ID: 20DCS036 : Prins Kumariya

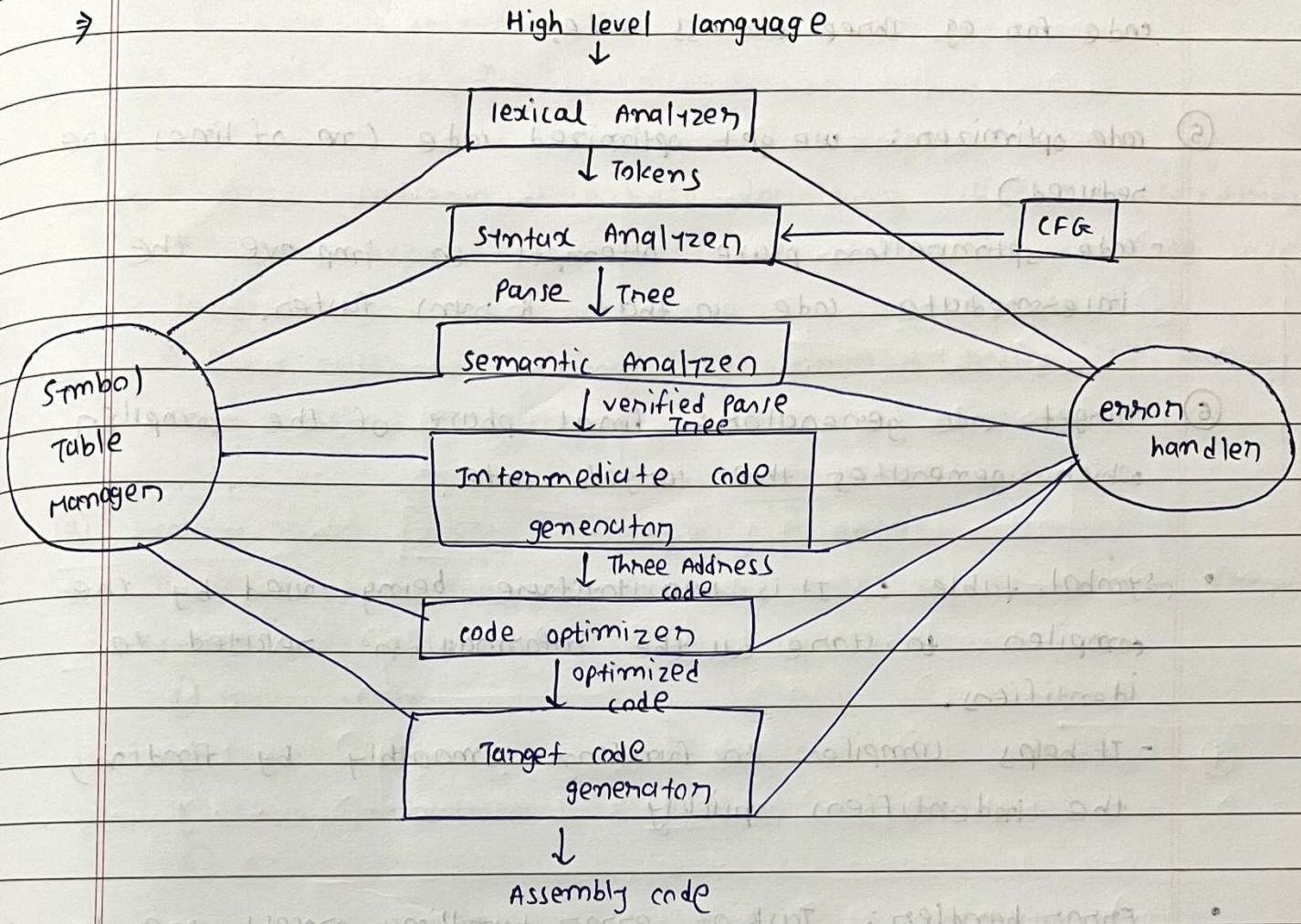
CS450: Design of language processor

Date __/__/__ Assignment

Saathi

① Draw and explain structure of compilers in detail.

⇒



① lexical Analyzer: reads the prog. and converts it into tokens, using a tool called lex tool.
- removes white spaces, comments, tabs etc,

② syntax Analyzer: constructs the parse tree
- takes the tokens one by one and uses CFG to construct the parse tree.
- syntax errors will be detected here in this phase.

③ semantic Analyzer: verifies the parse tree, whether its meaningful or not?
- It uses parse tree & info in the symbol table to check the source program for semantic consistency.

Date __/__/____

Saathi

④ Intermediate code generation : generates the intermediate code for eg. Three address code,

⑤ code optimizer : we get optimized code (no. of lines are reduced)
- code optimization phase attempts to improve the intermediate code so that it runs faster.

⑥ Target code generator : final phase of the compiler which generates the target code.

- symbol table : It is data structure being used by the compiler to store all the information related to identifiers.
  - It helps compiler to function smoothly by finding the indentifiers quickly.

- Error handler : Task of error handling process are to detect each error, report it to the user and make recover strategy.

② Explain Tokens, patterns and lexemes.

⇒ • Tokens : A token is a sequence of characters that can be treated as a unit / single logical entity.
  → Typical tokens are :
  Keywords, identifiers, operators etc.
  Example : (for, if, while), (variable name, function name) etc.

- lexemes : It is a sequence of characters in the source program that is matched by the pattern for a token.

Page No.

Date ___/___/_____

Saathi

→ sequence of i/p characters that comprise a single token is called a lexeme.

eg, "float", "=", ";"

- Pattern: Pattern is a rule describing all those lexemes that can represent a particular token in source code language.

→ These rules are defined by grammer rules, by means of patterns.

(3) Explain Error recovery strategies in detail.

→ The different strategies that a parse uses to recover from syntatic errors are:

1) Panic mode
2) Phrase level
3) Error productions
4) global correction

(1) Panic mode: on discovering an error, the parser discards input symbols one at a time until synchronizing token is found.

→ synchronizing tokens are usually delimeters such as semi-colon or end.

→ when multiple errors in same statement are there, this method is useful.

(2) Phase level: on discovering an error, parser performs local correction on the remaining input that allows it to continue.

Example: Insert a missing semicolon or ~~delete~~ delete an extra semicolon etc.

Page No.

Date __/__/____

(3) Error productions: The parser is constructed using augmented grammer with error productions. if an error production is used by the parser, appropriate error diagonstics can be generated to indicate the erroneous constructs recognized by the input.

(4) Global corrections: given incorrect input string x and grammer G, certain algorithms can be used to find parse tree for a string y, such that the number of insertions, deletions and changes of tokens is as small as possible.

⇒ These methods are in general too costly in terms of time and space.

(4) = Explain the role of lexical Analyzer.

⇒ → In lexical analysis phase, lexical Analyzer converts source prog. to stream of tokens.

→ lexical Analysis also called SCANNING.

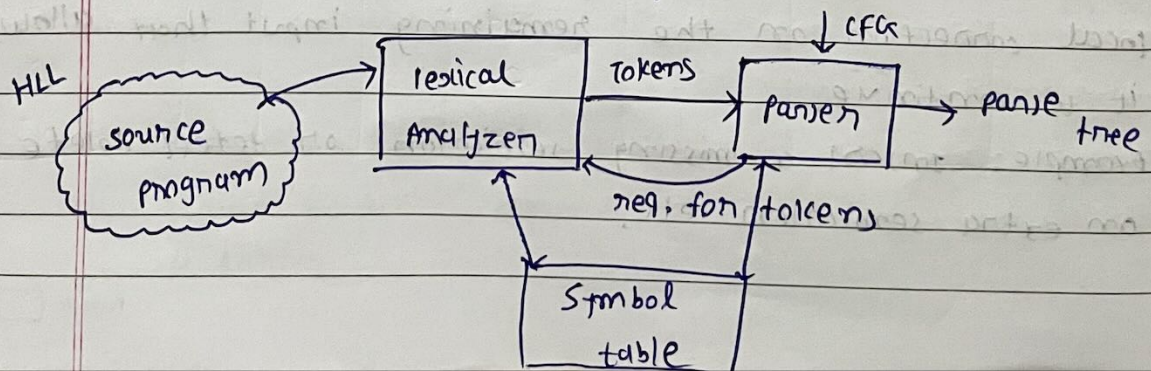
* Functions:

i) reads the i/p program character by character and it produces a stream of tokens and passes the data to the syntax Analyzer when it demands.

ii) removing white spaces / tabs.

iii) remove comments.

iv) lexical analyzer generates error and give the line No. of the error.



Date ___ / ___ / _____

Saathi

(5) find out first and FOLLOW set for all the non terminals

S → ACB | CbB | Bq

A → da | BC

B → g | ∈

C → h | ∈

⇒

First(S) = { d, g, h, ∈, a }

First(A) = { d, g, h, ∈ }

First(B) = { g, ∈ }

First(C) = { h, ∈ }

Follow(S) = { $ }

Follow(A) = { C }

Follow(B) = { a, $, C }

Follow(C) = { C }

(6) Elimination of left recursion

A → ABd | Aa | a

B → Be | b

⇒

A → aA'

A' → BdA' | aA' | ∈

B → bB'

B' → eB' | ∈

(7) Elimination of left factoring.

S → assbS | asaSb | abb | b

⇒

S → as' | b

s' → sA | bb

A → sbS | aSb

Saathi

Date ___ / ___ / _____

(8) for following grammer

$D \rightarrow TL;$

$L \rightarrow L, id \mid id$

$T \rightarrow int \mid float$

(i) remove left recursion

$D \rightarrow TL;$

$L \rightarrow idL'$

$L' \rightarrow ,idL' \mid \epsilon$

$T \rightarrow int \mid float$

(ii) find first and Follow

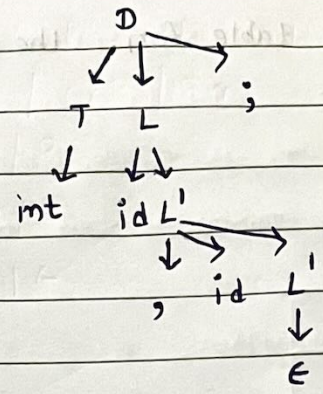
First (D) = { int / float }
First (L) = { id }
First (L') = { , , ∈ }
First (T) = { int, float }

Follow (D) = { $ }
Follow (L) = { ; }
Follow (L') = { ; }
Follow (T) = { id }

(iii) construct LL(1) Parsing table.

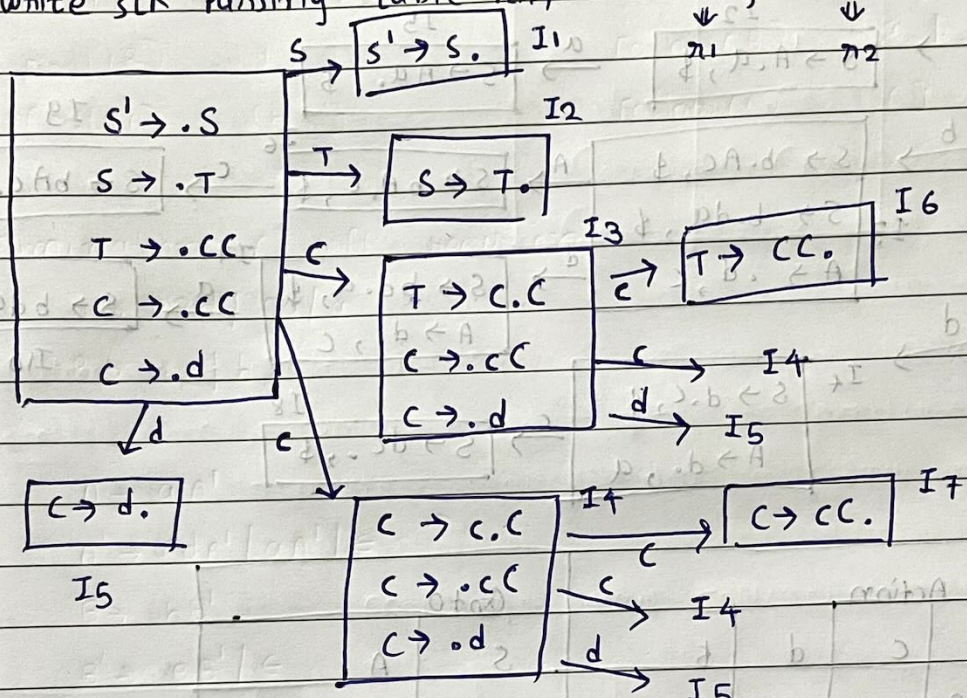
	;	,	id	int	float	$
D				D → TL;	D → TL;	
L			L → idL'			
L'	L' → ∈	L' → ,idL'				
T				T → int	T → float	

(iv) Parse the following string and draw Parse tree for input : int id, id;



(g) Write SLR parsing table for : $S \to T$, $T \to CC$, $C \to cC$, $C \to d$

$S \to T$: r_1 ; $T \to CC$: r_2 ; $C \to cC$: r_3 ; $C \to d$: r_4



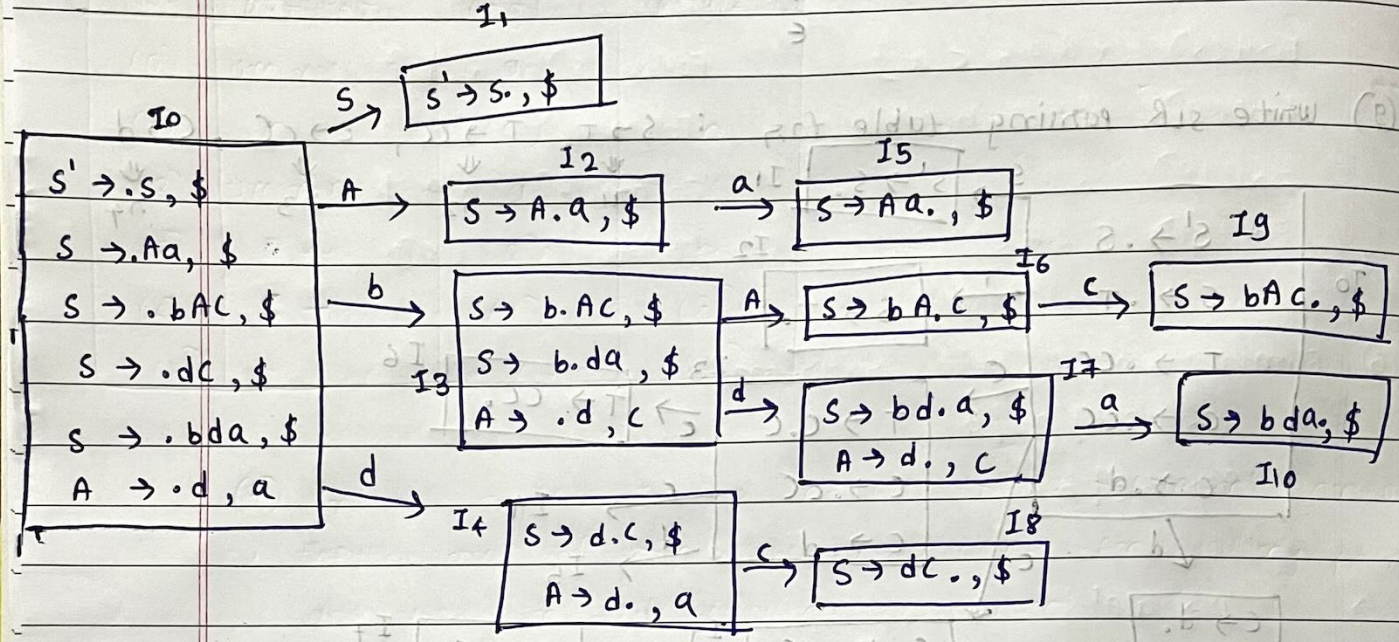
	Action			Goto		
	c	d	$	S	T	C
I_0	S_4	S_5		1	2	3
I_1			accept			
I_2			r_1			
I_3	S_4	S_5				6
I_4	S_4	S_5				7
I_5	r_4	r_4				
I_6			r_2			
I_7	r_3	r_3				

Date ___ / ___ / _____

(10) construct an LALR(1) parsing table for the following grammer:

$S \to Aa \mid bAC \mid dC \mid bda$

$A \to d$



	Action					Goto	
	a	b	c	d	$	S	A
I_0		s3		s4		1	2
I_1					accepted		
I_2	s5						
I_3				s7			6
I_4	r5		s8				
I_5					r1		
I_6			s9				
I_7	s10		r5				
I_8					r3		
I_9					r2		
I_{10}					r4		

Page No.

Date ___/___/_____

(11) Explain quadruple, triple and indirect triple with suitable example.

→ There are three types of representation used for three address code :

① Quadruple :

→ quadruple is a structure with at the most four fields such as op, arg1, arg2 and result.

→ The op field is used to represent the internal code for operator.

→ The arg1 and arg2 represent two operands.

→ And result field is used to store result of an expression.

Example : $x = -a * b + -a * b$

$t_1 = -a$
$t_2 = t_1 * b$
$t_3 = -a$
$t_4 = t_3 * b$
$t_5 = t_2 + t_4$
$x = t_5$

Three Address code

Quadruple :

No.	operator	Arg 1	Arg 2	result
(0)	uminus	a		t_1
(1)	*	t_1	b	t_2
(2)	uminus	a		t_3
(3)	*	t_3	b	t_4
(4)	+	t_2	t_4	t_5
(5)	=	t_5		x

② Triple : to avoid entering temp. names into symbol table, we might refer a temp. value by the position of the statement that computes it.

Page No.

Saathi

Date __/__/____

→ If we do so, three address statements can be represented by records with only three fields: op, arg1 and arg2

Example of Triple for above Three Address code:

NO.	operator	Arg1	Arg2
(0)	uminus	a	
(1)	*	(0)	b
(2)	uminus	a	
(3)	*	(2)	b
(4)	+	(1)	(3)
(5)	=	x	(4)

(3) Indirect Triple: In the indirect triple representation on the listing of triples has been done, and listing pointers are used instead of using statement.

Example: consider same expression

	statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

No.	operator	Arg1	Arg2
(0)	uminus	a	
(1)	*		b
(2)	uminus	a	
(3)	*		b
(4)	+		
(5)	=	x	

list of pointers to table

Date ___/___/_____

Saathi

(12) what is a syntax directed translation scheme? Explain with an example.

⇒ → The syntax directed translation scheme is a context free Grammer.

→ It is used to evaluate the order of semantic rules

→ In translation scheme, the semantic rules are embedded within the right side of production.

→ The position at which an action is to be executed is shown by enclosed between braces. It is written within the right side of the production.

(*) Implementation of syntax directed translation:

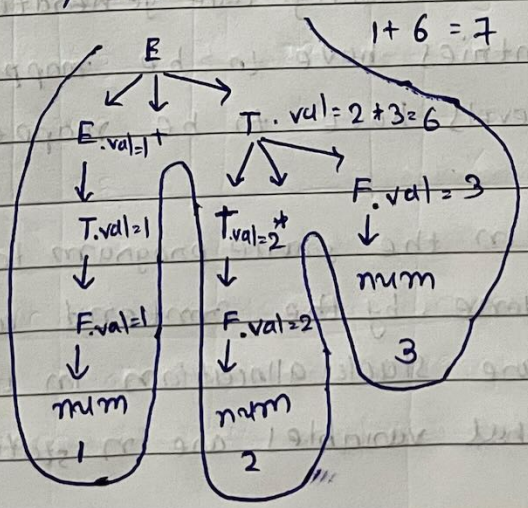
→ STD is implemented by constructing a parse tree and performing the actions in a ~~the~~ left to right depth first order.

→ STD is implementing by parse the input and a parse tree as a result.

Example:

Grammer	semantic rules
$E \to E + T$	{ E.val := E.val + T.val }
$E \to T$	{ E.val := T.val }
$T \to T * F$	{ T.val := T.val * F.val }
$T \to F$	{ T.val := F.val }
$F \to num$	{ F.val := num.val }

→ 1+2*3



Saathi

Date ___ / ___ / _____

(13) List and elaborate on the issues of the code generator.

⇒ 1) Input to code generator: Input to the code generator consists of the intermediate representation of the source program.

→ Types of intermediate languages:

1. Postfix notation
2. Quadruples
3. syntax trees or DAGs

→ The detection of semantic errors should be done before submitting input to the code generator.

→ The code generation phase requires complete error free intermediate code as an input.

2) Target program: The target program is the output of the code generator. The output can be:

a) Assembly language: It allows subprograms to be separately compiled.

b) Relocatable machine language: It makes the process of code generation easier.

c) Absolute machine language: It can be placed in a fixed location in memory and can be executed immediately.

3) Memory management: during code generation process the symbol table entries have to be mapped to actual p addresses and levels have to be mapped to instruction address.

→ Mapping name in the source program to address of data is co-operating done by the frontend and code

→ local variables are stack allocation in the activation record while global variables are in static area.

Page No.

Date __/__/____

4) Instruction selection: Nature of instruction set of the target machine should be complete and uniform.
→ when you consider the efficiency of target machine then the instruction speed and machine idioms are imp. factors.
→ The quality of the generate code can be determined by its speed and size.

5) Register allocation: during register allocation, we select the set of variables that will reside in registers at a point in the program.
→ During a subsequent register assignment phase, we pick the specific register that a variable will reside in.
→ finding an optimal assignment of registers to variables is difficult, even with single register value.
→ Mathematically the problem is NP complete.

6) Evaluation order: The efficiency of the target code can be affected by the order in which the computations are performed. some computation orders need fewer registers to hold results of intermediate than others.

(14) what is the Activation tree & record?
⇒ (*) Activation tree: an activation tree is used to depict the way control enters and leaves activations. In an activation tree,
a) Each node represents an activation of procedure.
b) The root represents the activation of the main program.
c) The node for a is the parent of the node b if and only if control flows from activation a to b.
d) The node for a is to the left of the node for b if and only if the lifetime of a occurs before the lifetime of b.

Date __/__/____

Saathi

(*) Activation Records: various field of activation record are:

1. Temporary values: The temp, variables are needed during evaluation of expressions. such variables are stored in the temp. field of activation record.
2. local variables: The local data is a data that is local to the execution procedure is stored in this field of activation record.
3. Saved machine registers: This field holds the information regarding the status of machine just before the procedure is called.
4. control link: This field points to the activation record of the calling procedure.
5. Actual parameters: This field holds the info. about the actual parameters. These actual parameters are passed to the called procedure.
6. Access link: It refers to the non local data in other activation record.
7. return values: This field is used to store the result of a function call.

(15) Explain peephole optimization method.

→ Peephole optimization is a type of code optimization performed on a small part of the code.

→ It basically works on the theory of replacement in which a part of code is replaced by shorter and faster code without a change in output.

→ objective of it:

1. To improve performance
2. to reduce memory footprint
3. to reduce code size

(*) Peephole optimization Techniques:

A. Redundant load and store elimination: in this techinique,

Page No.

redundancy is eliminated.

Examp. Initial code: y = x+5;
i = y;
z = i;
w = z * 3;

optimized code: y = x+5;
w = y * 3;

B. constant folding: The code that can be simplified by the user itself, is simplified. Here simplification to be done at runtime are replaced with simplified code to avoid additional computation.

Ex. Initial code: x = 2+3;
optimized code: x = 6;

C. Strength Reduction: The operators that consume higher execution time are replaced by the operators consuming less execution time.

Ex. Initial code: y = x*2;
optimized: y = x+x;

D. NULL sequences: Useless operations are deleted.

a := a+0;
a := a*1;
a := a/1;
a := a-0;

E. Combine operations: several operations are replaced by a single equivalent operation.

F. Dead Code Elimination: Dead code refers to portions of the program that are never executed or do not affect the

Date ___ / ___ / _____

Saathi

program's observable behaviour. Eliminating dead code helps improve the efficiency and performance of the compiled program by reducing unnecessary computations.