Assignment - 01

Date :
Page :

Q1) Describe various phases of a compiler.

1) Lexical Analysis : Input : source program
Output : sequence of tokens.
Group character into tokens.

2) Syntax Analysis : checks whether tokens follow grammar rules of the language
ex) $x = a + b * c$;

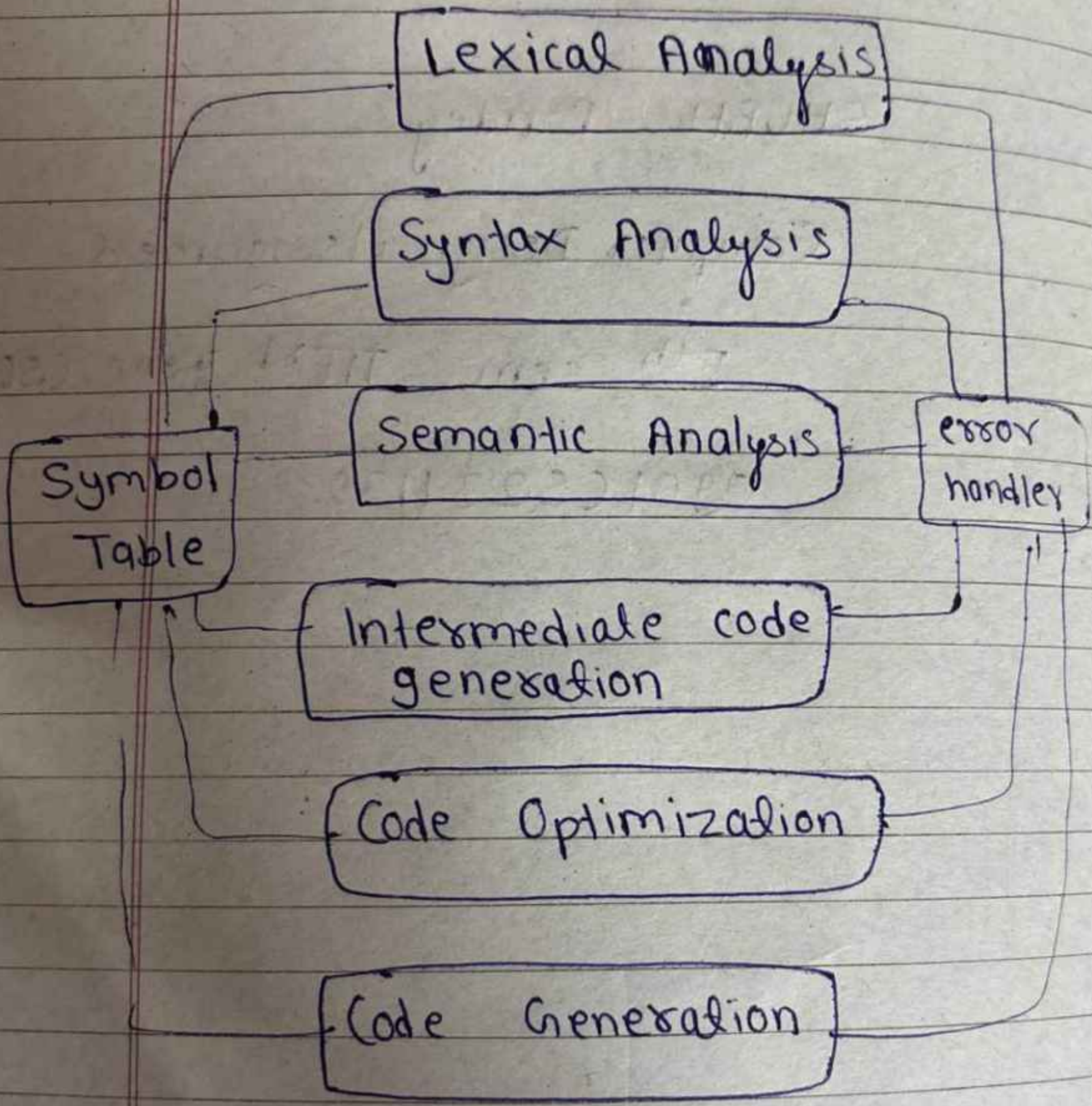
3) Semantic Analysis : checks semantic consistency, Type checking, etc.

4) Intermediate Code Generation : converts source code into a machine independent form.

5) Code Optimization : Improves code efficiency, eliminates redundancy, reduces memory usage etc.

Date :
Page :

Source Code



Date :
Page :

Q Describe bootstraping & Porting

Bootstraping: means writing a compiler for a programming lang in the same lang.

It is of 3 steps
1) Minimal Compilex
2) Self- compilex
3) Full - compilex

Porting: means adapting software to run on diff machine. ex Java Programs.

Q] Compare Single & multipass Compiler.

Ans

	Single Pass	Multipass
①	Single Scan is done	Multi Scan is done
②	Faster compilation	Slower Compilation
③	less memory needed	more memory needed
④	Simple to implement	more complex, modern.
⑤	ex: Pascal	ex C, C++.

Date :
Page :

Q Write Short notes on

① Cross Compiler: It is a compiler that runs on one machine but generiale code for a diff machin
ex: GCC can act as cross compiler.

② Incremental Compiler: It compiler only the changed parts of a program instead of recompiling the source code. Reduce compilat^h time
ex : Eclipse, VS code

③ Major Data Structures : Symbol Table, Syntax Tree, Intermediale Representation, Error Log Table, other supporting structures.

④ Interpreter: An interpreter directly executes source code instructions line by line, without converting the whole ~~i~~ program into machine code.

Date :
Page :

Assignment - 02

Q Explain input buffering in detail

Ans During lexical analysis, the source program is read char by char If every char is read one by one, it becomes very slow because disk I/O is expensive. Input buffering solves this char are read into a buffer in larger blocks & L.A fetches character from this buffer.

Q] Explain how tokens are recognized

Ans A token is the smallest meaningful unit in a program. The L.A reads the source code as a stream of characters & group them into Lexemes.

① Input Buffering
② Finite Automata
③ Pattern Matching
④ Token creation.

Date :
Page :

Q] Write Short Notes on:

① Regular Grammar: A regular Grammar that can be expressed using regular expressions & accepted by finite automata.

Two types of Grammar:

① Right Linear Grammar.
② Left Linear Grammar.

② Error Reporting: During Compilation, the compiler detects error & reports them to the user.

Types of error:

① Lexical error
② Syntax error
③ Semantic error
④ Runtime error.

③ Role of Lexical Analysis

It is 1st phase of compiler.

Date :
Page :

① Converts input character
② Removes whitespace & comments
③ Maintains the symbol Table
④ Provides error reporting for illegal token
⑤ Simplifies parsing by providing for illegal stream of tokens.

(4) Token, Pattern, Lexeme

These are basic terms in Lexical Analysis

Token: A category of symbols with meaning.

Pattern: A rule that describes how token are formed.

Lexeme: The actual substring of the source program that matches a program.

Assignment-03

Date :
Page :

Q1) S → ABBa|bCA
A → ϵBCDE
B → CDA|cd
C → eC|ϵ
D → bSF|a

	S	A	B	C	D
First	b, ϵ, c, e	ϵ, c, e	c, e, ϵ (b, a)	e, ϵ	b, a
follow	$	e, b, a, c	e, b, a, c	e b a $	e b a c

Q2] Construct LR parsing Table for the following Grammar.

S → aAC | bB
A → eD
B → f | g
C → h | i
D → bE | ε
E → eD | dD

Ans Compute First & follow :

	S	A	B	C	D	E
First	a, b	e	f, g	h, i	b, ε	e, d
Follow	$	h, i	$	$	h, i	h, i

Page :

M[S, a]	= S → a AC
M[S, b]	S → bB
M[A, e]	S → eD
M[B, f]	B → f
M[B, g]	B → g
M[C, h]	C → h
M[C, i]	C → i
M[D, b]	D → bE
M[D, h]	D → ε
M[D, i]	D → ε
M[E, e]	E → eD
M[E, d]	E → dD

Q] Construct LR(0) parsing table for the following Grammar

$S \rightarrow CA \mid CCB$
$A \rightarrow CA \mid a$
$B \rightarrow CCB \mid b$

Sol Create Augmented Grammar

$S' \rightarrow \cdot S$
$S \rightarrow \cdot CA$
$S \rightarrow \cdot CCB$
$A \rightarrow \cdot CA$

Date :
Page :

A → a

B → ·CCB

B → b

I_0 : S → · S
S → · CA
S → · CCB

I_1 → S → S.

I_3 S → CA.

I_2 S → C · A
S → C · CB
A → · CA
A → · a

I_4 A → a.

I_5 S → CC · B
A → C · A
A → · CA
A → · a
B → · CCB
B → · b

I6 A → CA.

I7 B → b.

I8 B → CC · B
A → C · A
A → · CA
B → · CCB
B → · b

	Action			
	(a)	b	C	$
0	—	—	S2	—
1	—	—	—	acc
2	S4	—	S5	acc
3	r1	r1	r1	r1
4	r4	r4	r4	r4
5	S4	S7	S8	—
6	r3	r3	r3	r3
7	r6	r6	r6	r6
8	S4	—	S10	—
9	r2	r2	r2	r2
10	S4	S7	S8	—
11	r5	r5	r5	r5

Date :
Page :

Q construct LR(1) passing Table for the following Gramma

S → aSbS| bSaS|ϵ

Sol^n Augmented Grammar:

S' → .S
S → .aSbS
S → .bSaS
S → ϵ

Goto (I0, a) :. S → a.SbS, $
LI contain
S → a.SbS, $
S → .aSbS, b
S → .bSaS, b
S → .ϵ, b

I2: S → b.SaS, $
S → .aSbS, a
S → .bSaS, a
S → .ϵ, a

I3: S' → S.$

Date :
Page :

State	a	b	$	Goto (s)
0	S1	S2	r3	3
1	S1	S5	-	4
2	S6	S2	-	7
3	-	-	acc	-
4	S1	S5	r3	6
5	S6	S2	r3	8
6	r1	r1	r1	-
7	r2	r2	r2	-
8	r2	r2	r2	-

Date :
Page :

Assignment - 4

Q Explain various storage allocation techniques on symbol Table.

1) Static Allocation: A symbol table stores info about identifiers during compilation. Fixed size & location. Lifetime for entire program

2) Stack Allocation: memory is allocated in stack segment at runtime when a function is called

local variables & function parameters are stored this way.

3) Heap Allocation.

memory allocated at runtime as needed.

Date :
Page :

Q2 Explain in brief various data structure that can be used in symbol table.

Ans Array / List

- Simple array of records with one entry per identifer
- Search: Linear O(n)

Simple but inefficient for large programs.

Linked List:

Each symbol is a node in a Linked List

Support dynamic growth

search: Linear in List length

can implement separate

Date :
Page :

chaining in hash tables.

Hash Table:

Most commonly used in modern compilers

Symbols hashed into table entries.

Search / insert / delete: Average O(1), wosrst O(n) if Collisions.

Collision handled via chaining.

Binary Search Tree:

Ordered Structure of Symbol

Search / insert / delete: O(log n)

useful if ordered traversal of symbols is needed.

Date :
Page :

Stack :

- Useful to handle scopes in nested blocks
- Push Symbols when entering scope, pop when leaving scope.
- Often combined with hash tables for fast LOOKUP with scope awareness.

Q Differentiate Static & Dynamic binding.

Static Binding	Dynamic Binding
The variable or function to memory is determined at compile time	The binding is determined at runtime.
Compile time	runtime.

Date :
Page :

Less flexible	more flexible
Faster	Slower
Static variables	Objects in OOP Languages

Q Write Short Note on

① Activation Record

A data structure that stores information about a single function call during program execution

manages memory for local variables parameters return address & control information.

① Return address
② Parameters
③ Local variable
④ Saved Registers

Date :
Page :

⑤ Dynamic Link.

② Syntax & Semantic Error

Violation of programming Language grammar.

Detected at compile time by the parser.

Semantic Errors:

Errors related to meaning of program rather than structure.

Detected at compile time or runtime depending on runtime.

- Ex Type mismatch, undeclared variables, incompatible operations.