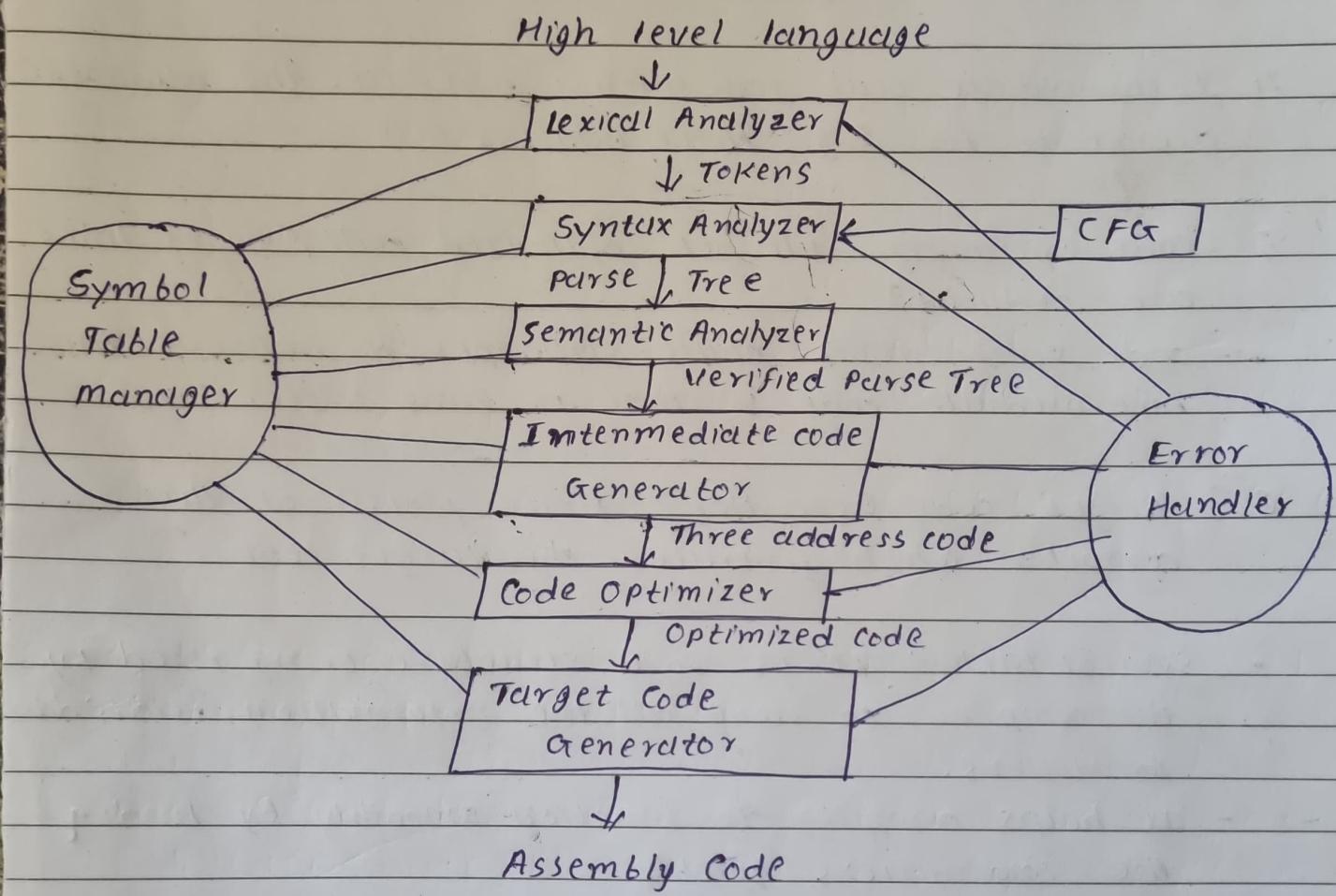


Assignment

1) Draw and explain structure of compiler 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 parse tree.

- Takes the tokens one by one and uses CFG to construct the parse tree
- Syntax errors will be detected 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.

4. Intermediate code generation : generates the intermediate code for e.g. 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 identifiers 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.

- Lexems: It is a sequence of characters in the source program that is matched by the pattern for a token.
 ↗ Sequence of input characters that comprise a single token is called a lexeme.
 e.g. "float", "=", ";".
- Pattern: Pattern is a rule describing all those lexems that can represent a particular token in source code language.
 → These rules are defined by grammar rules, by means of patterns.

3) Explain Error recovery strategies in detail.

These different strategies that a parser uses to recover from syntactic error are:

- 1) Panic mode
- 2) phase level
- 3) Error productions
- 4) Global corrections.

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 delimiters 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 semi colon or delete an extra semi colon etc.

3. Error productions : The parser is constructed using augmented grammar with error productions. If an error production is used by the parser appropriate error diagnostics can be generated to indicate the erroneous constructs recognised by the input.

4. Global correction : Given incorrect input string x and grammar G , certain algorithms can be used to find parse tree for a string y , such that the number of insertions, deletion and changes of tokens is as small as possible.

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

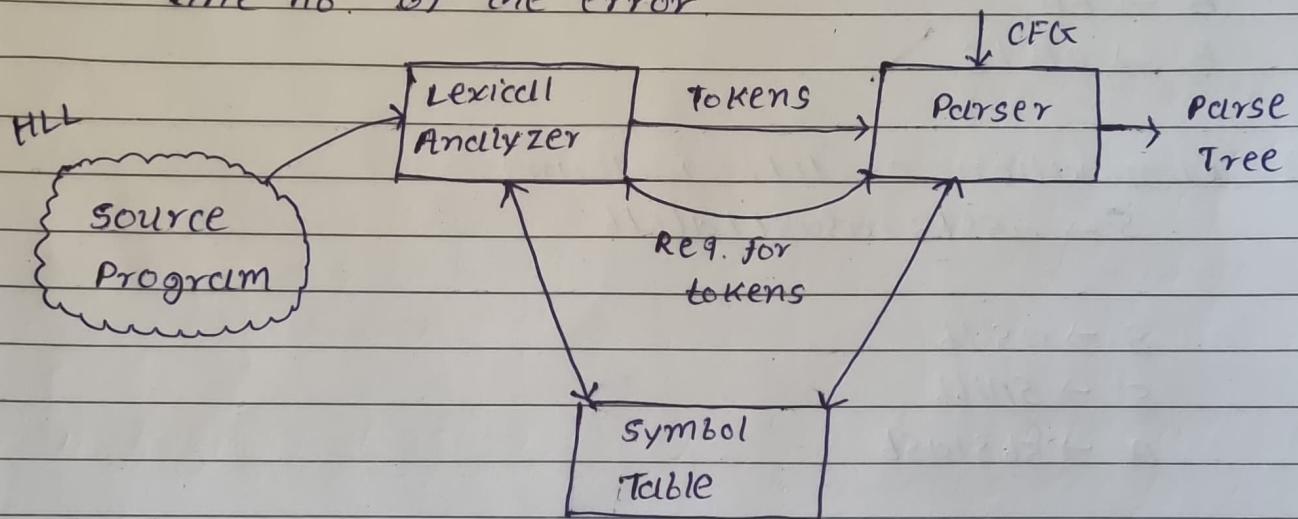
4). Explain the role of lexical analyzer.

→ In lexical analysis phase , lexical analyzer converts source program to stream of tokens.

→ Lexical analysis is 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.



5) Find out first and follow set for all the non-terminals.

$$S \rightarrow ACB \mid CBB \mid Bd$$

$$A \rightarrow dd \mid BC$$

$$B \rightarrow g \mid E$$

$$C \rightarrow h \mid E$$

$$\rightarrow \text{First}(S) = \{d, g, h, c, \epsilon\}$$

$$\text{First}(A) = \{d, g, h, \epsilon\}$$

$$\text{First}(B) = \{g, E\}$$

$$\text{First}(C) = \{h, E\}$$

$$\text{Follow}(S) = \{\$\}$$

$$\text{Follow}(A) = \{c\}$$

$$\text{Follow}(B) = \{d, \$, c\}$$

$$\text{Follow}(C) = \{c\}$$

6) Elimination of left recursion.

$$A \rightarrow ABd \mid BAcd$$

$$B \rightarrow BeB$$

$$\rightarrow A \rightarrow aA'$$

$$A' \rightarrow BdA' \mid cdA' \mid \epsilon$$

$$B \rightarrow bB'$$

$$B' \rightarrow eB' \mid \epsilon$$

7) Elimination of left factoring.

$$S \rightarrow aSSbS \mid aSaSB \mid abbb \mid b$$

$$\rightarrow S \rightarrow aS' \mid b$$

$$S' \rightarrow SA1bb$$

$$A \rightarrow SB \mid aSB$$

8.) For following grammar

$$D \rightarrow TL; \quad L \rightarrow L, id \mid id \quad ; \quad T \rightarrow int \mid float$$

i) Remove left recursion

$$D \rightarrow TL;$$

$$L \rightarrow id L'$$

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

$$T \rightarrow int \mid float$$

ii) Find First and Follow

$$\text{First}(D) = \{ \text{int}, \text{float} \}$$

$$\text{First}(L) = \{ id \}$$

$$\text{First}(L') = \{ , \epsilon \}$$

$$\text{First}(T) = \{ \text{int}, \text{float} \}$$

$$\text{Follow}(D) = \{ \$ \}$$

$$\text{Follow}(L) = \{ ; \}$$

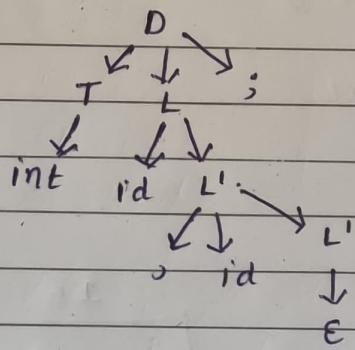
$$\text{Follow}(L') = \{ ; \}$$

$$\text{Follow}(T) = \{ id \}$$

(iii) construct LL(1) parsing table

$\$$	$:$,	id	int	$float$	$\$$
D						
L				$D \rightarrow TL;$	$D \rightarrow TL;$	
L'	$L' \rightarrow E$		$L' \rightarrow idL'$			
T	.			$T \rightarrow int$	$T \rightarrow float$	

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



9) Write SLR parsing table for : $S \rightarrow T, T \rightarrow CC, C \rightarrow cC, C \rightarrow d$

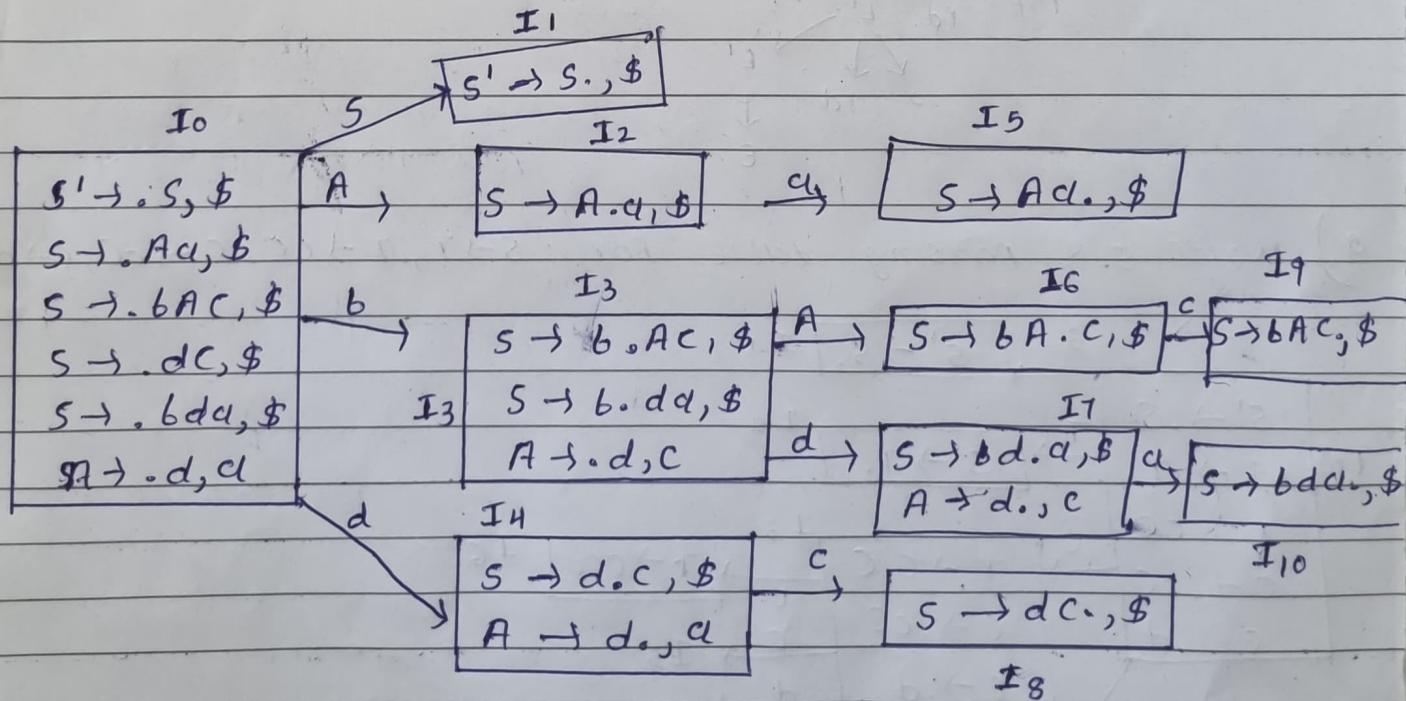
	s	$s' \rightarrow s.$	I_1
I_0	$s' \rightarrow .s$	T	I_2
	$s \rightarrow .T$		I_2
	$T \rightarrow .CC$	C	I_3
	$C \rightarrow .CC$		I_6
	$C \rightarrow .d$		I_4
	d		I_5
	c		
I_5	$c \rightarrow d.$	$c \rightarrow c.C$	I_4
		$C \rightarrow CC.$	I_7
		$C \rightarrow .CC$	I_4
		d	I_5

	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				

10.) Construct an LALR(1) Parsing table for the following grammar:-

$$S \rightarrow Acl \mid bAc \mid dC \mid bdd \mid$$

$$A \rightarrow d$$



$$\begin{aligned}
 x &= -a * b + -a + b \\
 t_1 &= -a \\
 t_2 &= t_1 + b \\
 t_3 &= -a \\
 t_4 &= t_3 * b \\
 t_5 &= t_2 + t_4 \\
 x &= t_5
 \end{aligned} \quad \left. \right\} \text{Three address code}$$

Quadruple

No operator	Arg1	Arg2	result
(0) uminus	a		t1
(1) *	t1	b	t2
(2) uminus	a		t3
(3) *	t3	b	t4
(4) +	t2	t4	t5
(5) =	t5		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.

→ 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

	Action						goto
	a	b	c	d	\$	s	A
I ₀			S ₃			1	2
I ₁					accepted		
I ₂	S ₅						
I ₃				S ₇			6
I ₄	r ₅		S ₈				
I ₅					r ₁		
I ₆		S ₉					
I ₇	S ₁₀		r ₅				
I ₈					r ₃		
I ₉					r ₂		
I ₁₀					r ₄		

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 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:-

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 points are used instead of using statement.

Example:- consider same expression.

Statement	No	operator	Arg1	Arg2
(0) (14)	(0)	uminus	a	
(1) (15)	(1)	*		b
(2) (16)	(2)	uminus	a	
(3) (17)	(3)	*.		b
(4) (18)	(4)	+		
(5) (19)	(5)	=	x	

list of pointers to table.

12). What is a syntax directed translation scheme. Explain with an example.



- The Syntax directed translation scheme is a context free grammar.
- 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:-

- SID is implemented by constructing a parse tree and performing the actions in a left to right depth first order.
- SID is implemented by parse tree the input and a parse tree as a result.

Example:- Grammer

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow \text{num}$$

Semantic rules

$$\{ E.\text{val} := E.\text{val} + T.\text{val} \}$$

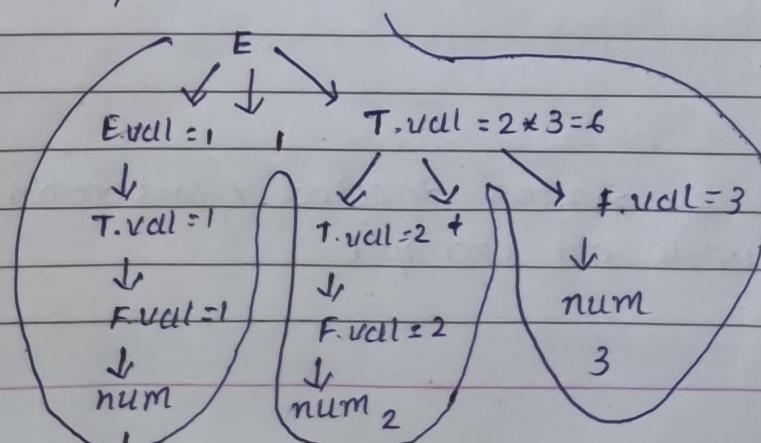
$$\{ E.\text{val} := T.\text{val} \}$$

$$\{ T.\text{val} := T.\text{val} * F.\text{val} \}$$

$$\{ T.\text{val} := F.\text{val} \}$$

$$\{ F.\text{val} := \text{num}.val \}$$

$$\rightarrow 1 + 2 * 3$$



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 error should be done before submitting input to the code generator.

→ The code generator 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:

- i) Assembly language
- ii) Relocatable machine language
- iii) Absolute machine language

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

→ Mapping name in the source program in address if data is co-operating done by the front end code.

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

- 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. functions.
 - The quality of the generated 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 further registers to hold results of intermediate than other.

14.)) What is the Activation tree and 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 .

*** Activation Records:** Various fields 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 method.
3. Saved machine registers: This field holds the info 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: The 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. 16

→ 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 :-

- 1) To improve performance
- 2) To reduce memory footprint
- 3) To reduce code size

④ Peephole optimization Techniques:-

A. Redundant load and store elimination:-

→ In this technique redundancy is eliminated.

Ex. 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 the 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 operations that consume higher execution time are replaced by the operators consuming less execution time.

Ex. Initial code : $y = x + 2;$

Optimized code : $y = x + 1;$

D. NULL sequences:-

→ useless operations are deleted.

$a := a + 0;$

$a := a + 1;$

$a := a ||;$

$a := a - 0;$

E. Combine operations:-

→ several operations are replaced by a single equivalent operation.

F. Deadcode Elimination:-

→ Dead code refers to portions of the program that are never executed or do not affect the program's observable behaviour. Eliminating dead code helps improve the efficiency and performance of the compiled program by reducing unnecessary computations.