COMPUTER SCIENCE (868)

Aims

1. To enable students to comprehend basic concepts and practices for problem solving using computers.
2. To develop an understanding of how computers store and process data.
3. To develop the ability to describe the major components of computer hardware, their functions and interaction.
4. To develop the ability to analyze applications as systems of interacting objects.
5. To enable students to solve problems using algorithmic techniques using the approach of classes and objects.
6. To develop the ability to devise, test, code, debug, documents and validate programs to implement various algorithms.
7. To develop an appreciation of the implications of computer use in everyday life in contemporary society.
8. To create an awareness of ethical issues related to computing.

CLASS XI

There will be two papers in the subject:

Paper I: Theory -	3 hours ….100 marks
Paper II: Practical -	3 hours ….100 marks

PAPER I -THEORY

Paper 1 shall be of 3 hours duration and be divided into two parts.

Part I (30 marks): *This part will consist of compulsory short answer questions, testing knowledge, application and skills relating to elementary/fundamental aspects of the entire syllabus.*

Part II (70 marks): *This part will be divided into two Sections, A and B. Candidates are required to answer* ***three*** *questions out of* ***five*** *from Section A and* ***four*** *questions out of* ***six*** *from Section B. Each question in this part shall carry 10 marks.*

SECTION A

Basic Computer hardware and software:

1. Review of Number systems (decimal, binary, hexadecimal) and arithmetic.

   (a) Number systems, base of a number system - decimal, binary, octal, hexadecimal representation, conversion between various representations, character representations (ASCII, ISCII, Unicode).

   (b) Representations for integers, real numbers, limitations of finite representations.

   (c) Internal structure of a computer, a simple decimal load and store computer and its machine language, instruction format, registers, program counter, instruction register; register addressing modes, instruction cycle, assembly language for the same computer, simple algorithms in assembly language.

2. Review of basic computer hardware and software; basic concepts about operating systems, file systems.

   CPU, the clock, cache memory, primary memory, secondary memory, input and output devices, communication devices (the aim is not to describe/discuss an exhaustive list of devices but to understand what parts are present in a typical computer and what is the function of each part).

   The boot process, operating system (resource management and command processor), file system.

   i) Boot process, operating systems - resource management, command processing.

   ii) Directories, files and hierarchical file system.

   iii) Programming languages (machine language, assembly language, high level language).

iv) Compilers and interpreters.

v) Application software.

3. Propositional logic, well formed formulae, truth values and interpretation of well formed formulae, truth tables, basic ideas and results about consistency and completeness of propositional logic (no proofs).
4. Logic and hardware, basic gates (AND, NOT, OR) and their universality, other gates (NAND, NOR, XOR); inverter, half adder, full adder.
5. Control unit, system clock.
6. Memory - construction of a memory bit using a flip-flop, D-flip-flop and its use in constructing registers.
7. Memory organization and access; parity; memory hierarchy - cache, primary memory, secondary memory.
8. Instruction set and its representation; address, addressing and addressing modes; instruction cycle; machine language; stored program computer (program as data).
9. Assembly language syntax and semantics (the assembly language should be for the computer above); assembler and the assembly process; simple assembly language programs.

SECTION B

The programming element in the syllabus is aimed at problem solving and **not** on merely rote learning of the commands and syntax of particular programming languages. Students have the option to use either C++ or Java in order to implement the high-level language concepts and algorithms mentioned in the syllabus and to use them for solving problems. Care must be taken that 'standard and recent' versions of the languages are used on the computer. Development environments such as SUN Forte, Eclipse, BlueJ, Borland C++ or Microsoft Visual C++, GNU C++ on Linux could be used.

1. Introduction to High Level Language Concepts

a) Review of programming in Classes IX and X

i) Primitive data types supported by the language (integers, floating point numbers, characters, booleans, etc. - will depend on the language), variables (and their declaration - based on language), assignment, difference between the left-hand side and right-hand side of an assignment.

ii) Expressions - arithmetic and logical, evaluation of expressions, type of an expression (depends on language). Operators, associativity and precedence of operators.

iii) Statements, blocks (where relevant), scope and visibility of variables.

iv) Conditional statements (if and if-then-else), switch, break, default.

v) Loops (for, while-do, do-while).

vi) Simple input/output using standard input/output.

vii) Functions/subroutines as procedural abstractions. Using functions/subroutines in programs.

viii) Arguments and argument passing in functions/subroutines.

ix) Scope of variables.

x) Structured types, arrays as an example of a structured type. Use of arrays in sorting and searching. Two-dimensional arrays. Use of two-dimensional arrays to represent matrices. Matrix arithmetic using arrays. Use of arrays to solve linear equations (Gauss elimination method).

xi) Review of input/output using standard input and standard output. Input/output using sequential files. Opening, closing files. Creating and deleting files. Formatting output. Concept of a token and separator. Extracting tokens from the input.

xii) Characters, ASCII representation, strings as a composite data type; functions on strings (e.g. length, substring, concatenate, equality, accessing individual characters in a string, inserting a string in another string at a given location).

xiii) Simple type casting for primitive types; inter-conversion between character/string types and numeric types.

xiv) Distinction between compile time and run time errors. Run time errors due to finite representations - overflow, underflow. Other run time errors.

xv) Basic ideas about linking, loading, execution.

b) Objects and classes.

c) Analysis of some real world applications in terms of objects and operations on objects.

d) Interface or public contract of a class.

e) Classes as types.

f) Examples of problem solving using objects.

g) Encapsulation and visibility (private and public).

h) Static variables and functions.

i) Errors - compile time and run time. Exceptions and exception handling.

j) Simple data structures - stack, queue, deque (should be defined as classes); use of these data structures in problem solving.

2. 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 non-trivial and make the student do both algorithm design as well use the programming language to implement the algorithm. Some sample problems are given at the end of the syllabus.

3. Social context of computing and ethical issues

a) Intellectual property and corresponding laws and rights, software as intellectual property.

b) Software patents, copyrights, and trademarks, software licensing and piracy.

c) Free software foundation and its position on software, open source software.

d) Privacy, email etiquette.

PAPER II-PRACTICAL

Paper II (Practical) will be evaluated internally by the school. The evaluation shall consist of a 3-hour practical examination at the end of the year and evaluation of all assignments done throughout the year.

The terminal examination of three hours duration shall consist of three programming tasks from which the candidates shall have to attempt any one. Candidates will be required to write a program in C++/Java on the computer to solve the task and give adequate written documentation to explain the development process.

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.

Marks (out of a total of 100) will be distributed as given below:

Continuous Evaluation

Programming in C++/Java through the year - 40 marks

Terminal Evaluation

Program implementation in C++/Java on the computer	- 40 marks
Documentation, Viva-Voce	- 20 marks

Given below are some sample problems that can be taken up. The list however, is only suggestive. These problems are of different levels of difficulty. They are indicative of the kinds of problems students can try in their labs. Most problems require the student to devise an algorithm or to design suitable classes to model the problem.

Suggested sample problems:

1. Implement a Calculator class that models a hand held calculator. It should have (at least) the following functionality: addition, subtraction, multiplication, integer division, remainder, unary minus, enter, clear.

2. A student has a name, roll number, class in which studying, home address and a date of birth. First design a suitable class for Date. Write constructors and *get* and *set* functions. Then design a class for student. Write constructors, *get* and *set* functions and a function to calculate the age of a student (use today's date in the function). The age should be returned as a triple (year, month, day). You will have to define a class for Triple and return an object of this type as the age.

3. Write a class *Convert* with methods as follows:

   a) takes 4 arguments representing miles, yards, feet and inches and convert them into kilometers, meters and centimeters.

   b) takes an argument representing degrees Fahrenheit and converts it to degrees centigrade.

   c) a kilobyte is interpreted in two ways: some times it is 1000 bytes (actually correct), but often (and traditionally) it is 2^{10} which is 1024. Similar discrepancies arise for mega, giga, tera and peta (each is 1000 (or 2^{10}) times the previous one). The function should take the 10^3 (standard kilo) and give the equivalent value using 2^{10} as a kilo for all the above.

4. Older computers used Binary Coded Decimals (BCD) to represent integers. In this each digit is represented by using 4 bits. So 0 is 0000, 1 is 0001, 2 is 0010, ... 9 is 1001. The remaining 4 bit patterns, namely 1010 onwards are illegal. So 32 bits can store upto 8 digits. To represent a negative number the sign is also encoded using 4 bits. However, we are interested only in positive BCD numbers. For example assuming we have 32 bits the number 123 will be represented as (0000) 5 times followed by 000100100011. Define a class *BCD* with a single constructor, which takes an *integer* argument and creates its BCD equivalent. Write methods for adding, subtracting BCD numbers. Also, write methods to check for the relations <, > and == between two BCD numbers.

5. n is a perfect number if the sum of all the factors of the number (including 1) excluding itself is n. For example:

   6 = 1+2+3

   28=1+2+4+7+14

   n is a prime number if it is divisible only by 1 and itself.

   Define a class called NumberProblems which has the following functions:

   int nthPrime(int n) - which returns the nth prime number. So

   nthPrime(1) = 2 (remember 1 is not considered a prime number)

   nthPrime(3) = 5

   nthPrime(6) = 13

   Write another function:

   void perfectNosBelow(int n) - which first prints out the nth prime number then prints out all perfect numbers less than the nth prime number. Each perfect number should be printed on a single line along with its factors (see below). So for example the output from *perfectNosBelow(5)* will be:

   5th prime number=11

   6= (1,2,3)

CLASS XII

There will be two papers in the subject:

Paper I: Theory- 3 hours …100 marks

Paper II: Practical- 3 hours …100 marks

PAPER I-THEORY

Paper 1 shall be of 3 hours duration and be divided into two parts.

Part I (30 marks): *This part will consist of compulsory short answer questions, testing knowledge, application and skills relating to elementary/fundamental aspects of the entire syllabus.*

Part II (70 marks): *This part will be divided into two Sections, A and B. Candidates are required to answer* ***three*** *questions out of* ***five*** *from Section A and* ***four*** *questions out of* ***six*** *from Section B. Each question in this part shall carry 10 marks.*

SECTION A

1. Boolean Algebra

(i) Propositional logic, well formed formulae, truth values and interpretation of well formed formulae, truth tables, basic ideas and results about consistency and completeness of prepositional logic (no proofs).

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

(iii) Basic theorems of Boolean algebra; principle of duality; idempotent law; commutative law; associative law; distributive law; operations with 0, 1 and complements; absorption law; 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).

2. Computer Hardware

(i) Elementary logic gates (NOT, AND, OR, NAND, NOR, XOR, XNOR) and their uses in circuits.

(ii) Applications of Boolean algebra and logic gates to half adders, full adders, encoders, decoders, multiplexers, use of NAND, NOR as universal gates.

SECTION B

The programming element in the syllabus is aimed at problem solving and **not** on merely rote learning of the commands and syntax of particular programming languages. Students have the option to use either C++ or Java in order to implement the high-level language concepts and algorithms mentioned in the syllabus and to use them for solving problems. Care must be taken that 'standard and recent' versions of the languages are used on the computer- it is recommended that students mention the version of the language being used while writing answers in order to avoid ambiguity. For example, development environments such as SUN Forte, Eclipse, BlueJ, Borland C++ or Microsoft Visual C++, GNU C++ on Linux could be used.

1. Programming in C++/Java

(i) Review of programming and algorithms from Class XI

a) Primitive data types supported by the language (integers, floating point numbers, characters, booleans etc. - will depend on the language), variables (and their declaration - based on language), assignment, difference between the left-hand side and right-hand side of an assignment.

b) Expressions - arithmetic and logical, evaluation of expressions, type of an expression (depends on language). Operators, associativity and precedence of operators.

c) Statements, blocks (where relevant), scope and visibility of variables.

d) Conditional statements (if and if-then-else), switch, break, default.

e) Loops (for, while-do, do-while).

f) Simple input/output using standard input/output.

g) Functions/subroutines as procedural abstractions. Using functions/subroutines in programs.

h) Arguments and argument passing in functions/subroutines.

i) Scope of variables.

j) Structured types, arrays as an example of a structured type. Use of arrays in sorting and searching. Two-dimensional arrays. Use of two-dimensional arrays to represent matrices. Matrix arithmetic using arrays. Use of arrays to solve linear equations (Gauss elimination method).

k) Review of input/output using standard input and standard output from class IX. Input/output using sequential files. Opening, closing files. Creating and deleting files. Formatting output. Concept of a token and separator. Extracting tokens from the input.

l) Characters, ASCII representation, strings as a composite data type; functions on strings (e.g. length, substring, concatenate, equality, accessing individual characters in a string, inserting a string in another string at a given location)

m) Simple type casting for primitive types; inter-conversion between character/string types and numeric types.

n) Distinction between compile time and run time errors. Run time errors due to finite representations - overflow, underflow. Other run time errors.

o) Basic ideas about linking, loading, execution.

p) Objects and classes.

q) Analysis of some real world applications in terms of objects and operations on objects.

r) Interface or public contract of a class.

s) Classes as types.

t) Examples of problem solving using objects.

u) Encapsulation and visibility (private and public).

v) Static variables and functions.

w) Errors - compile time and run time. Exceptions and exception handling.

x) Simple data structures - stack, queue, deque (should be defined as classes); use of these data structures in problem solving.

(ii) Recursion and recursive functions.

(iii) Recursive data structures - lists, binary trees, tree traversals, binary search tree.

(iv) Problem solving using recursive data structures.

(v) Basic concept of inheritance (only single inheritance should be used), base and derived classes.

(vi) Member access in derived classes.

(vii) Redefinition of member variables and member functions.

(viii) Object construction in the presence of inheritance.

(ix) Libraries and use of library classes (details will vary with the language used - container class library can be used as an example).

5. Implementation of algorithms to solve problems

A list of suggested algorithms and general problems is given at the end of this syllabus. These are of an indicative nature to the type of problems that may be set for the students and that should be used in Classes XI and XII. The students will be expected to know the algorithmic solution to solve a particular problem and then be able to implement the solution using either C++ or Java.

PAPER II-PRACTICAL

Paper 2 (Practical) of three hours duration will be evaluated by the teacher and a Visiting Examiner appointed locally and approved by the Council. The practical will consist of two parts: Planning Session and Examination Session. ***On completion of Planning Session, students may proceed to the Examination Session.***

1. Planning Session

Candidates will be required to choose one question from those set in the question paper for the practical examination and prepare an algorithm and a hand written program in C++/Java and other relevant documentation to solve the problem.

2. Examination Session

The computer program handed in at the end of the planning session shall be returned to the candidates. Candidates will then be required to key-in and run their programs individually on the computer and submit the printout results to the Visiting Examiner.

Marks (out of a total of 100) 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 in C++/Java through the year (Internal evaluation)	- 10 marks
Programming in C++/Java through the year (Visiting Examiner)	- 10 marks

Terminal Evaluation

Program implementation in C++/Java on the computer	- 80 marks

(which includes preparing the algorithm, writing and testing of the programme, execution and viva voce.)

Given below are some sample problems that can be taken up. The list however, is only suggestive. These problems are of different levels of difficulty. They are indicative of the kinds of problems students can try in their labs. Most problems require the student to devise an algorithm or to design suitable classes to model the problem.

Suggested sample problems:

1. Define class Point to model points in the X-Y plane. Define functions to translate a point along the X and Y axes respectively. Define a function that calculates the distance from another point.

2. Now use the Point class to define a Rectangle class where a rectangle is modeled by specifying all the four corners of the rectangle. Write separate functions to:

   i) Calculate the perimeter and area of the rectangle.

   ii) Translate the rectangle along the X-axis, Y-axis and X-Y axes (simultaneously).

   iii) Rotate the rectangle by an angle (in degrees) around an axis passing through the center of the rectangle and passing through the X-Y plane.

   iv) Check if the rectangle intersects (i.e. overlaps partially or completely) another rectangle. It should return true if there is an overlap and false otherwise. An overlap of rectangles is defined as: there is at least one point in common between the two rectangles.

3. We want to build the basic backbone of a simple railway reservation system. The following classes are required to model the system: Person, PersonGroup, Date, Train, RailReservationRecord. You have to invent the requisite attributes and functions for the various classes based on a description of the functionality and constraints of RailReservationRecord object given below:

   a) Reservation is made from a start station to a destination station on a single train, in a particular class, for a particular date, for a maximum of 6 persons. So we are not taking care of journeys requiring two or more trains.

   b) Reservation can be made up to a maximum of 60 days and up to a minimum of 1 day before the date of departure.

c) Reservations already made can be canceled for the entire group of persons or for a subset of the group of persons in the reservation record.

d) Assume that the trains have infinite capacity. So you need not make checks to see whether space is available or not.

e) For reserving accommodation each passenger's name, gender and age is required.

f) Each RailReservationRecord object must have a unique reference number/string for access purposes.

You can use classes you may have defined in some of your earlier labs. You can also invent more classes than the ones listed above if you think they are necessary. All your classes should have proper access declarators private, public, etc.

4. We want to model the following situation using a class hierarchy. Shapes can be closed or open. Closed shapes can be one of: polygon, circle or ellipse. A polygon is a triangle, a quadrilateral or an n-gon (that is, an n sided polygon where n>4). Triangles are isosceles, equilateral or scalene. A closed shape is represented by an array of points. The shape is realized by joining the points in the array, by a straight line, in order of increasing index. Finally, the last point is joined to the first point by a straight line thus closing the figure.

   There are three operations common to all shapes: *perimeter*, *area* and *is Convex*. The function *perimeter* calculates and returns the perimeter of the shape, *area* similarly, calculates and returns the area of the shape and *is Convex* returns true if the shape *is convex* and false otherwise.

   Define the class hierarchy above. Define the functions in the right classes in the hierarchy so that over riding is minimal. The design should be such that objects can be created only for those classes that are at the leaves of the hierarchy tree (that is, classes which do not have any direct sub classes).

5. Write a program (as a, possibly static, method of a class) which takes a string argument and prints out all permutations of the string with each permutation being printed out only once if characters are not repeated in the string. For example, if the string argument is "abc" one possible printout is -

   abc acb bca bac cab cba (the order in which the permutations are printed is not important). Your program should use a recursive function.

6. A popular children's puzzle is the 15-puzzle. This puzzle has a frame that can hold 16 tiles. It has tiles numbered from 1 to 15 and one empty space. The goal is to slide the tiles using the empty space such that all tiles are arranged in ascending order row wise. For example if 0 represents the empty space a little trial and error shows that the following initial configuration can be taken to the goal configuration:

1	2	3	4
5	6	0	7
9	10	12	8
13	14	11	15

   Write a program which, given an initial configuration prints out the moves which will take the puzzle to the goal configuration.

   Note that random initial configurations may not always have a solution. (A much harder problem is to determine if given an arbitrary initial configuration whether it is solvable - that is the goal configuration is reachable).

EQUIPMENT

There should be enough computer systems 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 shall need 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 128 MB RAM and a PIII or Equivalent Processor.
- Good Quality printers.

Software:

- Any suitable Operating System can be used.

The criteria used in the selection of software should be:

- It should have a good user interface so that the beginners may learn to use it easily.
- It should be used widely and be easily available.
- The material related to the software should be abundantly available.

The schools should ensure that latest versions of software are used.