CS3501-Compiler Design

Question Bank

Unit-I

INTRODUCTION TO COMPILERS & LEXICAL ANALYSIS

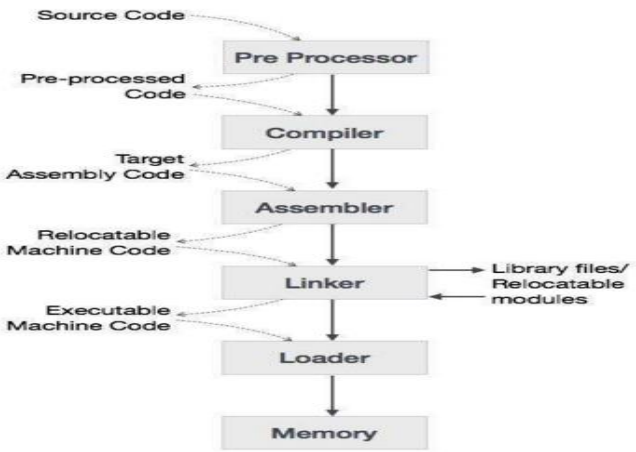
Structure of a compiler – Lexical Analysis – Role of Lexical Analyzer – Input Buffering – Specification of Tokens – Recognition of Tokens – Lex – Finite Automata – Regular Expressions to Automata – Minimizing DFA.

Part-A

1. **Mention few cousins of the Compiler. M/J 2012**
   - Pre-processor
   - Assembler
   - Loader
   - Link-Editor

2. **What are the two parts of a compilation? Explain. M/J'2016**
   There are two parts to compilation: **analysis and synthesis**.
   i)Analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.
   ii) Synthesis part constructs the desired target program from the intermediate representation. During analysis, the operations implied by the source program are determined and recorded in a hierarchical structure called a **tree.** A special kind of tree called **a syntax tree**

3. **Illustrate diagrammatically how a language is processed. M/J'2016**



4. **What are the possible error recovery actions in lexical analyzer? M/J 2012, A/ M'2015**
   1. Deleting an extraneous character
   2. Inserting a missing character
   3. Replacing an incorrect character by a correct character
   4. Transposing two adjacent characters

5. Define tokens, Patterns and lexemes M/J'2013, N/D'2016

Tokens: A token is a pair consisting of a token name and an optional attribute value.A token name is an abstract symbol representing a kind of lexical unit eg:a particular keyword or a sequence of input character denoting an identifier.
Patterns: A Pattern is a rule describing the set of lexemes that can represent a particular token in source program. Ex: **relation : all six relational operator**
Lexeme: A lexeme is a sequence of characters in the source program that is matched by the pattern for the token. For example in the Pascal's statement const pi = 3.1416; the substring pi is a lexeme for the token identifier

TOKEN	SAMPLE LEXEMES	INFORMAL DESCRIPTION OF PATTERN
const	Const	const
if	if	if
relation	<,<=,=,<>,>,>=	< or <= or = or <> or >= or >
id	pi,count,D2	letter followed by letters and digits
num	3.1416,0,6.02E23	any numeric constant
literal	"core dumped"	any characters between " and " except"

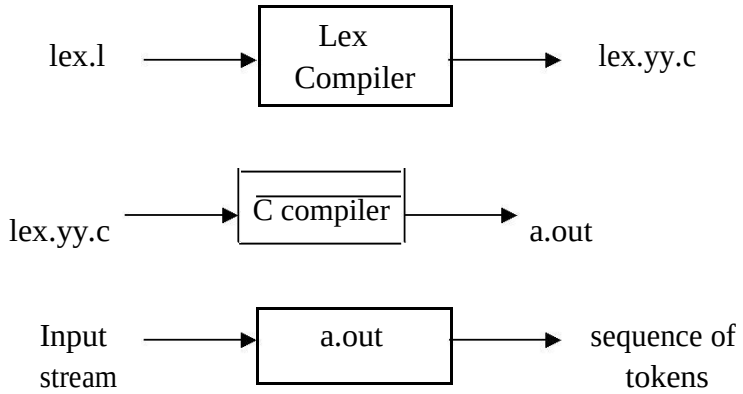
6. Mention the issues in a lexical analyzer. M/J'2013

1)Simpler design is the most important consideration.
- A parser including the conventions for comments and white space is significantly more complex

2) Compiler efficiency is improved.
- Specialized buffering techniques for reading input characters and processing tokens

3) Compiler portability is enhanced.
- Input alphabet peculiarities and other device-specific anomalies can be restricted to the lexical analyzer.

7. What are the components of LEX? N/D'2015



8. State any two reasons as to why phases of compiler should be grouped. M/J'2014

(i) Operation of compilation.-Analysis (Front End) and Synthesis(Back End)
(ii) Number of Passes- One Pass and Multi-Pass

9. Define Lexeme. M/J'2014

A lexeme is a sequence of characters in the source program that is matched by the pattern for the token. For example in the Pascal's statement
const pi = 3.1416; the substring **pi** is a lexeme for the token "identifier"

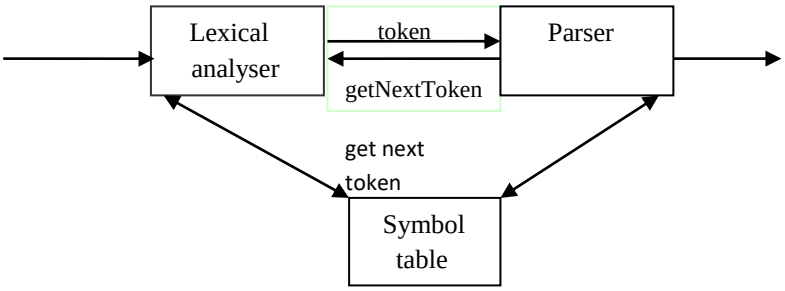
10. List the operation on languages. M/J'2016

- A **language** is any countable set of strings over some fixed alphabet.

Operations on languages:

- The following are the operations that can be applied to languages:
  1.Union
  2.Concatenation
  3.Kleene closure
  4.Positive closure

11. State the interactions between the lexical analyzer and the parser. N/D'2015



- Its main task is to read the input characters and produces output a sequence of tokens that the parser uses for syntax analysis.
- As in the figure, upon receiving a **"get next token"** command from the parser the lexical analyzer reads input characters until it can identify the next token.

12. Define error recovery strategies. N/D'2015

- Panic mode Recovery
- Phrase level
- Error Production
- Global Correction
- Other Error recovery action
  1)Deleting an extraneous character.
  2) Inserting a missing character.
  3)Replacing an incorrect character by a correct character.
  4)Transforming two adjacent characters

13. What is a symbol table? N/D'2016, M/J'2014

- A Symbol table is a data structure containing a record for each identifier with fields for the attributes of the identifier.
- The data structure allows us to find the record for each identifier quickly and to store or retrieve data quickly.

- Whenever an identifier is detected in any of the phases, it is stored in the symbol table.
- Ex: Var posi, init, rate : real;

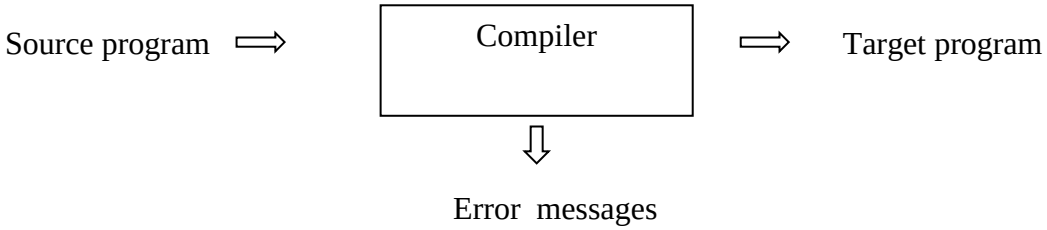
pos	Real	…
init	Real	…
rate	Real	…

14. List out various compiler construction tools. N/D'2016

- Scanner Generators
- Parser Generators
- Syntax-Directed Translation Engines
- Automatic Code Generators
- Data-Flow Engines

15. Define a compiler?

A Compiler is a program that reads a program written in one language (source language) and translate it into an equivalent program in another language (target language). While compilation, the reports to its user the presence of errors in the source program.



16. What are the functions of preprocessors?

- Macro processing
- File inclusion
- Rational preprocessors
- Language Extensions

17. Define Interpreter. APRIL/MAY 11

An interpreter is another common kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user, as shown in Fig.

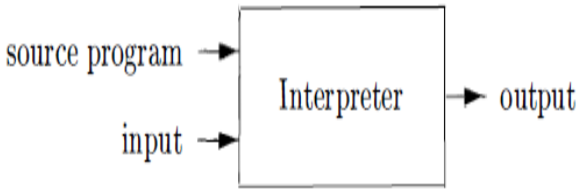


Figure 1.3: An interpreter

18. Describe Assembly code with an example.

Assembly code is a mnemonic version of machine code. In which names are used instead of binary codes for operations, and names are also given to memory addresses.

A typical sequence of assembly instructions might be

MOV a , R1 0001 01 00 00000000
ADD #2 , R1 0011 01 10 00000010
MOV R1 , b 0010 01 00 00000100

(i) In Which the first four bits are instruction are, with 0001, 0010,0011 standing for load, store and add.
(ii) The next two bits designate a register. (ie) 01-R1
(iii) The next two bits are 00-> ordinary address and 10-> immediate mode.

Last eight bits refer to memory address.

19. Compare compiler & interpreter.

S.No	Compiler	Interpreter
1	Program is analyzed only once & the code is generated.	The source program gets interpreted every time so that it can be executed.
2	Compilers produce object code.	It does not produce object code.
3	More efficient than interpreter.	Less efficient.
4	It is a complex program & requires high memory.	It is simpler & requires less memory.

20. What are the tools available for analysis part? Describe about any two.

- **STRUCTURE EDITORS:** The structure editor not only performs the text-creation and modification functions of an ordinary text editor, but it also analyzes the program text, putting an appropriate hierarchical structure on the source program.
- **Pretty Printers:** A pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visible
- **Static Checkers:** A static checker reads a program, analyzes it, and attempts to discover potential bugs without running the program.
- **Interpreters:** An interpreter might build a syntax tree and then carry out the operations at the nodes as it walks the tree.

PART B

1. Explain the various phases of compiler in detail, with a neat sketch A/M2012, N/D 2012, M/J'2013, N/D'2013, M/J'2014,M/J'2016, N/D'2016

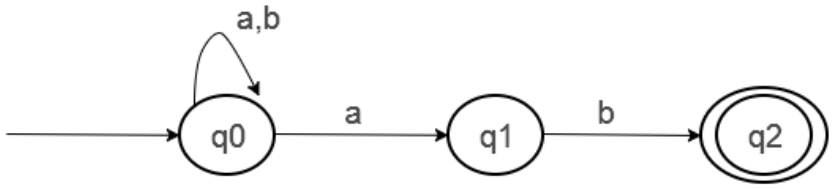
Contents:

- Explain all the Phases of Compiler
- Neatly Represent the Diagram

2. **Explain language processing system with neat diagram.(8) (cousins of compiler) M/J'2016**

   Contents:
   - Explain all the Cousins of Compiler
   - Neatly Represent the Diagram

3. **Convert the following NFA into Equivalent DFA**



Ans:

Steps for converting NFA to DFA:

Step 1: Convert the given NFA to its equivalent transition table

Step 2: Create the DFA's start state

Step 3: Create the DFA's transition table

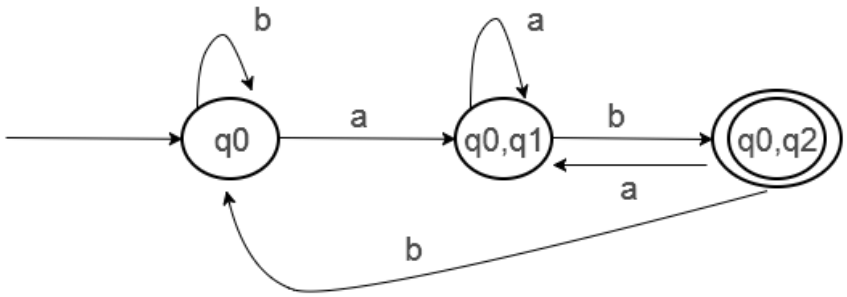
Step 4: Create the DFA's final states

Step 5: Simplify the DFA

Transition Table

State	a	B
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}
{q0,q2}	{q0,q1}	q0

Transition Diagram



4. **Convert the regular expression abb (a|b) to DFA using direct method and minimize it.(AU–April / May 2017)**

   Contents:
   - Define Regular expression
   - Steps followed to the convention

- Follow the Standard Rules
- Convert the Regular Expression
- Write the Minimization Algorithm Steps and illustrate the converted Regular expression

5. **Draw the Transition Diagram for relational operators and unsigned numbers.(AU –April /May 2017)**
   Contents:
   - Define Finite Automata
   - Write down the Two Notations
   - Explain in detail the both notations
   - Give any one example

6. **Explain In detail About the Input Buffering Methods.**
   Contents:
   - Define Input Buffering
   - Write down the Two Methods of Buffering
   - Explain in detail the both methods with example
   - Write the Advantages and Disadvantages

7. **How to Specify the Tokens and explain the concepts. (8 Marks)**
   Contents:
   - Define Tokens
   - Write down the Three types of specifications
   - Explain in detail the both types with example

8. **Explain the Lex Tools in details (5 Marks)**
   Contents:
   - Define LEX
   - Explain in detail with example

Part–C (1 x 15 = 15 Marks)

1. What are the Phases of compiler? Explain the Phases in detail. Write down the output ofeach phases for the expression a: = b + c * 60(AU–April / May 2017)

Contents:
- Define Compiler
- Explain all the Phases of Compiler
- Neatly Represent the Diagram
- Write the Example Expression to be match with all phases ofcompiler

Unit-II

SYNTAX ANALYSIS

Role of Parser – Grammars – Context-free grammars – Writing a grammar Top Down Parsing – General Strategies – Recursive Descent Parser Predictive Parser-LL(1) – Parser-ShiftReduce Parser-LR Parser- LR (0)Item Construction of SLR Parsing Table – Introduction to LALR Parser – Error Handling and Recovery in Syntax Analyzer-YACC tool – Design of a syntax Analyzer for a Sample Language

Part-A

1. Define Define Recursive Descent Parsing?

A Recursive Descent Parser (RDP) is a type of top-down parsing technique used incomputer science to analyze and process a language's syntax.

2. **Differentiate Top Down parsing and Bottom Up parsing?. A/M 2012**

Top Down parsing	**Bottom Up parsing**
It is a parsing strategy that first looks atthe highest level of the parse tree andworks down the parse tree by using therules of grammar.	It is a parsing strategy that first looks atthe lowest level of the parse tree andworks up the parse tree by using the rulesof grammar.
This parsing technique uses Left MostDerivation.	This parsing technique uses Right MostDerivation.
Example: Recursive Descent parser.	Example: Its Shift Reduce parser.

3. Compare Syntax tree and Parse tree. N/D 2012

Syntax tree:

During analysis, the operations implied by the source program are determined and recorded in a hierarchical structure called a **tree.** A special kind of tree called **a syntax tree** is used, in which each node represents an operation and the children of a node represent the arguments of the operation.

Parse Tree:

A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding replacement order. Each interior node of a parse tree is labeled by some nonterminal A and that the children of the node are labeled from left to right by symbols in the right side of the production by which this A was replaced in the derivation. The leaves of the parse tree are terminal symbols.

4. Write the rule to eliminate left recursion in a grammar. N/D 2012

A grammar is said to be *left recursive* if it has a non-terminal *A* such that there is a derivation A=>Aα for some string α. Top-down parsing methods cannot handle left-recursive grammars. Hence, left recursion can be eliminated as follows:

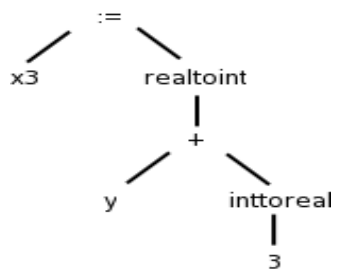
If there is a production A → Aα |β it can be replaced with a sequence of two productions

A → βA'
A'→ αA' | ε without changing the set of strings derivable from A

5. Mention the role of semantic analysis. N/D 2012

There is more to a front end than simply syntax. The compiler needs semantic information, e.g., the types (integer, real, pointer to array of integers, etc) of the objects involved. This enables checking for semantic errors and inserting type conversion where necessary.

For example, if y was declared to be a real and x3 an integer, We need to insert (unary, i.e., one operand) conversion operators "inttoreal" and "realtoint" as shown on the right.



6. Draw the syntax tree for the expression: a=b*-c +b*-c N/D 2012

A syntax tree depicts the natural hierarchical structure of a source program. A **DAG** (Directed Acyclic Graph) gives the same information but in a more compact way because common sub-expressions are identified. A syntax tree for the assignment statement a:=b*-c+b*-c appear in the figure.

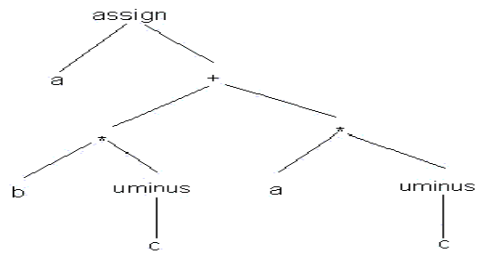


Fig: Graphical Representation of a := b * -c + b * -c

7. Eliminate left recursion from the following grammar A->Ac/Aad/bd/€. M/J'2013

A->bdA'/A'

A'->cA'/adA'/€

8. Eliminate left recursion from the grammar S-> Aa | b ; A-> Ac | Sd | €. N/D'2013

Ans: S-> Aa | b ; A-> Ac|Aad |bd|€

Equivalent Rule:

S-> Aa | b

A->bdA'/A'

A'->cA'/adA'/€

9. Write a CF grammar to represent palindrome. N/D'2014

S->aSa | bSb | a | b| €

String={aba,bab,abba,ababa,...}

10. Write the role of parser. A/ M'2015

- The parser or syntactic analyzer obtains a string of tokens from the lexical analyzer and verifies that the string can be generated by the grammar for the source language.
- It reports any syntax errors in the program.
- It also recovers from commonly occurring errors so that it can continue processing its input.

11. Write a grammar for branching statement. M/J'2016

S ->iEtS | iEtSeS | a

E -> b

12. Write the algorithm for FIRST and FOLLOW in parser. M/J'2016

First() : Let α be a string of grammar symbols. First(α) is the set that includes every terminal that appears leftmost in α or in any string originating from α.

Follow () : Let A be a non-terminal. Follow(A) is the set of terminals a that can appear directly to the right of A in some sentential form. ($S \Rightarrow \alpha A a \beta$, for some α and β).

Rules for follow( ):

1. If *S* is a start symbol, then FOLLOW(*S*) contains $.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except ε is placed in follow(*B*).
3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains ε, then everything in FOLLOW(*A*) is in FOLLOW(*B*).

13. Define LL (1) Grammar.

A grammar whose predictive parser has no multiply defined entries is known as LL(1)grammar. LL (1) means left to right scan of the input to generate the left most derivationby using 1 symbol of look ahead (can be extended to k symbol look ahead)

14. **Define parser.**

The parser is that phase of the compiler which takes a token string as input and with thehelp of existing grammar, converts it into the corresponding IntermediateRepresentation (IR). The parser is also known as Syntax Analyzer.

15. What is meant by handle pruning? N/D'2016

An Handle of a string is a sub string that matches the right side of production and whose reduction to the non terminal on the left side of the production represents one step along the reverse of a rightmost derivation.

The process of obtaining rightmost derivation in reverse is known as Handle Pruning.

Consider the grammar: E → E+E/E*E/(E)/id

And the input string id1+id2*id3

The rightmost derivation is :

→ **E**+**E**

→ E+**E*E**

→ E+E***id3**

→ E+**id2***id 3

→ **id1**+id2***id 3

16. What are the errors identified in syntax analyzer?

- Syntax errors are errors in the program text; they may be either lexical or grammatical:

  (a) A lexical error is a mistake in a lexeme, for examples, typing tehn instead of then, or missing off one of the quotes in a literal.

  (b) A grammatical error is a one that violates the (grammatical) rules of the language, for example if x = 7 y := 4 (missing then).

  (c) A syntactic error, such as an arithmetic expression with missing semicolon or unbalanced parenthesis.

17. Write the error recovery actions in parser?

- Panic mode recovery
- Phrase level recovery
- Error Production.
- Global Correction.

18. What do you mean by viable prefixes?

The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. An equivalent definition of a viable prefix is that it is a prefix of a right-sentential form that does not continue past the right end of the rightmost handle of that sentential form.

19. Write the limitations of Recursive decent parser.

- Back tracking is available
- Taking more time and more space
- Not efficient
- Reducing the performance of compiler.

20. List out the conflicts occurred in shift-reduce parsing.

<u>Conflicts in shift-reduce parsing:</u>

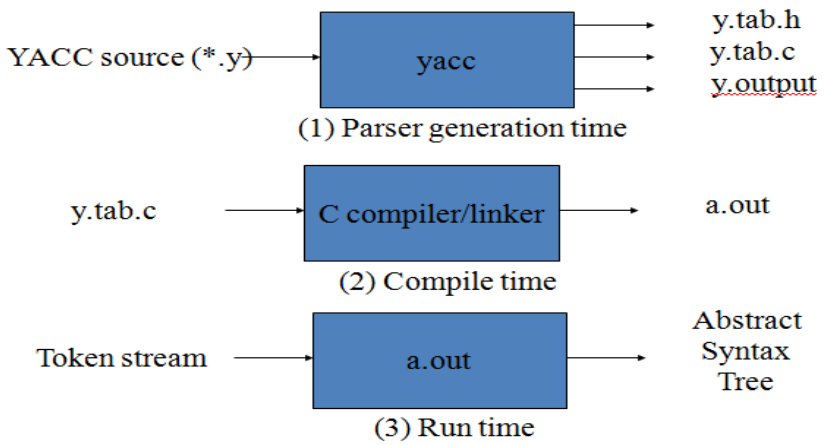
There are two conflicts that occur in shift shift-reduce parsing:

a. **Shift-reduce conflict**: The parser cannot decide whether to shift or to reduce.
b. **Reduce-reduce conflict**: The parser cannot decide which of several reductions to make.

21. Define YACC.

Yacc - Yet Another Compiler-Compiler

- Tool which will produce a parser for a given grammar.
- YACC (Yet Another Compiler Compiler) is a program designed to compile a LALR(1) grammar and to produce the source code of the syntactic analyzer of the language produced by this grammar.



22. What is LL (1) grammar? Give the properties of LL (1) grammar.

L : Scan input from Left to Right
L : Construct a Leftmost Derivation
1 : Use "1" input symbol as lookahead in conjunction with stack to decide on the parsing action
LL(1) grammars == they have no multiply-defined entries in the parsing table.

Properties of LL(1) grammars:

1. Grammar can't be ambiguous or left recursive
2. Grammar is LL(1) $\Leftrightarrow$when $A \rightarrow \alpha \mid \beta$
   a. α & β do not derive strings starting with the same terminal a
   b. Either α or β can derive $\in$, but not both.
   c. If β derives $\in$, then α does not derive any string beginning with a terminal in FOLLOW(A)

The parsing table entries are single entries. So each location has not more than one entry. This type of grammar is called LL(1) grammar.

23. What are the disadvantages of operator precedence parsing?

<u>Advantages of operator precedence parsing:</u>

- It is easy to implement.
- Once an operator precedence relation is made between all pairs of terminals of a grammar , the grammar can be ignored. The grammar is not referred anymore during implementation.

<u>Disadvantages of operator precedence parsing:</u>

- It is hard to handle tokens like the minus sign (-) which has two different precedence.
- Only a small class of grammar can be parsed using operator-precedence parser.

Part-B

1. **Discuss in detail about the role of parser.**

<u>Contents:</u>

- Explain the process of Parser work
- Types of Parser and explain
- Neatly Represent the Diagram

2. **Write an Algorithm and construct SLR Parsing Table for the following context freegrammar. Check whether the string id + id id * is a valid string (AU–April / May 2013)**

   E→E+T | T
   T→T*F | F
   F→F*| a| b

   Contents:
   - Define SLR
   - Steps involved for SLR Parser
   - Neatly Represent the given CFG Grammar
   - Construct the Parsing table
   - Check the Input String with stack implementation

3. **Explain LR Parsing algorithm with example (AU–Nov / Dec 2017)**

   Contents:
   - Define LR
   - Differentiate LL vs LR
   - Explain the Concepts
   - Advantages & Disadvantages

4. **Explain in detail YACC (AU–Nov / Dec 2016)**

   Contents:
   - Define YACC
   - Explain the Concepts of Input and output files
   - Advantages & Disadvantages
   - Example programs

5. **Give the LALR for the given grammar.S->AAA->Aa|b**

   Contents:
   - Define LALR
   - Neatly Represent the given CFG Grammar
   - Construct the Parsing table

Part – C

1. **Construct Stack implementation of shift reduce parsing for the grammarE->E+EE->E*EE->(E)E->id and the input string id1+id2*id3**

   Contents:
   - Explain the Concepts of Shift Reduce parser
   - Neatly Represent the given CFG Grammar
   - Check the Input String with stack implementation

Unit-III

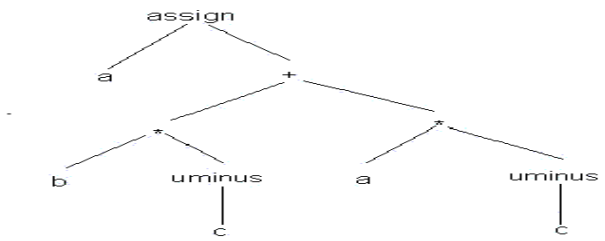
SYNTAX DIRECTED TRANSLATION & INTERMEDIATE CODE GENERATION

Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-Attribute Definitions- Design of predictive translator – Type Systems-Specification of a simple type Checker Equivalence of Type Expressions-Type Conversions. Intermediate Languages: Syntax Tree, Three Address Code, Types and Declarations, Translation of Expressions, Type Checking, Back patching.

Part – A

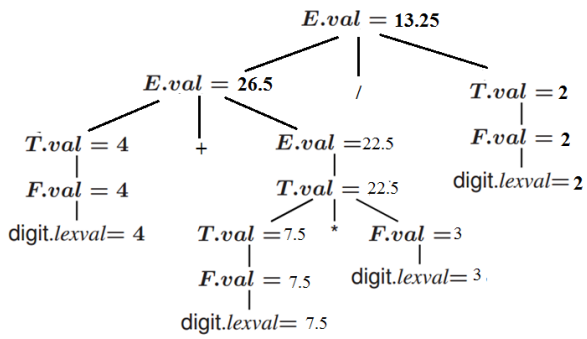
1. **What are the various methods of implementing three address statements? M/J'2013**
   There are three types of intermediate representation:-
   1. Syntax Trees
   2. Postfix notation
   3. Three Address Code

2. **Translate the arithmetic expression a * -(b+c) into syntax tree and postfix notation. N/D'201**



Postfix Notation: a b c uminus * b c uminus * + assign

3. **Construct a decorated parse tree according to the syntax directed definition for the following input statement: (4+7.5*3)/2. A/ M'2015**



4. **Write the 3-address code x=*y ; a=&x, A/ M'2015**

   x:=*y MOV $*R_y$, x

   a:=&x MOV $\&R_x$, a

5. **Write down syntax Directed definition of a simple desk calculator(AU –Nov / Dec 2016)**
   Syntax Directed Definition (SDD) is a kind of abstract specification. The combination of context free grammar and semantic rules

6. **What are synthesized attributes? N/D'2015**
   a. Synthesized Attributes. They are computed from the values of the attributes of the children nodes.
   b. Inherited Attributes. They are computed from the values of the attributes of both the siblings and the parent nodes.

7. **Mention the rules for type checking (AU– April / May 2016)**
   A compiler must check that the source program should follow the syntactic and semantic conventions of the source language and it should also check the type rules of the language.

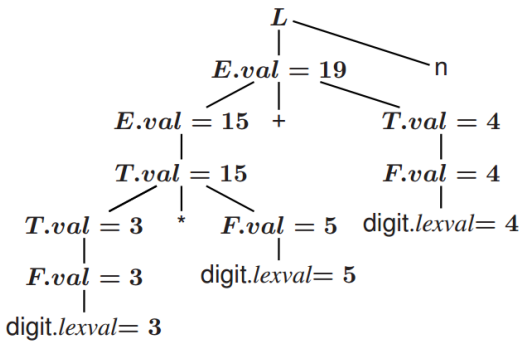
8. **Write down syntax directed definition of a simple desk calculator. N/D'2016**

PRODUCTION	SEMANTIC RULES
L -> E **n**	*print (E.val)*
E -> E1 + T	*E.val:=E1.val+T.val*
E -> T	*E.val:=T.val*
T -> T1 * F	*T.val:=T1.val*F.val*
T -> F	*T.val:=F.val*
F -> (E)	*F.val:=E.val*
F -> **digit**	*F.val=**digit**.lexval*

Syntax-Directed Definition of a simple calculator

9. **Define syntax directed definition.**
   - Syntax Directed Definitions are a generalization of context-free grammars in which:
     1. Grammar symbols have an associated set of Attributes;
     2. Productions are associated with Semantic Rules for computing the values of attributes.
   - Such formalism generates Annotated Parse-Trees where each node of the tree is a record with a field for each attribute (e.g., X.a indicates the attribute a of the grammar symbol X).

10. **Define S-Attributes.**
    S-Attributed Definitions
    Definition : An S-Attributed Definition is a Syntax Directed Definition that uses only synthesized attributes.
    • Evaluation Order. Semantic rules in a S-Attributed Definition can be evaluated by a bottom-up, or Post Order, traversal of the parse-tree.
    • Example. The above arithmetic grammar is an example of an S-Attribute d Definition. The annotated parse-tree for the input 3*5+4n is:



11. Define Dependency Graph.

- Dependency Graphs are the most general technique used to evaluate syntax directed definitions with both synthesized and inherited attributes.
- A Dependency Graph shows the interdependencies among the attributes of the various nodes of a parse-tree.
- There is a node for each attribute;
- If attribute b depends on an attribute c there is a link from the node for c to the node for b (b ← c).
- Dependency Rule: If an attribute b depends from an attribute c, then we need to fire the semantic rule for c first and then the semantic rule for b.

12. Mention the two rules of type checking. NOV/DEC2011

a. A type checker verifies that the type of a construct matches that expected by its context.
b. Type information gathered by a type checker may be needed when code is generated.

13. What is the significance of intermediate code? MAY/JUNE 14, APRIL/MAY 11, APRIL/MAY

- Retargeting is facilitated; a compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.
- A machine-independent code optimizer can be applied to the intermediate representation.

14. What are the functions used to create the nodes of syntax trees?

- Mknode (op, left, right)
- Mkleaf (id,entry)
- Mkleaf (num, val)

15. What are the functions for constructing syntax trees for expressions?

- The construction of a syntax tree for an expression is similar to the translation of the expression into postfix form.
- Each node in a syntax tree can be implemented as a record with several fields.

16. Define back patching. (AU – Nov / Dec 2013)

Back patching is a technique for converting flow-of-control statements into a single pass. During the code generation process, it is the action of filling in blank labels with undetermined data. During bottom-up parsing, back patching is utilized to generate quadruples for boolean expressions.

17. Define type systems.

Type systems allow defining interfaces between different parts of a computer program, and then checking that the parts have been connected in a consistent way. This checking can happen statically (at compile time), dynamically (at run time), or as a combination of both.

18. Define a syntax-directed translation?

Syntax-directed translation specifies the translation of a construct in terms of Attributes associated with its syntactic components. Syntax-directed translation uses a context free grammar to specify the syntactic structure of the input. It is an input- output mapping.

19. Define an attribute. Give the types of an attribute?

An attribute may represent any quantity, with each grammar symbol, it associates a set of attributes and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production. Example: a type, a value, a memory location etc.,

a. Synthesized attributes.

b. Inherited attributes.

20. Compare synthesized attributes and inherited attributes

Synthesized attributes	Inherited attributes
The production must have non-terminalas its head.	The production must have non-terminalas a symbol in its body
It can be evaluated during a singlebottom-up traversal of parse tree.	It can be evaluated during a single top-down and sideways traversal of parsetree.
Synthesized attribute is used by both S-attributed SDT and L-attributed SDT.	Inherited attribute is used by only L-attributed SDT.

Part – B

1. Explain in detail about Specification of a simple type checker. (AU – April / May 2017)

Contents:

- Explain the concept of type checking expression
- Explain in Types checking statement
- Explain in Equivalence of type expression
- Neatly Represent the Diagram

2. **Discuss the following in detail about the Syntax Directed Definitions.Inherited Attributes and Synthesized attributes**

Contents:

- Define SDD
- Steps involved for SDD

- Explain in detail two methods of attributes
- Neatly Represent the given CFG Grammar

3. **Create variants of Syntax tree. Explain in detail about it with suitable examples**
   Contents:
   - Define Syntax Tree
   - Explain the Concepts
   - Explain in details the variants of Tree Structure

4. **State the rules for type checking with example**
   Contents:
   - Define Type checking
   - Explain the Concepts of two types of checking
   - Advantages & Disadvantages

5. **What is Type conversion? What are the two types of type conversion? Formulate therules for the type conversion.**
   Contents:
   - Explain the concept of type conversion or casting
   - Explain the Concepts of two types of conversion
   - Give an Example of each types.

Part– C

1. **Write the Translation scheme for translating assignment statement having scalarvariables and array reference to three - address statements(AU April / May 2013)**
   Contents:
   - Explain the Concepts of Three Address code
   - Write the applications
   - Implementation of Three address code and types
   - Give an Example of each types.

Unit-IV

RUN-TIME ENVIRONMENT AND CODE GENERATION

Runtime Environments – source language issues – Storage organization – Storage Allocation Strategies: Static, Stack and Heap allocation – Parameter Passing-Symbol Tables – Dynamic Storage Allocation – Issues in the Design of a code generator – Basic Blocks and Flow graphs – Design of a simple Code Generator –Optimal Code Generation for Expressions– Dynamic Programming Code Generation.

Part -A

1. **Define Flow Graph. A/M 2012**

   Basic block

   A sequence of consecutive statements which may be entered only at the beginning and when entered are executed in sequence without halt or possibility of branch , are called basic blocks.

   Flow graph

   - The basic block and their successor relationships shown by a directed graph is called a flow graph.
   - The nodes of a flow graph are the basic blocks.

2. **How to perform register assignment for outer loop? A/M 2012**

   If an outer loop L1 contains an inner loop L2 , the names allocated registers in L2 need not be allocated registers in L1 - L2 . Similarly, if we choose to allocate x a register in L2 but not L1 , we must load x on entrance to L2 and store x on exit from L2 . We leave as an exercise the derivation of a criterion for selecting names to be allocated registers in an outer loop L, given that choices have already been made for all loops nested within L.

3. **Give examples of static checks. M/J'2013**

   A compiler must check that the source program follows both the syntactic and semantic conventions of the source language. This checking, called static checking, ensures that certain kinds of programming errors will be detected and reported.

   Examples of static checks include:

   Type checking
   Flow-of-control checks
   Uniqueness checks
   Name-related checks

4. **List out the various storage allocation strategies. N/D'2014**

   1. **Static allocation** – lays out storage for all data objects at compile time
   2. **Stack allocation** – manages the run-time storage as a stack.
   3. **Heap allocation** – allocates and de-allocates storage as needed at run time from a data area known as heap.

5. **What do you mean by binding of names? (AU April / May 2017)**
   Name binding is the process of finding the declaration for each name that is explicitly or implicitly used in a template. The compiler may bind a name in the definition of a template, or it may bind a name at the instantiation of a template.

6. **What is the limitation of static allocation? APRIL/MAY 11**
   1. The size of the data object and constraints on its position in memory must be known at compile time.
   2. Recursive procedures are restricted, because all activations of a procedure use the same bindings for local names.
   3. Data structures cannot be created dynamically, since there is no mechanism for storage allocation at run time.

7. **List the different storage allocation strategies.**
   The strategies are:
   - Static allocation
   - Stack allocation
   - Heap allocation

8. **Brief about the techniques for parameter passing.(AU –Nov / Dec 2013)**
   When using call by value, the compiler adds the R-value of the actual parameters that were passed to the calling procedure to the called procedure's activation record.

9. **What is dynamic scoping?**
   In dynamic scoping a use of non-local variable refers to the non-local data declared in most recently called and still active procedure. Therefore each time new findings are set up for local names called procedure. In dynamic scoping symbol tables can be required at run time.

10. **What is heap allocation?**
    Heap allocation is the most flexible allocation scheme. Allocation and deallocation ofmemory can be done at any time and any place depending upon the user's requirement.Heap allocation is used to allocate memory to the variables dynamically and when thevariables are no more used then claim it back

11. **How the activation record is pushed onto the stack.**
    Activation record is used to manage the information needed by a single execution of aprocedure. An activation record is pushed into the stack when a procedure is called andit is popped when the control returns to the caller function.

12. **What are the properties of optimizing compiler?**
    The source code should be such that it should produce minimum amount of target code.
    There should not be any unreachable code.
    Dead code should be completely removed from source language.
    The optimizing compilers should apply following code improving transformations on source language.
    a. common subexpression elimination

b. dead code elimination
c. code movement
d. strength reduction

13. What are the various ways to pass a parameter in a function?

- Call by value
- Call by reference
- Copy-restore
- Call by name

14. Suggest a suitable approach for computing hash function.

- Using hash function we should obtain exact locations of name in symbol table. The hash function should result in uniform distribution of names in symbol table.
- The hash function should be such that there will be minimum number of collisions. Collision is such a situation where hash function results in same location for storing the names.

15. Give short note about call-by-name?

Call by name, at every reference to a formal parameter in a procedure body the name of the corresponding actual parameter is evaluated. Access is then made to the effective parameter.

16. How parameters are passed to procedures in call-by-value method?

This mechanism transmits values of the parameters of call to the called program. The transfer is one way only and therefore the only way to returned can be the value of a function.

```
Main ( )
{ print (5);
} Int
Void print (int n)
{ printf ("%d", n); }
```

17. Define static allocations and stack allocations

Static allocation is defined as lays out for all data objects at compile time.
Names are bound to storage as a program is compiled, so there is no need for a run time support package.
Stack allocation is defined as process in which manages the run time as a Stack. It is based
on the idea of a control stack; storage is organized as a stack, and activation records are pushed and popped as activations begin and end.

18. Write the grammar for flow-of-control statements?

The following grammar generates the flow-of-control statements, if-then, if- then-else, and while-do statements.

```
S -> if E
then S1
| If E then S1 else S2
```

| While E do S1.

19. What are the control-flow constraints? N/D'2015

- The unnecessary jumps can be eliminated in either the intermediate code or the target code by the following types of peephole optimizations. We can replace the jump sequence

```
goto L1
    ….
L1:  gotoL2
```

by the sequence

```
goto  L2
    ….
L1: goto  L2
```

- If there are now no jumps to L1, then it may be possible to eliminate the statement L1:goto L2 provided it is preceded by an unconditional jump .Similarly, the sequence

```
if a < b  goto L1
    ….
L1: goto L2
```

can be replaced by

```
Ifa < b goto L2
    ….
L1: goto L2
```

20. Write three address code sequence for the assignment statement. M/J'2016

D= (a-b)+(a-c)+(a-c)

temp1:= a-c

temp2:=a-b

temp3:=temp2+temp1

D:=temp3+temp1

Part-B

1. **Explain in detail the following with respect to code generation phase . (AU – Nov /Dec 2016)**

   Contents:

   - Explain the concept of type checking expression
   - Explain in Types checking statement
   - Explain in Equivalence of type expression
   - Neatly Represent the Diagram

2. **What are the different storage allocation strategies?(AU April / May 2017)**

   Contents:

   - Define Storage allocation
   - Write the three Strategies
   - Explain in detail all three strategies
   - Neatly Represent with advantage and disadvantage

3. **Explain in detail about the parameter passing (AU– April / May 2017)**
   Contents:
   - Define Syntax Tree
   - Explain the Concepts
   - Explain in details the variants of Tree Structure

4. **Explain about Runtime storage management.(AU –Nov / Dec 2017)**
   Contents:
   - Define Runtime Environment
   - Explain the Concepts of two types of checking
   - Advantages & Disadvantages

5. **Generate optimal code using Dynamic Programming techniques for the assignmentstatement , X: = (a/b - c) / d**. assume unit instruction costs **(AU Nov / Dec 2013)**
   Contents:
   - Explain the concept of type conversion or casting
   - Explain the Concepts of two types of conversion
   - Give an Example of each types.

Part– C

1. **Discuss the Various issues in Design of code generator (AU –April / May 2017)**
   Contents:
   - Explain the Concepts of Three Address code
   - Write the applications
   - Implementation of Three address code and types
   - Give an Example of each types.

Unit-V

CODE OPTIMIZATION

Principal Sources of Optimization – Peep-hole optimization – DAG- Optimization of Basic Blocks – Global Data Flow Analysis – Efficient Data Flow Algorithm – Recent trends in Compiler Design

Part -A

1. **List out two properties of reducible flow graph. A/M 2012**
   - Reducible flow graphs are special flow graphs, for which several code optimization transformations are especially easy to perform, loops are unambiguously defined, dominators can be easily calculated.
   - Data flow analysis problems can also be solved efficiently.

2. **List the advantage and application of DAG. N/D 2012, M/J'2013**
   - We can automatically detect common sub expressions.
   - We can determine the statements that compute the values, which could be used outside the block.
   - We can determine which identifiers have their values used in the block.

3. **What are the uses of register and address descriptor in code generator? N/D 2012**
   - A register descriptor keeps track of what is currently in each register.
   - An address descriptor keeps track of the location where the current value of the name can be found at run time.

4. **Define Live variable. N/D 2012**

   The framework is similar to reaching definitions, except that the transfer function runs backward. A variable is live at the beginning of a block if it is either used before definition in the block or is live at the end of the block and not redefined in the block.

5. **What is data flow analysis ? N/D 2012, N/D'2014**

   Data-flow analysis" refers to a body of techniques that derive information about the flow of data along program execution paths. For example, one way to implement global common sub expression elimination requires us to determine whether two textually identical expressions evaluate to the same value along any possible execution path of the program.

6. **Define Basic blocks and Flow graph. M/J'2013, N/D'2014**

   Basic block

   A sequence of consecutive statements which may be entered only at the beginning and when entered are executed in sequence without halt or possibility of branch , are called basic blocks.

   Flow graph
   - The basic block and their successor relationships shown by a directed graph is called a flow graph.
   - The nodes of a flow graph are the basic blocks.

7. **What is constant folding ? M/J'2013**
   - We can eliminate both the test and printing from the object code. More generally, deducing at compile time that the value of an expression is a constant and using the constant instead is known

as constant folding.

- One advantage of copy propagation is that it often turns the copy statement into dead code.
- ✓ For example,
  a=3.14157/2 can be replaced by
  a=1.570 thereby eliminating a division operation.

8. What are the properties of optimizing compiler? M/J'2013, N/D'2013, M/J'2016

1. Transformation must preserve the meaning of programs.
2. Transformation must, on the average, speed up the programs by a measurable amount
3. A Transformation must be worth the effort.

9. What is the Next-Use information? N/D'2013

If the name in a register is no longer needed, then we remove the name from the register and the register can be used to store some other names.

Input: Basic block B of three-address statements

Output: At each statement i: x= y op z, we attach to i the liveliness and next-uses of x, y and z.

Method: We start at the last statement of B and scan backwards.

1. Attach to statement i the information currently found in the symbol table regarding the next-use and liveliness of x, y and z.
2. In the symbol table, set x to "not live" and "no next use".
3. In the symbol table, set y and z to "live", and next-uses of y and z to i.

10. Define Loop unrolling with an example. N/D'2013

In region-based scheduling, the boundary of a loop iteration is a barrier to code motion. Operations from one iteration cannot overlap with those from another. One simple but highly effective technique to mitigate this problem is to unroll the loop a small number of times before code scheduling. A for-loop such as

```
for (i = 0; i < N; i++) {
SCi) ;
}
```

can be written as in Fig . 10.16(a) . Similarly, a repeat-loop such as

```
repeat
S;
unt il C
```

11. Name the techniques in loop optimization. M/J'2014

Three techniques are important for loop optimization:

code motion, which moves code outside a loop;

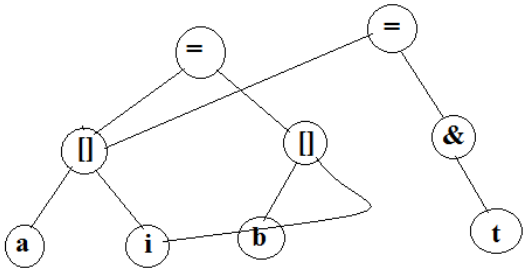
Induction-variable elimination, which we apply to replace variables from inner loop.

Reduction in strength, which replaces and expensive operation by a cheaper one, such as a multiplication by an addition.

12. How would you represent the dummy blocks with no statements indicated in global data flow analysis. M/J'2014

- We say that the beginning points of the dummy blocks at the entry and exit of a statement's region are the beginning and end points, respectively, of the statement. The equations are inductive, or syntax-directed, definition of the sets in[S], out[S], gen[S], and kill[S] for all statements S.

13. Draw DAG to represent a[i] =b[i]; a[i] = &t. N/D'2014



14. What role does the target machine play on the code generation phase of the compiler? A/ M'2015

Familiarity with the target machine and its instruction set is a prerequisite for designing a good code generator.
The target computer is a byte-addressable machine with 4 bytes to a word.
It has *n* general-purpose registers, R0, R1, . . . , Rn-1.
It has two-address instructions of the form:
op source, destination
where, *op* is an op-code, and *source* and *destination* are data fields.
It has the following op-codes :

MOV (move *source* to *destination*)
ADD (add *source* to *destination*)
SUB (subtract *source* from *destination*)

The *source* and *destination* of an instruction are specified by combining registers and memory locations with address modes.

15. Generate code for the following C statement assuming three registers are available: x=a/(b+c)-d*(e+f) A/ M'2015

Three address code:

t1= e+f
t2=b+c
t3=a/t2
t4=d*t1
t5=t3-t4
x=t5

Assembly Code (Target Code):

```
MOV e, R1
ADD f,R1
MOV b,R2
ADD c,R2
```

```
MOV a,R3
DIV R3,R2
MUL d,R1
SUB R1,R2
MOV R2,x
```

16. Write the algorithm that orders the DAG nodes for generating optimal target code. A/ M'2015

The heuristic ordering algorithm attempts to make the evaluation of a node immediately follow the evaluation of its leftmost argument.
The algorithm shown below produces the ordering in reverse.

Algorithm:

```
1.  while unlisted interior nodes remain do begin
2.      select an unlisted node n, all of whose parents have been listed;
3.      list n;
4.      while the leftmost child m of n has no unlisted parents and is not a leaf
            do begin
5.              list m;
6.              n : = m
        end
        end
```

17. Define Dead code elimination. N/D'2015

A variable is live at a point in a program if its value can be used subsequently; otherwise that values never get used. While the programmer is unlikely to introduce any dead code intentionally, it may appear as the result of previous transformations. An optimization can be done by eliminating dead code.

Example:

```
i=0;
if(i=1)
{
 a=b+5;
}
```

Here, 'if' statement is dead code because this condition will never get satisfied.

18. What are the issues in the design of code generator? N/D'2015

The following issues arise during the code generation phase :

1. Input to code generator
2. Target program
3. Memory management
4. Instruction selection
5. Register allocation
6. Evaluation order

19. What are the global common sub expressions? N/D'2015

An occurrence of an expression E is called a common sub-expression if E was previously computed, and the values of variables in E have not changed since the previous computation. We can avoid recomputing the expression if we can use the previously computed value.

For example

t_1: =4*i t_2: =a
[t1]
t_3: =4*j t_4:=4*i
t_5: =n
t_6: =b [t_4] +t_5

The above code can be optimized using the common sub-expression elimination as

t_1: =4*i
t_2: =a [t_1] t_3:
=4*j
t_5: =n
t_6: =b [t_1] +t_5

The common sub expression t_4: =4*i is eliminated as its computation is already in t_1. And value of i is not been changed from definition to use.

20. What is DAG? M/J'2016

- A DAG for a basic block is a **directed acyclic graph** with the following labels on nodes:
  Leaves are labeled by unique identifiers, either variable names or constants.
  Interior nodes are labeled by an operator symbol.
  Nodes are also optionally given a sequence of identifiers for labels to store the computed values.
- DAGs are useful data structures for implementing transformations on basic blocks.
- It gives a picture of how the value computed by a statement is used in subsequent statements.
- It provides a good way of determining common sub - expressions.

21. What are the characteristics of peephole optimization.

- A simple but effective technique for improving the target code is peephole optimization, a method for trying to improving the performance of the target program by examining a short sequence of target instructions (called the peephole) and replacing these instructions by a shorter or faster sequence, whenever possible.
- Characteristic of peephole optimizations:
  - Redundant-instructions elimination
  - Flow-of-control optimizations
  - Algebraic simplifications
  - Use of machine idioms
  - Unreachable Code

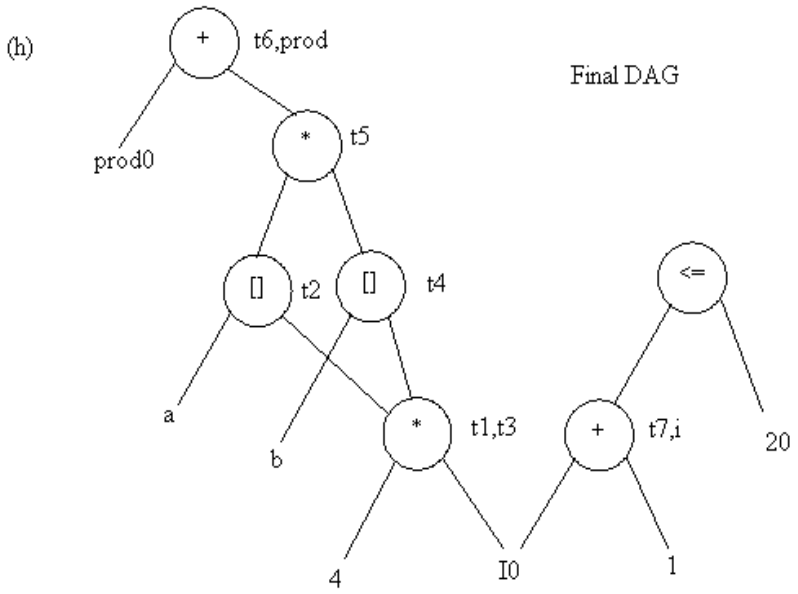
Part -B

1. **Explain in detail about Global Data flow analysis of structural programs.(16) NOV/DEC2012, M/J'2013, N/D'2013, M/J'2014, N/D'2015, N/D'2016**

   Contents
   - Define Global Data flow analysis
   - Explain Points and Paths:
   - Draw Diagram Neatly
   - Write Data-flow analysis of structured programs:

2. **For the flow graph shown below, write the three address code and construct the DAG. (8) M/J'2013**

   (1) t1 = 4*i
   (2) t2 = a[t1]
   (3) t3 = 4 *i
   (4) t4 = b[t2]
   (5) t5= t2 * t4
   (6) t6 = prod + t5
   (7) prod = t6
   (8) t7 = i+1
   (9) i = t7
   (10) if i<=20 goto (1)



3. **Write short notes on structure preserving transformation of basic blocks.(8) NOV/DEC2013, N/D'2014, N/D'2015**

   Contents
   - Structure-Preserving Transformation
   - Dead-code elimination
   - Renaming temporary variables
   - Interchange of statements

4. **Construct DAG and three address statement for the following C Program. N/D'2013, N/D'2014**

```
i=1;
s=0;
while(i<=10)
{
s=s+a[i][i]
i=i+1;
}
```

   Contents
   - Define DAG
   - Write the three address code
   - Draw the DAG disgram

5. **Define a directed acyclic graph. Construct a DAG and write the sequence of instructions for the expression a+a*(b-c)+(b-c)*d. (16)MAY/JUNE 14**

   Contents
   - Construct a DAG and Equivalence of Instructions
   - Three address code for given ecpression
   - Write the Instruction code

6. **Explain the steps carried out for generating code from DAGs with an example.(8) N/D'2015**

   Contents
   - Rearranging the order
   - Write Generated code sequence for basic block
   - Write the algorithm
   - Give a suitable example

PART C

1. **Discuss about the following:**

   i). Copy Propagation ii) Dead-code Elimination and iii) Code motion(6) NOV/DEC2012, MAY/JUNE 2012

   Contents

   - Copy Propagation
   - Dead-Code Eliminations
   - Constant folding
   - Loop Optimizations
   - Code Motion

2. **Explain peephole optimization. (8) NOV/DEC2013 ,NOV/DEC2012, MAY/JUNE 2012**

   Contents

   - Write the Reduntant Loads And Stores
   - Write the Unreachable Code
     - Elimination Of Common Subexpressions
     - Elimination Of Dead Code
     - Reduction In Strength
     - Use Of Machine Idioms
   - Getting Better Performance