Question 1:- What are the two parts of compilation explain briefly?

Answer

Answer 1

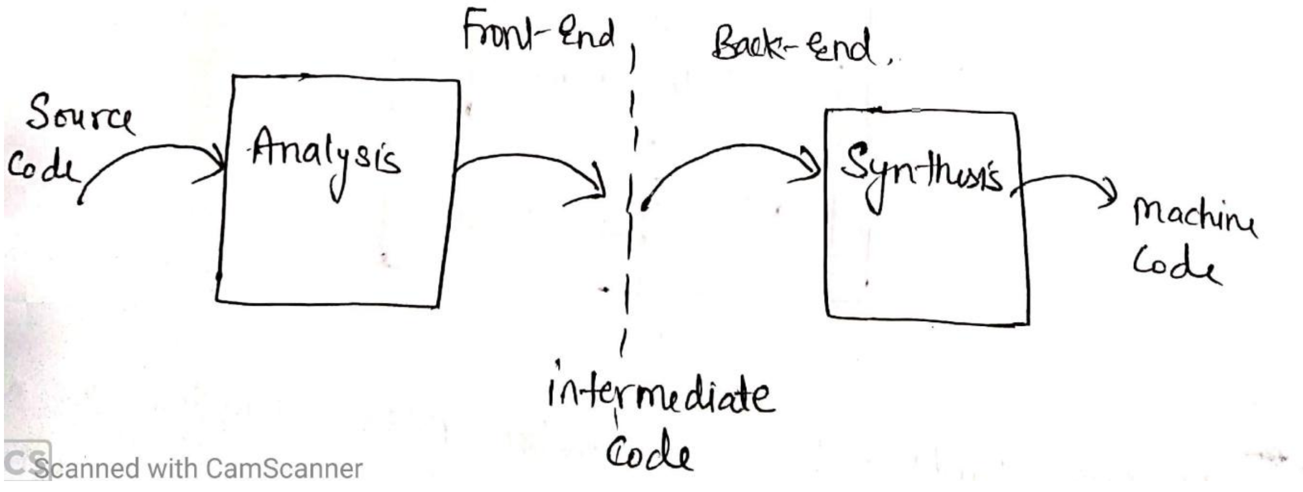
There are two parts of compilation

(i) Analysis

(ii) Synthesis

Analysis Phase :- This phases is known as the front end of the compiler, the analysis phase of the compiler reads the source program, divides it into core parts and then checks for lexical analyzer, syntax Analyzer and semantic Analyzer

Synthesis Phase :- This phases is known as the back-end of the compiler, the synthesis phases the target program with the help of intermediate source code representation.

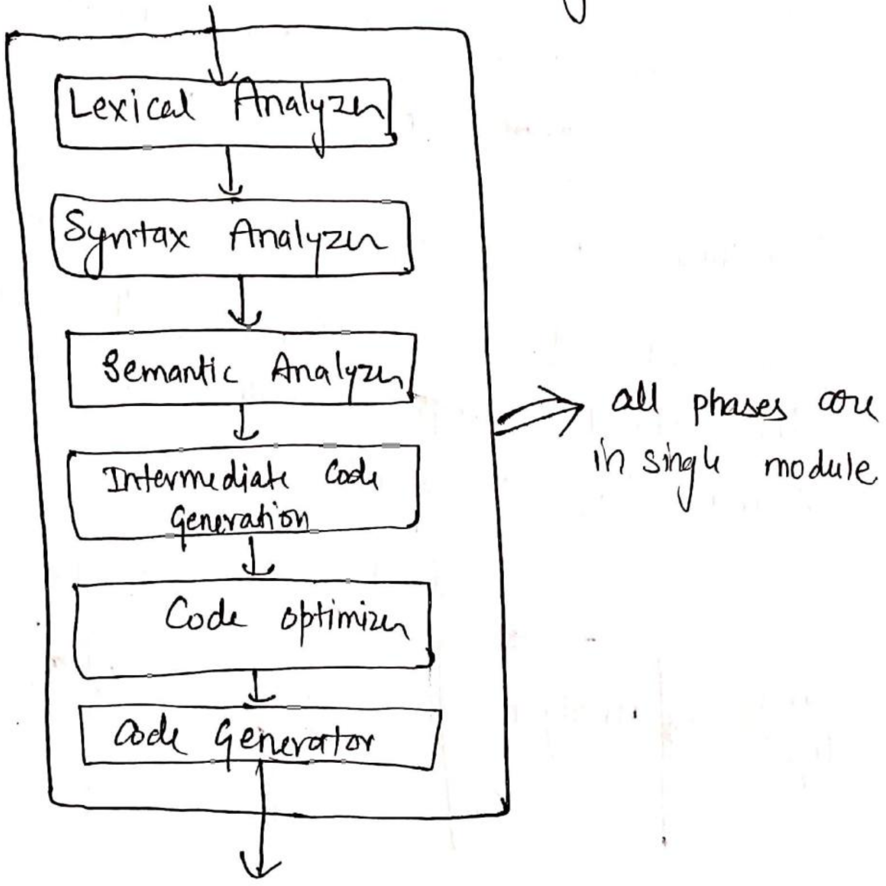


Question 2: What are the classifications of compiler?

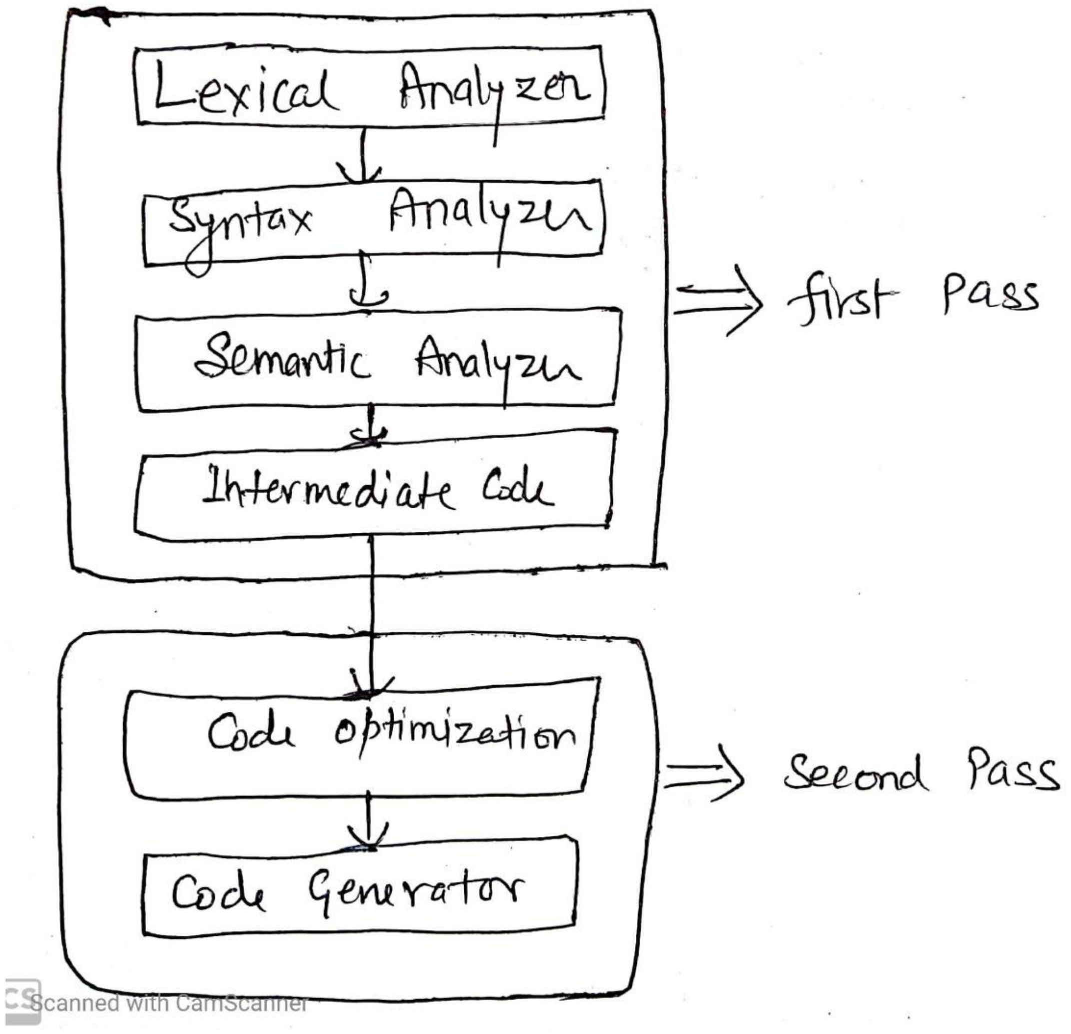
Answer

There are mainly 2 classification of compiler
- Single pass compiler
- multi Pass compiler

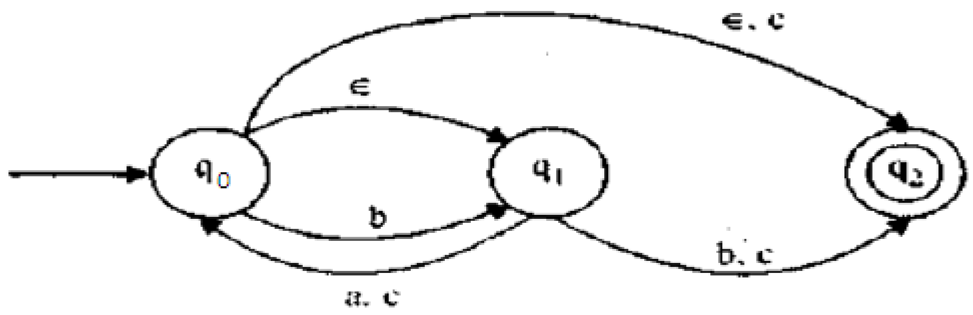
Single Pass compiler :-
If we combine all phases of compiler in single module it is known as single Pass Compiler



Multi Pass Compiler :- A multi pass compiler is a type of compiler that processes the source code or abstract syntax tree of a program multiple times. In multipass Compiler we divided phases in two pass



Question 3: Convert the following NFA to equivalent DFA and hence minimize the number of states in the DFA.



Answer 3

$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} = S_1$

$\epsilon\text{-closure}(\delta(S_1, a)) = \epsilon\text{-closure}(q_0)$

$= \{q_0, q_1, q_2\} = S_1$

$\epsilon\text{-closure}(\delta(S_1, b)) = \epsilon\text{-closure}(q_1, q_2)$

$= \{q_1, q_2\} = S_2$

$\epsilon\text{-closure}(\delta(S_1, c)) = \epsilon\text{-closure}(q_2, q_0)$

$= \{q_0, q_1, q_2\} = S_1$

$\epsilon\text{-closure}(\delta(S_2, a)) = \epsilon\text{-closure}(q_0)$

$= \{q_0, q_1, q_2\} = S_1$

$\epsilon\text{-closure}(\delta(S_2, b)) = \epsilon\text{-closure}(q_2)$

$= q_2 = S_3$

$\epsilon\text{-closure}(\delta(S_2, c)) = \epsilon\text{-closure}(q_0)$

$= \{q_0, q_1, q_2\} = S_1$

$$\epsilon\text{-closure}(\delta(S_3, a)) = \epsilon\text{-closure}(\phi)$$
$$= \phi.$$

$$\epsilon\text{-closure}(\delta(S_3, b)) = \epsilon\text{-closure}(\phi)$$
$$= \phi.$$

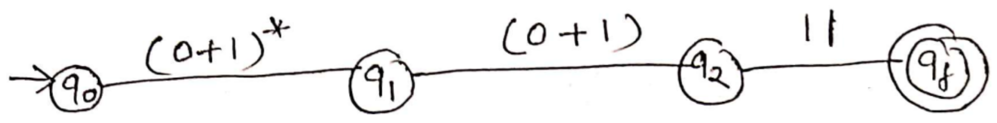
$$\epsilon\text{-closure}(\delta(S_3, c)) = \epsilon\text{-closure}(\phi)$$
$$= \phi.$$

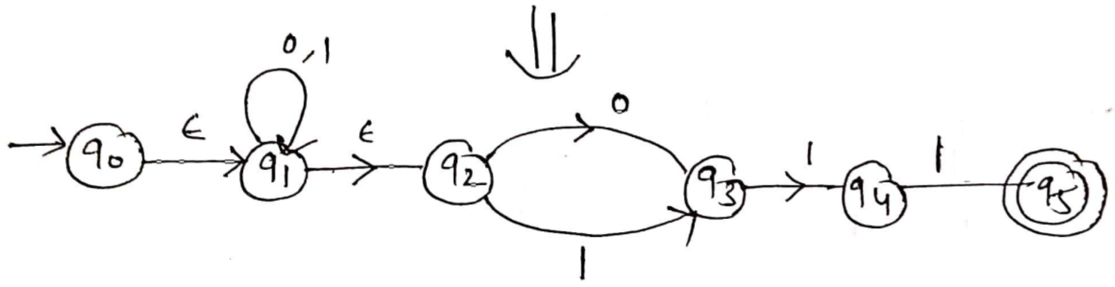
State	a	b	c
S_1	S_1	S_2	S_1
S_2	S_1	S_3	S_1
S_3	ϕ	ϕ	ϕ.

Question 4: Construct the minimized DFA for the regular expression (0+1)*(0+1) 11.

Answer

NFA ϵ - move diagram for the regular expression is





Now remove the ϵ from the above diagram

ϵ - closure $(q_0) = \{q_0, q_1, q_2\} = S_1$

ϵ-closure $(\delta(S_1, 0)) = \epsilon$-closure$(q_1, q_3)$

$= \{q_1\ q_2, q_3\} = S_2$

ϵ-closure $(\delta(S_1, 1)) = \epsilon$-closure$(q_1, q_3)$

$= \{q_1, q_2, q_3\} = S_2$

ϵ- closure$(\delta(S_2, 0)) = \epsilon$-closure$(q_1, q_3) = S_2$

ϵ-closure $(\delta(S_2, 1)) = \epsilon$-closure (q_1, q_3, q_4)

$= \{q_1, q_2, q_3, q_4\} = S_3$

$\epsilon\text{-closure}(\delta(S_3, 0)) = \epsilon\text{-closure}(q_1, q_3)$

$= S_2$

$\epsilon\text{-closure}(\delta(S_3, 1)) = \epsilon\text{-closure}(q_1, q_3, q_4, q_5)$

$= \{q_1, q_2, q_3, q_4, q_5\} = S_4$

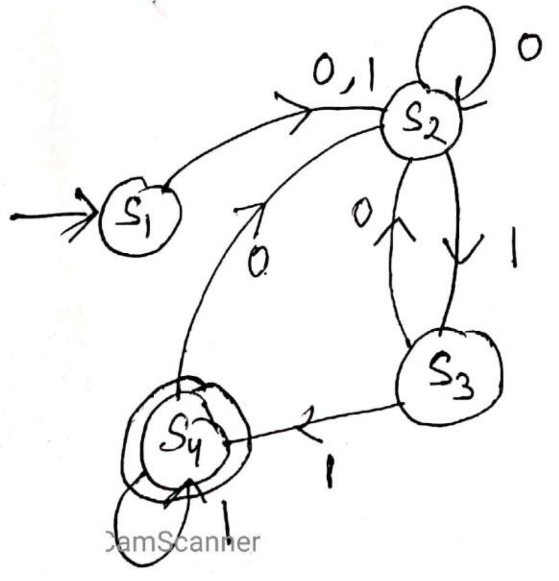
$\epsilon\text{-closure}(\delta(S_4, 0)) = \epsilon\text{-closure}(q_1, q_3)$

$= S_2$

$\epsilon\text{-closure}(\delta(S_4, 1)) = \epsilon\text{-closure}(q_1, q_3, q_4, q_5)$

$= S_4.$

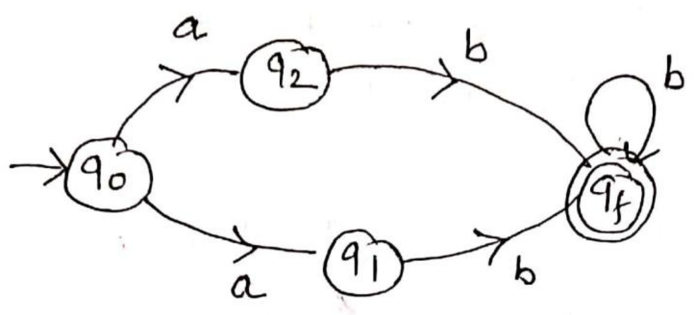
	0	1
$\rightarrow S_1$	S_2	S_2
S_2	S_2	S_3
S_3	S_2	S_4
S_4	S_2	S_4



Question 5:Draw NFA for the regular expression ab*/ab.

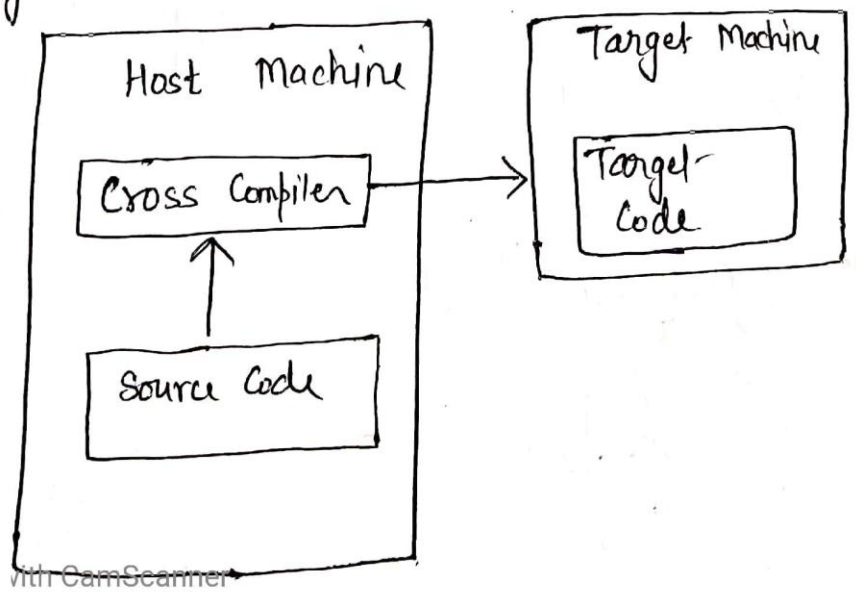
Answer

NFA for the Regular expression ab* | ab



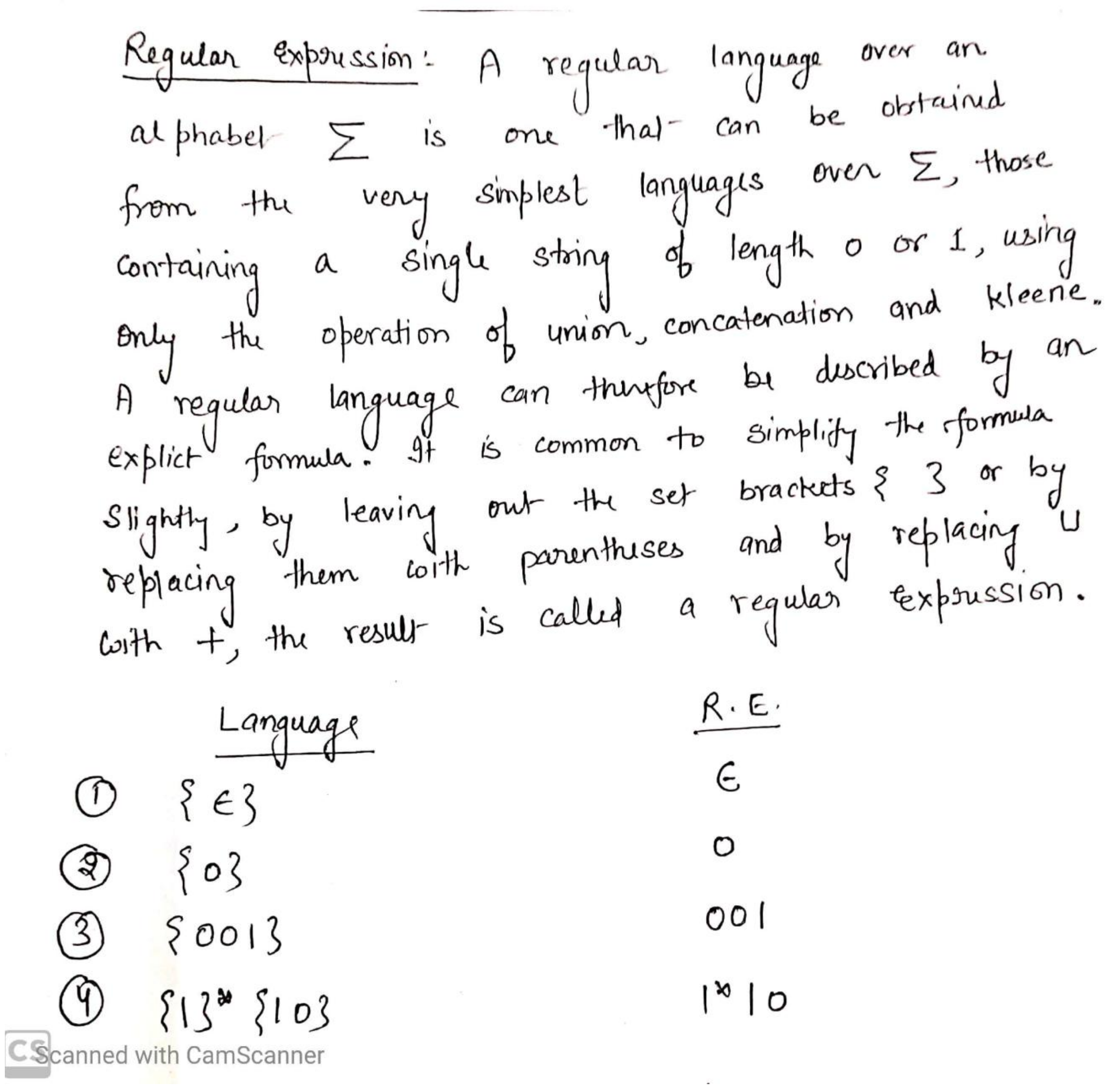
Question 6:What is cross compiler?

Cross Compiler :- A cross compiler is a compiler capable of creating executable code for a platform other than the one which the compiler is running.



Question 7:What do you mean by regular expression?

Answer



Regular expression: A regular language over an alphabet Σ is one that can be obtained from the very simplest languages over Σ, those containing a single string of length 0 or 1, using only the operation of union, concatenation and kleene. A regular language can therefore be described by an explict formula. It is common to simplify the formula slightly, by leaving out the set brackets { } or by replacing them with parentheses and by replacing ∪ with +, the result is called a regular expression.

	Language	R.E.
1	$\{\epsilon\}$	ϵ
2	$\{0\}$	0
3	$\{001\}$	001
4	$\{1\}^* \{10\}$	1^*10

Question:8 Differentiate compiler and interpreter.

Answer

Difference b/w Compiler and Interpreter

Compiler	Interpreter
① Scan the entire program and translate it as whole into machine code	① Translate the program one statement at a time.
② Generates intermediate object Code which is further require linking, hence require more memory	② No intermediate object code is generated, hence are memory efficient
③ Execution time is faster	③ Execution time is slower
④ Debugging is hard	④ Debugging is easy.
⑤ C, C++ ex.	⑤ Paython ex.

Question 9:Discuss the challenges in compiler Design

Answer

Challenges of Compiler Designing

* lexical Analysis.
* Regular expression
* Syntax Analysis
* Error Recovery
* Semantic Analysis
* The compiler itself should be bug free.

Question 10:What is translator?

Answer

Translator :- A program written in high-level language is called as source code. To convert the source code into machine code, translator are needed.

A translator takes a program written in a source language as input and converts into a program in ~~targest~~ language as output.

Types of Translator

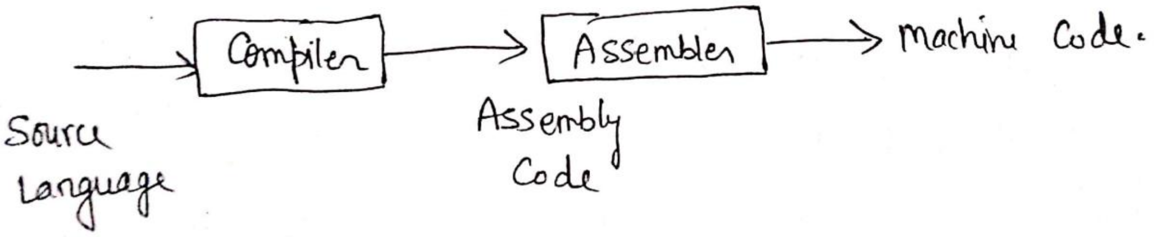
1. Compiler
2. Interpreter

Question 11: What is compiler and assembler.

Answer

Compiler :- Compiler is a program which converts Source language into target language.

Assembler :- Assembler is a program which converts Assembly language into machine code.



Question 12: Write down the regular expression for

- The set of all string over {a,b}such that fifth symbol from right is ‘**a**’.
- The set of all string over {a,b}such that every block of four consecutive symbol contain at least two a.

Answer

(i) $(a+b)^* a (a+b)(a+b)(a+b)(a+b)$

(ii) $a^* + a^*b + a^*ba + a^*bb$

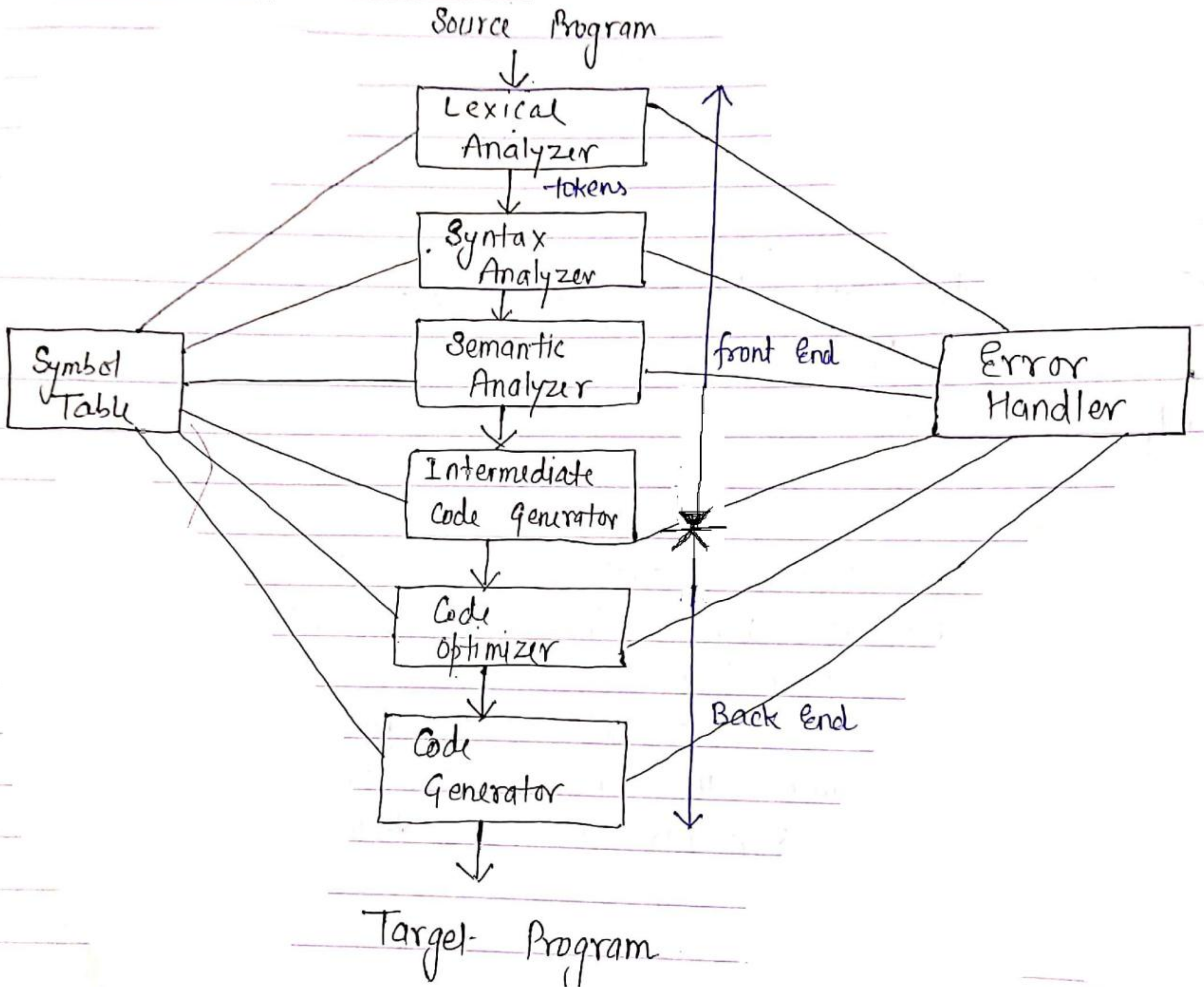
$a^*baa + a^*bab + a^*bba + a^*baaa$

$a^*baab + a^*baba + a^*bbaa$

Question 13:Explain the necessary phases and passes of compiler design.

Answer

Phases of a Compiler :-



Lexical Analyzer :- Lexical analyzer reads the source program character by character and returns the tokens of the source program.

A token describes a pattern of characters having some meaning in the source program (such as identifiers, operators, keywords, numbers, delimiters and so on).

$$x = a + b * 60;$$

$$\downarrow$$

$$\boxed{\text{Lexical Analyzer}}$$

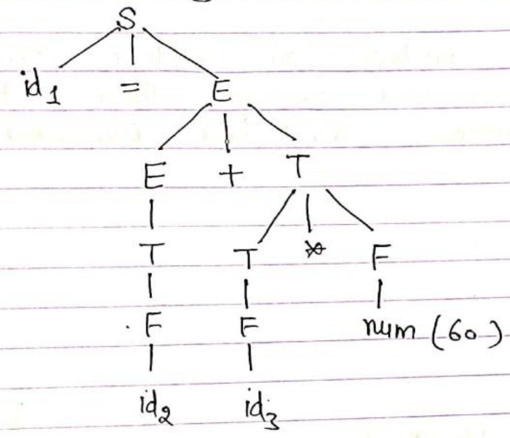
$$\downarrow$$

$$\langle id,1\rangle \ \langle =\rangle \ \langle id,2\rangle \langle +\rangle \ \langle id,3\rangle \ \langle *\rangle \ \langle 60\rangle$$

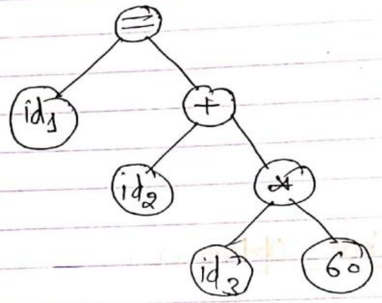
Syntax Analyzer :- A syntax analyzer creates the syntactic structure of the given program. A syntax analyzer is also called a parser.

Ex:- $x = a + b * 60$
$id_1 = id_2 + id_3 * 60$

Grammar :- $S \to id = E;$
$E \to E+T / T$
$T \to T*F / F$
$F \to id / num$

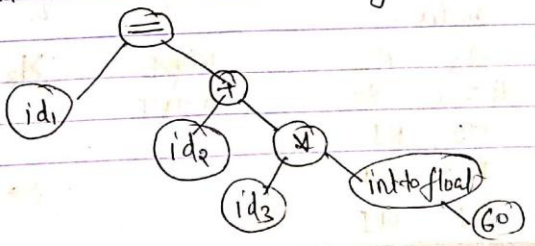


parse tree



Syntax tree

Semantic Analyzer :- A semantic analyzer checks the source program for semantic errors and collects the type information for the code generation. Type - checking is an important part of semantic analyzer. Normally semantic information cannot be represented by a CFG used in syntax analyzers.



Intermediate Code Generator :-

t_1 = inttofloat (60);
t_2 = $id_3 * t_1$
t_3 = $id_2 + t_2$
$id_1 = t_3$

A compiler may produce an explicit intermediate codes representing the source program. These intermediate codes are generally machine architecture independent.

Code Optimizer :- $t_1 = id_3 * 60.0$
$id_1 = id_2 + t_1$

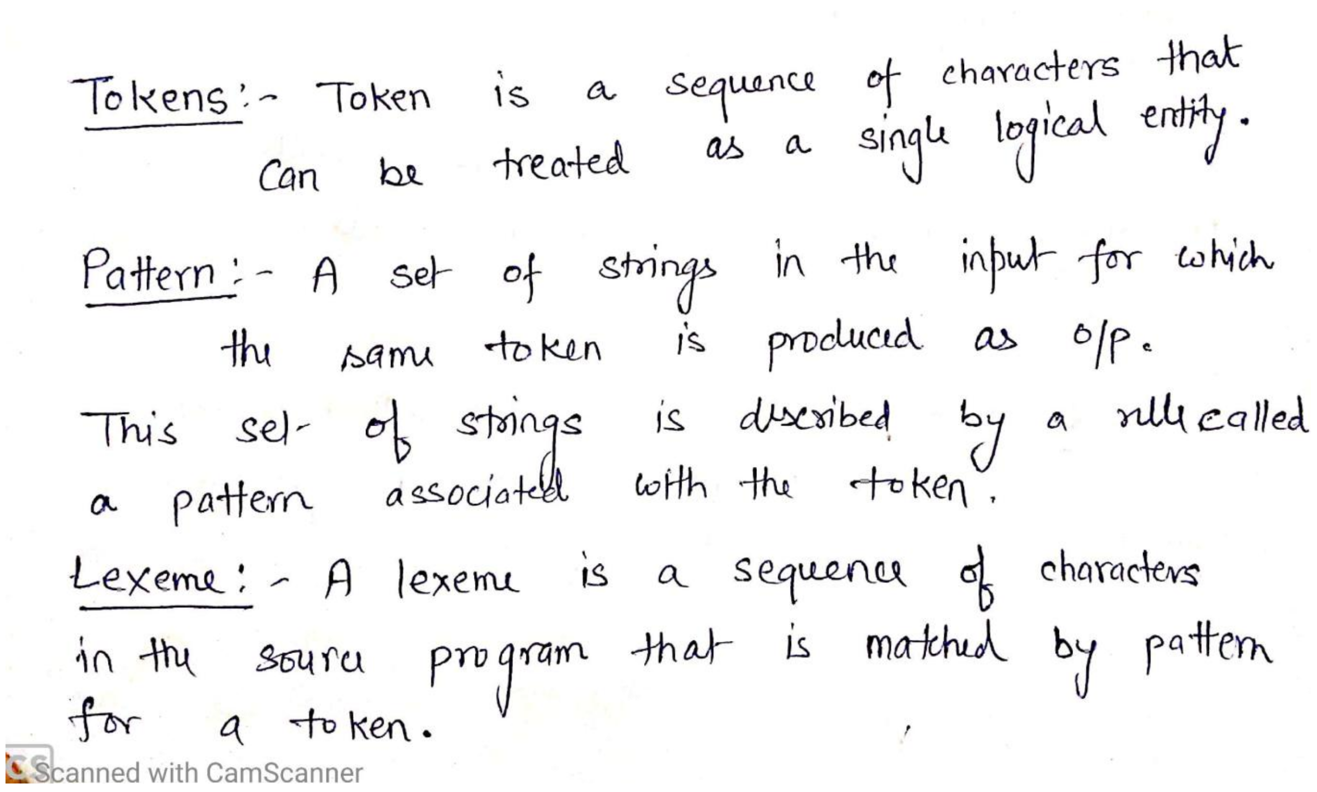
The code optimizer optimizes the code produced by the intermediate code generator in the terms of time and space.

Code Generator :- Produces the target-language in a specific architecture. The target program is normally a relocatable object file containing the machine codes.

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

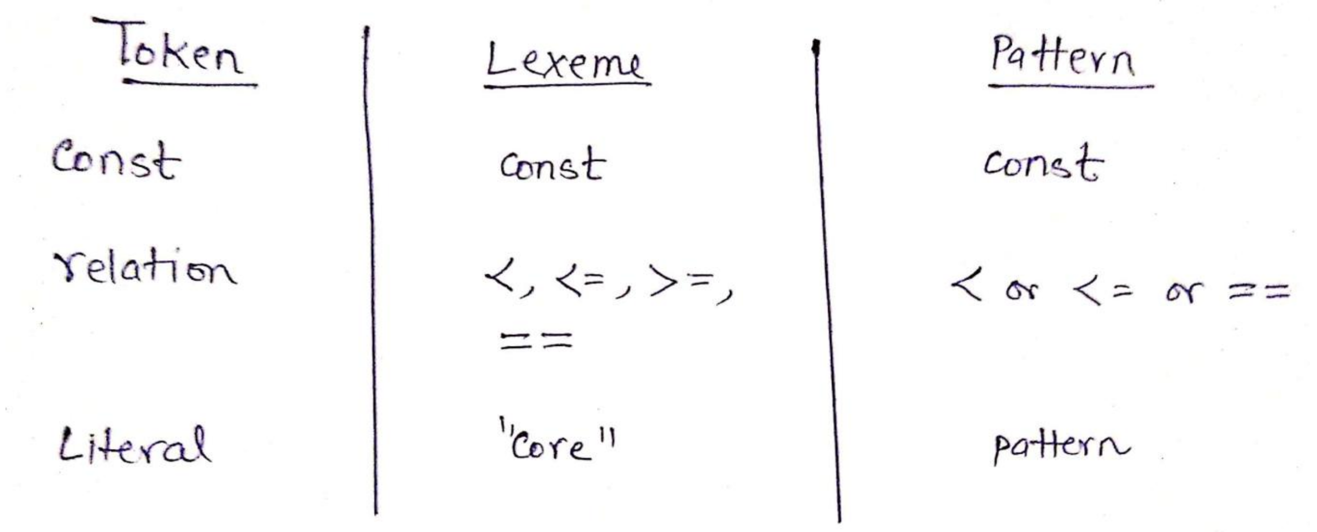
Question 14 Explain the term lexeme, pattern, tokens.

Answer


Tokens :- Token is a sequence of characters that can be treated as a single logical entity.

Pattern :- A set of strings in the input for which the same token is produced as o/p.
This set of strings is described by a rule called a pattern associated with the token.

Lexeme :- A lexeme is a sequence of characters in the source program that is matched by pattern for a token.



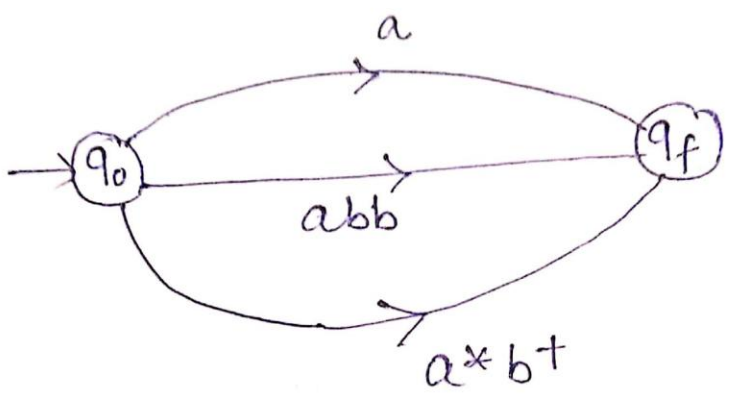
Token	Lexeme	Pattern
Const	const	const
relation	<, <=, >=, ==	< or <= or ==
Literal	"core"	pattern

Question 17:Construct the NFA for the regular expression a/abb/a*b+ by using Thomson's Construction methodology.

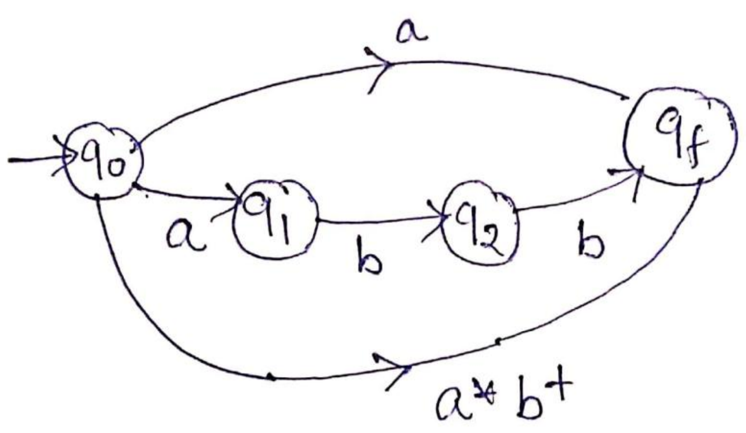
Answer

NFA for the R.E $a/abb/a^*b^+$ by using thomson's construction methodology

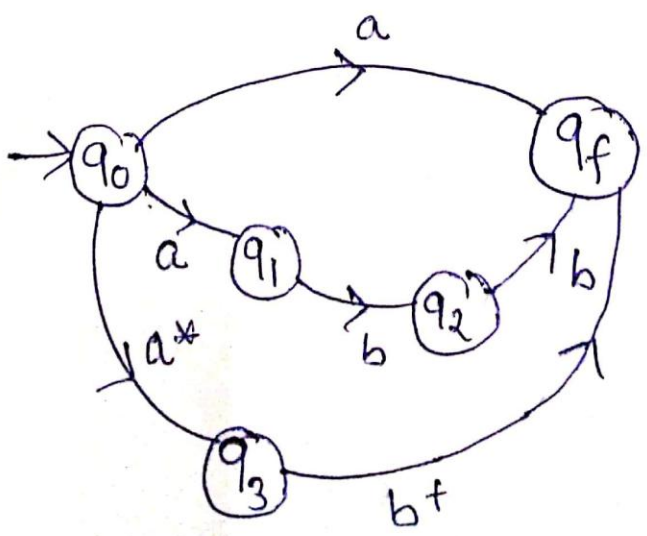
Step ①



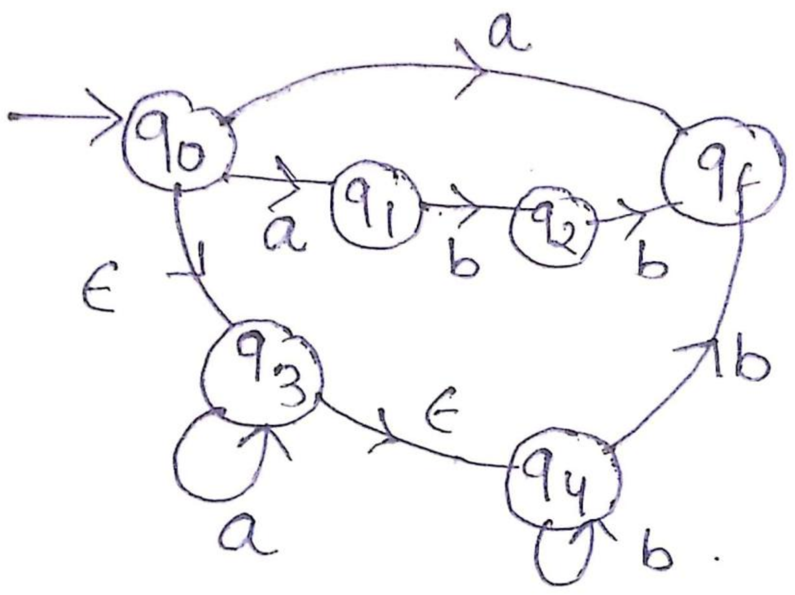
Step ②



Step ③



Step ④



Prepared By:

Prabhat Shukla

UCER, CS/IT Department

Prayagraj

Note: For any discrepancy contact

Whatsapp no.: - 8957419537

Email Id:- shuklaprabhat2k7@gmail.com