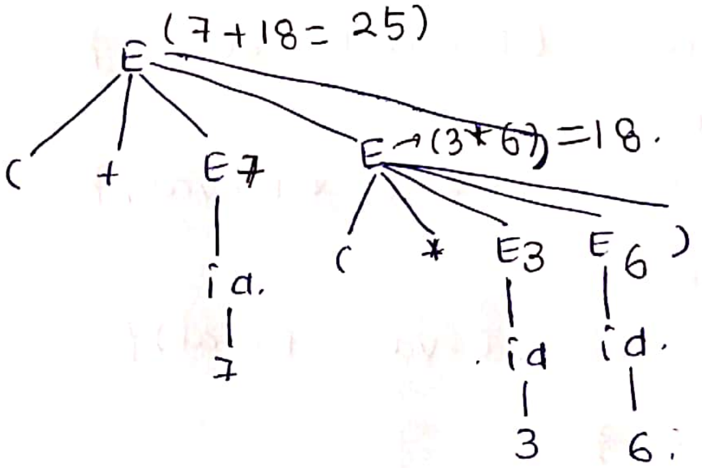
+ 7 * 3 6.



Scanned by CamScanner

Name-Ashutosh Soni Id-2018UCP1505

Now using the grammer and semantic action we convert it into infix..



using

E.val = (E.val * E.val)

E.val = (E.val + E.val).

for Above Infix expression is

(7 + (3*6))

Que 6:

Ans:-

3 Address code for above program:-

1) If(n >= 0) goto 4.

2) printf("Error! Factorial of negative number doesnot exist");

3). goto 12

4). if(i<=n) goto 6.

5) goto 11.

6). T1 = factorial * i

7) factorial = T1.

8) T2 = i+1

Scanned by CamScanner

Name- Ashutosh Soni Id-2018UCP1505

9) i = T2.

10) goto 4.

11) printf(" Factorial. of %d =%d", n, factorial);.

12) return 0
(exit)

leader → 1, 2, 4, 5, 6, 11.

B1
d1: if (n>=0) goto 4.

B2 d2: printf("Error! Factorial of negative number doesnot exist");

d3: goto 12

B3 d4: if (i<=n) goto 6

B4 d5: goto 11

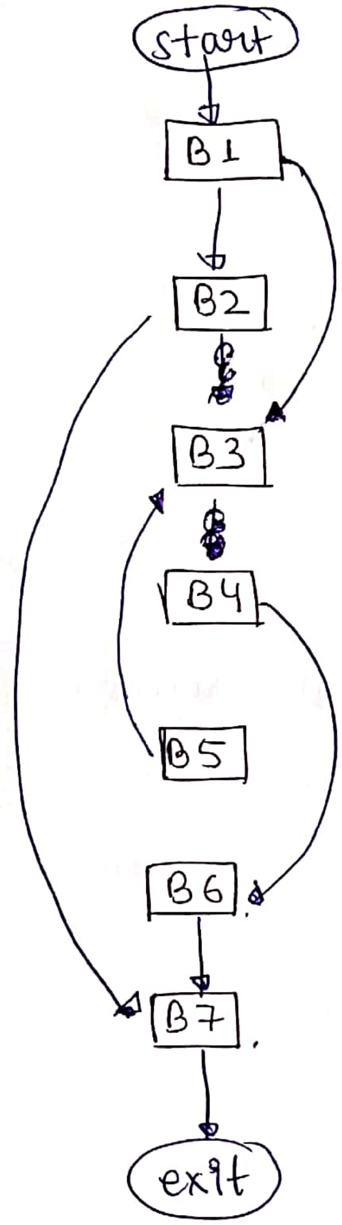
B5. d6: T1 = factorial * i.
d7: factorial = T1
d8: T2 = i+1
d9: i = T2.
d10: goto 4

B6. d11: printf(" Factorial of %d = %d", n, factorial);
d12: return 0.

Scanned by CamScanner

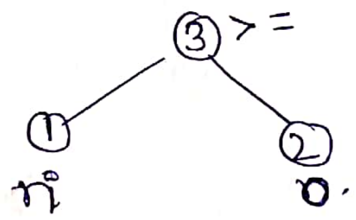
Name- Ashutosh Soni Id- 2018UCP1505

Control flow.

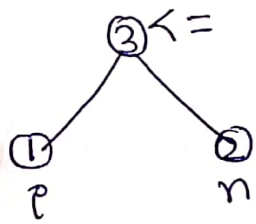


DAG for this program.

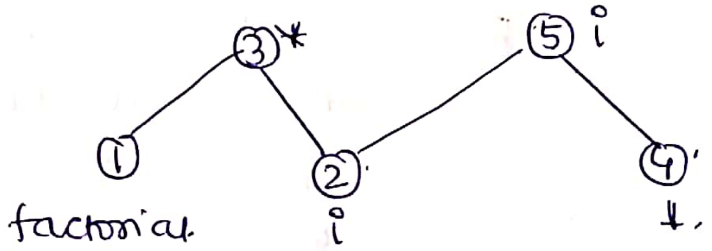
Block 1:



Block 3:



Block 5:



Scanned by CamScanner