



**FAST National University of Computer and Emerging Sciences,
Karachi Campus**

Compiler Construction (CS-4031)

Fall 2025

Assignment – 1: (CLO - 1)

Submission Deadline: 31st October 2025

Max Marks: 125

General Instructions:

- Your work should be original, not copied from internet or any other resource.
- Assignment should be submitted before due date.

1. Context-Free Language (CFL)

1. Design a CFL for arithmetic expressions supporting multiple operators (+, -, *, /), nested parentheses, and operator precedence.
 - a) Show both leftmost and rightmost derivations for the expression:
a * (b + c / d) - e.
 - b) Comment on ambiguity and propose a modification to eliminate it.
2. Prove that the language $L = \{ a^n b^m c^n \mid n, m \geq 1 \}$ is context-free but not regular.
Construct a pushdown automaton (PDA) equivalent to your grammar and justify the result using the Pumping Lemma for regular languages.
3. Create a CFL for a conditional construct that supports if–elif–else with relational operators (>, <, ==, !=).
Demonstrate the derivation of the following input:
if(a>0) x=1 elif(a==0) x=2 else x=3.
4. Design a CFL to represent recursive function calls (e.g., function calling itself directly or indirectly).
 - a) Give a complete derivation for:
main(sum(add(a, b))).
 - b) Explain how recursion depth can be represented through grammar rules.
5. Compare and contrast two grammars defining arithmetic syntax: one that requires explicit semicolons and another that infers statement termination implicitly. Discuss which design is more suitable for compiler parsing and error recovery.

2. Context-Free Grammar (CFG)

1. Construct a CFG for palindromic strings of the form **ww^R over {a, b, c}**. Show a complete derivation and parse tree for: **abccba**.



**FAST National University of Computer and Emerging Sciences,
Karachi Campus**

Compiler Construction (CS-4031)

Fall 2025

2. Design a CFG for compound assignment statements and expressions with nested operations.
 - a) Include constructs like `+=`, `-=`, `*=`, `/=`.
 - b) Show derivation for: `x += y * (z - w);`
3. Create a CFG for nested control structures that combine while and if-else statements.
 - a) Derive a string showing two levels of nesting, such as:

```
while(a>0){  
    if(b<5) b=b+1;  
}
```

4. Discuss ambiguity in a grammar for arithmetic expressions that include unary minus and binary operators. Propose an unambiguous grammar that enforces correct operator precedence and associativity.
5. Construct a CFG for a **switch-case-default** construct similar to C/C++.
 - a) Provide a derivation for:

```
switch(x){  
    case 1: y=10;  
    case 2: y=20;  
    default: y=0;  
}
```

3. Lexical Analyzer

1. Write regular expressions for identifiers, integers, floating-point numbers, keywords, and comments in a C-like language. Test your rules on the input:
`float _x1 = 10.5; // initialize variable.`

2. Develop a complete token specification for a language fragment supporting for, while, break, arithmetic operators, and relational operators.

Simulate the lexical analysis for this: `for(i=0; i<=10; i++) break;`

3. Given the code:

```
if(x<=y && y!=0)  
    z = x/y + 1;
```

- a) Identify and classify all tokens by type, lexeme, and category.
 - b) Show how the lexer differentiates between keywords and identifiers.
4. Discuss the design of a lexical analyzer for C identifiers and reserved keywords, explaining:
 - a) How invalid tokens and keywords are detected.
 - b) The role of finite automata (FA) in recognizing identifiers and numbers.



**FAST National University of Computer and Emerging Sciences,
Karachi Campus**

Compiler Construction (CS-4031)

Fall 2025

5. Extend the Python lexical structure to introduce a custom operator @@ for string concatenation and a new keyword repeat. Illustrate how both would be added to the lexical analyzer and token table.

4. Symbol Table Manager

1. Design a **symbol table** structure for a language supporting variables, arrays, and functions with **nested scopes**.
 - ◆ Demonstrate insertion, lookup, and deletion operations during block entry and exit.
2. Explain the behavior of the symbol table during parsing of this program:

```
int main(){
    int x=5;
    {
        float x=2.5;
        int y=x+1;
    }
}
```

 - Show how shadowing and scope resolution are handled.
3. Create a scenario where the lexical analyzer updates the symbol table, causing a conflict or duplicate entry. Describe how such conflicts are resolved.
4. Evaluate symbol table designs using hash tables vs linked lists for large programs. Discuss efficiency.
5. Design a symbol table schema for an object-oriented mini language supporting classes, inheritance, and method overloading.
 - Provide example entries for a Base and a Derived class, showing attribute and method information.

5. Practical – FLEX / Lex Implementation

1. Write a .l file for identifiers, numbers, and basic operators. Test with x1 + 42.
2. Compile .l file using flex and gcc. Show step-by-step commands and outputs.
3. Run your lexer on sample inputs (if x>0 else y=2) and show token outputs.
4. Modify the lexer to recognize a new keyword repeat. Demonstrate outputs.
5. Evaluate your lexer's handling of invalid inputs (123abc). Suggest improvements.