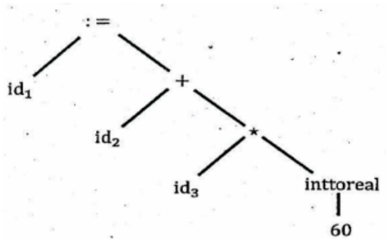


Fig. 1.6

1.3.13 The Synthesis Phases

i) **Intermediate Code Generation (or) ICG**

- After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program.
- We can think of this intermediate representation as a program for an abstract machine.
- This intermediate representation should have two important properties; it should be easy to

Symbol Table

1	position	....
2	Initial	....
3	rate	....
4		

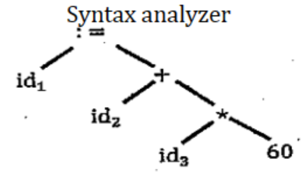
Position: initial + rate*60

↓

Lexical analyzer

Id_1:=id_2+id_3*60

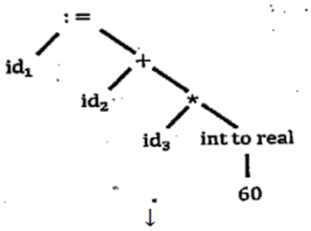
Syntax analyzer



↓

Semantic analyzer

↓



↓

Intermediate code generator

↓

Temp 1 : int to real(60)
Temp 2: id3*temp1
Temp 3 = id3 + temp2

↓

Code optimizer
Temp 1: = id3*60.0
Id1: = id2 + temp1
Code optimizer

↓

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

Fig.1.7

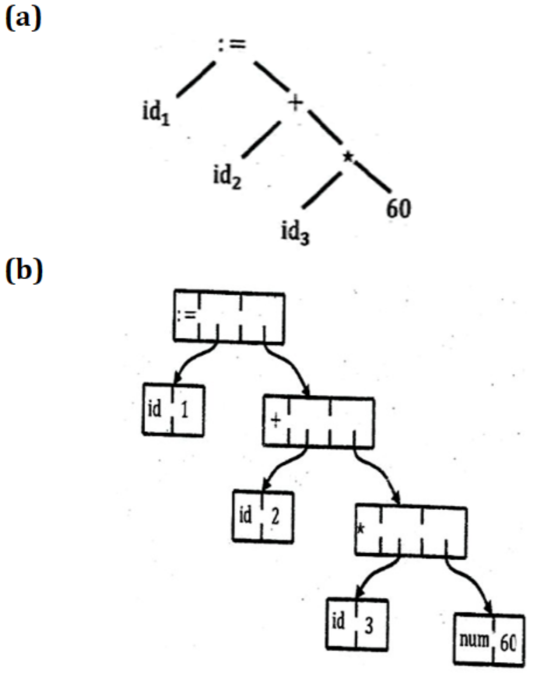


Fig 1.8 Data structure in (b) is for the tree in (a)

ii) Code Optimization :-

- This phase attempts to improve the intermediate code , so that faster running code will result
- Some optimizations are trivial. For example, a natural algorithm generates the intermediate code (1.3), using an instruction for each operator in the tree representation after semantic analysis, even though there is a better way to perform the same calculation, using the two instructions.

  Temp1: = id3 * 60.0
  Id1: = id2 + temp1