LECTURE 13

MORE ON LL PARSING

SUBJECTS

Error Recovery

Non LL(1) Grammar

LL(K) Parsers

LL(1) ERROR RECOVERY

What happens when the parser discovers an error?

- **Approach 1**: stop all parsing activity and return an error message

- **Approach 2**: try to continue parsing (if possible) and see if there are more errors along the way

At Nothing!

Which approach does your compiler take?

LL(1) ERROR RECOVERY

uOttawa
L'Université canadienne
Canada's university

An error is detected when:

- The terminal on top of the stack does not match the next input token
- The parsing table cell from which we are supposed to pull the next production is empty

What does the parser do?

- It enters the ***panic-mode*** error recovery
- Based on the idea of skipping symbols on the input until a token in the SYNCH set appears

LL(1) ERROR RECOVERY

uOttawa
L'Université canadienne
Canada's university

Let S be a set of tokens called a synchronization set (SYNCH)

Let s ∈ S ➔ s is called a synchronization token

Million dollar question: how to construct the synchronization set?

- Many heuristics have been proposed
- We will cover a simple method

Place all symbols in FOLLOW(A) into the SYNCH(A) set for nonterminal A

- If we skip tokens until an element of SYNCH(A) is seen and we pop A from the stack, it's likely that parsing can continue

LL(1) ERROR RECOVERY

The panic-mode error recovery can be implemented using the SYNCH set(s) as follows:

- **Scenario 1:** If there is a nonterminal at the top of the stack, discard input tokens until you find a synch token, then pop the non-terminal
- **Scenario 2:** If there is a terminal at the top of the stack, we could try popping it to see whether we can continue
  - Assume that the input string is actually missing that terminal

LL(1) ERROR RECOVERY EXAMPLE

uOttawa
L'Université canadienne
Canada's university

FIRST Sets

```
FIRST(expr)= {num, id}
FIRST(expr')= {+, -, ε}
FIRST(term)= {num, id}
FIRST(term')= {*, /, ε}
FIRST(factor)= {num, id}
```

FOLLOW Sets

```
FOLLOW(expr)={$}
FOLLOW(expr')={$}
FOLLOW(term)= {+, -, $}
FOLLOW(term')= {+, -, $}
FOLLOW(factor)= {*, /, +, -, $}
```

Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & \langle \text{term} \rangle \langle \text{expr}' \rangle \\
\langle \text{expr}' \rangle & ::= & + \langle \text{expr} \rangle \\
 & | & - \langle \text{expr} \rangle \\
 & | & \varepsilon \\
\langle \text{term} \rangle & ::= & \langle \text{factor} \rangle \langle \text{term}' \rangle \\
\langle \text{term}' \rangle & ::= & * \langle \text{term} \rangle \\
 & | & / \langle \text{term} \rangle \\
 & | & \varepsilon \\
\langle \text{factor} \rangle & ::= & \texttt{num} \\
 & | & \texttt{id}
\end{array}
$$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	-
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε
term	<term>→ <factor><term'>	<term>→ <factor><term'>	-	-	-	-	-
term'	-	-	<term'>→ ε	<term'>→ ε	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε
factor	factor→ num	factor → id	-	-	-	-	-

LL(1) ERROR RECOVERY EXAMPLE

FIRST Sets

```
FIRST(expr)= {num, id}
FIRST(expr')= {+, -, ε}
FIRST(term)= {num, id}
FIRST(term')= {*, /, ε}
FIRST(factor)= {num, id}
```

FOLLOW Sets

```
FOLLOW(expr)={$}
FOLLOW(expr')={$}
FOLLOW(term)= {+, -, $}
FOLLOW(term')= {+, -, $}
FOLLOW(factor)= {*, /, +, -, $}
```

Grammar

$$
\begin{array}{lcl}
\langle expr\rangle & ::= & \langle term\rangle\langle expr'\rangle \\
\langle expr'\rangle & ::= & +\langle expr\rangle \\
 & | & -\langle expr\rangle \\
 & | & \varepsilon \\
\langle term\rangle & ::= & \langle factor\rangle\langle term'\rangle \\
\langle term'\rangle & ::= & *\langle term\rangle \\
 & | & /\langle term\rangle \\
 & | & \varepsilon \\
\langle factor\rangle & ::= & \texttt{num} \\
 & | & \texttt{id}
\end{array}
$$

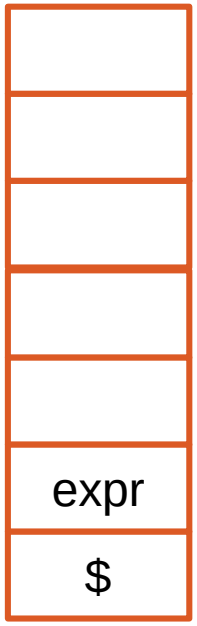
	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

id	*	/	/	+	num	+	id	$

String of errors



	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

id	*	/	/	+	num	+	id	$

term
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

id	*	/	/	+	num	+	id	$

factor
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

id	*	/	/	+	num	+	id	$

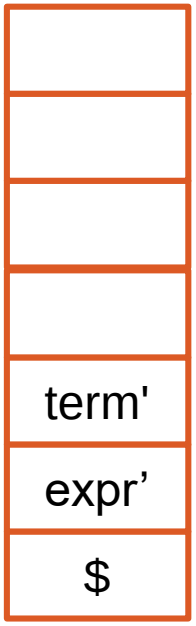
id
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

*	/	/	+	num	+	id	$

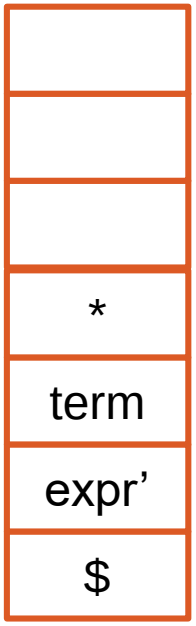


	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

*	/	/	+	num	+	id	$



	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

/	/	+	num	+	id	$

term
expr'
$

Error: the cell corresponding to row term and column / is empty!

Start discarding tokens, until you find a synch token

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

/	+	num	+	id	$

term
expr'
$

Error: the cell corresponding to row term and column / is empty!

Start discarding tokens, until you find a synch token

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

+	num	+	id	$

term
expr'
$

Error: the cell corresponding to row term and column / is empty!

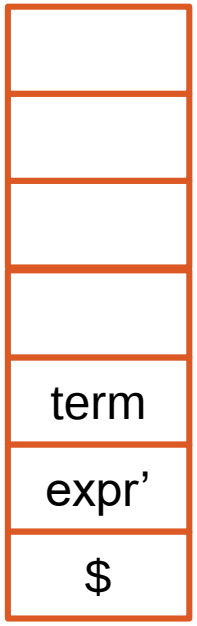
Start discarding tokens, until you find a synch token

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

+	num	+	id	$



We have found a synch token!

Pop term from the stack and attempt to continue…

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

+	num	+	id	$

expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

+	num	+	id	$

+
expr
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

num	+	id	$

expr
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

num	+	id	$

term
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

num	+	id	$

factor
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

num	+	id	$

num
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

+	id	$

term’
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr’	-	-	<expr’>→ +<expr>	<expr’>→ -<expr>	-	-	<expr’>→ ε **(s)**
term	<term>→ <factor><term’>	<term>→ <factor><term’>	**(s)**	**(s)**	-	-	**(s)**
term’	-	-	<term’>→ ε **(s)**	<term’>→ ε **(s)**	<term’>→ *<term>	<term’>→ /<term>	<term’>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

+	id	$

expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

+	id	$

+
expr
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

id	$

expr
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

id	$

term
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

id	$

factor
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:

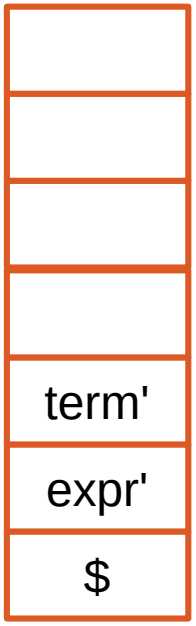
id	$

id
term'
expr'
$

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

Example:



$

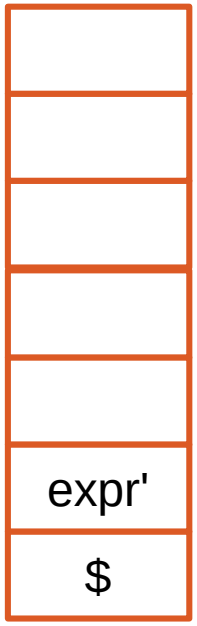
	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

$



	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

PARSING USING LL(1)

uOttawa
L'Université canadienne
Canada's university

Example:

$

$

We did our best to continue parsing…

	num	id	+	-	*	/	$
expr	<expr>→ <term><expr'>	<expr>→ <term><expr'>	-	-	-	-	**(s)**
expr'	-	-	<expr'>→ +<expr>	<expr'>→ -<expr>	-	-	<expr'>→ ε **(s)**
term	<term>→ <factor><term'>	<term>→ <factor><term'>	**(s)**	**(s)**	-	-	**(s)**
term'	-	-	<term'>→ ε **(s)**	<term'>→ ε **(s)**	<term'>→ *<term>	<term'>→ /<term>	<term'>→ ε **(s)**
factor	factor→ num	factor → id	**(s)**	**(s)**	**(s)**	**(s)**	**(s)**

A NON LL(1) GRAMMAR

Consider the grammar:

```
<stmt> ::= if <expr> then <stmt>
         | if <expr> then <stmt> else <stmt>
```

Needs left factoring, which gives:

```
<stmt> ::= if <expr> then <stmt><stmt'>
<stmt'> ::= else <stmt> | ε
```

Let's get the FIRST and FOLLOW sets

FIRST(stmt) = {if}

FIRST(stmt')={else, ε}

1) FIRST(terminal) is {terminal}

2) If A→aα, and a is a terminal:
{a}∈FIRST(A)

3) If A→Bα, and rule B→ε does **NOT** exist:
FIRST(B)∈FIRST(A)

4) If A→Bα, and rule B→ε **DOES** exist:
{(FIRST(B) − ε) ∪ FIRST(α)}∈FIRST(A)

A NON LL(1) GRAMMAR

uOttawa
L'Université canadienne
Canada's university

Consider the grammar:

```
<stmt> ::= if <expr> then <stmt>
         | if <expr> then <stmt> else <stmt>
```

Needs left factoring, which gives:

```
<stmt> ::= if <expr> then <stmt><stmt'>
<stmt'> ::= else <stmt> | ε
```

Let's get the FIRST and FOLLOW sets

FIRST(stmt) = {if}

FIRST(stmt')={else, ε}

FOLLOW(stmt) = {$, else}

FOLLOW(stmt')={$, else}

1) $\{\$\} \in FOLLOW(S)$

2) If $A \rightarrow \alpha B$:
$FOLLOW(A) \in FOLLOW(B)$

3) If $A \rightarrow \alpha B\gamma$, and rule $\gamma \rightarrow \varepsilon$ does **NOT** exist:
$FIRST(\gamma) \in FOLLOW(B)$

4) If $A \rightarrow \alpha B\gamma$, and rule $\gamma \rightarrow \varepsilon$ **DOES** exist:
$\{(FIRST(\gamma) - \varepsilon) \cup FOLLOW(A)\} \in FOLLOW(B)$

***Note**: This is a partial grammar (used to demonstrate a concept), therefore we did not specify the production rules associated with expr. Consequently, its FIRST and FOLLOW sets will not be calculated.*

A NON LL(1) GRAMMAR

FIRST(stmt) = {if}
FIRST(stmt')={else, ε}
…

FOLLOW(stmt) = {$, else}
FOLLOW(stmt')={$, else}
…

```
<stmt> ::= if <expr> then <stmt><stmt'>

<stmt'> ::= else <stmt> | ε
```

	if	then	else	$
stmt				
stmt'				

A NON LL(1) GRAMMAR

uOttawa
L'Université canadienne
Canada's university

FIRST(stmt) = {if}
FIRST(stmt')={else, ε}
…

FOLLOW(stmt) = {$, else}
FOLLOW(stmt')={$, else}
…

```
<stmt> ::= if <expr> then <stmt><stmt'>

<stmt'> ::= else <stmt> | ε
```

	if	then	else	$
stmt	<stmt>→ if <expr> then <stmt><stmt'>	-	-	-
stmt'				

A NON LL(1) GRAMMAR

FIRST(stmt) = {if}
FIRST(stmt')={else, ε}
…

FOLLOW(stmt) = {$, else}
FOLLOW(stmt')={$, else}
…

```
<stmt> ::= if <expr> then <stmt><stmt'>

<stmt'> ::= else <stmt> | ε
```

	if	then	else	$
stmt	<stmt>→ if <expr> then <stmt><stmt'>	-	-	-
stmt'			<stmt'>→ else <stmt>	

A NON LL(1) GRAMMAR

uOttawa
L'Université canadienne
Canada's university

FIRST(stmt) = {if}
FIRST(stmt')={else, ε}
…

FOLLOW(stmt) = {$, else}
FOLLOW(stmt')={$, else}
…

```
<stmt> ::= if <expr> then <stmt><stmt'>

<stmt'> ::= else <stmt> | ε
```

	if	then	else	$
stmt	<stmt>→ if <expr> then <stmt><stmt'>	-	-	-
stmt'			<stmt'>→ else <stmt>	<stmt'>→ ε

A NON LL(1) GRAMMAR

uOttawa
L'Université canadienne
Canada's university

FIRST(stmt) = {if}
FIRST(stmt')={else, ε}
…

FOLLOW(stmt) = {$, else}
FOLLOW(stmt')={$, else}
…

```
<stmt> ::= if <expr> then <stmt><stmt'>

<stmt'> ::= else <stmt> | ε
```

	if	then	else	$
stmt	<stmt>→ if <expr> then <stmt><stmt'>	-	-	-
stmt'	-	-	<stmt'>→ else <stmt>, <stmt'>→ ε	<stmt'>→ ε

A NON LL(1) GRAMMAR

uOttawa
L'Université canadienne
Canada's university

The problem arises because for an input token `else` and stack top of `stmt'`, we do not know which production to choose:

- `<stmt'>→else <stmt>`
- `<stmt'>→ε`

Therefore, this is not an LL(1) grammar

	if	then	else	$
stmt	`<stmt>→ if <expr> then <stmt><stmt'>`	-	-	-
stmt'	-	-	`<stmt'>→ else <stmt>,` `<stmt'>→ ε`	`<stmt'>→ ε`

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

We have already studied LL(1) parser

- With 1 token look-ahead

We will touch briefly on LL(k) parsers

- With "k" tokens look-ahead
- This is useful since not all grammars are LL(1) compatible

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

Consider the following grammar:

<Z> ::= <X><Y><Z>| ab

<Y> ::= cb | ε

<X> ::= ac | b<Y>c

First and Follow sets:

FIRST(Z)={a,b}

FIRST(Y)={c, ε}

FIRST(X)={a,b}

FOLLOW(Z)={$}

FOLLOW(Y)={a,b,c}

FOLLOW(X)={a,b,c}

First Rules

1) FIRST(terminal) is {terminal}

2) If A→aα, and a is a terminal:
{a}∈FIRST(A)

3) If A→Bα, and rule B→ε does **NOT** exist:
FIRST(B)∈FIRST(A)

4) If A→Bα, and rule B→ε **DOES** exist:
{(FIRST(B)- ε)∪FIRST(α)}∈FIRST(A)

Follow Rules

1) {$}∈FOLLOW(S)

2) If A→ αB:
FOLLOW(A) ∈ FOLLOW(B)

3) If A→αBγ, and rule γ→ε does **NOT** exist:
FIRST(γ) ∈ FOLLOW(B)

4) If A→αBγ, and rule γ→ε **DOES** exist:
{(FIRST(γ)-ε)∪FOLLOW(A)}∈FOLLOW(B)

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

Grammar	FIRST	FOLLOW
<Z> ::= <X><Y><Z>| ab	FIRST(Z)={a,b}	FOLLOW(Z)={$}
<Y> ::= cb | ε	FIRST(Y)={c, ε}	FOLLOW(Y)={a,b,c}
<X> ::= ac | b<Y>c	FIRST(X)={a,b}	FOLLOW(X)={a,b,c}

	a	b	c	$
X				
Y				
Z				

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

Grammar	FIRST	FOLLOW
<Z> ::= <X><Y><Z>| ab	FIRST(Z)={a,b}	FOLLOW(Z)={$}
<Y> ::= cb | ε	FIRST(Y)={c, ε}	FOLLOW(Y)={a,b,c}
<X> ::= ac | b<Y>c	FIRST(X)={a,b}	FOLLOW(X)={a,b,c}

	a	b	c	$
X	X→ac	X→bYc	-	-
Y				
Z				

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

Grammar

```
<Z> ::= <X><Y><Z>| ab
<Y> ::= cb | ε
<X> ::= ac | b<Y>c
```

FIRST

FIRST(Z)={a,b}

FIRST(Y)={c, ε}

FIRST(X)={a,b}

FOLLOW

FOLLOW(Z)={$}

FOLLOW(Y)={a,b,c}

FOLLOW(X)={a,b,c}

	a	b	c	$
X	X→ac	X→bYc	-	-
Y	Y→ε	Y→ε	Y→cb, Y→ε	-
Z				

LL(K) PARSERS

uOttawa
L'Université canadienne
Canada's university

Grammar

```
<Z> ::= <X><Y><Z>| ab
<Y> ::= cb | ε
<X> ::= ac | b<Y>c
```

FIRST

FIRST(Z)={a,b}

FIRST(Y)={c, ε}

FIRST(X)={a,b}

FOLLOW

FOLLOW(Z)={$}

FOLLOW(Y)={a,b,c}

FOLLOW(X)={a,b,c}

	a	b	c	$
X	X→ac	X→bYc	-	-
Y	Y→ε	Y→ε	Y→cb, Y→ε	-
Z	Z→ab	Z→XYZ	-	-

Not an LL(1) Grammar

LL(K) PARSERS

Grammar

```
<Z> ::= <X><Y><Z>| ab

<Y> ::= cb | ε

<X> ::= ac | b<Y>c
```

We have shown that the above grammar is not LL(1)

Maybe it is an LL(2) grammar

- Re-create the sets **FIRST(A)** while considering **TWO** terminals appearing **first** in a string derived from A
- Re-create the sets **FOLLOW (A)** while considering **TWO** terminals appearing **after A** in a sentential form

- Make sure that a single production exists in every table cell

LL(2) PARSING TABLE

```
<Z> ::= <X><Y><Z>| ab
<Y> ::= cb | ε
<X> ::= ac | b<Y>c
```

FIRST(Z)={ab, ac,bc}
FIRST(Y)={cb, ε}
FIRST(X)={ac, bc}

FOLLOW(Z)={a$, b$, c$}
FOLLOW(Y)={ab, ac,bc}
FOLLOW(X)={cb,ab,ac,bc}

	aa	ab	ac	a$	ba	bb	bc	b$	ca	cb	cc	c$
X												
Y												
Z												

LL(2) PARSING TABLE

```
<Z> ::= <X><Y><Z>| ab
<Y> ::= cb | ε
<X> ::= ac | b<Y>c
```

FIRST(Z)={ab, ac,bc}
FIRST(Y)={cb, ε}
FIRST(X)={ac, bc}

FOLLOW(Z)={a$, b$, c$}
FOLLOW(Y)={ab, ac,bc}
FOLLOW(X)={cb,ab,ac,bc}

	aa	ab	ac	a$	ba	bb	bc	b$	ca	cb	cc	c$
X			X→ ac				X→bYc					
Y		Y→ε	Y→ε				Y→ε			Y→cb		
Z		Z→ab	Z→XYZ				Z→XYZ					

THANK YOU!

QUESTIONS?