Unit - III

Chapter 1: Turing Machine (TM)

↳ Definition
↳ Structural representation
↳ construction of TM *

Chapter-II: compiler.

→ General Language processing System
→ Definition of compiler
→ Phases of compiler. *
→ Lexical Analysis
→ Input Buffering | *.

*

RG → RL → FA
CFG → CFL → PDA.

Type 0 → Unrestricted Grammar → Recursively enumerable Language → Turing Machine

Turing Machine (TM):

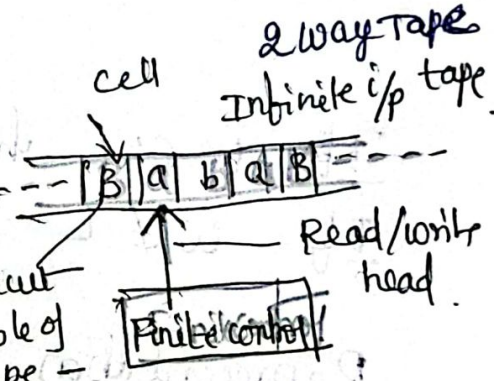
A turing machine is an abstract machine which accepts the language (Recursively enumerable language) generated by type 0 grammar (unrestricted grammar).
It was invented in 1936 by Alan Turing.

TM → (unrestricted grammar. → Recursively enumerable language (REL))
(UG)

* UG generates / formed REL, which is accepted / recognized by Turing machine
Turing machine have infinite tape.

Model of Turing Machine: Turing Machine model is composed of 3 components

① Infinite input tape
② Read / write head.
③ Finite control



① **Input tape**: Tape devided into no. of cells. Each cell hold symbol. then remaining cells hold blank symbol (B)

② **Read / write head**: capable of read & write to input tape and also can move left and right side of tape.

③ **Finite control (FC)** FC takes care of the actions to be peformed by turing machine. It keep tracks of different states of TM. The FC takes decision how to move & where to move Left or Right

A TM is a 7 tuple system.

TM = $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F$

Q : Finite set of states.

Σ = Input alphabet.

Γ = alphbets in input tape.

q_0 = initial state.

B = Blank space

F = Final state (s).

δ : Transition function.

DTM $\rightarrow \delta : Q \times \Gamma \rightarrow Q \times \Gamma \{ , R \}$

NTM $\rightarrow \delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$

If $w \in L$ halts at final state.

If $w \notin L$ halts at non final state.

Representation of TM :

1) Transition table
2) Transition diagram.
3) Instantaneous description

Transition Table :

Transition table is a tabular representation of the transition function.

- → columns represent the symbols in input tape (Γ)
- → Rows represent the state.
- → Entries represent the corresponding values
- → Start state denoted by arrow mark
- → The Final / accept state represent by star ($*$)

Transition table represents the (δ) transition function

Transition Table :

δ	Tape symbols (Γ)					
	a	b	X	Y	B	
q_0	(q_1, X, R)			(q_3, Y, R)		
q_1	(q_1, a, R)	(q_2, Y, L)		(q_1, Y, R)		
q_2	(q_2, a, L)		(q_0, X, R)	(q_2, Y, L)		
q_3				(q_3, Y, R)	(q_4, B, R)	
q_4						

Here a & b are input symbols (Σ), excluding the symbol B

- → B indicates blank symbol, specifies string end & infinite 'B'
- → Undefined entries specifies that there is no transition &
- → When there is transition to dead state, the machine halts, i/p string is rejected by Machine

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times (R, L)$

Where $Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma = \{a, b\}$

Γ = Symbol in i/p tape $= \{a, b, X, Y, B\}$

q_0 = initial state $= q_0$

$F = q_4$ = Final state

R = Blank character

Transition diagram: Transition diagram consist of set of nodes/ states.

→ There is a node for each state in Q, each is represented by circle

→ edge labeled from one state to another state

→ start state with arrow represents

→ Final / accepting state by a double circle

The transition from one state to another state represents in the form of

(X, Y, D)

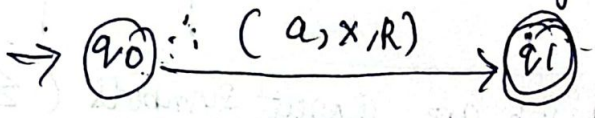
Where X → old tape symbol.

Y → new or same tape symbol

D → direction of tape for state (L or R)

Ex: $\delta(q_0, a) = (q_1, X, R)$ — moving to Right

old tape symbol, new tape symbol



Instantaneous Description: An ID of TM in a string in $\alpha q \beta$, where q is current state, $\alpha\beta$ is the string made from tape symbols denoted by Γ, i.e $\alpha + \beta = \Gamma^*$. The read write head points to the first character of the substring β. The initial q_0 is denoted by $q\alpha\beta$, where q is start state & read write head points to the 1st symbol of α from left.

ID of TM remembers the following

1. The content of all the cells in input tape, starting from rightmost cell up to least the last cell, non blank symbol containing cells up to cells being scanned
2. Cells currently being scanned by read-write head
3. The state of the machine.

1Q) Construct the TM for $L = \{a^n b^n / n \geq 1\}$

Soln:

Given $L = \{a^n b^n / n \geq 1\}$

if $n = 1 = ab$

$n = 2 = aabb$

$n = 3 = aaabbb$

$L = \{ab, aabb, aaabbb, \cdots\}$

TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

Let consider $W = aabb$.

then i/p tape ... | B | a | a | b | b | B | B | ...

Initially Read/write pointed to the first alphabet in string.

... | B | a | a | b | b | B | B | ...

Here no. of a's = no. of b's.

Step 1: Read a and write x in that cell & move right until reach b. Let the state $q_0 = q_0$.

Note: Every a's as X and Every b's as Y. If TM contains an equal no. of x and y then it reaches the final state.

Step1.

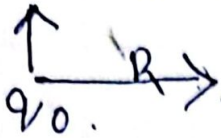
Step2: Move left to find the rightmost X and move one cell right to leftmost a & repeat the same cycle.

Step3: While searching for b if blank B is encounted, then M halts without accepting.

Step4: After changing b to y if M finds no more a's, then M checks no more b's remaining, then accepting the string otherwise not.

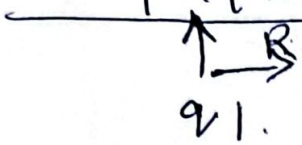
1) Read a & change to X
2) Move right to first b
3) If none reject
4) Change b to y
5) Move left to leftmost X
6) Repeat the same steps. Until no more a's and no more b's.

(1) B | a | a | b | b | B | - - -



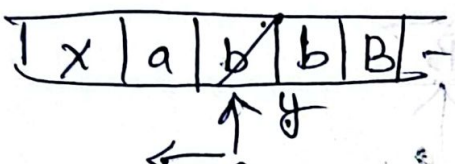
$\delta(q_0, a) = (q_1, x, R)$

(2) x | a | b | b | B - - -



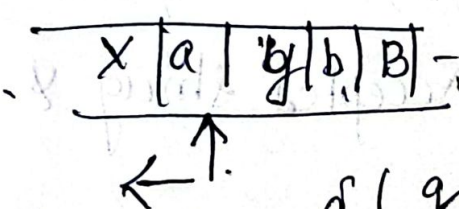
$\delta(q_1, a) = (q_1, a, R)$

(3) | x | a | b | b | B - - -



$\delta(q_1, b) = (q_2, y, L)$

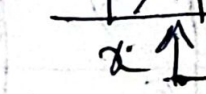
(4) x | a | y | b | B - - -



Now at q_0

$\delta(q_2, a) = (q_2, a, L)$

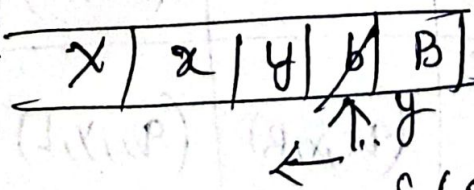
(5) x | a | y | b | B - - -



$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_1, y) = (q_1, y, R)$

(6) x | x | y | b | B - - -

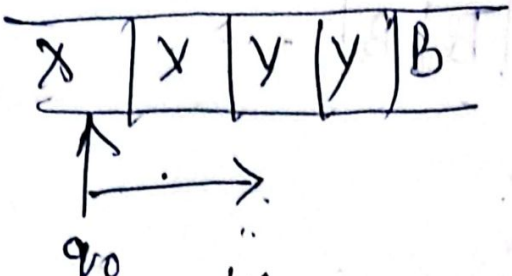


$\delta(q_1, b) = (q_2, y, L)$

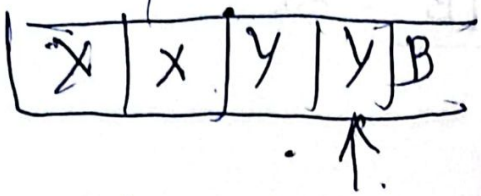
$\delta(q_2, y) = (q_2, y, L)$

move left

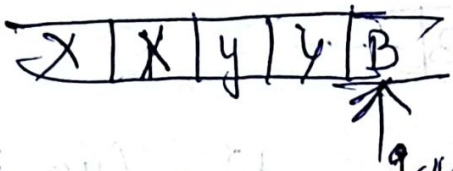
$\delta(q_2, x) = (q_0, x, R)$



$$\delta(q_0, y) = (q_3, y, R)$$



$$\delta(q_3, y) = (q_4, y, R)$$



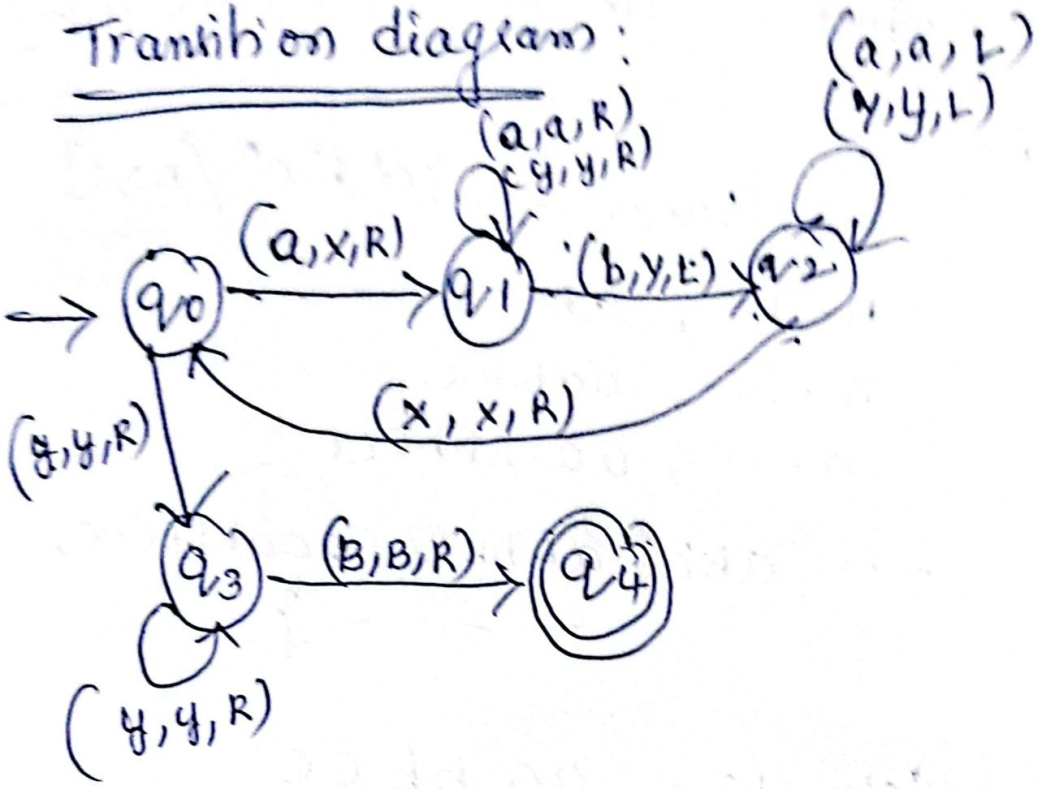
$$\delta(q_4, B) = (q_4, B, R)$$

Final state Reached B, Accepted string x

Transition Table :

	a	b	x	y	B
→ q_0	(q_1, x, R)	–	–	(q_3, y, R)	
q_1	(q_1, a, R)	(q_2, y, L)	–	(q_1, y, R)	
q_2	(q_2, a, L)	–	(q_0, x, R)	(q_2, y, L)	
q_3	–	–	–	(q_3, y, R)	(q_4, B, R)
* q_4					

Transition diagram:



Turing Machine = $M = (Q, \Sigma, \delta, \Gamma, q_0, B, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

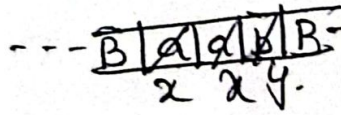
$\Sigma = \{a, b, Y\}$

$\Gamma = \{a, b, X, Y, B\}$

$q_0 = \{q_0\}$

$F = \{q_4\}$

Ex: Consider string aaab accepted or not.



Here After reading the 2nd a, could not find b. and no. of a's are not equal to no. of b

∴ The string aaab is not accepted by Turing Machine.

Transition Functions (δ) for $a^n b^n$

$$\delta(q_0, a) = (q_1, X, R)$$
$$\delta(q_0, Y) = (q_3, Y, R)$$
$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, Y) = (q_1, Y, R)$$
$$\delta(q_1, b) = (q_2, Y, L)$$
$$\delta(q_2, a) = (q_2, a, L)$$
$$\delta(q_2, Y) = (q_2, Y, L)$$
$$\delta(q_2, X) = (q_0, X, R)$$
$$\delta(q_3, Y) = (q_3, Y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

ID of the string aaabbb :

BaaabbbB → BXaabbbB
BXaaabbbB → BXaaYbbB
BXaaYbbB → BXXaYbB
BXXaYbbB → BXXaYYbB
BXXaYYbB → BXXXYYbB
BXXXYYbB → BXXXYYYB

Halt in accepting state.

Note :

Showing one by one how a's replacing with X and b's replacing with Y until it accepts string

2Q) Construct the turing machine for :
$L = \{a^n b^n c^n / n \geq 1\}$

Sol:- Given $L = \{a^n b^n c^n / n \geq 1\}$

$n = 1 \rightarrow abc$
$n = 2 \rightarrow aabbcc$
$n = 3 \rightarrow aaabbbccc$

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

$TM = Q = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Let $w = aabbcc$.

then input tape consist B a a b b c c B

Read/write head.

Logic for construction of TM :

1) Read a & write x then moven to right till reach rightmost b i.e 1st b.

2) Read b & replace/write y, then move to right till reach rightmost / 1st c.

3) Read c & replace by z, move left Unit reach x.

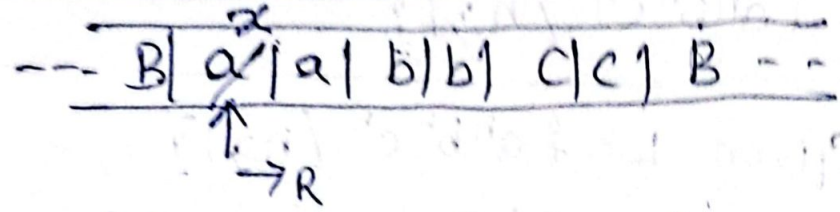
Then repeat the steps 1, 2 & 3.

If reaches B, then read it

No. of a's = No. of b's = No. of c's

String reaches final state & string accepted

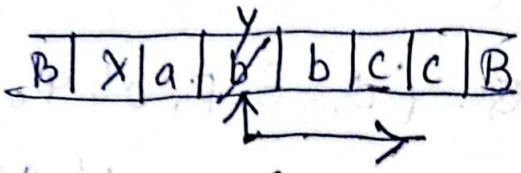
Transition diagram:



$\delta(q_0, a) = (q_1, x, R)$

Read 2nd a $\delta(q_1, a) = (q_1, a, R)$

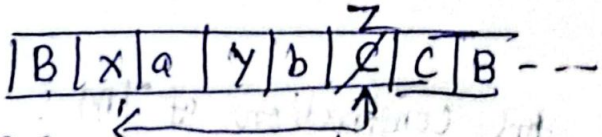
move right



$\delta(q_1, b) = (q_2, y, R)$ move

Read 2nd b $\delta(q_2, b) = (q_2, b, R)$

Read 1st c



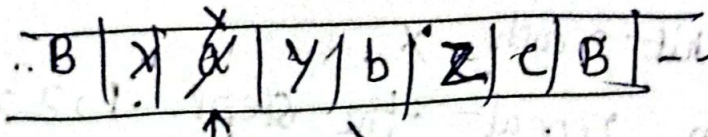
$\delta(q_2, c) = (q_3, z, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, y) = (q_3, y, L)$

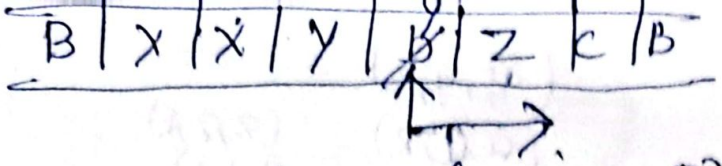
$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, x) = (q_0, x, R)$



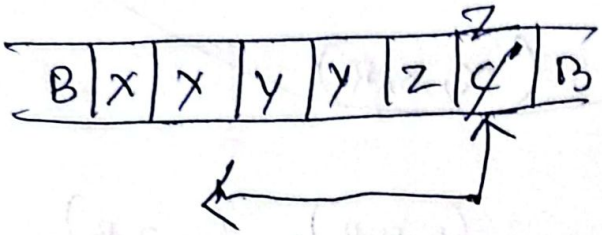
$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_1, y) = (q_1, y, R)$



$\delta(q_1, B) = (q_2, Y, R)$.

$\delta(q_2, Z) = (q_2, Z, R)$.

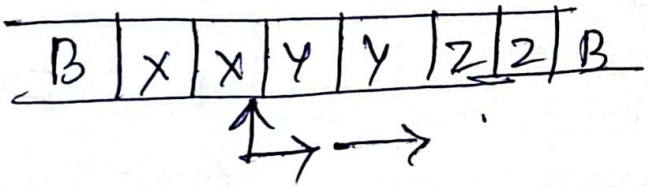


$\delta(q_2, C) = (q_3, Z, L)$.

$\delta(q_3, Z) = (q_3, Z, L)$.

$\delta(q_3, Y) = (q_3, Y, L)$

$\delta(q_3, X) = (q_0, X, R)$



$\delta(q_0, Y) = (q_4, Y, R)$

$\delta(q_4, Y) = (q_4, Y, R)$

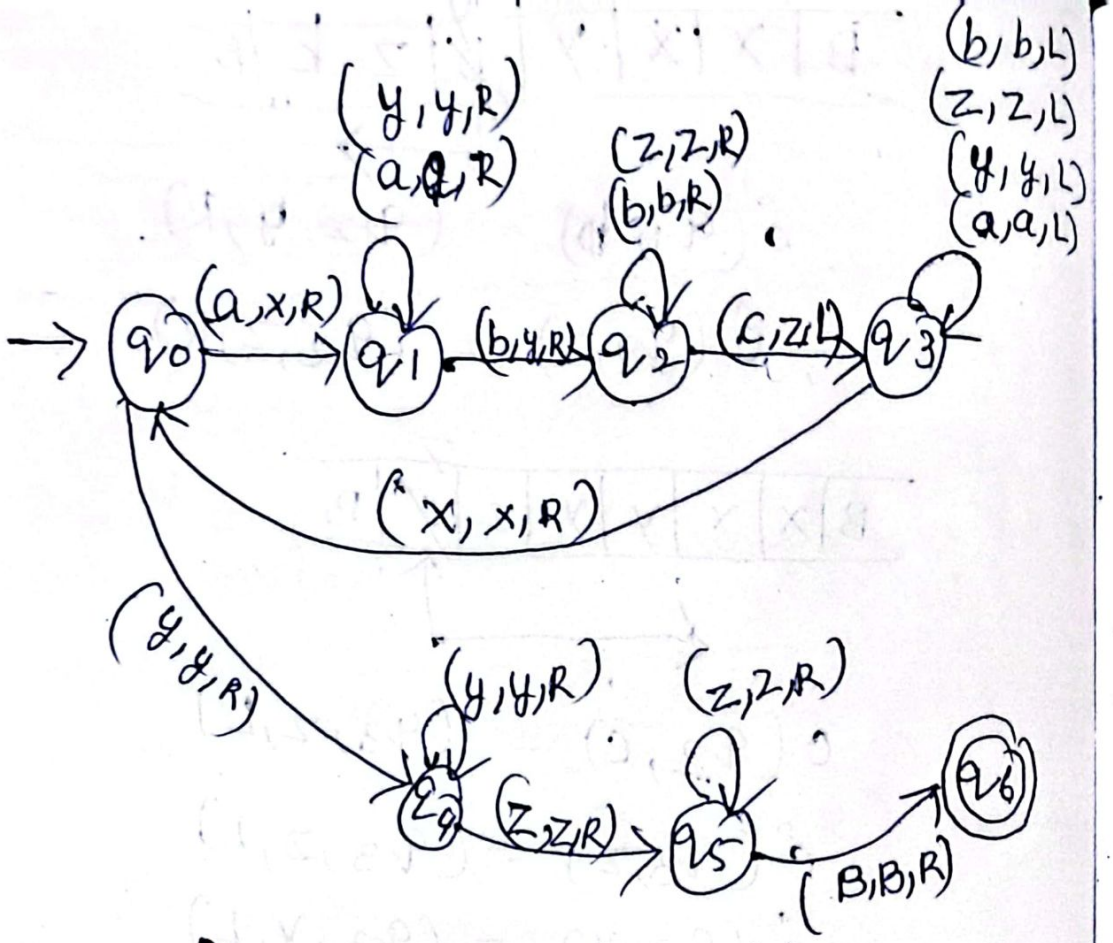
$\delta(q_4, Z) = (q_5, Z, R)$

$\delta(q_5, Z) = (q_5, Z, R)$

$\delta(q_5, B) = (q_6, B, R)$

Reached 'B' symbol

String is accepted.



Reaches q_6 & accepted string.

Transition Table:

δ	a	b	c	x	y	z	B
→q_0	(q_1,z,R)				(q_4,y,R)		
q_1	(q_1,a,R)	(q_2,y,R)			(q_1,y,R)		
q_2		(q_2,B,R)	(q_3,z,L)			(q_2,z,R)	
q_3	(q_3,a,L)	(q_3,b,L)		(q_0,x,R)	(q_3,y,L)	(q_3,z,L)	
q_4					(q_4,y,R)	(q_5,z,R)	
q_5						(q_5,z,R)	(q_6,B,R)
*q_6							

$TM = \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma = \{a, b, c\}$

Γ = symbols in input tape

$\Gamma = \{a, b, c, x, y, z, B\}$.

$q_0 = q_0$ = Initial state.

B = Blank symbol.

F = Final state = q_6.

3Q): construct the turing machine for $L = \{ w w^R / w \in (a,b)^+ \}$. or Even Palindrome

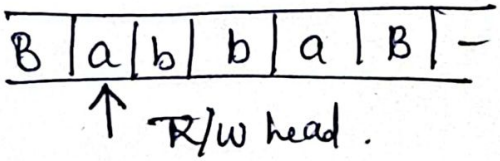
Soln:

Given $L = \{ w w^R / w \in (a,b)^+ \}$

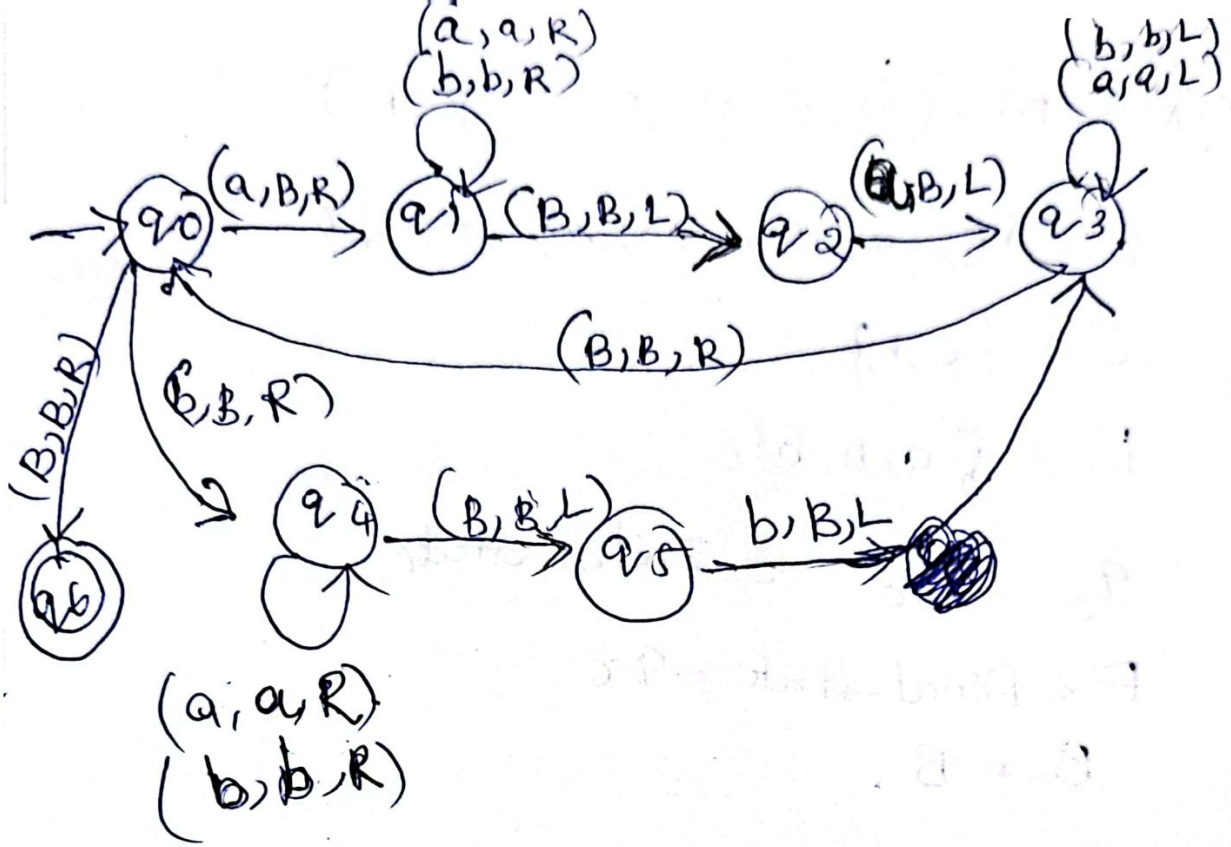
$L = \{ aa, bb, abba, baab, abbbba, \ldots \}$

$w = ab$, $w^R = ba$.

$ww^R = abba$.



1. Read a and write B, in place of 'a' and move right till reach B.
2. move left Read 'a' and write 'B' & move left untill reach B
3. move right. read 'b' & write B & move right and reach B.
4. move left and read B b & write B move right till B.
5. No more symbols in the tape all 'B' symbols moving towards right ∴ string accepted.



δ	a	b	B
$\rightarrow q_0$	(q_1, B, R)	(q_4, B, R)	(q_6, B, R)
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, B, L)
q_2	(q_3, B, L)	—	—
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, B, R)
q_4	(q_4, a, R)	(q_4, b, R)	(q_5, B, L)
q_5	—	q_3, B, L	—
$*q_6$	—	—	—

TM = M = $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$

$q_0 = q_0$ = Initial state,

F = final state = q_6

B = B.

Q) construct the turing machine for
$L = \{ wcw^R / w \in (a+b)^* \}$
& odd palindrome.

Soln:

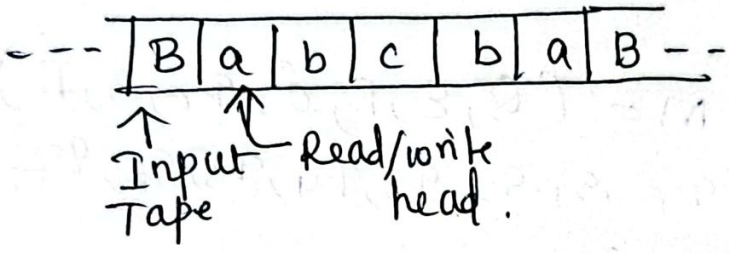
$TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

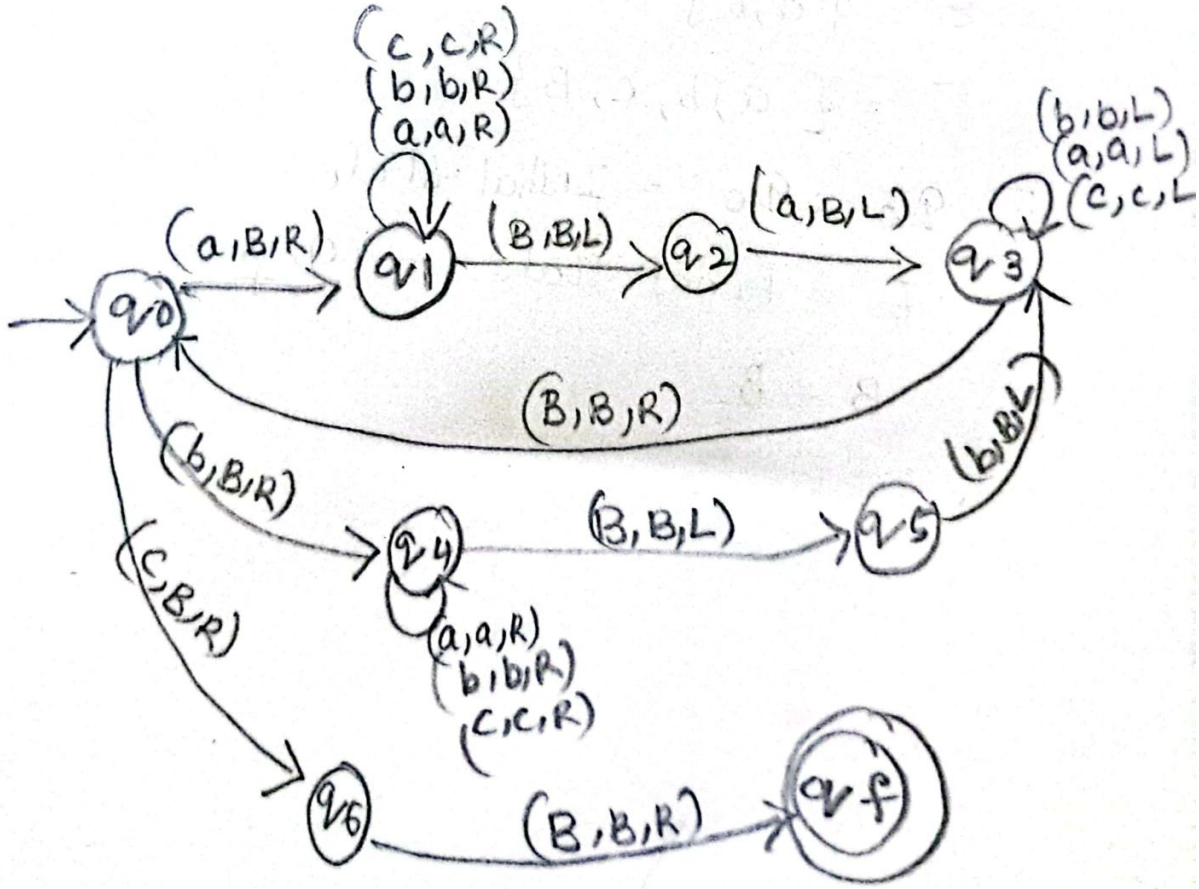
Given $\Sigma = \{a, b\}$.

Let $q_0 = q_0$, $B = B$.

$w = ab$, $w^R = ba$.

$wcw^R = abcba$.





δ	a	b	c	B
$\rightarrow q_0$	(q_1, B, R)	(q_4, B, R)	(q_6, B, R)	–
q_1	(q_1, a, R)	(q_1, b, R)	(q_1, c, R)	$(q_2, B, L$
q_2	(q_3, B, L)	–	–	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_3, c, L)	$(q_0, B, R$
q_4	(q_4, a, R)	(q_4, b, R)	(q_4, c, R)	$(q_5, B, L$
q_5	–	(q_3, B, L)	–	–
q_6	–		–	(q_f, B, R)
$*q_f$	–		–	

TM = $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, c, B\}$

$q_0 = q_0$ = Initial state

F = Final state = q_f

$B = B$

UNIT 3(chapter-2)

Syllabus:

Compiler: Definition of Compiler, Language processing systems ,Phases of Compiler, Lexical Analysis, Input Buffering.

Compilers

Compiler is a language translator or is a program which translates a program written in one language (the source language) to an equivalent program in other language (the target language).

The source language usually is a high-level language like Java, C, Fortran etc. whereas the target language is machine code that a computer's processor understands.

The source language is optimized for humans. It is more user-friendly, to some extent platform-independent. They are easier to read, write and maintain. Hence, it is easy to avoid errors. Ultimately, programs written in a high-level language must be translated into machine language by a compiler. The target machine language is efficient for hardware but lacks readability.

- Translates from one representation of the program to another.
- Typically, from high level source code to low level machine code or object code.
- Source code is normally optimized for human readability.
- Machine code is optimized for hardware.
- Redundancy is reduced.
- Information about the intent is lost.

Goals of translation

Good performance for generated code: The metric for the quality of the generated code is the ratio between the size of handwritten code and compiled machine code for same program. A better compiler is one which generates smaller code. For optimizing compilers this ratio will be lesser.

Good compile time performance: A handwritten machine code is more efficient than a compiled code in terms of the performance it produces.

Correctness: A compiler's most important goal is correctness - all valid programs must compile correctly.

Pictorial Representation

- Compiler is part of program development environment
- The other typical components of this environment are editor, assembler, linker, loader, debugger, profiler etc.
- The compiler (and all other tools) must support each other for easy program

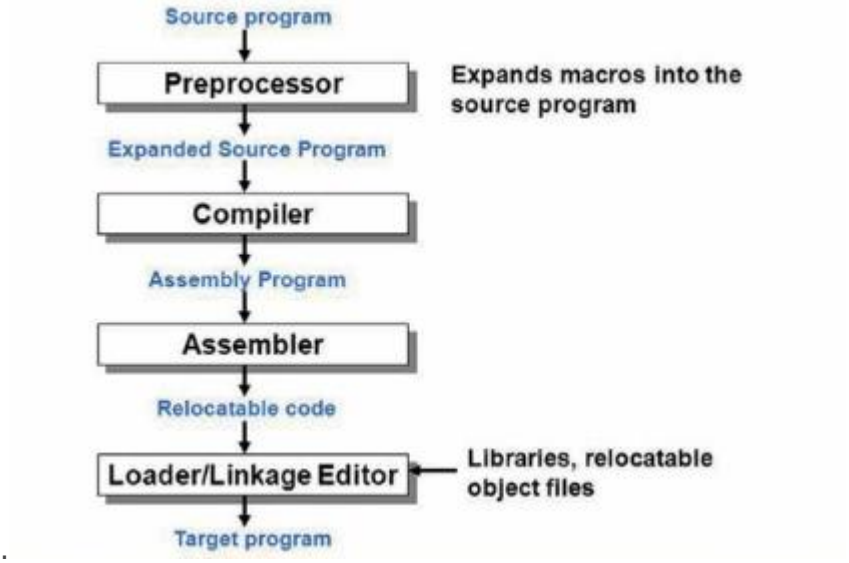
development

All development systems are essentially a combination of many tools. For compiler, the other tools are debugger, assembler, linker, loader, profiler, editor etc. If these tools have support for each other than the program development becomes a lot easier.

Language processing systems:

Language processing systems in compiler design play a crucial role in transforming high-level programming languages into machine-readable code

In a language processing system, the source code is first preprocessed. The modified source program is processed by the compiler to form the target assembly program which is then translated by the assembler to create relocatable object codes that are processed by linker and loader to create the target program. It is based on the input the translator takes and the output it produces, and a language translator can be defined as any of the following.



Preprocessor:

A preprocessor produce input to compilers. They may perform the following functions.

1. Macro processing: A preprocessor may allow a user to define macros that are short hands for longer constructs. Eg: #define PI 3.14 Whenever the PI is encountered in a program, it is replaced by the value 3.

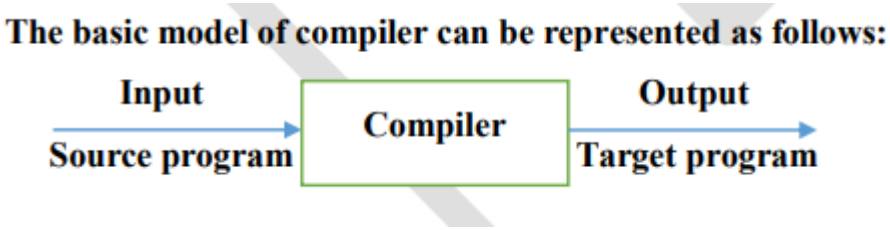
2. File inclusion: A preprocessor may include header files into the program text. Eg: #include By this statement, the header file stdio.h can be included and user can make use of the functions in this header file. This task of preprocessor is called file inclusion.

3. Rational preprocessor: these preprocessors augment older languages with more modern flow-of-control and data structuring facilities.

4. Language Extensions: These preprocessor attempts to add capabilities to the language by certain amounts to build-in macro.

Compiler :

Compiler is a program that can read a program in one language the source language and translate it into an equivalent program in another language the target language;
If some errors are encountered during the process of translation, then compiler displays them as error messages.



The compiler takes the source program in high level language such as C, PASCAL, FORTRAN and converts into low level language or machine level language such as assembly language. Assembler .
Programmers found difficult to write or read programs in machine language. They begin to use a mnemonic (symbols) for each machine instruction, which they would subsequently translate into machine language. Such a mnemonic machine language is now called an assembly language.

Programs known as assembler were written to automate the translation of assembly language into machine language.
The input to an assembler program is called source program, the output is a machine language translation (object program).

Loader/Linker – editor

Loader is a program which performs two functions, loading and link editing.

Loading is a process in which the relocatable machine code is read and the relocatable addresses are altered.

Then that code with altered instructions and data is placed in the memory at proper location.

The job of link editor is to make a single program from several files of relocatable machine code.

If code in one file refers the location in another file, then such a reference is called external reference. The link editor resolves such external references also.

PHASES OF A COMPILER:

The compilation process is a sequence of various phases. Each phase takes input from its previous stage, has its own representation of source program, and feeds its output to the next phase of the compiler. Let us understand the phases of a compiler.

Compiler operates in various phases , each phase transforms the source program from one representation to another.every phase takes input from its previous phase and feeds its output to the next phase of the compiler.

It is desirable to have relatively few phases, since it takes time to read and write immediate files. Following diagram depicts the phases of a compiler through which t goes during the compilation. There are 6 phases in a compiler.Each of this phase help in converting the high-level langue the machine code. The phases of a compiler are:

1. Lexical Analyzer (Scanner),
2. Syntax Analyzer (Parser),
3. Semantic Analyzer,
4. Intermediate Code Generator(ICG),
5. Code Optimizer(CO) , and
6. Code Generator(CG).

In addition to these, it also has **Symbol table management**, and **Error handler** phases.
The Phases of compiler divided in to two parts, first three phases we are called as **Analysis part** remaining three called as **Synthesis part.**

The synthesis part constructs the desired target program from the intermediate representation and the information in the symbol table.

The analysis part is often called the front end of the compiler; the synthesis part is the back end.
If we examine the compilation process in more detail, we see that it operates as a sequence of phases, each of which transforms one representation of the source program to another.

A typical decomposition of a compiler into phases is shown in Fig. In practice, several phases may be grouped together, and the intermediate representations between the grouped phases need not be constructed explicitly .

PHASE, PASSES OF A COMPILER:

In some application we can have a compiler that is organized into what is called passes. Where a pass is a collection of phases that convert the input from one representation to a completely different representation. Each pass makes a complete scan of the input and produces its output to be processed by the subsequent pass.For example a two pass Assembler.

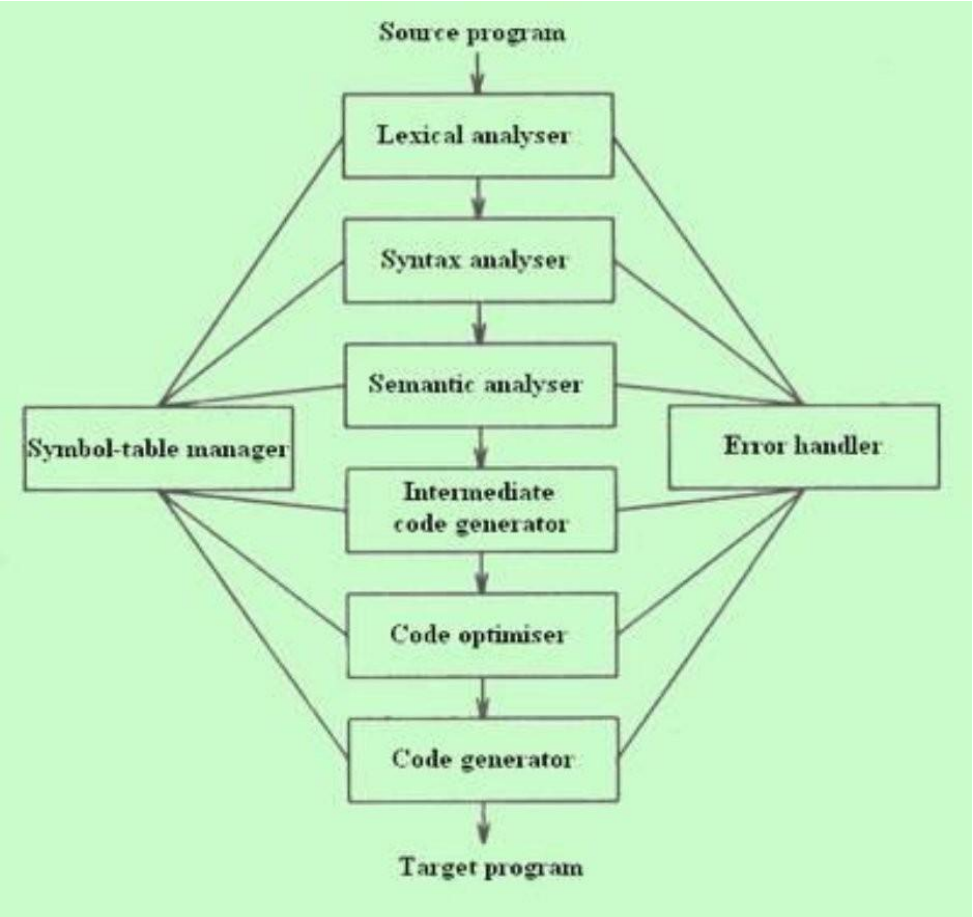


Figure : Phases of a Compiler

LEXICAL ANALYZER (SCANNER): The Scanner is the first phase that works as interfacebetween the compiler and the Source language program and performs the following functions:

- Reads the characters in the Source program and groups them into a stream of tokens in which each token specifies a logically cohesive sequence of characters, such as an identifier, a Keyword , a punctuation mark, a multi character operator like := .
- The character sequence forming a token is called a **lexeme** of the token.
- The Scanner generates a token-id, and also enters that identifiers name in the Symboltable if it doesn‘t exist.
- If the token is not valid i.e., does not fall into any of the identifiable groups, then thelexical analyser reports an error.
- Lexical analysis involves recognizing the tokens in the source program and reporting errors, if any.
- Also removes the Comments, and unnecessary spaces.

- Token: A token is a syntactic category. Sentences consist of a string of tokens. Forexample constants, identifier, special symbols, keyword, operators etc are tokens.

- Lexeme: Sequence of characters in a token is a lexeme. For example, 100.01,counter, const, "How are you?" etc are lexemes.

The format of the **token representation** < **Token name, Attribute value**>

The token type tells the category of token and token value gives us the information regarding token. The token value is also called token attribute.

- Lexical analysis created the symbol table. The token value can be a pointer to symbol table in case of **identifier and constant.**
- The lexical analyzer reads the input program and generates table for tokens.

Consider the example a= b+c* 60 or position=initial+rate *60

Lexeme	**Token**	**Token representation**
a	Identifier	<id,1>
=	Assignment Operator	<=>
b	Identifier	<id,2>
+	Additional Operator	<+>
c	Identifier	<id,3>
*	Multiplication Operator	<*>
60	Constant	60

Then the input of a=b+c* 60 becomes as id1=id2+id3*60 and the information about the token stored in the symbol table(only identifiers excluding the all other data)

The symbol table is mainly known as the data structure of the compiler. It helps in storing the identifiers with their name and types. It makes it very easy to operate the searching and fetching process.

The symbol table connects or interacts with all phases of the compiler and error handler for updates. It is also accountable for scope management.

Example: **a= b+c* 60 or position=initial+rate *60**

Symbol name	Type	scope	address
a / position	float	global	
b / initial	float	global	
c / rate	float	global	

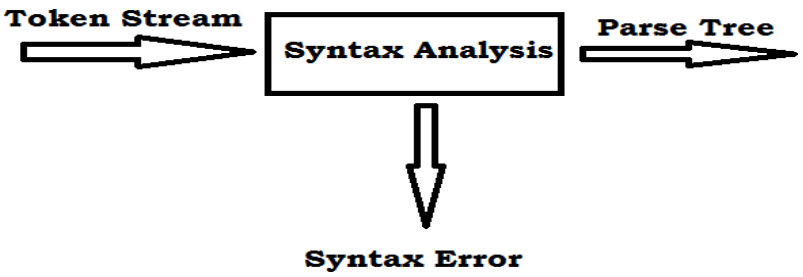
SYNTAX ANALYZER (PARSER):

A syntax analyzer or parser takes the input from a lexical analyzer in the form of token streams. The parser analyzes the source code (token stream) against the production rules to detect any errors in the code. The output of this phase is a parse tree .

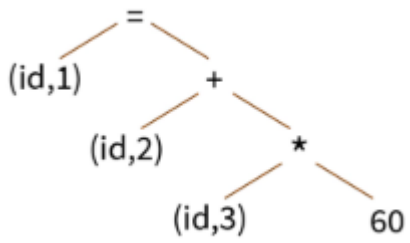
The parser (syntax analyzer) receives the source code in the form of tokens from the lexical analyzer and performs syntax analysis, which create a tree-like intermediate representation that depicts the grammatical structure of the token stream.
The Parser interacts with the Scanner, and its subsequentphase Semantic Analyzer and performs the following functions:

A typical representation is a abstract syntax tree in

- which each interior node represents an operation
- the children of the node represent the arguments of the operation



Parse tree or Syntax tree:



SEMANTICANALYZER: This phase receives the syntax tree as input, and checks the semantically correctness of the program. Though the tokens are valid and syntactically correct, itmay happen that they are not correct semantically.

Therefore the semantic analyzer checks the semantics (meaning) of the statements formed.

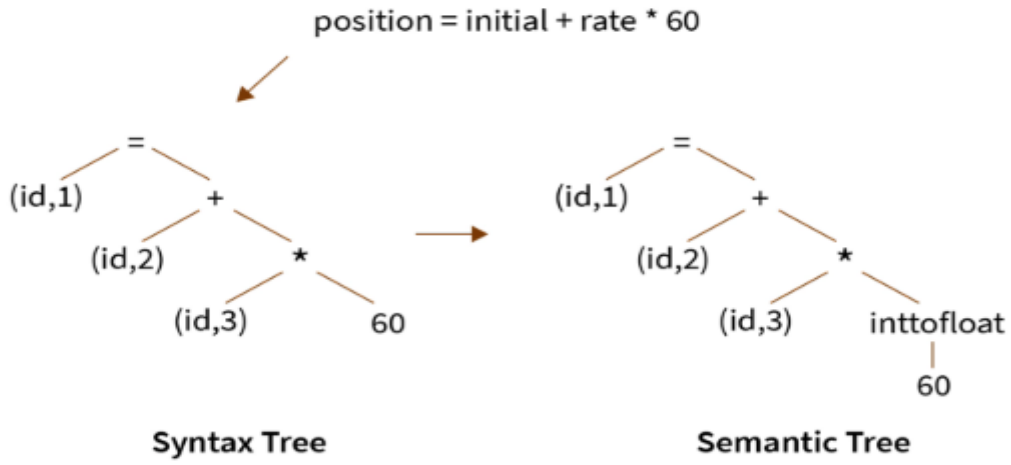
Semantic Analyzer will check for

- Type mismatches,
- incompatible operands,
- a function called with improper arguments,
- an undeclared variable, etc.

Here in the parse tree id2 ,id3 are float and result stored in float.

Type mismatching with 60 , which is integer, so converted in to real/float

The Syntactically and Semantically correct structures are produced here the **updated parse tree.**



INTERMEDIATE CODE GENERATOR(ICG): This phase takes the syntactically and semantically correct structure as input, and produces its equivalent intermediate notation of the source program. The Intermediate Code should have two important properties specified below:

- It should be easy to produce, and Easy to translate into the target program.
  Example intermediate code forms are:
- Three address codes,
- Polish notations, etc.

The interior nodes corresponding to inttoreal, +, *, are assigned temporary variables.
The temporary variable corresponding to the interior node will be one of the 3 addresses that comprises of the three address code.
This is typically the left hand side variable of the 3-address code.
The right hand side expression is derived using the left and right children of the interior node and the operator to join them is based on the interior node. If, there is only one child for the interior node, then the right hand side of the 3-address code will have only one operand.

In this example, “inttoreal” is an interior node which is assigned temporary variable “t1”.
The operator is “inttoreal” and the only right hand side operand is “60”. So, the corresponding 3 – address code is given in statement

t1 = inttoreal(60)

Consider one more interior node, “*”. It is assigned temporary variable “t2” and its left and right children are “id3” and temporary variable “t1”. Hence, the corresponding 3-address code is given in statement

t2= id3 * t1

for the other two interior nodes, the corresponding three-address codes are given in statement

t3 = id2 + t2
id1 = t3

So output of Intermediate code generator is as follows:

t1 = inttoreal(60)
t2= id3 * t1
t3 = id2 + t2
id1 = t3

CODE OPTIMIZER: This phase is optional in some Compilers, but so useful and beneficial in terms of saving development time, effort, and cost. This phase performs the following specific functions:

→Attempts to improve the IC so as to have a faster machine code. Typical functions include –Loop Optimization, Removal of redundant computations, Strength reduction, Frequency reductions etc.

- Sometimes the data structures used in representing the intermediate forms mayalso be changed.

The modified code is given as:

t1 = id3 * 60.0
id1 = id2 + t1

CODE GENERATOR:

This is the final phase of the compiler and generates the target code, normally consisting of the relocatable machine code or Assembly code or absolute machine code.

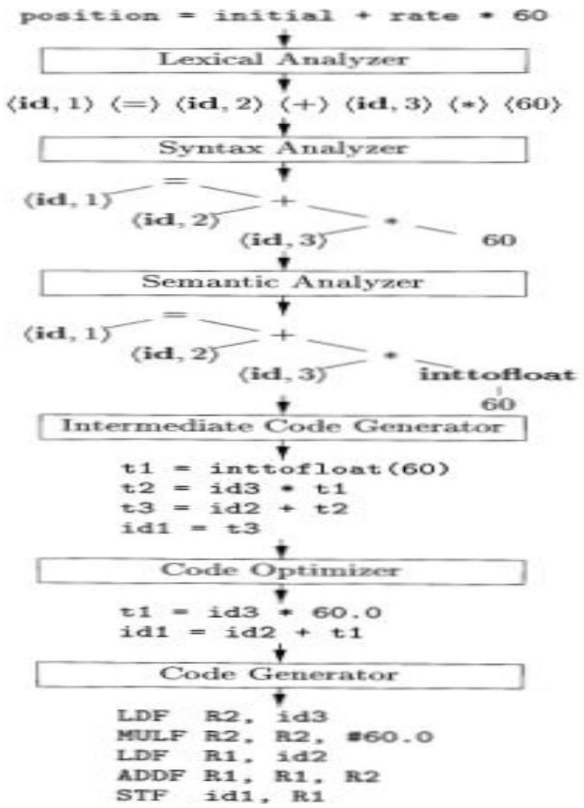
- Memory locations are selected for each variable used, and assignment of variables toregisters is done.
- Intermediate instructions are translated into a sequence of machine instructions as follow:

```
LDF  R2 ,  id3
MULF   R2,  R2,   #60.0
LDF  R1,   id2
ADDF   R1,  R1,  R2
STF    id1,    R1
```

The Compiler also performs the **Symbol table management** and **Error handling** throughout thecompilation process. Symbol table is nothing but a data structure that stores different source language constructs, and tokens generated during the compilation. These two interact with all phases of the Compiler.

For example the source program is an assignment statement; the following figure shows how thephases of compiler will process the program.

Fig: Translation of assignment statement for **Position=initial +rate*60**

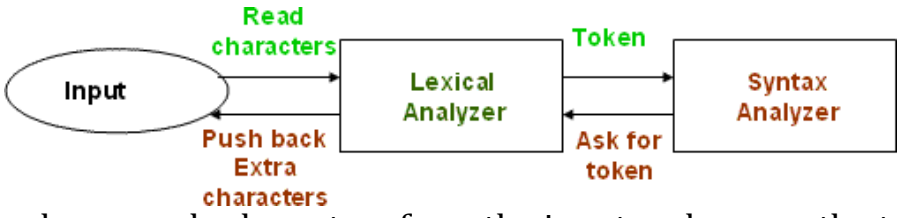


The input source program is **Position=initial +rate*60**

Lexical Analysis

Refer the content from Phase1

INPUT BUFFERING:



The lexical analyzer reads characters from the input and passes the tokens to the syntax

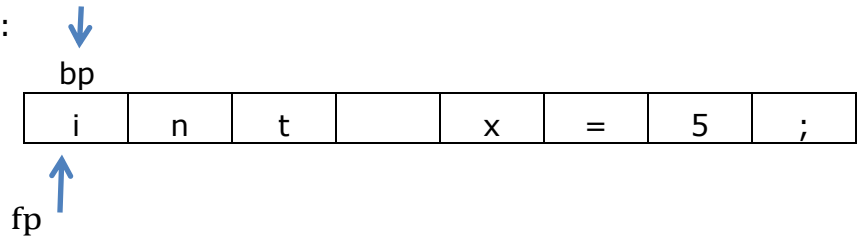
analyzer whenever it is asked for a token.

Lexical Analysis has to access hard disk/ **secondary memory** each time to identify tokens. It is time-consuming and costly. So, the input strings are stored into a buffer and then scanned by Lexical Analysis.

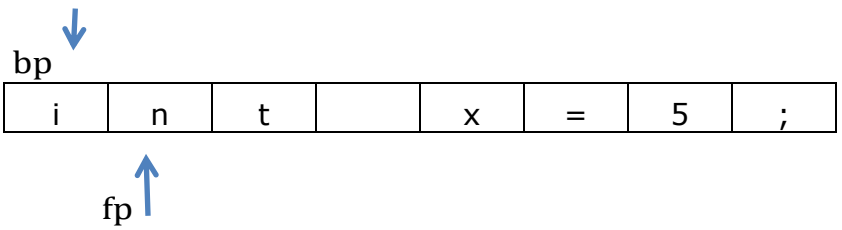
Input buffering in a compiler is a technique used to manage the flow of characters from the source program to the lexical analyzer efficiently. It ensures that the reading of the source code is performed smoothly without excessive I/O operations

Lexical Analysis scans input string from left to right one character at a time to identify tokens. It uses two pointers **begin ptr(bp)** and **forward pointer(fp)** to keep track of the pointer of the input scanned.

Ex:

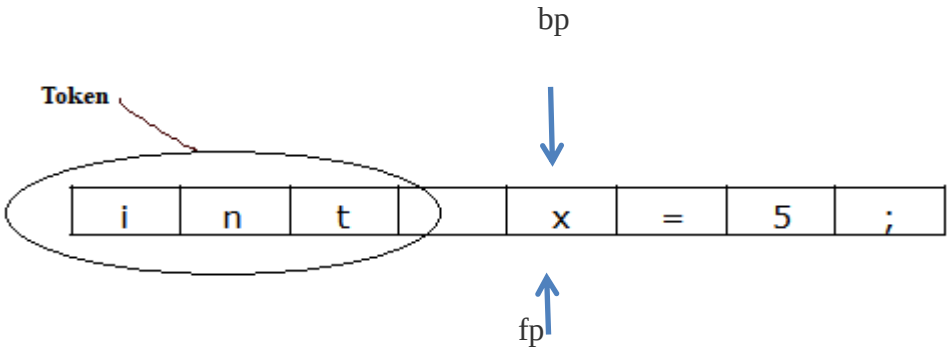


Initially both the pointers point to the first character of the input string



The forward ptr moves ahead to search for end of lexeme. As soon as the blank space is encountered, it indicates end of lexeme. In above example as soon as ptr (fp) encounters a blank space the lexeme

"int" is identified.



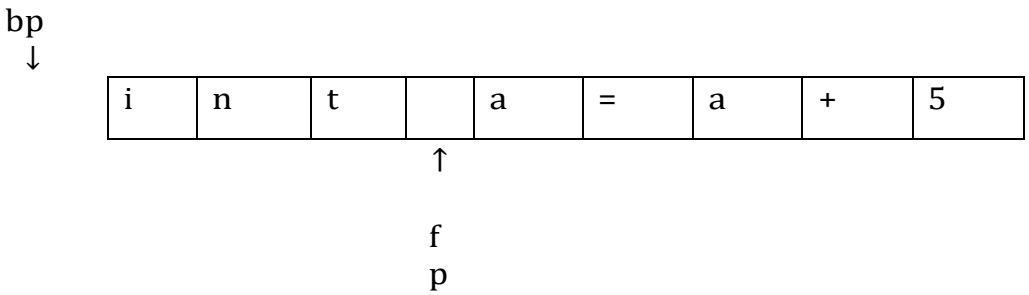
The fp will be moved ahead at white space, when fp encounters white space, it ignore and moves ahead. then both the begin ptr(bp) and forward ptr(fp) are set at next token.

The input character is thus read from secondary storage, but reading in this way from secondary storage is costly. hence buffering technique is used.A block of data is first read into a buffer, and then second by lexical analyzer. there are two methods used in this context:

1. One Buffer Scheme, and
2. Two Buffer Scheme.

One buffer scheme

- In one buffer scheme, only one buffer is used store the input string.
- If the lexeme is very long then it crosses the buffer boundary, to scan rest of thelexeme the buffer has to refilled, that makes the overwriting the first part of lexeme.



one buffer scheme

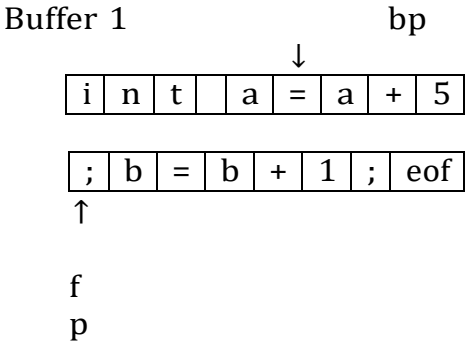
It is the problem with this scheme.

Two buffer scheme

- To overcome the above said problem, two buffers are used to store the input string.The first buffer and second buffer are scanned alternately.
- When the end of the current buffer is reached the other buffer is filled.
- The problem with this scheme is that if length of the lexeme is longer than

length ofthe buffer then scanning input cannot be scanned completely.

- Initially, both the pointers bp and fp are pointing the first buffer. Then the fp movestowards right in search of the end of lexeme.
- When blank character is identified, the string between bp and fp is identified ascorresponding token.
- To identify the boundary of the first buffer end of buffer character should be placedat the end of first buffer.
- In the same way, end of second buffer is also recognized by the end of first buffermark present at the end of second buffer.
- When fp encounters first eof, then one can recognize end of first buffer and hencefilling up of second buffer is started.
- In the same way when second eof is obtained then it indicates end of second buffer.
- Alternately, both the buffers can be filled up until end of the input program andstream of tokens identified.
- This eof character introduced at the end is called sentinel which is used to identifythe end of buffer.



Code for input buffering

If forward at end

of first half then begin reload second half;

forward := forward + 1 end

else if forward at end of second half then

begin reload second half;

move forward to beginning of first half end

else forward := forward + 1;