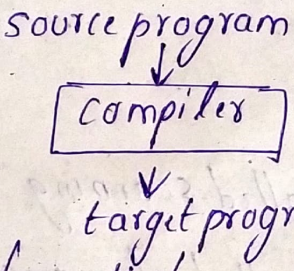
ASSIGNMENT - I

(1) Define compiler. explain about phases of a compiler with suitable example

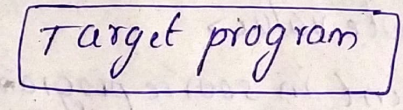
A) Compiler :-

→ Compiler is a program that need a program in one language (source lang) and translates it into equivalent program in another language (target lang)

→ During compilation, role of compiler is to detect the errors in source code & report them to user



→ The target program is also called as executable program



phases of a compiler :-

in compiler we see that there are 2 phases

1.) Analysis 2.) synthesis

Analysis :

→ It breaks up source program into pieces and check it with pattern for converting tokens those tokens are needed for rest compilation

→ It checks whether the source program is syntactically or semantically correct or not

→ finally it converts the source program into intermediate representation called intermediate code generator

synthesis :

→ it construct target program from the intermediate representation and put information in symbol table.

→ it applies optimization, if optimization doesn't expose any new problems

In another Analysis part carried out in 4 phases and synthesis part carried out in 6 2 phases

1) lexical analysis
2) syntax analysis
3) semantic analysis
4) intermediate code generation
5) code optimization
6) code generation

1) lexical analysis :- it also called scanning . It scans the source program and broken up into groups called strings called token in form

< token name, attribute value>

ex :- Assignment statement in source program

position = initial + rate * 60

2-) syntax Analysis :- it is also called parsing . It creates tree like structure shows that grammatical structure of token structure

3) Semantic Analysis :- it uses syntax tree and information in symbol table to check source program is semantically correct or not

type checking is important in semantic analysis

4) intermediate code generator :- After semantic analysis, many compilers generate intermediate representation for source program

→ Intermediate representation can be done help of three address code

ex: $t_1 = \text{into to real}(60)$
$t_2 = id_3 * t_1$
$t_3 = id_2 + t_2$
$id_1 = t_3$

5.) code optimization :- code optimization improves intermediate code to get better target program
→ thes optimization improves running time of target program

ex: $t_1 = id_3 * 60.0$
$id_1 = id_2 + t_1$

6.) code generator :- it takes intermediate representation of source program and maps it into target language.

ex: LDF R2, id3
MULF R2 ~~id~~ # 60.0
LDF R1, id2
ADDF R1, R2
STF id1, R1

---

(2) Explain about syntax directed translation with suitable example

A.) Syntax directed Translation

syntax directed definitions :-

→ syntax directed definition (SDD) is context free grammer together with attributes and rules attributes are associated with grammer symbols and rules associated with productions

→ Attributes may be of any kind: numbers, type, string or memory locations

→ if x is symbol and a is attribute then we write x.a to denote value of at particular node labelled x in parse tree

→ We shall deal with two kinds of attributes

1. synthesized attribute.
2. inherited attribute

Synthesized attribute :

→ synthesized attribute for non termal. A at parse tree node N is defined by semantic rules associated with production at N and A is head

→ synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself

→ An SDD that involves only synthesized attributes called S-attributed

ex :- consider grammer for desk calculator

$L \rightarrow En$
$E \rightarrow E_1 + T$
$E \rightarrow T$
$T \rightarrow T_1 * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow$ digit

syntax directed defination for disk calculator is given in below table with semantic actions for each production

production	semantic rules
$L \rightarrow En$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit .lexval

inherited Attribute:

→ inherited attribute at node N is defined only in terms of attributes values of N's parent, N itself and N's attribute

→ An SDD is called L-attribute definition of it follows below rules with production $A \to X_1 X_2 \cdots X_n$ for computing X_j - a

→ L-attributes definition consist of both s-attribute & L-attribute

Ex:- consider grammer

$D \to TL$
$T \to int$
$T \to float$
$L \to L_1, id$
$L \to id$

Syntax definition for simple type declaration shown below table

production	semantic rules
$D \to TL$	L.inh = T.type
$T \to int$	T.type = integer
$T \to float$	T.type = float
$L \to L_1, id$	L_1.inh = L.inh addType(id.entry, L.inh)
$L \to id$	addtype (id.entry, L.inh)

③ Explain about the roal of the lexical analysis

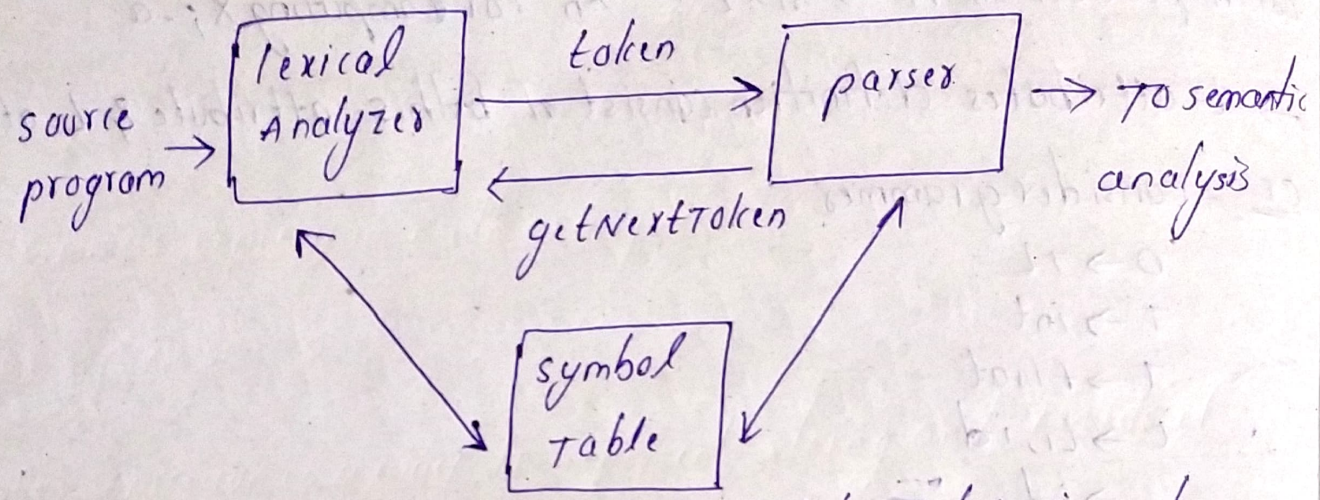
A) Role of lexical Analyzer.

→ Lexical analyser reads input characters in the source program group them into lexemes and produce sequence of tokens as output

→ output of lexical analyser (sequence of tokens) sent to parser for syntax analysis

→ lexical analyser interacts with yr symbol table to collect

information about tokens before passing to parser
→ Interaction between lexical analyser and parser is implemented by method getNextToken() issued by parser



→ some lexical analysers do additional task like stripping out comments and whitespaces another one is relating error msg to source program with line numbers
→ it is divided into 2 processes one of deleting comments and white space and another one is for producing sequence of tokens

(4) Explain about input buffering and types:

A:) Input buffering:-
→ Input buffering is special technique developed to reduce amount of time taken to process large characters during compilation
→ In input buffering schema, lexical analyzer scans data form left to right one character at a time
→ In this schema, Buffers are used and each buffer size is N. if less the number of characters of input file than N then use special

character eof to end of source file

→ if uses 2 pointers to keep track of the portion of input

1. pointer lexeme Begin, marks the beginning of current lexeme
2. pointer forward, scans forward until a pattern match is found

Types of input buffering

There are 2 types used in this i/p buffering

1) one buffer schema
2) two buffer schema

one buffer schema:

→ In this one buffer schema, only one buffer is used to store i/p string

→ problem in this schema is if lexeme is very long then it cross boundary, to scan rest we reset the buffer. it will overwrite the starting part of lexeme

→ To overcome the above problem 2 buffer schema is introduced

two buffer schema:

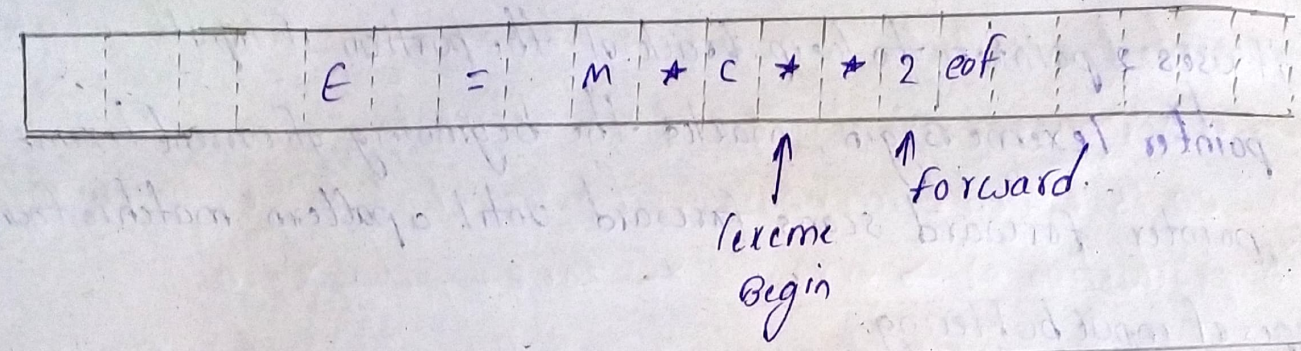
→ In this schema, two buffers are used to ~~stre~~ store input string

→ Here first buffer and second buffer scanned alternatively

→ while end of the one buffer is reached other buffer is filled

→ once lexeme is determined, forward is set to character at its end. after lexeme is recorded as token in symbol table then lexeme Begin set to character immediately after lexeme just found.

→ each lexeme is ~~defi~~ identified by scanning one character that character must be removed before matching to pattern



(5) Define regular expression with examples

A) Regular expression :

→ Regular expression is a way to represent string and words of given language

→ we were able to describe identifiers by set of letters and digits by the language operators union, concatenation closure as shown below

letter (letter | digit)*

→ larger regular expressions are built from smaller one in one of the following ways

→ suppose r and s are regular expression then

1. (r)|(s) is regular expression denoting language L(r) ∪ L(s)
2. (r)(s) is regular expression denoting language L(r)L(s)
3. r* is regular expression denoting language (L(r))*
4. r is regular expression denoting language L(r)

→ some algebric laws for regular expressions used in language are shown below

law	Description
$r/s = s/r$	/ is commutative
$r/(s/t) = (r/s)/t$	/ is associative
$r(s/t) = rs/rt ; (s/t)r = sr/tr$	concatenation is distributive
$\varepsilon r = r\varepsilon = r$	ε is identity for concatenation
$r^* = (r/\varepsilon)^*$	ε is guaranteed in closure
$r^{**} = r^*$	* is idempotent

→ some examples are in Regular expressions are

$R = ab^*b$

$L = \{a, aba, aab, aba, aaa, abab \ldots\}$

$R = \{a+ab\}^*$

$R = a^*b^*c^*$

$L = \{\varepsilon, 00, 0000, 000000, \ldots\}$

$R = [(0+1)^* 0 (0+1)^* 1 (0+1)^*] + [(0+1)^* 1 (0+1)^* 0 (0+1)^*]$

→ 0 or 1

0+1

→ 0 or 11 or 111

0+11+111

→ Regular expression over $\Sigma = \{a, b, c\}$ that represent all string of length 3. $(a+b+c)(a+b+c)(a+b+c)$

→ string having one or more a's at