Compiler Construction → Project Phase 2

Objective:

The objective of Phase 02 is to design and implement a **syntax analyzer** for the **same custom language** created by the student in Phase 01. The parser must validate the grammatical structure of programs produced using the **student's own tokens, keywords, operators, and punctuations** defined earlier.
This phase emphasizes understanding of **formal grammars, parsing techniques, and compiler front end integration**, following classical compiler design principles.

Mandatory Phase 01 Dependency

Students must **reuse their Phase 01 lexical analyzer** without changing previously defined tokens. Phase 02 will be evaluated only if

- **Phase 01 keywords are used**
- **Phase 01 operators and punctuations appear in grammar**
- **Phase 01 test language is extended, not replaced**

Any mismatch between scanner and parser will result in zero marks.

Project Description

Each student will implement a **parser** using **YACC Bison** , based on the grammar of their **own designed language**.

1. Grammar Design

Each student must design a **context free grammar** that supports at least the following constructs using **their own keywords**.
Mandatory constructs

- Program start and end
- Variable declaration
- Assignment statement
- Conditional statement
- Loop statement
- Input or output statement

Example constructs only for understanding
If your keyword for if is Agar
If your keyword for loop is JabTak
If your keyword for return is Wapas
The grammar **must differ from other students** by

- Different keywords
- Different statement structure
- Different block symbols or terminators

2. Syntax Rules to Support

Each student must support at least **five statement types**, chosen and named by the student.
Examples
• Declaration statement
• Assignment statement
• Conditional statement
• Loop statement
• Output statement
Each statement must have a **clear grammar rule** written by the student.

3. Parser Implementation

Students must
• Connect Phase 01 scanner with Phase 02 parser
• Use token values from lexical analyzer
• Validate syntax of complete programs
On successful parsing
Print “Syntax analysis successful”
On syntax error
Print Line number, Expected token and Found token

4. Error Handling Requirement

Each parser must report at least
• One missing token error
• One invalid statement structure error
Error message must include
• Line number
• Nature of error

5. Documentation Requirements

Students must submit a PDF containing
• Grammar CFG
• FIRST and FOLLOW of at least two non terminals
• Parse tree of one valid program
• Explanation of how Phase 01 tokens are used
Hand drawn parse tree is allowed and encouraged.

6. Deliverables

1. Document PDF
   Grammar
   FIRST FOLLOW
   Parse tree
   Explanation
2. Source Code
   parser.y or equivalent Integrated scanner.l

3. Test Programs
   One valid program
   One invalid program
4. Output Files
   Successful parse output
   Syntax error output
5. Demo Viva
   Live parsing
   Explanation of grammar rule
   Error demonstration

Sample Production Rules for Guidance Only

The following example shows how production rules may look in a custom student designed language.
Keywords and structure are only for understanding. Students must replace them with their own.

Example 1 Program Structure

Assume a student language starts with a program keyword and ends with a terminator.
Sample idea
Program starts with Shuru
Program ends with Khatam
Sample production rules
Program → START stmt_list END
START → SHURU
END → KHATAM
stmt_list → stmt stmt_list
stmt_list → stmt
Explanation
A program begins, contains one or more statements, and then ends.

Do same for Variable Declaration, Assignment Statement, Conditional Statement and Output Statement

Step 1 Token Sharing from Phase 01

Students must ensure that
Tokens defined in Flex scanner.l
Are declared in parser.y using %token
Example
%token ID NUMBER
%token ADADI AGAR WARNA
%token RAKHDO KARO
Tokens names must exactly match Phase 01.

Step 2 Grammar Section in YACC

Grammar rules go between %% symbols.
Example structure

```
%%
program : START stmt_list END
;
stmt_list : stmt stmt_list
| stmt
;
%%
```

Each student must write their own rules.

Step 3 Error Handling in YACC

Students must implement error reporting using
yyerror function
They should print
Line number
Nature of error
This verifies understanding during viva.

Step 4 Compilation Flow

Students must demonstrate

```
flex scanner.l
yacc parser.y
gcc lex.yy.c y.tab.c
```

Running the executable on their own test program.

Important Instruction to Students

The example is only structural guidance.
Direct copying of grammar, rule names, or structure will result in rejection during viva.
Each student language must reflect their own design choices from Phase 01.