Ans1) (a)

Parameter	**Context Sensitive Grammar**	**Context Free Grammar**
Language	Context Sensitive Language	Context Free Language
Automata	Linear Bounded Automata	Push Down Automata
Also called	Type 1 Grammar	Type 2 Grammar
Power	More Powerful	Less Powerful

(b)

Key	**Top Down Parsing**	**Bottom Up Parsing**
Strategy	Top-down approach starts evaluating the parse tree from the top and move downwards for parsing other nodes.	Bottom-up approach starts evaluating the parse tree from the lowest level of the tree and move upwards for parsing the node.
Attempt	Top-down parsing attempts to find the left most derivation for a given string.	Bottom-up parsing attempts to reduce the input string to first symbol of the grammar.
Derivation Type	Top-down parsing uses leftmost derivation.	Bottom-up parsing uses the rightmost derivation.
Objective	Top-down parsing searches for a production rule to be used to construct a string.	Bottom-up parsing searches for a production rule to be used to reduce a string to get a starting symbol of grammar.

(c)

Feature	Pass1	Pass2
Purpose	Scans the input code and creates a symbol table	Analyzes the input code and generates the machine code
What it does	Reads the source code line by line and creates a symbol table for each variable and function.	Reads the symbol table and generates the machine code for each statement in the source code.
When it happens	First pass	Second pass
Complexity	Less complex	More complex
Efficiency	More efficient	Less efficient
Applications	Used for compilers that target simple languages	Used for compilers that target complex languages

(d)

Feature	Inherited attributes	Synthesized attributes
Determined by	Attributes of the parent node	Attributes of the child nodes
Example	Type of an expression	Value of an expression
When it is evaluated	During the parse tree traversal	During the code generation
Used for	Semantic analysis	Code generation

Ans4) (a)

Lexical Analyzer Source Code :

```
%{
	/* Definition section */
	#include "y.tab.h"
%}
/* Rule Section */
%%
	[aA] {return A;}
	[bB] {return B;}
	\n {return NL;}
	. {return yytext[0];}
%%
int yywrap()
{
	return 1;
}
```

Parser Source Code :

```
%{
	/* Definition section */
	#include<stdio.h>
	#include<stdlib.h>
%}
%token A B NL
/* Rule Section */
%%
stmt: S NL {
printf("valid string\n");
exit(0);
}
;
S: A S B |
;
```

```
%%
int yyerror(char *msg)
{
        printf("invalid string\n");
        exit(0);
}
//driver code
main()
{
        printf("enter the string\n");
        yyparse();
}
```

(b)

1. Token Generator (Lexer):

The token generator, also known as a lexer or scanner, is responsible for scanning the input source code and breaking it into tokens. It performs lexical analysis by identifying and classifying the individual tokens based on predefined rules or patterns.

2. Token Recognizer:

The token recognizer, also known as a parser or syntax analyzer, receives the tokens generated by the lexer and checks whether they conform to the grammar rules of the programming language. It constructs a syntax tree or performs further analysis based on the sequence and structure of the tokens.

Example:

Consider a simple arithmetic expression: `2 * (3 + 4)`. The scanner would tokenize this expression into the following sequence of tokens:

TOKEN_INTEGER_LITERAL(2)

TOKEN_MULTIPLICATION_OPERATOR(*)

TOKEN_OPEN_PARENTHESIS("(")

TOKEN_INTEGER_LITERAL(3)

TOKEN_ADDITION_OPERATOR(+)

TOKEN_INTEGER_LITERAL(4)

TOKEN_CLOSE_PARENTHESIS(")")

Token Recognizer would use the tokens provided by the scanner to build a parse tree

```
    *
   /   \
 2      +
      /   \
    3      4
```

Ans6) (a)

A **syntax-directed definition** (SDD) is a method for specifying the semantics of a programming language. It is a generalization of context-free grammars that allows for the definition of attributes associated with each production rule. These attributes can be used to represent the meaning of a program construct.

SDDs are used in compilers to implement semantic analysis. Semantic analysis is the process of determining the meaning of a program construct. It is used to check for errors, generate code, and perform other tasks.

SDDs are written in a special language called **attribute grammars**. Attribute grammars are a formal notation for specifying the semantics of a programming language. They are used to define the attributes associated with each production rule and the rules for computing the values of these attributes.

There are two types of SDDs:

- **S-attributed SDDs** are SDDs in which the attributes are only associated with the non-terminal symbols in the grammar. The attributes of the terminal symbols are not defined.
- **L-attributed SDDs** are SDDs in which the attributes are associated with both the non-terminal and terminal symbols in the grammar.

Ans7)

- **Target machine:** The code generator must be able to generate code for the target machine. This includes considering the instruction set architecture, the memory model, and the calling convention.
- **Optimization:** The code generator can be used to optimize the generated code. This can be done by performing instruction selection, register allocation, and loop unrolling.
- **Code generation strategy:** There are different strategies for generating code. The most common strategy is to generate code one statement at a time. However, other strategies, such as generating code for entire functions or blocks of code, can also be used.
- **Code quality:** The code generator must generate high-quality code. This means that the code must be correct, efficient, and portable.

Ans8)

(a)

- **Ambiguity:** The same sequence of characters can be interpreted in multiple ways. For example, "while" can be a keyword or a two-word identifier.
- **Whitespace:** Whitespace characters can be ignored, which can make it difficult to identify the boundaries of tokens.
- **Comments:** Comments are typically ignored by the lexical analyzer, but it can be difficult to distinguish them from tokens.
- **Keywords:** Reserved words that have a special meaning in the programming language.
- **Identifiers:** Names that are used to refer to variables, functions, and other entities in the program.
- **Literals:** Constants that have a fixed value.

(b)

Feature	**LA attributes**	**S attributes**
Determined by	Attributes of the left-hand side of a production	Attributes of the right-hand side of a production
Example	Type of an expression	Value of an identifier
When it is evaluated	During the parse tree traversal	During the code generation
Used for	Semantic analysis	Code generation

(c)

Peephole optimization: It is performed on a very small set of instructions in a segment of code. A part of code is replaced by shorter and faster code without a change in output. The peephole is machine-dependent optimization.

The objectives of peephole optimization are:

1. To improve performance
2. To reduce memory footprint
3. To reduce code size

A. Redundant load and store elimination: In this technique, redundancy is eliminated.

```
Initial code:
y = x + 5;
i = y;
z = i;
w = z * 3;

Optimized code:
y = x + 5;
w = y * 3; //* there is no i now

//* We've removed two redundant variables i & z whose value were just being
copied from one another.
```

B. Constant folding: The code that can be simplified by the user itself, is simplified. Here simplification to be done at runtime are replaced with simplified code to avoid additional computation.

```
Initial code:
x = 2 * 3;

Optimized code:
x = 6;
```

C. Strength Reduction: The operators that consume higher execution time are replaced by the operators consuming less execution time.

```
Initial code:
y = x * 2;

Optimized code:
y = x + x;    or     y = x << 1;

Initial code:
y = x / 2;

Optimized code:
y = x >> 1;
```

D. Null sequences/ Simplify Algebraic Expressions : Useless operations are deleted.

```
a := a + 0;

a := a * 1;

a := a/1;

a := a - 0;
```

E. Combine operations: Several operations are replaced by a single equivalent operation.

F. Dead code Elimination: A part of the code which can never be executed, eliminating it will improve processing time and reduces set of instruction.