CS 4240 Phase 1

David Zhang

February 23, 2014

1 Lexical Rules

Pre-parse the input to remove comments with the following DFA. The resulting NOT-COMMENT tokens are concatenated together.

Everything under the token column represents either generated tokens or custom table actions on the character buffer used to produce the text associated with tokens.

start state	symbol	next state	token
START	$\Sigma - \{/, ”\}$	START	NOT-COMMENT
START	“	STRING	
START	/	SLASH	
STRING	$\Sigma - \{\backslash, ”\}$	STRING	
STRING	$\backslash$	STRING-SLASH	
STRING	”	START	NOT-COMMENT
STRING-SLASH	Σ	STRING	
SLASH	$\Sigma - \{*\}$	START	NOT-COMMENT
SLASH	*	COMMENT	
COMMENT	$\Sigma - \{*\}$	COMMENT	
COMMENT	*	COMMENT-END	
COMMENT-END	$\Sigma - \{*, /\}$	COMMENT	
COMMENT-END	*	COMMENT-END	
COMMENT-END	/	START	COMMENT

The DFA for uncommented code.

Note, any time backtracking is mentioned, it essentially is the same as treating the current state as the start state and doing the corresponding transitions or token generations. This is included to simplify the table by removing duplication of the start state transitions.

Note, all ids are later matched character by character with keywords to determine if they are keywords.

Note, drop character for the error state means that the last read character is ignored, and the state remains unchanged.

start state	symbol	next state	token
START	+	START	PLUS
START	-	START	MIN
START	*	START	MULT
START	/	START	DIV
START	=	START	EQ
START	(	START	LPAREN
START	)	START	RPAREN
START	,	START	COMMA

START	&	START	AND
START	—	START	OR
START	[	START	LSQUARE
START	]	START	RSQUARE
START	;	START	SEMI
START	⟨	LANGLE	
START	⟩	RANGLE	
START	:	COLON	
START	0-9	INT-LIT	
START	“	STRING-LIT	
START	a-zA-Z	ID	
START	whitespace	START	ignore
START	others	ERROR	drop character
LANGLE	$\Sigma - \{=, \rangle\}$	START	LESS , backtrack
LANGLE	⟩	START	NOTEQ
LANGLE	=	START	LESSEQ
RANGLE	$\Sigma - \{=\}$	START	GREATER, backtrack
RANGLE	=	START	GREATEREQ
COLON	=	START	ASSIGN
COLON	$\Sigma - \{=\}$	START	COLON, backtrack
INT-LIT	0-9	INT-LIT	
INT-LIT	$\Sigma - 0 - 9$	START	INT-LIT, backtrack
ID	a-zA-Z0-9_	ID	
ID	$\Sigma - a - zA - Z0 - 9\_$	START	ID, backtrack
STRING-LIT	$\Sigma - \backslash$	STRING-LIT	
STRING-LIT	”	START	STRING-LIT
STRING-LIT	\	STRING-LIT-SLASH	
STRING-LIT-SLASH	n	STRING-LIT	
STRING-LIT-SLASH	t	STRING-LIT	
STRING-LIT-SLASH	”	STRING-LIT	
STRING-LIT-SLASH	\	STRING-LIT	
STRING-LIT-SLASH	ˆ	STRING-LIT-CTL	
STRING-LIT-SLASH	0-9	STRING-LIT-CODE-1	
STRING-LIT-SLASH	whitespace	STRING-LIT-SPACE	ignore 2 characters
STRING-LIT-SLASH	others	ERROR	drop character
STRING-LIT-CTL	@A-Z[\]ˆ_ .	STRING-LIT	
STRING-LIT-CTL	others	ERROR	drop character
STRING-LIT-CODE-1	0-9	STRING-LIT-CODE-2	
STRING-LIT-CODE-1	others	ERROR	drop character
STRING-LIT-CODE-2	0-9	STRING-LIT	
STRING-LIT-CODE-2	others	ERROR	drop character
STRING-LIT-SPACE	whitespace	STRING-LIT-SPACE	ignore
STRING-LIT-SPACE	\	STRING-LIT	ignore
STRING-LIT-SPACE	others	ERROR	drop character

2 Grammar Rules

Raw Grammar:

symbol	rule
⟨tiger-program⟩	let ⟨declaration-segment⟩ in ⟨stat-seq⟩ end
⟨declaration-segment⟩	⟨type-declaration-list⟩ ⟨var-declaration-list⟩ ⟨funct-declaration-list⟩

⟨type-declaration-list⟩	⟨type-declaration⟩ ⟨type-declaration-list⟩
⟨type-declaration-list⟩	NULL
⟨var-declaration-list⟩	⟨var-declaration⟩ ⟨var-declaration-list⟩
⟨var-declaration-list⟩	NULL
⟨funct-declaration-list⟩	⟨funct-declaration⟩ ⟨funct-declaration-list⟩
⟨funct-declaration-list⟩	NULL
⟨type-declaration⟩	type id = ⟨type⟩ ;
⟨type⟩	⟨type-id⟩
⟨type⟩	array [INTLIT] of ⟨type-dim⟩ of ⟨type-id⟩
⟨type-dim⟩	[INTLIT] ⟨type-dim⟩
⟨type-dim⟩	NULL
⟨type-id⟩	int
⟨type-id⟩	string
⟨type-id⟩	id
⟨var-declaration⟩	var ⟨id-list⟩ : ⟨type-id⟩ ⟨optional-init⟩ ;
⟨id-list⟩	id
⟨id-list⟩	id , ⟨id-list⟩
⟨optional-init⟩	NULL
⟨optional-init⟩	:= ⟨const⟩
⟨funct-declaration⟩	function id (⟨param-list⟩) ⟨ret-type⟩ begin ⟨stat-seq⟩ end ;
⟨param-list⟩	NULL
⟨param-list⟩	⟨param⟩ ⟨param-list-tail⟩
⟨param-list-tail⟩	NULL
⟨param-list-tail⟩	, ⟨param⟩ ⟨param-list-tail⟩
⟨ret-type⟩	NULL
⟨ret-type⟩	: ⟨type-id⟩
⟨param⟩	id : ⟨type-id⟩
⟨stat-seq⟩	⟨stat⟩ ⟨stat-seq⟩
⟨stat-seq⟩	⟨stat⟩
⟨stat⟩	⟨lvalue⟩ := ⟨expr⟩ ;
⟨stat⟩	if ⟨expr⟩ then ⟨stat-seq⟩ endif ;
⟨stat⟩	if ⟨expr⟩ then ⟨stat-seq⟩ else ⟨stat-seq⟩ endif ;
⟨stat⟩	while ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	for id := ⟨expr⟩ to ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	⟨opt-prefix⟩ id (⟨expr-list⟩) ;
⟨stat⟩	break ;
⟨stat⟩	return ⟨expr⟩ ;
⟨expr⟩	⟨expr⟩ ⟨binary-operator⟩ ⟨expr⟩
⟨expr⟩	⟨const⟩
⟨expr⟩	⟨lvalue⟩
⟨expr⟩	- ⟨expr⟩
⟨expr⟩	(⟨expr⟩)
⟨binary-operator⟩	*
⟨binary-operator⟩	/
⟨binary-operator⟩	+
⟨binary-operator⟩	-
⟨binary-operator⟩	=
⟨binary-operator⟩	<
⟨binary-operator⟩	>
⟨binary-operator⟩	<=
⟨binary-operator⟩	>=
⟨binary-operator⟩	<>
⟨binary-operator⟩	&

⟨binary-operator⟩	|
⟨binary-operator⟩	:=
⟨opt-prefix⟩	⟨lvalue⟩ :=
⟨opt-prefix⟩	NULL
⟨const⟩	INTLIT
⟨const⟩	STRLIT
⟨const⟩	nil
⟨expr-list⟩	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	NULL
⟨expr-list-tail⟩	, ⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list-tail⟩	NULL
⟨lvalue⟩	id ⟨lvalue-tail⟩
⟨lvalue-tail⟩	[⟨expr⟩] ⟨lvalue-tail⟩
⟨lvalue-tail⟩	NULL

Modified Grammar:

symbol	rule
⟨tiger-program⟩	let ⟨declaration-segment⟩ in ⟨stat-seq⟩ end
⟨declaration-segment⟩	⟨type-declaration-list⟩ ⟨var-declaration-list⟩ ⟨funct-declaration-list⟩
⟨type-declaration-list⟩	⟨type-declaration⟩ ⟨type-declaration-list⟩
⟨type-declaration-list⟩	NULL
⟨var-declaration-list⟩	⟨var-declaration⟩ ⟨var-declaration-list⟩
⟨var-declaration-list⟩	NULL
⟨funct-declaration-list⟩	⟨funct-declaration⟩ ⟨funct-declaration-list⟩
⟨funct-declaration-list⟩	NULL
⟨type-declaration⟩	type id = ⟨type⟩ ;
⟨var-declaration⟩	var ⟨id-list⟩ : ⟨type-id⟩ ⟨optional-init⟩ ;
⟨funct-declaration⟩	function id (⟨param-list⟩) ⟨ret-type⟩ begin ⟨stat-seq⟩ end ;
⟨type⟩	⟨type-id⟩
⟨type⟩	array [INTLIT] of ⟨type-dim⟩ of ⟨type-id⟩
⟨type-dim⟩	[INTLIT] ⟨type-dim⟩
⟨type-dim⟩	NULL
⟨type-id⟩	id
⟨id-list⟩	id ⟨id-list-tail⟩
⟨id-list-tail⟩	, id ⟨id-list-tail⟩
⟨id-list-tail⟩	NULL
⟨optional-init⟩	:= ⟨const⟩
⟨optional-init⟩	NULL
⟨param-list⟩	⟨param⟩ ⟨param-list-tail⟩
⟨param-list⟩	NULL
⟨param-list-tail⟩	, ⟨param⟩ ⟨param-list-tail⟩
⟨param-list-tail⟩	NULL
⟨ret-type⟩	: ⟨type-id⟩
⟨ret-type⟩	NULL
⟨param⟩	id : ⟨type-id⟩
⟨stat-seq⟩	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	NULL
⟨stat⟩	if ⟨expr⟩ then ⟨stat-seq⟩ ⟨stat-if-tail⟩
⟨stat⟩	while ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	for id := ⟨expr⟩ to ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	break ;

⟨stat⟩	return ⟨expr⟩ ;
⟨stat⟩	id ⟨stat-func-or-assign⟩
⟨stat-func-or-assign⟩	(⟨expr-list⟩) ;
⟨stat-func-or-assign⟩	⟨lvalue-tail⟩ := ⟨stat-assign⟩
⟨stat-if-tail⟩	else ⟨stat-seq⟩ endif ;
⟨stat-if-tail⟩	endif ;
⟨stat-assign⟩	- ⟨unaryminus⟩ ⟨stat-assign-tail⟩
⟨stat-assign⟩	(⟨expr⟩) ⟨stat-assign-tail⟩
⟨stat-assign⟩	⟨const⟩ ⟨stat-assign-tail⟩
⟨stat-assign⟩	id ⟨stat-assign-id⟩
⟨stat-assign-id⟩	(⟨expr-list⟩) ;
⟨stat-assign-id⟩	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-tail⟩	⟨expr-tail⟩
⟨stat-assign-tail⟩	⟨orexpr-tail⟩
⟨stat-assign-tail⟩	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	⟨compare-tail⟩
⟨stat-assign-tail⟩	⟨term-tail⟩
⟨expr⟩	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr-tail⟩	⟨orop⟩ ⟨orexpr⟩ ⟨expr-tail⟩
⟨expr-tail⟩	NULL
⟨orexpr⟩	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr-tail⟩	⟨andop⟩ ⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr-tail⟩	NULL
⟨andexpr⟩	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	NULL
⟨compare⟩	⟨term⟩ ⟨compare-tail⟩
⟨compare-tail⟩	⟨addop⟩ ⟨term⟩ ⟨compare-tail⟩
⟨compare-tail⟩	NULL
⟨term⟩	⟨factor⟩ ⟨term-tail⟩
⟨term-tail⟩	⟨mulop⟩ ⟨factor⟩ ⟨term-tail⟩
⟨term-tail⟩	NULL
⟨factor⟩	⟨unaryminus⟩
⟨factor⟩	- ⟨unaryminus⟩
⟨unaryminus⟩	(⟨expr⟩)
⟨unaryminus⟩	⟨const⟩
⟨unaryminus⟩	⟨lvalue⟩
⟨const⟩	INTLIT
⟨const⟩	STRLIT
⟨const⟩	nil
⟨orop⟩	|
⟨andop⟩	&
⟨compop⟩	=
⟨compop⟩	<>
⟨compop⟩	>
⟨compop⟩	<
⟨compop⟩	>=
⟨compop⟩	<=
⟨addop⟩	+
⟨addop⟩	-
⟨mulop⟩	*
⟨mulop⟩	/
⟨expr-list⟩	⟨expr⟩ ⟨expr-list-tail⟩

⟨expr-list⟩	NULL
⟨expr-list-tail⟩	, ⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list-tail⟩	NULL
⟨lvalue⟩	id ⟨lvalue-tail⟩
⟨lvalue-tail⟩	[⟨expr⟩] ⟨lvalue-tail⟩
⟨lvalue-tail⟩	NULL

non-terminal	first set
⟨lvalue-tail⟩	[, ϵ
⟨lvalue⟩	id
⟨expr-list-tail⟩	,, ϵ
⟨expr-list⟩	(, nil, STRLIT, INTLIT, id, -, ϵ
⟨mulop⟩	*, /
⟨addop⟩	+, -
⟨compop⟩	=, <, >, <=, >=, <>
⟨andop⟩	&
⟨orop⟩	|
⟨const⟩	nil, STRLIT, INTLIT
⟨unaryminus⟩	(, nil, STRLIT, INTLIT, id
⟨factor⟩	(, nil, STRLIT, INTLIT, id, -
⟨term-tail⟩	*, / , ϵ
⟨term⟩	(, nil, STRLIT, INTLIT, id, -
⟨compare-tail⟩	+, -, ϵ
⟨compare⟩	(, nil, STRLIT, INTLIT, id, -
⟨andexpr-tail⟩	=, <, >, <=, >=, <>, ϵ
⟨andexpr⟩	(, nil, STRLIT, INTLIT, id, -
⟨orexpr⟩	(, nil, STRLIT, INTLIT, id, -
⟨orexpr-tail⟩	& , ϵ
⟨expr⟩	(, nil, STRLIT, INTLIT, id, -
⟨expr-tail⟩	|, ϵ
⟨stat-assign⟩	id, -, (, nil, STRLIT, INTLIT
⟨stat-assign-id⟩	[, (, *, /, +, -, =, <, >, <=, >=, <>, & , |, ϵ
⟨stat-assign-tail⟩	*, / , +, -, =, <, >, <=, >=, <>, & , |, ϵ
⟨stat-if-tail⟩	else, endif
⟨stat-func-or-assign⟩	(, :=, [
⟨stat⟩	if, while, for, break, return, id
⟨stat-seq⟩	if, while, for, break, return, id
⟨stat-seq-tail⟩	if, while, for, break, return, id, ϵ
⟨param⟩	id
⟨ret-type⟩	:, ϵ
⟨param-list-tail⟩	,, ϵ
⟨param-list⟩	id, ϵ
⟨optional-init⟩	:=, ϵ
⟨id-list-tail⟩	,, ϵ
⟨id-list⟩	id
⟨type-id⟩	id
⟨type-dim⟩	[, ϵ
⟨type⟩	array, id
⟨funct-declaration⟩	function
⟨var-declaration⟩	var
⟨type-declaration⟩	type
⟨funct-declaration-list⟩	function, ϵ
⟨var-declaration-list⟩	var, ϵ

⟨type-declaration-list⟩	type, ϵ
⟨declaration-segment⟩	function, var, type, ϵ
⟨tiger-program⟩	let

Note, the dollar sign is the end of tokens symbol.

non-terminal	follow set
⟨lvalue-tail⟩	:=, *, / , +, -, =, <, >, <=, >=, <>, &
⟨lvalue-tail⟩	|, if, while, for, break, return, id, endif, end,), ,, ,], then, do, to, ;
⟨expr-list-tail⟩	)
⟨expr-list⟩	)
⟨expr-tail⟩	if, while, for, break, return, id, endif, end,), ,, ,], then, do, to, ;
⟨orexpr-tail⟩	if, while, for, break, return, id, endif, end, |,), ,, ,], then, do, to, ;
⟨andexpr-tail⟩	if, while, for, break, return, id, endif, end, & ,|,), ,, ,], then, do, to, ;
⟨compare-tail⟩	if, while, for, break, return, id, endif, end
⟨compare-tail⟩	& ,|,), ,, ,], then, do, to, ;, =, <, >, <=, >=, <>
⟨term-tail⟩	if, while, for, break, return, id, endif, end, & ,|,)
⟨term-tail⟩	, ,], then, do, to, ;, =, <, >, <=, >=, <>, +, -
⟨stat-assign-tail⟩	if, while, for, break, return, id, endif, end
⟨stat-assign-id⟩	if, while, for, break, return, id, endif, end
⟨stat-seq-tail⟩	endif, end
⟨ret-type⟩	begin
⟨param-list-tail⟩	)
⟨param-list⟩	)
⟨optional-init⟩	function, in
⟨id-list-tail⟩	:
⟨type-dim⟩	of
⟨funct-declaration-list⟩	in
⟨var-declaration-list⟩	function, in
⟨type-declaration-list⟩	var, function, in
⟨declaration-segment⟩	in

Note, if there is no corresponding rule, then that means a parser error is generated.

Note, errors are handled by dropping tokens until a valid token is found.

symbol	next token	rule
⟨addop⟩	+	
⟨addop⟩	-	
⟨andexpr⟩	(	
⟨andexpr⟩	nil	
⟨andexpr⟩	STRLIT	
⟨andexpr⟩	INTLIT	
⟨andexpr⟩	id	
⟨andexpr⟩	-	
⟨andexpr-tail⟩	=	
⟨andexpr-tail⟩	<	
⟨andexpr-tail⟩	>	
⟨andexpr-tail⟩	>=	
⟨andexpr-tail⟩	<=	
⟨andexpr-tail⟩	<>	
⟨andexpr-tail⟩	if	ϵ
⟨andexpr-tail⟩	while	ϵ

⟨andexpr-tail⟩	for	ϵ
⟨andexpr-tail⟩	break	ϵ
⟨andexpr-tail⟩	return	ϵ
⟨andexpr-tail⟩	id	ϵ
⟨andexpr-tail⟩	endif	ϵ
⟨andexpr-tail⟩	end	ϵ
⟨andexpr-tail⟩	&	ϵ
⟨andexpr-tail⟩	|	ϵ
⟨andexpr-tail⟩	)	ϵ
⟨andexpr-tail⟩	,	ϵ
⟨andexpr-tail⟩	]	ϵ
⟨andexpr-tail⟩	then	ϵ
⟨andexpr-tail⟩	do	ϵ
⟨andexpr-tail⟩	to	ϵ
⟨andexpr-tail⟩	;	ϵ
⟨andop⟩	&	
⟨compare⟩	(	
⟨compare⟩	nil	
⟨compare⟩	STRLIT	
⟨compare⟩	INTLIT	
⟨compare⟩	id	
⟨compare⟩	-	
⟨compare-tail⟩	+	
⟨compare-tail⟩	-	
⟨compare-tail⟩	if	ϵ
⟨compare-tail⟩	while	ϵ
⟨compare-tail⟩	for	ϵ
⟨compare-tail⟩	break	ϵ
⟨compare-tail⟩	return	ϵ
⟨compare-tail⟩	id	ϵ
⟨compare-tail⟩	endif	ϵ
⟨compare-tail⟩	end	ϵ
⟨compare-tail⟩	&	ϵ
⟨compare-tail⟩	|	ϵ
⟨compare-tail⟩	)	ϵ
⟨compare-tail⟩	,	ϵ
⟨compare-tail⟩	]	ϵ
⟨compare-tail⟩	then	ϵ
⟨compare-tail⟩	do	ϵ
⟨compare-tail⟩	to	ϵ
⟨compare-tail⟩	;	ϵ
⟨compare-tail⟩	=	ϵ
⟨compare-tail⟩	<	ϵ
⟨compare-tail⟩	>	ϵ
⟨compare-tail⟩	<=	ϵ
⟨compare-tail⟩	>=	ϵ
⟨compare-tail⟩	<>	ϵ
⟨compop⟩	=	
⟨compop⟩	<	
⟨compop⟩	>	
⟨compop⟩	<=	
⟨compop⟩	>=	
⟨compop⟩	<>	

⟨const⟩	nil	
⟨const⟩	STRLIT	
⟨const⟩	INTLIT	
⟨declaration-segment⟩	function	
⟨declaration-segment⟩	var	
⟨declaration-segment⟩	type	
⟨declaration-segment⟩	in	ϵ
⟨expr-list⟩	)	ϵ
⟨expr-list⟩	(	
⟨expr-list⟩	nil	
⟨expr-list⟩	STRLIT	
⟨expr-list⟩	INTLIT	
⟨expr-list⟩	id	
⟨expr-list⟩	-	
⟨expr-list-tail⟩	)	ϵ
⟨expr-list-tail⟩	,	
⟨expr⟩	(	
⟨expr⟩	nil	
⟨expr⟩	STRLIT	
⟨expr⟩	INTLIT	
⟨expr⟩	id	
⟨expr⟩	-	
⟨expr-tail⟩	|	
⟨expr-tail⟩	if	ϵ
⟨expr-tail⟩	while	ϵ
⟨expr-tail⟩	for	ϵ
⟨expr-tail⟩	break	ϵ
⟨expr-tail⟩	return	ϵ
⟨expr-tail⟩	id	ϵ
⟨expr-tail⟩	endif	ϵ
⟨expr-tail⟩	end	ϵ
⟨expr-tail⟩	)	ϵ
⟨expr-tail⟩	,	ϵ
⟨expr-tail⟩	]	ϵ
⟨expr-tail⟩	then	ϵ
⟨expr-tail⟩	do	ϵ
⟨expr-tail⟩	to	ϵ
⟨expr-tail⟩	;	ϵ
⟨factor⟩	(	
⟨factor⟩	nil	
⟨factor⟩	STRLIT	
⟨factor⟩	INTLIT	
⟨factor⟩	id	
⟨factor⟩	-	
⟨funct-declaration⟩	function	
⟨funct-declaration-list⟩	function	
⟨funct-declaration-list⟩	in	ϵ
⟨id-list⟩	id	
⟨id-list-tail⟩	:	ϵ
⟨id-list-tail⟩	,	
⟨lvalue⟩	id	
⟨lvalue-tail⟩	[	
⟨lvalue-tail⟩	:=	ϵ

⟨lvalue-tail⟩	*	ϵ
⟨lvalue-tail⟩	/	ϵ
⟨lvalue-tail⟩	+	ϵ
⟨lvalue-tail⟩	-	ϵ
⟨lvalue-tail⟩	=	ϵ
⟨lvalue-tail⟩	<	ϵ
⟨lvalue-tail⟩	>	ϵ
⟨lvalue-tail⟩	<=	ϵ
⟨lvalue-tail⟩	>=	ϵ
⟨lvalue-tail⟩	<>	ϵ
⟨lvalue-tail⟩	&	ϵ
⟨lvalue-tail⟩	|	ϵ
⟨lvalue-tail⟩	if	ϵ
⟨lvalue-tail⟩	while	ϵ
⟨lvalue-tail⟩	for	ϵ
⟨lvalue-tail⟩	break	ϵ
⟨lvalue-tail⟩	return	ϵ
⟨lvalue-tail⟩	id	ϵ
⟨lvalue-tail⟩	endif	ϵ
⟨lvalue-tail⟩	end	ϵ
⟨lvalue-tail⟩	)	ϵ
⟨lvalue-tail⟩	,	ϵ
⟨lvalue-tail⟩	]	ϵ
⟨lvalue-tail⟩	then	ϵ
⟨lvalue-tail⟩	do	ϵ
⟨lvalue-tail⟩	to	ϵ
⟨lvalue-tail⟩	;	ϵ
⟨mulop⟩	*	
⟨mulop⟩	/	
⟨optional-init⟩	:=	
⟨optional-init⟩	function	ϵ
⟨optional-init⟩	in	ϵ
⟨orexpr⟩	(	
⟨orexpr⟩	nil	
⟨orexpr⟩	STRLIT	
⟨orexpr⟩	INTLIT	
⟨orexpr⟩	id	
⟨orexpr⟩	-	
⟨orexpr-tail⟩	&	
⟨orexpr-tail⟩	if	ϵ
⟨orexpr-tail⟩	while	ϵ
⟨orexpr-tail⟩	for	ϵ
⟨orexpr-tail⟩	break	ϵ
⟨orexpr-tail⟩	return	ϵ
⟨orexpr-tail⟩	id	ϵ
⟨orexpr-tail⟩	endif	ϵ
⟨orexpr-tail⟩	end	ϵ
⟨orexpr-tail⟩	|	ϵ
⟨orexpr-tail⟩	)	ϵ
⟨orexpr-tail⟩	,	ϵ
⟨orexpr-tail⟩	]	ϵ
⟨orexpr-tail⟩	then	ϵ
⟨orexpr-tail⟩	do	ϵ

⟨orexpr-tail⟩	to	ϵ
⟨orexpr-tail⟩	;	ϵ
⟨orop⟩	|	
⟨param⟩	id	
⟨param-list⟩	)	ϵ
⟨param-list⟩	id	
⟨param-list-tail⟩	)	ϵ
⟨param-list-tail⟩	,	
⟨ret-type⟩	begin	ϵ
⟨ret-type⟩	:	
⟨stat-assign⟩	id	
⟨stat-assign⟩	-	
⟨stat-assign⟩	(	
⟨stat-assign⟩	nil	
⟨stat-assign⟩	STRLIT	
⟨stat-assign⟩	INTLIT	
⟨stat-assign-id⟩	end	ϵ
⟨stat-assign-id⟩	endif	ϵ
⟨stat-assign-id⟩	id	ϵ
⟨stat-assign-id⟩	return	ϵ
⟨stat-assign-id⟩	break	ϵ
⟨stat-assign-id⟩	for	ϵ
⟨stat-assign-id⟩	while	ϵ
⟨stat-assign-id⟩	if	ϵ
⟨stat-assign-id⟩	|	
⟨stat-assign-id⟩	&	
⟨stat-assign-id⟩	<>	
⟨stat-assign-id⟩	>=	
⟨stat-assign-id⟩	<=	
⟨stat-assign-id⟩	>	
⟨stat-assign-id⟩	<	
⟨stat-assign-id⟩	=	
⟨stat-assign-id⟩	-	
⟨stat-assign-id⟩	+	
⟨stat-assign-id⟩	/	
⟨stat-assign-id⟩	*	
⟨stat-assign-id⟩	(	
⟨stat-assign-id⟩	[	
⟨stat-assign-tail⟩	end	ϵ
⟨stat-assign-tail⟩	endif	ϵ
⟨stat-assign-tail⟩	id	ϵ
⟨stat-assign-tail⟩	return	ϵ
⟨stat-assign-tail⟩	break	ϵ
⟨stat-assign-tail⟩	for	ϵ
⟨stat-assign-tail⟩	while	ϵ
⟨stat-assign-tail⟩	if	ϵ
⟨stat-assign-tail⟩	|	
⟨stat-assign-tail⟩	&	
⟨stat-assign-tail⟩	<>	
⟨stat-assign-tail⟩	>=	
⟨stat-assign-tail⟩	<=	
⟨stat-assign-tail⟩	>	
⟨stat-assign-tail⟩	<	

⟨stat-assign-tail⟩	=	
⟨stat-assign-tail⟩	-	
⟨stat-assign-tail⟩	+	
⟨stat-assign-tail⟩	/	
⟨stat-assign-tail⟩	*	
⟨stat-func-or-assign⟩	(	
⟨stat-func-or-assign⟩	:=	
⟨stat-func-or-assign⟩	[	
⟨stat-if-tail⟩	else	
⟨stat-if-tail⟩	endif	
⟨stat⟩	if	
⟨stat⟩	while	
⟨stat⟩	for	
⟨stat⟩	break	
⟨stat⟩	return	
⟨stat⟩	id	
⟨stat-seq⟩	if	
⟨stat-seq⟩	while	
⟨stat-seq⟩	for	
⟨stat-seq⟩	break	
⟨stat-seq⟩	return	
⟨stat-seq⟩	id	
⟨stat-seq-tail⟩	endif	ϵ
⟨stat-seq-tail⟩	end	ϵ
⟨stat-seq-tail⟩	if	
⟨stat-seq-tail⟩	while	
⟨stat-seq-tail⟩	for	
⟨stat-seq-tail⟩	break	
⟨stat-seq-tail⟩	return	
⟨stat-seq-tail⟩	id	
⟨term⟩	-	
⟨term⟩	id	
⟨term⟩	INTLIT	
⟨term⟩	STRLIT	
⟨term⟩	nil	
⟨term⟩	(	
⟨term-tail⟩	*	
⟨term-tail⟩	/	
⟨term-tail⟩	)	ϵ
⟨term-tail⟩	|	ϵ
⟨term-tail⟩	&	ϵ
⟨term-tail⟩	end	ϵ
⟨term-tail⟩	endif	ϵ
⟨term-tail⟩	id	ϵ
⟨term-tail⟩	return	ϵ
⟨term-tail⟩	break	ϵ
⟨term-tail⟩	for	ϵ
⟨term-tail⟩	while	ϵ
⟨term-tail⟩	if	ϵ
⟨term-tail⟩	-	ϵ
⟨term-tail⟩	+	ϵ
⟨term-tail⟩	<>	ϵ
⟨term-tail⟩	>=	ϵ

⟨term-tail⟩	<=	ϵ
⟨term-tail⟩	>	ϵ
⟨term-tail⟩	<	ϵ
⟨term-tail⟩	=	ϵ
⟨term-tail⟩	;	ϵ
⟨term-tail⟩	to	ϵ
⟨term-tail⟩	do	ϵ
⟨term-tail⟩	then	ϵ
⟨term-tail⟩	]	ϵ
⟨term-tail⟩	,	ϵ
⟨tiger-program⟩	let	
⟨type⟩	array	
⟨type⟩	id	
⟨type-declaration-list⟩	type	
⟨type-declaration-list⟩	var	ϵ
⟨type-declaration-list⟩	function	ϵ
⟨type-declaration-list⟩	in	ϵ
⟨type-declaration⟩	type	
⟨type-dim⟩	[	
⟨type-dim⟩	of	ϵ
⟨type-id⟩	id	
⟨unaryminus⟩	(	
⟨unaryminus⟩	nil	
⟨unaryminus⟩	STRLIT	
⟨unaryminus⟩	INTLIT	
⟨unaryminus⟩	id	
⟨var-declaration-list⟩	function	ϵ
⟨var-declaration-list⟩	in	ϵ
⟨var-declaration-list⟩	var	
⟨var-declaration⟩	var	