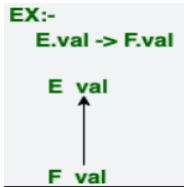
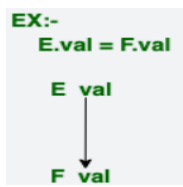


## Assignment 2

### Q1. Differentiate between synthesized attributes and inherited attributes

S.NO	Synthesized Attributes	Inherited Attributes
1.	An attribute is said to be Synthesized attribute if its parse tree node value is determined by the attribute value at child nodes.	An attribute is said to be Inherited attribute if its parse tree node value is determined by the attribute value at parent and/or siblings node.
2.	The production must have non-terminal as its head.	The production must have non-terminal as a symbol in its body.
3.	A synthesized attribute at node n is defined only in terms of attribute values at the children of n itself.	An Inherited attribute at node n is defined only in terms of attribute values of n's parent, n itself, and n's siblings.
4.	It can be evaluated during a single bottom-up traversal of parse tree.	It can be evaluated during a single top-down and sideways traversal of parse tree.
5.	Synthesized attributes can be contained by both the terminals or non-terminals.	Inherited attributes can't be contained by both, It is only contained by non-terminals.
6.	Synthesized attribute is used by both S-attributed SDT and L-attributed SDT.	Inherited attribute is used by only L-attributed SDT.
7.	 <p>EX:- E.val → F.val</p>	 <p>EX:- E.val = F.val</p>

### Q2. How Lex (flex) tool can be used to create Lexical analyzer? Explain with help of example

#### • Lex:

- reads in a collection of regular expressions, and uses it to write a C or C++ program that will perform lexical analysis. This program is almost always faster than one you can write by hand.

#### Contents of a **lex** program:

##### Declarations

%%

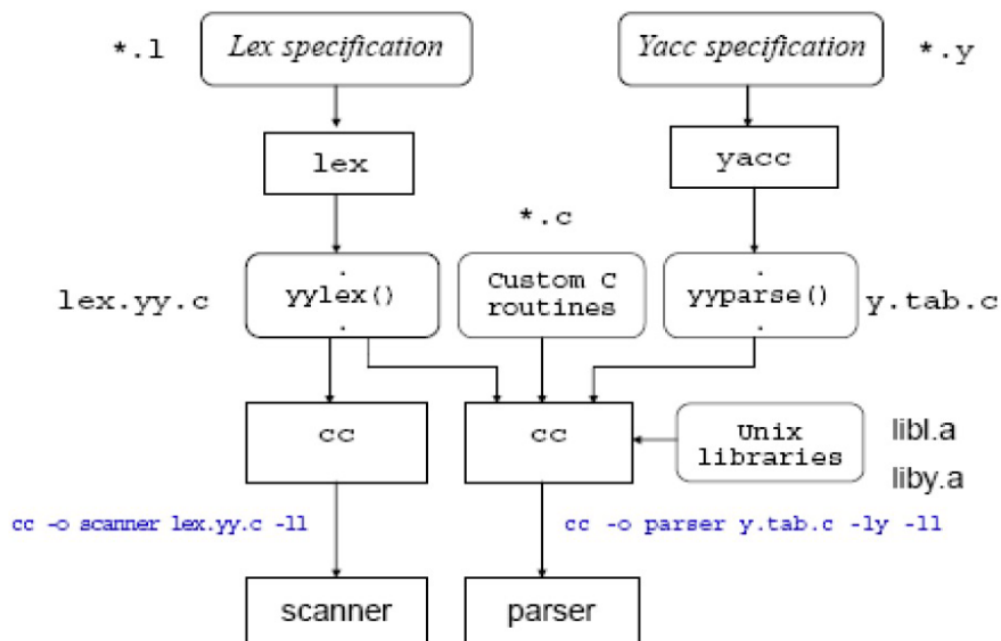
##### Translation rules

%%

##### Auxiliary functions

- The declarations section can contain declarations of variables, manifest constants, and regular definitions. The declarations section can be empty.
- The translation rules are each of the form  
pattern {action}

# Using lex and yacc tools



## Example lex program

```
%%
zip printf("ZIP");
zippy printf("ZIPPY");
```

Output:

```
ZIPPY
ali ZIP
veli and ZIPPY here
ZIPZIPPY
```

Q3. Define the following a) Topological Sort b) Dependency Graph c) Attribute d) Circular SDT

### a) Topological sorting

for **Directed Acyclic Graph (DAG)** is a linear ordering of vertices such that for every directed edge  $u \rightarrow v$ , vertex  $u$  comes before  $v$  in the ordering.

**Note:** Topological Sorting for a graph is not possible if the graph is not a **DAG**.

b)

A **dependency graph** depicts the flow of information among the attribute instances in a particular parse tree; an edge from one attribute instance to another means that the value of the first is needed to compute the second. Edges express constraints implied by the semantic rules.

### c) Attribute Grammar

Attribute grammar is a special form of context-free grammar where some additional information (attributes) are appended to one or more of its non-terminals in order to provide context-sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions.

Attribute grammar is a medium to provide semantics to the context-free grammar and it can help specify the syntax and semantics of a programming language. Attribute grammar (when viewed as a parse-tree) can pass values or information among the nodes of a tree

d) In a syntax-directed translation (SDT) scheme, circular SDT occurs when there's a cycle in the dependency graph of attributes in a syntax tree, creating a circular dependency. This is problematic because it makes the attribute evaluation non-deterministic or impossible since each attribute relies on another in a cycle, preventing a clear order of evaluation.

