COMPUTER SCIENCE (868)

Aims (Conceptual)

(1) To understand algorithmic problem solving using data abstractions, functional and procedural abstractions, and object based and object-oriented abstractions.

(2) To understand: (a) how computers represent, store and process data at different levels of abstraction that mediate between the machine and the algorithmic problem solving level and (b) how they communicate with the outside world.

(3) To create awareness of ethical issues related to computing and to promote safe, ethical behavior.

(4) To make students aware of future trends in computing.

Aims (Skills)

To devise algorithmic solutions to problems and to be able to code, validate, document, execute and debug the solution using the Java programming system.

CLASS XII

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

Paper I shall be of 3 hours duration and be divided into two parts.

Part I (20 marks): *This part will consist of compulsory short answer questions, testing knowledge, application and skills relating to the entire syllabus.*

Part II (50 marks): *This part will be divided into three Sections, A, B and C. Candidates will be required to answer **two** questions out of **three** from Section A (each carrying 10 marks) and **two** questions out of **three** from Section B (each carrying 10 marks) and **two** questions out of **three** from Section C (each carrying 5 marks). Therefore, a total of **six** questions are to be answered in Part II.*

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 ⇒ q ≡ ~p ∨ q); law of biconditional ((p ⇔ q) ≡ (p ⇒ q) ∧ (q ⇒ p)); identity (p ≡ p); law of negation (~ (~p) ≡ p); law of excluded middle (p ∨ ~p ≡ **true**); law of contradiction (p∧~p ≡ **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 multi-dimensional), 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 the Council.

The paper shall consist of three programming problems from which a candidate has to attempt any one. The practical consists of the two parts:

1. Planning Session
2. Examination Session

The total time to be spent on the Planning session and the Examination session is three hours. A maximum of 90 minutes is permitted for the Planning session and 90 minutes for the Examination session.

Candidates are to be permitted to proceed to the Examination Session <u>only after</u> the 90 minutes of the Planning Session are over.

Planning Session

The candidates will be required to prepare an algorithm and a hand written Java program to solve the problem.

Examination Session

The program handed in at the end of the Planning session shall be returned to the candidates. The candidates will be required to key-in and execute the Java program on seen and unseen inputs individually on the Computer and show execution to the Visiting Examiner. A printout of the program listing including output results should be attached to the answer script containing the algorithm and handwritten program. This should be returned to the examiner. The program should be sufficiently documented so that the algorithm, representation and development process is clear from reading the program. Large differences between the planned program and the printout will result in loss of marks.

Teachers should maintain a record of all the assignments done as part of the practical work through the year and give it due credit at the time of cumulative evaluation at the end of the year. Students are expected to do a **minimum of twenty-five** assignments for the year.

EVALUATION:

Marks (out of a total of **30**) should be distributed as given below:

Continuous Evaluation

Candidates will be required to submit a work file containing the practical work related to programming assignments done during the year.

Programming assignments done throughout the year (Internal Evaluation)	10 marks
Programming assignments done throughout the year (Visiting Examiner)	5 marks

Terminal Evaluation

Solution to programming problem on the computer	15 Marks

Marks should be given for choice of algorithm and implementation strategy, documentation, correct output on known inputs mentioned in the question paper, correct output for unknown inputs available only to the examiner.

NOTE:
Algorithm should be expressed clearly using any standard scheme such as a pseudo code.

EQUIPMENT

There should be enough computers to provide for a teaching schedule where at least three-fourths of the time available is used for programming.

Schools should have equipment/platforms such that all the software required for practical work runs properly, i.e. it should run at acceptable speeds.

Since hardware and software evolve and change very rapidly, the schools may have to upgrade them as required.

Following are the recommended specifications as of now:

The Facilities:

- A lecture cum demonstration room with a MULTIMEDIA PROJECTOR/ an LCD and O.H.P. attached to the computer.
- A white board with white board markers should be available.
- A fully equipped Computer Laboratory that allows one computer per student.
- Internet connection for accessing the World Wide Web and email facility.
- The computers should have a minimum of 1 GB RAM and a P IV or higher processor. The basic requirement is that it should run the operating system and Java programming system (Java compiler, Java runtime environment, Java development environment) at acceptable speeds.
- Good Quality printers.

Software:

- Any suitable Operating System can be used.
- JDK 6 or later.
- Documentation for the JDK version being used.
- A suitable text editor. A development environment with a debugger is preferred (e.g. BlueJ, Eclipse, NetBeans). BlueJ is recommended for its ease of use and simplicity.

SAMPLE TABLE FOR PRACTICAL WORK

S. No.	Unique Identification Number (Unique ID) of the candidate	Assessment of Practical File		Assessment of the Practical Examination (To be evaluated by the Visiting Examiner only)				TOTAL MARKS (Total Marks are to be added and entered by the Visiting Examiner)
		Internal Evaluation 10 Marks	Visiting Examiner 5 Marks	Algorithm	Java Program with internal Documentation	Hard Copy (printout)	Output	
				3 Marks	7 Marks	2 Marks	3 Marks	30 Marks
1.								
2.								
3.								
4.								
5.								
6.								
7.								
8.								
9.								
10.								

Name of the Visiting Examiner:_____

Signature: _____

Date:_____