CSE 420 Lecture-1

Compiler → The mother of all program.

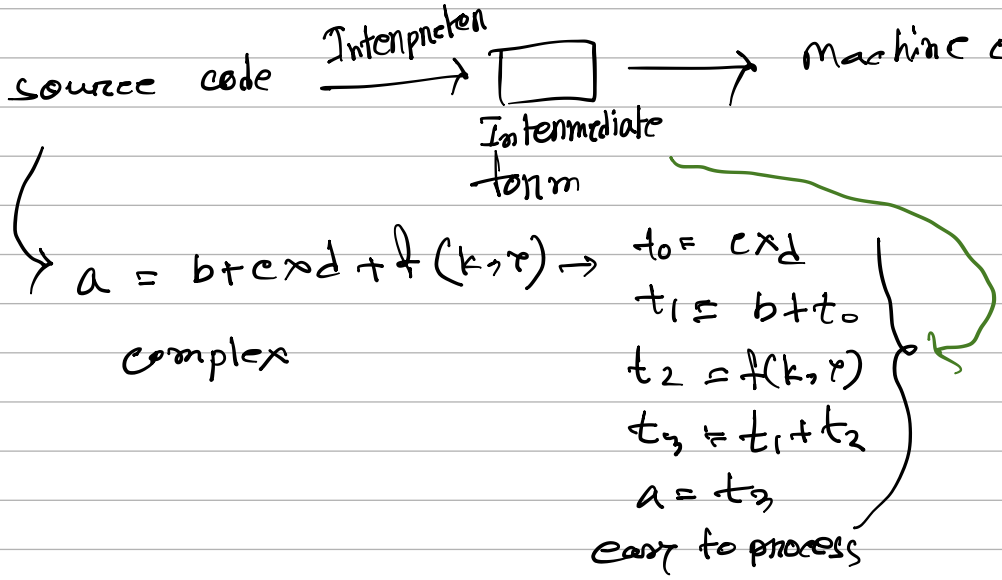
It has to be 100% accurate & bugfree from the beginning. Otherwise all the programs will get crashed.

How did we write the compiler itself?
What compiler compiled the first compiler?

↳ first compiler was written in machine code.

Only high level languages need to be compiled.

What is the difference between compiler & interpreter?



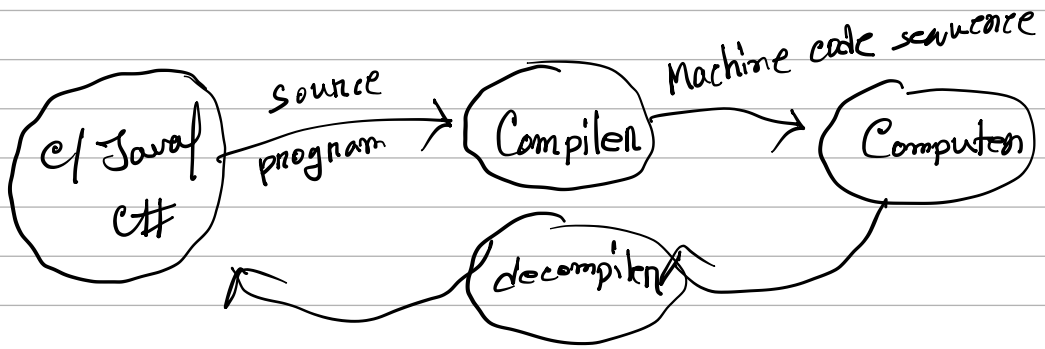
Interpreter is slower. It is more controlled environment.

source code → compiler → Machine code

cuda → special version of C.

↳ only executable by GPU not CPU.

Here compiler plays the role of translator.



✓ words → basic unit of meaning

↓

✓ Grammer → How we arrange the words

↓

✓ Semantics/meaning → Rice ate mice } words have meaning, Grammer is correct but semantic is incorrect

Sequence of characters → tokens

↑ lexical analizer does the task

↓

Syntax checking → Syntax " " " "

↓

Semantic Analysis

↓

Machine code Generation

This should be the case. But sometime there is space for improvement.

a = 3 + 10
b = a + 10 } → a = 13, b = 23

This is done by a intermediate state called.

Machine independent code optimization.

Advance compilers add another step to the end which is "machine dependent code optimization."

Tokens

↓

Syntax checking

↓

Semantic analysis

↓

Machine independent code optimization

↓

Machine code

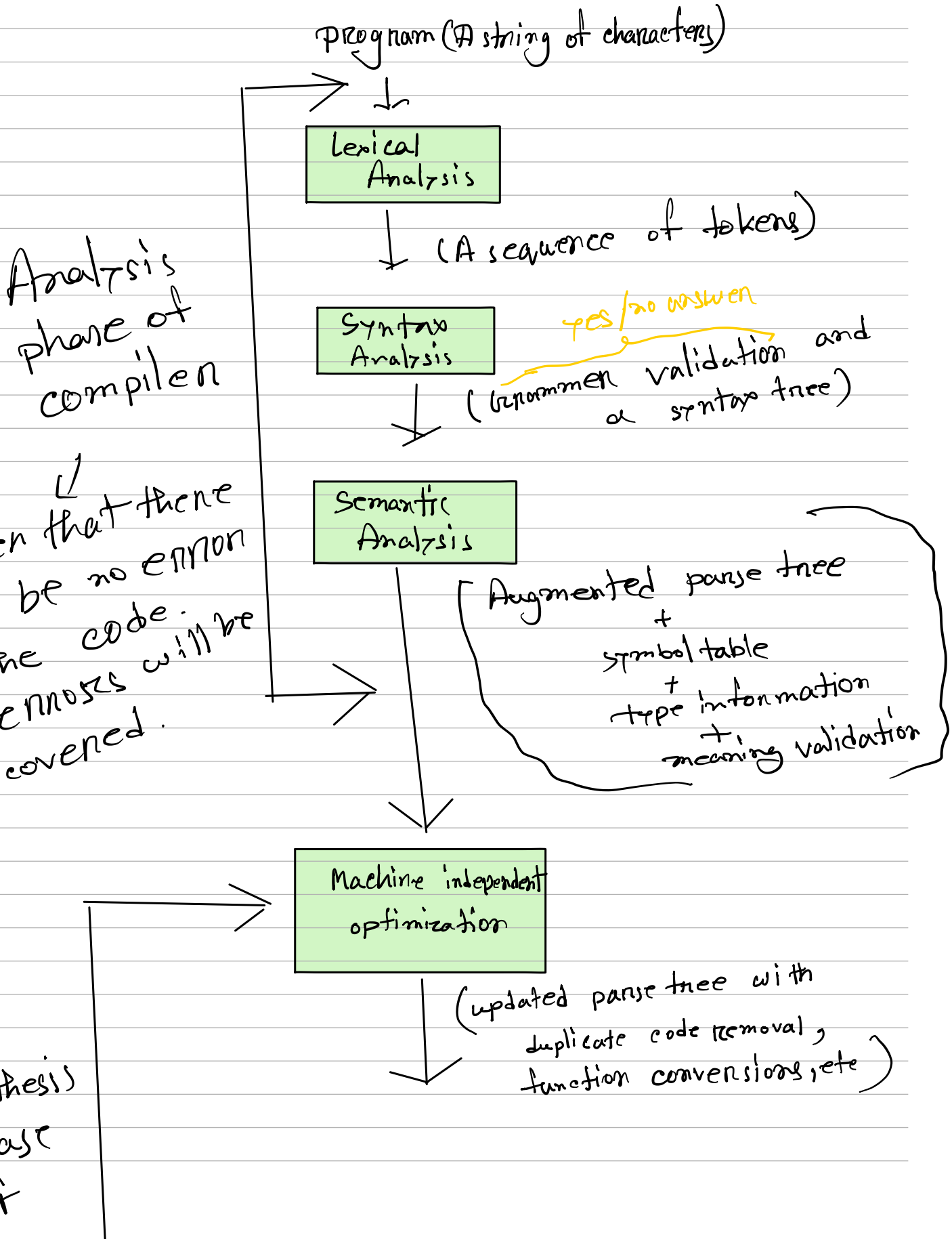
↓

Machine dependent code optimization } depends on the architecture of machine.

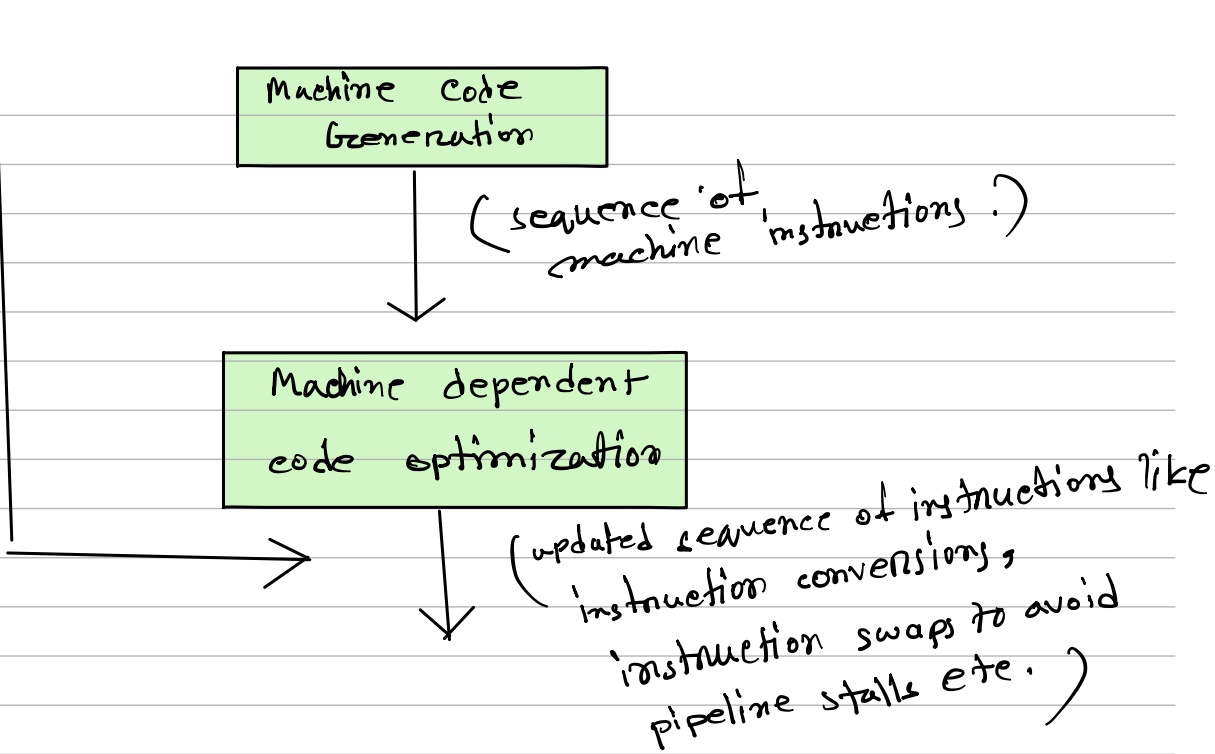
Lecture 2

Regular Expression → DFA

In case of compiler, the translation process is one-way.



the compiler.



Synthesis → breaking down complex steps in simpler steps.

* Token
  - → keyword (if, while, def, for, etc)
  - → identifier (names you give to variables, functions, classes, etc)
  - → numbers (integer, fractional numbers)
  - → Delimeter (; , { ,) , " " , etc)
  - → strings (whole thing inside quotation considered as a token
  - → operators (+ , * , − , / , etc)

* Lexical analyzer does the job of taking sequence of words & characters, and returns the sequence of tokens.

* for each token we have a regular expression.

↓

alphabate $\{a, b\} = \Sigma$

three operations: (i) concatination
(ii) kleen's closure
(iii) OR

* RE of identifier:

$$(\_ \mid [a \ldots z \ A \ldots Z])\ (\ )^*$$

* integer

$$[0 \ldots 9][0 \ldots 9]^*$$

$0 \mid 1 \mid 2 \mid 3 \ldots$

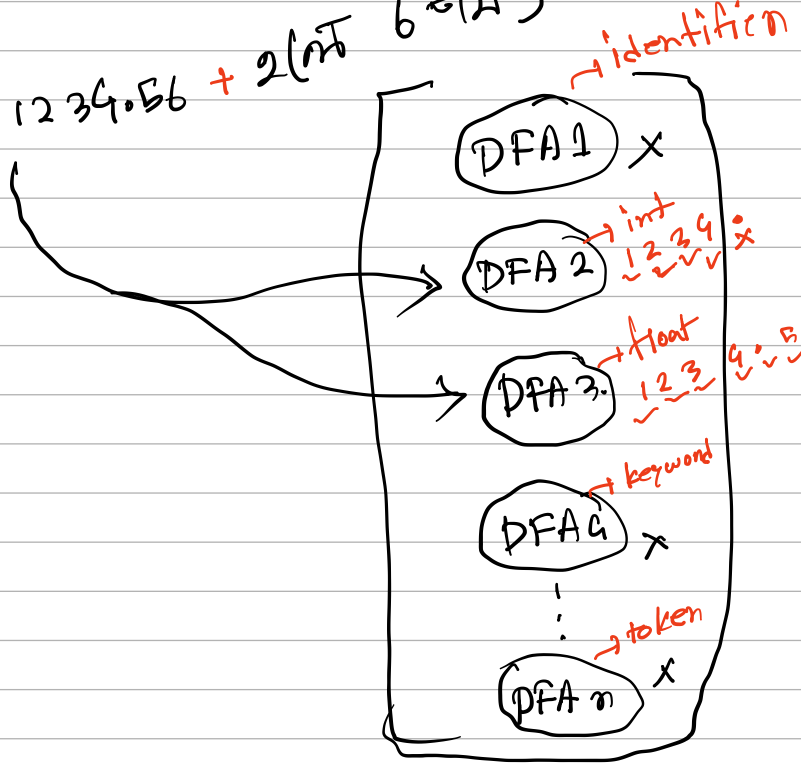
we define → Token → RE → DFA ← converted into

→1234

* we start from first character and match with regular expressions by constructing DFA. If we end up in final state we can conclude that this can be identified as integer token.

but 1234c ↳ here we can disagree

1234.56 + ২টো 6 পর্যন্ত float এ যাবে তারপর আবার operator এবং নতুন DFA start হবে আবার।



Lexical Analyzer

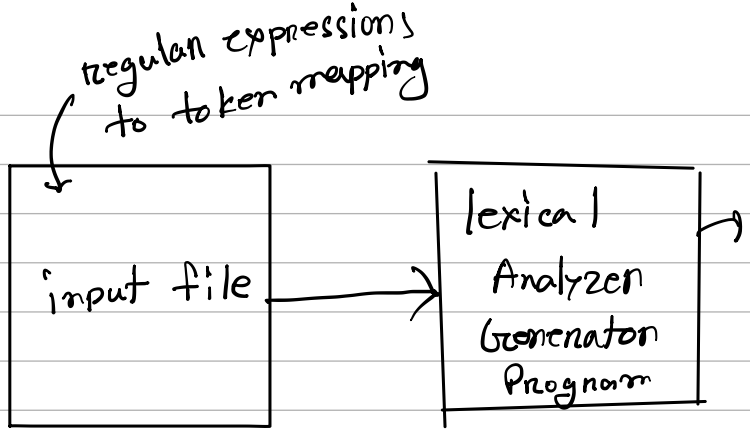
Parallelly makes progress in all DFAs that can still process the next character.

তাই 1234.56 হবে float হিসেবে tokenize হবে আর + কে operator ধরবে।

{ if → keyword
ifat → identifier ?

↳ here till 'if', both the identifier DFA & keyword DFA is progressing parallelly but when we get 'a' we can not start a new DFA cause idefier DFA is already progressed till 'if'. this is why we don't consider the keyword & just consider the identifier token.

DFA for keyword must be placed before DFA for identifier → otherwise all if, while, for etc will be tokenized as idetifier.



generates a C program

↓

It has all the DFAs

+

Code for parallel matching of characters to DFA

+

Buffer management logic

where we stopped? where should we start from?

FLEX generates a lex.yy.c type file. It can be added to another. output of lexical analysis is useful for syntax analysis.

→ In the lexical analysis stage, where is the error?

↳ when none of the DFAs can go to the end stage. [invalid token type]

↳ only identifiable error in this stage.