------------------ TERMINOLOGY------------------

|  |  |
| --- | --- |
| **Terms** | **Definition** |
| Domain terms |  |
| Compiler | A program that can read a program in one language i.e. the source language and translate it to an equivalent program in another language the target language |
| Interpreter | "directly executes the operations specified in the source code on inputs from the user" |
| Lexical analysis (scanning) | reads input characters and groups them into lexemes, and classifies them as a token type  scanner: handles the matching of the id's and numbers |
| Token | "an abstract symbol representing a kind of lexical unit"  (the type ie ID STRING NUMBER…) |
| Lexeme | "a sequence of characters in the source program that are identified by the lexical analyzer as an instance of a token"  (the actual char sequence that is being stored ie '42' 'Hello') |
| Pattern | "a description of the form that the lexemes of a token may take" |
| Syntax Analysis (parsing) | parsing: the act of matching a program source code to the grammar of the Program Language (TDP slide 12)  -if it doesn’t match, we will get a syntax error from the parser |
| Syntax | the structure of the symbols that make up a legal program |
| Semantics | the meaning of the program |
| Grammar | defines the valid sentences that are in the language |
| Terminal | The elementary symbols |
| Non-terminal |  |
| Production rule | the non-terminal on the left of the arrow can replace/match the phrase on the right of the arrow |
| Regular expression | ε = {ε} //empty string  (r)(s) (or possibly rs) is a RE = L(r)L(s) //cartesian product  (r)|(s) (or possibly r|s) is a RE = L(r) ∪ L(s) //union  [abc] = a|b|c  [a-z] = the alphabet a-z  \* //zero or more  + //one or more  ? //zero or one  \symbol //will override the symbol |
| Alphabet | a finite set of symbols |
| String | a finite sequence of symbols from the alphabet |
| Language | a countable set of strings over some fixed alphabet |
| Sentence | REGULAR PDF SLIDE 7 |
| Context free grammar | made up of the following components:   1. A set of terminal symbols 2. A set of non-terminal symbols 3. A set of production rules 4. A designated starting non-terminal |
| Concrete syntax tree (parse tree) |  |
| Abstract syntax tree |  |
| Ambiguous grammar | more than one derivation can produce the same sentence (top-down-parsing slide 11)  ambiguous can create a tree in multiple ways, while unambiguous cannot |
| Ect. |  |
| Structs |  |
| pointers |  |
| pipeline architecture | the standard design model of a compiler  **s**canner -> **p**arser -> **o**ptimizer -> code **g**eneration |
| automita theory |  |

------------- MATCHING PROBLEMS-------------

* Can a string/sentence be produced by a given regular expression?
* Does a parse tree match a derivation from a particular grammar?
* Is a grammar ambiguous?
* Does an automaton accept a given string?

------------- CREATION PROBLEMS--------------

* Create a regular expression or grammar that defines the language given an English description of a language
* Write a recursive descent parser for a given grammar

--------------EXERCISE PROBLEMS--------------

Chapter 1: (pg 26)(pg 37)

1.2

1. A compiler translates the source code, the interpreter interprets the meaning.

Source program -> compiler -> target program

Source program -> interpreter -> output

1. Interpreter gives better error diagnostics because it goes statement by statement. A compiler runs faster than interpreter
2. Assembly is more readble, the machine language is specific to hardware
3. C is a lower level high level language, C is a more universal which we can build on
4. Tasks an assembler needs to perform:
   1. Assembler is fed assembly code from the compiler and produces relocated machine code as output

1.3

**Chapter 2: pg 74, 91**

Exercise 2.2.1:

1. Grammar

S -> SS\*

S -> Sa\*

S -> SS+a\*

S -> Sa+a\*

S -> aa+a\*

Exercise 2.2.2:

Exercise 2.2.3:

Exercise 2.2.4:

Exercise 2.2.5:

Exercise 2.2.6:

Exercise 2.4.1:

a)

**Chapter 3:**

Exercise 3.1.1 (PG 137)

Exercise 3.3.2 (PG 148)

Exercise 3.3.3

Exercise 3.3.4

Exercise 3.3.5

Exercise 3.3.6

Exercise 3.4.1 (PG 159)

Exercise 3.4.2

Exercise 3.6.2 (PG 174)

Exercise 3.6.3

Exercise 3.6.4