Compiler Design [UNIT-1 & UNIT-2]

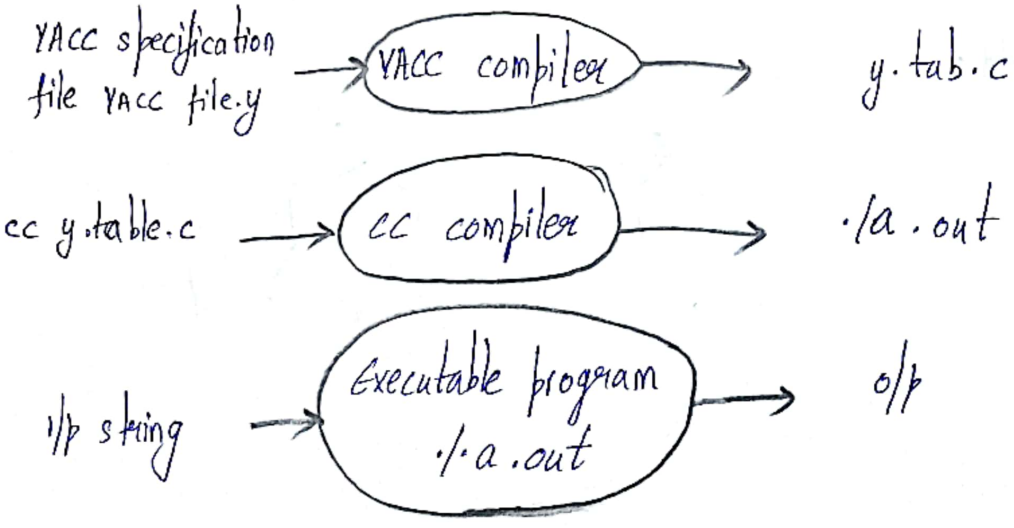
Assignment-1 (VI SEM)

Short Answer Questions:

[1] Discuss Parser Generator - YACC

Ans
- YACC stands for Yet Another Compiler Compiler.
- YACC provides a tool to produce a parser for a given grammar.
- Basically YACC is LALR parser generator.
- YACC can report conflicts or ambiguities (if at all) in the form of error messages.

The typical YACC translator can be represented as:



YACC : PARSER GENERATOR MODEL.

YACC specifications

→ YACC specification file consists of three parts.

Declarative section

TRANSLATION ROLE

C fns.

Example: Syntax:

```
%{
    /* declaration section */
%}
%%
%%
    /* translation rule section */
%%
    /* required C fns */
```

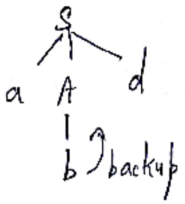
Example: Grammar

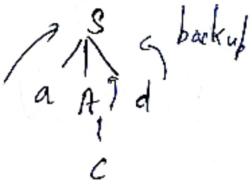
S → aAd | aB

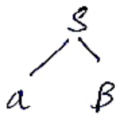
A → b | c

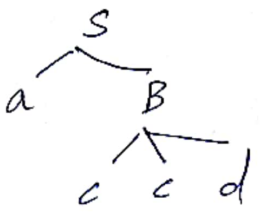
B → ccd | ddc

I/p: accd









[2] Enumerate the rules for FIRST() & FOLLOW() functions

Ans FIRST() function :

FIRST(x) is a set of terminal symbols that are first symbols appearing at R.H.S in derivation of x.

if $x \Rightarrow \varepsilon$ then 'ε' is also in FIRST(x).

following are the rules used to compute the, FIRST()

① if the terminal symbol 'a' then

$$FIRST(a) = \{a\}$$

② if there is a rule $X \rightarrow \varepsilon$ then

$$FIRST(X) = \{\varepsilon\}$$

③ for the rule $A \rightarrow X_1, X_2, X_3 \dots X_K$

$$FIRST(A) = FIRST(X_1) \cup FIRST(X_2) \dots \cup FIRST(X_K)$$

where,

$$K < X_j \leq n$$

such that $1 \leq j \leq K-1$

→

FOLLOW() function:

FOLLOW() function is defined as the set of terminal symbols that appear immediately to the right of A.

In other words,

$$FOLLOW(A) = \{ a \mid S \Rightarrow \alpha A a \beta$$

where, α & β are some grammar symbols may be terminated or non-terminal.

RULES:

1. For the start symbol S place $ in FOLLOW(S).
2. if there is a production $A \rightarrow \alpha B \beta$ then everything in FIRST(β) without 'ε' is to be placed in FOLLOW(B).
3. if there is a production $A \rightarrow \alpha B \beta$ or $A \rightarrow \alpha \beta$

FIRST(β) = {ε} then

FOLLOW(A) = FOLLOW(B)

FOLLOW(B) = FOLLOW(A)

[3] Depict General format for LEX program.

Ans Structure of LEX program:

→ A LEX program has the following form:

```
%{ declaration
%}
%%
  rules
%%
  c fns/auxiliary fns
```

Declaration section

includes declarations of variables, constant and regular definations.

Transition rules: pattern {Action}

- Each pattern is a regular expression, which may use the regular definations of the declaration section.
- The actions are fragments of code, typically written in c.

C functions / Auxiliary fns

- This section holds whatever additional fns are used in the actions.
- These fns can be compiled separately and loaded with the lexical Analyzer.

Long Answer Questions:

14 Construct LL(1) parsing table and the parse the input string id + id * id.

Ans Consider the Grammar

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

STEP 1: Computing FIRST & FOLLOW

→ $E \rightarrow TE'$

is a rule in which the first symbol at R.H.S is T.

Now, $T \rightarrow FT'$ in which the first symbol at R.H.S is F &

where, $F \rightarrow (E) \mid id$

$\therefore FIRST(E) = FIRST(T) = FIRST(F)$

As, $F \to (\epsilon)$

$F \to id$

Hence $FIRST(\epsilon) = FIRST(T) = FIRST(F) = \{(, id\}$

SYMBOLS	FIRST
ϵ	$\{(, id\}$
ϵ'	$\{+, \varepsilon\}$
T	$\{(, id\}$
T'	$\{*, \varepsilon\}$
F	$\{(, id\}$

$\to \epsilon' \to +TE' / \varepsilon$

$FIRST(\epsilon') = \{+, \varepsilon\}$

$\to T' \to *FT' / \varepsilon$

$FIRST(T') = \{*, \varepsilon\}$

FOLLOW(ϵ)

(i) As there is a rule $F \to (\epsilon)$ the symbol ')' appears immediately to the right of ϵ.

∴ ')' will be in FOLLOW(ϵ).

since ϵ is a start symbol, add \$ to FOLLOW of ϵ.

Hence, $FOLLOW(\epsilon) = \{), \$\}$

FOLLOW(ϵ')

(i) $\epsilon \to TE'$

(ii) $\epsilon' \to +TE'$

$\epsilon \to TE'$ the computational rule is $\boxed{A \to \alpha B \beta}$

$A = E$, $\alpha = T$, $B = E'$, $\beta = \varepsilon$ then by rule 3

→

(4)

Everything in FOLLOW(A) = FOLLOW(B)

i.e FOLLOW(E) = FOLLOW(E')

∴ FOLLOW(E') = {), $ }

E' → +TE', the computational rule is A → αBβ.

∴ A = E', α = +T, B = E', β = ε then by rule 3.

Everything, in FOLLOW(A) = FOLLOW(B)

i.e FOLLOW(E') = FOLLOW(E')

∴ FOLLOW(E') = {), $ }

FOLLOW(T)

① E → TE'

② E' → +TE'

① E → TE' the computational rule is A → αBβ

A = E, α = ε, B = T, β = E' then by rule ②.

FOLLOW(B) = { FIRST(β) - ε }

∴ FOLLOW(T) = { FIRST(E') - ε }

= { { +, # } - # }

= { + }

$E' \to +TE'$ we will map it $A \to \alpha B \beta$

$A = E'$, $\alpha = +T$, $B = E'$, $\beta = \varepsilon$ then by rule 3

$$FOLLOW(A) = FOLLOW(B)$$

i.e $FOLLOW(E') = FOLLOW(T)$

$$FOLLOW(E') = \{), \$ \}$$

Finally, $FOLLOW(T) = \{ + \} \cup \{), \$ \}$

$$= \{ +,), \$ \}$$

FOLLOW(T')

① $T \to FT'$

② $T' \to *FT'$

$T \to FT'$ map it with $A \to \alpha B \beta$ then

$A = T$, $\alpha = F$, $B = T'$, $\beta = \varepsilon$ then by rule 3

$$FOLLOW(A) = FOLLOW(B)$$

$$FOLLOW(T') = FOLLOW(T)$$

$$= \{ +,), \$ \}$$

$T' \to *FT'$ we will map $A \to \alpha B \beta$

$A = T'$, $\alpha = *F$, $B = T'$, $\beta = \varepsilon$ then by rule 3

$$FOLLOW(A) = FOLLOW(B)$$

$\therefore$ $FOLLOW(T') = FOLLOW(T)$

$$FOLLOW(T') = \{ +,), \$ \}.$$

FOLLOW(F)

① $T \to FT'$

$T' \to *FT'$

$T \to FT'$, we will map $A \to \alpha B \beta$

$A = T$, $\alpha = \varepsilon$, $B = F$, $\beta = T'$ then by rule 2.

$$FOLLOW(B) = \{FIRST(\beta) - \varepsilon\}$$

$$\therefore FOLLOW(F) = FIRST(T') - \varepsilon$$

$$= \{*\}$$

$T' \to *FT'$ we will map $A \to \alpha B \beta$

$A = T'$, $\alpha = *F$, $B = T'$, $\beta = \varepsilon$ then by rule 3.

$$FOLLOW(A) = FOLLOW(B)$$

$$\therefore FOLLOW(T') = FOLLOW(T)$$

$$= \{+,), \$\}$$

Finally, $FOLLOW(F) = \{*\} \cup \{+,), \$\}$

$$= \{+, *,), \$\}$$

symbols	FIRST	FOLLOW
E	$\{(, id\}$	$\{), \$\}$
E'	$\{+, \varepsilon\}$	$\{), \$\}$
T	$\{(, id\}$	$\{+,), \$\}$
T'	$\{*, \varepsilon\}$	$\{+,), \$\}$
F	$\{(, id\}$	$\{+, *,), \$\}$

STEP 2: Construction of predictive parsing table.

For the rule $A \to \alpha$ of grammar G.

(1) For each 'a' in FIRST(α) create entry

$M[A, a] = A \to \alpha$

where, 'a' is terminal symbol

(2) For ε in FIRST(α) create entry

$M[A, b] = A \to \alpha$

where, 'b' is symbols from FOLLOW(A).

(3) If ε is in FIRST(α) and \$ is in FOLLOW(A) then create entry in the table.

$M[A, \$] = A \to \alpha$.

(4) All the remaining entries in the table M are marked as SYNTAX ERROR.

Question: $E \to TE'$

$E' \to +TE' / \varepsilon$

$T \to FT'$

$T' \to *FT' / \varepsilon$

$F \to (E) / id$.

$E \to TE'$

$A \to \alpha$

$A = E$, $\alpha = TE'$

$\therefore E' = \varepsilon$

FIRST $(TE') = \{ (', id), (\varepsilon) \}$

$= \{ (, id \}$

$M[E, (] = E \to TE'$

$M[E, id] = E \to TE'$

$E \to +TE'$

$E' \to \varepsilon$

$A \to \alpha$

$A = E'$

$\alpha = +TE'$

FIRST $(+TE') = \{+\}$

$\therefore M[E', +] = E' \to +TE'$

$A \to \alpha$

$A = E'$, $\alpha = \varepsilon$

FOLLOW $(E') = \{), \$ \}$

$M[E',)] = E' \to \varepsilon$

$M[E', \$] = E' \to \varepsilon$

$T \to FT'$

$A \to \alpha$

$A = T$, $\alpha = FT'$

FIRST $(FT') =$ FIRST $(F) = \{ (, id \}$

$\therefore T' = \varepsilon$

$M[T, (] = T \to .FT'$

$M[T, id] = T \to FT'$

$T' \to *FT'$

$A \to \alpha$

$A = T', \alpha = *FT'$

$FIRST(*FT') = \{*\}$

$M[T', *] = T' \to *FT'$

$T' \to \epsilon$

$A \to \alpha$

$A = T', \alpha = \epsilon$

$FOLLOW(T') = \{+,), \$\}$

$M[T', +] = T' \to \epsilon$

$M[T',)] = T' \to \epsilon$

$M[T', \$] = T' \to \epsilon$

$F \to (\epsilon)$

$A \to \alpha$

$A = F, \alpha = (\epsilon)$

$FIRST((\epsilon)) = \{(\}$

$M[F, (] = F \to (\epsilon)$

$F \to id$

$A \to \alpha$

$A = F, \alpha = id$

$FIRST(id) = \{id\}$

$M[F, id] = F \to id$.

	id	+	*	(	)	$
E	$E \to TE'$	error	error	$E \to TE'$	error	error
E'	error	$E' \to +TE'$	error	error	$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$	error	error	$T \to FT'$	error	error
T'	error	$T' \to \epsilon$	$T' \to *FT'$	error	$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$	error	error	$F \to (E)$	error	error

PRT 3 : Parsing the i/p string

i/p : id + id * id

E → TE'
E' → +TE'/ε
T → FT'
T' → *FT'/ε
F → (E) | id

Stack	i/p	Action
$E	id + id * id $	-
$E'T	id + id * id $	E → TE'
$E'T'F	id + id * id $	T → FT'
$E'T'id	id + id * id $	F → id
$E'T'	+ id * id $	-
$E'	+ id * id $	T' → ε
$E'T+	+ id * id $	E' → +TE'
$E'T	id * id $	-
$E'T'F	id * id $	T → FT'
$E'T'id	id * id $	F → id
$E'T'	* id $	-
$E'T'F*	* id $	T' → *FT'
$E'T'F	id $	-
$E'T'id	id $	F → id
$E'T'	$	-
$E'	$	T' → ε
$	$	E' → ε

→

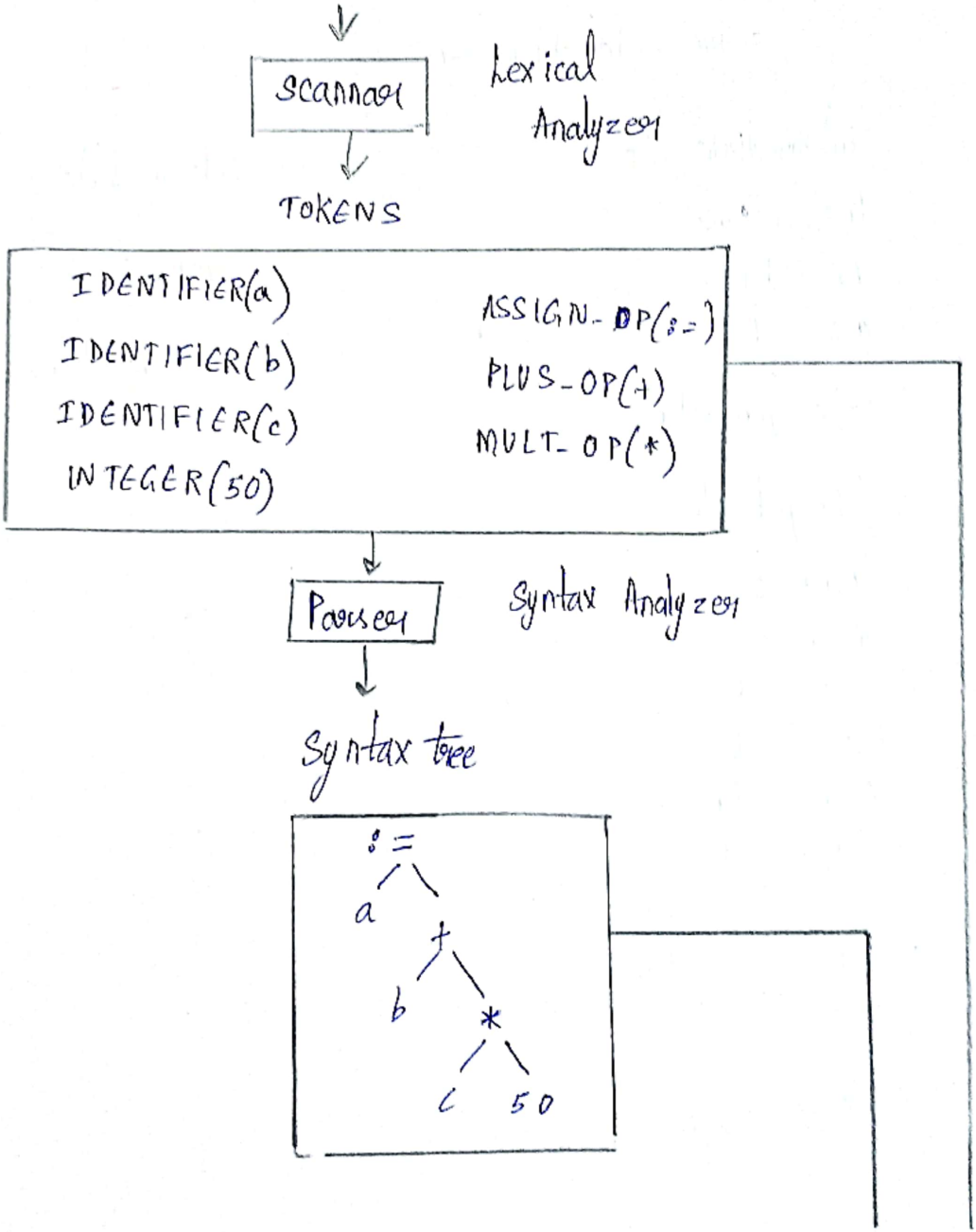
[5] Explain the Phases of Compiler? Write down the output of each phase for the following.

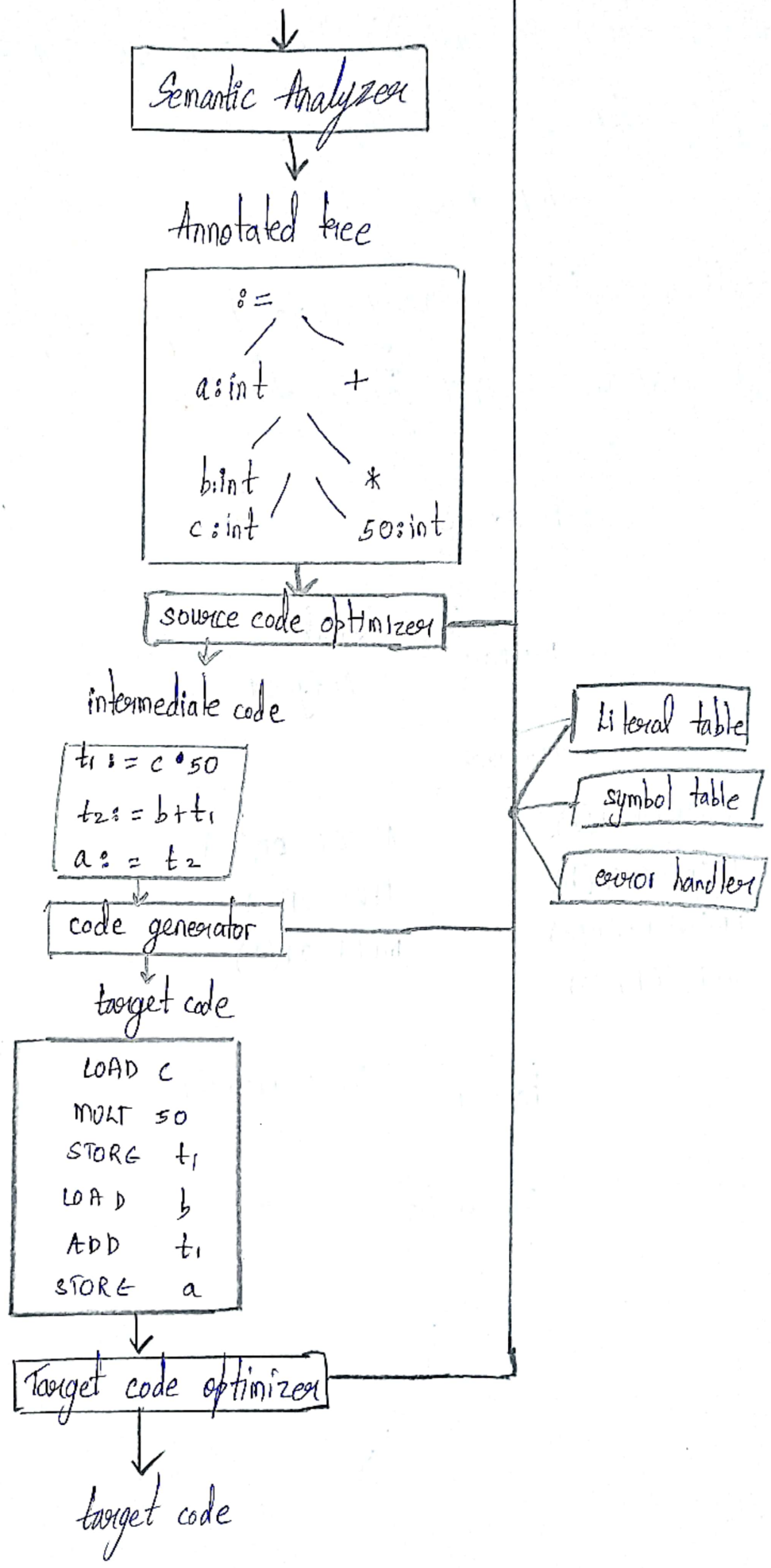
a := b + c * 50.

Ans A compiler consists internally a no. of steps, or phases, that performs distinct logical operations.

source code

a := b + c * 50





Phases of a Compiler:

- SCANNAR: This phase of the compiler does the actual reading of the source program, which is usually in the form of a stream of character.
- The SCANNAR performs what is called lexical Analysis, it collects sequence of character into meaningful units called Tokens.

- PARSER:
- The parser receives the source code in the form of tokens from the scanner and perform syntax Analysis, which determines the structure of the program.
- The results of syntax analysis are usually represented as a parse tree or syntax tree

- SEMANTIC ANALYZER:
- The semantics of a program are its "meaning".
- The semantics of a program determine its runtime behaviour, but most programming languages have features that can be determined prior to execution & yet can't be conveniently expressed as syntax or Analyzed by the parser.

- SOURCE CODE OPTIMIZER:
- Compilers often include a no. of code improvements or optimizations steps.
- Individual compilers exhibit a wide variation not only in the kinds of optimizations performed but also in the placement of the optimization phases.
- Source code optimizers may use 3-addr code by referring to its o/p as intermediate code.
- Intermediate code refers as Intermediate representation (IR)
- CODE GENERATOR:
- The code generator takes the intermediate code or IR and generates code for the target machine.
- TARGET CODE OPTIMIZER:
- In this phase, the compiler attempts to improve the target code generated by the code generator.
- Such improvements includes choosing addressing modes to improve performance, replacing slow inst by faster ones, and eliminating redudant unnecessary operations.

X ——— X