Compiler

Mahmudul Hasan (Moon)

***Difference between Compiler and Interpreter:**

No	Compiler	Interpreter
1	Compiler Takes Entire program as input	Interpreter Takes Single instruction as input .
2	Intermediate Object Code is Generated	No Intermediate Object Code is Generated
3	Compiler is faster at mapping inputs to outputs.	Interpreter is slower than a compiler.
4	Conditional Control Statements are Executes faster	Conditional Control Statements are Executes slower
5	Memory Requirement : More(Since Object Code is Generated)	Memory Requirement is Less
6	Program need not be compiled every time	Every time higher level program is converted into lower level program
7	Error diagnostics is less than a compiler.	Error diagnostics is better than a compiler.
8	Example : C Compiler	Example : BASIC

1.Define compiler and interpreter. Write the difference between them. (page 2)
2.Define Preprocessor, Assembler, Linker, Loader, Byte code, lexical analysis, lexical analyzer, Intermediate code generation

In computer science, a **preprocessor** is a program that processes its input data to produce output that is used as input to another program.

An **assembler** translates the **assembly** program into machine code (object).

A **linker** tool is used to link all the parts of the program together for execution.

In computer systems a **loader** is the part of an operating system that is responsible for loading programs and libraries.

Byte code is program **code** that has been compiled from source **code** into low-level **code** for a software interpreter. It may be executed by a virtual **machine** (such as a JVM) or further compiled into **machine code**, which is recognized by the processor.

Lexical analysis is the first phase of a **compiler**. It takes the modified source code from language preprocessors that are written in the form of sentences.

The **lexical analyzer** breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

Intermediate code generation is the process by which a **compiler**'s **code generator** converts some **intermediate representation** of source **code** into a form that can be readily executed by a machine.

3.Why Java use both compiler and interpreter? (page 2 ex:1.1)
4. Write the names of the different phases of a Compiler. (page 5 figure:1.6)
5.Figure 1.7 er process ta practice korte hobe different equation dia.

Chapter 2

1.Describe the model of a compiler front end . (page 41er figure 2.3 draw and discribe)

2.Define context free grammar, describe the four component of a context free grammer.(page 42 er 2.2.1 er point 4 ta)
A **context-free grammar** (CFG) is a set of recursive rules (or productions) used to generate patterns of strings. or
A **context-free grammar** (CFG) is a certain type of formal **grammar**, a set of production rules that describe all possible strings in a given formal language.

3.Define parse tree. Write the properties of a parse tree. (page 43 er 2.2.3 er point 4 ta)
A **parse** is an ordered, rooted **tree** that represents the syntactic structure of a string according to some context-free grammar. Or
A **parse tree** is an entity which represents the structure of the derivation of a terminal string from some non-terminal symbol.

4.What is ambiguity of a grammar? How can we remove the ambiguity of a grammar?
In computer science, an **ambiguous grammar** is a context-free **grammar** for which there exists a string that can have more than one leftmost derivation or parse tree.

An **unambiguous grammar** is a context-free **grammar** for which every valid string has a unique leftmost derivation or parse tree.

How can we remove ambiguity from an ambiguous grammar:
In an ambiguous grammar it has more than one left most derivation or parse tree. So, when we derive the language from a grammar, we should be aware about ambiguity that's means we choose that grammar which can make the grammar unambiguous. When we make a parse tree, we must aware about the precedence of the operator. If there is any scope to give associativity we must apply this. After giving precedence and associativity we can make one left most tree that's means the ambiguity is removed.

5.Define associativity and precedence of operators.
The **associativity** of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.

When several operations are in an expression, which operator is executed first this order is called **operator precedence**.

Chapter 3

1.Difference between lexical analysis and parsing (Syntax analysis).

Lexical Analysis	Syntax analysis
1.Lexical analysis is the first phase of a compiler	1.It start after lexical analysis is completed.
2.It takes the modified source code from language preprocessors that are written in the form of sentences.	2. A syntax analyzer or parser takes the input from a lexical analyzer in the form of token streams.
3.It determines the individual tokens in a program.	3.It determine the phase of a program.
4. Token structure can be described by regular expressions	4. Phrase structure must be described using a context-free grammar

2.Define tokens, patterns, lexemes. (page 111)

3.What is lexical errors? Write the error recovery action. (page 113,114 sec: 3.1.4)

4.What is input buffer?

Input buffer is the process…..

• To ensure that a right lexeme is found or not.

• Hence a two-buffer scheme is introduced to handle large look ahead safely.

• Techniques for speeding up the process of lexical analyzer.

5.Write the terms for parts of strings. (page 119)

6.How much operation can be done in a language? (page 120 er uporer 4 ta operation)

7.Draw the figure of different transition diagram.

8.Define finite automata. Write the term that consist of a NFA. (page 147)

Finite automata is a **state machine** that takes a string of symbols as input and changes its state accordingly.

9.Construct the NFA of a string and convert it to DFA. (155 er exercise)

10.Algorithm 3.23 er description (same as automata)

Chapter 4

1.Define Panic mode recovery, Phrase level recovery, Error Production, Global Correction.

Panic Mode Recovery

- In this method, successive characters from input are removed one at a time until a designated set of synchronizing tokens is found. Synchronizing tokens are deli-meters such as ; or }
- Advantage is that it's easy to implement and guarantees not to go to infinite loop
- Disadvantage is that a considerable amount of input is skipped without checking it for additional errors

Error production

- If user has knowledge of common errors that can be encountered then, these errors can be incorporated by augmenting the grammar with error productions that generate erroneous constructs.
- If this is used then, during parsing appropriate error messages can be generated and parsing can be continued.
- Disadvantage is that it’s difficult to maintain.

Global Correction

- The parser examines the whole program and tries to find out the closest match for it which is error free.
- The closest match program has less number of insertions, deletions and changes of tokens to recover from erroneous input.
- Due to high time and space complexity, this method is not implemented practically.

2.Given a grammar, construct the lest most and right most derivation.

3.Given a grammar, eliminate its left recursion.

4. Given a grammar, eliminate its left factoring.