**Question**
Differentiate between lexemes and tokens. Give examples to support your discussion.

* Tokens- Sequence of characters that have a collective meaning.
* · Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token
* · Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.

A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Token pg. 111

A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes.

Pattern pg. 111

A pattern is a description of the form that the lexemes of a token may take. In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword. For identifiers and some other tokens, the pattern is more complex structure that is matched by many strings.

Figure 3.2: Examplesof tokens pg.112

[Token] [Informal Description] [Sample Lexemes]

if characters i, f if

else characters e, l, s, e else

comparison < or > or <= or >= or == or != <=, !=

id letter followed by letters and digits pi, score, D2

number any numeric constant 3.14159, 0, 6.02e23

literal anything but ", surrounded by "'s "core dumped"

To better understand this relation to a lexer and parser we will start with the parser and work backwards to the input.

To make it easier to design a parser, a parser does not work with the input directly but takes in a list of tokens generated by a lexer. Looking at the token column in Figure 3.2 we see tokens such as if, else, comparison, id, number and literal; these are names of tokens. Typically with a lexer/parser a token is a structure that holds not only the name of the token, but the characters/symbols that make up the token and the start and end position of the string of characters that make up the token, with the start and end position being used for error reporting, highlighting, etc.

Now the lexer takes the input of characters/symbols and using the rules of the lexer converts the input characters/symbols into tokens. Now people who work with lexer/parser have their own words for things they use often. What you think of as a sequence of characters/symbols that make up a token are what people who use lexer/parsers call lexeme. So when you see lexeme, just think of a sequence of characters/symbols representing a token. In the comparison example, the sequence of characters/symbols can be different patterns such as < or > or else or 3.14, etc.

Another way to think of the relation between the two is that a token is a programming structure used by the parser that has a property called lexeme that holds the character/symbols from the input. Now if you look at most definitions of token in code you may not see lexeme as one of the properties of the token. This is because a token will more likely hold the start and end position of the characters/symbols that represent the token and the lexeme, sequence of characters/symbols can be derived from the start and end position as needed because the input is static.