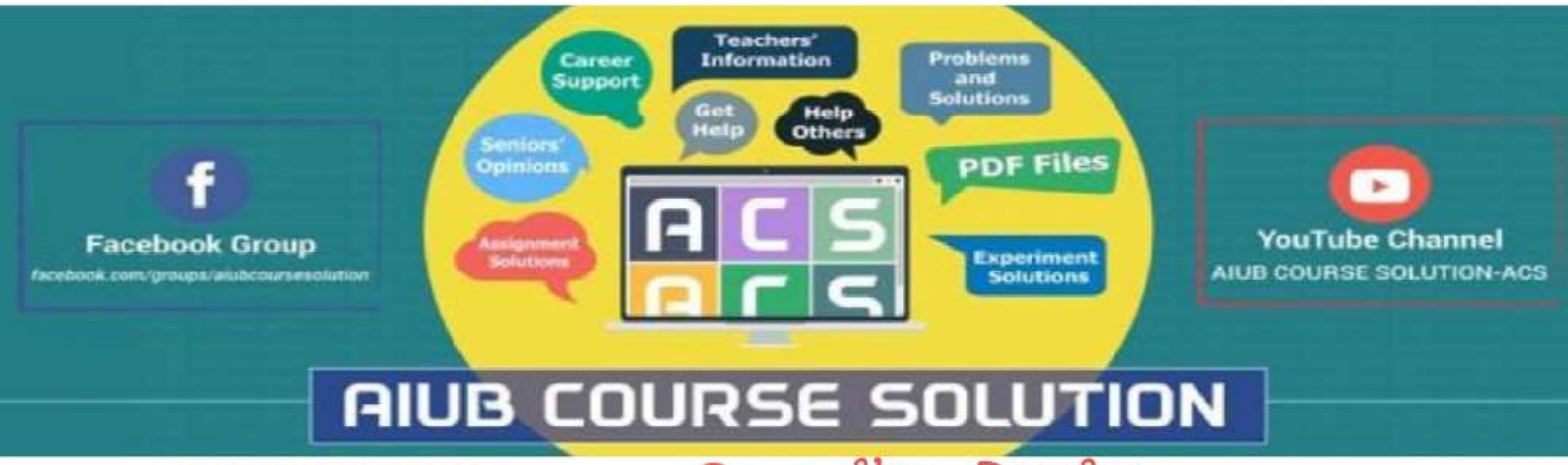


COURSE NAME: Compiler Design

SEMESTER: 7th

SOLVED BY

Nibraz Nasiha

Compiler Design

Compiler: is a program that reads a program in one languages & translate in one language. (higher level to lower level)

Program is some source language → Compiler → execute code for target machine

Example: X = a+b * 10 → Compiler →

```
MOV id3 R2
MUL #10.0, R2
MOV id2, R1
ADD R2, R1
MOV R1, id1
```

Interpreter: An interpreter is common kind of language. Instead of producing a target program as a translation, an interpreter appears to directly execute the operation specified in the source program on inputs

supplied by the users.

Source program ⟶ Co Interpreter → Output
input ⟶

Preprocessor: performs operations on the source file prior to compilation.

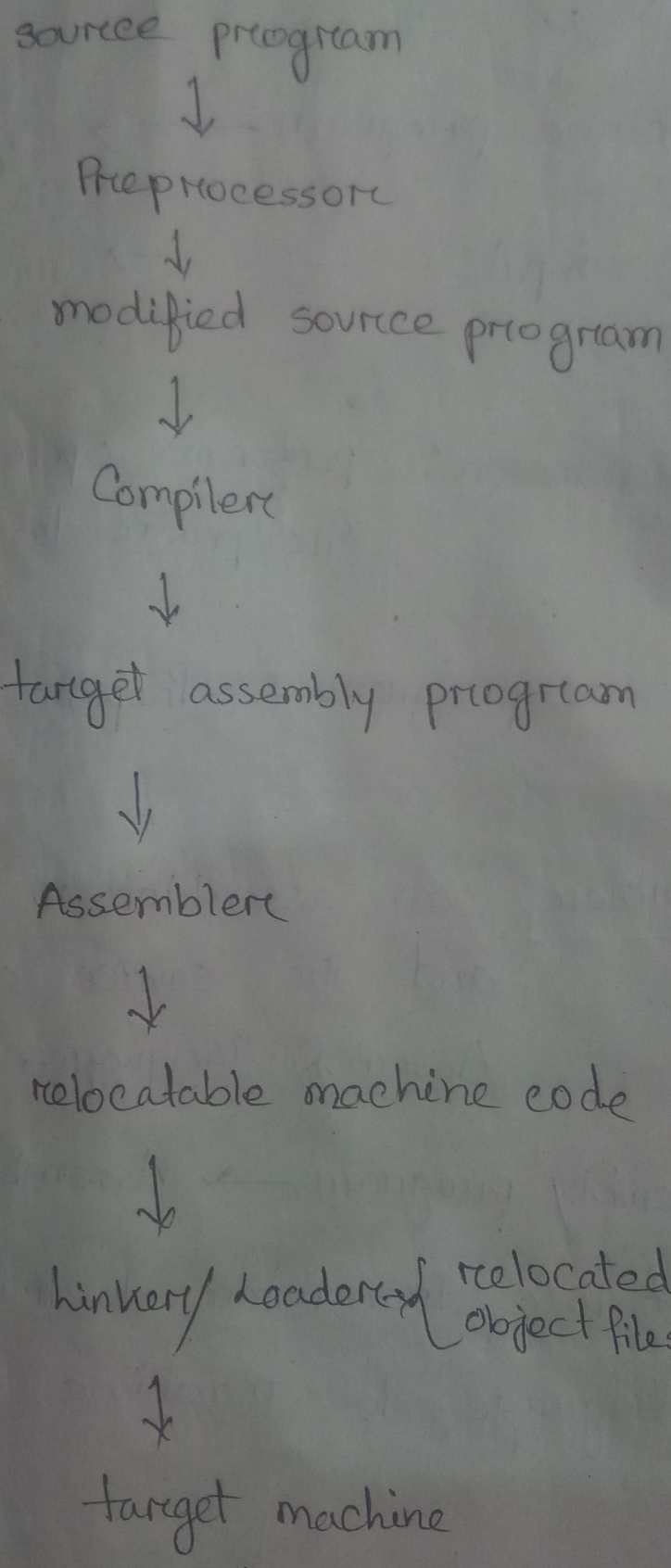
Source program → Preprocessor → Preprocessed Source Program

Assembler: an assembler is a type of compiler and the source code is written in Assembly language.

Assembly program → Assembler → Machine Program.

* is a translator to translate assembly language to machine code.

Language Processing System



am | 12.30 pm

Difference between ○ Compiler & □ Interpreter

○ Compiler	□ Interpreter
○ takes whole program as a input.	□ single instruction as input.
○ Intermediate object code is generated.	□ No intermediate object code is generated.
○ Memory requirement more	□ Memory requirement less
○ Program need not be compiled every time.	□ Every time higher level program is converted into lower level program.

Linker: Linker or Link editor is computer system program that takes one or more object files and combines them into a single executable file, library file, or another 'object' file.

Loader: Loader is a special type of program that copies programs from a storage device to the main memory, when they can be execute.

(*) What is the phase of Compilers?

→ We've two phases of compilers → Analysis
                                → Synthesis

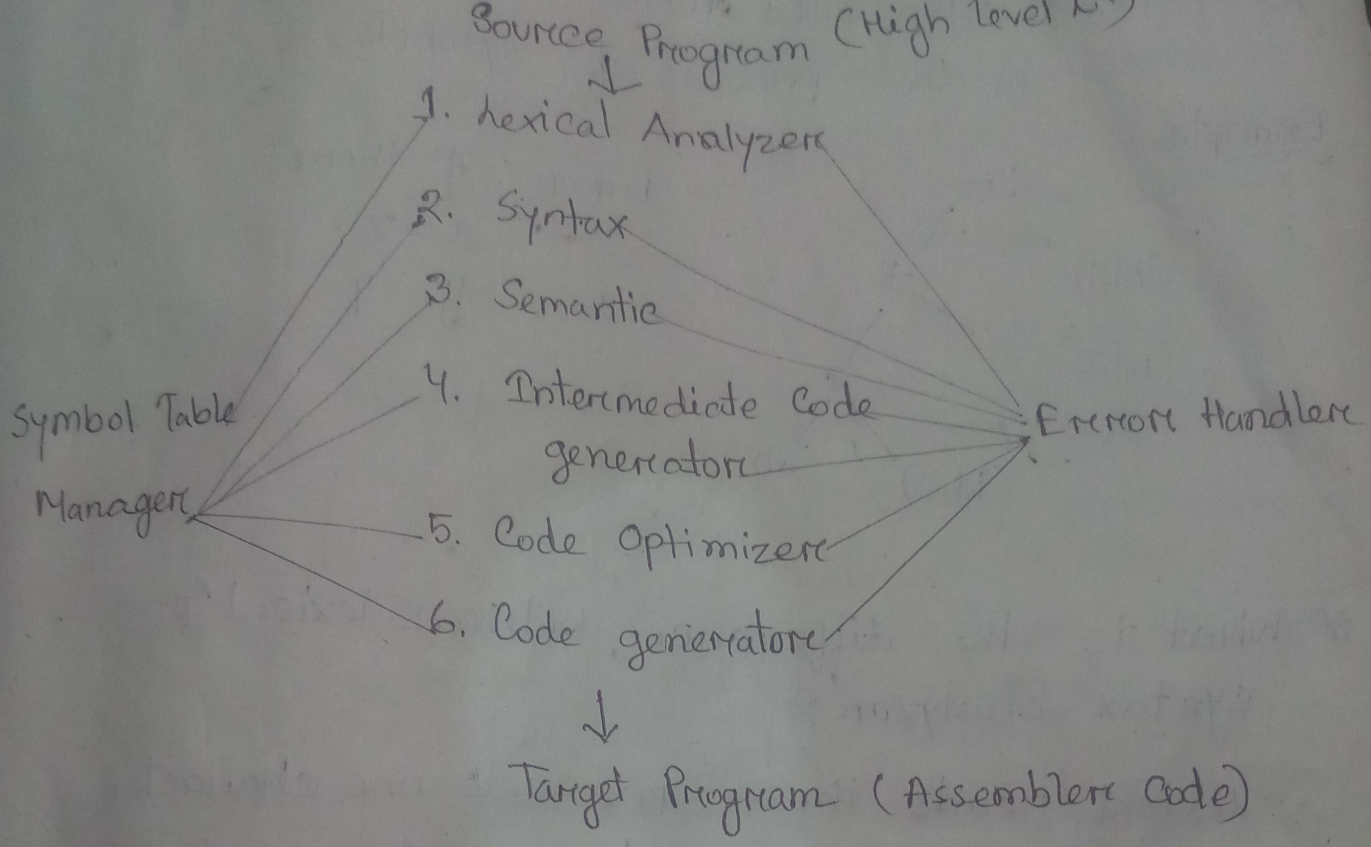
Phase	Works
1) Analysis :	creates an intermediate representation from source code.
2) Synthesis :	creates an equivalent target program from the intermediate representation.

≈ Analysis Phase → Lexical
                 → Syntax
                 → Semantic
                 → Intermediate Code Generator

≈ Synthesis Phase → Code optimizer
                  → Code geneteator

(*) Symbol Table : is used to store the information about the occurrence of various entities like objects, variables, function name, classes etc.

Process the Phase of a Compiler



Defination & Examples

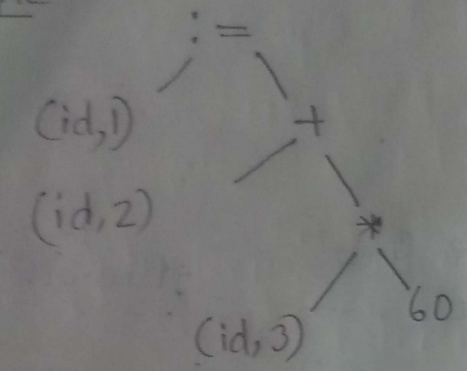
1) Lexical Analyzer: Also called 'linear Phase'/'Analysis'
It's a 1st phase of compiler.
{lexing or tokenization}

The lexical analyzer ~~breaks these sy~~ takes modified source code from language preprocessor that ~~the~~ are written in the form of sentences.

Example: posilion = initial + rate * 60
(id,1) (=)(id,2) (+) (id,3) (*) (60)

2. Syntax: Also called Hierarchical or parsing is the process of symbols.

Example



Error: if there,
x = 7
y := 4

(*) What is the difference between lexical & syntax analyzer?

→ lexical reads the source code one character at a time and converts it into meaningful tokens.

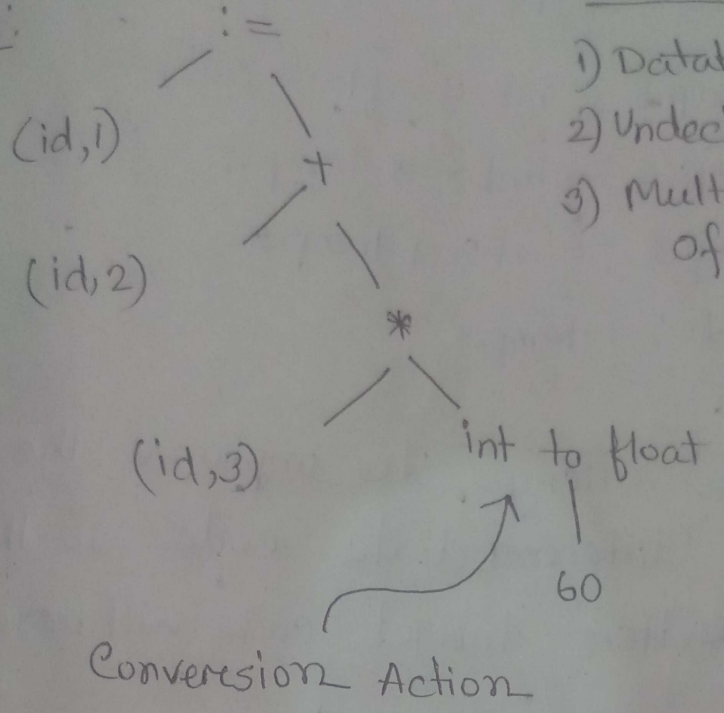
Where, syntax takes those tokens & produce a parse tree as an output.

3. Semantic: In compiler instruction, usually after parsing, to gather necessary semantic information from the source code.

Online class routine

11.45 am | 12.30 pm

Example:



Errors

1) Datatype mismatch
2) Undeclared variable
3) Multiple declaration of variable scope
4) Actual & formal parameter mismatch

4. Intermediate Code Generator: receives input from its predecessor phase, semantic ~~phase~~ analyzer, in the form of an annotated syntax tree.

It also called 'three Address Code'

Because → at most one operator on the right side
→ generate a temporary name
→ fewer than three operators.

position = initial + rate * 60
(id1) (id2) (id3)

Example: temp1 := int to float (60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3

5. Code optimizer: attemps to improve the intermediate code so that better target code will result.

Example:
position = initial + rate * 60
(id1) (id2) (id3)

temp1 := id3 * 60
id1 = id2 + temp1

6. Code generator: The final phase of compiler is to generate code for specific machine.

Example:
Position = initial + rate * 60
(id1) (id2) (id3)

MOVF R2, id3
MULF R2, #60.0
MOVF R1, id2
ADDF R1, R2
MOPF id1, R1

Online class routine

| 11.45 am | 12.30 PM |

Front end & Back end of a compiler:

Front end:
- Lexical
- ↓
- syntax
- ↓
- Semantic
- ↓
- Intermediate C.G.

(Analysis Phase) Font End

Back end:
- Code optimizer
- ↓
- Code generator

(synthesis phase) Back End

Errror handler: Each of the six phases of a compiler can encounter errrors.

* What is Context free grammar?

→ Context free grammar or 'Grammer' is a set of recursive rules used to generate patterns of strings. It describes all regular languages but can't describes all possible languages. That is used specify the syntax of language.

Example:

```
if (expression)
    // statement

else
    // statement
```

A grammar consists of:

→ set of variables
→ set of terminals
→ list of production
→ start symbol

Formal Definition: It's 4-tuple (V, Σ, S, P)

Where, V is a finite set of variable.

Σ is a finite ~~set of~~ alphabet of terminals

S is a start variable

P is the finite set of production

form → $V \rightarrow (V \cup \Sigma)^*$

Example: $0^n 1^n$ Here is a grammer,

$S \rightarrow 0S1$

$S \rightarrow \varepsilon$

S is the only variable, terminals are 0 & 1.

There are two productions.

Derivation: ~~$S \to 0S1$~~
~~$S \to \varepsilon$~~

sequence of production rules, in order to get the input string.

- Left most derivation
- Right most derivation

(*) Derive the string 'abb' for left most & right most derivation using CFG given by

$S \to AB / \varepsilon$
$A \to aB$
$B \to sb$

'abb'

Left most Derivation

S	$S \to AB$
AB	$A \to aB$
aB B	$B \to sb$
a sb B	$S \to \varepsilon$
a b B	$B \to sb$
a b sb	$S \to \varepsilon$
'a b b'	

Right M. D

S	
AB	$S \to AB$
A sb	$B \to sb$
A ε b	$S \to \varepsilon$
aB b	$S \to aB$
a sb b	$B \to sB$
'abb'	$S \to \varepsilon$

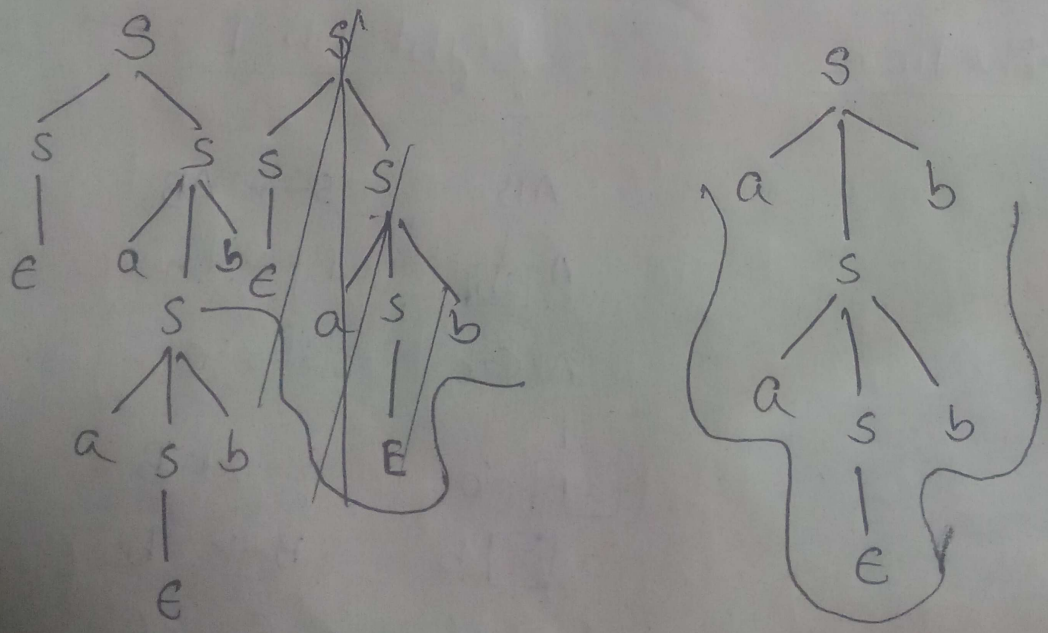
* What is parse tree?

→ parse tree/ parsing tree/ derivation tree or concrete syntax tree is an ordered, rooted tree that represents the syntatic structure of a string according to some context free grammer.

* Check the whether the given grammer is ambiguous or not.

$S \rightarrow asb / ss$

$S \rightarrow \epsilon$ 'aabb'



It's a ambiguous.