# Mini Compiler Project Report

Name: Prabsharan Singh Randhawa
Student Number: 239610260

**1. Regular Grammar (Lexical Rules)**

Identifiers: ID → Letter (Letter | Digit | _ )*

Integer Literal: INT → Digit+

Real Literal: REAL → Digit+ . Digit+

Assignment: ASSIGN → ':='

Operators: PLUS → '+'; MINUS → '-'; MUL → '*'; DIV → '/'; POW → '^'

Punctuation: COLON → ':'; SEMICOLON → ';'; COMMA → ','; LPAREN → '('; RPAREN → ')'

**2. Context-Free Grammar (Syntax)**

statement → declaration | assignment

declaration → idList : typeName ;

idList → ID | ID , idList

typeName → integer | real

assignment → ID := expression ;

expression → term ((+|-) term)*

term → factor ((*|/) factor)*

factor → power (^ factor)*

power → ID | INTEGER_LITERAL | REAL_LITERAL | ( expression )

**3. DFA Description for Lexical Analyzer**

• Start → Identifier state on Letter/_

• Identifier state → stays on Letter/Digit/_

• Start → Integer state on Digit

• Integer state → stays on Digit, or goes to Real state on '.'

• Real state → stays on Digit

• ':' → Colon state, or if followed by '=' → Assignment state

• Operators and punctuation each have their own accepting states.

**4. Recursive Descent Parser Functions**

```
void statement() { ... }
void declaration() { ... }
void idList() { ... }
void typeName() { ... }
void assignment() { ... }
void expression() { ... }
void term() { ... }
void factor() { ... }
void power() { ... }
```

**5. Source Code**

```
<Source code omitted here for preview, but included separately.>
```

**6. Example Outputs**

Valid Expressions (All Successful)

a: integer; → Compilation Successful

x: real; → Compilation Successful

a := 3 + 5 * 2; → Compilation Successful

b := (3 + c) * 7 - 5; → Compilation Successful

x := 2 ^ 3 ^ 2; → Compilation Successful

Invalid Expressions (All Errors)

2a := 5; → Error: expected ID, found INTEGER_LITERAL

a := (3 + 5; → Error: expected RPAREN

a := 3 + * 7; → Error: unexpected token

a, b integer; → Error: expected ':'

a := 3 + (b - ); → Error: unexpected token