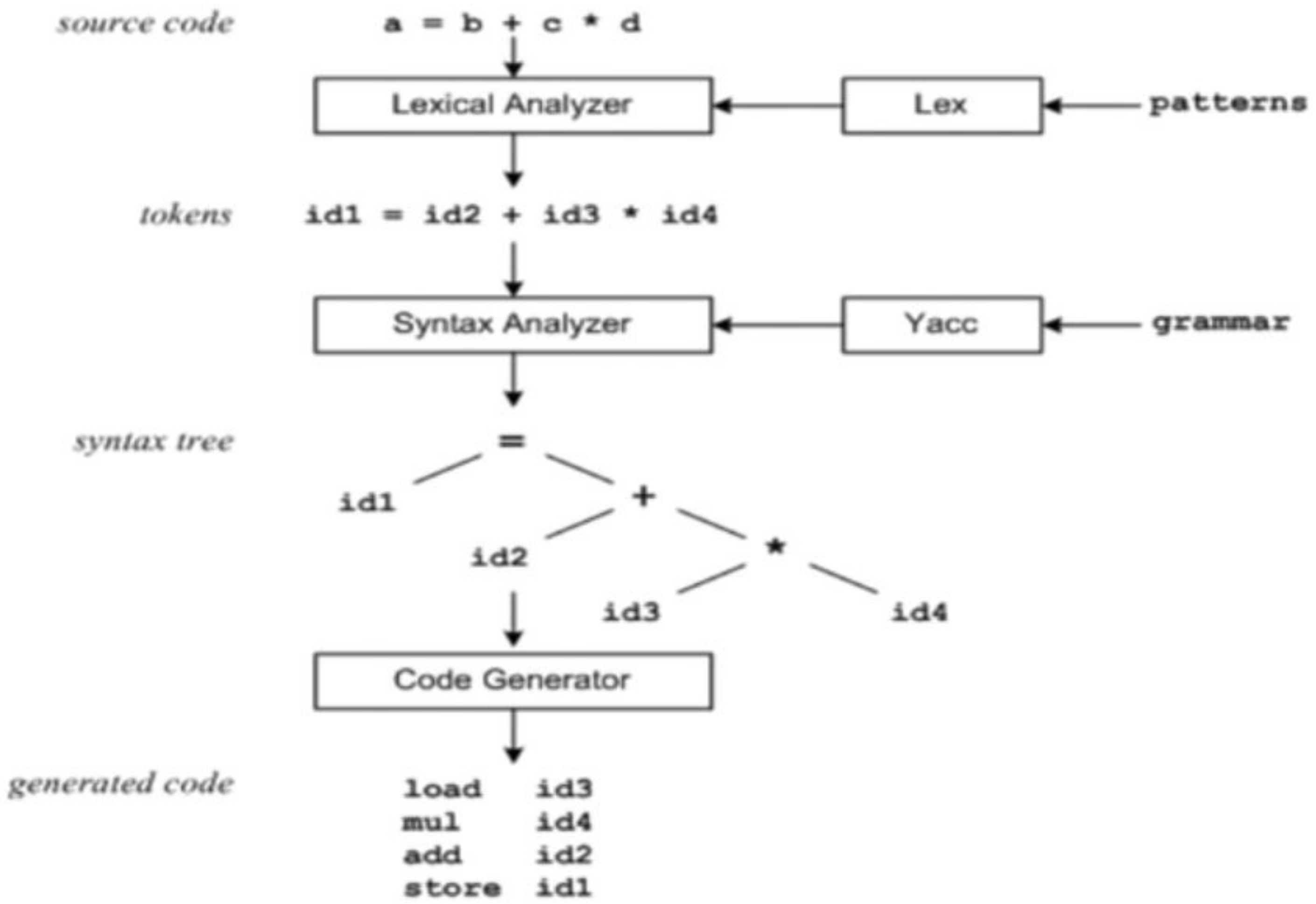
Project Step 1: Lexical Analyzer

Programming Languages

Due: March 9, 2026, 23:59

Project Description

- In this project, you will design your own programming language
- Step 1 will be on lexical analysis and you will write a lex file
- Step 2 will be on parsing and you will write a yacc file



```
load   id3
mul    id4
add    id2
store  id1
```

Lexical analyzer steps

- PART A. Design of your own language
- PART B. Write the Lexical Analyzer
- PART C. Write Example Programs in your language to check whether you can get the correct tokens

PART A: Designing the language

- Give a name and design your language by preparing the complete grammar in BNF form. (This will also help you very much in yacc part.)
- Your language must allow a good set of language constructs such as:
    - Terminal values like integers, floating numbers, characters, text, true/false, etc.
    - Defining constants and variables
    - Functions/methods
    - A few operators
    - Conditional statements: if-then, if-then-else, or if-elseif-elseif-...-else
    - Looping (probably like a while loop)
    - IO (print and scan) statements
    - Commenting should be possible

Design

- You are free to design your own language.
- It can be for a specific aim
- It can be imperative/functional/logic etc.
- Or you can think of specific-purpose programming languages, for example a "fuzzy" programming language where you can define fuzzy variables. So that you can define `x=[3,7,uniform]` means a variable that is a fuzzy variable with a specific distribution (any value between 3 and 7), and you can define another one `y=[4,7,uniform]` and then you can have a function `isPossible(x+y,13)` that will return true because x+y can be 13 but `isPossible(x*y,3)` will return false because x*y cannot be 3.
- Writing a smart contract programming language like solidity is also a good idea

PART B: Lexical Analyzer

- Here you will write a lex file `newlang.l` which will take a source file and return a new file with the tokens.
- Since you will write the parser later, your lexical analyzer is supposed to write the tokens on the screen (See the example youtube video[2])
- For example if you have a line like
  - `raining=false;`
- Your output will be something like
  - `VARIABLE EQUALS BOOLEAN SEMICOLON`

PART C: Examples

- You will write example programs in your language.
- You are supposed to demonstrate **all** the constructs in your language with this program.
- Your programs must contain comments explaining what is being done.
- We should be able to understand your syntax and play with the code and still get correct results.

Recommended programming environment

1. Get a linux or mac system. You can do one of the following:
   - Install Ubuntu (or any other Linux distro)
   - Install VMware and install Ubuntu into it
   - Use Windows subsystem to get an Ubuntu shell (this is probably the most convenient) (https://docs.microsoft.com/en-us/windows/wsl/install-win10#install-the-windows-subsystem-for-linux)
   - https://www.freecodecamp.org/news/how-to-install-wsl2-windows-subsystem-for-linux-2-on-windows-10/
2. Install flex and bison using `sudo apt install` ....
3. Install `git`
4. Create a repo (or create one in GitHub and clone it to your local machine)
5. Write your programs… (**Everyone must have a GitHub account and commit separately so that we can see who wrote where, etc.**) If your nickname is not your name, write your name to your GitHub profile so that we can understand who is who.

Submission

- You will submit as a repository in GitHub (You can get the code from https://github.com/muratakcs/pl-lex-starter-kit so that you have a starting point. **Do not fork** because you will need to keep your repo private but forked public repos cannot be turned into private repos). Your repo will contain:
    - A project report (an `README.md` file) that includes:
        - Project group members
        - Name of your programming language
        - Grammar in BNF form
        - explains the syntax of your language
        - Also explain any design decisions you make
    - Your lex file (`plname.l`)
    - And your example programs (`exampleprog1.yourextension`)...
    - A **makefile** file

Submission

- To summarize:
  - Only one of you copies the starter kit, invites other members as collaborator, you collaborate, commit separately
  - once your repo has the following files ready:
  - `README.md`
  - `plname.l`
  - `exampleprog1.yourextension`
  - `Makefile`
- Invite us (https://github.com/muratakcs, https://github.com/ErcxCS) as collaborators (https://help.github.com/en/github/setting-up-and-managing-your-github-user-account/inviting-collaborators-to-a-personal-repository)

Extra resources

- [1] https://github.com/jengelsma/lex-tutorial

- [2] https://www.youtube.com/watch?v=54bo1qaHAfk (using lex without yacc)
- [3] https://www.youtube.com/watch?v=__-wUHG2rfM (This is on yacc but it gives a better idea how to use lex together with yacc)