1. **Define Compiler**: A compiler is a software tool that translates code written in a high-level programming language into machine code or an intermediate form that a computer's processor can execute.

Properties of a Compiler:

- **Lexical Analysis**: Tokenizes the source code.
- **Syntax Analysis**: Parses tokens to check grammar.
- **Semantic Analysis**: Ensures meaningfulness.
- **Optimization**: Enhances performance.
- **Code Generation**: Produces machine code.
- **Error Handling**: Manages errors gracefully.

3. **Define Bootstrapping**: Bootstrapping is the process of writing a simple compiler or assembler in an initial programming language, which is then used to write more complex versions of itself.

4. **Define Token and Pattern**:

- **Token**: A sequence of characters that represents a basic element in the source code, such as keywords, operators, or identifiers.
- **Pattern**: The set of rules or regular expressions that define how tokens are formed.

5. **Error Recovery Techniques**:

- **Panic Mode**: Skips tokens until a synchronizing token is found.
- **Phrase-Level Recovery**: Replaces a fragment with a legal phrase.
- **Error Productions**: Adds rules to handle errors.
- **Global Correction**: Uses algorithms to find minimal changes.

6. **State Left Recursive Grammar**: Left recursive grammar is a type of context-free grammar where a non-terminal symbol appears on the left side of its own production rule, leading to potential infinite recursion in parsers.

7. **Define Syntax Directed Definition**: Syntax directed definitions specify the semantic rules associated with a grammar, defining how values are computed based on the structure of the parse tree.

8. **Define Cross Compiler**: A cross compiler generates executable code for a platform different from the one on which the compiler is run, enabling development for diverse environments.

9. **Identify Tokens and Lexemes from Code**:

   - **Tokens**: void, main, (,), {, int, a, ,, b, ;, scanf, (, "%d%d", , &, a, ,, &, b,), ;, printf, (, "%d", ,, a, +, b,), ;

   - **Lexemes**: Each actual string in the source code corresponding to the tokens, such as `void`, `main`, `(`, `)`, `{`, `int`, `a`, `b`, `;`, `scanf`, etc.

10. **List out Types of Parsers**:

   - **Top-Down Parsers**: Predictive, Recursive Descent.

   - **Bottom-Up Parsers**: LR, SLR, LALR, Canonical LR.

11. **Define Ambiguous Grammar with Example**: Ambiguous grammar generates more than one parse tree for some sentences. Example: The grammar `E -> E + E | E * E | id` is ambiguous as the expression `id + id * id` can have multiple parse trees.

12. **List out Applications of SDT**:

   - **Translation**: Source-to-source transformation.

   - **Type Checking**: Ensuring type correctness.

   - **Code Optimization**: Enhancing code performance.

   - **Intermediate Code Generation**: Producing intermediate representations.

13. **List out Data Structures Used in Compiler**:

   - **Abstract Syntax Tree (AST)**: Represents program structure.

   - **Symbol Table**: Stores variable/function information.

   - **Parse Tree**: Represents syntactic structure.

   - **Intermediate Code**: Platform-independent code form.

   - **Control Flow Graph (CFG)**: Represents program flow.