1) What is a Compiler? Explain the Analysis Phase.

Definition of Compiler

A **compiler** is a software program that converts **high-level source code** (written in languages like C, C++, Java) into **machine code** (executable by a computer). It ensures that the program follows the syntax and semantics of the programming language and translates it into an optimized form for execution.

Phases of a Compiler

A compiler consists of two main phases:

1. **Analysis Phase (Front-end)** – Understands the source code and converts it into an intermediate representation.
2. **Synthesis Phase (Back-end)** – Converts the intermediate representation into machine code.

Analysis Phase (Front-end of Compiler)

The **Analysis Phase** is responsible for breaking down the source code into meaningful parts and checking for correctness. It consists of:

1. **Lexical Analysis**
   - Converts the source code into tokens (smallest units like keywords, identifiers, operators).
   - Uses **finite automata** to recognize tokens.
   - Example: For `int x = 10;`, tokens are: `int`, `x`, `=`, `10`, `;`.
2. **Syntax Analysis (Parsing)**
   - Checks if the tokens form valid grammatical structures according to the language's grammar.
   - Uses **parsing techniques** like **LL(1), LR(1)**.
   - Example: `int x = 10;` is valid, but `int = x 10;` is invalid.
3. **Semantic Analysis**
   - Ensures that the code makes logical sense (e.g., checks type compatibility).
   - Example: `int x = "hello";` is invalid because `"hello"` is a string, not an integer.
4. **Intermediate Code Generation**
   - Translates the code into an intermediate representation (IR) like **Three-address code (TAC)** or **Abstract Syntax Tree (AST)**.

Example: `a = b + c;` can be converted into:

```
t1 = b + c
a = t1
```

○

---

2) What is Code Optimization? Write Any Three Machine-Independent Optimization Techniques with Examples.

Definition of Code Optimization

Code optimization is the process of improving the intermediate code to make it **more efficient** without changing its functionality. It helps in:

- Reducing execution time (**faster execution**).
- Reducing memory usage (**better resource utilization**).

Optimization is done **before** generating machine code and is classified as:

- **Machine-Independent Optimization** (does not depend on CPU architecture).
- **Machine-Dependent Optimization** (specific to hardware architecture).

Machine-Independent Optimization Techniques

These optimizations focus on improving the **intermediate code** and are not dependent on machine architecture.

1. **Constant Folding**
   - Replaces expressions with their computed constant values at compile-time.

     Example:
     ```
     int x = 5 * 10;
     ```
   - Compiler replaces `5 * 10` with `50`:
     ```
     int x = 50;
     ```
2. **Common Subexpression Elimination (CSE)**
   - Eliminates duplicate expressions that compute the same value.

     Example:
     ```
     int x = (a + b) * c;
     int y = (a + b) * d;
     ```

Here, `(a + b)` is computed twice.

Optimized code:

```
int t = a + b;
int x = t * c;
int y = t * d;
```

3. **Dead Code Elimination**
    - Removes code that **does not affect the output**.

      Example:

      ```
      int x = 5;
      x = 10;  // Overwrites x before using it
      printf("%d", x);
      ```

   The first assignment (`x = 5;`) is unnecessary.

   Optimized code:

   ```
   int x = 10;
   printf("%d", x);
   ```

---

3) What is a Symbol Table? What Information is Stored in a Symbol Table?

Definition of Symbol Table

A **symbol table** is a **data structure** used by the compiler to store information about identifiers (variables, functions, objects, etc.) used in the program. It helps in **semantic analysis, type checking, and optimization**.

Information Stored in Symbol Table

1. **Variable Name (Identifier Name)** – The name of the variable or function.
2. **Data Type** – The type of the identifier (int, float, char, etc.).
3. **Scope** – The part of the program where the identifier is accessible (local, global).
4. **Memory Address** – The memory location where the variable is stored.
5. **Value** – If the variable is a constant, its value is stored.
6. **Function Parameters** – Number and types of parameters for functions.

7. **Array Size and Dimensions** – If an identifier is an array, its size and dimensions are stored.

Example of a Symbol Table

Identifier	Type	Scope	Memory Address	Value
`x`	int	Global	1000	10
`y`	float	Local	2000	-
`add()`	Func	Global	3000	-

Symbol Table Operations

1. **Insert()** – Add a new identifier.
2. **Lookup()** – Search for an identifier.
3. **Modify()** – Update information.
4. **Delete()** – Remove an identifier.