There will be two papers in the subject:

Paper I: Theory……..... 3 hours….70 marks

Paper II: Practical……. 3 hours….30 marks

PAPER I –THEORY – 70 MARKS

SECTION A

1. Boolean Algebra

a) Propositional logic, well formed formulae, truth values and interpretation of well formed formulae (wff), truth tables, satisfiable, unsatisfiable and valid formulae. Equivalence laws and their use in simplifying wffs.

*Propositional variables; the common logical connectives (~ (not)(negation), ∧ (and)(conjunction), ∨ (or)(disjunction), ⇒ (implication), ⇔ (biconditional); definition of a well-formed formula (wff); `representation of simple word problems as wff (this can be used for motivation); the values **true** and **false**; interpretation of a wff; truth tables; satisfiable, unsatisfiable and valid formulae.*

*Equivalence laws: commutativity of ∧, ∨; associativity of ∧, ∨; distributivity; De Morgan's laws; law of implication ($p \Rightarrow q \equiv \sim p \vee q$); law of biconditional ($(p \Leftrightarrow q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$); identity ($p \equiv p$); law of negation ($\sim(\sim p) \equiv p$); law of excluded middle ($p \vee \sim p \equiv$ **true**); law of contradiction ($p \wedge \sim p \equiv$ **false**); tautology and contingency simplification rules for ∧, ∨. Converse, inverse and contra positive. Chain rule, Modus ponens.*

(b) Binary valued quantities; basic postulates of Boolean algebra; operations AND, OR and NOT; truth tables.

(c) Basic theorems of Boolean algebra (e.g. duality, idempotence, commutativity, associativity, distributivity, operations with 0 and 1, complements, absorption, involution); De Morgan's theorem and its applications; reducing Boolean expressions to sum of products and product of sums forms; Karnaugh maps (up to four variables).

Verify the laws of Boolean algebra using truth tables. Inputs, outputs for circuits like half and full adders, majority circuit etc., SOP and POS representation; Maxterms & Minterms, Canonical and Cardinal representation, reduction using Karnaugh maps and Boolean algebra.

2. Computer Hardware

a) Elementary logic gates (NOT, AND, OR, NAND, NOR, XOR, XNOR) and their use in circuits.
b) Applications of Boolean algebra and logic gates to half adders, full adders, encoders, decoders, multiplexers, NAND, NOR as universal gates.

Show the correspondence between Boolean methods and the corresponding switching circuits or gates. Show that NAND and NOR gates are universal by converting some circuits to purely NAND or NOR gates.

SECTION B

The programming element in the syllabus (Sections B and C) is aimed at algorithmic problem solving and **not** merely rote learning of Java syntax. The Java version used should be 5.0 or later. For programming, the students can use any text editor and the javac and java programs or any other development environment: for example, BlueJ, Eclipse, NetBeans etc. BlueJ is strongly recommended for its simplicity, ease of use and because it is very well suited for an 'objects first' approach.

3. Implementation of algorithms to solve problems

The students are required to do lab assignments in the computer lab concurrently with the lectures. Programming assignments should be done such that each major topic is covered in at least one assignment. Assignment problems should be designed so that they are sufficiently challenging. Students must do algorithm design, address correctness issues, implement and execute the algorithm in Java and debug where necessary.

Self-explanatory.

4. Programming in Java (Review of Class XI Sections B and C)

Note that items 4 to 13 should be introduced almost simultaneously along with classes and their definitions.

While reviewing, ensure that new higher order problems are solved using these constructs.

5. Objects

(a) Objects as data (attributes) + behaviour (methods); object as an instance of a class. Constructors.

(b) Analysis of some real-world programming examples in terms of objects and classes.

(c) Basic input/output using Scanner and Printer classes from JDK; input/output exceptions. Tokens in an input stream, concept of whitespace, extracting tokens from an input stream (String Tokenizer class).

6. Primitive values, Wrapper classes, Types and casting

Primitive values and types: byte, int, short, long, float, double, boolean, char. Corresponding wrapper classes for each primitive type. Class as type of the object. Class as mechanism for user defined types. Changing types through user defined casting and automatic type coercion for some primitive types.

7. Variables, Expressions

Variables as names for values; named constants (final), expressions (arithmetic and logical) and their evaluation (operators, associativity, precedence). Assignment operation; difference between left hand side and right hand side of assignment.

8. Statements, Scope

Statements; conditional (if, if else, if else if, switch case, ternary operator), looping (for, while, do while, continue, break); grouping statements in blocks, scope and visibility of variables.

9. Methods

Methods (as abstractions for complex user defined operations on objects), formal arguments and actual arguments in methods; different behaviour of primitive and object arguments. Static method and variables. The **this** Operator. Examples of algorithmic problem solving using methods (number problems, finding roots of algebraic equations etc.).

10. Arrays, Strings

Structured data types – arrays (single and multidimensional), address calculations, strings. Example algorithms that use structured data types (e.g. searching, finding maximum/minimum, sorting techniques, solving systems of linear equations, substring, concatenation, length, access to char in string, etc.).

Storing many data elements of the same type requires structured data types – like arrays. Access in arrays is constant time and does not depend on the number of elements. Address calculation (row major and column major), Sorting techniques (bubble, selection, insertion). Structured data types can be defined by classes – String. Introduce the Java library String class and the basic operations on strings (accessing individual characters, various substring operations, concatenation, replacement, index of operations). The class StringBuffer should be introduced for those applications that involve heavy manipulation of strings.

11. Recursion

Concept of recursion, simple recursive methods (e.g. factorial, GCD, binary search, conversion of representations of numbers between different bases).

Many problems can be solved very elegantly by observing that the solution can be composed of solutions to 'smaller' versions of the same problem with the base version having a known simple solution. Recursion can be initially motivated by using recursive equations to define certain methods. These definitions are fairly obvious and are easy to understand. The definitions can be directly converted to a program. Emphasize that any recursion must have a base case. Otherwise, the computation can go into an infinite loop.

The tower of Hanoi is a very good example of how recursion gives a very simple and elegant solution where as non-recursive solutions are quite complex.

SECTION C

Inheritance, Interface, Polymorphism, Data structures, Computational complexity

12. Inheritance, Interfaces and Polymorphism

a) Inheritance; super and derived classes; member access in derived classes; redefinition of variables and methods in subclasses; abstract classes; class Object; protected visibility. Subclass polymorphism and dynamic binding.

Emphasize inheritance as a mechanism to reuse a class by extending it. Inheritance should not normally be used just to reuse some methods defined in a class but only when there is a genuine specialization (or subclass) relationship between objects of the super class and that of the derived class.

b) Interfaces in Java; implementing interfaces through a class; interfaces for user defined implementation of behaviour.

Motivation for interface: often when creating reusable classes some parts of the exact implementation can only be provided by the final end user. For example, in a class that sorts records of different types the exact comparison operation can only be provided by the end user. Since only he/she knows which field(s) will be used for doing the comparison and whether sorting should be in ascending or descending order be given by the user of the class.

Emphasize the difference between the Java language construct interface and the word interface often used to describe the set of method prototypes of a class.

13. Data structures

a) Basic data structures (stack, queue, circular queue, dequeue); implementation directly through classes; definition through an interface and multiple implementations by implementing the interface. Conversion of Infix to Prefix and Postfix notations.

Basic algorithms and programs using the above data structures.

Data structures should be defined as abstract data types with a well-defined interface (it is instructive to define them using the Java interface construct).

b) Single linked list (Algorithm and programming), binary trees, tree traversals (Conceptual).

The following should be covered for each data structure:

Linked List (single): insertion, deletion, reversal, extracting an element or a sublist, checking emptiness.

Binary trees: apart from the definition the following concepts should be covered: root, internal nodes, external nodes (leaves), height (tree, node), depth (tree, node), level, size, degree, siblings, sub tree, completeness, balancing, traversals (pre, post and in-order).

14. Complexity and Big O Notation

Concrete computational complexity; concept of input size; estimating complexity in terms of methods; importance of dominant term; constants, best, average and worst case.

Big O notation for computational complexity; analysis of complexity of example algorithms using the big O notation (e.g. Various searching and sorting algorithms, algorithm for solution of linear equations etc.).

PAPER II: PRACTICAL – 30 MARKS

This paper of three hours' duration will be evaluated by the Visiting Examiner appointed locally and approved by CISCE.

The paper shall consist of three programming problems from which a candidate has to attempt any one.