- **Stack implementation of operator precedence parser**

Ex: - 2

Use Stack implementation of operator precedence parser to check this sentence id + id by this grammar: E→ E+E | E*E | id

	id	**+**	*****	**$**
id		>	>	>
+	<	>	<	>
*****	<	>	>	>
$	<	<	<	

Sol:

Stack		**Input**
$	<	id + id $
$ <id	>	+id $
$<id>		+id $
$<E+		+id$
$<E+	<	id$
$<E+<id	>	$
$<E+<id>		$
$<E+E>		$
$ E		$ →**accepted**

Stack		**Input**
$	<	id + id$
$ <id	>	+ id$
$<id>		+ id$
$<E+		+ id$
$<E+	<	id$
$<E+<id	>	$
$<E+<id>		$
$<E+E>		$
$ E		$
accept		

H.W

Try input **id*(id ↑ id)-id/id** with the following relations

	+	-	*	/	↑	id	(	)	$
+	>	>	<	<	<	<	<	>	>
-	>	>	<	<	<	<	<	>	>
*	>	>	>	>	<	<	<	>	>
/	>	>	>	>	<	<	<	>	>
↑	>	>	>	>	<	<	<	>	>
id	>	>	>	>	>			>	>
(	<	<	<	<	<	<	<	=	
)	>	>	>	>	>			>	>
$	<	<	<	<	<	<	<		

2.6.9 LR parser

This section presents an efficient bottom-up syntax analysis technique that can be used to parse a large class of context-free grammars. These technique is called LR parsing; the L is for **left-right** scanning of the input, the R for constructing a **rightmost** derivation in reverse. This method present three techniques for construct an LR parsing table for grammar.

LR parsing

- الجدول يعطى في السوال
- Grammar الاصلية في السوال تقسم الى مجاميع وبدون فواصل وترقم.
- نقوم بوضع رقم 0 في بداية الجدول عند الحل
- S = shift الرقم مع s ينقل كما هو مع ما تم سحبه
- R = reduce الرقم مع r هو رقم القانون في الجدول

Example:-

E→ E+T |T
T→ T*F|F
F→ (E) | id

State	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Sol:

1- E→ E+T
2- E→ T
3- T→ T*F
4- T→ F
5- F→ (E)
6- F→ id

Parse **id * id + id $**

No	Stack	input
1	0 ← من الجدول تقاطع الـ id مع 0 وهي s5 وتعني shift →	**id * id + id $**
2	0 id5 ← تقاطع الـ 5 مع * هي r6 وحسب قانون رقم 6 تحول الـ F→id →	*** id + id $**
3	0 F3	*** id + id $**
4	0 T2	***id + id $**
5	0 T2*7	**id + id $**
6	0 T2*7 id5	**+ id $**
7	0 T2*7 F10	**+ id $**
8	0 T2	**+ id $**
9	0 E1	**+ id $**
10	0 E1+6	**id $**
11	0 E1+6 id5	**$**
12	0 E1+6 F3	**$**
13	0 E1+6 T9	**$**
14	0 E1	**$**
Accept		

2.6.7 Construction of predictive parsing tables

Construction of predictive parsing tables: The following algorithm can be used to construct a predictive parsing table for a grammar G. The idea behind the algorithm is the following:

Suppose A→ α is a production with a in FIRST(α). Then the parser will expand A by α when the current input symbol is a. The only complication occurs when α→ λ. In this case, we should again expand A by α if the current input symbol is in FOLLOW(A).

Ex:

E→ E+T | T
T → T*F | F
F → (E) | id

Parse the input id * id + id by using predictive parsing:

1- We must solve the left recursion and left factoring if it founded in the grammar

E → T$\bar{E}$
$\bar{E}$ → +T$\bar{E}$ | λ
T → F$\bar{T}$
$\bar{T}$ → *F$\bar{T}$ | λ
F → (E) | id

2- We must find the first and follow to the grammar:

	First	Follow
E	(, id	$,)
$\bar{E}$	+, λ	$,)
T	(, id	+, $,)
$\bar{T}$	*, λ	+, $,)
F	(, id	+, *, $,)

3- We must find or construct now the predictive parsing table

Since FIRST (T$\bar{E}$) = FIRST (T) = {(, id}, production E→T$\bar{E}$ causes M [E, ( ] and M [E,id] to acquire the entry E→TE'.

Production E'→+TE' causes m [E', +] to acquire $\bar{E}$→+T$\bar{E}$. Production $\bar{E}$→ λ causes M [$\bar{E}$,)] and M [$\bar{E}$, $] to acquire $\bar{E}$→ λ since FOLLOW ($\bar{E}$) = {), $}. So the parsing table produced by the previous algorithm.

NONTER-	INPUT SYMBOL					
MINAL	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (E)$		

Ex:

https://www.youtube.com/watch?v=WgFi2ssIR3A