Lecture 1 January 10, 2019

CPSC - 411

Jasspreet Kaur jakaur@ucalgary.ca.

	Weightage
Assignments (Upto 4)	40%
Midterm Exam	25%
final Exam	35%
	100%

Compiler Construction: principles and practice by K.C. Louden

What is a Compiler:-

$$\begin{tikzcd}\text{source language} \arrow[r] & \boxed{\text{compiler}} \arrow[r] & \text{target language}\end{tikzcd}$$

compiler is a program which reads "program" in a source language and translates it into a program (often executable) in a target language.

Interpreter (officially) is not a compiler.

reads in a source program and produces the result of running or executing that program.

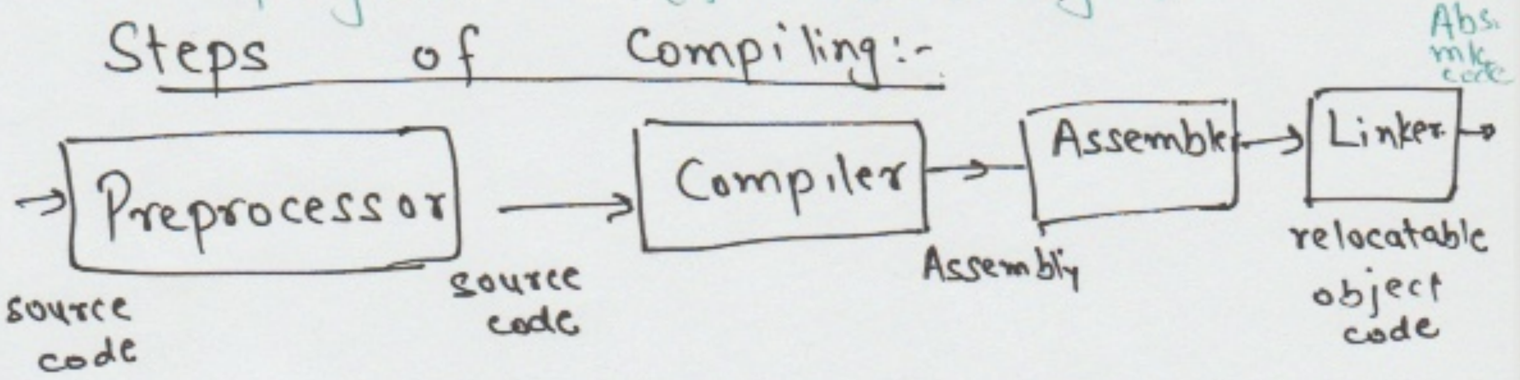
Compiler converts whole program in one go whereas interpreter does the same by taking one line at a time.

Interpreter	Compiler
- translates one statement at a time	- scans entire program and translate it as a whole target code.
- Take less time to analyze source code but overall execution time is slower	- Takes large amount of time to analyze the code but overall execution is faster.
- Ex, Prolog, Scheme, Miranda, Haskell are interpreted languages	- C, Pascal, PL/1, Fortran, C++ are compiled languages.

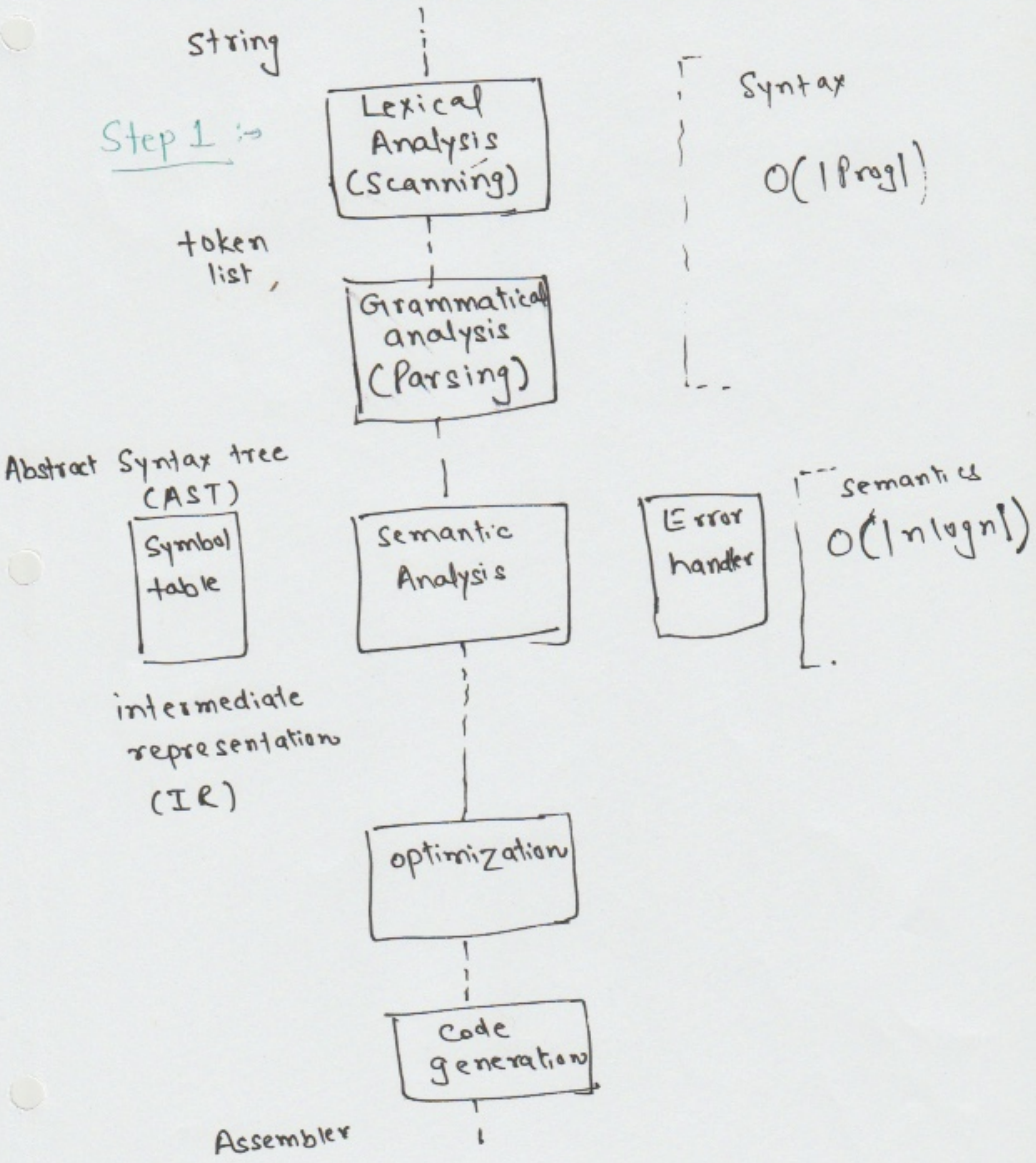
What should a compiler do ??

- Preserve the meaning of the original source code
- Produce reasonably (space and time) efficient target code
- Perform its job quickly. (order of the size of the program or O (nlogn).

Steps of Compiling:-



Overview of Compiler :→



Expression through a compiler

position = initial + rate * 60

The first question one must ask is whether this a valid expression in the given source language

(a) formal definition of language

(b) An effective membership test

(c) A plan for how to handle failure (i.e. error recovery!)

Usually the syntax of languages is specified in two stages

1. Specification of the Lexemes of the language (that is the substrings of the input which will have special significance to the compiler)

   Lexeme:- Basic lexical unit of a language. consists of one / several words.

   and their correspondance to tokens...

2. A specification of the grammar of the language based on tokens (obtained by translation from lexemes)

which are terminal symbols of the languages. This is usually done by providing a context free grammar.

Context free Grammer (CFG):- set of recursive rewriting rules (or productions) use to generate patterns of strings.
Most arithmatic expressions are generated by context free grammer.

Token :→ consists of one / more characters like words in natural language.

Step 1 : Lexical Analysis

Group characters together to form lexemes which are translated into takens which are the terminals of the grammar used in the next step.

Lexeme	Token
"127"	INT (00000000 1111111)
"length"	ID ("length")
" + "	ADD

This can be efficiently implemented using DFA (deterministic finite automata). These are often specified using "patterns" which are essentially regular expressions.

DFA :- Finite state machine (FSM) that will accepts or reject strings of symbols and only produces a unique compution of automation for each input string

Step 2 Grammatical Analysis:

January 15, 2019

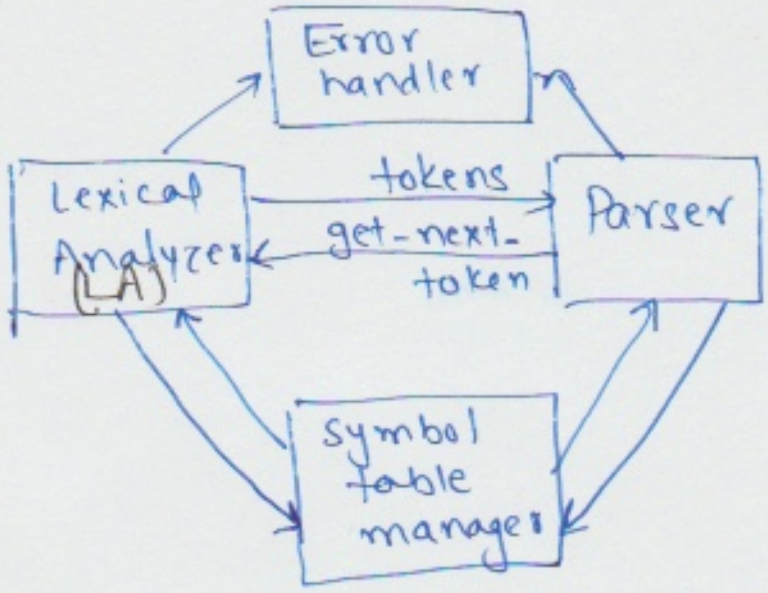
Lexical Analysis } → Scanning:-

part of compiler that does reading of the program.

Role of Lexical Analyzer:-

- read the input characters and produce a' sequence of tokens. for syntax analyzer
- To ignore white space (space, tab, newline characters in the source) and comments.
- To keep track of line number so that meaningful error messages can be generated.
- May enter identifiers into symbol table, enter literals into literal table.

Role of Scanner in a Compiler



Interaction of lexical analyzer with parser.

- LA collects the information about token into their associated attributes
  - the token influence parsing decision
  - the attributes influence the translation of token

Tokens:-	Keyword	special symbols			
keywords	if	+	;	/*	ID=letter
identifier	else	-	,	*/	NUM= digi
constants	int	*	(		
string literal	return	/	)		
brackets	void	<=	[		
[] {} ()	while	>=			
		==	]		
		!=	{		
			}		

Lexical Errors :

- is a sequence of character that does not match the pattern of any token.

- addition of an extra character
- removal of a character that should be present
- replacement of a correct with incorrect character
- Transposition of characters

Syntax Tree (ST):- dictionary → keeps track of information about the symbols that the source program is using

Purpose of ST:-

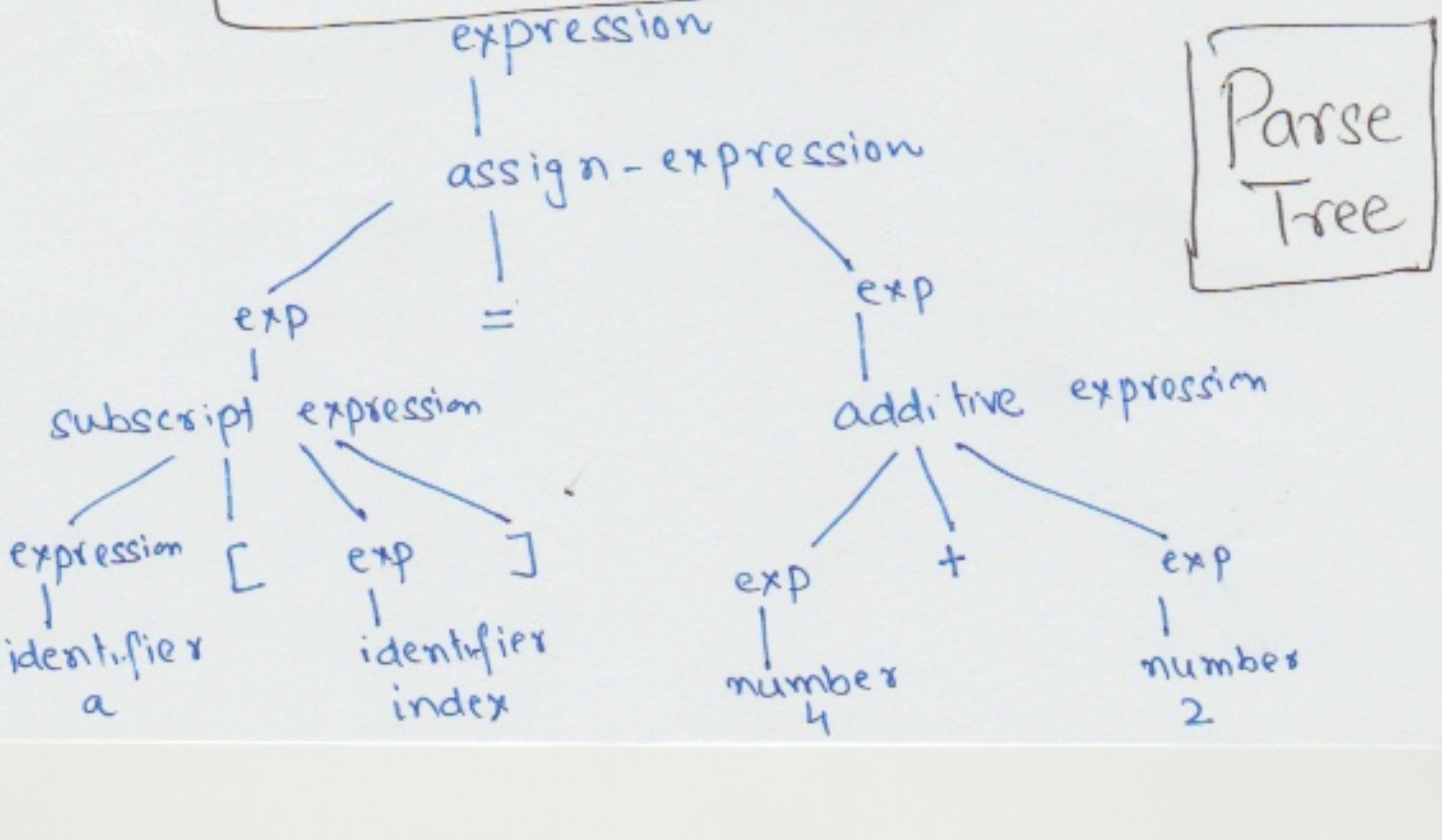
Attribute:- representation of the semantics of the name / information associated with symbol.

Parser / Syntax Analyzer → Receives source code in the form of tokens from scanner and perform syntax analysis which determine the structure of the program.

~~He~~ Similar to grammatical analysis in natural language

- Syntax analysis determines the structural elements of the program as well as their relationship
- The results of syntax analysis are usually represented as a Parse tree or a Syntax Tree.

$\boxed{a[index] = 4 + 2}$



- It represents a structural element called expression, which is an assignment expression consisting of a subscripted exp. on Left and an integer arithmatic exp. on right.

- Parse tree is a useful aid to visualize the syntax of the program/ program element
- Parser tend to generate syntax tree, which is condensation of information contained in parse tree.

Abstract Syntaxt Tree

Many nodes have disappeared (including token nodes

e.g. if we know that an exp. is subscript operation, then it is no ~~long~~ longer necessary to keep [ ]

Semantic Analysis :-

is its meaning as opposed to its syntax | structure.

- Once sentence is understood we can try to understand meaning
- But meaning is too hard for compiler.

Compilers perform limited analysis to catch inconsistencies.

- Semantics of a program determines its run-time behaviour.

Examples:-

Jack ~~and~~ said John left $\boxed{\text{his}}$ assignment at home

Jack said Jack left his assignmen at home. Jack left her notes at home

→ Semantic Analysis in Programming

- Programming language defines strict rules to avoid such ambiguities

```
{
int Jack = 3
  {
    int Jack = 4;
    cout << Jack ;
  }
}
```

Static Semantics

Most program languages have features that can be determined prior to execution and yet cannot be conveniently expressed as syntax and analyzed by parser.

Dynamic Semantics: property of the program that can only be detected by executing it.

Optimization:-

Automatically modifies program

- Run faster
- Use less memory
- conserve some resources

e.g. $x = y * 0$

$x = 0$

$a[index] = 4 + 2 \rightarrow$ precomputed by compiler

$a[index] = 6$

Code Generation:

Takes intermediate code and generates the code for target machine

- A translation into another language
- Analogous to human translation.

Issues:-

- How are erroneous programs handled
- language design
  - determine what is easy and hard to compile