

## PRINCIPLES OF COMPILER DESIGN

### 1. Introduction to compilers: -

A compiler is a program that reads a program written in one language (source language (or) high level language) and translates it into an equivalent program in another language. (target language (or) low level language)



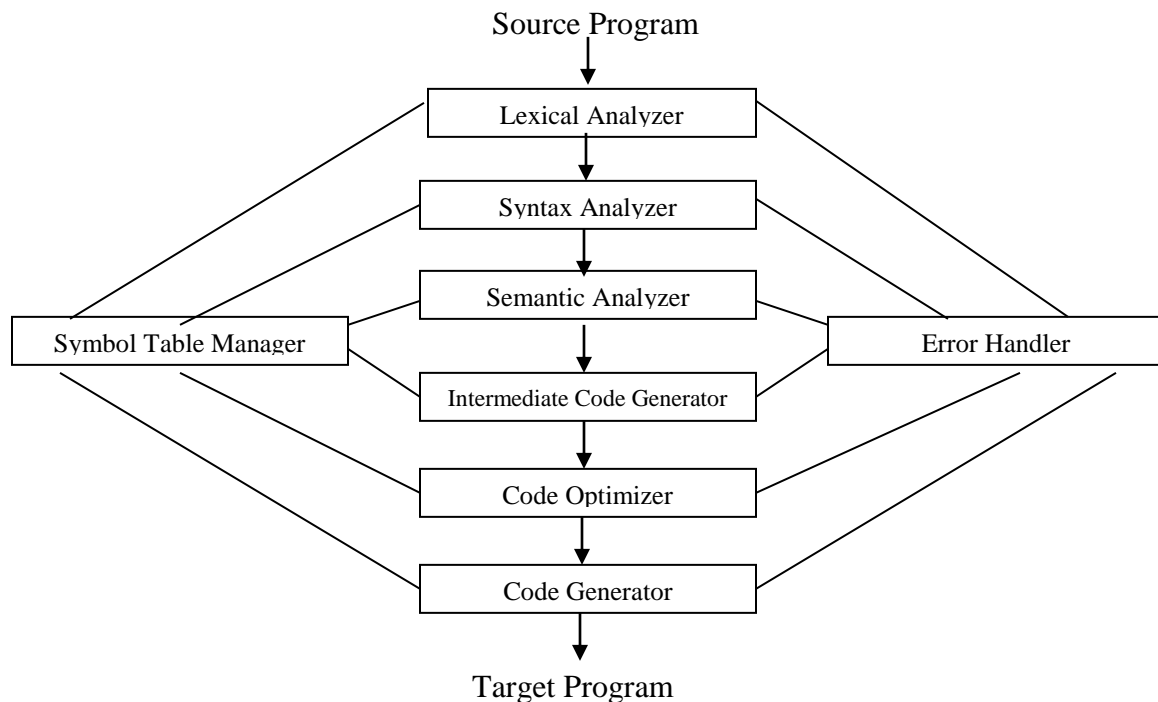
Compiler: - It converts the high level language into an equivalent low level language program.

Assembler:- It converts an assembly language(low level language) into machine code.(binary representation)

### PHASES OF COMPILER

There are two parts to compilation. They are

- (i) Analysis Phase
- (ii) Synthesis Phase



### Analysis Phase:-

The analysis phase breaks up the source program into constituent pieces. The analysis phase of a compiler performs,

- 1. Lexical analysis
- 2. Syntax Analysis
- 3. Semantic Analysis

## 1. Lexical Analysis (or) Linear Analysis (or) Scanning:-

The lexical analysis phase reads the characters in the program and groups them into tokens that are sequence of characters having a collective meaning.

Such as an Identifier, a Keyword, a Punctuation, character or a multi character operator like ++.

“ The character sequence forming a token is called **lexeme**”

For Eg. Pos = init + rate \* 60

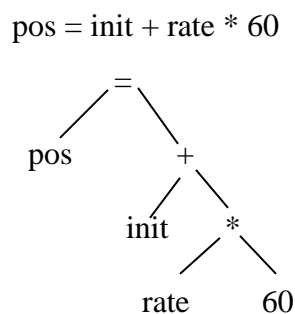
Lexeme	Token	Attribute value
rate	ID	Pointer to symbol table
+	ADD	
60	num	60
init	ID	Pointer to symbol table

## 2. Syntax Analysis (or) Hierarchical Analysis:-

Syntax analysis processes the string of descriptors (tokens), synthesized by the lexical analyzer, to determine the syntactic structure of an input statement. This process is known as **parsing**.

ie, Output of the parsing step is a representation of the syntactic structure of a statement.

Example:-



## 3. Semantic Analysis:-

The semantic analysis phase checks the source program for *semantic errors*.

Processing performed by the semantic analysis step can be classified into

- Processing of declarative statements
- Processing of executable statements

During semantic processing of declarative statements items of information are added to the *lexical tables*.

Example:- (symbol table or lexical table)

real a, b;

id	a	real	length	.....
id	b	real	length	.....

### Synthesis Phase:-

1. Intermediate code generation
2. Code optimization
3. Code Generator

#### 1. Intermediate code generation:-

After syntax and semantic analysis some compilers generate an explicit intermediate representation of the source program. This intermediate representation should have two important properties.

- a. It should be easy to produce
- b. It should be easy to translate into the target program.

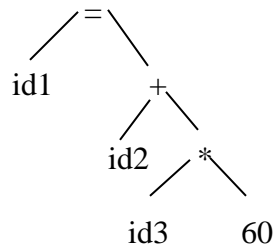
We consider the intermediate form called “Three Address Code”. It consists of sequence of instructions, each of which has atmost three operands.

Example:-

```
pos = init + rate * 60
pos = init + rate * int to real (60)
```

Might appear in three address code as,

```
temp1 = int to real (60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```



#### 2. Code Optimization:-

The code optimization phase attempts to improve the intermediate code, so that faster running machine code will result.

#### 3. Code Generation:-

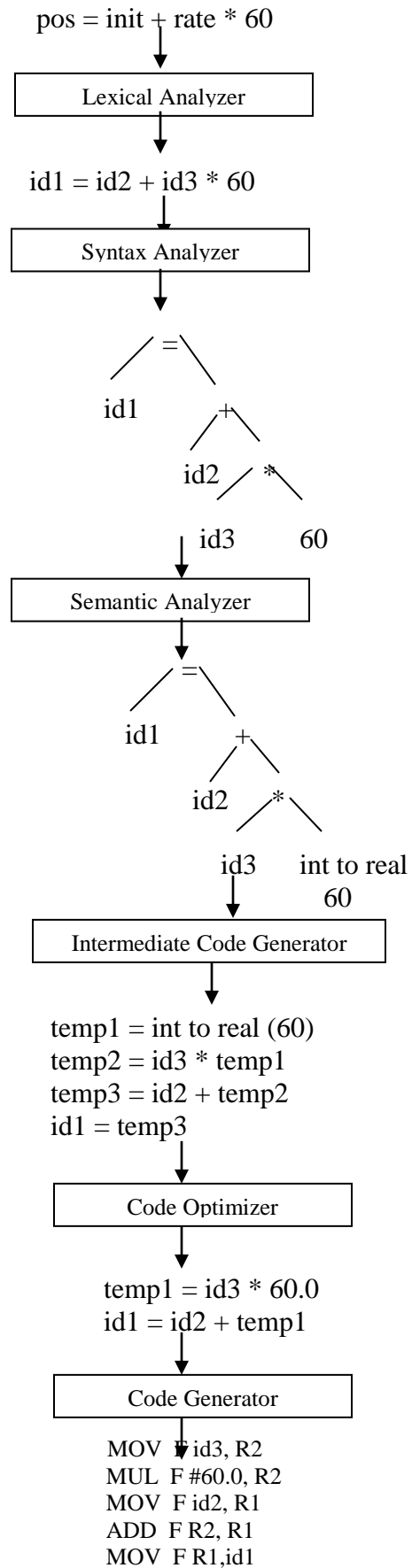
The final phase of the compiler is the generation of the target code or machine code or assembly code.

Memory locations are selected for each of the variables used by the program. Then intermediate instructions are translated into a sequence of machine instructions that perform the same task.

Example:-

```
MOV F id3, R2
MUL F #60.0, R2
MOV F id2, R1
ADD F R2, R1
MOV F R1, id1
```

## Translation of a statement



### Symbol table management:-

A symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

### Error Handler:-

*Each phase can encounter errors.*

- The lexical phase can detect errors where the characters remaining in the input do not form any token of the language.
- The syntax analysis phase can detect errors where the token stream violates the structure rules of the language.
- During semantic analysis, the compiler tries to construct a right syntactic structure, but no meaning to the operation involved.
- The intermediate code generator may detect an operator whose operands have incompatible types.
- The code optimizer, doing control flow analysis may detect that certain statements can never be reached.
- The code generator may find a compiler created constant that is too large to fit in a word of the target machines.

## References

1. J. Tremblay, P.G. Sorenson, "The Theory and Practice of Compiler Writing ", McGRAW-HILL, 1985.
2. W.M. Waite, L.R. Carter, "An Introduction to Compiler Construction", Harper Collins, New York, 1993
3. A.W. Appel, "Modern Compiler Implementation in, Cambridge University Press, 1998
4. Internet Papers.
5. Aho, R. Sethi, J.D. Ullman, "Compilers- Principles, Techniques and Tools" Addison-Wesley, 2007.