Question and Answers Document

1. State error recovery techniques in top down parsing.

Top-down parsing is a parser that starts at the top of the parse tree and works its way down, trying to match the input string to the grammar. If it encounters an error, it can try to recover by backtracking and trying a different path through the grammar.

There are a few different error recovery techniques that can be used in top-down parsing. One technique is to simply ignore the error and continue parsing. This can be effective if the error is not critical to the meaning of the input string.

Another technique is to try to find a way to repair the error. This can be done by inserting or deleting characters from the input string, or by changing the order of the characters.

Finally, the parser can also try to generate a parse tree that is consistent with the error. This can be done by using a technique called "error-tolerant parsing."

2. What is Boot ‘Strapping?

Bootstrapping is a technique used to initialize a compiler or interpreter. The compiler or interpreter is first written in a language that is easy to implement, such as assembly language. This language is then used to write a compiler or interpreter for a more powerful language, such as C or Java. The compiler or interpreter for the more powerful language is then used to compile or interpret the original compiler or interpreter. This process is repeated until the compiler or interpreter is written in the most powerful language available.

3. Difference between Top Down parsing and Bottom up parsing.

Top-down parsing is a parser that starts at the top of the parse tree and works its way down, trying to match the input string to the grammar. Bottom-up parsing is a parser that starts at the bottom of the parse tree and works its way up, trying to match the input string to the grammar.

Top-down parsing is typically faster than bottom-up parsing, but it is more difficult to implement. Bottom-up parsing is typically easier to implement than top-down parsing, but it is slower.

4. What are ambiguous grammars? Give an example.

An ambiguous grammar is a grammar that can have more than one parse tree for a given

input string. For example, the following grammar is ambiguous:

```
S -> NP VP
NP -> Det N
VP -> V NP | V
Det -> "the" | "a"
N -> "man" | "woman"
V -> "saw" | "ate"
```

The input string "the man saw the woman" can be parsed in two different ways:

* S -> NP VP
* NP -> Det N
* N -> "man"
* VP -> V NP
* V -> "saw"
* NP -> Det N
* N -> "woman"

* S -> NP VP
* NP -> Det N
* N -> "man"
* VP -> V
* V -> "saw"
* NP -> Det N
* N -> "woman"

5. What is Yacc? Explain the Syntax.

Yacc is a parser generator that can be used to generate parsers for context-free grammars. Yacc is a recursive descent parser generator, which means that it generates a parser that parses the input string by recursively calling itself.

The syntax of Yacc grammars is as follows:

```
%token <token-name>
%left <production-name>
%right <production-name>
%nonassoc <production-name>
```

<start-symbol> : <production>
<production> : <non-terminal> | <non-terminal> <production>
<non-terminal> : <terminal> | <non-terminal> <terminal>
<terminal> : <character-literal> | <string-literal> | <identifier>
```

The `%token` directive defines a token. The `%left`, `%right`, and `%nonassoc` directives define the associativity of a production. The `<start-symbol>` directive defines the start symbol of the grammar. The `<production>` directive defines a production. The `<non-terminal>` directive defines a non-terminal. The `<terminal>` directive defines a terminal.

6. Define shift-reduce conflict and Reduce-reduce conflict.

A shift-reduce conflict is a conflict that occurs when the parser can either shift the next input symbol onto the stack or reduce the top two symbols on the stack. A reduce-reduce conflict is a conflict that occurs when the parser can reduce the top two symbols on the stack in two different ways.
```