Adalynn Everly
Mason Berry
testing-team-04
Software Testing
Deliverable 2 - Test Cases

Unit Test Cases Table

Function	Test Path	Inputs	Expected Output
open_character_stream	[start, 1, 3, 4, end]	File with content "test content"	BufferedReader that can read "test content"
open_character_stream	[start, 1, 3, 4, end]	"nonexistentfile.txt"	FileNotFoundException
open_character_stream	[start, 1, 2, 4, end]	null	Buffered reader exists and reads from system.in
get_char	[start, 1, end]	BufferedReader with "abc"	'a' (97)
get_char	[start, 1, end]	BufferedReader with "abc" (fourth call - EOF)	-1
get_char	[start, 1, end]	BufferedReader that throws IOException	0
unget_char	[start, 1, end]	ch = 'a', BufferedReader with "ab"	0
open_token_stream	[start, 1, 2, 4, end]	null	BufferedReader (not null) Returns br from call to open_character_stream with null which opens br for system.in
open_token_stream	[start, 1, 2, 4, end]	""	Same as above
open_token_stream	[start, 1, 3, 4, end]	Valid file path fname = "test.txt"	BufferedReader (not null) Returns br for test.txt

get_token	[start, 1, 2, 3, 4, end] EOF immediately	BufferedReader with "hello world 123" (fourth call - EOF)	null
get_token	[start, 1, 2, 3, 5, 6, 7, 8, 6, 7, 8, 6, 7, 8, 6, 7, 8, 6, 9, 10, end] Whitespace loop into EOF	BufferedReader with " \n\r token \n\r " (second call)	null
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 13, end] Single special char	BufferedReader with "( ) [ ] , ` '" (first call)	"("
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 15, 16, 18, 19, 22, 23, 24, 25, 26, 27 (loop (23, 24, 25, 26, 27) until **is_token_end** returns true from \"), 23, 28, 31, 34, 35, 36, 37, end) Token is a string	BufferedReader with "\"hello world\"""	“hello world”
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 15, 16, 18, 19, 22, 23, 24, 25, 26, 27 (loop (23, 24, 25, 26, 27) until **is_token_end** reads ‘\n’and returns true), 23, 28, 31, 34, 35, 37, end] String without closing quotation, followed by a newline	BufferedReader with "\"unterminated string\n"	"unterminated string
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, (loop 23, 24, 25, 26, 27 until get_char returns -1), 26, 26.5, 28, 29, 30, end] Comment token	BufferedReader with ";comment"	;comment

get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 16, 18, 19, 22, 23, 24, 25, 26, 27 (loop (23, 24, 25, 26, 27) until **is_token_end** returns true from ' ') 23, 28, 31, 34, 38, 41, end] Standard token followed by white space	BufferedReader with "hello world 123" (first call)	hello
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 16, 18, 19, 22, 23, 24, 25, 26, 27 (loop (23, 24, 25, 26, 27) until **is_token_end** returns true from ')') 23, 28, 31, 32, 33, end] Standard token followed by special char	BufferedReader with "abc)"	abc
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 16, 18, 19, 22, 23, 24, 25, 26, 27 (loop (23, 24, 25, 26, 27) until **is_token_end** returns true from ';') 23, 28, 31, 34, 38, 39, 40, end] Standard token followed by semicolon	BufferedReader with "word;"	word
get_token	[start, 1, 2, 3, 5, 6, 9, 11, 12, 14, 16, 17, 18, 19, 20, 21, end] Single semicolon followed by EOF	BufferedReader with ";"	;
is_token_end	[start, 1, 2, end]	str_com_id = 0, res = -1	true
is_token_end	[start, 1, 3, 4, 5, end]	str_com_id = 1, res = ""	true
is_token_end	[start, 1, 3, 4, 6, end]	str_com_id = 1, res = 'a'	false
is_token_end	[start, 1, 3, 7, 8, 9, end]	str_com_id = 2, res = '\n'	true
is_token_end	[start, 1, 3, 7, 8, 10, end]	str_com_id = 2, res = 'a'	false

is_token_end	[start, 1, 3, 7, 11, 12, end]	str_com_id = 0, res = '('	true
is_token_end	[start, 1, 3, 7, 11, 13, 14, end]	str_com_id = 0, res = ' '	true
is_token_end	[start, 1, 3, 7, 11, 13, 15, end] (Path is also covered every time a get_token test builds a token with consecutive chars)	str_com_id = 0, res = ‘a’ (semicolon)	false
token_type	[start, 1, 2, end]	"and"	1
token_type	[start, 1, 3, 4, end]	"("	2
token_type	[start, 1, 3, 5, 6, end]	"variable"	3
token_type	[start, 1, 3, 5, 7, 8, end]	“22”	41
token_type	[start, 1, 3, 5, 7, 9,10, end]	""hello""	42
token_type	[start, 1, 3, 5, 7, 9, 11, 12, end]	"#a"	43
token_type	[start, 1, 3, 5, 7, 9, 11, 13, 14, end]	“;justAComment”	5
token_type	[start, 1, 3, 5, 7, 9, 11, 13, 15, end]	"@"	0
print_token	[start, 1, 2, 3, 4, 5, 7, 9, 11, 13, 15, 17, end]	"@invalid"	error,"@invalid".
print_token	[start, 1, 2, 3, 5, 6, 7, 9, 11, 13, 15, 17, end]	"and"	keyword,"and".
print_token	[start, 1, 2, 3, 5, 7, 8, 9, 11, 13, 15, 17, end]	"("	lparen.
print_token	[start, 1, 2, 3, 5, 7, 9, 10, 11, 13, 15, 17, end]	"variable"	identifier,"variable".
print_token	[start, 1, 2, 3, 5, 7, 9, 11, 12, 13, 15, 17, end]	"1"	numeric,1.

print_token	[start, 1, 2, 3, 5, 7, 9, 11, 13, 14, 15, 17, end]	""hello""	string,"hello".
print_token	[start, 1, 2, 3, 5, 7, 9, 11, 13, 15, 16, 17, end]	"#a"	character,"a".
print_token	[start, 1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 18, end]	";comment"	comment,";comment".
is_comment	[start, 1, 2, end]	";comment"	true
is_comment	[start, 1, 3, end]	"not_comment"	false
is_keyword	[start, 1, 2, end]	"and"	true
is_keyword	[start, 1, 3, end]	"not_keyword"	false
is_char_constant	[start, 1, 2, end]	"#a"	true
is_char_constant	[start, 1, 3, end]	"#1"	false
is_num_constant	[start, 1, 2, 6, end]	"1"	true
is_num_constant	[start, 1, 7, end]	"abc"	false
is_num_constant	[start, 1, 2, 3, 4, 2, 3, 4, 2, 6, end]	"123"	true
is_num_constant	[start, 1, 2, 3, 5, end]	"1a"	false
is_str_constant	[start, 1, 2, 3, 5, (loop (2, 3, 5) until charAa(i) == ““), 3, 4, end]	""valid""	true
is_str_constant	[start, 1, 7, end]	"not a string"	false
is_str_constant	[start, 1, 2, 3, 5, (loop (2, 3, 5) until !(i < str.length)), 2, 6, end]	""unterminated"	false
is_identifier	[start, 1, 2, 3, 4, (loops (2, 3, 4) until end of token) 2, 6, end]	"validVar"	true
is_identifier	[start, 1, 7, end]	"123abc"	false

is_identifier	[start, 1, 2, 3, 4, 2, 3, 5, end]	"a#"	false
print_spec_symbol	[start, 1, 2, end]	"("	lparen.
print_spec_symbol	[start, 1, 3, 4, end]	")"	rparen.
print_spec_symbol	[start, 1, 3, 5, 6, end]	"["	lsquare.
print_spec_symbol	[start, 1, 3, 5, 7, 8, end]	"]"	rsquare.
print_spec_symbol	[start, 1, 3, 5, 7, 9, 10, end]	"""	quote.
print_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 12, end]	"`"	bquote.
print_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 13, 14, end]	","	comma.
print_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 13, end]	"@"	(no output)
is_spec_symbol	[start, 1, 2, end]	'('	true
is_spec_symbol	[start, 1, 3, 4, end]	')'	true
is_spec_symbol	[start, 1, 3, 5, 6, end]	'['	true
is_spec_symbol	[start, 1, 3, 5, 7, 8, end]	']'	true
is_spec_symbol	[start, 1, 3, 5, 7, 9, 10, end]	"'"	true
is_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 12, end]	'`'	true
is_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 13, 14, end]	','	true
is_spec_symbol	[start, 1, 3, 5, 7, 9, 11, 13, 15, end]	'a'	false

End-To-End Test Cases

Test 1: testMainWithValidFile()
Input: File containing "and (variable) 123 "string" #a ;comment"
Expected Output:
keyword,"and".
lparen.
identifier,"variable".
rparen.
numeric,123.
string,"string".
character,"a".

comment,";comment".

main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1 -> is_keyword{1,2}, 2}, 5}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,2}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,2}, 4}, 7 -> print_spec_symbol{1,2}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5 -> is_identifier{1,2,3,4,5,6,7}, 6}, 9}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,2}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,2}, 4}, 7 -> print_spec_symbol{3,4}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7 -> is_num_constant{1,2,3,4,6}, 8}, 11}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 14,15,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,4,6}, 34,35,36,37,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9 -> is_str_constant{1,2,3,4,6}, 10}, 13}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11 -> is_char_constant{1,2}, 12}, 15}, 10 -> get_token{1,2,3,5,6,16,17,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,7,8,9}, 24,25,26,27,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13 -> is_comment{1,2}, 14}, 17}, 10 -> get_token{1,2,3,4}, 10]

Test 2: testMainWithNonexistentFile()
Input: "nonexistent_file.txt" **Expected Output:** Error message containing "doesn't exists" or NullPointerException
main[1,3,4,5,6 -> open_token_stream{1,3 -> open_character_stream{1,3 -> FileNotFoundException caught, exception printed, null returned}}, 7,8,9 -> get_token{NullPointerException or returns null immediately}, 10]

Test 3: testMainWithTooManyArgumentsErrorMessage()
Input: ["file1.txt", "file2.txt"] **Expected Output:** "Error! Please give the token stream"

main[1,5]

Test 4: testMainWithEmptyFile()
Input: Empty file "" **Expected Output:** No output (empty)
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,4}, 10]

Test 5: testTokenEndingWithEOF()
Input: File containing "token" (no newline) **Expected Output:** "identifier,"token"."
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5 -> is_identifier{1,2,3,4,5,6,7}, 6}, 9}, 10 -> get_token{1,2,3,4}, 10]

Test 6: testUnterminatedString()
Input: ""unterminated string\n" **Expected Output:** Contains "unterminated string"
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 14,15,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,5,6}, 34,35,36,37,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9 -> is_str_constant{1,2,3,5,7}, 10}, 13}, 10 -> get_token{1,2,3,4}, 10]

Test 7: testTokenFollowedBySpecialSymbol()
Input: "word(" **Expected Output:** "identifier,"word"." and "lparen."
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,4,11 -> is_spec_symbol{1,2}, 12}, 31,32,33,41}, 9 -> print_token{1,2 -> token_type{1,3,5 -> is_identifier{1,2,3,4,5,6,7}, 6}, 9}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,2}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,2}, 4}, 7 -> print_spec_symbol{1,2}}, 10 -> get_token{1,2,3,4}, 10]

Test 8: testTokenFollowedBySemicolon()
Input: "word;" **Expected Output:** "identifier,"word"." and "comment,";"."
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,4,13,14}, 38,39,40,41}, 9 -> print_token{1,2 -> token_type{1,3,5 -> is_identifier{1,2,3,4,5,6,7}, 6}, 9}, 10 -> get_token{1,2,3,5,6,16,17,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,7,8,9}, 24,25,26,27,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13 -> is_comment{1,2}, 14}, 17}, 10 -> get_token{1,2,3,4}, 10]

Test 9: testErrorTokenTypes()
Input: "@invalid #1 123abc" **Expected Output:** Multiple "error," outputs
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13,15}, 3}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11 -> is_char_constant{1,3}, 12,13,15}, 3}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7 -> is_num_constant{1,2,3,5,7}, 8,9,11,13,15}, 3}, 10 -> get_token{1,2,3,4}, 10]

Test 10: testWhitespaceSkipping()
Input: " \n\r token \n\r " **Expected Output:** "identifier,"token"."
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1,3,5 -> is_identifier{1,2,3,4,5,6,7}, 6}, 9}, 10 -> get_token{1,2,3,4}, 10]

Test 11: testAllSpecialSymbols()
Input: "( ) [ ] ' ` ," **Expected Output:** "lparen.", "rparen.", "lsquare.", "rsquare.", "quote.", "bquote.", "comma."
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,2}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,2}, 4}, 7 -> print_spec_symbol{1,2}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,4}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,4}, 4}, 7 -> print_spec_symbol{3,4}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,6}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,6}, 4}, 7 -> print_spec_symbol{5,6}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,8}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,8}, 4}, 7 -> print_spec_symbol{7,8}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,10}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,10}, 4}, 7 -> print_spec_symbol{9,10}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,12}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,12}, 4}, 7 -> print_spec_symbol{11,12}}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,14}, 13}, 9 -> print_token{1,2 -> token_type{1,3 -> is_spec_symbol{1,14}, 4}, 7 -> print_spec_symbol{13,14}}, 10 -> get_token{1,2,3,4}, 10]

Test 12: testCommentVariations()
Input: ";comment ending with tab\t\n;comment ending with return\r\n;comment ending with newline\n" **Expected Output:** Multiple "comment," outputs
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,16,17,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,7,8,9}, 24,25,26,27,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13 -> is_comment{1,2}, 14}, 17}, 10 -> get_token{1,2,3,5,6,16,17,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,7,8,9}, 24,25,26,27,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13 -> is_comment{1,2}, 14}, 17}, 10 -> get_token{1,2,3,5,6,16,17,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,7,8,9}, 24,25,26,27,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9,11,13 -> is_comment{1,2}, 14}, 17}, 10 -> get_token{1,2,3,4}, 10]

Test 13: testStringVariations()
Input: ""string_with_tab\t" "string_with_return\r" "normal_string"" **Expected Output:** Multiple "string," outputs
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 14,15,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,8,9}, 24,25,26,27,34,35,36,37,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9 -> is_str_constant{1,2,3,4,6}, 10}, 13}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 14,15,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,8,9}, 24,25,26,27,34,35,36,37,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9 -> is_str_constant{1,2,3,4,6}, 10}, 13}, 10 ->

get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 14,15,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,3,4,6}, 34,35,36,37,41}, 9 -> print_token{1,2 -> token_type{1,3,5,7,9 -> is_str_constant{1,2,3,4,6}, 10}, 13}, 10 -> get_token{1,2,3,4}, 10]

Test 14: testMainWithNoArgumentsArray()
Input: Empty args array [] **Expected Output:** Depends on stdin input
main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{varies based on stdin}, 9 -> print_token{varies based on token type}, 10 -> get_token{continues until stdin EOF}, 10]

Test 15: testIsIdentifierStartsWithNonLetter()

Input: "1abc 9xyz #invalid @symbol" **Expected Output:**

error,"1abc".
error,"9xyz".
error,"#invalid".

error,"@symbol".

main[1,2,4,5,6 -> open_token_stream{1,2 -> open_character_stream{1,2,3,4}}, 7,8,9 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1 -> is_keyword{1,3}, 3 -> is_spec_symbol{1,15}, 5 -> is_identifier{1,7}, 7 -> is_num_constant{1,2,3,5}, 9 -> is_str_constant{1,7}, 11 -> is_char_constant{1,3}, 13 -> is_comment{1,3}, 15}, 3}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1 -> is_keyword{1,3}, 3 -> is_spec_symbol{1,15}, 5 -> is_identifier{1,7}, 7 -> is_num_constant{1,2,3,5}, 9 -> is_str_constant{1,7}, 11 -> is_char_constant{1,3}, 13 -> is_comment{1,3}, 15}, 3}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1 -> is_keyword{1,3}, 3 -> is_spec_symbol{1,15}, 5 -> is_identifier{1,7}, 7 -> is_num_constant{1,7}, 9 -> is_str_constant{1,7}, 11 -> is_char_constant{1,3}, 13 -> is_comment{1,3}, 15}, 3}, 10 -> get_token{1,2,3,5,6,7,8,9,10,11 -> is_spec_symbol{1,15}, 12,18 -> get_char{1}, 19,20,21,22,23 -> is_token_end{1,2,15}, 28,29,30,41}, 9 -> print_token{1,2 -> token_type{1 -> is_keyword{1,3}, 3 -> is_spec_symbol{1,15}, 5 -> is_identifier{1,7}, 7 -> is_num_constant{1,7}, 9 -> is_str_constant{1,7}, 11 -> is_char_constant{1,3}, 13 -> is_comment{1,3}, 15}, 3}, 10 -> get_token{1,2,3,4}, 10]