School of Computing
CIA-I Exam – April 2022
Course Code: CSE402
Course Name: Compiler Engineering
Year, Semester and Branch: IIIrd Year -VI Sem - CSE
ANSWER KEY
PART-A (10*2=20 marks)

Answer all

1. (aa)*(bb)*b
2. infinite loop and back tracking
3.

```
%{
#include<stdio.h>
int i = 0;
%}
%%
([a-zA-Z0-9])* {i++;}
"\n" {printf("%d\n", i); i = 0;}
%%
int main()
{
yylex();
return 0;
}
```

4.

It is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar.	It is a parsing strategy that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar.
Top-down parsing attempts to find the left most derivations for an input string.	Bottom-up parsing can be defined as an attempts to reduce the input string to start symbol of a grammar.
In this parsing technique we start parsing from top (start symbol of parse tree) to down (the leaf node of parse tree) in top-down manner.	In this parsing technique we start parsing from bottom (leaf node of parse tree) to up (the start symbol of parse tree) in bottom-up manner.
This parsing technique uses Left Most Derivation.	This parsing technique uses Right Most Derivation.
It's main decision is to select what production rule to use in order to construct the string.	It's main decision is to select when to use a production rule to reduce the string to get the starting symbol.

5.

- Tokens- Sequence of characters that have a collective meaning.
- Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token
- Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.

6.
Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variables i.e. it stores information about the scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

7.
Action used in Bottom-Up Parser. It generates the Parse Tree from Leaves to the Root. **The input string will be reduced to the starting symbol**. This reduction can be produced by handling the rightmost derivation in reverse, i.e.,from starting symbol to the input string.

8. classifier table, transition table and token type table
9. %token it refers list of tokens forwarded from scanner and %left defines left association for the given operator
10.

```
S : S1 { printf("input accepted"); }
S1 : IF '(' LEXPR ')' '{' '}';
LEXPR : ID LT ID ;
```

PART-B (3*10=30 marks)

Answer all

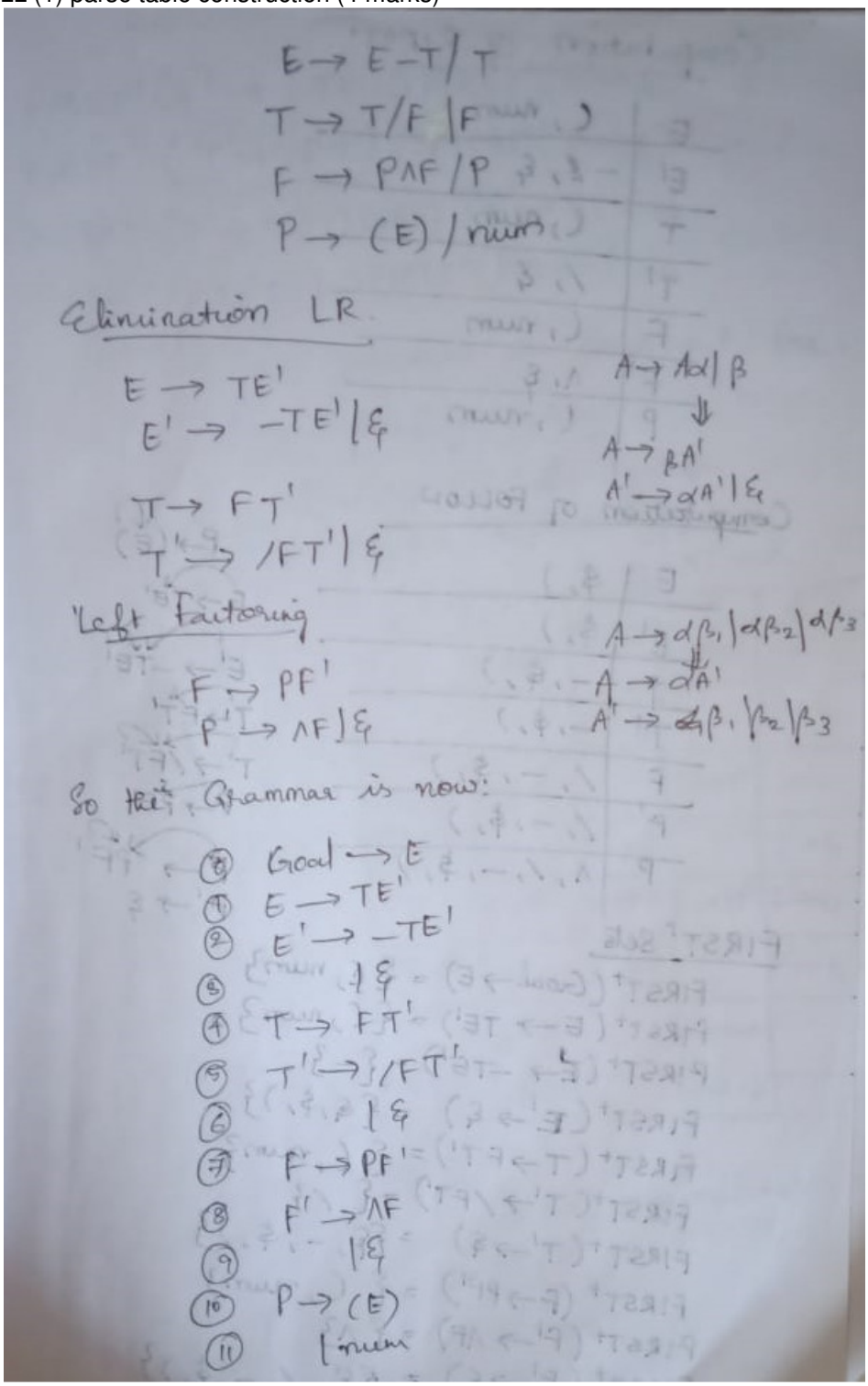
11. Explanation of scanner, parser, elaborator, optimizer, instruction selector, instruction scheduler, and register allocator with a proper example.

12.

Left recursion elimination (2 marks)

First & Follow identification (4 marks)

LL (1) parse table construction (4 marks)



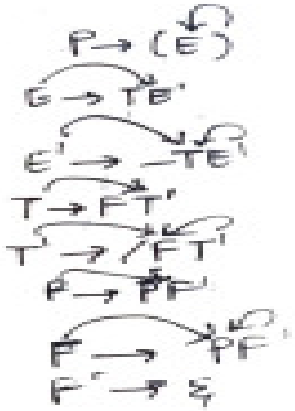
Computation of FIRST:

Computation of FIRST

E	(, num
E'	−, ε
T	(, num
T'	/, ε
F	(, num
F'	^, ε
P	(, num

Computation of FOLLOW

E	$,)
E'	$,)
T	−, $,)
T'	−, $,)
F	/, −, $,)
F'	/, −, $,)
P	^, /, −, $,)



FIRST+ Sets

FIRST+ (Goal → E) = {(, num}
FIRST+ (E → TE') = {(, num}
FIRST+ (E' → −TE') = {−}
FIRST+ (E' → ε) = {ε, $,)}
FIRST+ (T → FT') = {(, num}
FIRST+ (T' → /FT') = {/}
FIRST+ (T' → ε) = {ε, −, $,)}
FIRST+ (F → PF') = {(, num}
FIRST+ (F' → ^F) = {^}
FIRST+ (F' → ε) = {ε, /, −, $,)}
FIRST+ (P → (E)) = {(}
FIRST+ (P → num) = {num}

$FIRST^+(E' \rightarrow -TE') \cap FIRST^+(E' \rightarrow \epsilon) = \phi$

$FIRST^+(T' \rightarrow /FT') \cap FIRST^+(T' \rightarrow \epsilon) = \phi$

$FIRST^+(F' \rightarrow \wedge F) \cap FIRST^+(F' \rightarrow \epsilon) = \phi$

$FIRST^+(P \rightarrow (E)) \cap FIRST^+(P \rightarrow num) = \phi$

So, the grammar is backtrack-free.

LL(1) Parsing table Construction

Table[NT, T] = P_i

if $T_j \in$ or $T \in FIRST^+(P_i)$

$P_i = NT \rightarrow \alpha$

	−	/	∧	(	)	num	$
E				$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow -TE'$				$E' \rightarrow \epsilon$		$E' \rightarrow \epsilon$
T				$T \rightarrow PT'$		$T \rightarrow PT'$	
T'	$T' \rightarrow \epsilon$	$T' \rightarrow /FT'$			$T' \rightarrow \epsilon$		$T' \rightarrow \epsilon$
F				$F \rightarrow PF'$		$F \rightarrow PF'$	
F'	$F' \rightarrow \epsilon$	$F' \rightarrow \epsilon$	$F' \rightarrow \wedge F$		$F' \rightarrow \epsilon$		$F' \rightarrow \epsilon$
P				$P \rightarrow (E)$		$P \rightarrow num$	
Goal				Goal → E		Goal → E	

13. a) (5 marks)

```
%{
#include<stdio.h>
%}
%%
bool|int|float printf("Keyword");
[-+/*%] printf("operators");
[,.'"]+ printf("Punctuation Chars");
[!@#$%^&*()]+ printf("Special Chars");
[a-zA-Z]+ printf("Identifiers");
%%
main(){yylex();}
```

b) (5 marks)

Pseudo code for S_{init}, S_0, S_1, S_2 and S_{out}