INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

INFORMATION TECHNOLOGY

DEFINITION AND TERMINOLGY

Department	**INFORMATION TECHNOLOGY**				
Course Title	**COMPILER DESIGN**				
Course Code	ACSC40				
Program	B.Tech				
Semester	V				
Course Type	Core				
Regulation	UG-20				
Course Structure	Theory			Practical	
	Lecture	Tutorials	Credits	Laboratory	Credits
	3	1	4	-	-
Course Coordinator	Dr.U Sivaji, Associate Professor				

COURSE OBJECTIVES:

The students will try to learn:

I	The process of translating a high-level language to machine code required for compiler construction.
II	The Software tools and techniques used in compiler construction such as lexical analyser and parser generators.
III	The data structures used in compiler construction such as abstract syntax trees, symbol tables, three-address code, and stack machines.
IV	The deeper insights into the syntax and semantic aspects of programming languages, dynamic memory allocation and code generation .

COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO 1	**Summarize** phases of a compiler in the construction of language processors.	Understand
CO 2	**Make use of** finite automata for designing a lexical analyzer for a specific programming language constructs.	Apply
CO 3	**Choose** top down, bottom up parsing methods for developing a parser with representation of a parse table or tree.	Apply

CO 4	**Outline** syntax directed translations, intermediate forms for performing semantic analysis along with code generation.	Understand
CO 5	**Relate** symbol table, type checking and storage allocation strategies used in run-time environment.	Understand
CO 6	**Select** code optimization techniques on intermediate code form for generating target code.	Apply

DEFINITION AND TERMINOLOGY:

S.No	DEFINITION	CO's
	MODULE I	
	INTRODUCTION TO COMPILERS	
1	**Define Compiler** A compiler is a computer program that translates computer code written in one programming language into another programming language.	CO 1
2	**Define interpreter** An interpreter is a computer program that is used to directly execute program instructions written using one of the many high-level programming languages.	CO 1
3	**Define Translator** A translator is a program that takes as input a program written in one language and produces as output a program in another language.	CO 1
4	**What are the types of translators** There are three types a) Interpreter b) Compiler c) Preprocessor	CO 1
5	**What is a Token** Token is a sequence of characters that can be treated as a single logical entity.	CO 2
6	**Define Pattern** A set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token.	CO 2
7	**List Some Compilers** Aa) Ada compilers b) ALGOL compilers c) BASIC compilers d) C compilers e) C compilers f) C++ compilers g) COBOL compilers h) Java compilers	CO 1
8	**Define Lexeme** A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.	CO 1
9	**Define front end** The front end analyzes the source program and produces intermediate code	CO 1

10	**What are specifications of tokens**	CO 1
	There are 3 specifications of tokens: a) Strings b) Language c) Regular expression	
11	**Define Language**	CO 2
	A language is a set of strings, chosen form some $\sum^*$ or A language is a subset of $\sum^*$.A language which can be formed over $\sum$ can be Finite or Infinite. Language that contains strings over $\sum = \{$ a, b $\}$ are $\{\epsilon$, a, b, aa, ab $\}$.	
12	**What is a regular expression.**	CO 2
	Regular expression is an Algebraic way to represent a language.	
13	**What do you mean by automata?**	CO 2
	Automation is defined as a system where information is transmitted and used for performing some functions without direct participation of man.	
14	**What is the difference between compiler and interpreter**	CO 1
	A compiler converts the high level instruction into machine language while an interpreter converts the high level instruction into an intermediate form.	
15	**What is the function of parser in compiler?**	CO 1
	a)It checks if the tokens from lexical analyzer, occur in pattern that are permitted by the specification for the source language. b)It also imposes on tokens a tree-like structure that is used by the sub-sequent phases of the compiler.	
16	**Define Lexical Analyzer**	CO 2
	Lexical analyzer (the "lexer") parses individual symbols from the source code file into tokens.	
17	**what is meant by Phase of a compiler**	CO 1
	A phase is a logically interrelated operation that takes source program in one representation and produces output in another representation.	
18	**What is a Loader**	CO 1
	A loader is a program that places programs into memory and prepares them for execution.	
19	**List the properties under which regular languages are not closed**	CO 2
	Subset, superset, infinite union and infinite intersection.	
20	**What is the Regular expression for strings without two consecutive one's 11?**	CO 2
	Regular expression for all strings which may begin with either 0 or 1 and without consecutive ones $(1+\epsilon)(0+01)^*$	
21	**Define Identifier**	CO 2
	Identifiers are the set or string of letters and digits beginning with a letter.	

22	**Define pass** The transversal of a compiler through the entire program is known as a Pass.	CO 1
23	**Define single pass compiler** A one-pass compiler is a compiler that passes through the parts of each compilation unit only once, immediately translating each part into its final machine code.	CO 1
24	**Define multi pass compiler** A multi-pass compiler which converts the program into one or more intermediate representations in steps between source code and machine code, and which reprocesses the entire compilation unit in each sequential pass.	CO 1
25	**what is meant by Synthesis Phase** The synthesis phase generates the target program with the help of intermediate source code representation and symbol table.	CO 1
26	**Define Regular expression for strings consists ether two consecutive zero's 00 or two consecutive one's 11?** $(0+1)^*(00+11)(0+1)^*$	CO 2
. 27	**What is the regular language for the regular expression a+** { a, aa, aaa, ... } or $\{ a^n \mid n \geq 1\}$	CO 2
28	**What is the analysis model of a compiler** Analysis phase reads the source program and splits it into multiple tokens and constructs the intermediate representation of the source program. And also checks and indicates the syntax and semantic errors of a source program	CO 1
29	**What is the back end of the compiler?** The back end synthesizes the target program from the intermediate code	CO 1
	MODULE II	
	SYNTAX ANALYSIS	
1	**Define Parser.** A parser takes input in the form of sequence of tokens and produces output in the form of parse tree.	CO 3
2	**List the types of parsers.** There are two types of parsers a) Topdown parsing b) Bottom parsing	CO 3
3	**What is bottom up parsing.** In the bottom up parsing, the parsing starts with the input symbol and construct the parse tree up to the start symbol by tracing out the rightmost derivations of string in reverse.	CO 3
4	**What is top-down parsing.**	CO 3

	Top-down parsing constructs parse tree for the input string, starting from root node and creating the nodes of parse tree in pre-order. It is done by leftmost derivation for an input string.	
5	**What are the tasks performed by parser.**	CO 3
	a)Helps you to detect all types of Syntax errors b)Find the position at which error has occurred c)Clear and accurate description of the error. d)Recovery from an error to continue and find further errors in the code. e)The parse must reject invalid texts by reporting syntax errors	
6	**What are the common errors occur in parser.**	CO 3
	Common Errors that occur in Parsing a)Lexical: Name of an incorrectly typed identifier b)Syntactical: unbalanced parenthesis or a missing semicolon c)Semantical: incompatible value assignment d)Logical: Infinite loop and not reachable code	
7	**Define handle.**	CO 3
	VA handle of a string is a substring that matches the right side of a 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.	
8	**What is Handle pruning.**	CO 3
	If A $\rightarrow \beta$ is a production then reducing β to A by the given production is called handle pruning i.e., removing the children of A from the parse tree. A rightmost derivation in reverse can be obtained by handle pruning.	
9	**Define shift reduce parser.**	CO 3
	Shift Reduce parsing is a bottom-up parsing that reduces a string w to the start symbol of grammar. It scans and parses the input text in one forward pass without backtracking.	
10	**List out the actions of shift reduce parser.**	CO 3
	A shift-reduce parser can make four possible actions a) shift b) reduce c) accept d) error.	
11	**What are the conflicts occurs in shift reduce parser**	CO 3
	There are two conflicts occur in shift-Reduce parser. a)Shift-Reduce conflict b)Reduce-Reduce conflict	
12	**What is shift in compiler? .**	CO 3
	The input pointer advances to the next input symbol by the shift step and the next input symbol is known as shifted symbol and the symbol is pushed upon stack. The shifted symbol is considered as a single node of the parse tree.	
13	**What is reduce in compiler?**	CO 3

	When a complete grammar rules RHS is replaced by LHS it is termed as reduce-step. The stack performs a pop function which facilitates in popping off the handle and replacing with the LHS non-terminal symbol.	
14	**What do you mean by LR parser?**	CO 3
	The LR parser is a non-recursive, shift-reduce, bottom-up parser. It uses a wide class of context-free grammar which makes it the most efficient syntax analysis technique.	
15	**What is LR(K) parser.**	CO 3
	LR parsers are also known as LR(k) parsers, where L stands for left-to-right scanning of the input stream; R stands for the construction of right-most derivation in reverse, and k denotes the number of lookahead symbols to make decisions.	
16	**What are the types of LR Parsers.**	CO 3
	LR parsing is divided into four parts: a) LR (0) parsing, b) SLR parsing, c) CLR parsing d) LALR parsing.	
17	**What is LR algorithm.**	CO 3
	The LR algorithm requires stack, input, output and parsing table. In all type of LR parsing, input, output and stack are same but parsing table is different.	
18	**What is the use of Input buffer.**	CO 3
	Input buffer is used to indicate end of input and it contains the string to be parsed followed by a dollar Symbol.	
19	**What is parsing table.**	CO 3
	Parsing table is a two dimensional array. It contains two parts: a) Action part b) GoTo part.	
20	**What is Augmented Grammar.**	CO 3
	Augmented grammar G‘ will be generated if we add one more production in the given grammar G. It helps the parser to identify when to stop the parsing and announce the acceptance of the input. for example S‘$\rightarrow$ S , S $\rightarrow$ AA , A $\rightarrow$ aA | b	
21	**What is SLR (1) Parser.**	CO 3
	a) It is Simple LR Parser b)It works on smallest class of grammar c)Few number of states, hence very small table c)Simple and fast construction	
22	**What is LR (1) Parser.**	CO 3
	a)It is LR Parser b)It works on complete set of LR(1) Grammar c)Generates large table and large number of states d)Slow construction	
23	**What is LALR (1) Parser.**	CO 3
	a) It is Look-Ahead LR Parser b) Works on intermediate size of grammar c) Number of states are same as in SLR(1)	
24	**What are the differences between LL and LR**	CO 3

	LL: a) Leftmost derivation is done b) Parse tree is built top-down. c) Starts with the root non- terminal on the stack. LR: a) Rightmost derivation is done. b) Parse tree is built bottom-up. c) Ends with the root non-terminal on the stack.	
25	**What is CLR(1) parsing.**	CO 3
	CLR refers to canonical lookahead. CLR parsing use the canonical collection of LR (1) items to build the CLR (1) parsing table. CLR (1) parsing table produces the more number of states as compare to the SLR (1) parsing.	
26	**What is LR(1) item.**	CO 3
	LR (1) item is a collection of LR (0) items and a look ahead symbol. LR (1) item = LR (0) item + look ahead The look ahead is used to determine that where we place the final item. The look ahead always add $ symbol for the argument production.	
27	**What are the main functions of yacc?**	CO 3
	a)YACC stands for Yet Another Compiler Compiler. b)YACC provides a tool to produce a parser for a given grammar. c)YACC is a program designed to compile a LALR (1) grammar.	
28	**What is panic-mode error recovery.**	CO 3
	In the case when the parser encounters an error, this mode ignores the rest of the statement and not process input from erroneous input to delimiter, like a semi-colon. This is a simple error recovery method.	
29	**What is phrase-level error recovery.**	CO 3
	Compiler corrects the program by inserting or deleting tokens. This allows it to proceed to parse from where it was. It performs correction on the remaining input. It can replace a prefix of the remaining input with some string this helps the parser to continue the process.	
30	**What is canonical collection of LR(0) items.**	CO 3
	An LR (0) item is a production G with dot at some position on the right side of the production. LR(0) items is useful to indicate that how much of the input has been scanned up to a given point in the process of parsing.	
	MODULE III	
	SYNTAX-DIRECTED TRANSLATION AND INTERMEDIATE CODE GENERATION	
1	**Define Syntax Directed Translation**	CO 4
	Syntax Directed Translations are augmented rules to the grammar that facilitate semantic analysis.	
2	**Define Syntax tree**	CO 4
	A Syntax tree is a graphical depiction of a string derivation.	
3	**Define Attribute grammar**	CO 4

	Attribute grammar is a special form of context-free grammar where some additional information (attributes) is appended to one or more of its non-terminals in order to provide context-sensitive information.	
4	**Define Synthesized attributes.**	CO 4
	These attributes get values from the attribute values of their child nodes.	
5	**Define Inherited attributes.**	CO 4
	Inherited attributes can take values from parent and/or siblings.	
6	**List the types of Attribute Grammar.**	CO 4
	Synthesized attributes and inherited attributes.	
7	**What is S-attributed SDT ?**	CO 4
	If an SDT uses only synthesized attributes, it is called as S-attributed SDT.	
8	**What is L-attributed SDT ?**	CO 4
	This form of SDT uses both synthesized and inherited attributes with restriction of not taking values from right siblings.	
9	**Define abstract syntax tree?**	CO 4
	An abstract syntax tree, or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language.	
10	**What is polish notation?**	CO 4
	Prefix notation is also known as Polish Notation. In this operator is prefixed to operands.	
11	**What is three address code?**	CO 4
	Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code.It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler.	
12	**List the Intermediate forms of source programs**	CO 4
	Abstract syntax tree, polish notation and three address code	
13	**List the types of three address codes.**	CO 4
	a)Quadruple b)Triples c) Indirect Triples	
14	**What is quadruple in three address code?**	CO 4
	Each instruction in quadruples presentation is divided into four fields: operator, arg1, arg2, and result.	
15	**What is triple in three address code?**	CO 4
	Each instruction in triples presentation has three fields : op, arg1, and arg2.The results of respective sub-expressions are denoted by the position of expression.	
16	**What is Indirect Triples notation of three address code?**	CO 4

	This representation is an enhancement over triples representation. It uses pointers instead of position to store results.	
17	**What is Reversed polish notation?**	CO 4
	Postfix notation is also known as Reversed Polish Notation. In this the operator is postfixed to the operands.	
18	**What is postfix notation of (a+b)*c**	CO 4
	ab+c*	
19	**What is postfix notation of a*(b+c)**	CO 4
	abc+*	
20	**What is postfix notation of (a+b)*(c+d)**	CO 4
	ab+cd+*	
21	**What is Annotated Parse Tree?**	CO 4
	A parse tree showing the values of attributes at each node is called an Annotated parse tree.	
22	**What is High Level IR?**	CO 4
	High-level intermediate code representation is very close to the source language itself. They can be easily generated from the source code and we can easily apply code modifications to enhance performance.	
23	**What is Low Level IR?**	CO 4
	Low-level intermediate code representation is close to the target machine, which makes it suitable for register and memory allocation, instruction set selection, etc.	
24	**What is three address code for the expression a*b+c**	CO 4
	t1= a*b, t2=t1+c	
25	**What is three address code for the expression (a+b)*c**	CO 4
	t1= a+b, t2=t1*c	
26	**What is the SDT for following statement E=E+T**	CO 4
	{E.value:=E.value+T.value}	
27	**What is the SDT for following statement T=T*F**	CO 4
	{T.value:=T.value*F.value}	
28	**List the functions used for constructing syntax trees.**	CO 4
	a) mknode(op,left,right) b) mkleaf(id,entry) c) mkleaf(num,val)	
29	**What is the SDT for following statement F=digit**	CO 4
	{F.value:=digit.lexicalvalue}	
30	**What is the SDT for following statement T=T/F**	CO 4
	{T.value:=T.value/F.value}	
	MODULE IV	
	TYPE CHECKING AND RUN TIME ENVIRONMENT	
1	**Define Type Checking?**	CO 5
	Type checking is simply testing for type errors in given program, either by the compiler or during program execution.	
2	**What is Static checking?**	CO 5

	Static checking includes the syntax checks performed by the parser and semantic checks such as type checks, flow-of- control checks, uniqueness checks, and name-related checks.	
3	**What is Dynamic checking?**	CO 5
	Dynamic type checking is the process of verifying the type safety of a program at runtime. Common dynamically-typed languages include Groovy, JavaScript, Lisp, Objective-C, PHP, Prolog, Python, Ruby, Small talk .	
4	**Explain Type Expression?**	CO 5
	Type expressions are built from basic types and constructors, a natural concept of equivalence between two type expressions is structural equivalence. i.e., two expressions are either the same basic type or formed by applying the same constructor to structurally equivalent types.	
5	**Define Function Overloading?**	CO 5
	Function overloading or method overloading is the ability to create multiple functions of the same name with different implementations.	
6	**Explain structural equivalence of type expression?**	CO 5
	Type expression are built from basic types and constructors ,a natural concept of equivalence between two type expressions is structural equivalence i.e. two expressions are either the same basic type or formed by applying the same constructor to same equivalent type.	
7	**List any two uses of type checking?**	CO 5
	Depending on Language Type checker can prevent 1)Application of a function to wrong number of arguments 2)use of undeclared variables in expressions	
8	**What is flow of control checks?**	CO 5
	Statements that cause flow of control to leave a construct must have some place to which to transfer the flow of control.	
9	**What are Type systems?**	CO 5
	A type system is a set of rules for assigning type expressions to the syntactic constructs of a program.	
10	**Define strongly typed language?**	CO 5
	Type Inference are rules that determine the type of a language construct based on how it is used.	
11	**What is meant by Type Inference?**	CO 5
	Type inference defines the type of an expression based on the types of its constituent parts Perl, Python, and Lisp are dynamically typed.	
12	**What is Dynamic Typed language?Mention examples**	CO 5
	A dynamically typed language is one in which some of the constructs of a language can only be typed at run time. Perl, Python, and Lisp are dynamically typed.	

13	**What is meant by Operator overloading?** Operator overloading is a technique by which operators used in a programming language are implemented in user-defined types with customized logic that is based on the types of arguments passed	CO 5
14	**Define Activation trees.** A program consist of procedures, a procedure definition is a declaration that, in its simplest form, associates an identifier (procedure name) with a statement (body of the procedure). Each execution of procedure is referred to as an activation of the procedure.	CO 5
15	**What are the properties of Activation trees.** a)Each node represents an activation of a procedure. b)The root shows the activation of the main function. c)The node for procedure 'x' is the parent of node for procedure 'y' if and only if the control flows from procedure x to procedure y.	CO 5
16	**What is Control stack.** Control stack or runtime stack is used to keep track of the live procedure activations i.e. the procedures whose execution have not been completed. A procedure name is pushed on to the stack when it is called (activation begins) and it is popped when it returns (activation ends).	CO 5
17	**What are the various fields of activation record?** Activation records consist of a)local variables b)Temporary values c)Machine status d)Access link e)Control link f)Return value	CO 5
18	**List the various storage allocation strategies.** There are three allocation strategies. a) Static Storage Allocation b) Stack Storage Allocation c) Heap Storage Allocation	CO 5
19	**What is Static storage allocation.** Any program if we create memory at compile time, memory will be created in the static area and memory is created only once.	CO 5
20	**What is Stack storage allocation** Storage is organized as a stack and activation records are pushed and popped as activation begin and end respectively. Locals are contained in activation records so they are bound to fresh storage in each activation.	CO 5
21	**What is Heap storage allocation** Heap allocation is used to dynamically allocate memory to the variables and claim it back when the variables are no more required. Memory allocation and deallocation can be done at any time and at any place depending on the requirement of the user.	CO 5
22	**What is Symbol table.**	CO 5

	Symbol table is an important data structure created and maintained by compilers in order to store information about various entities such as variable names, function names, objects, classes, interfaces, etc.	
23	**List the various implementation of symbol table.**	CO 5
	A symbol table can be implemented in one of the following ways: a)Linear list b)Binary search tree c)Hash table	
24	**What are the operations in symbol table.**	CO 5
	There are two operations. a)insert() b)lookup()	
25	**What is lookup() operation.**	CO 5
	Lookup() operation is used to search a name in the symbol table to determine: a) if the symbol exists in the table. b) if the name is used in the scope. c) if the symbol is initialized. d) if the symbol declared multiple times.	
26	**What are the types of symbol table? .**	CO 5
	A compiler maintains two types of symbol tables: a global symbol table which can be accessed by all the procedures and scope symbol tables that are created for each scope in the program.	
27	**Contrast between static and Dynamic allocation.**	CO 5
	Static: a)Memory is allocated before the execution of the program begins. b)Implemented using stacks. c) In this type of allocation Memory cannot be resized after the initial allocation. Dynamic: a)Memory is allocated during the execution of the program. b)Implemented using heap. c)In this type of allocation Memory can be dynamically expanded and shrunk as necessary.	
28	**What are the control access links in the activation record.**	CO 5
	Control link points to activation record of the caller.Access Link is used to refer to non-local data held in other activation records.	
MODULE V		
CODE OPTIMIZATION AND CODE GENERATOR		
1	**Define Code Optimization.**	CO 6
	The code optimization in the synthesis phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources (i.e. CPU, Memory) so that faster-running machine code will result.	
2	**What are the main objectives of code optimization.**	CO 6
	a)The optimization must be correct, it must not, in any way, change the meaning of the program. b)Optimization should increase the speed and performance of the program. c)The compilation time must be kept reasonable. d)The optimization process should not delay the overall compiling process.	
3	**What are the specific characteristics of maraging steels?List the types of code optimization**	CO 6

	The optimization process can be broadly classified into two types a)Machine-Independent Optimization b)Machine-Dependent Optimization	
4	**What are the code optimization techniques.**	CO 6
	Techniques are: a)Compile Time Evaluation b)Common sub-expression elimination c)Dead Code Elimination d)Code Movement e)Strength Reduction	
5	**Define Machine independent optimization**	CO 6
	In this optimization, the compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations.	
6	**Define Machine dependent optimization**	CO 5
	Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references.	
7	**List the different loop optimization techniques.**	CO 6
	There are three techniques: a)Code motion b)Induction-variable elimination c)Reduction in strength	
8	**What are function preserving transformations.**	CO 6
	The transformations are Sources of Optimization a)Common Subexpression Elimination b)Copy Propagation c)Dead-Code Elimination d) Constant Folding	
9	**What is common sub expression elimination.**	CO 6
	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.	
10	**Define copy propagation.**	CO 6
	Assignments of the form f : = g called copy statements, or copies for short. The idea behind the copy-propagation transformation is to use g for f, whenever possible after the copy statement f: = g. Copy propagation means use of one variable instead of another.	
11	**What is Dead code elimination.**	CO 6
	A variable is live at a point in a program if its value can be used subsequently; otherwise, it is dead at that point. A related idea is dead or useless code, statements that compute values that never get used.	
12	**What is constant folding.**	CO 6
	Deducing at compile time that the value of an expression is a constant and using the constant instead is known as constant folding.	
13	**What is loop optimization.**	CO 6

	In loops, especially in the inner loops, programs tend to spend the bulk of their time. The running time of a program may be improved if the number of instructions in an inner loop is decreased, even if we increase the amount of code outside that loop.	
14	**Define code motion.**	CO 6
	Code motion is used to decrease the amount of code in loop. This transformation takes a statement or expression which can be moved outside the loop body without affecting the semantics of the program	
15	**What is meant by reduction in strength?**	CO 6
	Strength reduction is used to replace the expensive operation by the cheaper once on the target machine.Addition of a constant is cheaper than a multiplication. So we can replace multiplication with an addition within the loop.	
16	**What is Basic Block.**	CO 6
	A number of sequences are included in the source codes, which are executed in sequence and are termed as the basic blocks of the code. When the first instruction is executed, all the instructions of the same basic block are executed in the sequence of appearance by not losing the program flow control.	
17	**What are the characteristics of basic block.**	CO 6
	a)They do not contain any kind of jump statements in them. b)There is no possibility of branching or getting halt in the middle. c)All the statements execute in the same order they appear. d)They do not lose lose the flow control of the program.	
18	**Define Flow graph.**	CO 6
	A flow graph is a directed graph with flow control information added to the basic blocks.	
19	**How to determine basic block.**	CO 6
	a)All the statements that follow the leader (including the leader) till the next leader appears form one basic block. b)The first statement of the code is called as the first leader. c)The block containing the first leader is called as Initial block.	
20	**How to determine leader statement in basic block.**	CO 6
	a)First statement of the code. b)Statement that is a target of the conditional or unconditional goto statement. c)Statement that appears immediately after a goto statement.	
21	**Define Induction variable.**	CO 6
	A variable x is an Induction Variable of a loop if every time the variable x changes values, it is incremented or decremented by some constant	
22	**List the types of basic block optimization.**	CO 6
	There are two type of basic block optimization. a)Structure-Preserving Transformations b)Algebraic Transformations	

23	**What are the Structure-Preserving Transformation on basic blocks.**	CO 6
	a)Common sub-expression elimination b)Dead code elimination c)Renaming of temporary variables d)Interchange of two independent adjacent statements	
24	**What is an example of algebraic transformations .**	CO 6
	Algebraic transformations are used to convert the set of expressions computed by a basic block into an algebraically equivalent set. For example, the exponential expression x: = y * * 2 can be replaced with x: = y * y	
25	**Why is it called directed acyclic graph?**	CO 6
	Directed Acyclic Graph (DAG) is a tool that depicts the structure of basic blocks, helps to see the flow of values flowing among the basic blocks, and offers optimization too. DAG provides easy transformation on basic blocks.	
26	**What is meant by code generation?**	CO 6
	This is the final phase of compilation. which takes input as a optimized code and convert in to machine/assembly language.	
27	**List the code generator descriptors.**	CO 6
	There are two descriptors: a)Register descriptor b)Address descriptor	
28	**Define peephole optimization.**	CO 6
	This optimization technique works locally on the source code to transform it into an optimized code. By locally, we mean a small portion of the code block at hand. These methods can be applied on intermediate codes as well as on target codes.	
29	**What is the Unreachable code.**	CO 6
	Unreachable code is a part of the program code that is never accessed because of programming constructs. Programmers may have accidently written a piece of code that can never be reached	
30	**How registers are allocated and assigned in order to generate good code?**	CO 6
	Selecting the set of variables that will reside in registers at each point in the program picking the specific register that a variable will reside in it.	

Course Coordinator: **HOD IT**
Dr.U Sivaji