- What is lexical analyzer?

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of Tokens. Lexical Analysis can be implemented with the Deterministic finite Automata.

- What is the output of Lexical analyzer?

The output is a sequence of tokens that is sent to the parser for syntax analysis

- What is LEX source Program?

It is a language used for specifying or representing Lexical Analyzer. It denotes the regular expression of the form. Regular Expression (Ri)→ Notation to represent a collection of input symbols

- What is token?

Token is basically a sequence of characters that are treated as a unit as it cannot be further broken down

- What is lexeme?

A lexeme is an actual character sequence forming a specific instance of a token, such as num

- List the different sections available in LEX compiler?

lex program consists of three parts: the definition section, the rules section, and the user subroutines.

```
        ...definition section ...
%%
... rules section ...
%%
... user subroutines ...
```

- How can we define the translation rules?

The translation rule in consideration has val as an attribute for both the non-terminals – E & T. Right-hand side of the translation rule corresponds to attribute values of the right-side nodes of the production rule and vice-versa

- What is the input for LEX Compiler?

Lex reads an input stream specifying the lexical analyzer

- What is the output of LEX compiler?

C program

- What is top-down parsing?

In top-down parsing,the parse tree is generated from top to bottom, i.e., from root to leaves & expand till all leaves are generated

- What is parse tree?

Parse tree is the hierarchical representation of terminals or non-terminals. These symbols (terminals or non-terminals) represent the derivation of the grammar to yield input strings

- What is ambiguous grammar?

A CFG is said to be ambiguous if there exists more than one derivation tree for the given input string

- What is the output of parser?

A parser takes input in the form of sequence of tokens and produces output in the form of parse tree. Parsing is of two types: top down parsing and bottom up parsing

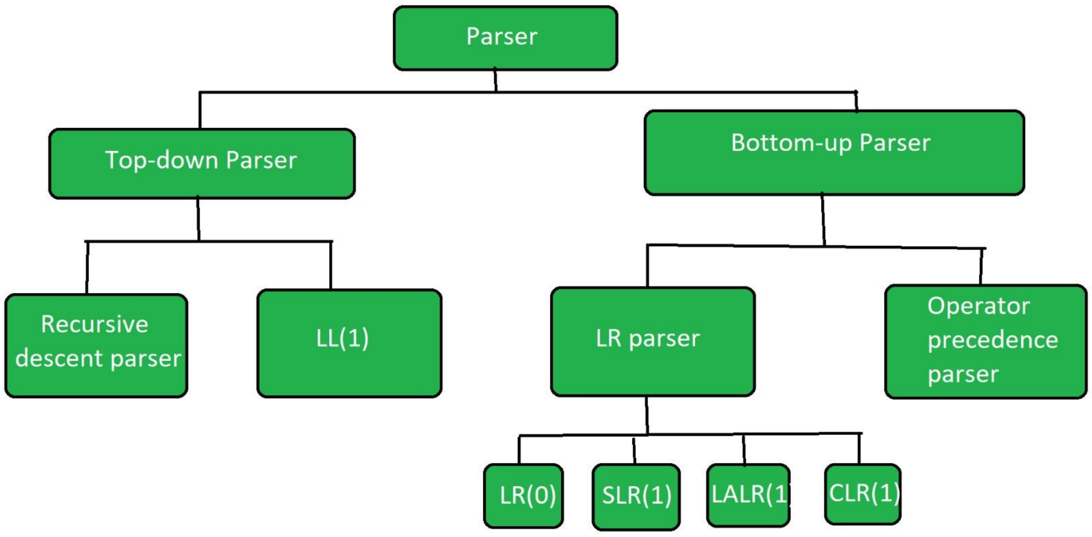
- What is Predictive parser?

A predictive parser is a recursive descent parser with no backtracking or backup

- What is Recursive Decent Parser?

A recursive descent parser is a type of parsing tool that works on a recursive basis, in other words, on the basis of using one instance of a command or event to generate another. Recursive descent parsers can be used to parse different types of code, such as XML, or other inputs.

- How many types of Parsers are there?



- What is LR Parser?

LR Parser is a class of Bottom-Up Parser that is used to parse Context-Free Grammars. LR Parsing is known as LR (K) parsing where. L represents Left to Right Scanning of Input. R represents Rightmost Derivation. K is the number of input symbols of Look ahead that are used in developing parsing decisions

- Why bottom-up parsing is also called as shift reduce parsing?

Bottom Up Parsers / Shift Reduce Parsers Build the parse tree from leaves to root

- What are the different types of bottom up parsers?

(Ans in the above figure)

- What is YACC?

YACC is known as Yet Another Compiler Compiler**. It is used to produce the source code of the syntactic analyzer of the language produced by LALR(1) grammar. The input of YACC is the rule or grammar, and the output is a C program**

- What is LALR parsing?

LALR Parser is lookahead LR parser. It is the most powerful parser which can handle large classes of grammar. The size of CLR parsing table is quite large as compared to other parsing table. LALR reduces the size of this table.LALR works similar to CLR. The only difference is , it combines the similar states of CLR parsing table into one single state.

- What are the operations of Parser?

shift, reduce, accept, and error

- What is the use of parsing table?

parsing uses a stack and parsing table to parse the input and generate a parse tree

- What is three address code?

Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code.It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.

```
a = b op c
```

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

- What are the record structures we use to represent three address code

It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression

- What is Abstract Syntax tree?

An abstract syntax tree (AST) is a way of representing the syntax of a programming language as a hierarchical tree-like structure. This structure is used for generating symbol tables for compilers and later code generation. The tree represents all of the constructs in the language and their subsequent rules.

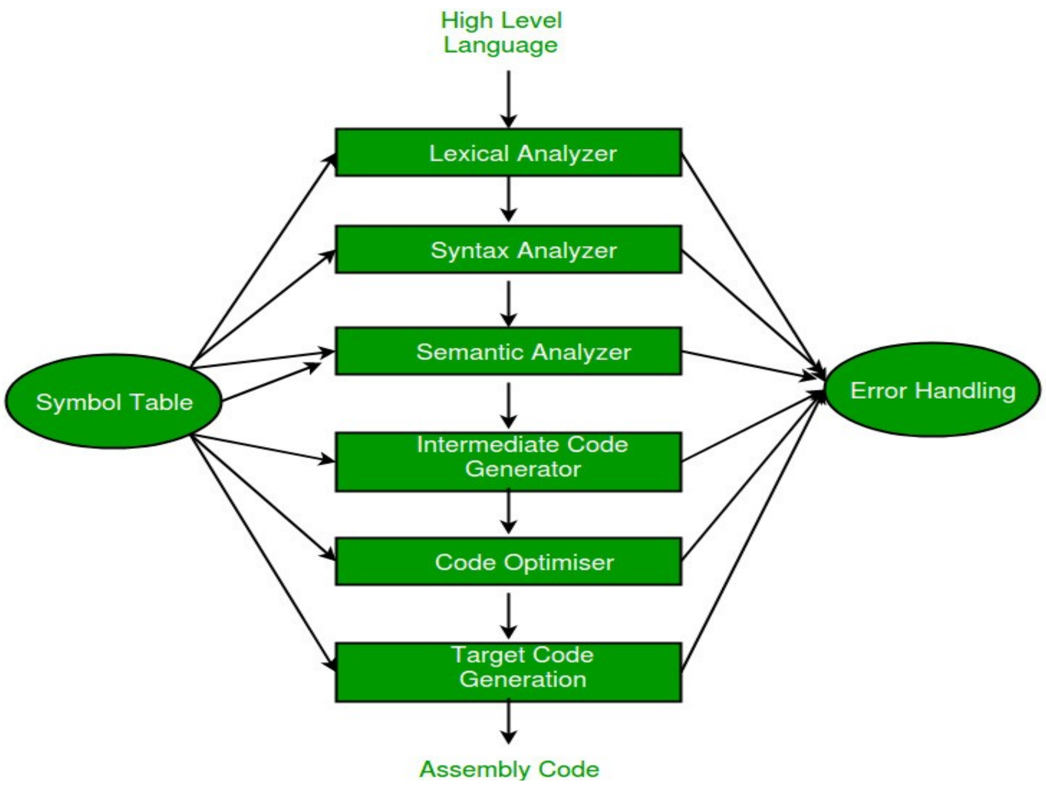
- What is DAG representation

The Directed Acyclic Graph (DAG) is used to represent the structure of basic blocks, to visualize the flow of values between basic blocks, and to provide optimization techniques in the basic block

- What are the two parts of a compilation? Explain briefly.

Analysis and Synthesis are the two parts of compilation. The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program. The synthesis part constructs the desired target program from the intermediate representation.

- List the various phases of a compiler



- What is a symbol table?

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variables

- What does a semantic analysis do?

Semantic analysis is the task of ensuring that the declarations and statements of a program are semantically correct, i.e, that their meaning is clear and consistent with the way in which control structures and data types are supposed to be used.

- Why lexical and syntax analyzers are separated out?

Separation allows the simplification of one or the other. 2) Compiler efficiency is improved. Optimization of lexical analysis because a large amount of time is spent reading the source program and partitioning it into tokens. 3) Compiler portability is enhanced.

- What is a operator precedence parser?

An operator-precedence parser is a simple shift-reduce parser that is capable of parsing a subset of LR(1) grammars. More precisely, the operator-precedence parser can parse all LR(1) grammars where two consecutive nonterminals and epsilon never appear in the right-hand side of any rule

- What are the various types of intermediate code representation?

The intermediate code can be represented in the form of postfix notation, syntax tree, directed acyclic graph (DAG), three-address code, quadruples, and triples.

- What are the contents of activation record?

An activation record contains all the necessary information required to call a procedure. An activation record may contain the following units (depending upon the source language used). Stores temporary and intermediate values of an expression. Stores local data of the called procedure

Example of top down parser?

Example of bottom up parser?

- What is shift and reduce operation, how its related to stack ?

Shift: This involves moving symbols from the input buffer onto the stack. Reduce: If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of a production rule is popped out of a stack and LHS of a production rule is pushed onto the stack.

- Peephole technique

Peephole optimization is a type of **code Optimization** performed on a small part of the code. It is performed on a very small set of instructions in a segment of code.

The small set of instructions or small part of code on which peephole optimization is performed is known as peephole or window.

It basically works on the theory of replacement in which a part of code is replaced by shorter and faster code without a change in output. The peephole is machine-dependent optimization.

- Loop hole optimization

Loop Optimization is the process of increasing execution speed and reducing the overheads associated with loops. It plays an important role in improving cache performance and making effective use of parallel processing capabilities

- Left recursion

- Left factoring

- Which parser cannot run left recursive grammar

A top-down parser cannot handle left recursive productions

- Activation record

An activation record is a contiguous block of storage that manages information required by a single execution of a procedure.

- LALR parser

LALR Parser is lookahead **LR parser**. It is the most powerful parser which can handle large classes of grammar.

- LL parser

an LL parser (Left-to-right, leftmost derivation) is a top-down parser for a restricted context-free language. It parses the input from Left to right, performing Leftmost derivation of the sentence

- SLR parser

SLR (1) refers to simple LR Parsing. It is same as LR(0) parsing. The only difference is in the parsing table.To construct SLR (1) parsing table, we use canonical collection of LR (0) item. In the SLR (1) parsing, we place the reduce move only in the follow of left hand side

- Loop jamming

Loop jamming is the combining the two or more loops in a single loop. It reduces the time taken to compile the many number of loops

- Loop unrolling

Loop unrolling is a technique used to increase the number of instructions executed between executions of the loop branch logic. This reduces the number of times the loop branch logic is executed.

- Deadcode elmination

Dead Code Elimination is an optimization that removes code which does not affect the program results. You might wonder why someone would write this type of source code, but it can easily creep into large, long-lived programs even at the source code level.

- Sub expression elimination

compiler optimization that searches for instances of identical expressions (i.e., they all evaluate to the same value), and analyzes whether it is worthwhile replacing them with a single variable holding the computed value.

- Constant propogation

Constant Propagation is one of the local code optimization technique in Compiler Design. It can be defined as the process of replacing the constant value of variables in the expression. In simpler words, we can say that if some value is assigned a known constant, than we can simply replace the that value by constant.

- Eg of intermediate code generation

- Difference between lr 0 (multiple entries)and lr 1(unique entries)

The only difference between LR(0) and SLR(1) is this extra ability to help decide what action to take when there are conflicts. Because of this, any grammar that can be parsed by an LR(0) parser can be parsed by an SLR(1) parser

- Annotated parse tree

AN ANNOTATED PARSE TREE. is a parse tree showing the values of the attributes at each node. The process of computing the attribute values at the nodes is called annotating or decorating the parse tree. stand respectively for an integer number and the newline character.