Department of Computer Science

University of Delhi

Curriculum

Bachelor in Computer Science (Honors)

under UGCF 2022

Approved in
PG Committee meeting held on May 4, 2022
Faculty of Mathematical Sciences meeting held on May 25, 2022

Table of Contents

1. Preamble
2. Discipline Specific Graduate Attributes
3. Table of Core Courses
4. Syllabi of Core Courses

Preamble

The new curriculum of the four year undergraduate program under NEP, called as UGCF, for computer science aims to develop the core competence in computing and problem solving amongst its graduates. Informally, “Learning to learn” has been the motto of the department since its inception. The curriculum thus focuses on building theoretical foundations in computer science to enable its pupils to think critically when challenged with totally different and new problems. It imbibes the following **Student-Centric** features of NEP 2020:

Flexibility to Exit: In order to support early exits, the curriculum aims to develop employability skills early. This has been done so that the outcomes of the 4 yr degree is not compromised as we believe that all but a few students will go for the full 4 year degree. As programming is at the heart of computing it is proposed to have two programming courses early so that the students can develop good programming skills in the first year. At the same time students are familiarized with the hardware of computers early on.

Employability: Industry demand in the IT sector has changed considerably in the past few years. With the humongous amount of data coming from all the domains like medical data, social networking data, astronomical data, education, etc., automating information extraction and analysis of data is the only way forward to leverage the available data for the future. The curriculum aims to equip the students with tools and techniques of Artificial Intelligence, Machine Learning and a pathway on Data Science if the student so desires.

Having said this, there is no replacement for the foundational courses like programming, data structures and algorithms. With two courses on programming and three courses on data structures and algorithms together, a strong foundation will be laid down for problem solving.

Flexibility to Choose: The curriculum provides multiple pathways (discipline specific electives) for the students to choose their micro-specialization, if any. This includes tracks on

Data-Science, Machine Learning and its application, Security, Networks, Theory and Internet Technologies. Each pathway provides learning progression in the chosen area of specialization. The project based pathway on "problem solving with computers" will inculcate the high order thinking skills in the students. Students will be expected to model real life problems mathematically and use their programming skills to solve them.

Multidisciplinarity: The curriculum provides two pathways (generic elective), one of computer science (CS) and the other of information technology (IT), to the students from other disciplines. Those who want to earn a minor in CS will be required to choose the first pathway whereas those who simply want to apply IT in the domain of their interest can choose the second pathway.

Research: With the option to obtain specialization in an area of their choice, the curriculum prepares the students to take up research projects in their final year. Strong mathematical background built up with the help of three papers on "Mathematics for Computing" prepares the students to choose the path of higher studies.

Discipline Specific Graduate Attributes (DSGA)

1. Proficiency in writing readable, correct, efficient, and secure programs of modest complexity.
2. Ability to design efficient algorithms using appropriate data structures for new problems.
3. Understanding of computer architecture, operating systems, computer networks and database management systems and their role in the performance of software applications.
4. Understanding of theoretical foundations and limits of computing.
5. Ability to develop good quality software by following the processes of software development life cycle.
6. Ability to extract information and analyze large volumes of data employing a range of techniques for artificial intelligence and learning.
7. Ability to design parallel algorithms to exploit the strength of multiple computing units in a computer.
8. Ability to develop an end to end compiler using compiler designing tools and techniques.
9. Ability to protect the data and software from various types of cyber attacks.

Table of Core Courses

Semester	DSC -No.	Title	L	T*	P*	Total credits	Desirable but not Prerequisites	DSGA Contributed to
I	DSC 01	Programming using Python	3	0	1	4	Nil	DSGA1
I	DSC 02	Computer System Architecture	3	0	1	4	Nil	DSGA3
I	DSC 03	Mathematics for computing	3	0	1	4	Nil	DSGA4
II	DSC 04	Object Oriented Programming with C++	3	0	1	4	Nil	DSGA1
II	DSC 05	Discrete Mathematical Structures	3	0	1	4	Nil	DSGA4
II	DSC 06	Probability for Computing	3	0	1	4	Nil	DSGA4
III	DSC 07	Data Structures	3	0	1	4	DSC 04/ a course in C/C++ at plus 2 level/**	DSGA1, DSGA2
III	DSC 08	Operating	3	0	1	4	DSC	DSGA3

		Systems					04/a course in C/C++ at plus 2 level/**	
III	**DSC 09**	**Numerical Optimization**	**3**	**0**	**1**	**4**	Nil	DSGA4
IV	**DSC10**	**Design and Analysis of Algorithms**	**3**	**0**	**1**	**4**	DSC07	DSGA1, DSGA2
IV	**DSC 11**	**Database Management System**	**3**	**0**	**1**	**4**	DSC01/ a course in Python at plus 2 level/** , DSC08	DSGA3
IV	**DSC 12**	**Computer Networks**	**3**	**0**	**1**	**4**	DSC04/a course in C/C++ at plus 2 level/** , DSC 07, DSC 08	DSGA3
V	**DSC 13**	**Algorithms and Advanced Data Structures**	**3**	**0**	**1**	**4**	DSC07, DSC10	DSGA1, DSGA2.
V	**DSC 14**	**Theory of Computation**	**3**	**1**	**0**	**4**	DSC04/a course in C/C++ at plus 2 level/** DSC05	DSGA4

V	**DSC 15**	**Software Engineering**	**3**	**0**	**1**	**4**	DSC01,/ DSC04,/ a course in C/C++/Python at plus 2 level/**	DSGA5
VI	**DSC 16**	**Artificial Intelligence**	**3**	**0**	**1**	**4**	DSC 01 DSC03 DSC 06 DSC 07 DSC 09 DSC10 DSC14	DSGA6
VI	**DSC 17**	**Machine Learning**	**3**	**0**	**1**	**4**	DSC 01 DSC03 DSC 06 DSC 07 DSC 09 DSC10	DSGA6
VI	**DSC 18**	**Introduction to Parallel Programming**	**3**	**0**	**1**	**4**	DCS 02, DSC04, DSC 07, DSC 08	DSGA7
VII	**DSC 19**	**Compiler Design**	**3**	**0**	**1**	**4**	DSC-14:	DSGA8
VIII	**DSC 20**	**Information Security**	**3**	**0**	**1**	**4**	DSC 01 DSC04, DSC 07 DSC 08 DSC 11 DSC 12	DSGA1, DSGA9

Note

1. **Batch size for Practicals will be (8-10) and Tutorials will be (12-15).**
2. **Wherever DCS04 is a prerequisite, a course in C/C++ at plus 2 level will be acceptable.**
3. **Wherever DCS01 is a prerequisite, a course in Python at plus 2 level will be acceptable.**

Syllabi of Core Courses

This section gives the detailed syllabus of the core courses. Each course describes the course objective, learning outcomes, the units and the reading material. The reading material has 2 -3 components: main resource(/s), additional text material, and online resources. Main resources are kept to a minimum possible and no more than 3. Additional resources and the online material may be used to enhance the knowledge of the subject.

DSC 01: Programming using Python

Course Objective

The course is designed to introduce programming concepts using Python to students. The course aims to develop structured as well as object-oriented programming skills using Python. The course also aims to achieve competence amongst its students to develop correct and efficient Python programs to solve real life problems.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Develop, document, and debug modular Python programs of reasonable complexity.
2. Implement arrays and user defined functions in Python.
3. Write programs in Python using dynamic memory allocation, handling external files, interrupts and exceptions.

4. Solve real life problems of reasonable complexity using suitable and efficient programming constructs in Python.
5. Solve real life problems of reasonable complexity using the concepts of object oriented programming in Python.

Syllabus

Unit 1 Introduction to Programming: Problem solving strategies; Structure of a Python program; Syntax and semantics; Executing simple programs in Python.

Unit 2 Creating Python Programs: Identifiers and keywords; Literals, numbers, and strings; Operators; Expressions; Input/output statements; Defining functions; Control structures (conditional statements, loop control statements, break, continue and pass, exit function), default arguments.

Unit 3 Built-in data structures: Mutable and immutable objects; Strings, built-in functions for string, string traversal, string operators and operations; Lists creation, traversal, slicing and splitting operations, passing list to a function; Tuples, sets, dictionaries and their operations.

Unit 4 Object Oriented Programming: Introduction to classes, objects and methods; Standard libraries.

Unit 5 File and exception handling: File handling through libraries; Errors and exception handling.

References

1. Taneja, S., Kumar, N. *Python Programming- A modular Approach*, 1st edition, Pearson Education India, 2018.
2. Balaguruswamy E. *Introduction to Computing and Problem Solving using Python,* 2nd edition, McGraw Hill Education, 2018.

Additional References

(i) Brown, Martin C. *Python: The Complete Reference*, 2nd edition, McGraw Hill Education, 2018.

(ii) Guttag, J.V. *Introduction to computation and programming using Python,* 2nd edition, MIT

Press, 2016.

Suggested Practical List

1. WAP to find the roots of a quadratic equation
2. WAP to accept a number ‘n’ and
    a. Check if ’n’ is prime
    b. Generate all prime numbers till ‘n’
    c. Generate first ‘n’ prime numbers

    This program may be done using functions
3. WAP to create a pyramid of the character ‘*’ and a reverse pyramid

```
   *
  ***
 *****
*******
*********

*********
 *******
  *****
   ***
    *
```

4. WAP that accepts a character and performs the following:
    a. print whether the character is a letter or numeric digit or a special character
    b. if the character is a letter, print whether the letter is uppercase or lowercase
    c. if the character is a numeric digit, prints its name in text (e.g., if input is 9, output is NINE)
5. WAP to perform the following operations on a string
    a. Find the frequency of a character in a string.
    b. Replace a character by another character in a string.
    c. Remove the first occurrence of a character from a string.
    d. Remove all occurrences of a character from a string.
6. WAP to swap the first n characters of two strings.
7. Write a function that accepts two strings and returns the indices of all the occurrences of the second string in the first string as a list. If the second string is not present in the first string then it should return -1.

8. WAP to create a list of the cubes of only the even integers appearing in the input list (may have elements of other types also) using the following:
    a. 'for' loop
    b. list comprehension

9. WAP to read a file and
    a. Print the total number of characters, words and lines in the file.
    b. Calculate the frequency of each character in the file. Use a variable of dictionary type to maintain the count.
    c. Print the words in reverse order.
    d. Copy even lines of the file to a file named 'File1' and odd lines to another file named 'File2'.
10. WAP to define a class Point with coordinates x and y as attributes. Create relevant methods and print the objects. Also define a method distance to calculate the distance between any two point objects.
11. Write a function that prints a dictionary where the keys are numbers between 1 and 5 and the values are cubes of the keys.
12. Consider a tuple t1=(1, 2, 5, 7, 9, 2, 4, 6, 8, 10). WAP to perform following operations:
    a) Print half the values of the tuple in one line and the other half in the next line.
    b) Print another tuple whose values are even numbers in the given tuple.
    c) Concatenate a tuple t2=(11,13,15) with t1.
    d) Return maximum and minimum value from this tuple
13. WAP to accept a name from a user. Raise and handle appropriate exception(s) if the text entered by the user contains digits and/or special characters.

DSC 02: Computer System Architecture

Course Objective

This course introduces the students to the fundamental concepts of digital computer organization, design and architecture. It aims to develop a basic understanding of the building blocks of the computer system and highlights how these blocks are organized

together to architect a digital computer system.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Design and Simplify Combinational and sequential circuits using basic building blocks.
2. Represent data in binary form, convert numeric data between different number systems and perform arithmetic operations in binary.
3. **Explain** instruction cycle, pipelining and interrupts.
4. **Explain** data communication between CPU, memory and I/O devices.
5. Simulate the design of a basic computer using a software tool.

Syllabus

Unit 1 Digital Logic Circuits: Logic Gates, Truth Tables, Boolean Algebra, Digital Circuits, Combinational Circuits, Introduction to Sequential Circuits, Circuit Simplification using Karnaugh Map, Don't Care Conditions, Flip-Flops, Characteristic Tables, Excitation Table.

Unit 2 Digital Components (Fundamental building blocks): Designing of combinational circuits- Half Adder, Full Adder, Decoders, Encoders, Multiplexers, Registers and Memory (RAM , ROM and their types) , Arithmetic Microoperations, Binary Adder, Binary Adder-Subtractor.

Unit 3 Data Representation and Basic Computer Arithmetic: Number System, r and (r-1)'s Complements, data representation and arithmetic operations.

Unit 4 Basic Computer Organization and Design: Bus organization, Micro programmed vs Hardwired Control , Instruction Codes, Instruction Format, Instruction Cycle, Instruction pipelining, Memory Reference, Register Reference and Input Output Instructions, Program Interrupt and Interrupt Cycle.

Unit 5 Processors: General register organization, Stack Organization, Addressing Modes, Overview of Reduced Instruction Set Computer (RISC) , Complex Instruction Set Computer (CISC), Multicore processor and Graphics Processing Unit (GPU)

Unit 6 Memory and Input-Output Organization: Memory hierarchy (main, cache and auxiliary memory), Input-Output Interface, Modes of Transfer: Programmed I/O, Interrupt initiated I/O, Direct memory access.

References

1. David A. Patterson and John L. Hennessy. "*Computer Organization and Design : The Hardware/Software interface*", 5th edition, Elsevier, 2012.
2. Mano, M. *Computer System Architecture*, 3rd edition, Pearson Education, 1993.

Additional References

(i) Mano, M. *Digital Design*, Pearson Education Asia, 1995.

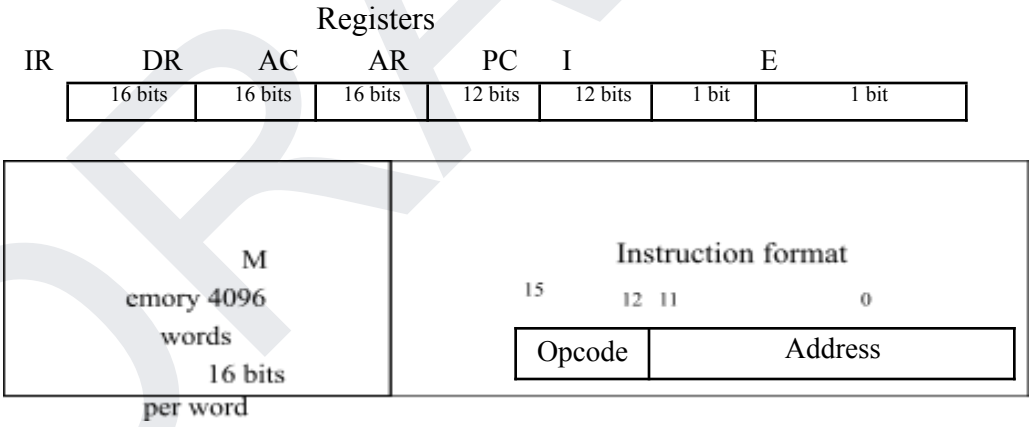
(ii) Null, L., & Lobur, J. *The Essentials of Computer Organization and Architecture*. 5th edition. (Reprint) Jones and Bartlett Learning, 2018.

(iii) Stallings, W. *Computer Organization and Architecture Designing for Performance* 8th edition, Prentice Hall of India, 2010.

Suggested Practical List

(Use Simulator – CPU Sim 3.6.9 or any higher version for the implementation)

1. Create a machine based on the following architecture:



Basic Computer Instructions

Memory Reference			Register Reference	
Symbol	Hex		Symbol	Hex
AND	0xxx	Direct	CLA	7800
ADD	1xxx		CLE	7400
LDA	2xxx		CMA	7200
STA	3xxx		CME	7100
BUN	4xxx		CIR	7080

BSA	5xxx	Addressing	CIL	7040
ISZ	6xxx		INC	7020
AND_I	8xxx	Indirect Addressing	SPA	7010
ADD_I	9xxx		SNA	7008
LDA_I	Axxx		SZA	7004
STA_I	Bxxx		SZE	7002
BUN_I	Cxxx		HLT	7001
BSA_I	Dxxx		INP	F800
ISZ_I	Exxx		OUT	F400

Refer to Chapter-5 of reference 1 for description of instructions.

Design the register set, memory and the instruction set. Use this machine for the assignments of this section.

2. Create a Fetch routine of the instruction cycle.
3. Write an assembly program to simulate ADD operation on two user-entered numbers.
4. Write an assembly program to simulate SUBTRACT operation on two user-entered numbers.
5. Write an assembly program to simulate the following logical operations on two user-entered numbers.
    i. AND
    ii. OR
    iii. NOT
    iv. XOR
    v. NOR
    vi. NAND
6. Write an assembly program for simulating following memory-reference instructions.
    i. ADD
    ii. LDA
    iii. STA
    iv. BUN
    v. ISZ
7. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:
    i. CLA
    ii. CMA
    iii. CME
    iv. HLT

8. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:
    i. INC
    ii. SPA
    iii. SNA
    iv. SZE

9. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:
    i. CIR
    ii. CIL

10. Write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).
11. Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.

DSC 03: Mathematics for computing

Course Objective

This course introduces the students to the fundamental concepts and topics of linear algebra and vector calculus, whose knowledge is important in other computer science courses. The course aims to build the foundation for some of the core courses in later semesters.

Course Learning Outcomes

After successful completion of this course, the student will be able to:

1. Perform operations on matrices and sparse matrices
2. Compute the determinant, rank and eigenvalues of a matrix

3. Perform diagonalization
4. Perform operations on vectors, the dot product and cross product
5. Represent vectors geometrically and calculate the gradient, divergence, curl
6. Apply linear algebra and vector calculus to solve problems in sub-disciplines of computer science.

Syllabus

Unit 1 Introduction to Matrix Algebra: Echelon form of a Matrix, Rank of a Matrix, Determinant and Inverse of a matrix, Solution of System of Homogeneous & Non-Homogeneous Equations: Gauss elimination and Solution of System of Homogeneous Equations: Gauss Jordan Method.

Unit 2 Vector Space and Linear Transformation: Vector Space, Sub-spaces, Linear Combinations, Linear Span, Convex Sets, Linear Independence/Dependence, Basis & Dimension, Linear transformation on finite dimensional vector spaces, Inner Product Space, Schwarz Inequality, Orthonormal Basis, Gram-Schmidt Orthogonalization Process.

Unit 3 EigenValue and EigenVector: Characteristic Polynomial, Cayley Hamilton Theorem, Eigen Value and Eigen Vector of a matrix, Eigenspaces, Diagonalization, Positive Definite Matrices, Applications to Markov Matrices

Unit 4 Vector Calculus: Vector Algebra,Laws of Vector Algebra, Dot Product, Cross Product,Vector and Scalar Fields, Ordinary Derivative of Vectors, Space Curves, Partial Derivatives, Del Operator, Gradient of a Scalar Field, Directional Derivative, Gradient of Matrices, Divergence of a Vector Field, Laplacian Operator, Curl of a Vector Field.

References

1.Strang Gilbert. *Introduction to Linear Algebra*, 5th Edition, Wellesley-Cambridge Press, 2021.

2. Kreyszig Erwin. *Advanced Engineering Mathematics*, 10th Edition, Wiley, 2015.

3. Strang Gilbert. *Linear Algebra and Learning from Data*, 1st Edition, Wellesley-CambridgePress,2019.

4.Jain R. K., Iyengar S.R. K. *Advanced Engineering Mathematics*, 5th Edition, Narosa, 2016.

Additional References

(i) Deisenroth, Marc Peter, Faisal A. Aldo and Ong ChengSoon. *Mathematics for Machine Learning*, 1st Edition, Cambridge University Press, 2020.

(ii) Lipschutz Seymour and Lipson Marc. *Schaum's Outline of Linear Algebra*, 6th Edition, McGraw Hill, 2017.

Suggested Practical List

1. Create and transform vectors and matrices (the transpose vector (matrix) conjugate transpose of a vector (matrix))

2. Generate the matrix into echelon form and find its rank.

3. Find cofactors, determinant, adjoint and inverse of a matrix.

4. Solve a system of Homogeneous and non-homogeneous equations using Gauss elimination method.

1. Solve a system of Homogeneous equations using the Gauss Jordan method.

6. Generate basis of column space, null space, row space and left null space of a matrix space.

7.Check the linear dependence of vectors. Generate a linear combination of given vectors of Rn/ matrices of the same size and find the transition matrix of given matrix space.

8.Find the orthonormal basis of a given vector space using the Gram-Schmidt orthogonalization process.

9.Check the diagonalizable property of matrices and find the corresponding eigenvalue and

verify the Cayley- Hamilton theorem.

10.Application of Linear algebra: Coding and decoding of messages using nonsingular matrices.

eg code “Linear Algebra is fun” and then decode it.

11.Compute Gradient of a scalar field.

12.Compute Divergence of a vector field.

13.Compute Curl of a vector field.

DSC 04: Object Oriented Programming with C++

Course Objective

This course is designed to introduce programming concepts using C++ to students. The course aims to develop structured as well as object-oriented programming skills using C++ programming language. The course also aims to achieve competence amongst its students to develop correct and efficient C++ programs to solve problems spanning multiple domains.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Write simple programs using built-in data types of C++.
2. Implement arrays and user defined functions in C++.
3. Write programs using dynamic memory allocation, handling external files, interrupts and exceptions.
4. Solve problems spanning multiple domains using suitable programming constructs in C++.
5. Solve problems spanning multiple domains using the concepts of object oriented programming in C++.

Syllabus

Unit 1 Introduction to C++: Overview of Procedural and Object-Oriented Programming, Using main() function, Header Files, Compiling and Executing Simple Programs in C++.

Unit 2 Programming Fundamentals: Data types, Variables, Operators, Expressions, Arrays, Keywords, Decision making constructs, Iteration, Type Casting, Input-output statements, Functions, Command Line Arguments/Parameters

Unit 3 Object Oriented Programming: Concepts of Abstraction, Encapsulation. Creating Classes and objects, Modifiers and Access Control, Constructors, Destructors, Implementation of Inheritance and Polymorphism, Template functions and classes

Unit 4 Pointers and References: Static and dynamic memory allocation, Pointer and Reference Variables, Implementing Runtime polymorphism using pointers and references

Unit 5 Exception and File Handling: Using try, catch, throw, throws and finally; Nested try, creating user defined exceptions, File I/O Basics, File Operations

References

1. Stephen Prata *C++ Primer Plus*, 6th Edition, Pearson India, 2015
2. E Balaguruswamy *Object Oriented Programming with C++*, 8th edition, McGraw-Hill Education, 2020
3. D.S. Malik, *C++ Programming: From Problem Analysis to Program Design*, 6th edition, Cengage Learning, 2013

Additional References

(i) Herbert Schildt, C++: *The Complete Reference*, 4th Edition, McGraw Hill, 2003

(ii) A. B. Forouzan, Richard F. Gilberg, *Computer Science: A Structured Approach using C++,* 2nd edition, Cengage Learning, 2010

Suggested Practical list

1. Write a program to compute the sum of the first n terms of the following series:

   $$S = 1 - \frac{1}{2^2} + \frac{1}{3^3} - \frac{1}{4^4} + \frac{1}{5^5} - \ldots.$$

   The number of terms n is to be taken from the user through the command line. If the command line argument is not found then prompt the user to enter the value of n.

2. Write a program to remove the duplicates from an array.
3. Write a program that prints a table indicating the number of occurrences of each alphabet in the text entered as command line arguments.
4. Write a menu driven program to perform string manipulation (without using inbuilt string functions):
   a. Show address of each character in string
   b. Concatenate two strings.
   c. Compare two strings
   d. Calculate length of the string (use pointers)
   e. Convert all lowercase characters to uppercase
   f. Reverse the string
   g. Insert a string in another string at a user specified position
5. Write a program to merge two ordered arrays to get a single ordered array.

6. Write a program to search a given element in a set of N numbers using Binary search (i) with recursion (ii) without recursion.
7. Write a program to calculate GCD of two numbers (i) with recursion (ii) without recursion.
8. Create a Matrix class. Write a menu-driven program to perform following Matrix operations (exceptions should be thrown by the functions if matrices passed to them are incompatible and handled by the main() function):
    a. Sum
    b. Product
    c. Transpose
9. Define a class Person having name as a data member. Inherit two classes Student and Employee from Person. Student has additional attributes as course, marks and year and Employee has department and salary. Write display() method in all the three classes to display the corresponding attributes. Provide the necessary methods to show runtime polymorphism.
10. Create a Triangle class. Add exception handling statements to ensure the following conditions: all sides are greater than 0 and sum of any two sides are greater than the third side. The class should also have overloaded functions for calculating the area of a right angled triangle as well as using Heron's formula to calculate the area of any type of triangle.
11. Create a class Student containing fields for Roll No., Name, Class, Year and Total Marks. Write a program to store 5 objects of Student class in a file. Retrieve these records from the file and display them.
12. Copy the contents of one text file to another file, after removing all whitespaces.

DSC 05: Discrete Mathematical Structures

Course Objective

This course is designed as a foundational course to make students learn about the mathematical constructs that are used in Computer Science such as Boolean algebra, sets, relations, functions, principles of counting, and recurrences. In this course, the knowledge of mathematical notation, ideas and concepts learnt at the pre-college levels is extended to orient the students towards mathematical thinking required in Computer Science.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Relate mathematical concepts and terminology to examples in the domain of Computer Science.
2. Model real world problems using various mathematical constructs.
3. Use different proofing techniques; construct simple mathematical proofs using logical arguments.
4. Formulate mathematical claims and construct counterexamples.

Syllabus

Unit 1 Sets, Functions, Sequences and Summations, Relations: Sets: Set Operations, Computer Representation of Sets, Countable and Uncountable Sets, Principle of Inclusion and Exclusion, Multisets; Functions: One-to-one and Onto Functions, Inverse Functions and Compositions of Functions, Graphs of Functions; Sequences and Summations: Sequences, Special Integer Sequences, Summations; Relations: Properties of Binary Relations, Equivalence relations and Partitions, Partial Ordering Relations and Lattices.

Unit 2 Logic and Proofs: Propositional Logic, Propositional Equivalences, Use of first-order logic to express natural language predicates, Quantifiers, Nested Quantifiers, Rules of Inference, Introduction to Proofs, Proof Methods and Strategies, Mathematical Induction.

Unit 3 Number Theory: Division and Integers, Primes and Greatest Common Divisors, Representation of Integers, Algorithms for Integer Operations, Modular Exponentiation, Applications of Number Theory.

Unit 4 Combinatorics/Counting: The Pigeonhole Principle, Permutations and Combinations, Binomial Coefficients, Generalized Permutations and Combinations, Generating Permutations and Combinations.

Unit 5 Graphs and Trees: Graphs: Basic Terminology, Multigraphs and Weighted Graphs, Paths and Circuits, Eulerian Paths and Circuits, Hamiltonian paths and Circuits, Shortest Paths, Spanning Trees, Graph Isomorphism, Planar Graphs; Trees: Trees, Rooted Trees, Path Lengths in Rooted Trees.

Unit 6 Recurrence: Recurrence Relations, Generating Functions, Linear Recurrence Relations with Constant Coefficients and their solution.

References

1. Liu, C.L., Mohapatra, D.P. *Elements of Discrete Mathematics: A Computer Oriented Approach*, 4th edition, Tata McGraw Hill, 2017.
2. Rosen, K.H.. *Discrete Mathematics and Its Applications*, 8th edition, Mc Graw Hill, 2018.

Additional References

(i) Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 4th edition, Prentice Hall of India. 2022.
(ii) Trembley, J.P., Manohar, R. *Discrete Mathematical Structures with Application to Computer Science*, Tata McGraw Hill, 1997.
(iii) Albertson, M. O. and Hutchinson, J. P. *Discrete Mathematics with Algorithms*, John Wiley and Sons, 1988.

Suggested Practical list

1. Create a class SET. Create member functions to perform the following SET operations:
    1) ismember: check whether an element belongs to the set or not and return value as true/false.
    2) powerset: list all the elements of the power set of a set .
    3) subset: Check whether one set is a subset of the other or not.
    4) union and Intersection of two Sets.
    5) complement: Assume Universal Set as per the input elements from the user.
    6) set Difference and Symmetric Difference between two sets.
    7) cartesian Product of Sets.

    Write a menu driven program to perform the above functions on an instance of the SET class.
2. Create a class RELATION, use Matrix notation to represent a relation. Include member functions to check if the relation is Reflexive, Symmetric, Anti-symmetric, Transitive. Using these functions check whether the given relation is: Equivalence or Partial Order relation or None
3. Write a Program that generates all the permutations of a given set of digits, with or without repetition.
4. For any number n, write a program to list all the solutions of the equation $x_1 + x_2 + x_3 + ...+ x_n = C$, where C is a constant (C<=10) and $x_1, x_2, x_3, ..., x_n$ are nonnegative integers, using brute force strategy.

5. Write a Program to evaluate a polynomial function. (For example store $f(x) = 4n^2 + 2n + 9$ in an array and for a given value of n, say n = 5, compute the value of f(n)).
6. Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency Matrix representation.
7. Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency List representation.
8. Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.

DSC 06: Probability for Computing

This course introduces the students to the fundamental concepts and topics of probability and statistics, whose knowledge is important in other computer science courses. The course aims to build the foundation for some of the core courses in later semesters.

Course Learning Outcomes

After successful completion of this course, the student will be able to:

1. Use probability theory to evaluate the probability of real-world events.
2. Describe discrete and continuous probability distribution functions and generate random numbers from the given distributions.
3. Find the distance between two probability distributions
4. Define and quantify the information contained in the data.
5. Perform data analysis in a probabilistic framework.
6. Visualize and model the given problem using mathematical concepts covered in the course.

Syllabus

Unit 1 Basic Probability: Introduction to the notion of probability, Random experiment, Sample space and Events, Probability defined on events, Algebra of events. Conditional probabilities, independent events, Bayes' theorem.

Unit 2 Random Variables: Introduction to Random Variables, Probability mass/density functions, Cumulative distribution functions. Discrete Random Variables (Bernoulli, Binomial, Poisson, Multinomial and Geometric). Continuous Random Variables (Uniform, Exponential and Normal). Expectation of a Random Variable, Expectation of Function of a Random Variable and Variance. Markov inequality, Chebyshev's inequality, Central Limit Theorem, Weak and Strong Laws of Large Numbers.

Unit 3 Joint Distributions: Jointly distributed Random Variables, Joint distribution functions, Independent Random Variables, Covariance of Random Variables, Correlation Coefficients, Conditional Expectation.

Unit 4 Markov Chain and Information Theory: Introduction to Stochastic Processes, Chapman–Kolmogorov equations, Classification of states, Limiting and Stationary Probabilities. Random Number Generation, Pseudo Random Numbers, Inverse Transformation Method, Rejection Method, Uncertainty, Information and Entropy, Mutual Information, KL Divergence.

References

1. Ross **S**heldon M. *Introduction to Probability Models*, 12th Edition, Elsevier, 2019.
2. Trivedi, K.S. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd edition, Wiley, 2015.
3. Deisenroth, Marc Peter, Faisal A. Aldo and Ong Cheng Soon, *Mathematics for Machine Learning*, 1st edition, Cambridge University Press, 2020.
4. Ian F. Blake, *An Introduction to Applied Probability*, John Wiley.

Additional References

(i) Johnson James L. *Probability and Statistics for Computer Science*, 6th edition, Wiley, 2004.

(ii) Forsyth David *Probability and Statistics for Computer Science*, 1st edition, Springer, 2019.

(iii) Freund J.E. *Mathematical Statistics with Applications*, 8th edition, Pearson Education, 2013.

(iv) Devore Jay L. *Probability and Statistics for Engineering and the Sciences*, 9th edition, Cengage Learning, 2020

Suggested Practical List

The goal of this lab is to develop data interpretation skills. Following exercises are designed to enable students to understand data characteristics either by visualization or by interpreting computed measures. All the exercises are to be completed using MS Excel functions and graphs. At the end of each exercise, the student should be able to draw a conclusion and state in a concise manner.Teachers are expected to guide students to obtain real data available through the internet for the following exercises.

1. Plotting and fitting of Binomial distribution and graphical representation of probabilities.

2. Plotting and fitting of Multinomial distribution and graphical representation of probabilities.

3.Plotting and fitting of Poisson distribution and graphical representation of probabilities.

4.Plotting and fitting of Geometric distribution and graphical representation of probabilities.

5.Plotting and fitting of Uniform distribution and graphical representation of probabilities.

6.Plotting and fitting of Exponential distribution and graphical representation of probabilities.

7.Plotting and fitting of Normal distribution and graphical representation of probabilities.

8.Calculation of cumulative distribution functions for Exponential and Normal distribution.

9.Given data from two distributions, find the distance between the distributions.

10.Application problems based on the Binomial distribution.

11.Application problems based on the Poisson distribution.

12.Application problems based on the Normal distribution.

13.Presentation of bivariate data through scatter-plot diagrams and calculations of covariance.

14.Calculation of Karl Pearson’s correlation coefficients.

15.To find the correlation coefficient for a bivariate frequency distribution.

16.Generating Random numbers from discrete (Bernoulli, Binomial, Poisson) distributions.

17.Generating Random numbers from continuous (Uniform, Normal) distributions.

18.Find the entropy from the given data set.

DSC 07: Data Structures

Course Objective

The course aims at developing the ability to use basic data structures like arrays, stacks, queues, lists, trees to solve problems. C++ is chosen as the language to understand implementation of these data structures.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Compare two functions for their rates of growth.
2. Understand abstract specification of data-structures and their implementation.
3. Compute time and space complexity of operations on a data-structure.
4. Identify the appropriate data structure(s) for a given application and understand the trade-offs involved in terms of time and space complexity.
5. Apply recursive techniques to solve problems.

Syllabus

Unit 1 Growth of Functions, Recurrence Relations: Functions used in analysis, asymptotic notations, asymptotic analysis, solving recurrences using recursion trees, Master Theorem.

Unit 2 Arrays, Linked Lists, Stacks, Queues, Deques: Arrays: array operations, applications, sorting, two-dimensional arrays, dynamic allocation of arrays; Linked Lists: singly linked lists, doubly linked lists, circularly linked lists, Stacks: stack as an ADT, implementing stacks using arrays, implementing stacks using linked lists, applications of stacks; Queues: queue as an ADT, implementing queues using arrays, implementing queues using linked lists, double-ended queue as an ADT, implementing double-ended queues using linked lists. Time complexity analysis.

Unit 3 Recursion: Recursive functions, linear recursion, binary recursion.

Unit 4 Trees, Binary Trees: Trees: definition and properties, tree traversal algorithms and their time complexity analysis; binary trees: definition and properties, traversal of binary trees and their time complexity analysis.

Unit 5 Binary Search Trees, Balanced Search Trees: Binary Search Trees: insert, delete, search operations, time complexity analysis of these operations; Balanced Search Trees: insert, search operations, time complexity analysis of these operations. Time complexity analysis.

Unit 6 Binary Heap, Priority Queue: Binary Heaps: heaps, heap operations, implementing heaps; Priority Queues: priority queue as an ADT. Time complexity analysis.

Unit 7 Graph Representations and Traversal Algorithms: Graph as an ADT, data structures for graphs, traversal of graphs, directed graphs. Time complexity analysis.

References

1. Goodrich, M.T., Tamassia, R., & Mount, D., *Data Structures and Algorithms Analysis in C++,* 2nd edition, Wiley, 2011.
2. Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 4th edition, Prentice Hall of India, 2022.
3. Drozdek, A., *Data Structures and Algorithms in C++*, 4th edition, Cengage Learning, 2012.

Additional references

(i) Sahni, S. *Data Structures, Algorithms and applications in C++*, 2nd edition, Universities Press, 2011.
(ii) Langsam Y., Augenstein, M. J., & Tanenbaum, A. M. *Data Structures Using C and C++*, Pearson, 2009.

Suggested Practical List

1. Write a program, using templates, to sort an array of size n. Give the user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

2. Write a program to implement singly linked list as an ADT that supports the following operations:
   (i) Insert an element x at the beginning of the singly linked list
   (ii) Insert an element x at i^{th} position in the singly linked list
   (iii) Remove an element from the beginning of the singly linked list
   (iv) Remove an element from i^{th} position in the singly linked list.
   (v) Search for an element x in the singly linked list and return its pointer
   (vi) Concatenate two singly linked lists

3. Write a program to implement doubly linked list as an ADT that supports the following operations:
   (i) Insert an element x at the beginning of the doubly linked list
   (ii) Insert an element x at i^{th} position in the doubly linked list
   (iii) Insert an element x at the end of the doubly linked list
   (iv) Remove an element from the beginning of the doubly linked list
   (v) Remove an element from i^{th} position in the doubly linked list.
   (vi) Remove an element from the end of the doubly linked list

(vii) Search for an element x in the doubly linked list and return its pointer
(viii) Concatenate two doubly linked lists

4. Write a program to implement circular linked list as an ADT which supports the following operations:
   (i) Insert an element x at the front of the circularly linked list
   (ii) Insert an element x after an element y in the circularly linked list
   (iii) Insert an element x at the back of the circularly linked list
   (iv) Remove an element from the back of the circularly linked list
   (v) Remove an element from the front of the circularly linked list
   (vi) Remove the element x from the circularly linked list
   (vii) Search for an element x in the circularly linked list and return its pointer
   (viii) Concatenate two circularly linked lists

5. Implement a stack as an ADT using Arrays.

6. Implement a stack as an ADT using the Linked List ADT.

7. Write a program to evaluate a prefix/postfix expression using stacks.

8. Implement Queue as an ADT using the circular Arrays.

9. Implement Queue as an ADT using the Circular Linked List ADT.

10. Implement Double-ended Queues as an ADT using the Doubly Linked list ADT.

11. Write a program to implement Binary Search Tree as an ADT which supports the following operations:
   (i) Insert an element x
   (ii) Delete an element x
   (iii) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position in the BST
   (iv) Display the elements of the BST in preorder, inorder, and postorder traversal
   (v) Display the elements of the BST in level-by-level traversal
   (vi) Display the height of the BST

12. Write a program to implement a balanced search tree as an ADT.

13. Write a Program to implement a priority queue as an ADT using heap data structure.

DSC 08: Operating Systems

Course Objective

The course provides concepts that underlie all operating systems not tied to any particular operating system. The emphasis is done to explain the need and structure of an operating system using its common services such as process management (creation, termination etc.), CPU Scheduling, Process Synchronization, Handling Deadlocks, main memory management, virtual memory, secondary memory management. The course also introduces various scheduling algorithms and structures/techniques used by operating systems to provide these services.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Understand the need of an Operating System & Define Multiprogramming and Multithreading concepts.
2. Implement Process Synchronization service (Critical Section, Semaphores), CPU scheduling service with various algorithms.
3. Learn Main memory Management (Paging, Segmentation) algorithms, Handling of Deadlocks
4. Identify and appreciate the File systems Services, Disk Scheduling service

Pre-Requisite

Familiarity with data structures and any of the high-level languages such as C/C++ is required to do lab assignments.

Syllabus

Unit 1 Introduction: Operating Systems (OS) definition and its purpose, Multiprogrammed and Time Sharing Systems, OS Structure, OS Operations: Dual and Multi-mode, OS as resource manager.

Unit 2 Operating System Structures: OS Services, System Calls: Process Control, File Management, Device Management, and Information Maintenance, Inter-process Communication, and Protection, System programs, OS structure- Simple, Layered, Microkernel, and Modular.

Unit 3 Process Management : Process Concept, States, Process Control Block, Process Scheduling, Schedulers, Context Switch, Operation on processes, Threads, Multicore Programming, Multithreading Models, PThreads, Process Scheduling Algorithms: First Come First Served, Shortest-Job-First, Priority & Round-Robin, Process Synchronization:The critical-section problem and Peterson's Solution, Deadlock characterization, Deadlock handling.

Unit 4 Memory Management : Physical and Logical address space, Swapping, Contiguous memory allocation strategies - fixed and variable partitions, Segmentation, Paging.

Virtual Memory Management: Demand Paging and Page Replacement algorithms: FIFO Page Replacement, Optimal Page replacement, LRU page replacement.

Unit 5 File System: File Concepts, File Attributes, File Access Methods, Directory Structure: Single-Level, Two-Level, Tree-Structured, and Acyclic-Graph Directories.
Mass Storage Structure: Magnetic Disks, Solid-State Disks, Magnetic Tapes, Disk Scheduling algorithms: FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOk Scheduling.

References

1. Silberschatz, A., Galvin, P. B., Gagne G. *Operating System Concepts,* 9th edition, John Wiley Publications, 2016.

Additional References

(i) Dhamdhere, D. M., *Operating Systems: A Concept-based Approach,* 2nd edition, Tata McGraw-Hill Education, 2017.
(ii) Kernighan, B. W., Rob Pike, R. *The Unix Programming Environment*, Englewood Cliffs, NJ: Prentice-Hall, 1984.
(iii) Stallings, W. *Operating Systems: Internals and Design Principles,* 9th edition, Pearson Education, 2018.
(iv) Tanenbaum, A. S. *Modern Operating Systems*, 3rd edition, Pearson Education, 2007.

Suggested Practical List

1. Execute various LINUX commands for:
   i. Information Maintenance: wc, clear, cal, who, date, pwd
   ii. File Management: cat, cp, rm, mv, cmp, comm, diff, find, grep, awk
   iii. Directory Management : cd, mkdir, rmdir, ls
2. Execute various LINUX commands for:
   i. Process Control: fork, getpid, ps, kill, sleep
   ii. Communication: Input-output redirection, Pipe
   iii. Protection Management: chmod, chown, chgrp
3. Write a program(using fork() and/or exec() commands) where parent and child execute:
   i. same program, same code.
   ii. same program, different code.
   iii. before terminating, the parent waits for the child to finish its task.
4. Write a program to to report behaviour of Linux kernel including kernel version, CPU type

and model. (CPU information)

5. Write a program to report behaviour of Linux kernel including information on 19 configured memory, amount of free and used memory. (Memory information)
6. Write a program to copy files using system calls.
7. Write a program to implement FCFS scheduling algorithm.
8. Write a program to implement SJF scheduling algorithm.
9. Write a program to implement non-preemptive priority based scheduling algorithm.
10. Write a program to implement SRTF scheduling algorithm.
11. Write a program to calculate sum of n numbers using Pthreads. A list of n numbers is divided into two smaller list of equal size, two separate threads are used to sum the sublists.
12. Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

DSC 09: Numerical Optimization

Course Objective

The course aims to provide students an experience of mathematically formulating a large variety of optimization/decision problems emerging out of various fields like data science, machine learning, business and finance. The course focuses on learning techniques to optimize the problems in order to obtain the best possible solution.

Course Learning Outcomes

At the end of the course, students will be able to:

1. Mathematically formulate the optimization problems using the required number of independent variables.
2. Define constraint functions on a problem.
3. Check the feasibility and optimality of a solution.
4. Apply conjugate gradient method to solve the problem.

Syllabus

Unit 1 Introduction: Mathematical Formulation using example, Continuous versus Discrete Optimization, Constrained and Unconstrained Optimization, Global and Local Optimization, Stochastic and Deterministic Optimization, Convexity, Optimization Algorithms

Unit 2 Fundamentals of Unconstrained Optimization: Concept of a Solution - Recognizing a Local Minimum, Nonsmooth Problems. Overview of Algorithms - Two Strategies: Line Search and Trust Region, Search Directions for Line Search Methods, Models for Trust-Region Methods, Scaling. Line Search - Convergence of Line Search Methods, Rate of Convergence - Convergence Rate of Steepest Descent, Newton's Method, Quasi-Newton Methods. Trust Region - The Cauchy Point algorithm, Global Convergence - Reduction Obtained by the Cauchy Point, Convergence to Stationary Points.

Unit 3 Conjugate Gradient Methods: Basic Properties of the Conjugate Gradient Method, A Practical Form of the Conjugate Gradient Method, Rate of Convergence.

Unit 4 Calculating Derivatives: Finite-Difference Derivative Approximations, Approximating the Gradient, Approximating a Sparse Jacobian, Approximating the Hessian, Approximating a Sparse Hessian

Unit 5 Theory of Constrained Optimization: Local and Global Solutions, Smoothness, Examples - A Single Equality Constraint, A Single Inequality Constraint, Two Inequality Constraints, Tangent Cone and Constraint Qualifications, First-Order Optimality Condition, Second-Order Conditions - Second-Order Conditions and Projected Hessians. Linear and Non-linear Constrained Optimization. Augmented Lagrangian Methods.

References

1. J. Nocedal and S.J. Wright, Numerical Optimization, Second Edition, Springer Series in Operations Research, 2006.

DSC 10: Design and Analysis of Algorithms

Course Objective

The course is designed to develop understanding of different algorithm design techniques and use them for problem solving. The course shall also enable the students to verify correctness of algorithms and analyze their time complexity.

Course Learning Outcomes

On successful completion of this course, the student will be able to:

1. Compute and compare the asymptotic time complexity of algorithms.
2. Prove correctness of algorithms.
3. Use appropriate algorithm design technique(s) for solving a given problem.
4. Appreciate the difference between tractable and intractable problems.

Syllabus

Unit 1 Searching, Sorting, Selection: Linear Search, Binary Search, Insertion Sort, Selection Sort, Bubble Sort, Heapsort, Linear Time Sorting, Selection Problem, running time analysis and correctness.

Unit 2 Graphs: Review of graph traversals**,** graph connectivity, testing bipartiteness, Directed Acyclic Graphs and Topological Ordering.

Unit 3 Divide and Conquer: Introduction to divide and conquer technique, Merge Sort, Quick Sort, Maximum-subarray problem, Strassen's algorithm for matrix multiplication.

Unit 4 Greedy algorithms: Introduction to the Greedy algorithm design approach, application to minimum spanning trees, fractional knapsack problem, etc. with correctness, and analysis of time complexity.

Unit 5 Dynamic Programming: Introduction to the Dynamic Programming approach, application to subset sum, integer knapsack problem etc., correctness, and analysis of time complexity.

Unit 6 Intractability: Concept of polynomial time computation, polynomial time reductions, decision vs optimization problems, Introduction to NP, NP-hard and NP-Complete classes.

Unit 7 Advanced Analysis Technique: Amortized Analysis.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 4th edition, Prentice Hall of India, 2022.
2. Kleinberg, J., Tardos, E. *Algorithm Design,* 1st edition, Pearson, 2013.

Additional References

(i) Basse, S., Gelder, A. V., *Computer Algorithms: Introduction to Design and Analysis,* 3rd edition, Pearson, 1999.

Suggested Practical List

1. i. Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).
   ii. Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).

2. Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).
3. Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.
4. Write a program to sort the elements of an array using Radix Sort.
5. Write a program to sort the elements of an array using Bucket Sort.
6. Display the data stored in a given graph using the Breadth-First Search algorithm.
7. Display the data stored in a given graph using the Depth-First Search algorithm.
8. Write a program to determine a minimum spanning tree of a graph using the Prim's algorithm.
9. Write a program to implement Dijkstra's algorithm to find the shortest paths from a given source node to all other nodes in a graph.
10. Write a program to solve the weighted interval scheduling problem.
11. Write a program to solve the 0-1 knapsack problem.

For the algorithms at S.No 1 and 2, test run the algorithm on 100 different input sizes varying from 30 to 1000. For each size find the number of comparisons averaged on 10 different input instances; plot a graph for the average number of comparisons against each input size. Compare it with a graph of nlogn.

DSC 11: Database Management System

Course Objective

The course introduces the students to the fundamentals of database management system and its architecture. Emphasis is given on the popular relational database system including data models and data manipulation. Students will learn about the importance of database structure and its designing using conceptual approach using Entity Relationship Model and formal approach using Normalization. The importance of file indexing and controlled execution of transactions will be taught. The course would give students hands-on practice of structured query language in a relational database management system and glimpse of basic database administration commands.

Course Learning Outcomes

On successful completion of the course, students will be able to:

1. Use database management system software to create and manipulate the database.
2. Create conceptual data models using entity relationship diagrams for modeling real-life situations and designing the database schema.
3. Use the concept of functional dependencies to remove redundancy and update anomalies.
4. Apply normalization theory to get a normalized database scheme.
5. Write queries using relational algebra, a procedural language.
6. Implement relational databases and formulate queries to get solutions of a broad range of data retrieval and data update problems using SQL .
7. Write and execute SQL queries through a high-level language via ODBC connection
8. Database administration commands such as creating/removing users, granting/revoking different privileges to the database users; creating assertions, triggers, and indexes.
9. Learn the importance of index structures and concurrent execution of transactions in database systems.

Syllabus

Unit 1 Introduction to Database: Purpose of database system, Characteristics of database approach, data models, database management system, database system architecture, three-schema architecture, components of DBMS, data independence, and file system approach vs database system approach.

Unit 2 Entity Relationship Modeling: Conceptual data modeling - motivation, entities, entity types, attributes, relationships, relationship types, constraints on relationship, Entity Relationship diagram notation.

Unit 3 Relational Data Model: Update anomalies, Relational Data Model - Concept of relations, schema-instance distinction, keys, relational integrity constraints, referential integrity and foreign keys, relational algebra operators and queries.

Unit 4 Structured Query Language (SQL): Querying in SQL, DDL to create database and tables, table constraints, update database-update behaviors, DML, aggregation functions group by and having clauses, retrieve data from the database, generate and query views. Access and manipulate databases using ODBC. Basic Database administration SQL commands.

Unit 5 Database Design: Mapping an Entity Relationship model to relational database, functional dependencies and Normal forms, 1NF, 2NF, 3NF and BCNF decompositions and desirable properties of them.

Unit 6 File indexing and Transaction Processing: Data Storage and Indexes- Need of file indexes, file organizations, index structures, single- and multi-level indexing, concurrent execution of transactions, ACID properties, need of data recovery and log file.

References

1. Elmasri, R., Navathe, B. S. *Fundamentals of Database Systems*, 7th Edition, Pearson Education, 2015.
2. Krogh, J. W. *MySQL Connector/Python Revealed: SQL and NoSQL Data Storage Using MySQL for Python Programmers*, Apress, 2018.
3. Murach J. *Murach's MySQL,* 3rd edition, Pearson, 2019.

Additional References

(i) Ramakrishnan, R., Gehrke J. *Database Management Systems*, 3rd Edition, McGraw-Hill, 2014.
(ii) Silberschatz, A., Korth, H. F., Sudarshan S. *Database System Concepts*, 7th Edition, McGraw Hill, 2019.
(iii) Connolly, T. M., Begg, C. E. *Database Systems: A Practical Approach to Design, Implementation, and Management*, 6th edition, Pearson, 2019.

Suggested Practical List

It has three components.

I. Create and use the following student-society database schema for a college to answer the given (sample) queries using the standalone SQL editor.

STUDENT	**Roll No**	**StudentName**	**Course**	**DOB**
	Char(6)	Varchar(20)	Varchar(10)	Date

SOCIETY	**SocID**	**SocName**	**MentorName**	**TotalSeats**
	Char(6)	Varchar(20)	Varchar(15)	Unsigned int

ENROLLMENT	Roll No Char(6)	SID Char(6)	DateOfEnrollment Date

Here Rollno (ENROLLMENT) and SID (ENROLLMENT) are foreign keys.

1. Retrieve names of students enrolled in any society.
2. Retrieve all society names.
3. Retrieve students' names starting with letter ‘A’.
4. Retrieve students' details studying in courses ‘computer science’ or ‘chemistry’.
5. Retrieve students’ names whose roll no either starts with ‘X’ or ‘Z’ and ends with ‘9’
6. Find society details with more than N TotalSeats where N is to be input by the user
7. Update society table for mentor name of a specific society
8. Find society names in which more than five students have enrolled
9. Find the name of youngest student enrolled in society ‘NSS’
10. Find the name of most popular society (on the basis of enrolled students)
11. Find the name of two least popular societies (on the basis of enrolled students)
12. Find the student names who are not enrolled in any society
13. Find the student names enrolled in at least two societies
14. Find society names in which maximum students are enrolled
15. Find names of all students who have enrolled in any society and society names in which at least one student has enrolled
16. Find names of students who are enrolled in any of the three societies ‘Debating’, ‘Dancing’ and ‘Sashakt’.
17. Find society names such that its mentor has a name with ‘Gupta’ in it.
18. Find the society names in which the number of enrolled students is only 10% of its capacity.
19. Display the vacant seats for each society.
20. Increment Total Seats of each society by 10%
21. Add enrollment fees paid (‘yes’/’No’) field in the enrollment table.
22. Update date of enrollment of society id ‘s1’ to ‘2018-01-15’, ‘s2’ to current date and ‘s3’ to ‘2018-01-02’.
23. Create a view to keep track of society names with the total number of students enrolled in it.
24. Find student names enrolled in all the societies.
25. Count number of societies with more than 5 student enrolled in it
26. Add column Mobile number in student table with default value ‘9999999999’
27. Find the total number of students whose age is > 20 years.
28. Find names of students who are born in 2001 and are enrolled in at least one society.
29. Count all societies whose name starts with ‘S’ and ends with ‘t’ and at least 5 students are enrolled in the society.
30. Display the following information:

Society name Mentor name Total Capacity Total Enrolled Unfilled Seats

II. Do the following database administration commands:
create user, create role, grant privileges to a role, revoke privileges from a role, create index

III. Execute queries given in part I through a high-level language using ODBC connection.

DSC 12: Computer Networks

Course Objective

The course objectives of this course are to:

understand the concepts behind computer networks and data communication. Learn the use of different layers in standard reference models used for communication. Learn the main features of protocols used at various layers. Understand the utility of different networking devices..

Course Learning Outcomes

On successful completion of the course, the student will be able to:

1. describe the hardware and software components used in a network.
2. compare OSI and TCP/IP reference models at various layers.
3. describe, analyze and compare different data link, network, transport and application layer protocols.
4. design/implement data link and network layer protocols in a simulated networking environment.

Syllabus

Unit 1 Introduction: Types of computer networks, Internet, Intranet, network topologies, network classifications. layered architecture approach, OSI Reference Model, TCP/IP Reference Model.

Unit 2 Physical Layer: Analog signal, digital signal, digital modulation techniques (ASK, PSK, QAM), encoding techniques, the maximum data rate of a channel, transmission media (guided transmission media, wireless transmission, satellite communication), multiplexing (frequency division multiplexing, time-division multiplexing, wavelength division multiplexing).

Unit 3 Data Link MAC Layer: Data link layer services, error detection and correction techniques, error recovery protocols (stop and wait, go back n, selective repeat), multiple access

protocols with collision detection, MAC addressing, Ethernet, data link layer switching, point-to-point protocol.

Unit 4 Network layer: Networks and Internetworks, virtual circuits and datagrams, addressing, subnetting, Routing algorithm (Distance vector and Dijkstra routing), Network Layer protocol-(ARP, IPV4, ICMP, IPV6).

Unit 5 Transport and Application Layer: Process to process Delivery- (client-server paradigm, connectionless versus connection-oriented service, reliable versus unreliable); User Datagram Protocols, TCP/IP protocol, Flow Control. FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), Telnet (Remote login protocol), WWW (World Wide Web), HTTP (HyperText Transfer Protocol), URL (Uniform Resource Locator).

References

1. Tanenbaum, A.S. & Wethrall, D.J, *Computer Networks*, Pearson Education, 2012.
2. Forouzan, B. A., *Data Communication and Networking,* McGraw-Hill Education, 2017.

Additional References

(i) Kurose, J. F., & Ross, K. W. *Computer Networking: A Top-Down Approach,* Pearson Education India, 2017.

(ii) Stallings, W. *Data and Computer Communications,* 10th edition, Pearson Education India, 2017.

Suggested Practical List

1. Simulate Cyclic Redundancy Check (CRC) error detection algorithm for the noisy channel.
2. Simulate and implement the Stop and Wait protocol for the noisy channel.
3. Simulate and implement go back n sliding window protocol.
4. Simulate and implement selective repeat sliding window protocol.
5. Simulate and implement distance vector routing algorithm
6. Simulate and implement the Dijkstra algorithm for shortest-path routing.

DSC 13: Algorithms and Advanced Data Structures

Course Objective

This course is designed to build upon the fundamentals in data structures and algorithm design and gain exposure to more data structures and algorithms for new problems.

Course Learning Outcomes

On successful completion of this course, the student will be able to:

1. Comprehend and use data structures for lists.
2. Use hash tables for dictionaries.
3. Comprehend and use data structures and algorithms for string matching.
4. Comprehend and use disc based data structures.
5. Implement and analyze advanced data structures and algorithms for graphs.
6. Appreciate the strength of randomization in data structures and algorithms.

Syllabus

Unit 1 List and Iterator ADTs: Vectors, Lists, Sequences

Unit 2 Hash Tables, Dictionaries: Hash Functions, Collision resolution schemes.

Unit 3 Strings: String Matching: KMP algorithm; Tries: Standard Tries, Compressed Tries, Suffix Tries, Search Engines

Unit 4 More on Trees: 2-4 Trees, B Trees

Unit 5 More on Graphs: Bellman Ford Algorithm, Union Find Data Structures - application Kruskal's algorithm

Unit 6 Randomization: Randomized Quicksort, Randomized Select, Skip List

Unit 7 Network Flows: Ford Fulkerson algorithm for max flow problem.

References

1. Goodrich, M.T, Tamassia, R., & Mount, D.,. *Data Structures and Algorithms Analysis in C++,* 2nd edition, Wiley, 2011.
2. Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein C. *Introduction to Algorithms*, 4th edition, Prentice Hall of India, 2022.
3. Kleinberg, J., Tardos, E. *Algorithm Design,* 1st edition, Pearson, 2013.
4. Drozdek, A.,. *Data Structures and Algorithms in C++*, 4th edition, Cengage Learning. 2012.

Suggested Practical List

1. Write a program to sort the elements of an array using Randomized Quick sort (the program should report the number of comparisons).
2. Write a program to find the i^{th} smallest element of an array using Randomized Select.

3. Write a program to determine the minimum spanning tree of a graph using Kruskal's algorithm.
4. Write a program to implement the Bellman Ford algorithm to find the shortest paths from a given source node to all other nodes in a graph.
5. Write a program to implement a B-Tree.
6. Write a program to implement the Trie Data structure, which supports the following operations:
    I. Insert
    II. Search
7. Write a program to search a pattern in a given text using the KMP algorithm.
8. Write a program to implement a Suffix tree.

DSC 14: Theory of Computation

Course Objective

This course introduces formal models of computation, namely, finite automaton, pushdown automaton, and Turing machine; and their relationships with formal languages. Students will also learn about the limitations of computing machines as this course addresses the issue of which problems can be solved by computational means (decidability vs undecidability). The course is aimed to -

1. formalize the notion of problems via formal languages
2. make students aware of the notion of computation using "abstract computing devices" called automata
3. familiarize students with the hierarchy of classes of problems or formal languages (regular, context-free, context-sensitive, decidable, and undecidable) and the hierarchy of classes of automata (finite automata, pushdown automata, and Turing machines)

Course Learning Outcomes

On successful completion of the course, a student will be able to:

1. design a finite automaton, pushdown automaton or a Turing machine for a problem at hand.
2. apply pumping lemma to prove that a language is non-regular/non-context-free.
3. describe limitations of a computing machines and
4. recognize what can be solved and what cannot be solved using these machines.

Pre-requisites

A course in programming language, Design and Analysis of Algorithms

Syllabus

Unit I Introduction: Alphabets, string, language, basic operations on language, concatenation, union, Kleene star.

Unit II Finite Automata and Regular: Regular expressions, Deterministic Finite Automata (DFA), Non-deterministic Finite Automata (NFA), relationship between NFA and DFA, Transition Graphs (TG), properties of regular languages, the relationship between regular languages and finite automata, pumping lemma, Kleene's theorem.

Unit III Context-Free Languages (CFL): Context-Free Grammars (CFG), deterministic and non-deterministic Pushdown Automata (PDA), relationship between CFG and PDA, parse trees, leftmost derivation, Ambiguities in grammars, pumping lemma for CFL, properties of CFL, Chomsky Normal Form.

Unit IV Turing Machines and Models of Computations: Turing machine as a model of computation, configuration of Turing machine, Recursive and recursively enumerable languages, Church Turing Thesis, Universal Turing Machine, decidability, Halting problem.

References

1. Michael Sipser, *Introduction to the Theory of Computation*, Cengage, 2014
2. Harry R. Lewis and Christos H. Papadimitriou, *Elements of the Theory of Computation*, 2nd Edition, Prentice Hall of India (PHI), 2002
3. Daniel I.A. Cohen, *Introduction to Computer Theory*, 2nd Edition, Wiley India Pvt. Ltd., 2011.

Additional References

(i) J.E. Hopcroft, R. Motwani, and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 3rd edition, Addison Wesley, 2006.

(ii) Peter Linz, *An Introduction to Formal Languages and Automata*, 6th edition, Jones & Bartlett Learning, 2017.

DSC 15: Software Engineering

Course Objective

This course will provide fundamental approaches and techniques used to develop good quality software. This includes learning of various software development process frameworks, requirement analysis, design modeling, qualitative and quantitative software metrics, risk management, and testing techniques**.**

Course Learning Outcomes

On successful completion of the course, a student will be able to:

1. Understand the software development models.
2. Analyze and model customer requirements and build design models.
3. Estimate and prepare schedule for software projects.
4. Analyze the impact of risks involved in software development.
5. Design and build test cases, and to perform software testing.

Pre-requisites:

A course in programming language

Syllabus

Unit I Introduction: Software Engineering - A Layered Approach; Software Process – Process Framework, Umbrella Activities; Process Models – Waterfall Model, Incremental Model, and Evolutionary process Model (Prototyping, Spiral Model); Introduction to Agile, Agile Model – Scrum. [9 Hours]

Unit II Software Requirements Analysis and Specification: Use Case Approach, Software Requirement Specification Document, Flow-oriented Model, Data Flow Model. [6 Hours]

Unit III Design Modeling: Translating the Requirements model into the Design Model, The Design Process, Design Concepts - Abstraction, Modularity and Functional Independence; Structure Charts. [8 Hours]

Unit IV Software Metrics and Project Estimation: Function based Metrics, Software Measurement, Metrics for Software Quality; Software Project Estimation (FP based estimations); Project Scheduling (Timeline charts, tracking the schedule). [7 Hours]

Unit V Quality Control and Risk Management: Quality Control and Quality Assurance, Software Process Assessment and Improvement; Software Risks, Risk Identification, Risk Projection, Risk Mitigation, Monitoring and Management. [5 Hours]

Unit VI Software Testing: Strategic Approach to Software Testing, Unit Testing, Integration Testing, Validation Testing, System Testing; Black-Box and White Box Testing, Basis Path Testing. [10 Hours]

References

1. Pressman, R.S. *Software Engineering: A Practitioner's Approach*, 9th edition, McGraw-Hill, 2020.
2. Aggarwal, K.K., Singh, Y. *Software Engineering*, 3rd edition, New Age International Publishers, 2007.

Additional References

(i) Jalote, P. *An Integrated Approach to Software Engineering*, 3rd Edition, Narosa Publishing House, 2005.

(ii) Sommerville, I. *Software Engineering*, 9th edition, Addison Wesley, 2011.

(iii)The Definitive Guide to Scrum: The Rules of the Game, Ken Schwaber, Jeff Sutherland, July 2016.

Suggested Practical list

The students document, design and code a module of a Software Project using an appropriate Software Process model. The Software Project should address the following concepts of Software Engineering.

1. Problem Statement, Process Model
2. Requirement Analysis: Create Data Flow, Data Dictionary, Use Cases, Sequence Diagram, Software Requirement Specification Document
3. Project Management: Timeline Chart, Compute FP, Effort, Cost, Risk Table.
4. Design Engineering: Architectural Design, Pseudocode of a small module.
5. Coding: Develop at least a single module using any programming Language

6. Testing: Compute Basis path set for at least a single module from the project, Generate test cases.

Some of the Sample Projects are given below though they are not limited to this.

1. Criminal Record Management: Implement a criminal record management system for jailers, police officers and CBI officers
2. DTC Route Information: Online information about the bus routes and their frequency and fares.
3. Car Pooling: To maintain a web-based intranet application that enables the corporate employees within an organization to avail the facility of carpooling effectively.
4. Patient Appointment and Prescription Management System
5. Organized Retail Shopping Management Software
6. Online Hotel Reservation Service System
7. Examination and Result computation System
8. Automatic Internal Assessment System
9. Parking Allocation System
10. Wholesale Management System

DSC 16 : Artificial Intelligence

Course Objective

The course objectives of this course are to:
Understand the foundations, basic concepts and techniques of Artificial Intelligence (AI).

Apply informed search techniques for different applications.

Impart knowledge about the use of core AI techniques having applicability to a wide range of real-world problems.

Learn about various knowledge representation techniques and writing Prolog programs.

Learn about the latest techniques for developing AI systems.

Course Learning Outcomes

On successful completion of this course, students will be able to:

1. identify problems that are amenable to solutions by specific AI methods.
2. appreciate the utility of different types of AI agents.
3. apply different informed search techniques for solving real world problems.
4. use knowledge representation techniques for AI systems..
5. understand human level, data driven and end to end approaches to AI.

Prerequisite:

Knowledge of any Programming Language, Basics of algorithms, Data Structures, and Theory of Computation.

Syllabus

Unit 1 Introduction: Introduction to artificial intelligence, background and applications, Turing test, Weak AI, Strong AI, Narrow AI, Artificial General Intelligence, Super AI, rational agent approaches to AI, introduction to intelligent agents, their structure, behavior and task environment.

Unit 2 Problem Solving and Searching Techniques: Problem characteristics, production systems, control strategies, breadth-first search, depth-first search, hill climbing and its variations, heuristics search techniques: best-first search, A* algorithm, constraint satisfaction problem, means-end analysis, introduction to game playing, min-max and alpha-beta pruning algorithms.

Unit 3 Knowledge Representation: Propositional logic, First-Order Predicate logic, resolution principle, unification, semantic nets, conceptual dependencies, frames, and scripts, production rules, Introduction to Programming in Logic (PROLOG).

Unit 4 Understanding Natural Languages: Components and steps of communication, the contrast between formal and natural languages in the context of grammar, Chomsky hierarchy of grammars, parsing, and semantics, Parsing Techniques, Context-Free and Transformational Grammars, Recursive and Augmented transition nets.

Unit 5 AI The Present and the Future: Symbolic AI, Data-driven AI and Machine Learning, Bias-Variance tradeoff in Machine Learning, End-to-end Deep Learning based AI, some applications of symbolic and data driven AI, Interpretable and Explainable AI, Ethics of AI: benefits and risks of AI.

Text Books

1. Russell, Stuart, J. and Norvig, Peter, *Artificial Intelligence - A Modern Approach*, Pearson, 4th edition, 2020.
2. Rich, Elaine and Knight, Kelvin, *Artificial Intelligence*, 3rd edition, Tata McGraw Hill, 2010.
3. Bratko, Ivan, *Prolog Programming for Artificial Intelligence*, Addison-Wesley, Pearson Education, 4th edition, 2012.

Reference Books

(i) Kaushik, Saroj, *Artificial Intelligence*, Cengage Learning India, 2011.
(ii) Patterson, DAN,W, *Introduction to A.I. and Expert Systems* – PHI, 2007.
(iii) Clocksin, W., F. and Mellish, *Programming in PROLOG*, 5th edition, Springer, 2003.

Suggested Practical List

1. Write a program to implement two agents to communicate with each other with message passing in Python/Prolog.
2. Write a program to implement the Hill climbing search algorithm in Python/Prolog.
3. Write a program to implement the Best first search algorithm in Python/Prolog.
4. Write a program to implement A* search algorithm in Python/Prolog.
5. Write a program to implement the min-max search algorithm in Python/Prolog.
6. Write a program to solve the Water-Jug Problem in Python/Prolog.
7. Implement sudoku problem (minimum 9×9 size) using constraint satisfaction in Python/Prolog.
8. Write a prolog program to implement the family tree and demonstrate the family relationship.
9. Write a prolog program to implement knowledge representation using frames with appropriate examples.

10. Write a Prolog program to implement conc(L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.

11. Write a Prolog program to implement reverse(L, R) where List L is original and List R is reversed list.

12. Write a PROLOG program to generate a parse tree of a given sentence in English language assuming the grammar required for parsing.

13. Write a prolog program to recognize context free grammar $a^n b^n$.

14. Using any sample dataset, implement any classification problem in Python language.

15. In the above classification problem (program 14), generate explanations for the decision taken using LIME tool.

DSC 17: Machine Learning

Course Prerequisites

Mathematics for Computing (I II and III)

Course Objective

The course aims at introducing the basic concepts and techniques of machine learning so that a student can apply machine learning techniques to a problem at hand.

Course Learning Outcomes

On successful completion of this course, the student will be able to:

1. Differentiate between supervised and unsupervised learning tasks.
2. Appreciate the need of preprocessing, feature scaling and feature selection.
3. Understand the fundamentals of classification, regression and clustering
4. Implement various machine learning algorithms learnt in the course.

Syllabus

Unit 1 Introduction: Basic definitions and concepts, key elements, supervised and unsupervised learning, introduction to reinforcement learning, applications of ML.

Unit 2 Preprocessing: Feature scaling, feature selection methods. dimensionality reduction (Principal Component Analysis).

Unit 3 Regression: Linear regression with one variable, linear regression with multiple variables, gradient descent, over-fitting, regularization. Regression evaluation metrics.

Unit 4 Classification: Decision trees, Naive Bayes classifier, logistic regression, k-nearest neighbor classifier, perceptron, multilayer perceptron, neural networks, back-propagation algorithm, Support Vector Machine (SVM). Classification evaluation metrics.

Unit 5 Clustering: Approaches for clustering, distance metrics, K-means clustering, hierarchical clustering.

References

1. Mitchell, T.M. *Machine Learning*, McGraw Hill Education, 2017.

2. James, G., Witten. D., Hastie. T., Tibshirani., R. *An Introduction to Statistical Learning with Applications in R*, Springer, 2014.

3. Alpaydin, E. *Introduction to Machine Learning*, MIT press, 2009.

Additional References

(i) Flach, P., *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*, Cambridge University Press, 2015.

(ii) Christopher & Bishop, M., *Pattern Recognition and Machine Learning*, New York: Springer-Verlag, 2016.

(ii) Sebastian Raschka, *Python Machine Learning,* Packt Publishing Ltd, 2019.

Suggested Practical List

Use Python for practical labs for Machine Learning. Utilize publically available datasets from online repositories like https://data.gov.in/ and https://archive.ics.uci.edu/ml/datasets.php

For evaluation of the regression/classification models, perform experiments as follows:

- Scale/Normalize the data

- Reduce dimension of the data with different feature selection techniques
- Split datasets into training and test sets and evaluate the decision models
- Perform k-cross-validation on datasets for evaluation

Report the efficacy of the machine learning models as follows:

- MSE and R^2 score for regression models
- Accuracy, TP, TN, FP, TN, error, Recall, Specificity, F1-score, AUC for classification models

For relevant datasets make prediction models for the following

1. Naïve Bayes Classifier
2. Simple Linear Regression multiple linear regression
3. Polynomial Regression
4. Lasso and Ridge Regression
5. Logistic regression
6. Artificial Neural Network
7. *k*-NN classifier
8. Decision tree classification
9. SVM classification
10. K-Means Clustering
11. Hierarchical Clustering

DSC 18: Introduction to Parallel Programming

Course Prerequisites

Programming in C++, Data Structures, Computer System Architecture, Operating Systems

Course Objective

The course introduces the students to the basic concepts and techniques of parallel programming. It enables them to design and implement parallel algorithms. The course would give the students hands-on practice to write parallel programs using shared and distributed memory models using OpenMP and Message Passing Interface (MPI).

Course Learning Outcomes

On successful completion of this course, the student will be able to:

1. Appreciate the need of Parallel algorithms
2. Describe architectures for parallel and distributed systems.
3. Develop elementary parallel algorithms in shared memory models.
4. Develop elementary parallel algorithms in distributed memory models.

Syllabus

Unit I Introduction to Parallel Computing: Trends in microprocessor architectures, memory system performance, dichotomy of parallel computing platforms, physical organization of parallel platforms, communication costs in parallel machines, SIMD versus MIMD architectures, shared versus distributed memory, PRAM shared-memory model, distributed-memory model.

Unit II OpenMP programming for shared memory systems: Thread Basics, Controlling Thread and Synchronization Attributes, Multi-thread and multi-tasking, Context Switching, Basic OpenMP thread functions, Shared Memory Consistency Models and the Sequential Consistency Model, Race Conditions, Scoping variables, work-sharing constructs, critical sections, atomic operations, locks, OpenMP tasks, Introduction to tasks, Task queues and task execution, Accessing variables in tasks, Completion of tasks and scoping variables in tasks, Recursive task spawning and pitfalls

Unit III MPI programming for distributed memory systems: MPI basic communication routines (Introduction to MPI and basic calls, MPI calls to send and receive data, MPI call for broadcasting data, MPI Non-blocking calls, MPI Collectives (MPI Collectives and MPI broadcast, MPI Gathering and scattering collectives, MPI reduction and Alltoall collectives, MPI collectives design), Types of interconnects (Characterization of interconnects, Linear arrays, 2D mesh and torus, cliques)

Unit IV Applications: Matrix-matrix multiply, Odd-Even sorting, distributed histogram, Breadth First search, Dijkstra's algorithm

References

1. Grama, A., Gupta, A., Karypis, G., Kumar, V., *Introduction to Parallel Computing*, 2nd edition, Addison-Wesley, 2003.
2. Quinn, M. *Parallel Programming in C with MPI and OpenMP*, 1st Edition, McGraw-Hill, 2017.
3. Revdikar, L., Mittal, A., Sharma, A., Gupta, S., *A Naïve Breadth First Search Approach Incorporating Parallel Processing Technique For Optimal Network Traversal*, International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 5, May 2016.

Suggested Practical List

1. Implement Matrix-Matrix Multiplication in parallel using OpenMP

2. Implement distributed histogram Sorting in parallel using OpenMP
3. Implement Breadth First Search in parallel using OpenMP
4. Implement Dijkstra's Algorithm in parallel using OpenMP

DSC 19: Compiler Design

Course Objective

The basic objective of the compiler design course is to understand the basic principles of compiler design, its various constituent parts, algorithms and data structures required to be used in the compiler. It also aims to understand the use of basic compiler building tools.

Course Learning Outcomes

On successful completion of the course, the students will be able to:

1. Explain the concepts and different phases of compilation.
2. Represent language tokens using regular expressions and context free grammars.
3. Describe the working of lexical analyzers.
4. Understand the working of different types of parsers and parse a particular string.
5. Describe intermediate code representations using syntax trees and DAG's as well as use this knowledge to generate intermediate code in the form of three address code representations.
6. Apply optimization techniques to intermediate code and generate machine code for high level language program.
7. Use Lex and Yacc automated compiler generation tools.

Syllabus

Unit 1 Introduction: Overview of compilation, Phases of a compiler.

Unit 2 Lexical Analysis: Role of a Lexical analyzer, Specification and recognition of tokens, Symbol table, Error reporting, Regular expressions and definitions , Lexical Analyzer Generator-Lex.

Unit 3 Syntax Analysis: CFGs, left recursion, left factoring, Top-down parsing- LL parser,

Bottom-up parsing- LR parser, Parser Generator-yacc.

Unit 4 Intermediate representations: Syntax Directed Definitions, Evaluation Orders for Syntax Directed Definitions, Intermediate Languages: Syntax Tree, Three Address Code, Types and Declarations, Translation of Expressions, loops and conditional statements, Type Checking.

Unit 5 Storage organization & Code generation: Activation records, stack allocation, Issues in Code Generation – Design of a simple Code Generator.

Unit 6 Code optimization : Principal sources of optimization, Peephole optimization.

References

1. Aho, A., Lam, M., Sethi, R., & Ullman, J. D. *Compilers: Principles, Techniques, and Tools*, 2nd edition, Addison Wesley, 2006.

Additional References

(i) V Raghvan, *Principles of Compiler Design*, TMH, 2010.

(ii) Santanu Chattopadhayay, *Compiler Design*, PHI, 2005.

Suggested Practical List

1. Write a Lex program to count the number of lines and characters in the input file.
2. Write a Lex program to count the number of vowels and consonants in a given string
3. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e, and so on z by c.
4. Write a Lex program that finds the longest word (defined as a contiguous string of upper and lower case letters) in the input.
5. Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.
6. Write a Lex program to count the number of words, characters, blank spaces and lines in a C file.
7. Write a Lex specification program that generates a C program which takes a string "abcd" and prints the following output
   abcd
   abc
   a
8. Write a Lex program to recognize a valid arithmetic expression.
9. Write a YACC program to find the validity of a given expression (for operators + - * and /)A program in YACC which recognizes a valid variable which starts with a letter followed by a digit. The letter should be in lowercase only.

10. Write a program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).
11. Write a program in YACC to recognize the string „abbb", „ab" „a" of the language (an b n , n>=1).
12. Write a program in YACC to recognize the language (an b , n>=10). (output to say input is valid or not)

Additional Suggestive list of Practicals (Can be implemented in C++/Python)

1. Write a program to implement DFAs that recognize identifiers, constants, and operators of the mini language.
2. Write a program Design a Lexical analyzer for the above language. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Identifiers may be of restricted length.
3. Write a program to check the types of expressions in a language.
4. Write a translator to translate a 3-address code into assembly code.

DSC 20: Information Security

Course Objective

The goal of this course is to make a student learn basic principles of information security. Over the due course of time, the student will be familiarized with cryptography, authentication and access control methods along with software security. Potential security threats and vulnerabilities of systems are also discussed along with their impacts and countermeasures. This course also touches upon the implications of security in cloud and Internet of Things (IoT).

Course Learning Outcomes

On successful completion of this course, a student will be able to

1. Identify the major types of threats to information security.
2. Describe the role of cryptography in security.
3. Discover the strengths and weaknesses of private and public key cryptosystems.
4. Identify and apply various access control and authentication mechanisms.
5. Discuss data and software security and related issues.
6. Explain network security threats and attacks.
7. Articulate the need for security in cloud and IoT.

Syllabus

Unit 1 Overview: Computer Security Concepts, Threats, Attacks, and Assets, Security Functional Requirements, Fundamental Security Design Principles, Attack Surfaces and Attack Trees.

Unit 2 Cryptographic Techniques: Confidentiality with Symmetric Encryption, Message Authentication and Hash Functions, Public-Key Encryption, Digital Signatures and Key Management, Random and Pseudorandom Numbers, Practical Application: Encryption of Stored Data.

Unit 3 Data Security: User authentication and Access Control, Database and Data Center Security

Unit 4 Software Security: Types of Malicious Software, Threats, Viruses, Worms, SPAM E-Mail, Trojans, Payload — System Corruption, Payload — Attack Agent — Zombie, Bots, Payload — Information Theft — Keyloggers, Phishing, Spyware, Payload — Stealthing — Backdoors, Rootkits, Countermeasures. Overflow Attacks - Stack Overflows, Defending Against Buffer Overflows, Other Forms of Overflow Attacks. Handling Program Input, Writing Safe Program Code, Interacting with the Operating System and Other Programs.

Unit 5 Network Security: Denial-of-Service Attacks, Flooding Attacks, Distributed Denial-of-Service Attacks, Overview of Intrusion Detection, Honeypots, Firewalls, Secure Email and S/MIME, Secure Sockets Layer (SSL) and Transport Layer Security (TLS), HTTPS, IPv4 and IPv6 Security, Public-Key Infrastructure.

Unit 6 Wireless, Cloud and IoT Security: Cloud Computing, Cloud Security Concepts, Cloud Security Approaches, The Internet of Things, IoT Security. Wireless Security Overview, Mobile Device Security, IEEE 802.11 Wireless LAN.

References

1. Stallings, W. and Brown L., *Computer Security: Principles and Practice*, 4th edition, Pearson Education, 2018.

Additional References

(i) Pfleeger C.P., Pfleeger S.L., Margulies J. *Security in Computing*, 5th edition, Prentice Hall, 2015.

(ii) Lin S., Costello D.J., *Error Control Coding: Fundamentals and applications,* 2nd edition, Pearson Education, 2004.

(iii) Stallings W. *Cryptography and network security*, 7th edition, Pearson Education, 2018.

(iv) Berlekamp E. *Algebraic Coding Theory*, World Scientific Publishing Co., 2015.
(v) Stallings W. *Network security essentials Applications and Standards*, 6th edition, Pearson Education, 2018.
(vi) Whitman M.E., Mattord H.J., *Principle of Information Security*, 6th edition, Cengage Learning, 2017.
(vii) Bishop M., *Computer Security: Art and Science*, 2nd Revised Edition, Pearson Education, 2019.
(viii) Anderson R.J., *Security Engineering: A guide to building Dependable Distributed Systems*, 2nd Edition, John Wiley & Sons, 2008.

Suggested Practical List

1. Demonstrate the use of Network tools: ping, ipconfig, ifconfig, tracert, arp, netstat, whois.
2. Use of Password cracking tools : John the Ripper, Ophcrack. Verify the strength of passwords using these tools.
3. Use nmap/zenmap to analyze a remote machine.
4. Use Burp proxy to capture and modify the message.
5. Implement caesar cipher substitution operation.
6. Implement monoalphabetic and polyalphabetic cipher substitution operation.
7. Implement playfair cipher substitution operation.
8. Implement hill cipher substitution operation.
9. Implement rail fence cipher transposition operation.
10. Implement row transposition cipher transposition operation.
11. Implement product cipher transposition operation.
12. Illustrate the Ciphertext only and Known plaintext attacks.
13. Implement a stream cipher technique.