CS 4240 Phase 1

David Zhang Matheus Smythe Svolenski

February 23, 2014

The lexer/parser is run by compiling all the Java files and running the General.Runner class. The program takes in one argument for the tiger file, which must have a .tiger extension. The program outputs two files programname.tokens and programname.tokens.err, the first is a list of tokens. The second is any error output if it exists.

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	\	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	$\langle$	LANGLE	
START	$\rangle$	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	$\rangle$	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	^	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[\]^_ .	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

Below is the DFA table without any backtracking.

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	⟩	START	NOTEQ
LANGLE	=	START	LESSEQ
LANGLE	+	START	LESS , PLUS
LANGLE	-	START	LESS , MIN
LANGLE	*	START	LESS , MULT
LANGLE	/	START	LESS , DIV
LANGLE	(	START	LESS , LPAREN
LANGLE	)	START	LESS , RPAREN
LANGLE	,	START	LESS , COMMA
LANGLE	&	START	LESS , AND
LANGLE	|	START	LESS , OR
LANGLE	[	START	LESS , LSQUARE
LANGLE	]	START	LESS , RSQUARE
LANGLE	;	START	LESS , SEMI
LANGLE	⟨	LANGLE	
LANGLE	:	COLON	
LANGLE	0-9	INT-LIT	
LANGLE	“	STRING-LIT	
LANGLE	a-zA-Z	ID	
LANGLE	whitespace	START	LESS , ignore
LANGLE	others	ERROR	LESS , drop character
RANGLE	=	START	GREATEREQ
RANGLE	+	START	GREATER, PLUS
RANGLE	-	START	GREATER, MIN
RANGLE	*	START	GREATER, MULT
RANGLE	/	START	GREATER, DIV
RANGLE	(	START	GREATER, LPAREN
RANGLE	)	START	GREATER, RPAREN
RANGLE	,	START	GREATER, COMMA
RANGLE	&	START	GREATER, AND
RANGLE	|	START	GREATER, OR

RANGLE	[	START	GREATER, LSQUARE
RANGLE	]	START	GREATER, RSQUARE
RANGLE	;	START	GREATER, SEMI
RANGLE	⟨	LANGLE	GREATER
RANGLE	⟩	RANGLE	GREATER
RANGLE	:	COLON	GREATER
RANGLE	0-9	INT-LIT	GREATER
RANGLE	“	STRING-LIT	GREATER
RANGLE	a-zA-Z	ID	GREATER
RANGLE	whitespace	START	GREATER, ignore
RANGLE	others	ERROR	GREATER, drop character
COLON	=	START	ASSIGN
COLON	+	START	COLON, PLUS
COLON	-	START	COLON, MIN
COLON	*	START	COLON, MULT
COLON	/	START	COLON, DIV
COLON	(	START	COLON, LPAREN
COLON	)	START	COLON, RPAREN
COLON	,	START	COLON, COMMA
COLON	&	START	COLON, AND
COLON	|	START	COLON, OR
COLON	[	START	COLON, LSQUARE
COLON	]	START	COLON, RSQUARE
COLON	;	START	COLON, SEMI
COLON	⟨	LANGLE	COLON
COLON	⟩	RANGLE	COLON
COLON	:	COLON	COLON
COLON	0-9	INT-LIT	COLON
COLON	“	STRING-LIT	COLON
COLON	a-zA-Z	ID	COLON
COLON	whitespace	START	COLON, ignore
COLON	others	ERROR	COLON, drop character
INT-LIT	0-9	INT-LIT	
INT-LIT	+	START	INT-LIT, PLUS
INT-LIT	-	START	INT-LIT, MIN
INT-LIT	*	START	INT-LIT, MULT
INT-LIT	/	START	INT-LIT, DIV
INT-LIT	=	START	INT-LIT, EQ
INT-LIT	(	START	INT-LIT, LPAREN
INT-LIT	)	START	INT-LIT, RPAREN
INT-LIT	,	START	INT-LIT, COMMA
INT-LIT	&	START	INT-LIT, AND
INT-LIT	|	START	INT-LIT, OR
INT-LIT	[	START	INT-LIT, LSQUARE
INT-LIT	]	START	INT-LIT, RSQUARE
INT-LIT	;	START	INT-LIT, SEMI
INT-LIT	⟨	LANGLE	INT-LIT,
INT-LIT	⟩	RANGLE	INT-LIT,
INT-LIT	:	COLON	INT-LIT,
INT-LIT	“	STRING-LIT	INT-LIT,
INT-LIT	a-zA-Z	ID	INT-LIT,
INT-LIT	whitespace	START	INT-LIT, ignore
INT-LIT	others	ERROR	INT-LIT, drop character

ID	a-zA-Z0-9_	ID	
ID	+	START	ID, PLUS
ID	-	START	ID, MIN
ID	*	START	ID, MULT
ID	/	START	ID, DIV
ID	=	START	ID, EQ
ID	(	START	ID, LPAREN
ID	)	START	ID, RPAREN
ID	,	START	ID, COMMA
ID	&	START	ID, AND
ID	|	START	ID, OR
ID	[	START	ID, LSQUARE
ID	]	START	ID, RSQUARE
ID	;	START	ID, SEMI
ID	⟨	LANGLE	ID
ID	⟩	RANGLE	ID
ID	:	COLON	ID
ID	“	STRING-LIT	ID
ID	whitespace	START	ID, ignore
ID	others	ERROR	ID, drop character
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	^	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[\]^_ .	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

Given the raw grammar for the Tiger Language, provided in the Tiger Language Reference Manual, and shown below, we have generated a new grammar, by modifying it in order to ensure that it is not ambiguous, by enforcing operator precedences and left associativity, and to ensure that the grammar supports LL(1) parsing, by removing any left recursion and performing left factoring.

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] ⟨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

After performing the grammar's modifications, we came to the following 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] ⟨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

After ensuring that the new grammar meets all the requirements, the first and follow sets were generated for every non-terminal symbol of the grammar.

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

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

At last, the LL(1) parser table for Tiger was generated, as shown below.

Note that if there is no corresponding rule, then that means a parser error is generated. Also note that errors are handled by dropping tokens until a valid token is found.

symbol	next token	rule
⟨addop⟩	+	+
⟨addop⟩	-	-
⟨andexpr⟩	(	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr⟩	nil	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr⟩	STRLIT	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr⟩	INTLIT	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr⟩	id	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr⟩	-	⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	=	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	<	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	>	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	>=	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	<=	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨andexpr-tail⟩	<>	⟨compop⟩ ⟨compare⟩ ⟨andexpr-tail⟩
⟨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⟩	(	⟨term⟩ ⟨compare-tail⟩
⟨compare⟩	nil	⟨term⟩ ⟨compare-tail⟩
⟨compare⟩	STRLIT	⟨term⟩ ⟨compare-tail⟩
⟨compare⟩	INTLIT	⟨term⟩ ⟨compare-tail⟩
⟨compare⟩	id	⟨term⟩ ⟨compare-tail⟩
⟨compare⟩	-	⟨term⟩ ⟨compare-tail⟩
⟨compare-tail⟩	+	⟨addop⟩ ⟨term⟩ ⟨compare-tail⟩
⟨compare-tail⟩	-	⟨addop⟩ ⟨term⟩ ⟨compare-tail⟩
⟨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	nil
⟨const⟩	STRLIT	STRLIT
⟨const⟩	INTLIT	INTLIT
⟨declaration-segment⟩	function	⟨type-declaration-list⟩ ⟨var-declaration-list⟩ ⟨funct-declaration-list⟩
⟨declaration-segment⟩	var	⟨type-declaration-list⟩ ⟨var-declaration-list⟩ ⟨funct-declaration-list⟩
⟨declaration-segment⟩	type	⟨type-declaration-list⟩ ⟨var-declaration-list⟩ ⟨funct-declaration-list⟩
⟨declaration-segment⟩	in	ϵ
⟨expr-list⟩	)	ϵ
⟨expr-list⟩	(	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	nil	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	STRLIT	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	INTLIT	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	id	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list⟩	-	⟨expr⟩ ⟨expr-list-tail⟩
⟨expr-list-tail⟩	)	ϵ
⟨expr-list-tail⟩	,	, ⟨expr⟩ ⟨expr-list-tail⟩

⟨expr⟩	(	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr⟩	nil	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr⟩	STRLIT	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr⟩	INTLIT	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr⟩	id	⟨orexpr⟩ ⟨expr-tail⟩
⟨expr⟩	-	⟨orexpr⟩ ⟨expr-tail⟩
⟨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⟩	(	⟨unaryminus⟩
⟨factor⟩	nil	⟨unaryminus⟩
⟨factor⟩	STRLIT	⟨unaryminus⟩
⟨factor⟩	INTLIT	⟨unaryminus⟩
⟨factor⟩	id	⟨unaryminus⟩
⟨factor⟩	-	- ⟨unaryminus⟩
⟨funct-declaration⟩	function	function id (⟨param-list⟩) ⟨ret-type⟩ begin ⟨stat-seq⟩ end ;
⟨funct-declaration-list⟩	function	⟨funct-declaration⟩ ⟨funct-declaration-list⟩
⟨funct-declaration-list⟩	in	ϵ
⟨id-list⟩	id	id ⟨id-list-tail⟩
⟨id-list-tail⟩	:	ϵ
⟨id-list-tail⟩	,	, id ⟨id-list-tail⟩
⟨lvalue⟩	id	id ⟨lvalue-tail⟩
⟨lvalue-tail⟩	[	[⟨expr⟩] ⟨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⟩	)	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⟩	:=	:= ⟨const⟩
⟨optional-init⟩	;	ϵ
⟨orexpr⟩	(	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr⟩	nil	⟨andexpr⟩ ⟨orexpr-tail⟩

⟨orexpr⟩	STRLIT	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr⟩	INTLIT	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr⟩	id	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr⟩	-	⟨andexpr⟩ ⟨orexpr-tail⟩
⟨orexpr-tail⟩	&	⟨andop⟩ ⟨andexpr⟩ ⟨orexpr-tail⟩
⟨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	id : ⟨type-id⟩
⟨param-list⟩	)	ϵ
⟨param-list⟩	id	⟨param⟩ ⟨param-list-tail⟩
⟨param-list-tail⟩	)	ϵ
⟨param-list-tail⟩	,	, ⟨param⟩ ⟨param-list-tail⟩
⟨ret-type⟩	begin	ϵ
⟨ret-type⟩	:	: ⟨type-id⟩
⟨stat-assign⟩	id	id ⟨stat-assign-id⟩
⟨stat-assign⟩	-	- ⟨unaryminus⟩ ⟨stat-assign-tail⟩
⟨stat-assign⟩	(	(⟨expr⟩) ⟨stat-assign-tail⟩
⟨stat-assign⟩	nil	⟨const⟩ ⟨stat-assign-tail⟩
⟨stat-assign⟩	STRLIT	⟨const⟩ ⟨stat-assign-tail⟩
⟨stat-assign⟩	INTLIT	⟨const⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	;	ϵ
⟨stat-assign-id⟩	|	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	&	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	<>	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	>=	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	<=	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	>	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	<	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	=	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	-	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	+	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	/	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	*	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	[	⟨lvalue-tail⟩ ⟨stat-assign-tail⟩
⟨stat-assign-id⟩	(	(⟨expr-list⟩)
⟨stat-assign-tail⟩	;	ϵ
⟨stat-assign-tail⟩	|	⟨expr-tail⟩
⟨stat-assign-tail⟩	&	⟨orexpr-tail⟩
⟨stat-assign-tail⟩	<>	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	>=	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	<=	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	>	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	<	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	=	⟨andexpr-tail⟩
⟨stat-assign-tail⟩	-	⟨compare-tail⟩
⟨stat-assign-tail⟩	+	⟨compare-tail⟩

⟨stat-assign-tail⟩	/	⟨term-tail⟩
⟨stat-assign-tail⟩	*	⟨term-tail⟩
⟨stat-func-or-assign⟩	(	(⟨expr-list⟩) ;
⟨stat-func-or-assign⟩	:=	⟨lvalue-tail⟩ := ⟨stat-assign⟩ ;
⟨stat-func-or-assign⟩	[	⟨lvalue-tail⟩ := ⟨stat-assign⟩ ;
⟨stat-if-tail⟩	else	else ⟨stat-seq⟩ endif ;
⟨stat-if-tail⟩	endif	endif ;
⟨stat⟩	if	if ⟨expr⟩ then ⟨stat-seq⟩ ⟨stat-if-tail⟩
⟨stat⟩	while	while ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	for	for id := ⟨expr⟩ to ⟨expr⟩ do ⟨stat-seq⟩ enddo ;
⟨stat⟩	break	break ;
⟨stat⟩	return	return ⟨expr⟩ ;
⟨stat⟩	id	id ⟨stat-func-or-assign⟩
⟨stat-seq⟩	if	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq⟩	while	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq⟩	for	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq⟩	break	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq⟩	return	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq⟩	id	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	endif	ϵ
⟨stat-seq-tail⟩	end	ϵ
⟨stat-seq-tail⟩	enddo	ϵ
⟨stat-seq-tail⟩	else	ϵ
⟨stat-seq-tail⟩	if	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	while	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	for	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	break	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	return	⟨stat⟩ ⟨stat-seq-tail⟩
⟨stat-seq-tail⟩	id	⟨stat⟩ ⟨stat-seq-tail⟩
⟨term⟩	-	⟨factor⟩ ⟨term-tail⟩
⟨term⟩	id	⟨factor⟩ ⟨term-tail⟩
⟨term⟩	INTLIT	⟨factor⟩ ⟨term-tail⟩
⟨term⟩	STRLIT	⟨factor⟩ ⟨term-tail⟩
⟨term⟩	nil	⟨factor⟩ ⟨term-tail⟩
⟨term⟩	(	⟨factor⟩ ⟨term-tail⟩
⟨term-tail⟩	*	⟨mulop⟩ ⟨factor⟩ ⟨term-tail⟩
⟨term-tail⟩	/	⟨mulop⟩ ⟨factor⟩ ⟨term-tail⟩
⟨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⟩	<	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	let ⟨declaration-segment⟩ in ⟨stat-seq⟩ end
⟨type⟩	array	array [INTLIT] ⟨type-dim⟩ of ⟨type-id⟩
⟨type⟩	id	⟨type-id⟩
⟨type-declaration-list⟩	type	⟨type-declaration⟩ ⟨type-declaration-list⟩
⟨type-declaration-list⟩	var	ϵ
⟨type-declaration-list⟩	function	ϵ
⟨type-declaration-list⟩	in	ϵ
⟨type-declaration⟩	type	type id = ⟨type⟩ ;
⟨type-dim⟩	[	[INTLIT] ⟨type-dim⟩
⟨type-dim⟩	of	ϵ
⟨type-id⟩	id	id
⟨unaryminus⟩	(	(⟨expr⟩)
⟨unaryminus⟩	nil	⟨const⟩
⟨unaryminus⟩	STRLIT	⟨const⟩
⟨unaryminus⟩	INTLIT	⟨const⟩
⟨unaryminus⟩	id	⟨lvalue⟩
⟨var-declaration-list⟩	function	ϵ
⟨var-declaration-list⟩	in	ϵ
⟨var-declaration-list⟩	var	⟨var-declaration⟩ ⟨var-declaration-list⟩
⟨var-declaration⟩	var	var ⟨id-list⟩ : ⟨type-id⟩ ⟨optional-init⟩ ;