CD ASST - 1.

1. Define Lexical Analysis. Explain Lexical Analysis in Detail.

→ It is the first phase in the compiler designing

→ A lexer takes the modified source code in which is written in the form of sentences.

→ In other words, it helps you to convert a sequence of characters into a sequence of tokens

→ The lexical analyzer breaks this syntax into a series of tokens.

→ It removes any extra space or comment written in the source code

→ programs that performs Lexical analysis in compiler design are called lexical analyzers or lexers.

→ The role of the lexical Analyzer in compiler design is to read character streams from the source code, check for legal tokens and pass the data to the syntax analyzer when it demands.

* Example :-

How pleasant is the weather.

See this lexical Analysis example. Here we can easily recognize that there are five words.

How pleasant, the Weather, is.

This is very natural for us as we can recognize the separators blanks and also punctuation symbol

How PL easant IS Th ewe ather?

→ Now check this example, we can also read this. However it will take some time because

the Separators are put the odd places.
→ It is not something which comes to you immediately

2) State the reasons for Separating lexical analysis and Syntax analysis.

A. → The lexical analysis is Separated from the Syntax analysis because of following reasons.

1) Simpler design
2) Compiler design effiency is improved.
3) Compiler portability is enhanced.

* Simpler Design
→ Separation allows the Simplification of one or the other

→ Example: A parser with comments or white spaces is more complex.

Compiler efficiency is improved
→ Optimisation of lexical analysis because of large amount of time is spent reading the source program and partitioning it into tokens.

Compiler portability is enhanced.
→ Input alphabet peculiarities and other device-specific anomalies can be restricted to lexical analyser.

→ Separation of the steps of lexical & Syntax analysis allows optimization of lexical analyzer and thus improves the efficiency of the process.

→ It also simplifis the parser and keeps it portable as a lexical analyzer may not always be portable

3) Define Lexeme and pattern

* Lexem :-

→ A lexeme is a sequence of alphanumeric character in a token.

→ The term is used both of the study of a language and in lexical Analysis of computer program compilation

→ In the context of programming of a computer, The lexemes are a part of the input stream from which tokens are identified.

→ A lexeme is an abstract unit of morphological analysis in

linguistics that roughly corresponds to set of forms taken by a single word.

For example:-

Run, runs, ran and running are forms of the same lexeme conventionally written as RUN.

→ Find, finds, found and finding are the forms of the single lexeme FIND.

* pattern.

→ A set of strings in the input for which the same token is produced as an output.

→ This set of strings is described by a rule is called pattern associated with the token.

→ The pattern is having association with Tokens & lexems

For example

Token	lexeme	pattern
relation	<, <=, = <>, >=, >	< or <= or = or <> or >= or
id	pi	letter followed by letters & digit
num	3.14	any numeric constants
literal	more	any character between "and" or except pattern

4) Write a note on design of a lexical analyzer generation.

* Design of a lexical - Analyser generation.

→ Design of lexical - Analyser generator

linguistic that roughly corresponds to set of forms taken by a single word.

For example:-

Run, runs, ran and running are forms of the same lexeme conventionally written as as RUN.

→ Find, finds, found and finding are the forms of the single lexeme FIND.

* pattern.

→ A set of strings in the input for which the same token is produced as an output.

→ This set of strings is described by a rule is called pattern associated with the token.

→ The pattern is having association with Tokens & lexems

For example

Token relation	lexeme	pattern
	<, <=, = <>, >=, >	< or <= or = or <> or >= or >
		letter followed by letters & digits
	pi	any numeric constants
	3.14	any character between " and " except "

4) Write a note on design of a lexical analyzer generation

* Design of a lexical - Analyser generation

→ Design of lexical - Analyser generator

→ The Structure of generated analyser.
→ Pattern matching based on NFA's.
→ DFA's for lexical analyser
→ Implementing the look ahead operator.
→ x is as long as possible for any xy satisfying conditions 1-3.
→ Dead States in DFA's.

* 1) The Structure of the Generated Analyser.
* Overview the architecture of a lexical Analyser generated by lex.
* The program that serves as lexical analyser includes a fixed program that

Simulates an automation.

* The rest of the lexical analyser consists of components that Simulates an automation are created from the lex program by lex itself.

* 2. Pattern Matching based on NFA's :-

→ In the lexical Analyser, the NFA Simulates an NFA such that it must read the input beginning at the point on its input which we have to referred to it as lexeme begin.

→ As it moves the pointers called forward ahead in the input, it calculates the set of states it is in each point following algorithm

3. DFA's for lexical Analysis.

→ Another architecture, resembling the output of lex, is to convert the NFA for all the patterns into an equivalent DFA using the Subset construction algorithm.

→ Within each DFA state, if there are one or more accepting NFA states, determine the first pattern whose accepting state is represented and make that pattern the output of the DFA state

5) Determine what is FA & define how a regular expression to FA.

* Finite Automata.

Finite automata is a state machine that takes a string of symbols as an input. and changes it accordingly.

→ F.A is recognizer for regular expression string is fed into F.A it changes its state for each literal.

→ If the input string is successfully processed & the automata reaches its final states, it is accepted. i.e the string just fed away said to be a valid token of the language.

$(\varepsilon), Q \times \varepsilon \to Q$

* Convert regular Expression into Finite Automata

To convert the regular expression

into finite automata we can use Subset method.

→ Subset method is used to obtain FA from RE.

→ Construct a transition diagram for a given RE by using NDFA with ε moves.

→ Convert NFA with ε to NFA without ε.

→ Thompson construction is an algorithm in automata theory used to convert a given RE to NFA.

26/4/2022

1) what is Syntax direct? what is SDT & explain the types.

Syntax directed translation refers to compiler implementation where the source language is completely driven by parsers. The parsing process and parse tree are used to direct semantic analysis & the translation of source program

SDT:-

A common method of syntax directed translation is translating a string into a sequence of action to each rule of grammar.

SDT provides a simple way to attach semantics to any such syntax

* Two types
  - S-attributed SDT
  - L-attributed SDT
* S-attribute uses the only synthesized attributes
* It is called S-attributed SDT.
* Semantic actions are placed in right most phase of RNS
* Attributes of L attributed SDT's are evaluated by depth first & left to right parsing manners.
* Semantic actions are placed anywhere in RNS

2) what is the difference between SDT and SDD in compiler design

SDT:-

* It embeds program fragmentation (also called Semantic actions) within production bodies

SDD:-

* It Specifies the values of attributes by associating Semantic rules with productions

3) What is the role of Syntax directed translate in genes:-

* In Syntax directed trans-lation along with the g

we associate some informal notations and the notations are called as semantic rules.

* Grammar + Semantic rule = SDT.
* In Syntax directed translation every non terminal concept can get one or more than one attribute sometimes attribute depending on the type of attribute.
* In symbol directed translation, whenever a construct encounting in the programming language, then it is translated according to the semantic rules define in that

particular programming language.

Ex.

production	Semantic Rule
$E \rightarrow E+T$.	$E.val := E.val + T.val$.
$E \rightarrow T$.	$E.val := T.val$.
$E \rightarrow T*F$	$E.val := T.value$.
$E \rightarrow F$	$E.val := F.value$
$F \rightarrow (F)$	$F.val := F.value$.
$F \rightarrow num$	$F.val := num.lexical$.

* E.val is the attribute of E num
* lexical is attribute returned by the lexical analysis or analyzer.

4) What is the differences b/w S-attributes & L-attributes Syntax directed translation

S-attributed SDT:-

If an SDT uses only synthesized attributes. It is called S-attributes SDT.

* S-attributes SDT's are evaluated in bottom up parsing as the values of the parent nodes depend upon the values of the child nodes.
* Semantic actions are placed in the right most place of RHS.

L-Attributed SDT.

If an SDT uses both synthesized attributes & inherited attribute

a restriction inherited attribute can inherit values from left siblings only. It is called as L-attributed SDT.

* Attribute in L-attributed SDT's are evaluated by depth first & left to right parsing manner.
* Semantic actions are placed anywhere in RHS.

5) Where is SDT evaluation order? What are the components on parser.

* The evaluation order for SDT is a scheme of CFG.
* The Syntax Directed Translation Scheme 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 enclosed b/w braces.
* It is written within right side of production

For Example.

Production	Semantic rules
$S \rightarrow E\$$	{ print E.val }
$E \rightarrow E+E$	{ E.val := E.val + E.val }
$E \rightarrow E*E$	{ E.val := E.val * E.val }
$E \rightarrow (E)$	{ E.val := E.val }

$E \rightarrow I$	$\{E.val := I.val\}$
$I \rightarrow I\ digit$	$\{I.VAL := I.VAL + I.VAL\}$
$I\ digit$	$\{I.VAL := LEX\ VAL\}$

6) Which mapping is druited to implement Syntax directed Translations?

Syntax druited translation provides a method for describing as input output mapping & that description is undipendent of the implementation. To implemint SDT we can use a stack that consists

of a pair of variables

STATE & VALUE.

D. Which Techniques are used in Intermediate Code Generation?

* In the analysis synthesis model of a compiler, the front end of a compiler translates a source program into an independent Intermediate code, then the back end of the compiler uses this intermediate code to generate the target code.

* The benefits of using machine independent code are :

Lexical Analysis
↓
Syntax Analysis
↓
Semantic Analysis
↓
Intermode Generation
↓
Code Optimization
↓
Target Code Generation

Front end

Back end

2) What is the purpose of runtime environment. What is the difference between runtime and Compile Time?

* Runtime environment act as small OS and provide

all the funtionality necessary for a program to run.

* This includes interfaces to physical parts of the hardware, user interactions and software components.
* A runtime environment loads application and then runs on a platform.

Compile Time

* Time period for translation of source code like Java to intercode like class.

Runtime

* Time period b/w start & end of running intermediate code at runtime environment.

* This is to check the syntax and semantics of the code

* This is to run the code

What happens during Code Generation?

* Code generation is a mechanism where a compiler takes the source code as an input & converts it into machine code. This machine code is actually executed by the System.

* Code generation is generally considered the last phase of compilation

although there are multiple intermediate steps performed before the final executable is produced.

* These intermediate steps are used to perform optimization and other relevant processes.
* In intermediate phases, optimization processes are dependent on each other, so they are applied one after another based on dependency hierarchy.
* After passing multiple phases, a parse tree or an abstract tree is generated (i.e input to the code generation (syntax)

Now, code generation converts it into linear sequential instructions.

* Final optimized code is the machine code for execution & output generation.

What are the goals of Output Generation?

There are 3 goals for any good code generator.

They are:-

i) Correctness:-

It should produce a correct code.

to represent an expression.
& the value computed at
each instruction is stored
in temporary variable
generated by compiler.
* The compiler decides the
order of operation given
by 3 address code

Control flow:-

* Control flow is the order
in which computer executes
statements in a script
* Code is run in order from
the first line in the file
to the last line, unless
the computer runs across
the statements that change

the control flow, such as conditionals & loops.

6) Define advantages of Intermediate code generation. Write a note on dead code in Compiler Design.

A) Advantages of Intermediate Code Generation :-

* It is machine independent
* It creates the function of code optimization.
* It can perform efficient front end, a new compiler for a given back end can also be generated.

Dead Code :-

Dead code is one or more than one code statements which are :-

* Either never executed or unreachable.
* Or if executed, their output is never used.

CD ASST - 4.

1). What is machine Independent optimization? What is the importance of optimization of code.

Machine independent code optimization tries to make the intermediate code more effient by transforming a section of code that doesn't involve hardware components like CPU registers or any absolute memory location.

Generally, it optimizes code by eliminating redundancies reducing the no. of lines in code, eliminating useless code or reducing the frequency of repeated code.

* The importance of code optimisation is that modifies the code and improves the code quality which makes more effient.

* It also consumes less memory, executes more rapidly, or performs fewer input/output operations

2. what are difference b/w m/c dependent & undependent optimization.

Machine dependent code optimization.

* When converting the source code to object code or target code, the compiler goes through several phases.
* First, the source code is given to lexical analyser which produces tokens.
* Then, the output is given to syntax analyser which investigates whether the generated tokens are in logical ~~lexical~~ order
* Allocating sufficient amount of resources can improve the execution of the program in this optimization

Machine Independent Code Optimization

* Machine Independent optimisation, this code optimization phase attempts to improve the intermediate code to get a better target code as the output.
* The part of the intermediate code which is transformed here doesn't involve any CPU registers memory location

3) What is Peehole in CD? What is used to optimize the compiler?

* Peehole optimization is a type of optimization

performed on small part of the code.

* It is performed on a very small set of instructions in a segment of code.
* The small set of instructions or small part of code on which peehole optimization is performed is known as peehole or window.
* Compiler optimisation is generally implemented using a sequence of optimizing transformation algorithms which take a program & transform it to produce a semantically equivalent output program

that uses fewer resources or executes faster

4) What are the advantages & disadvantages of code optimization?

Advantages:-

* Easy to implement
* Effiient.
* Easy to communicate.
* Flexible.
* It considers the quality performance.
* Usually optimal.

Disadvantages.

* Need to be updated continuesly

* Often non-optimal
* Subjective
* often non-optimal & complicate
* may consume more time.

5) which phase of compiler can check Syntax errors? In which phase type checking is done.

A * Syntax analysis or parsing is the second phase of a compiler, in this phase of compiler, the Syntax errors are checked.
* After the phase of lexical analyser, the compiler is able to check Syntax errors.
* The checking is done at

Semantic analysis phase
& parsing is done at
Syntax analysis phase

6) What is the importance of optimising the code? Why is Semantic Analyser used?

A * Code optimization is any method of code modification to improve code quality and efficiency

* A program may be optimized so that it becomes a smaller size, consumes less memory.

* Executes more rapidly,

ou performs fewer input/
output operation.
* Hence brings more effiency
and flexibility

Semantic Analyser:-
* Semantic Analyser
uses Syntax tree &
Symbol table to check
whether the given table
program is semantically
consistent with language
defination
* It gathers type information
and stores it in either
Syntax tree or Symbol table

Functions are.

i) type checking :-
Ensures that the data types are used in a way consistent with their defination.

(ii) Label Checking :-
A program should contain labels references

(iii) Flow Control Chek :-
Keeps a chek that control structures are used in proper manner.

—x—