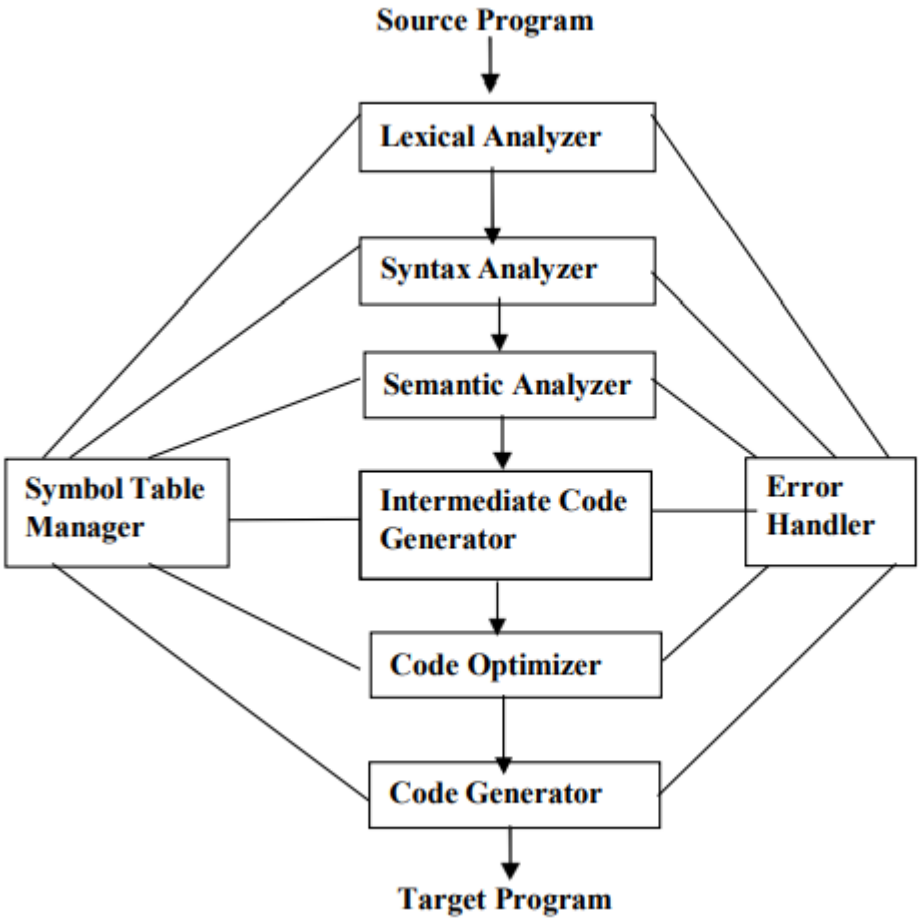
THE PHASES OF COMPILER

1) Lexical analysis - it contains a sequence of characters called tokens. Input is source program & the output is tokens.
2) Syntax analysis - input is token and the output is parse tree
3) Semantic analysis - input is parse tree and the output is expanded version of parse tree
4) Intermediate Code generation - Here all the errors are checked & it produce an intermediate code.
5) Code Optimization - the intermediate code is optimized here to get the target program.
6) Code Generation - this is the final step & here the target program code is generated.



Lexical Analysis:

In a compiler, Linear analysis is called lexical analysis or scanning. For example, in lexical analysis the characters in the assignment statement

position: = initial + rate * 60

would be grouped in to the following tokens

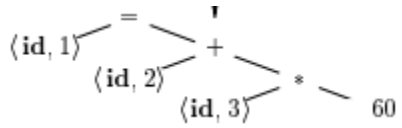
1. The identifier position
2. The assignment symbol: =
3. The identifier initial
4. The plus sign
5. The identifier rate
6. The multiplication sign

7. The number 60

The blanks are usually eliminated during lexical analysis

Syntax Analysis:

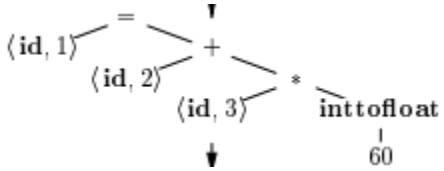
The second phase of the compiler is syntax analysis or parsing. The parser uses the first components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.



A typical representation is a syntax tree in which each interior node represents an operation and the children of the node represent the arguments of the operation.

Semantic Analysis:

The semantic analysis checks the source program for semantic errors and gathers type information for the subsequent code-generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operands of expressions and statements.



Intermediate Code Generator:

After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program. This intermediate representation can have a variety of forms. In three-address code, the source program might look like this,

temp1: = inttoreal (60)

temp2: = id3 * temp1

temp3: = id2 + temp2

id1: = temp3

Code Optimization:

The code optimization phase attempts to improve the intermediate code, so that faster running machine codes will result. Some optimizations are trivial. There is a great variation in the amount of code optimization different compilers perform. In those that do the most, called 'optimizing compilers', a significant fraction of the time of the compiler is spent on this phase.

temp1 := id3 * 60.0

id1 := id2 + temp1

Code Generation:

The final phase of the compiler is the generation of target code, consisting normally of relocatable machine code or assembly code. Memory locations are selected for each of the variables used by the program.

MOVF id3, R2

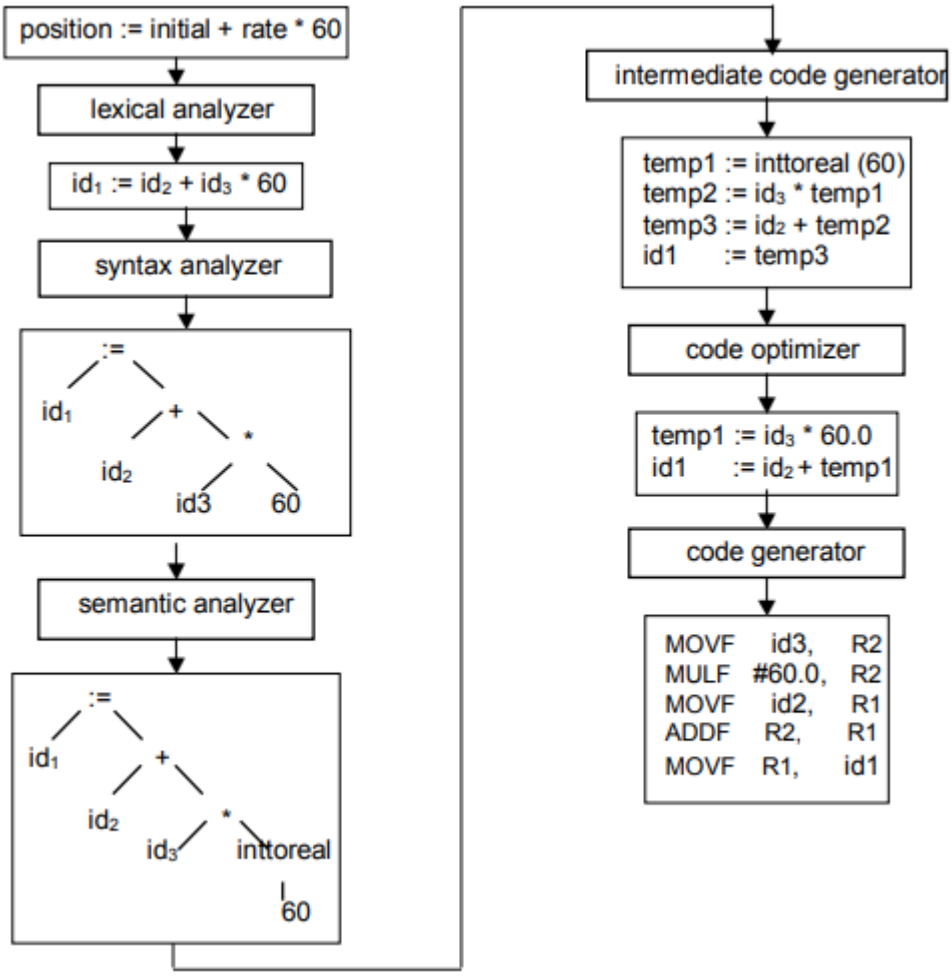
MULF #60.0, R2

MOVF id2, R1

ADDF R2, R1

MOVF R1, id1

The first and second operands of each instruction specify a source and destination, respectively. The F in each instruction tells us that instruction deal with floating point numbers.



Symbol-Table Management:

The symbol table is a data structure containing a record for each variable name, with fields for the attributes of the name. The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.