📘 Compiler Design – Unit I

🔥 Important 14-Mark Questions (Exam Ready Answers)

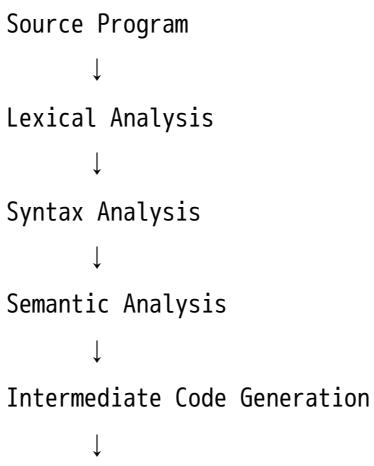
Q1. Explain the Phases of Compiler with Neat Diagram.

Answer:

Introduction

A compiler is a system software that converts a High Level Language (HLL) program into Machine Level Language (MLL).
The compilation process is divided into several phases. Each phase performs a specific task.

Diagram of Compiler Phases



```
Code Optimization
       ↓
Code Generation
       ↓
Target Program
```

---

1. Lexical Analysis

Lexical Analysis is the first phase of a compiler.

It converts the input program into tokens.

Functions:

- Removes spaces and comments
- Generates tokens
- Detects lexical errors

Example:

Program:

```
int a = 10;
```

Tokens:

```
int
a
=
10
;
```

---

2. Syntax Analysis

Syntax Analysis checks whether the program follows the grammar rules of the language.
It is also called Parsing.

Functions:

- Checks syntax errors
- Creates parse tree

Example:

Correct:

```
int a;
```

Incorrect:

```
int ;
```

---

3. Semantic Analysis

Semantic Analysis checks the meaning of the program.

Functions:

- Type checking
- Variable declaration checking

- Scope checking

Example:

```
int a;
a = "Hello";
```

This produces a type mismatch error.

---

4. Intermediate Code Generation

This phase generates machine-independent intermediate code.

Example:

```
t1 = a + b
```

---

5. Code Optimization

This phase improves the intermediate code to make execution faster and more efficient.

Example:

Before optimization:

```
a = 2 * 10;
```

After optimization:

```
a = 20;
```

---

6. Code Generation

This phase generates final machine code from intermediate code.

Symbol Table

The symbol table stores information about identifiers.

Identifier	Type
a	int

Error Handler

Error handler detects and reports errors during compilation.

Conclusion

Compiler phases systematically convert source code into machine code and help in efficient program execution.

Q2. Explain Lexical Analysis in Detail.

Answer:

Introduction

Lexical Analysis is the first phase of a compiler.

It reads the source program character by character and converts it into tokens.

Lexical Analyzer is also called:

```
Scanner
Tokenizer
```

Diagram

```
Source Program
       ↓
Lexical Analyzer
       ↓
Tokens
```

Example

Program:

```
int x = 5;
```

Tokens:

```
int
x
=
5
;
```

Important Terms

---

1. Token

A token is the smallest meaningful unit of a program.

Examples:

```
if
while
+
=
```

---

2. Lexeme

A lexeme is the actual sequence of characters in the source program.

Examples:

```
count
100
```

---

3. Pattern

A pattern is a rule used to identify tokens.

Example:

```
digit → 0|1|2|3...
```

Functions of Lexical Analyzer

- Generates tokens
- Removes comments and spaces
- Detects lexical errors
- Sends tokens to parser

Advantages

- Fast scanning process
- Simplifies syntax analysis
- Efficient token generation

Conclusion

Lexical Analysis is an important compiler phase that converts source code into meaningful tokens for further processing.

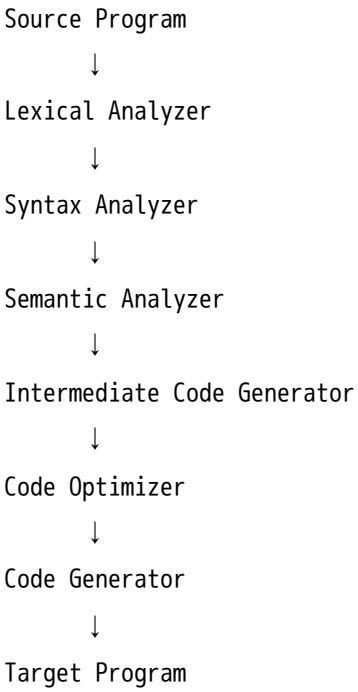
Q3. Explain Compiler Structure.

Answer:

Introduction

Compiler structure is the organized arrangement of different compiler phases used to translate source code into machine code.

Diagram



Components of Compiler Structure

Component	Function
Lexical Analyzer	Generates tokens

Component	Function
Syntax Analyzer	Checks grammar
Semantic Analyzer	Checks meaning
Intermediate Code Generator	Produces intermediate code
Code Optimizer	Improves efficiency
Code Generator	Produces machine code

Symbol Table

Stores information about identifiers such as:

- Variable name
- Data type
- Memory location

Error Handler

Detects and reports compilation errors.

Conclusion

Compiler structure provides a systematic and organized way to convert source code into machine code.

Q4. Explain Analysis-Synthesis Model.

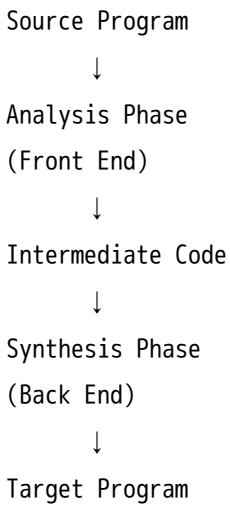
Answer:

Introduction

The compiler is divided into two major parts:

1. Analysis Phase
2. Synthesis Phase

Diagram



---

1. Analysis Phase

The Analysis phase analyzes the source program.

Includes:

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis

Functions:

- Checks errors
- Creates intermediate representation

---

2. Synthesis Phase

The Synthesis phase generates target machine code.

Includes:

- Code Optimization
- Code Generation

Functions:

- Improves code efficiency
- Produces machine code

Advantages

- Simplifies compiler design
- Better error detection
- Efficient compilation process

Conclusion

The Analysis-Synthesis model divides compilation into manageable phases and improves compiler efficiency.

Q5. Explain Bootstrapping and Porting.

Answer:

Bootstrapping

Definition

Bootstrapping is the process of developing a compiler using the same programming language that the compiler is intended to compile.

Example

Developing a C compiler using C language.

Advantages

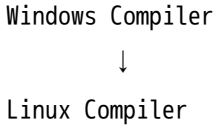
- Faster compiler development
- Easy maintenance
- Machine independence

Porting

Definition

Porting is the process of transferring a compiler from one machine or operating system to another.

Example



Advantages

- Reusability
- Multiple platform support
- Easy migration

Conclusion

Bootstrapping helps in compiler development, while porting allows a compiler to work on different systems.

Q6. Explain LEX Tool with Example.

Answer:

Definition

LEX is a tool used to automatically generate a lexical analyzer.

Working of LEX

Input:

```
Regular Expressions
```

Output:

Lexical Analyzer Program

Structure of LEX Program

```
Definitions
%%
Rules
%%
User Functions
```

Example

```
digit [0-9]

%%
{digit}+ printf("NUMBER");
%%
```

Advantages

- Automatic scanner generation
- Saves time
- Easy implementation

Conclusion

LEX is a useful tool for generating lexical analyzers efficiently.

Q7. Explain Tokens, Lexemes and Patterns.

Answer:

1. Token

A token is the smallest meaningful unit of a program.

Examples:

```
if
while
+
=
```

---

2. Lexeme

A lexeme is the actual character sequence matched by a token.

Examples:

```
count
100
```

---

3. Pattern

A pattern is a rule used to recognize tokens.

Example:

```
digit → 0|1|2|3...
```

Table

Lexeme	Token
int	Keyword
sum	Identifier
100	Number

Conclusion

Tokens, lexemes and patterns are important concepts used in lexical analysis.

Q8. Explain Input Buffering.

Answer:

Definition

Input buffering is a technique used to improve the speed of reading source code during lexical analysis.

Need for Input Buffering

Reading characters one by one is slow.

Buffering improves the efficiency of input processing.

Buffer Pair Technique

Two buffers are used alternately.

```
Buffer 1 ↔ Buffer 2
```

One buffer is processed while the other is loaded.

Advantages

- Faster input reading
- Efficient scanning
- Reduces processing time

Conclusion

Input buffering improves the efficiency and speed of lexical analysis in a compiler.