MID:

Q) What is a Language Processor?

→ It's a software that converts given language to target language.

Q) Write the Major types of Languge Processors.

→ Assemblers, Compilers, Interpreters

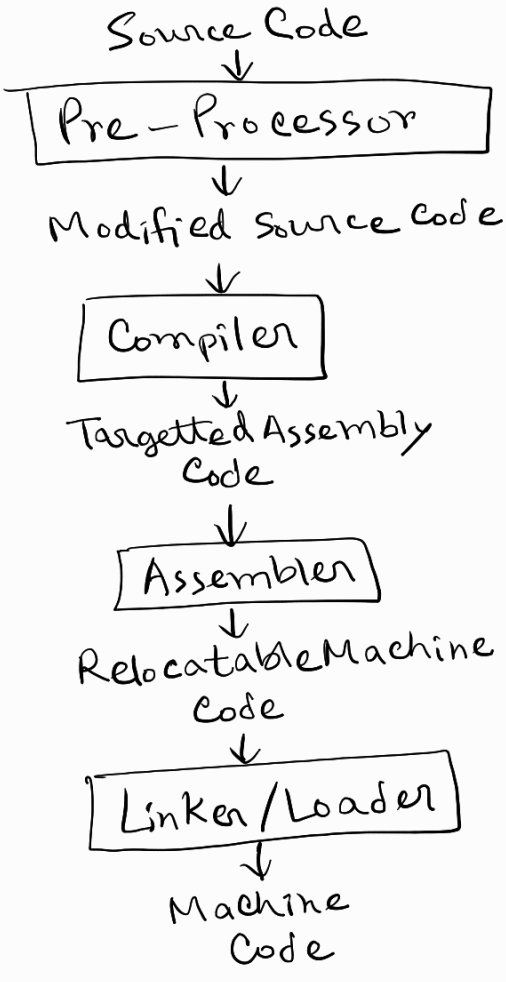
Q) What are the Differences between Compilers & Interpreters?

→

Compilers	Interpreters
• Compilers take a lot of time analyzing the entire source code before producing any output. It processes the entire source code at once and produces final machine code.	• Interpreters execute lines of code at runtime, line by line. So it takes less time to see code execution.
• Compiled code (the end-result machine code) runs faster. (Once compiled, no need to recompile source code to rerun program).	• Interpreted code runs slower, since trying to run the program again means re-interpreting every time.
• Shows all errors at once.	• Shows errors line by line

Q) Draw the Compilation Process Diagram

→



Q) What does Linker do?

→ It links other necessary files and combines them all into a single executable.

Q) What does Loader do?

→ Loads the executable to memory (RAM) for execution.

Q) Write the Preprocessing steps of Compilation Process:

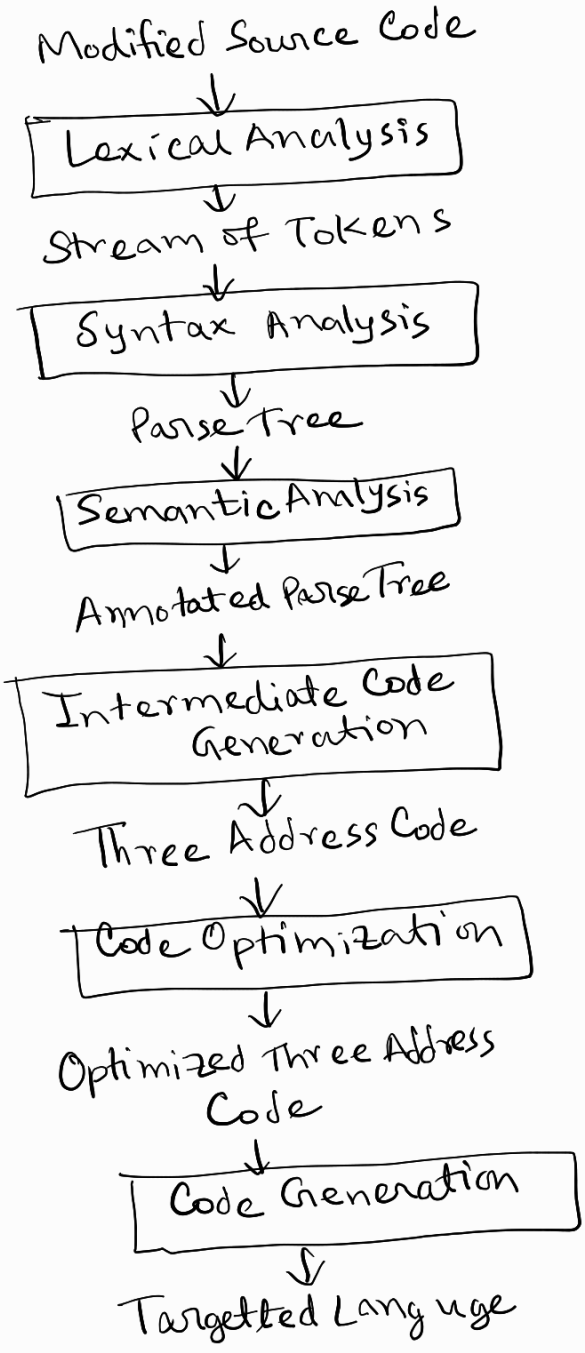
→
1. Removing Hashtags
2. Adding necessary files → Getting scope table
3. Macro Expansion → Replacing constants with actual values
4. Operator Conversion → " operator short forms like i++ to full forms like i=i+1.

Q) Write the Compiler steps:

1. Lexical Analysis
2. Syntax "
3. Semantic "
4. Intermediate Code Generation
5. Code Optimization (optional)
6. Code Generation

Q) Draw Compiler Steps diagram:

→



Every phase has an error handler connected to it

Q) Write the major types of Syntax Errors:

→
- missing or misplaced punctuations
- incorrect use of keywords
- mismatched brackets
- incorrectly structured statements

Q) Write the major types of Semantic Errors:

→
- Type mismatch
- Using undeclared variables
- Misuse of operators
- Logical Errors

Q) How are tokens generated?

→ Lexer reads lexemes → Lexemes matched against Patterns → Patterns that match produce tokens.

Q) What is difference between lexeme, pattern, grammar.

→

Lexeme	Pattern	Grammar
• Sequences of characters in source code that gets matched with Patterns to produce Tokens	• Regex rules lexemes are matched against	• Set of production conditions which get
• Found in Source Code	• Found in lexer	• Found in Parser

Q) Write down the Notational Conventions for grammars.

→

Terminals:
- should be lowercase
- can be bold

Example:
A → i

Non Terminal
- All uppercase letters

Example:
A → T + T

Q) What are the types of Parsers?

→
1. Universal Parsers (not used anymore)
2. Top-down Parsers (like LL parser) → Performs Leftmost Derivation
3. Bottom-up Parsers (like LR Parser) → Performs Rightmost Derivation
   - LR(0) Parser ← Look ahead symbol
   - SLR1 Parser

Q) What is Ambiguous grammar?

→ If the same grammar can be used to construct two different parse trees or more to derive the same input string the grammar is an ambiguous grammar.

Q) What is Bottom-Up Parsing?

→ Creating the parse tree for given input string, starting from the leaves to the root.

Also called Shift-Reduce Parsing

Top-Down এর বিপরীত টেকনিক

Q) What is Reduction?

→ A Reduction is a step, which replaces a substring matching the production body, with the head of production.

Q) What is a Handle?

→ A substring that matches the body of production, whose reduction means one step reverse of the rightmost derivation.

Q) What data structures are used by Shift-Reduce parsers?

→ Input Buffer (for input string) and Stack

Q) What are the Operations in Shift Reduce/Bottom-Up parsing?

→ 1. Shift → Current symbol in Input Buffer is pushed to stack

2. Reduce → Symbols at the top of the Stack gets replaced with their production heads

3. Accept → Announce successful completion

4. Error → Discover syntax error and calls error recovery functions.

Q) What conflicts may arise when parsing?

→ 1. Shift Reduce
2. Reduce-Reduce

Q) Write Error Recovery methods in Syntax Analyzer.

→
- Panic Mode Recovery — Skips parts of code
- Phase Level Recovery — Tries replacing parts of code
- Error Production Rule — Rules that handle common mistakes
- Global Correction — Replacing source code with Global code to make it run, keeping source code as intact as possible

Q) What is a Symbol Table?

→ A data structure used by compilers to store information about variables, functions, objects and other symbols in a program.

Q) Why are Hash Functions used to store/search items in Symbol Table?

→ Hash Functions make searching/storing items very fast.

Q) What is a Collision?

→ When Hash function gives same index for two different items/symbols.

Q) Write down ways to solve collisions.

→
1. Linear Probing → Find next empty index
2. Double Hashing → Use two hash functions together
3. Forward Chaining → Create linked lists.

Q) What are the types of Symbol tables?

→
1. Hashtables of list
2. List of Hashtables (more widely used, faster & efficient)

Q) What are the Actions in List of Hashtables?

→
1. Scope Entry:
   - createTable
   - level++
2. Process a Declaration:
   - check if symbols exist in current scope.
     - If yes → MultipleDeclaredVariable
     - " no → Store in table

3. Process of a Use:

- Search current scope
  - If found: Proceed
  - If not found:
    - Check parent scope
      - If found: Proceed
      - If not found:
        - ⋮
        - If reached Global Scope and not found: Undeclared variable

4. Scope exit:

- Delete current table
- level--

Q) What are the Actions in List of Hashtables ?

→ 1. Scope Entry:

- level++

2. Process a declaration:

- check if symbol exists already.
  - if yes → Compare level values.
    - if same → Multiple Declared Variable
    - else: Store in table
  - else: Store in table

3. Process a Use:

Search in current and Parent Scope:

⋮

→ if not found: Undeclared

4. Scope exit:

- Delete all symbols of that level in the table
- level--

FINAL: