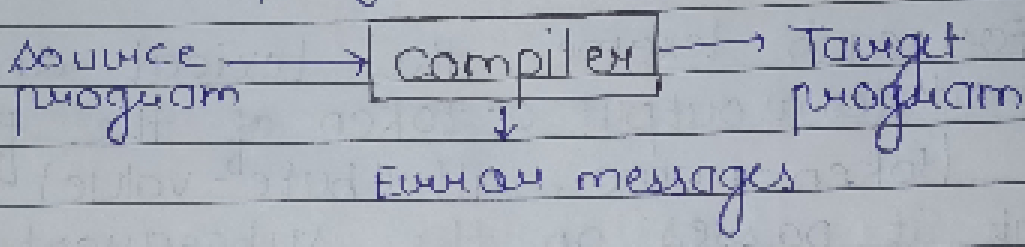
Date 29/02/24

UNIT I :

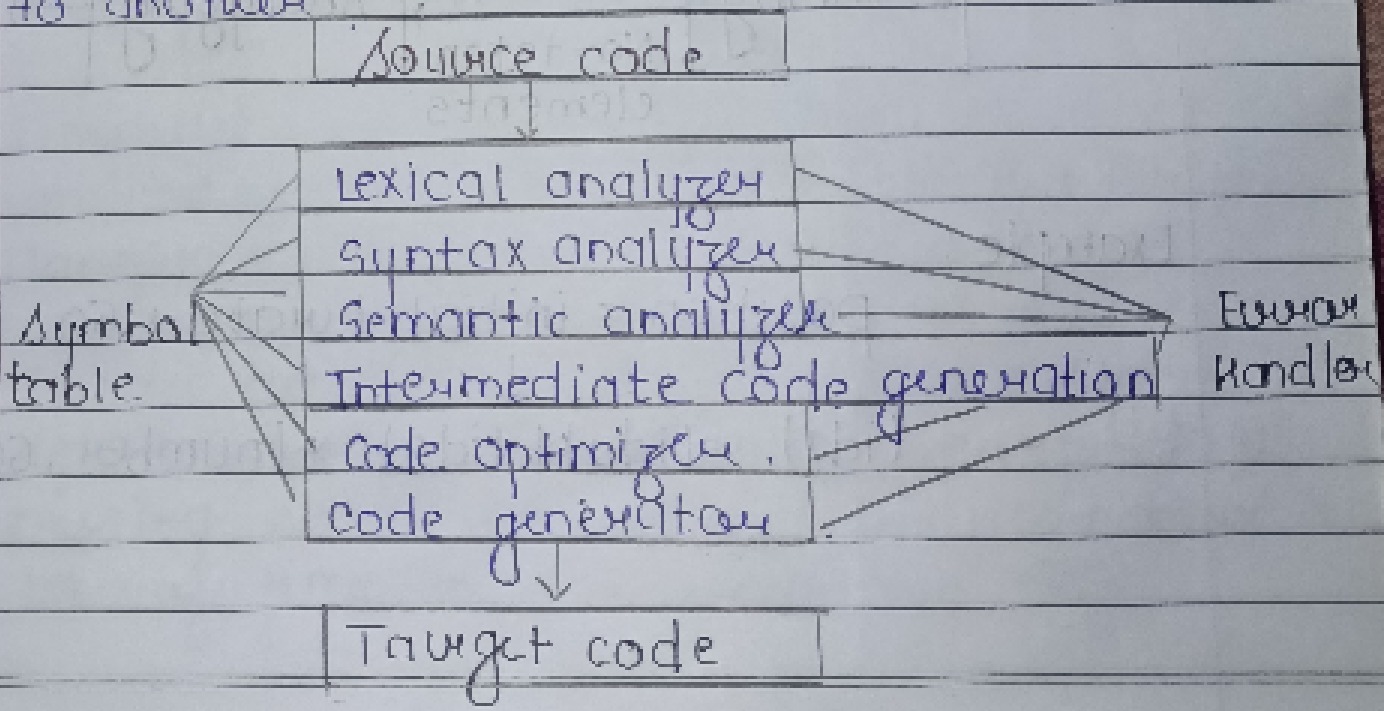
Introduction to Compiler

- Compiler is a program that converts high level language into its equivalent m/c language.
- An important part of this translation process is the compiler reports to its user the presence of errors in the source program



* Phases of Compiler

Conceptually, a compiler operates in phases, each of which transforms the source program from one representation to another.



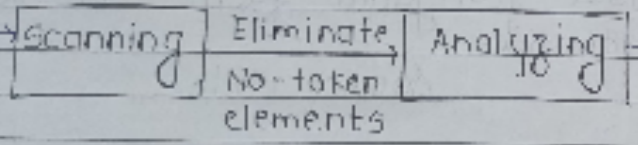
Q.5 (i) lexical analysis phase

Answer 1) The first phase of a compiler is called lexical analysis or scanning. It is also called as linear analysis.

2) It reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes.

3) For each lexeme, the lexical analyzer produces as output a token of the form (token-name, attribute value) that it passes on the subsequent phase, syntax analysis.

Lexical analysis



Example:

position = initial + rate * 60

↓

(id1) = (id2) + (id3) * (number, 60)

Example

```
int main()
{
  // 2 variables
  int a, b;
  a = 10;
  return 0;
}
```

All valid tokens are:

'int' 'main' '(' ')' '{' 'int' 'a' ',' 'b' ';' 'a' '=' '10' ';' 'return' '0' ';' '}'

Advantages:

1) Efficiency: It improves the efficiency of the parsing process because it breaks down the input into smaller, more manageable chunks.

2) Flexibility: It allows for the use of keywords and reserved words in programming language.

3) Error detection: The lexical analyzer can detect errors such as misspelled words, missing semicolons, etc. This can save lots of time in the debugging process.

* Disadvantages

1) Complexity : It is complex and requires a lot of computational power.

2) Increased code size : The addition of keyword and reserved words can increase the size of the code, making it more difficult to read and understand.

Q.5 Write difference bet^n phases & passes

Parameter	Phases	Passes
1) Definition	Functional stages of the compilation process.	Number of times the compiler traverses the entire source code
2) Focus.	Specific task performed.	Overall iteration through the code
3) Granularity	Broader, encompassing multiple steps.	Finer, individual traversal.
4) Independence.	They are independent.	They are dependent i.e sequential order.
5) Examples	Syntax analysis phase.	Single-pass compiler & multi-pass compiler.

Q.4 Explain the advantages of Analysis Synthesis model of compilation

Answer : Modular structure

Facilitates optimization

Advantages of analysis synthesis model of compilation

Flexibility

Portability

Emphasized error reporting

1) Modular structure

• Clear separation : It breaks down the complex process into well-defined phases, making the structure easier to understand.

• Maintainance : Each phase represents a distinct module making easier maintenance and updates.

• Reusability : The modularity allows for components reusability.

2) Portability:

- Intermediate representation: The use of an intermediate representation betn analysis and synthesis makes the compiler more adaptable to different target machine architecture.

3) Facilitates optimization

- Optimization scope: The intermediate representation provides a level of abstraction that allows the compiler to see the code's overall structure.

- Optimization passes: within the synthesis phase you can implement multiple phases.

4) Improved error reporting

- Specific errors: Each phase focuses on a specific aspect of compilation. It isolates errors related to syntax, semantics or code generation to the responsible phases.

5) Flexibility:

- Pass organization: While there are common flow of phases, the order and the number of passes within each phase can be adjusted for specific compiler needs.

Q.

Criteria	Token	Lexeme	Pattern
Definition	It is basically a sequence of char that are treated as a unit.	It is a sequence of char. in the source code that are matched by given predefined language rules.	It specifies a set of rules that a scanner follows to create a token
Interpretation of type Keyword	All reserved keywords of that language.	int, goto	The sequence of character that make the keyword.
- of type identifier	Name of a variable, function, etc	main a	It must start with the alphabet, followed by the alphabet or a digit.
- operator	All operators are considered tokens.	+, =	+, =
- Literal	A grammar rule or boolean literal	"Welcome!"	any string of character.

Q.3

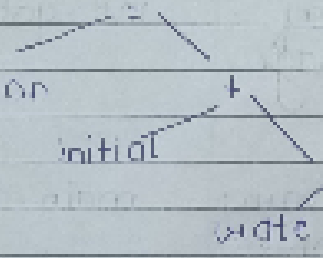
2) Syntax analysis

• The second phase of the compiler is syntax analysis or parsing.

• The parser uses the first components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.

Example position = initial + rate * 60

Syntax tree



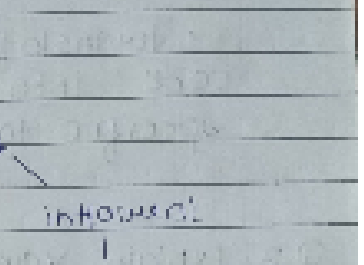
3) Semantic analysis

• The third phase of compiler is semantic analysis.

• It also gathers type info and saves it in either the syntax tree or the symbol table for subsequent use during intermediate code generation.

• An important part of semantic analysis is type checking.

Example



4) Intermediate code generation

• Translates the written program structure into an intermediate representation which is a language independent code format.

• Simplifies the process of code generation for different target architectures.

5) Code optimization

• Analyzes the intermediate code to improve its efficiency by applying various

optimization techniques

- This stage is not always present and its implementation can vary significantly based on the compiler design

5) Code generation

- Translate the optimized intermediate code into the target machine language specific to the intended hardware platform

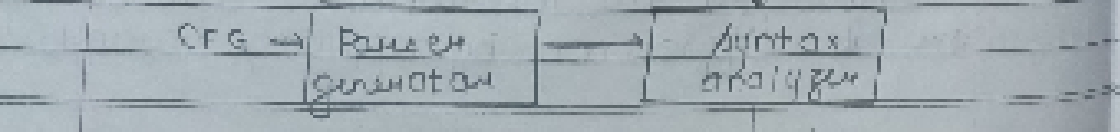
Q 2 Explain various compiler construction tools in details

Ans: They are specialized tools that help in the implementation of various phases of the compiler.

1) Parser Generator

- It produces syntax analyzers based on a grammar that takes input in the form of the syntax of a programming language

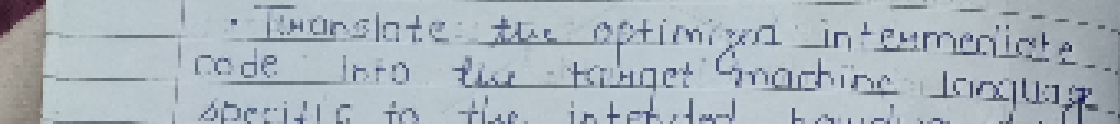
Example : LEPM, PIC



2) Scanner Generator

- It generates lexical analyzers from the input that consists of RE description based on tokens of a language. It generates FA to identify RE

- Example: lex



3) Syntax directed translation engines

- It takes the parse tree as input and generate intermediate code with three address formats. Each parse tree node has one or more translation associated with it

- Example : Attribute grammers

4) Automatic code generators.

- It takes intermediate code as input and convert it into machine language. Each intermediate language operation is translated using a set of rules and then sent into the code generator as input

- Example: LLVM, ER etc Backend

5) Data-flow analysis engines.

- It is used for code optimization and can generate the optimized code. It is essential part of code optimization that collects the info. the values that flow from one part of program to another.

Example: Dead code elimination

UNIT - III

Intermediate code generation

Sem

Q.1 Syntax directed translation (SDT)

- It has augmented rules to the grammer that facilitate semantic analysis
- It involves passing information bottom up and top-down to the parse tree in form of attributes attached to the node
- SDT rules use
  i) Lexical values of node
  ii) constant
  iii) Attribute associated with the non-terminals in their definitions

The general approach to syntax-directed translation is to construct parse tree or syntax tree and compute the values of attributes at the nodes of the tree by visiting them in some order.

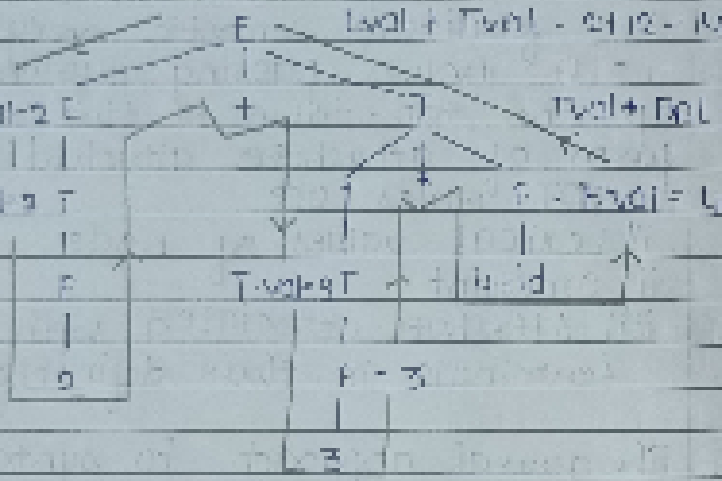
In many cases, translation can be done during parsing without building a explicit tree.

Example: (write the example in last)

$E \rightarrow E + T$	$\{E.val = E.val + T.val\}$
$E \rightarrow T$	$\{E.val = T.val\}$
$T \rightarrow T * F$	$\{T.val = T.val * F.val\}$

T → F	T.val = F.val
F → id	F.val = id.lexval

We have S: 2 + 3 * 4



Syntax directed definition

- It is a kind of context free abstract specification.
- It is a generalization of CFG in which each production $X \to \alpha$ is associated with it a set of production rules of the form $s = f(b_1, b_2, b_3, \ldots, b_k)$ where s is the attribute obtained from function f.
- The attribute can be a string, number, type or a memory location.

	Synthesized attribute	Inherited attribute
Definition	Its value is determined by the attribute value at child nodes.	Its value is determined by the attribute value of parent or sibling node.
Production	The production must have non-terminal as its head.	The production must have non-terminal as a symbol in its body.
Evaluated	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.
Considered by	It can be contained by both terminals and non-terminals.	It can't be contained by both, it is only contained by non-terminals.
Used by	Used by both S-attribute SDT and L-attribute SDT.	Only used by L-attribute SDT.
Example	E.val → E.val E.val	E.val → E.val E.val

Features	S-attributed definition	L-attributed definition
* Attribute type	Synthesized only	Both synthesized and inherited
* Value calculate	Based on children attributes	Based on left sibling and parent's attributes
* Evaluate	Bottom up	Depth-first, left-to-right
* Restriction	None	Inherited attribute depends only on left siblings
* Example	E.val = E₁.val + ... E.val ↑ E₁.val	E.val → E₁.val ... E.val ↓ E.val

Q: SDT scheme is implemented in which phase of compiler and why?

Ans: SDT is implemented in the semantic analysis phase of compiler.

Here's why:

- Purpose of semantic analysis:
  - It verifies the program's meaning beyond its syntactic correctness.
  - It focuses on aspects like type checking, variable usage, etc.
- & SDT
- Benefits in semantic analysis:
  - By integrating semantic actions into parsing, SDT allows for early detection of semantic errors during the parsing phase itself.
  - This is more efficient than has a separate semantic analysis pass after successful parsing.