Sushant Kumar Tiwari
21010127

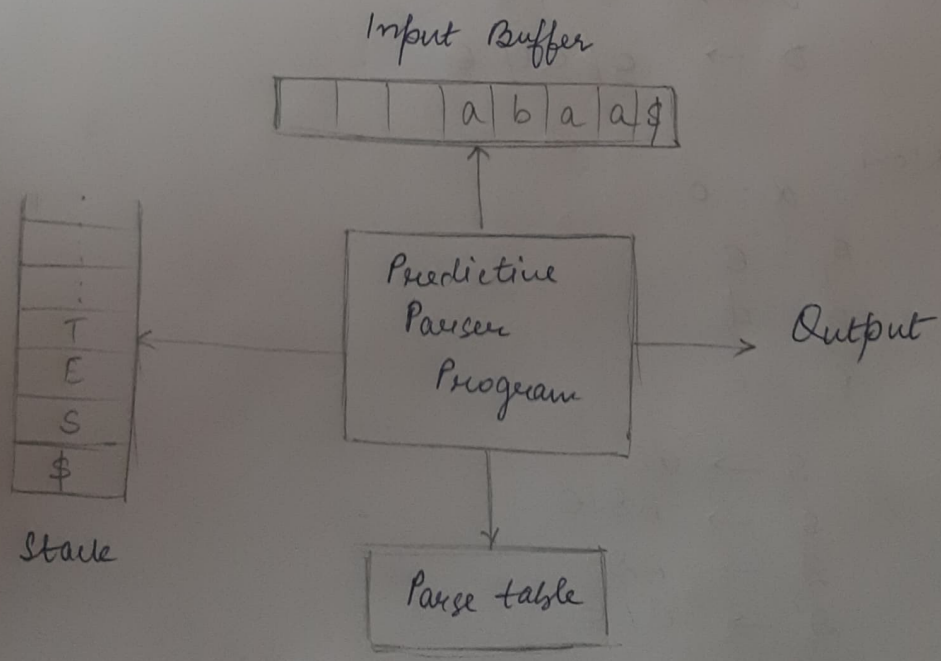
PART-A

1. Properties of operator precedence grammar are:
   (a) There are no two non terminal symbols adjacent to each other.
   (b) The R.H.S of any production does not contain any ϵ.

2. A handle is a substring that matches with the R.H.S of a production and its reduction gives a non-terminal symbol.

3. In case of a bottom-up parser, there can be multiple possibilities of using a production. So, in backtracking if a shift fails we can move back to different production route.

4. Model of LL(1) parser



Sushant Kumar Tiwari
21010127

5. Given,

$S \rightarrow a / ab / abc / abcd$

Here $\alpha = a$

$\beta_1 = \epsilon$

$\beta_2 = b$

$\beta_3 = bc$

$\beta_4 = bcd$ and ~~γ~~ no γ

$\therefore \quad A \rightarrow \alpha A' / \gamma_1 / \gamma_2 / \ldots$

$A' \rightarrow \beta_1 / \beta_2 / \beta_3 / \ldots$

$\therefore \quad S \rightarrow aS'$

$S' \rightarrow \epsilon / b / bc / bcd$

Now,

$\alpha = b$

$\beta_1 = \epsilon$

$\beta_2 = c$

$\beta_3 = cd$

$\gamma_1 = \epsilon$

$\therefore \quad S \rightarrow aS'$

$S' \rightarrow bD / \epsilon$

$D \rightarrow c / cd / \epsilon$

Now,

$\alpha = c$

$\beta_1 = \epsilon$

$\beta_2 = d$

$\gamma_1 = \epsilon$

$\therefore \quad S \rightarrow aS'$

$S' \rightarrow bD / \epsilon$

$D \rightarrow cD'$

$D' \rightarrow d / \epsilon$

Sushant Kumar Tiwari
21010127

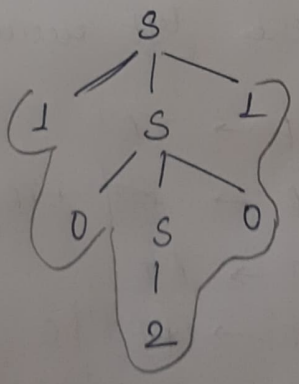
PART-B

6. Given,

$S \rightarrow 0S0 \mid 1S1 \mid 2$

Input string : 10201

Stack	Input Buffer	Action
$	10201$	shift
$1	0201$	shift
$10	201$	shift
$102	01$	Reduce S→2
$10S	01$	shift
$10S0	1$	Reduce S→0S0
$1S	1$	shift
$1S1	$	Reduce S→1S1
$S	$	Accepted



Sushant Kumar Tiwari
21010127

7(a) A Recursive descent parser is a top down parser.

In recursive descent parser each non terminal is defined as a function or procedure which is invoked whenever a non terminal symbol is encountered.

The functions are defined to read the sequence of input string and increment the pointer of the input string to next character and return the main pointer to the root of non-terminal.

During matching, if the input character matches with that in the procedure then it is consumed the the pointer moves forward.

In case of non-terminal symbols their respective procedures is called.

For example:

$$E \rightarrow E+T \mid \epsilon$$
$$T \rightarrow T*F \mid \epsilon$$
$$F \rightarrow (E) \mid id$$

Now, Removing the left recursive grammar

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \epsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \epsilon$$
$$F \rightarrow (E) \mid id$$

Now, defining procedures

```
E(){
    T();
    EPRIME();}
```

Sushant Kumar Tiwari
21010127

```
EPRIME() {
    if ( input == '+') {
        input ++;
        T();
        EPRIME();
    }
    else
    return;
}

T() {
    F();
    TPRIME();
}

TPRIME() {
    if (input == '*') {
        input ++;
        F();
        TPRIME();
    }
    else
    return;
}

F() {
    if (input == '(') {
        input++;
        E();
        if( input == ')')
            input++;
    }
```

Sushant Kumar Tiwari
21010127

```
else if( input == 'id')
    input ++;
}
```

If input is a terminal symbol then we will compare it with i[pos]

If input is a non terminal then we will call the function.

8(b)

$S \rightarrow AB \mid eDa$
$A \rightarrow ab \mid c$
$B \rightarrow de$
$C \rightarrow eC \mid \epsilon$
$D \rightarrow fD \mid \epsilon$

(i) First (S) = {AB, eDa} = {a, e}
First (A) = {a, c}
First (B) = {d}
First (C) = {e, ϵ}
First (D) = {f, ϵ}

Follow (S) = {$}
Follow (A) = {First(B)} = {d}
Follow (B) = Follow(S) = {$}
Follow (C) = Follow(S) = {$}
Follow (D) = {a}

Sushant Kumar Tewari
21010127

(ii) Predictive parser Table

	a	b	c	d	e	f	$
S	S→ AB				S→eDa		
A	A→ ab		A→c				
B				B→dc			
C					C→eC		C→ ϵ
D	D→ a					D→f	

~~M[S,~~ ① S → AB
S → eDa

Add S → AB to M[S, a]
Add S → eDa to M[S, e]

② A → ab/c
add A → ab to M(A, a)
add A → c to M(A, c)

③ B → dc
Add B → dc to M[B, d]

④ C → eC
C → ϵ
add C → eC to M(C, e]
add C → ϵ to M[C, $]

⑤ D → fD
D → ϵ
add D → fD to M[D, f]
add D → ϵ to M[D, a]