National University of Computer and Emerging Sciences

FAST School of Computing Spring 2025

CS4031-Compiler Construction

Assignment 01

Instructions for Submission:

Dear students, we will use auto-grading tools, so failure to follow the submission format below will result in zero marks.

1. Assignments must be completed in groups of 2 students. Groups cannot be changed later.
2. Combine all your work into one folder and name it as follows:
3. RollNumber1-RollNumber2-Section.zip. (For example: 22i1234-22i5678-A.zip)
4. Run and test your program on any machine before submission.
5. Submit the .ZIP file on Google Classroom before the deadline. Submissions through other means (e.g., email) will not be accepted.
6. Check your ZIP file for issues like corruption, viruses, or mistakenly including .exe files. If the file cannot be downloaded, you will receive zero marks.
7. Ensure displayed output is clear and well-presented. Use appropriate comments and indentation in your code.
8. Total Marks: 100.
9. Syntax errors in the code will result in zero marks for the corresponding part of the assignment.
10. Your code must be generic and reusable.
11. You cannot use built-in functions or import any libraries which are specialized in compiler construction functionality.
12. Submit your assignment 3 hours before the deadline to avoid last-minute issues (e.g., internet problems).
13. You may only use Java programming language.
14. Each assignment includes a demo and viva. Completing the assignment does not guarantee full marks. Poor viva performance can result in significant mark deductions or zero marks.

Deadline:

The deadline to submit the assignment is as per GCR. Submit your assignment on Google Classroom (Classroom Tab, not Lab). Only .ZIP files are acceptable; other formats will receive zero marks. Timely submission is your responsibility, and no exceptions will be made. No late submissions are allowed as per the policy mentioned in course outline.

Tip:

Start early to ensure timely completion of the assignment.

Plagiarism:

Plagiarism is strictly prohibited. If plagiarism is detected, you will receive zero marks. Copying from the internet is the easiest way to get caught.

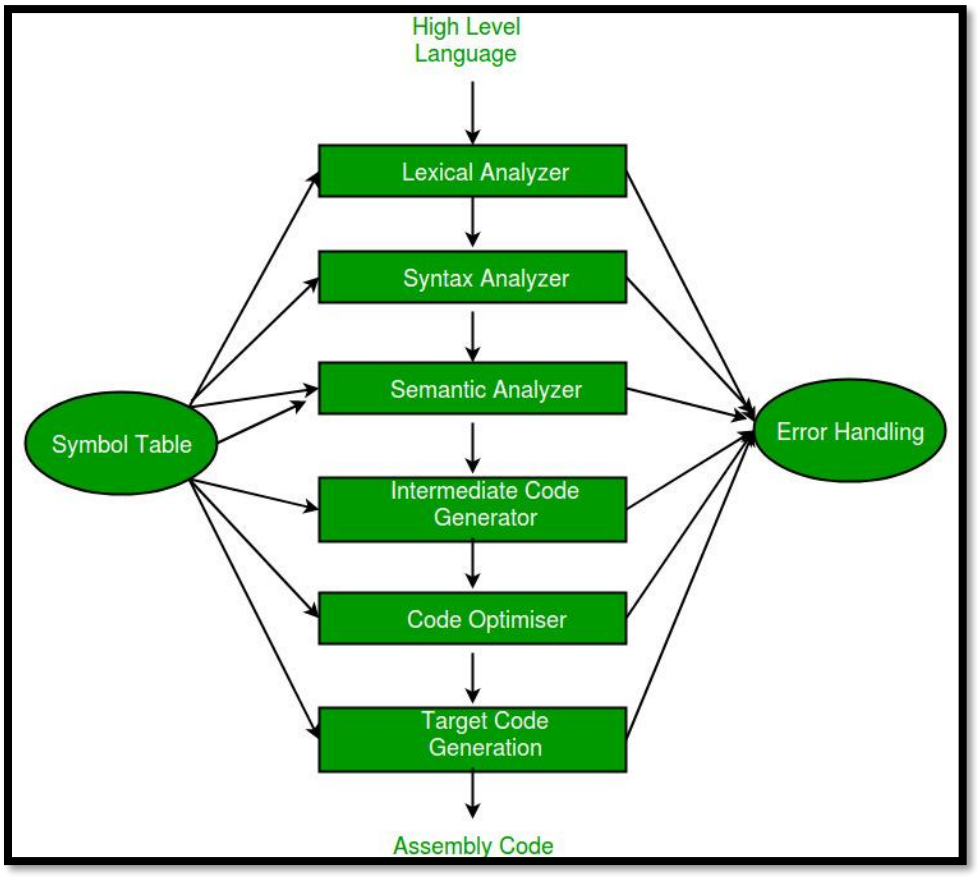
Note:

Follow these instructions exactly. Failure to comply will result in zero marks.

Introduction:

Compiler construction is about building tools called compilers, which convert high-level programming languages into machine code that a computer can understand and run. It combines both theory and practical application and has uses in many fields.

Each assignment will build on the previous one, so make sure to complete your work carefully and handle each part properly.



The overview above explains, in simple terms, how a compiler works. In each assignment, you will implement one or two parts of the process, making updates to the earlier parts if needed.

Resources:

- https://www.geeksforgeeks.org/introduction-of-compiler-design/
- https://medium.com/@Saman-Mahmood/symbol-table-in-compiler-construction-13970a0025d8
- https://medium.com/@Saman-Mahmood/phases-of-a-compiler-04a00753cddb
- https://medium.com/@mahrukhrizvi2/lexical-analysis-3449e7c2b430

Task:

For the first assignment, your team will work on both theoretical and practical tasks to create your own programming language. You can define your own keywords, identifiers, rules, and restrictions. You will implement the following:

1. **RE, NFA, and DFA**: Before implementing, you must first create a workflow for the NFA, Regular Expression, and DFA classes. Your program should be able to display the total number of states. You may be asked to display the unique states for each parse. You should opt to show a transition state table which would show your work more clearly.
2. **Lexical Analyzer**: This is the first phase of the compiler. It reads the source code, breaks it into meaningful elements called tokens, and prepares them for the next phases of compilation.
    - Number of tokens
    - Pre-processing
    - Case sensitivity
3. **Symbol Table**: A tool to track all variables, functions, and identifiers in your program. It will store details like names, types, and memory locations to help with error checking, scope management, and optimization. For now, handle these entries:
    - Datatypes (At least 4)
    - Input/Output
    - Strings
    - Comments
        1. Single
        2. Multiline
    - Constants
        1. Global
        2. Local
    - Arithmetic operations
4. **Error Handler**: A class to identify rule violations, showing the line where the error occurs.

- Each part will be updated in future iterations, so build carefully.
- You have the option to make a proper GUI or simply do a command line interface.
- Make sure to define a unique file type.
- Make sure to make a readme for all your keywords and rules. This will serve as a user manual for the TA to understand your language and use it to implement a program. Clarity is needed in case of missing information marks will be deducted in that section of the rubric.
- You will be asked to change code during the demo and will be asked for comprehensive viva.

Bonus:

- Upload on Git Hub and add TA as a collaborator. Make sure to add all members even if they didn't do any work. Before deadline.

Constraints:

- Your system should support four types of data: **true/false values (Boolean), whole numbers (Integer), decimal numbers (Decimal), and single letters (Character).**
- It should only recognize **lowercase letters (a to z)** as valid variable names or identifiers.
- The system should be able to perform **basic arithmetic operations** like addition (+), subtraction (-), multiplication (*), division (/), and remainder/modulus (%).
- Decimal numbers should be **accurate up to five decimal places**, following standard rounding rules.
- The system must also **support exponents** (raising numbers to a power) for both **whole numbers and decimal numbers**.
- It should be able to **ignore extra spaces** between elements and **handle multi-line comments** properly, even in tricky situations.
- Should have support for globals and local variables.
- User would be able to see the transition state table.