CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

EXERCISE:
fill in the parse table

(see next slide)

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E						
E'						
T						
T'						
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'						
T						
T'						
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'				
T						
T'						
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T						
T'						
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'						
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'			T' -> * F T			
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F						

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F				F -> (E)		

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }
FIRST(E') = { + , ε }
FIRST(T') = { * , ε }

FOLLOW(E') = FOLLOW(E) = {) , $ }
FOLLOW(T') = FOLLOW(T) = { + ,) , $ }
FOLLOW(F) = { + , * ,) , $ }

E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | id

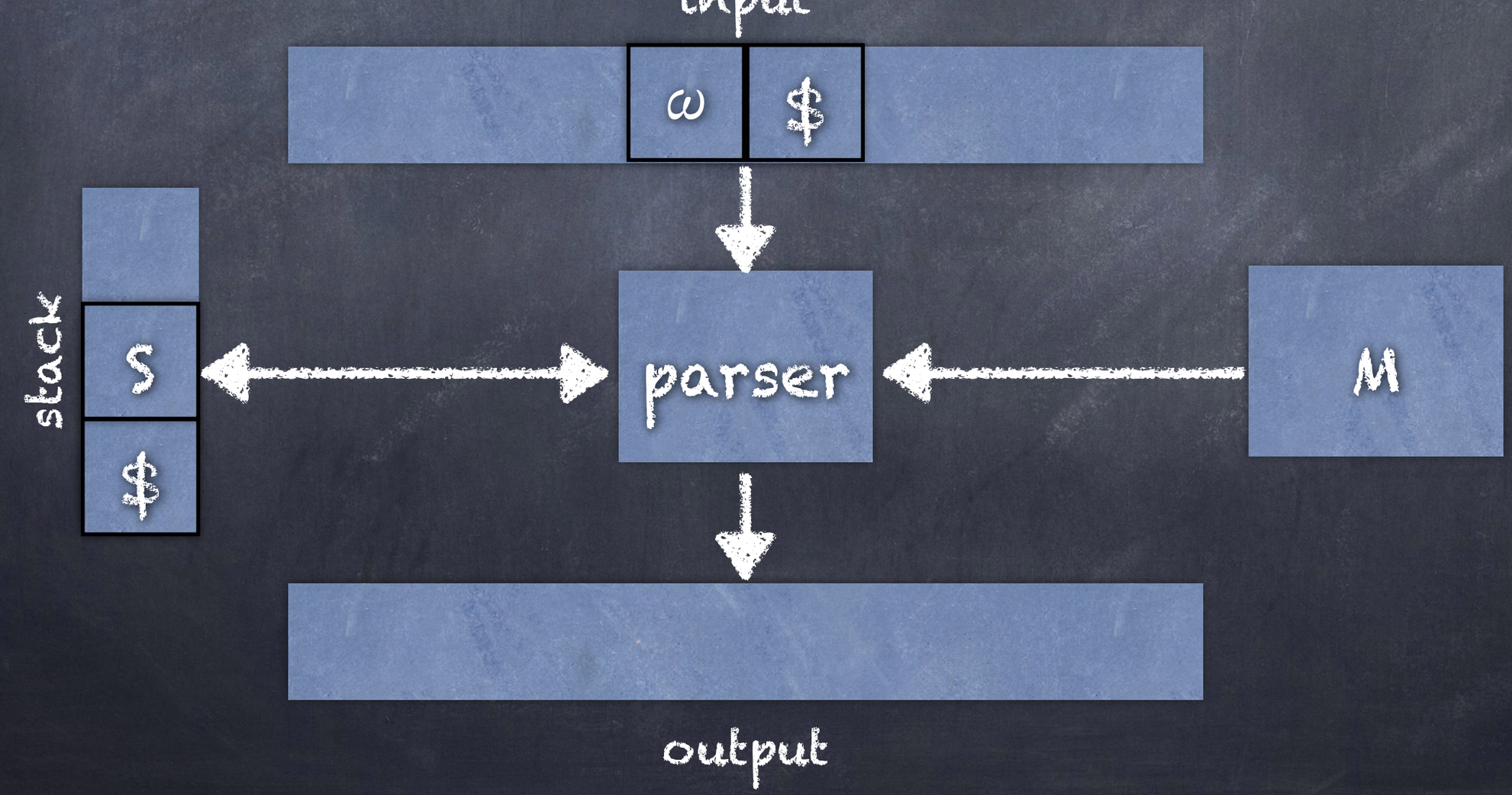
For each production A -> α of G:

- For each terminal a ∈ FIRST(α), add A -> α to M[A,a]
- If ε ∈ FIRST(α), then for each terminal b in FOLLOW(A), add A -> α to M[A,b]
- If ε ∈ FIRST(α) and $ ∈ FOLLOW(A), add A -> α to M[A,$]

Algorithm 4.34 [p. 226]

INPUT: A string ω and a parsing table M for a grammar $G=(N,T,P,S)$.

OUTPUT: If $\omega \in \mathcal{L}(G)$, a leftmost derivation of ω, error otherwise



Algorithm 4.34 [p. 226]

Let a be the first symbol of ω
Let X be the top stack symbol
while (X $\neq$ \$) {
  if (X == a) { pop the stack, advance a in ω }
  else if (X is a terminal) { error }
  else if (M[X,a] is blank) { error }
  else if (M[X,a] is $X \to Y_1 Y_2 \dots Y_k$) {
    output $X \to Y_1 Y_2 \dots Y_k$
    pop the stack
    push $Y_k \dots Y_2 Y_1$ onto the stack
  }
  Let X be the top stack symbol
}
Accept if a == X == \$

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> ε

```
if (X == a) { pop, advance a in ω }
else if (X is a terminal) { error }
else if (M[X,a] is blank) { error }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'		
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T'		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

INPUT (a)	STACK (X)	OUTPUT

```
while (X ≠ $) {
  if (X == a) { pop the stack, advance a in ω }
  else if (X is a terminal) { error }
  else if (M[X,a] is blank) { error }
  else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop the stack
    push Yk ... Y2 Y1 onto the stack
  }
  Let X be the top stack symbol
}
Accept if a == X == $
```

id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> ε
id + id * id $	$	

Since both a and X are $, accept

INPUT (a)	STACK (X)	OUTPUT
id + id * id $	E $	E -> T E'
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> + T E'
id + id * id $	+ T E' $	
id + id * id $	T E' $	T -> F T'
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> * F T'
id + id * id $	* F T' E' $	
id + id * id $	F T' E' $	F -> id
id + id * id $	id T' E' $	
id + id * id $	T' E' $	T' -> ε
id + id * id $	E' $	E' -> ε
id + id * id $	$	

Since both a and X are $, accept

Error recovery in predictive parsing

Basic idea: if input is unexpected given the current state of the parse, try to "synchronize" so that parse can continue (even though no machine code generation will occur once an error has been detected)

Strategies for error recovery

Skip input symbols until a synchronizing token appears.

1. synch(A) includes FOLLOW(A)
2. ... <expr> ; FOLLOW(<expr>) is {';'}, but if ';' is missing, then include all keyword-beginning statement starts in synch(A)
3. add contents of FIRST(A) to synch(A)
4. if A -> ε use this "absorb" A
5. if terminal a on the stack cannot be matched, pop it and issue a message "'a' was inserted". (Synch is set to all other tokens.)

synch is added to each blank M[X,a] when a ∈ FOLLOW(X)

FOLLOW(E) = FOLLOW(E') = {) , $ }

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'			T -> F T'		
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

synch is added to each blank M[X,a] when a ∈ FOLLOW(X)

FOLLOW(T) = FOLLOW(T') = { + ,) , $ }

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id			F -> (E)		

synch is added to each blank M[X,a] when a ∈ FOLLOW(X)

FOLLOW(F) = { + , * ,) , $ }

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

Modified parse-table M

NON TERMINALS	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

synch is added to each blank M[X,a] when a ∈ FOLLOW(X)

- FOLLOW(E) = FOLLOW(E') = {) , $ }
- FOLLOW(T) = FOLLOW(T') = { + ,) , $ }
- FOLLOW(F) = { + , * ,) , $ }

Algorithm 4.34 (modified w/error handling)

Let a be the first symbol of ω
Let X be the top stack symbol
while (X ≠ $) {
  if (x == a) { pop the stack, advance a in ω }
  else if (X is a terminal) { pop stack }
  else if (M[X,a] is blank) { advance a in ω }
  else if (M[X,a] is synch) { pop stack }
  else if (M[X,a] is $X \rightarrow Y_1 Y_2 \ldots Y_k$) {
    output $X \rightarrow Y_1 Y_2 \ldots Y_k$
    pop the stack
    push $Y_k \ldots Y_2 Y_1$ onto the stack
  }
  Let X be the top stack symbol
}
Accept if a == X == $

INPUT (a)	STACK (X)	OUTPUT
+ id * + id $	E $	???

```
if (x == a) { pop, advance a in ω }
else if (X is a terminal) { pop }
else if (M[X,a] is blank) { advance a in ω }
else if (M[X,a] is synch) { pop }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

INPUT (a)	STACK (X)	OUTPUT
+ id * + id $	E $	error, skip +

```
if (x == a) { pop, advance a in ω }
else if (X is a terminal) { pop }
else if (M[X,a] is blank) { advance a in ω }
else if (M[X,a] is synch) { pop }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

INPUT (a)	STACK (X)	OUTPUT
+ id * + id $	E $	error, skip +
+ id * + id $	E $	E -> T E'
+ id * + id $	T E' $	T -> F T'
+ id * + id $	F T' E' $	F -> id
+ id * + id $	id T' E' $	
+ id * + id $	T' E' $	T' -> * F T'
+ id * + id $	* F T' E' $	
+ id * + id $	F T' E' $	???

```
if (x == a) { pop, advance a in ω }
else if (X is a terminal) { pop }
else if (M[X,a] is blank) { advance a in ω }
else if (M[X,a] is synch) { pop }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

INPUT (a)	STACK (X)	OUTPUT
+ id * + id $	E $	error, skip +
+ id * + id $	E $	E -> T E'
+ id * + id $	T E' $	T -> F T'
+ id * + id $	F T' E' $	F -> id
+ id * + id $	id T' E' $	
+ id * + id $	T' E' $	T' -> * F T'
+ id * + id $	* F T' E' $	
+ id * + id $	F T' E' $	error, M[F,+] = synch, pop F

```
if (x == a) { pop, advance a in ω }
else if (X is a terminal) { pop }
else if (M[X,a] is blank) { advance a in ω }
else if (M[X,a] is synch) { pop }
else if (M[X,a] is X -> Y1 Y2 ... Yk) {
    output X -> Y1 Y2 ... Yk
    pop
    push Yk ... Y2 Y1
}
Let X be the top stack symbol
```

	id	+	*	(	)	$
E	E -> T E'			E -> T E'	synch	synch
E'		E' -> + T E'			E' -> ε	E' -> ε
T	T -> F T'	synch		T -> F T'	synch	synch
T'		T' -> ε	T' -> * F T		T' -> ε	T' -> ε
F	F -> id	synch	synch	F -> (E)	synch	synch

INPUT (a)	STACK (X)	OUTPUT
+ id * + id $	E $	error, skip +
+ id * + id $	E $	E -> T E'
+ id * + id $	T E' $	T -> F T'
+ id * + id $	F T' E' $	F -> id
+ id * + id $	id T' E' $	
+ id * + id $	T' E' $	T' -> * F T'
+ id * + id $	* F T' E' $	
+ id * + id $	F T' E' $	error, M[F,+] = synch, pop F
+ id * + id $	T' E' $	T' -> ε
+ id * + id $	E' $	E' -> + T E'
+ id * + id $	+ T E' $	
+ id * + id $	T E' $	T -> F T'
+ id * + id $	F T' E' $	F -> id
+ id * + id $	id T' E' $	
+ id * + id $	T' E' $	T' -> ε
+ id * + id $	E' $	E' -> ε
+ id * + id $	$	

Since there were errors, do not accept.