Name: Alishba Zafar
Regno: L1F22BSCS0287

1. Grammar

- **program** -> KHALI SIR '(' ')' block
- **block** -> '{' stmt_list '}'
- **stmt_list** -> stmt stmt_list | empty
- **stmt** -> declaration | assignment | conditional | loop | io_stmt | jump_stmt | block | ';' | error ';'
- **declaration** -> type var_list ';'
- **type** -> GINO | TARO | HARF
- **var_list** -> var_init | var_init ',' var_list
- **var_init** -> ID | ID '=' expression
- **assignment** -> ID '=' expression ';'
- **conditional** -> JE '(' expression ')' stmt | JE '(' expression ')' stmt NAHIN stmt
- **loop** -> TAK '(' expression ')' stmt | DORA '(' dora_init expression ';' dora_inc ')' stmt
- **io_stmt** -> DEKHA '(' print_list ')' ';'
- **expression** -> simple_expr | simple_expr rel_op simple_expr
- **simple_expr** -> term | term '+' term
- **term** -> ID | NUMBER | CHAR_LIT | THEEK | GHALAT
- **rel_op** -> EQ | '<' | '>'

2. FIRST and FOLLOW Sets

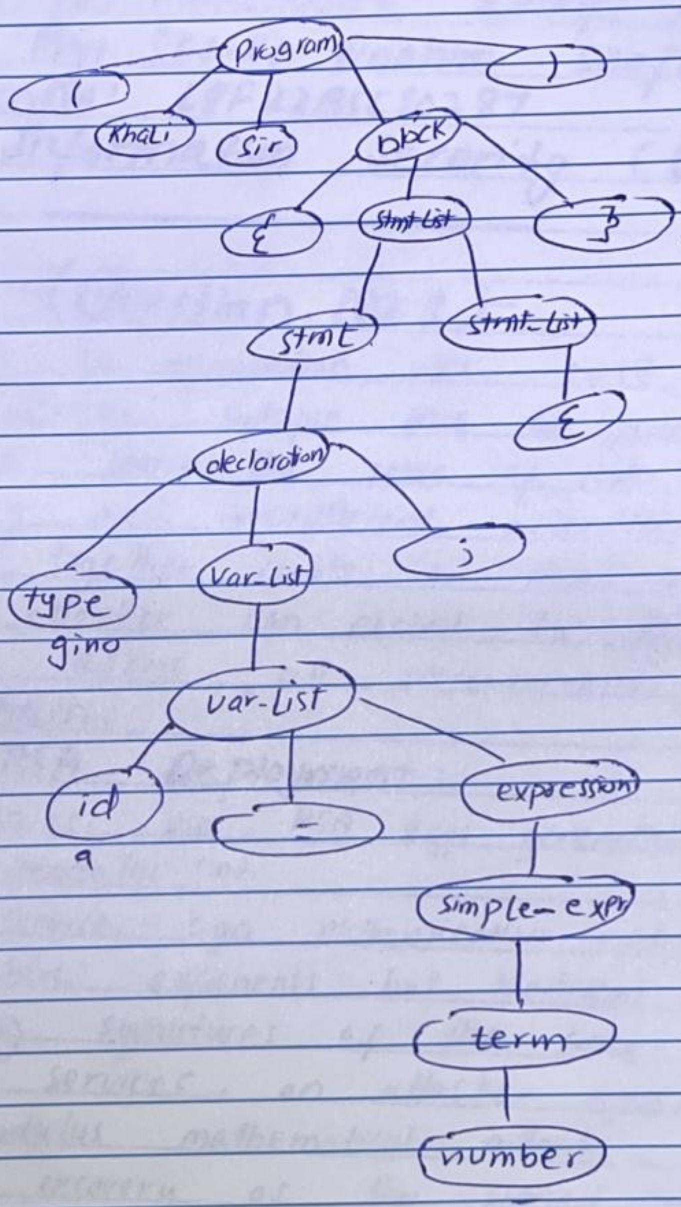
Non-Terminal	FIRST Set	FOLLOW Set
type	{ GINO, TARO, HARF }	{ ID }
stmt	{ GINO, TARO, HARF, ID, JE, TAK, DORA, DEKHA, WAPSI, RUK, CHALLO, '{', ';', error }	{ GINO, TARO, HARF, ID, JE, TAK, DORA, DEKHA, WAPSI, RUK, CHALLO, '{', ';', '}', NAHIN }

3. Parse Tree Structure

Sample Input: `KHALI SIR () { GINO a = 5; }`

No:

Date:


Program
(
Khali
Sir
block
)
{
stmt-List
}
stmt
stmt-List
ε
declaration
;
type
gino
var-List
var-List
id
a
=
expression
simple-expr
term
number
5

4. Explanation: Use of Phase 01 Tokens

Phase 01 (Lexical Analysis) provides the tokens that the Parser uses to verify syntax:

- **Token Identification**: The lexer identifies specific strings as tokens such as `GINO` (integer type) or `JE` (conditional).
- **Data Flow**: Tokens like `ID` and `NUMBER` carry their actual values into the parser's `yylval`, allowing the parser to identify which variable is being declared.
  +1
- **Structural Triggers**: The parser uses tokens like `SIR`, `(`, and `)` to verify that the program follows the mandatory `KHALI SIR ()` entry point structure.
- **Error Reporting**: When the lexer passes a token that does not fit the grammar rules (e.g., a `>` appearing where a `;` was expected), the parser triggers the `yyerror` function.
- **File Logging**: The `yyerror` function uses the current `yytext` (the token that caused the error) and `yylineno` to write a descriptive error message to `e.txt`.