# Compilers-II Project Lumiere Language

Syntax Analysis Phase Introduction and review

# Team Members

1. Nigam Niraula
2. Shaswot Paudel
3. Bikraj Shrestha
4. Gaurav Choudekar
5. Srimannarayana
6. Dheeraj
7. Armaan Shaik
8. Shivendraraje Gadekar

# Overview of parser

- Defining Parsing :
  - Converting a stream of tokens into an Abstract Syntax Tree (AST).
  - Ensures the code is following rules mentioned by grammar
- Structure of a parser:
  - Tokens: Provided by the lexer from previous Assignment (e.g., IF, ADD_OP, VAR).
  - Grammar Rules: Defines the structure of the language.
  - Actions: C code that executes when a rule is matched.

# Components of the parser

- Rules:
  - To Specify the structure of the language.
  - Defines how tokens can combine to form valid statements.
- Actions:
  - Code executes  when a production rule matches to given statement.
  - Updates the parser state or builds parts of the syntax tree.
- Precedence and Associativity:
  - Ensures correct parsing of operators (e.g., +, *).
  - Defined using %left, %right.

# Parsing Expressions..

- Arithmetic Expressions:
  - Parses expressions involving operators like +, -, *, /, ^.
  - Supports operator precedence and associativity.
- Boolean Expressions:
  - Parses boolean expressions with relational and logical operators.

```
ARITHMETIC_EXP : ARITHMETIC_EXP ADD_OP  MUL_EXP  {printf("ARITHMETIC_EXP");}
               | ARITHMETIC_EXP SUB_OP MUL_EXP
               | MUL_EXP
                   ;
```

# Compound and Simple statement Parsing

- Compound Statement Parsing:
  - Allows multiple statements inside blocks {}.
  - Supports grouping multiple statements.
- Simple statements :
  - Handles basic single-line statements like assignments, expressions.

```
CMPND_STATEMENT : CMPND_STATEMENT STATEMENT
                |
                ;
```

```
STATEMENT: CONDITIONAL_STATEMENT
         | EXPRESSION EOL{printf("Statement here\n");}
         | LOOP_STATEMENT
         | PREPROCESSOR_DECLERATION
```

# Handling Declarations

- Variable Declarations:
  - Declares variables with or without initialization.
- Array Declarations:
  - Declares arrays and allows initializing them with values.
- Type Support:
  - Handles basic types like int, float, char, boolean, and more.

```
DECLARATION : TYPE var_list { printf("here");}
            ;
```

```
| VAR LEFT_BRACE INTEGER_VALUE RIGHT_BRACE {printf("Array DECLARATION\n");}  // Array DECLARATION
| VAR LEFT_BRACE INTEGER_VALUE RIGHT_BRACE COMMA var_list {printf("Array and Variables DECLARATION\n");}
```

# Conditional Statements Parsing

- If-Else Statements:
  - Supports simple if, else if, and else conditions.
  - Can nest and chain multiple conditions.
- Else-If Handling:
  - Allows complex decision-making with multiple else if conditions.

```
IF_STATEMENT:IF LEFT_PAREN BOOLEAN_EXP RIGHT_PAREN LEFT_CURLY_BRACE CMPND_STATEMENT RIGHT_CURLY_BRACE ;

ELSE_STATEMENT: ELSE LEFT_CURLY_BRACE CMPND_STATEMENT RIGHT_CURLY_BRACE;
```

```
ELSEIF_STATEMENT: ELSE_IF LEFT_PAREN BOOLEAN_EXP RIGHT_PAREN LEFT_CURLY_BRACE CMPND_STATEMENT RIGHT_CURLY_BRACE;
```

# Loops in Parser

- Custom Loop Parsing (CHECK_UNTIL):
  - Implements a loop construct similar to a for loop.
  - Initial condition, condition checking, and loop body are parsed.
- Initial Conditions in Loops:
  - Can have multiple conditions or an empty loop body.

```
LOOP_STATEMENT  : CHECK_UNTIL LEFT_PAREN INITIAL_CONDITION RIGHT_PAREN LEFT_CURLY_BRACE CMPND
```

```
INITIAL_CONDITION :STATEMENT STATEMENT EXPRESSION  { printf("INIT_CONDITION\n");}
                  | STATEMENT STATEMENT {printf(" ; ; ");}
                  | EXPRESSION {printf("empty CONDITION");}
```

# Classes and OOPs aspects

- Class Definition Parsing:
  - Supports class declarations with inheritance.
- Public/Hidden Access:
  - Defines the scope of class members using access modifiers.

```
CLASS_STATEMENT: CLASS VAR CLASS_DEFINITION
CLASS_BODY: HIDDEN_STAT PUBLIC_STAT
```

# Error Handling

- Error Detection:
  - Catches syntax errors during parsing.
  - Error messages are printed via the yyerror function.
- Example Error:
  - Incorrect placement of an operator or missing semicolon triggers an error.
  - Helps in debugging incorrect code syntax.

```
void yyerror(char *s) {
    printf("Error: %s\n", s);
}
```

## Compilers-II Language Development Project

# Thank you !!!