CS-428A Compiler Construction

Assignment-02
"Top-Down Parser"

Ameer Hamza Chahudhary - 212370057

Instructor: Nazifa Fatima
Last Updated: 2023-07-31

Contents

1 Grammar-01

assignment_question → question_body ? (1)

question_body → identifier EQ expression (2) | logical_expression (3) | arithmetic_expression (4)

expression → logical_expression (5) | arithmetic_expression (6)

logical_expression → logical_term logical_expression_tail (7)

logical_expression_tail → OR logical_term logical_expression_tail (8) | ϵ (9)

logical_term → logical_factor logical_term_tail (10)

logical_term_tail → AND logical_factor logical_term_tail (11) | ϵ (12)

logical_factor → NOT logical_atom (13) | logical_atom (14)

logical_atom → identifier (15) | TRUE (16) | FALSE (17) | (logical_expression) (18)

arithmetic_expression → term arithmetic_expression_tail (19)

arithmetic_expression_tail → PLUS term arithmetic_expression_tail (20) | MINUS term arithmetic_expression_tail (21) | ϵ (22)

term → factor term_tail (23)

term_tail → MULTIPLY factor term_tail (24) | DIVIDE factor term_tail (25) | ϵ (26)

factor → identifier (27) | number (28) | (arithmetic_expression) (29)

identifier → ID (30)

number → INTEGER (31) | FLOAT (32)

1.1 FIRST Sets

- FIRST(logical_expression_tail) = {OR, ϵ}
- FIRST(logical_term_tail) = {AND, ϵ}
- FIRST(logical_factor) = {NOT, TRUE, FALSE, (, ID}
- FIRST(logical_atom) = {TRUE, FALSE, (, ID} FIRST(arithmetic_expression_tail) = {PLUS, MINUS, ϵ}
- FIRST(term_tail) = {MULTIPLY, DIVIDE, ϵ}
- FIRST(factor) = {(, ID, INTEGER, FLOAT}
- FIRST(identifier) = {ID}
- FIRST(number) = {INTEGER, FLOAT}
- FIRST(question_body) = {ID, NOT, TRUE, FALSE, (, INTEGER, FLOAT}
- FIRST(logical_term) = {NOT, TRUE, FALSE, (, ID}
- FIRST(term) = {(, ID, INTEGER, FLOAT}
- FIRST(logical_expression) = {NOT, TRUE, FALSE, (, ID}
- FIRST(arithmetic_expression) = {(, ID, INTEGER, FLOAT}
- FIRST(expression) = {NOT, TRUE, FALSE, (, ID, INTEGER, FLOAT}
- FIRST(assignment_question) = {ID, NOT, TRUE, FALSE, (, INTEGER, FLOAT}

1.2 FOLLOW Sets

- FOLLOW(assignment_question) = {$}
- FOLLOW(question_body) = {?}
- FOLLOW(expression) = {?,), OR, AND}
- FOLLOW(logical_expression) = {), ?}

- FOLLOW(logical_expression_tail) = {), ?}
- FOLLOW(logical_term) = {), OR, ?}
- FOLLOW(logical_term_tail) = {), OR, ?}
- FOLLOW(logical_factor) = {), OR, AND, ?}
- FOLLOW(logical_atom) = {), OR, AND, ?}
- FOLLOW(arithmetic_expression) = {), ?}
- FOLLOW(arithmetic_expression_tail) = {), ?}
- FOLLOW(term) = {+, -,), ?}
- FOLLOW(term_tail) = {+, -,), ?}
- FOLLOW(factor) = {*, /, +, -,), ?}
- FOLLOW(identifier) = {=,* , /, +, -,), ?, AND, OR}
- FOLLOW(number) = {*,/,+,-,),?}

1.3 Parse Table

Non-terminal	?	ID	NOT	TRUE	FALSE	(	INTEGER
assignment_question	1						
question_body		2	3	3	3	3	3
expression							3
logical_expression		7	7	7	7	7	7
logical_expression_tail	8						
logical_term		10	10	10	10	10	10
logical_term_tail	11						
logical_factor		13	14	14	14	14	14
logical_atom		15		16	17	18	15
arithmetic_expression		19				29	19
arithmetic_expression_tail	20						
term		23				29	23
term_tail	24						
factor		27				29	27
identifier		30					
number		31					31

FLOAT	AND	OR	MULTIPLY	DIVIDE	PLUS	MINUS	)	$
3	3	3	3	3	3	3	3	3
3								
7		7					7	7
							9	9
10							10	10
	11	12					12	12
14		14					14	14
15		15					15	15
29							29	29
			20	21	20	21	22	22
23							23	23
	24	25	24	25	22	22	26	26
28							29	29
32								

1.4 LL(1) Grammar Check

Since there are no conflicts in the parse table, the given grammar is LL(1).

1.5 Predictive Parser Moves

Sample-String: **NOT ID AND TRUE ?**

Left-Most Derivation: assignment_question → question_body ? (1) → logical_expression ? (3) → logical_term logical_expression_tail ? (7) → logical_factor logical_term_tail logical_expression_tail ? (10) → NOT logical_atom logical_term_tail logical_expression_tail ? (13) → NOT identifier logical_term_tail logical_expression_tail ? (15) → NOT ID logical_term_tail logical_expression_tail ? (30) →NOT ID AND logical_factor logical_term_tail logical_expression_tail ? (11) → NOT ID AND logical_atom logical_expression_tail ? (14) → NOT ID AND TRUE logical_term_tail logical_expression_tail ? (16) → NOT ID AND TRUE logical_expression_tail ? (12) → **NOT ID AND TRUE ?** (9)

Moves by Predictive Parser:

Stack	Input
assignment_question$	NOT ID AND TRUE ?$
question_body ?$	NOT ID AND TRUE ?$
logical_expression ?$	NOT ID AND TRUE ?$
logical_term logical_expression_tail ?$	NOT ID AND TRUE ?$
logical_factor logical_term_tail logical_expression_tail ?$	NOT ID AND TRUE ?$
NOT logical_atom logical_term_tail logical_expression_tail ?$	NOT ID AND TRUE ?$
logical_atom logical_term_tail logical_expression_tail ?$	ID AND TRUE ?$
ID logical_term_tail logical_expression_tail ?$	ID AND TRUE ?$
logical_term_tail logical_expression_tail ?$	AND TRUE ?$
AND logical_factor logical_term_tail logical_expression_tail ?$	AND TRUE ?$
logical_factor logical_term_tail logical_expression_tail ?$	TRUE ?$
logical_atom logical_term_tail logical_expression_tail ?$	TRUE ?$
TRUE logical_term_tail logical_expression_tail ?$	TRUE ?$
logical_term_tail logical_expression_tail ?$	?$
logical_expression_tail ?$	?$
?$	?$
$	$

ACCEPTED!

2 Grammar-02

sentence → noun_phrase verb_phrase (1)

noun_phrase → determiner noun noun_phrase′ (2) | proper_noun (3) | pronoun noun_phrase′ (4)

noun_phrase′ → adjective noun noun_phrase′ (5) | ϵ (6)

verb_phrase → verb verb_phrase' [7] | verb adverb verb_phrase' [8] | verb noun_phrase verb_phrase' [9] | verb preposition noun_phrase verb_phrase' [10]

verb_phrase' → adverb verb_phrase' [11] | ϵ [12]

adjective → happy [13] | red [14] | big [15]

adverb → quickly [16] | carefully [17]

determiner → the [18] | a [19] | an [20]

preposition → in [21] | on [22] | under [23]

verb → run [24] | jump [25] | sing [26] | eat [27]

noun → dog [28] | cat [29] | apple [30] | table [31]

proper_noun → John [32] | London [33] | July [34]

pronoun → he [35] | she [36] | it [37] | they [38]

2.1 FIRST Sets

- FIRST(sentence) = {the, a, an, he, she, it, they, John, London, July}
- FIRST(noun_phrase) = {the, a, an, he, she, it, they, John, London, July}
- FIRST(noun_phrase') = {ϵ, happy, red, big}
- FIRST(verb_phrase) = {run, jump, sing, eat}
- FIRST(verb_phrase') = {ϵ, quickly, carefully}
- FIRST(adjective) = {happy, red, big}
- FIRST(adverb) = {quickly, carefully}
- FIRST(determiner) = {the, a, an}
- FIRST(preposition) = {in, on, under}
- FIRST(verb) = {run, jump, sing, eat}
- FIRST(noun) = {dog, cat, apple, table}
- FIRST(proper_noun) = {John, London, July}
- FIRST(pronoun) = {he, she, it, they}

2.2 FOLLOW Sets

- FOLLOW(sentence) = {$}
- FOLLOW(noun_phrase) = {quickly, carefully, run, jump, sing, eat, $}
- FOLLOW(noun_phrase') = {quickly, carefully, run, jump, sing, eat, }
- FOLLOW(verb_phrase) = FOLLOW(verb_phrase') = {$}
- FOLLOW(adjective) = {dog, cat, apple, table} FOLLOW(adverb) = {quickly, carefully, $}
- FOLLOW(determiner) = {dog, cat, apple, table}
- FOLLOW(preposition) = {the, a, an, he, she, it, they, John, London, July}
- FOLLOW(verb) = {quickly, carefully, the, a, an, he, she, it, they, John, London, July, in, on, under, $} FOLLOW(noun) = {happy, red, big, quickly, carefully, run, jump, sing, eat, $}
- FOLLOW(proper_noun) = {quickly, carefully, run, jump, sing, eat, $}
- FOLLOW(pronoun) = {happy, red, big, quickly, carefully, run, jump, sing, eat, $}

2.3 Parse Table

Non-terminal	the	a	an	he	she	it	they	John	London	July
sentence										
noun_phrase	2	2	2	5	5	5	5	32	33	34
noun_phrase′	6	6	6	6	6	6	6	6	6	6
verb_phrase										
verb_phrase′										
adjective										
adverb										
determiner	18	19	20							
preposition										
verb										
noun										
proper_noun								32	33	34
pronoun				35	36	37	38			

happy	red	big	quickly	carefully	run	jump
5	5	5	5	5	5	5
5	5	5	5	5	5	5
					24	25
					11	11
13	14	15				
			16	17		
					24	25

sing	eat	in	on	under	(	)	$
5	5						
5	5	6	6	6	6	6	
26	27						
11	11				12	12	
		21	22	23			
26	27						
					28	29	30

2.4 LL(1) Grammar Check

Since there are no conflicts in the parse table, the given grammar is LL(1).

2.5 Predictive Parser Moves

Sample-String: 'John run quickly'

Left-Most Derivation: sentence → noun_phrase verb_phrase (1) → proper_noun verb_phrase (3) → John verb_phrase (32) → John verb adverb verb_phrase′ (8) → John run adverb verb_phrase′ (24) → John run quicky verb_phrase′ (16) → John run quickly (12)

Moves by Predictive Parser:

Stack	Input
S$	John run quickly$
noun_phrase verb_phrase$	John run quickly$
proper_noun verb_phrase$	John run quickly$
John verb_phrase$	John run quickly$
verb adverb verb_phrase′$	run quickly$
run adverb verb_phrase′$	run quickly$
quickly verb_phrase′$	quickly$
$	$

ACCEPTED!

3 Grammar-03

program → decl_list$^{(1)}$

decl_list → declaration decl_list′$^{(2)}$

decl_list′ → declaration decl_list′$^{(3)}$ | $\epsilon^{(4)}$ declaration → var_decl$^{(5)}$ | func_decl$^{(6)}$ var_decl → type ID ;$^{(7)}$ type → int $^{(8)}$ | float$^{(9)}$ | char$^{(10)}$ func_decl → type ID (params) compound_stmt$^{(11)}$ params → param_list$^{(12)}$ | void$^{(13)}$ **param_list → param param_list′**$^{(14)}$

param_list′ → , param param_list′ $^{(15)}$ | $\epsilon^{(16)}$ param → type ID$^{(17)}$ compound_stmt → { local_decls stmt_list } $^{(18)}$ **local_decls → local_decls′** $^{(19)}$

local_decls′ → var_decl local_decls$^{(20)}$**′** | $\epsilon^{(21)}$ **stmt_list → stmt_list′**$^{(22)}$

stmt_list′ → stmt stmt_list′$^{(23)}$ | $\epsilon^{(24)}$ stmt → expr_stmt$^{(25)}$ | compound_stmt$^{(26)}$ | selection_stmt$^{(27)}$ | iteration_stmt$^{(28)}$ | return_stmt$^{(29)}$ expr_stmt → expression ;$^{(30)}$ | ; $^{(31)}$ expression → ID = expression $^{(32)}$ | simple_expression$^{(33)}$ simple_expression → additive_expression relop additive_expression$^{(34)}$ | additive_expression$^{(35)}$

additive_expression → term addtitive_expression′$^{(36)}$

additive_expression′ → addop term additive_expression′$^{(37)}$ | $\epsilon^{(38)}$

term → factor term′ $^{(39)}$

term′ → mulop factor term′$^{(40)}$ | $\epsilon^{(41)}$ factor → (expression) $^{(42)}$ | ID$^{(43)}$ | NUM$^{(44)}$ relop → < $^{(45)}$ | <=$^{(46)}$ | >$^{(47)}$ | >=$^{(48)}$ | ==$^{(49)}$ | !=$^{(50)}$

addop → +$^{(51)}$ | -$^{(52)}$ mulop → *$^{(53)}$ | /$^{(54)}$ selection_stmt → if (expression) stmt$^{(55)}$ | if (expression) stmt else stmt$^{(56)}$ iteration_stmt → while (expression) stmt$^{(57)}$ return_stmt → return expression ;$^{(58)}$

3.1 FIRST Sets

- FIRST(decl_list') = {ϵ, int, float, char}
- FIRST(type) = {int, float, char}
- FIRST(params) = {void, int, float, char}
- FIRST(param_list') = {,, ϵ}
- FIRST(compound_stmt) = {{}
- FIRST(local_decls) = {ϵ, int, float, char}
- FIRST(local_decls') = {ϵ, int, float, char}
- FIRST(stmt_list) = {ϵ, ;, ID, (, NUM, if, return, {, while}
- FIRST(stmt_list') = {ϵ, ;, ID, (, NUM, if, return, {, while}
- FIRST(expr_stmt) = {;, ID, (, NUM}
- FIRST(expression) = {ID, (, NUM}
- FIRST(additive_expression') = {ϵ, +, -}
- FIRST(term') = {ϵ, *, /}
- FIRST(factor) = {(, ID, NUM}
- FIRST(relop) = {<, <=, >, >=, ==, !=}
- FIRST(addop) = {+, -}
- FIRST(mulop) = {*, /}
- FIRST(selection_stmt) = {if}
- FIRST(iteration_stmt) = {while}
- FIRST(return_stmt) = {return}
- FIRST(var_decl) = {int, float, char}
- FIRST(func_decl) = {int, float, char}
- FIRST(param) = {int, float, char}
- FIRST(term) = {(, ID, NUM}
- FIRST(declaration) = {int, float, char}
- FIRST(param_list) = {int, float, char}
- FIRST(additive_expression) = {(, ID, NUM}
- FIRST(simple_expression) = {(, ID, NUM}
- FIRST(decl_list) = {int, float, char}
- FIRST(program) = {int, float, char}
- FIRST(stmt) = {;, ID, (, NUM, if, return, {, while}

3.2 FOLLOW Sets

- FOLLOW(program) = {$}
- FOLLOW(decl_list) = {$}
- FOLLOW(decl_list') = {$}
- FOLLOW(declaration) = {int, float, char, $}
- FOLLOW(var_decl) = {int, float, char, $, ;, ID, (, NUM, if, return, {, while}
- FOLLOW(type) = {ID} FOLLOW(func_decl) = {int, float, char, $} FOLLOW(params) = {)}
- FOLLOW(param_list) = {)}
- FOLLOW(param_list') = {)}
- FOLLOW(param) = {,,)}
- FOLLOW(compound_stmt) = {else, ;, ID, (, NUM, if, return, {, while, int, float, char, $, }}
- FOLLOW(local_decls) = {;, ID, (, NUM, if, return, {, while}
- FOLLOW(local_decls') = {;, ID, (, NUM, if, return, {, while}
- FOLLOW(stmt_list) = {}}
- FOLLOW(stmt_list') = {}}
- FOLLOW(stmt) = {else, ;, ID, (, NUM, if, return, {, while, }}
- FOLLOW(expr_stmt) = {else, ;, ID, (, NUM, if, return, {, while, }}
- FOLLOW(expression) = {;,)}

- FOLLOW(simple_expression) = {;,)}
- FOLLOW(additive_expression) = {<, <=, >, >=, ==, !=, ;,)}
- FOLLOW(additive_expression') = {<, <=, >, >=, ==, !=, ;,)}
- FOLLOW(term) = {+, -, <, <=, >, >=, ==, !=, ;,)}
- FOLLOW(term') = {+, -, <, <=, >, >=, ==, !=, ;,)}
- FOLLOW(factor) = {*, /, +, -, <, <=, >, >=, ==, !=, ;,)}
- FOLLOW(relop) = {(, ID, NUM}
- FOLLOW(addop) = {(, ID, NUM}
- FOLLOW(mulop) = {(, ID, NUM}
- FOLLOW(selection_stmt) = {else, ;, ID, (, NUM, if, return, {, while, }}
- FOLLOW(iteration_stmt) = {else, ;, ID, (, NUM, if, return, {, while, }}
- FOLLOW(return_stmt) = {else, ;, ID, (, NUM, if, return, {, while, }}

3.3 Parse Table

	int	float	char	void	ID	NUM	;	*
program								
decl_list	1	1	1	1				
decl_list'								
declaration	5	6	6	6				
var_decl	7	7	7					
type	8	9	10					
func_decl	11	11	11					
params	13	12	12					
param_list	14	14	14					
param_list'					15	16	16	
param	17	17	17					
compound_stmt								
local_decls	19	19	19					
local_decls'					21	21	21	
stmt_list								
stmt_list'								
stmt					25	25	26	
expr_stmt					30	30	30	
expression	33	33	33		33	33		
additive_expr'								
term	39	39	39		39	39		
term'					41	41	41	40
factor	42	43	44					
relop								
addop								
mulop								54
selection_stmt								
iteration_stmt								
return_stmt							58	
simple_expr					34	34		
add_expr'								

{	}	(	)	if	else	while	return	+	-

{	}	(	)	if	else	while	return	+	-
			13	13	13	13			
16	16	16							
18	19								
			21	21	20	20			
21	21	21							
			22	22	22	22			
			23	23	23	23			
27	28	28	27	27	28	29			
30	30	30	30	30	30	30			
		33			33	33			
			37	38	37	38			
		39			39	39			
41	41	41					40	40	40
							51	52	
							53	54	53
		55							
		57							
							37	38	

/	<	<=	>	>=	==	!=	=	$

/	<	<=	>	>=	==	!=	=	$
36	36	36	36	36	36	36		
41	41	41	41	41	41	41	41	
				45	46	47	48	49
36	36	36	36	36	36	36		

3.4 LL(1) Grammar Check

Since there are no conflicts in the parse table, the given grammar is LL(1).

3.5 Predictive Parser Moves

Sample-String: **float ID (float ID){ }**

Left-Most Derivation: program → decl_list (1) → declaration decl_list'(2) → func_decl decl_list' (6) → func_decl (4) → type ID (params) compound_stmt (11) → float ID (params) compound_stmt (9) → float ID (param param_list') compound_stmt (14) → float ID (type ID param_list') compound_stmt (17) → float ID (float ID param_list') compound_stmt (9) → float ID (float ID) compound_stmt (16) → float ID (float ID) { local_decls stmt_list } (18) → float ID (float ID) { local_decls' stmt_list } (19) → float ID (float ID) { stmt_list } (21) → float ID (float ID) { stmt_list' } (22) → **float ID (float ID) { }** (24)

Moves by Predictive Parser:

Stack	Input
program$	float ID (float ID){ }$
decl_list$	float ID (float ID){ }$
declaration decl_list'$	float ID (float ID){ }$
func_decl decl_list'$	float ID (float ID){ }$
func_decl$	float ID (float ID){ }$
type ID (params) compound_stmt$	float ID (float ID){ }$
float ID (params) compound_stmt$	float ID (float ID){ }$$
ID (params) compound_stmt$	ID (float ID){ }$
(param param_list') compound_stmt$	(float ID){ }$
type ID param_list') compound_stmt$	float ID (float ID){ }$
float ID param_list') compound_stmt$	float ID (float ID){ }$
ID param_list') compound_stmt$	ID){ }$
) compound_stmt$	){ }$
compound_stmt$	{ }$
{ local_decls stmt_list }$	{ }$
local_decls' stmt_list }$	}$
stmt_list }$	}$
}$	}$

Stack	Input
$	$

ACCEPTED!

4 Grammar-04

$S \rightarrow AaBbCc^{(1)} \mid dDeEfFgG^{(2)} \mid hH^{(3)} \mid \epsilon^{(4)}$
$A \rightarrow aA^{(5)} \mid \epsilon^{(6)}$
$B \rightarrow bB^{(7)} \mid \epsilon^{(8)}$
$C \rightarrow cC^{(9)} \mid \epsilon^{(10)}$
$D \rightarrow dD^{(11)} \mid \epsilon^{(12)}$
$E \rightarrow eE^{(13)} \mid \epsilon^{(14)}$
$F \rightarrow fF^{(15)} \mid \epsilon^{(16)}$
$G \rightarrow gG^{(17)} \mid \epsilon^{(18)}$
$H \rightarrow hH^{(19)} \mid \epsilon^{(20)}$

4.1 FIRST Sets

- FIRST(S) = {a, d, h, ϵ}
- FIRST(A) = {a, ϵ}
- FIRST(B) = {b, ϵ}
- FIRST(C) = {c, ϵ}
- FIRST(D) = {d, ϵ}
- FIRST(E) = {e, ϵ}
- FIRST(F) = {f, ϵ}
- FIRST(G) = {g, ϵ}
- FIRST(H) = {h, ϵ}

4.2 FOLLOW Sets

- FOLLOW(S) = {$}
- FOLLOW(A) = {a}
- FOLLOW(B) = {b}
- FOLLOW(C) = {c}
- FOLLOW(D) = {e}
- FOLLOW(E) = {f}
- FOLLOW(F) = {g}
- FOLLOW(G) = {$}
- FOLLOW(H) = {$}

4.3 Parse Table

Parse Table	a	b	c	d	e	f	g	h	$
S	1	1	1	2				3	
A	5			6					
B		7			8				8

Parse Table	a	b	c	d	e	f	g	h	$
C			9			10			10
D				11					
E					13				
F						15			
G							17		
H								19	

4.4 LL(1) Grammar Check

Since there are no conflicts in the parse table, the given grammar is LL(1).

4.5 Predictive Parser Moves

Sample-String: '**ddefg**'

Left-Most Derivation: S → dDeEfFgG (2) → ddDeEfFgG (11) → ddeEfFgG (12) → ddefFgG (14) → ddefgG (16) → **ddefg** (18)

Moves by Predictive Parser:

Stack	Input
S$	ddefg$
dDeEfFgG$	ddefg$
dDeEfFgG$	defg$
eEfFgG$	efg$
fFgG$	fg$
gG$	g$
$	$

ACCEPTED!

5 Grammer-05

S → aA | Bb | cC A → d | ϵ B → eB | f C → gC | h | ϵ

5.1 FIRST Sets

- FIRST(S) = {a, c}
- FIRST(A) = {d, ϵ}
- FIRST(B) = {e, f}
- FIRST(C) = {g, h, ϵ}

5.2 FOLLOW Sets

- FOLLOW(S) = {$}
- FOLLOW(A) = {$}
- FOLLOW(B) = {b}
- FOLLOW(C) = {$}

5.3 Parse Table

Parse Table	a	b	c	d	e	f	g	h	$
S	S -> aA	S -> Bb	S -> cC						
A	A -> d			A -> ϵ					A -> ϵ
B		B -> f			B -> eB				
C							C -> gC	C -> h	C -> ϵ

5.4 LL(1) Grammar Check

Since there are no conflicts in the parse table, the given grammar is LL(1).

5.5 Predictive Parser Moves

Sample String: **cggh**

Left-Most Derivation:

S $\rightarrow$ cC (3) $\rightarrow$ cgC (8)$\rightarrow$ cggC (8) $\rightarrow$ **cggh** (9)

Moves by Predictive Parser:

Stack	Input
S$	cggh$
cC$	cggh$
gC$	ggh$
gC$	gh$
h$	h$
$	$

ACCPETED!