VIDYASAGAR UNIVERSITY

Midnapore, West Bengal

PROPOSED CURRICULUM&SYLLABUS (DRAFT) OF

BACHELOR OF SCIENCE (HONOURS)

MAJOR IN COMPUTER SCIENCE

4-YEAR UNDERGRADUATE PROGRAMME

(w.e.f. Academic Year 2023-2024)

Based on

Curriculum & Credit Framework for Undergraduate Programmes

(CCFUP), 2023& NEP, 2020

VIDYASAGAR UNIVERSITY
BACHELOR OF SCIENCE (HONOURS) MAJOR IN COMPUTER SCIENCE
(under CCFUP, 2023)

Level	YR.	SEM	Course Type	Course Code	Course Title	Credit	L-T-P	Marks		
								CA	ESE	TOTAL
B.Sc. (Hons.)	2nd	III	**SEMESTER-III**							
			Major-3	**COSHMJ03**	**T: Data Structure; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			Major-4	**COSHMJ04**	**T: Computer Architecture; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			SEC	**COSSEC03**	**P: PYTHON**	**3**	**0-0-3**	**10**	**40**	**50**
			AEC	**AEC03**	Communicative English -2 (***common for all programmes***)	**2**	**2-0-0**	**10**	**40**	**50**
			MDC	**MDC03**	Multidisciplinary Course -3 (***to be chosen from the list***)	**3**	**3-0-0**	**10**	**40**	**50**
			Minor-3 (Disc.-I)	**COSMIN03**	**T: Digital Logic P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
					Semester-III Total	**20**				**375**
		IV	**SEMESTER-IV**							
			Major-5	**COSHMJ05**	**T: OOPs using C++; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			Major-6	**COSHMJ06**	**T: Operating System; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			Major-7	**COSHMJ07**	**T: Computer Network; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			AEC	**AEC04**	MIL-2 (***common for all programmes***)	**2**	**2-0-0**	**10**	**40**	**50**
			Minor-4 (Disc.-II)	**COSMNI04**	**T: Data Structure; P: Practical**	**4**	**3-0-1**	**15**	**60**	**75**
			Summer Intern.	**INT**	**Internship/ Apprenticeship**	**4**	**0-0-4**	**-**	**-**	**50**
					Semester-IV Total	**22**				**400**
					TOTAL of YEAR-2	**42**				**775**

MJ = Major, MI = Minor Course, SEC = Skill Enhancement Course, AEC = Ability Enhancement Course, MDC = Multidisciplinary Course, CA= Continuous Assessment, ESE= End Semester Examination, T = Theory, P= Practical, L-T-P = Lecture-Tutorial-Practical, MIL = Modern Indian Language

MAJOR (MJ)

MJ-3: Data Structure **Credits 04(Full Marks: 75)**

OBJECTIVE OF THE COURSE

- Introduce fundamental concepts and importance of data structures in computing.
- Teach implementation of linear data structures such as arrays, linked lists, stacks, and queues.
- Explore non-linear data structures including trees, graphs, and heaps.
- Emphasize analyzing algorithm efficiency in terms of time and space complexity.
- Develop skills in designing and implementing efficient algorithms.
- Demonstrate real-world applications of data structures in software development for context and relevance.
- Provide hands-on programming experience with languages like C or Python.
- Enhance analytical and problem-solving skills through practical assignments.
- Prepare students for advanced topics in computer science and software engineering by building a strong foundation in complex data structures.

OUTCOME OF THE COURSE

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

- Describe the role of data structures in organizing and managing data efficiently.
- Identify the appropriate data structure for a given problem.
- Write programs to implement and manipulate **arrays**, **linked lists**, **stacks**, and **queues**.
- Implement and use **trees** (e.g., binary trees, AVL trees), **graphs**, and **heaps**.
- Design and implement algorithms for **sorting** (e.g., quicksort, mergesort), **searching** (e.g., binary search), and **traversal** (e.g., BFS, DFS).
- Apply these algorithms to real-world scenarios.
- Write, debug, and optimize programs in **C** or **Python** using data structures and algorithms.
- Use debugging tools and techniques to identify and fix errors.
- Break down complex problems into smaller components and solve them using appropriate data structures and algorithms.
- Demonstrate strong analytical and problem-solving skills.

MJ-3T: Data Structure **Credits 03**

Course contents:

Module- I Arrays **05 Hrs.**
Single and Multi-dimensional Arrays, Sparse Matrices (Array and Linked Representation)

Module- II Stacks **05 Hrs.**
Implementing single / multiple stack/s in an Array; Prefix, Infix and Postfix expressions, Utility and conversion of these expressions from one to another; Applications of stack; Limitations of Array representation of stack

Module- III Linked Lists **10 Hrs.**
Singly, Doubly and Circular Lists (Array and Linked representation); Normal and Circular representation of Stack in Lists; Self Organizing Lists; Skip Lists

Module- IV Queues **05 Hrs.**
Array and Linked representation of Queue, De-queue, Priority Queues

Module- V Recursion 05 Hrs.

Developing Recursive Definition of Simple Problems and their implementation; Advantages and Limitations of Recursion; Understanding what goes behind Recursion (Internal Stack Implementation)

Module- VI Trees 20 Hrs.

Introduction to Tree as a data structure; Binary Trees (Insertion, Deletion , Recursive and Iterative Traversals on Binary Search Trees); Threaded Binary Trees (Insertion, Deletion, Traversals); Height-Balanced Trees (Various operations on AVL Trees). Tree traversal techniques.

Module- VII Searching and Sorting 05 Hrs.

Linear Search, Binary Search, Comparison of Linear and Binary Search, Selection Sort, Insertion Sort, Bubble Sort, Quick Sort, Comparison of Sorting Techniques

Module- VIII Hashing 05 Hrs.

Introduction to Hashing, Efficiency of Rehash Methods, Resolving collision by Open Addressing, Coalesced Hashing, Separate Chaining, Dynamic and Extendible Hashing.

MJ-3P: Data Structures Lab Credits 01

List of Assignments:

1. **Searching in a List**
   Write a program to search for an element in a list. Provide the user with the option to perform either **Linear Search** or **Binary Search**. Use **template functions** to make the program generic.
2. **Sorting a List**
   Write a program using **templates** to sort a list of elements. Allow the user to choose between **Insertion Sort**, **Bubble Sort**, or **Selection Sort** for sorting.
3. **Linked List Implementation:** Implement a **Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list
   - Concatenating two linked lists (implement both a function and an overloaded + operator).
4. **Doubly Linked List Implementation:** Implement a **Doubly Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list.
5. **Circular Linked List Implementation:** Implement a **Circular Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list.
6. **Stack Operations Using Linked List**
   Implement **Stack** operations (push, pop, peek, etc.) using a **Linked List** implementation.
7. **Stack Operations Using Array**

Implement **Stack** operations (push, pop, peek, etc.) using an **Array** implementation. Use **templates** to make the stack generic.

8. **Queue Operations Using Circular Array**
   Implement **Queue** operations (enqueue, dequeue, etc.) using a **Circular Array** implementation.
   Use **templates** to make the queue generic.
9. **Double-Ended Queue (Deque) Operations**
   Create and perform operations on a **Double-Ended Queue (Deque)** using a **Linked List** implementation.
10. **Polynomial Operations Using Linked List**
    Write a program to represent a polynomial using a **Linked List** and perform operations such as adding two polynomials.
11. **Factorial and Factors**
    Write a program to calculate the factorial and compute the factors of a given number:
    (i) Using **recursion**
    (ii) Using **iteration**.
12. **Fibonacci Series**
    Write a program to display the Fibonacci series:
    (i) Using **recursion**
    (ii) Using **iteration**.
13. **GCD Calculation**
    Write a program to calculate the GCD of two numbers:
    (i) Using **recursion**
    (ii) Without recursion.
14. **Binary Search Tree (BST) Operations**
    Create a **Binary Search Tree** and implement the following operations:
    (a) Insertion (both **recursive** and **iterative** implementations)
    (b) Deletion by copying
    (c) Deletion by merging
    (d) Search for a number in the BST
    (e) Display **preorder**, **postorder**, and **inorder** traversals (recursively)
    (f) Display **preorder**, **postorder**, and **inorder** traversals (iteratively)
    (g) Display level-by-level traversals
    (h) Count the number of **non-leaf nodes** and **leaf nodes**
    (i) Display the height of the tree
    (j) Create a mirror image of the tree
    (k) Check if two BSTs are equal.
15. **Sparse Matrix Conversion**
    Write a program to convert a **Sparse Matrix** into its non-zero form and vice versa.
16. **Reverse Stack Using Additional Stack**
    Write a program to reverse the order of elements in a stack using an **additional stack**.
17. **Reverse Stack Using Additional Queue**
    Write a program to reverse the order of elements in a stack using an **additional queue**.
18. **Diagonal Matrix Implementation**
    Implement a **Diagonal Matrix** using a one-dimensional array.
19. **Lower Triangular Matrix Implementation**
    Implement a **Lower Triangular Matrix** using a one-dimensional array.
20. **Upper Triangular Matrix Implementation**

Implement an **Upper Triangular Matrix** using a one-dimensional array.

21. **Symmetric Matrix Implementation**
    Implement a **Symmetric Matrix** using a one-dimensional array.
22. **Threaded Binary Tree Implementation**
    Create a **Threaded Binary Tree** based on in-order traversal. Implement operations such as:
    - Finding the successor/predecessor of an element
    - Inserting an element
    - Performing in-order traversal.
23. **AVL Tree Operations**
    Implement various operations (searching, insertion, deletion) on an **AVL Tree**.
24. **Sorting Algorithms**
    Implement the following sorting algorithms:
    - Selection Sort
    - Insertion Sort
    - Bubble Sort
    - Quick Sort.

Suggested Readings:

1. Adam Drozdek, "Data Structures and algorithm in C++", Third Edition, Cengage Learning, 2012.
2. SartajSahni, Data Structures, "Algorithms and applications in C++", Second Edition, Universities Press, 2011.
3. Aaron M. Tenenbaum, Moshe J. Augenstein, Yedidyah Langsam, "Data Structures Using C and C++:, Second edition, PHI, 2009.
4. Robert L. Kruse, "Data Structures and Program Design in C++", Pearson, 1999.
5. D.S Malik, Data Structure using C++, Second edition, Cengage Learning, 2010.
6. Mark Allen Weiss, "Data Structures and Algorithms Analysis in Java", Pearson Education, 3rd edition, 2011
7. Aaron M. Tenenbaum, Moshe J. Augenstein, Yedidyah Langsam, "Data Structures Using Java, 2003.
8. Robert Lafore, "Data Structures and Algorithms in Java, 2/E", Pearson/ Macmillan Computer Pub, 2003
9. John Hubbard, "Data Structures with JAVA", McGraw Hill Education (India) Private Limited; 2 edition, 2009
10. Goodrich, M. and Tamassia, R. "Data Structures and Algorithms Analysis in Java", 4th Edition, Wiley, 2013
11. Herbert Schildt, "Java The Complete Reference (English) 9th Edition Paperback", Tata McGraw Hill, 2014.
12. D. S. Malik, P.S. Nair, "Data Structures Using Java", Course Technology, 2003.

MJ-4: Computer Architecture **Credits 04 (F.M.-75)**

OBJECTIVE OF THE COURSE

- Provide a comprehensive understanding of computer system design and functionality.
- Cover fundamental concepts of computer organization, including processor architecture, memory hierarchies, and input/output systems.
- Explore instruction set architectures (ISA) and their impact on hardware performance and efficiency.
- Delve into parallel processing and multi-core architectures to illustrate advancements in modern computing.
- Develop practical skills through laboratory sessions and programming assignments involving assembly language and hardware simulation tools.

OUTCOME OF THE COURSE

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

- Describe the fundamental components of a computer system, including **processors**, **memory**, and **I/O systems**.
- Compare and contrast different **instruction set architectures (ISA)**.
- Analyze the impact of ISA design on hardware performance and efficiency.
- Design and implement **data paths** and **control units** for a simple processor.
- Analyze how these architectures enhance system performance and scalability.
- Write, debug, and optimize **assembly language** programs.
- Use hardware simulation tools to test and validate programs.
- Design and optimize computer systems for real-world applications.
- Demonstrate the ability to work in teams on system design projects.

MJ-4T: Computer Architecture **Credits 03**

Module I: Introduction **20 Hrs.**

Logic gates, Boolean algebra, combinational circuits, circuit simplification, flip-flops and sequential circuits, decoders, multiplexers, registers, counters and memory units.

Module II: Data Representation and Basic Computer Arithmetic **10 Hrs.**

Number systems, complements, fixed and floating point representation, character representation, addition, subtraction, magnitude comparison, multiplication and division algorithms for integers

Module III: Basic Computer Organization and Design **8 Hrs.**

Computer registers, bus system, instruction set, timing and control, instruction cycle, memory reference, Organization of a basic single-bus computer system.

Module IV: Central Processing Unit **10 Hrs.**

Register organization, arithmetic and logical operations, Instruction formats, addressing modes, instruction codes, machine language, assembly language, RISC, CISC architectures, Hardwired and micro programmed control unit design.

Module V: Memory Organization **6 Hrs.**

Memory interfacing and addressing, cache memory organization.

Module VI: Input-Output Organization **6 Hrs.**

Input / Output: External Devices, I/O Modules, Programmed I/O, Interrupt-Driven I/O, Direct Memory Access, I/O Channels.

MJ-4P: Computer Architecture Lab **Credits 01**

Laboratory Assignments on Simulation/Hardware Kit

1. **8-bit Arithmetic Operations**
   Implement and simulate the following 8-bit arithmetic operations:
   - Addition
   - Multiplication
   - Division
2. **8-bit Register Design**
   Design and implement an **8-bit register** using simulation or hardware.
3. **Memory Unit Design and Operations**
   Design a **memory unit** and perform basic memory operations such as read and write.
4. **8-bit Simple ALU Design**
   Design and implement an **8-bit Arithmetic Logic Unit (ALU)** capable of performing basic arithmetic and logical operations.
5. **8-bit Simple CPU Design**
   Design and implement a basic **8-bit CPU** capable of executing simple instructions.
6. **Interfacing CPU and Memory**
   Interface the designed **CPU** with the **memory unit** and demonstrate data transfer and instruction execution.
7. **Microoperations and Instruction Set Design**
   - Create microoperations and associate them with instructions (excluding interrupts).
   - Design the **register set**, **memory**, and **instruction set**.
   - Use this machine for subsequent assignments in this section.

8. **Fetch Routine Implementation**
   Create and implement the **fetch routine** of the instruction cycle.
9. **Simulation of Register Reference Instructions**
   Simulate the machine to determine the contents of the following registers in hexadecimal after executing each of the given register reference instructions:

   AC (Accumulator), **E (Extend)**, **PC (Program Counter)**, **AR (Address Register)**, and **IR (Instruction Register)**.

   Instructions to Simulate:
   a. CLA (Clear AC)
   b. CLE (Clear E)
   c. CMA (Complement AC)
   d. CME (Complement E)

e. CIR (Circulate Right AC and E)
f. CIL (Circulate Left AC and E)
g. INC (Increment AC)
h. SPA (Skip if AC is Positive)
i. SNA (Skip if AC is Negative)
j. SZA (Skip if AC is Zero)
k. SZE (Skip if E is Zero)
l. HLT (Halt)

Initialization:

- AC = $(A937)_{16}$
- PC = $(022)_{16}$
- E = 1

10. **Simulation of Memory Reference Instructions (Direct Addressing)**
Simulate the machine for the following memory-reference instructions with **I = 0** (direct addressing) and **address part = 082**.

- Store the instruction at address **022** in RAM.
- Initialize the memory word at address **082** with the operand **B8F2**.
- Initialize **AC** with **A937**.

Instructions to Simulate:

a. ADD
b. AND
c. LDA
d. STA
e. BUN
f. BSA
g. ISZ

Determine the contents of the following registers in hexadecimal after execution:

AC, DR (Data Register), PC, AR, and IR.

11. **Simulation of Memory Reference Instructions (Indirect Addressing)**
Simulate the machine for the memory-reference instructions with **I = 1** (indirect addressing) and **address part = 082**.

- Store the instruction at address **026** in RAM.
- Initialize the memory word at address **082** with the value **298**.
- Initialize the memory word at address **298** with the operand **B8F2**.
- Initialize **AC** with **A937**.

Determine the contents of the following registers in hexadecimal after execution:

AC, DR, PC, AR, and IR.

12. **Machine Modification with New Instruction Format**

Modify the machine created in **Practical 1** according to the following instruction format:

Instruction Format:

0 2 3 4 15

Opcode I Address

- The instruction format contains a **3-bit opcode**, a **1-bit addressing mode (I)**, and a **12-bit address**.
- Addressing modes:
  - **I = 0**: Direct addressing
  - **I = 1**: Indirect addressing

Tasks:

a. Create a new **1-bit register (I)**.

b. Create two new microinstructions:

i. Check the opcode to determine the instruction type (Memory Reference/Register Reference/Input-Output) and jump accordingly.

ii. Check the **I bit** to determine the addressing mode and jump accordingly.

Suggested Readings:

1. M. Mano, Computer System Architecture, Pearson Education 1992
2. W. Stallings, Computer Organization and Architecture Designing for Performance, 8 Edition, Prentice Hall of India,2009
3. M.M. Mano , Digital Design, Pearson Education Asia,2013
4. Carl Hamacher, Computer Organization, Fifth edition, McGrawHill, 2012.

MJ-5: OOPs using C++ **Credits 04 (F.M.-75)**

OBJECTIVE OF THE COURSE

- Provide a deep understanding of object-oriented programming principles and their application using C++.
- Cover fundamental concepts such as classes, objects, inheritance, polymorphism, and encapsulation.
- Emphasize the creation and manipulation of complex data structures through dynamic memory management and operator overloading.
- Develop practical programming skills through hands-on projects and assignments involving real-world applications.
- Provide experience in debugging and testing C++ applications to ensure reliability and performance.
- Prepare students for advanced programming and software development roles.

OUTCOME OF THE COURSE

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

- Describe the core concepts of OOP, including classes, objects, inheritance, polymorphism, and encapsulation.
- Create classes and objects to model real-world entities.
- Implement constructors, destructors, and member functions to define object behavior.
- Use inheritance to create hierarchical class structures.
- Implement polymorphism through function overriding and virtual functions.
- Allocate and deallocate memory dynamically using new and delete operators.
- Overload operators to perform custom operations on user-defined data types.
- Utilize the Standard Template Library (STL) for efficient data handling (e.g., vectors, lists, maps).
- Design and implement C++ programs to solve real-world problems.
- Apply OOP principles to create modular, reusable, and maintainable code.
- Write test cases to ensure the reliability and performance of applications.

MJ-5: OOPs using C++ **Credits 03**

Module-I: Introduction to OOPs and C++ Element **15 Hrs.**

Structured vs. Object Oriented Programming, Object Oriented Programming Concepts, Benefits of Object oriented programming, Object Oriented Languages, Structure of a C++ program, Data Types, Operators and Control Structures, Iteration / Loop Construct, Arrays, Functions (User defined Function, Inline Function, Function Overloading), User Defined Data Types (Structure, Union and Enumeration).

Module II: Class, Object, Constructor & Destructor: **15 Hrs.**

Defining Classes, Encapsulation, Instantiating Objects, Member Functions, Accessibility labels, Static Members, Friend Function, Purpose of Constructors, Default Constructor, Parameterized Constructors, Copy Constructor, Destructor.

Module III: Pointer, Polymorphism & Inheritance: 20 Hrs.

Pointer (Pointer to Object, this Pointer, Pointer to Derive Class), Introduction to Polymorphism (Compile time Polymorphism, Run time Polymorphism), Operator Overloading, Overloading Unary and Binary Operators, Virtual Function, Pure Virtual Functions, Inheritance (Single Inheritance, Multiple Inheritance, Multilevel Inheritance, Hierarchical Inheritance, Hybrid Inheritance), Virtual Base Class, Abstract Class.

Module IV: Exception Handling: 10 Hrs.

Exceptions in C++ Programs, Try and Catch Expressions, Exceptions with arguments

Suggested Readings:

1. HerbtzSchildt, "C++: The Complete Reference", Fourth Edition, McGraw Hill.2003
2. BjarneStroustrup, "The C++ Programming Language", 4th Edition, Addison-Wesley ,2013.
3. BjarneStroustroup, "Programming -- Principles and Practice using C++", 2nd Edition, Addison-Wesley 2014.
4. E Balaguruswamy, "Object Oriented Programming with C++", Tata McGraw-Hill Education, 2008.
5. Paul Deitel, Harvey Deitel, "C++ How to Program", 8th Edition, Prentice Hall, 2011.
6. John R. Hubbard, "Programming with C++", Schaum's Series, 2nd Edition, 2000.
7. Andrew Koeni, Barbara, E. Moo, "Accelerated C++", Published by Addison-Wesley ,
1. 2000. 7. Scott Meyers, "Effective C++", 3rd Edition, Published by Addison-Wesley,
2. 2005.
8. Harry, H. Chaudhary, "Head First C++ Programming: The Definitive Beginner's Guide",
3. First Create space Inc, O-D Publishing, LLC USA.2014
9. Walter Savitch, "Problem Solving with C++", Pearson Education, 2007.
10. Stanley B. Lippman, JoseeLajoie, Barbara E. Moo, "C++ Primer", Published by
4. Addison-Wesley, 5th Edition, 2012
11. E Balagurusamy , Object Oriented Programming with C++, 5 th edition, Tata McGraw, 2011.
12. Deitel and Deitel , “C++: How to Program”, 9th Edition, Pearson, 2013.

MJ-5P: OOPs using C++ (Lab) Credits 01

C++ Programming Assignments

1. **Sum of Digits**
   Write a C++ program to find the sum of individual digits of a positive integer.
2. **Reverse a Number**
   Write a C++ program to print a given number in reverse order.
3. **Non-Fibonacci Numbers**
   Write a C++ program to print the first **100 non-Fibonacci numbers**.
4. **Decimal to Hexadecimal Conversion**
   Write a C++ program to convert a decimal number into a hexadecimal number.
5. **Binary Search**
   Write a C++ program to search for an element in an array using the **binary search** technique.
6. **Compound Interest with Default Arguments**
   Write a C++ program to calculate **compound interest** in a bank using **default arguments**.

7. **Student Details Using Classes and Objects**
   Write a C++ program to display student details using **classes and objects** (store objects as an array).
8. **Stack Implementation Using Array**
   Write a C++ program to implement a **stack** using an array.
9. **Matrix Multiplication with Dynamic Memory Allocation**
   Write a C++ program for **matrix multiplication** using:
   - Dynamic memory allocation
   - Copy constructor
   - Overloading of the assignment operator.
10. **Matrix Transpose**
    Write a C++ program to read a two-dimensional matrix and display its **transpose**.
11. **Inline Function Implementation**
    Write a C++ program to implement an **inline function**.
12. **Constructor and Destructor Implementation**
    Write a C++ program to implement **constructor** and **destructor**.
13. **Copy Constructor Functionality**
    Write a C++ program to implement the functionalities of a **copy constructor**.
14. **Constructor Overloading for Account Details**
    Write a C++ program to display the **account number** and **balance** using **constructor overloading**.
15. **Volume Calculation Using Function Overloading**
    Write a C++ program to find the volume of:
    - Cube
    - Rectangle
    - Cylinder

    using **function overloading**.
16. **Operator Overloading Using Friend Functions**
    Write a C++ program to overload the ++ and -- operators using **friend functions**.
17. **Complex Number Addition Using Operator Overloading**
    Write a C++ program to add two complex numbers using **binary operator overloading**.
18. **Inheritance Implementation**
    Write a C++ program to implement:
    - **Single inheritance**
    - **Multilevel inheritance**.
19. **Multiple Inheritance with Virtual Functions**
    Write a C++ program to draw a **rectangle**, **square**, and **circle** using **multiple inheritance** with **virtual functions**.
20. **Hybrid Inheritance Implementation**
    Write a C++ program to implement **hybrid inheritance**.
21. **Student Details Using Virtual Base Class**
    Write a C++ program to display student details using a **virtual base class**.
22. **Pure Virtual Function Implementation**
    Write a C++ program to implement a **pure virtual function**.

Reference Books:

1. E. Balaguruswami-Object Oriented programming with C++
2. Kris James-Success with C++
3. David Parsons-Object Oriented programming with C++
4. D. Ravichandran-Programming in C++
5. Dewhurst and Stark-Programming in C++

MJ-6: Operating System **Credits 04 (F.M.-75)**

OBJECTIVE OF THE COURSE

- Provide a comprehensive understanding of fundamental concepts and functions of modern operating systems.
- Cover the architecture and components of operating systems, including process management, memory management, file systems, and input/output systems.
- Emphasize the role of operating systems in resource allocation and system security.
- Offer hands-on experience in implementing and configuring operating system features through labs work.
- Equip students with the skills to analyze, design, and optimize operating systems,

OUTCOME OF THE COURSE

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

- Describe the architecture and components of modern operating systems.
- Explain the functions of process management, memory management, file systems, and I/O systems.
- Understand and implement process scheduling algorithms (e.g., FCFS, Round Robin, SJF).
- Explain memory management techniques like paging**,** segmentation**,** and virtual memory**.**
- Implement process synchronization mechanisms like semaphores .
- Understand and resolve issues like deadlocks and race conditions**.**
- Understand the structure and organization of file systems.
- Implement file system operations like file creation**,** deletion, and access control**.**
- Explain how operating systems allocate resources like CPU, memory, and I/O devices.
- Implement basic security measures like user authentication and access control**.**

MJ-6T: Operating System **Credits 03**

Module I: Introduction **10 Hrs.**

Basic OS functions, resource abstraction, types of operating systems–multiprogramming systems, batch systems , time sharing systems; operating systems for personal computers & workstations, process control & real time systems.

Case study on Linux system **6 Hrs.**

- Cloud computing (3 lectures)
- Linux evolution and Linux distros (2 lectures)
- Linux file system (1 lecture)

Module II: Operating System Organization **6 Hrs.**

Processor and user modes, kernels, system calls and system programs.

Module III: Process Management **16 Hrs.**

System view of the process and resources, process abstraction, process hierarchy, threads, threading issues, thread libraries; Process Scheduling, non-pre-emptive and pre-emptive scheduling algorithms; concurrent processes, critical section, semaphores, methods for inter- process communication; deadlocks.

Module IV: Memory Management **10 Hrs.**

Physical and virtual address space; memory allocation strategies – fixed and variable partitions, paging, segmentation, virtual memory

Module V: File and I/O Management **8 Hrs.**

Directory structure, file operations, file allocation methods, device management.

Module VI: Protection and Security **4 Hrs.**

Policy mechanism, Authentication, Internal access Authorization.

MJ-6P: Operating System Lab **Credits 01**

Programming Assignments on Linux System Programming and Scheduling Algorithms

1. **Parent and Child Processes Using fork() and exec()**
   Write a program using fork() and/or exec() commands to demonstrate the following scenarios:
   a. Parent and child execute the **same program** and the **same code**.
   b. Parent and child execute the **same program** but **different code**.
   c. Before terminating, the parent waits for the child to finish its task using wait().
2. **Linux Kernel and CPU Information**
   Write a program to report the behavior of the Linux kernel, including:
   - Kernel version
   - CPU type and model (CPU information).

3. **Linux Kernel and Memory Information**
   Write a program to report the behavior of the Linux kernel, including:
   - Configured memory
   - Amount of free and used memory (memory information).

4. **File Details**
   Write a program to print details of a file, including:
   - Owner access permissions
   - File access time
     The file name should be provided as a command-line argument.

5. **File Copy Using System Calls**
   Write a program to copy files using system calls (e.g., open(), read(), write(), close()).
6. **FCFS Scheduling Algorithm**
   Write a program to implement the **First-Come-First-Serve (FCFS)** CPU scheduling algorithm.
7. **Round Robin Scheduling Algorithm**
   Write a program to implement the **Round Robin (RR)** CPU scheduling algorithm.
8. **SJF Scheduling Algorithm**
   Write a program to implement the **Shortest Job First (SJF)** CPU scheduling algorithm.

9. **Sum of Numbers Using Thread Library**
   Write a program to calculate the sum of n numbers using the thread library (e.g., pthread in Linux).
10. **Memory Allocation Strategies**
    Write a program to implement the following memory allocation strategies:
    - **First-Fit**
    - **Best-Fit**
    - **Worst-Fit**.

Suggested Readings:

1. A Silberschatz, P.B. Galvin, G. Gagne, Operating Systems Concepts, 8th Edition, John Wiley Publications 2008.
2. A.S. Tanenbaum, Modern Operating Systems, 3rd Edition, Pearson Education 2007.
3. G. Nutt, Operating Systems: A Modern Perspective, 2nd Edition Pearson Education 1997.
4. W. Stallings, Operating Systems, Internals & Design Principles, 5th Edition, Prentice Hall of India. 2008.
5. M. Milenkovic, Operating Systems- Concepts and design, Tata McGraw Hill 1992.

MJ-7: Computer Network **Credits 04 (F.M.-75)**

OBJECTIVE OF THE COURSE

The course aims to provide students with a comprehensive understanding of computer networks, focusing on both theoretical concepts and practical applications. By the end of the course, students will:

- Gain a foundational knowledge of network architectures, including the OSI (Open Systems Interconnection) and TCP/IP (Transmission Control Protocol/Internet Protocol) models.
- Understand the functions of key network devices such as routers, switches, and hubs.
- Learn the principles of network design, addressing, and subnetting.
- Develop the skills to analyze, design, and implement network solutions.

OUTCOME OF THE COURSE

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

- Describe the OSI and TCP/IP models and their relevance in network communication.
- Differentiate between the functions of each layer in these models.
- Recognize the roles of routers, switches, hubs, and other network devices.
- Configure and troubleshoot these devices in a network environment.
- Explain the working of protocols like HTTP, FTP, TCP, and UDP.
- Design network topologies and perform subnetting for efficient IP addressing.
- Configure and manage networks using industry-standard tools.
- Diagnose and resolve common network problems using troubleshooting techniques.
- Implement basic network security measures to protect data and devices.
- Understand the importance of encryption, firewalls, and secure protocols.

MJ-7T: Computer Network **Credits 03**

Module I: Introduction to Computer Networks **8 Hrs.**
Network definition; network topologies; network classifications; network protocol; layered network architecture; overview of OSI reference model; overview of TCP/IP protocol suite.

Module II: Data Communication Fundamentals and Techniques **10 Hrs.**
Analog and digital signal; data-rate limits; digital to digital line encoding schemes; pulse code modulation; parallel and serial transmission; digital to analog modulation-; multiplexing techniques- FDM, TDM; transmission media.

Module III: Networks Switching Techniques and Access mechanisms **10 Hrs.**
Circuit switching; packets witching- connectionless datagram switching, connection-oriented virtual circuit switching; dial-up modems; digital subscriber line; cable TV for data transfer.

Module IV: Data Link Layer Functions and Protocol **10 Hrs.**
Error detection and error correction techniques; data-link control- framing and flow control; error recovery protocols- stop and wait ARQ, go-back-n ARQ; Point to Point Protocol on Internet.

Module V: Multiple Access Protocol and Networks **5 Hrs.**
CSMA/CD protocols; Ethernet LANS; connecting LAN and back-bone networks- repeaters, hubs, switches, bridges, router and gateways;

Module VI: Networks Layer Functions and Protocols **6 Hrs.**

Routing; routing algorithms; network layer protocol of Internet- IP protocol, Internet control protocols.

Module VII: Transport Layer Functions and Protocols **6 Hrs.**
Transport services- error and flow control, Connection establishment and release – three way handshake;

Module VIII: Overview of Application layer protocol **5 Hrs.**
Overview of DNS protocol; overview of WWW &HTTP protocol.

MJ-7P: Computer Network Lab **Credits 01**

Programming Tasks Using C++/Python/Java

1. **Cyclic Redundancy Check (CRC) Error Detection**
   Simulate the **CRC error detection algorithm** for a noisy channel.
2. **Stop and Wait Protocol**
   Simulate and implement the **Stop and Wait Protocol** for a noisy channel.
3. **Go-Back-N Sliding Window Protocol**
   Simulate and implement the **Go-Back-N Sliding Window Protocol**.
4. **Selective Repeat Sliding Window Protocol**
   Simulate and implement the **Selective Repeat Sliding Window Protocol**.
5. **Distance Vector Routing Algorithm**
   Simulate and implement the **Distance Vector Routing Algorithm**.
6. **Dijkstra's Algorithm for Shortest Path Routing**
   Simulate and implement **Dijkstra's Algorithm** to find the shortest path in a network.
7. **Packet Capture and Analysis Using Wireshark**
   Perform experiments using **Wireshark** for:
   a. Filtering packets
   b. Inspecting packets.
8. **HDLC Frame Implementation**
   Write a program for an **HDLC frame** to perform:
   i. Bit stuffing
   ii. Character stuffing.
9. **Distance Vector Algorithm for Path Transmission**
   Write a program for the **Distance Vector Algorithm** to find a suitable path for data transmission.
10. **Dijkstra's Algorithm for Shortest Routing Path**
    Implement **Dijkstra's Algorithm** to compute the shortest routing path.
11. **CRC-CCITT Polynomial Implementation**
    Use the **CRC-CCITT polynomial** to obtain the CRC code. Verify the program for:
    a. Without error
    b. With error.
12. **Stop and Wait Protocol and Sliding Window Protocol**
    Implement the **Stop and Wait Protocol** and **Sliding Window Protocol**.
13. **Congestion Control Using Leaky Bucket Algorithm**
    Write a program for **congestion control** using the **Leaky Bucket Algorithm**.

Simulation Tasks Using NS2/NS3

1. **Wired Network: Two Nodes and Packet Transmission**
   Write a **TCL script** to connect two nodes and send packets in a wired network.

2. **STAR Topology with SFQ Queue Management**
   Write a **TCL script** for a given **STAR topology** using **Stochastic Fair Queuing (SFQ)** at intermediate nodes. Use different colors for packets originating from different nodes.
3. **RING Topology with Dynamic Configuration**
   Write a **TCL script** for a given **RING topology** in a wired network using a **For loop** to make the topology dynamic.
4. **TCP Connection in Wired Network**
   Write a **TCL script** for a given topology in a wired network using a **TCP connection** and sending data through the nodes.
5. **UDP Connection in Wired Network**
   Write a **TCL script** for a given topology in a wired network using a **UDP connection** and sending data through the nodes.
6. **Point-to-Point Network with Duplex Links**
   Implement a **three-node point-to-point network** with duplex links. Set the queue size, vary the bandwidth, and find the number of packets dropped.
7. **Ping/Traceroute Over a 6-Node Network**
   Implement the transmission of **ping messages/traceroute** over a network topology consisting of **6 nodes**. Find the number of packets dropped due to congestion.
8. **Ethernet LAN with Multiple Traffic Nodes**
   Implement an **Ethernet LAN** using **n nodes**. Set multiple traffic nodes and plot the congestion window for different source/destination pairs.
9. **Point-to-Point Network with Duplex Links (Repeated)**
   Implement a **three-node point-to-point network** with duplex links. Set the queue size, vary the bandwidth, and find the number of packets dropped.
10. **Ping/Traceroute Over a 6-Node Network (Repeated)**
    Implement the transmission of **ping messages/traceroute** over a network topology consisting of **6 nodes**. Find the number of packets dropped due to congestion.
11. **Ethernet LAN with Multiple Traffic Nodes (Repeated)**
    Implement an **Ethernet LAN** using **n nodes**. Set multiple traffic nodes and plot the congestion window for different source/destination pairs.
12. **Wireless LAN: Simple ESS and Performance Analysis**
    Implement a **simple ESS (Extended Service Set)** with transmitting nodes in a **wireless LAN**. Simulate and determine the performance with respect to packet transmission.

Suggested Readings:

1. B. A. Forouzan: Data Communications and Networking, Fourth edition, THM, 2007.
2. A. S. Tanenbaum: Computer Networks**,** Fourth edition**,** PHI, 2002.

MINOR (MI)

MI – 3: Digital Logic **Credits 04(Full Marks: 75)**

OBJECTIVE OF THE COURSE

- Learn the basics of binary, octal, decimal, and hexadecimal number systems and conversions between them.
- Master the principles of Boolean algebra, including logic operations, truth tables, and De Morgan's laws.
- Gain proficiency in simplifying Boolean expressions using techniques such as Karnaugh maps and the Quine-McCluskey method.
- Develop skills in designing basic combinational logic circuits, including adders, subtractors, multiplexers, and decoders.
- Understand the behaviour and design of sequential circuits, including flip-flops, latches, counters, and registers.
- Explore circuit minimization techniques to reduce complexity and cost in digital designs.

OUTCOME OF THE COURSE

- Learn and apply Boolean algebra principles to simplify and analyse logical expressions.
- Develop skills in designing and implementing combinational and sequential logic circuits like multiplexers, decoders, and flip-flops.
- Learn the operation, behaviour, and application of basic memory elements like SR latches, JK, D, and T flip-flops.
- Develop the ability to analyse and design sequential circuits, including counters and shift registers.

MI – 3T: Digital Logic **Credits 03**

Course contents:

Number systems: **15 Hrs.**

Positional number systems; Binary, Octal, Hexadecimal, and Decimal number systems; conversion of a number in one system to the other; Representation of signed numbers-signed magnitude, one's complement, 2's complement representation techniques, Merits of 2's complement representation scheme; Various binary codes - BCD, excess -3, Gray code, ASCII, EBCDIC; Binary arithmetic- addition, subtraction, multiplication, and division of unsigned binary numbers.

Boolean algebra: **15 Hrs.**

Fundamental of Boolean Expression: Definition of Boolean Algebra, Postulates, Basic Logic gates: (OR, AND, NOT); Universal Logic Gates: (NAND & NOR); Basic logic operations: logical sum (OR), logical product (AND), complementation (NOT), anti-coincidence (EX-OR) and coincidence (EX-NOR) operations: Truth tables of Basic gates; Boolean Variables and Expressions; De-Morgan’s theorem; Boolean expressions Simplification- Algebraic technique, Karnaugh map technique, 3 variable and 4 variable Karnaugh map.

Combinational Circuits: **15 Hrs.**

Half Adder, Full Adder (3-bit), Half Subtractor, Full Subtractor (3-bit), and construction using Basic Logic Gates (OR, AND, NOT) and Universal Logic Gates (NAND & NOR), Multiplexer, Encoders, Demultiplexer, and Decoder circuits.

Sequential Circuits: **15 Hrs.**

Latch, RS, D, JK, T Flip Flops; Race condition, Master Slave JK Flip Flop; Registers: Serial Input Serial Output (SISO), Serial Input Parallel Output (SIPO), Parallel input Serial Output (PISO), Parallel Input Parallel Output (PIPO), Universal Shift Registers; Counters: Asynchronous Counter, Synchronous Counter.

MI – 3P: Digital Logic Lab **Credits 01**

List of Experiments: Digital Logic Design

1. **Verification of Logic Gates**
   Verify the truth tables of the following two-input logic gates:
   (i) OR gate
   (ii) AND gate
   (iii) NOR gate
   (iv) NAND gate
   (v) Exclusive-OR (XOR) gate
   (vi) Exclusive-NOR (XNOR) gate.
2. **Combinational Circuit Design**
   Design a simple combinational circuit with **four variables**, obtain the **minimal expression**, and verify the truth table using a **Digital Trainer Kit**.
3. **3-to-8 Line Decoder/Demultiplexer**
   Verify the functional table of a **3-to-8 line decoder/demultiplexer**.
4. **4-Variable Logic Function Using Multiplexer**
   Verify a **4-variable logic function** using an **8-to-1 multiplexer**.
5. **Full Adder Circuit**
   Design a **full adder circuit** and verify its functional table.
6. **Flip-Flop Functional Tables**
   Verify the functional tables of the following flip-flops:
   (i) JK Edge-Triggered Flip-Flop
   (ii) JK Master-Slave Flip-Flop
   (iii) D Flip-Flop.
7. **4-Bit Ring Counter**
   Design a **4-bit ring counter** using **D Flip-Flops** or **JK Flip-Flops** and verify the output.
8. **4-Bit Johnson's Counter**
   Design a **4-bit Johnson's counter** using **D Flip-Flops** or **JK Flip-Flops** and verify the output.
9. **4-Bit Universal Shift Register**
   Verify the operation of a **4-bit universal shift register** for different modes of operation.
10. **MOD-8 Ripple Counter**

- Draw the circuit diagram of a **MOD-8 ripple counter**.
- Construct the circuit using **T Flip-Flops**.
- Test it with a low-frequency clock and sketch the output waveforms.

11. **MOD-8 Synchronous Counter**

- Design a **MOD-8 synchronous counter** using **T Flip-Flops**.
- Verify the result and sketch the output waveforms.

12. **Single-Bit Comparator and 7-Segment Display**

(a) Draw the circuit diagram of a **single-bit comparator** and test the output.

(b) Construct a **7-segment display circuit** using a **decoder** and **7-segment LED**, and test it.

Suggested Readings:

1. Morris Mano, Charles R. Kime, Logic and computer design fundamentals, Pearson Prentice Hall, 2004
2. Basavaraj,B., Digital fundamentals, New Delhi: Vikas Publishing House, 1999.
3. Kandel Langholz, Digital Logic Design, Prentice Hall, 1988.
4. Rafiquzzaman & Chandra, Modern Computer Architecture, West Pub. Comp., 1988.

MI-4: Data Structure **Credits 04(Full Marks: 75)**

OBJECTIVE OF THE COURSE

- Introduce students to fundamental concepts of data structures and algorithms, including the importance of data organization and management in solving computational problems efficiently.
- Provide hands-on experience with implementing data structures such as arrays, linked lists, stacks, queues, trees, heaps, graphs, and hash tables using a programming language.
- Show how to apply data structures to solve real-world problems. This involves understanding which data structure is appropriate for a given situation and how to manipulate data structures to optimize performance.
- Develop problem-solving skills by practicing the design and implementation of algorithms using appropriate data structures, focusing on improving solutions' efficiency.

OUTCOME OF THE COURSE

- Develop the ability to design and analyse algorithms, understand their time and space complexities using Big O notation, and make informed decisions about the most efficient algorithms to use in different scenarios.
- Enhance their problem-solving skills by applying data structures and algorithms to solve computational problems effectively and efficiently.
- Learn how to choose the appropriate data structures for specific applications, optimizing data storage, retrieval, and manipulation in software systems.
- Have a solid foundation for advanced topics in computer science, such as algorithms, machine learning, artificial intelligence, databases, and systems design, where efficient data management is crucial.

MI-4T: Data Structure **Credits 03**

Course contents:

Arrays **3 Hrs.**

Single and Multi-dimensional Arrays, Sparse Matrices (Array and Linked Representation)

Stacks **5 Hrs.**

Implementing single/multiple stacks in an Array; Prefix, Infix, and Postfix expressions, Utility and conversion of these expressions from one to another; Applications of a stack; Limitations of Array representation of a stack

Linked Lists **7 Hrs.**

Singly, Doubly, and Circular Lists (Array and Linked representation); Normal and Circular representation of Stack in Lists; Self Organizing Lists; Skip Lists

Queues **5Hrs.**

Array and Linked representation of Queue, De-queue, and Priority Queues

Recursion **5 Hrs.**

Developing Recursive Definition of Simple Problems and their implementation; Advantages and

Limitations of Recursion; Understanding what goes behind Recursion (Internal Stack Implementation)

Trees **10 Hrs.**

Introduction to Tree as a data structure; Binary Trees (Insertion, Deletion, Recursive and Iterative Traversals on Binary Search Trees); Threaded Binary Trees (Insertion, Deletion, Traversals); Height-Balanced Trees (Various operations on AVL Trees). Tree traversal techniques.

Searching and Sorting **7 Hrs.**

Linear Search, Binary Search, Comparison of Linear and Binary Search, Selection Sort, Insertion Sort, Bubble Sort, Quick Sort, Comparison of Sorting Techniques

MI-4P: Data Structures Lab **Credits 01**

Programming Assignments

1. **Searching in a List**
   Write a program to search for an element in a list. Provide the user with the option to perform either **Linear Search** or **Binary Search**. Use **template functions** to make the program generic.
2. **Sorting a List**
   Write a program using **templates** to sort a list of elements. Allow the user to choose between **Insertion Sort**, **Bubble Sort**, or **Selection Sort**.
3. **Linked List Implementation**
   Implement a **Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list

4. **Doubly Linked List Implementation**
   Implement a **Doubly Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list.

5. **Circular Linked List Implementation**
   Implement a **Circular Linked List** using templates. Include functions for:
   - Insertion
   - Deletion
   - Searching for a number
   - Reversing the list.

6. **Stack Operations Using Linked List**
   Implement **Stack** operations (push, pop, peek, etc.) using a **Linked List** implementation.
7. **Stack Operations Using Array**
   Implement **Stack** operations (push, pop, peek, etc.) using an **Array** implementation. Use **templates** to make the stack generic.
8. **Queue Operations Using Circular Array**
   Implement **Queue** operations (enqueue, dequeue, etc.) using a **Circular Array** implementation. Use **templates** to make the queue generic.
9. **Double-Ended Queue (Deque) Operations**
   Create and perform operations on a **Double-Ended Queue (Deque)** using a **Linked List** implementation.
10. **Polynomial Addition Using Linked List**
    Write a program to represent a polynomial using a **Linked List** and perform the addition of two polynomials.
11. **Factorial and Factors of a Number**
    Write a program to calculate the **factorial** and compute the **factors** of a given number:
    (i) Using **recursion**
    (ii) Using **iteration**.
12. **Fibonacci Series**
    Write a program to display the **Fibonacci series**:
    (i) Using **recursion**
    (ii) Using **iteration**.
13. **GCD of Two Numbers**
    Write a program to calculate the **GCD** of two numbers:
    (i) Using **recursion**
    (ii) Without recursion.
14. **Sparse Matrix Conversion**
    Write a program to convert a **Sparse Matrix** into its non-zero form and vice versa.
15. **Reverse Stack Using Additional Stack**
    Write a program to reverse the order of elements in a stack using an **additional stack**.
16. **Reverse Stack Using Additional Queue**
    Write a program to reverse the order of elements in a stack using an **additional queue**.
17. **Diagonal Matrix Implementation**
    Write a program to implement a **Diagonal Matrix** using a **one-dimensional array**.
18. **Lower Triangular Matrix Implementation**
    Write a program to implement a **Lower Triangular Matrix** using a **one-dimensional array**.
19. **Upper Triangular Matrix Implementation**
    Write a program to implement an **Upper Triangular Matrix** using a **one-dimensional array**.
20. **Symmetric Matrix Implementation**
    Write a program to implement a **Symmetric Matrix** using a **one-dimensional array**.

Suggested Readings:

1. Gilberg and Forouzan: “Data Structure- A Pseudo code approach with C” by Thomson publication
2. “Data structure in C” by Tanenbaum, PHI publication / Pearson publication.
3. Pai: ”Data Structures & Algorithms; Concepts, Techniques & Algorithms ”Tata McGraw Hill.
4. “Fundamentals of data structure in C” Horowitz, Sahani & Freed, Computer Science Press.
5. “Fundamental of Data Structure” (Schaums Series) Tata-McGraw-Hill.

SKILL ENHANCEMENT COURSE (SEC)

SEC 3: PYTHON **Credits 03 (Full Marks: 50)**

OBJECTIVE OF THE COURSE -

The objectives of this course are to make the student understand programming language, programming, concepts of Loops, reading a set of Data, stepwise refinement, Functions, Control structure, Arrays. After completion of this course the student is expected to analyze the real-life problem and write a program in 'Python' language to solve the problem. The main emphasis of the course will be on problem solving aspect i.e., developing proper algorithms.

After completion of the course the student will be able to

- Develop efficient algorithms for solving a problem.
- Use the various constructs of a programming language viz. conditional, iteration and recursion.
- Implement the algorithms in "Python" language.

OUTCOME OF THE COURSE -

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

- Describe the fundamental concepts of programming, including **variables**, **data types**, **loops**, and **control structures**.
- Understand the role of **functions** and **arrays** in organizing code and data.
- Analyze problems and design **stepwise algorithms** to solve them.
- Use **arrays** (lists) to store and manipulate data.
- Identify and fix errors in Python programs using debugging techniques.
- Test programs to ensure they meet the problem requirements.
- Apply Python programming skills to solve practical problems, such as:
  - Mathematical calculations
  - Data processing
  - Simple automation tasks.
- Demonstrate the ability to translate real-life scenarios into functional code.
- Explore and use Python libraries (e.g., **math**, **random**) to enhance program functionality.
- Work with Python development tools like **IDLE** or **Jupyter Notebook**.
- Build a strong foundation in programming to prepare for advanced topics in computer science and software development.
- Develop skills relevant to careers in **software development**, **data analysis**, and **automation**

SEC 3P: PYTHON **Credits 03**

Programming Assignments

1. **Problem Solving and Debugging**
   a. Write a Python program to solve a simple problem (e.g., finding the largest of three numbers).
   b. Identify and fix errors in a given program (debugging).

c. Document the program with comments and a brief description of the problem-solving approach.

2. **Flowchart and Algorithm Design**
    a. Design a flowchart and write an algorithm for a problem (e.g., calculating the factorial of a number).
    b. Convert the flowchart and algorithm into a Python program.
3. **Structured Programming**
    a. Write a Python program using **top-down** and **bottom-up** programming methodologies to solve a problem (e.g., calculating the sum of digits of a number).
4. **Python Program Structure**
    a. Write a Python program to demonstrate the basic structure of a Python program (e.g., print "Hello, World!").
5. **Python Operators**
    a. Write a Python program to demonstrate the use of:
        i. Arithmetic operators
        ii. Relational operators
        iii. Logical operators
        iv. Assignment operators
        v. Ternary operators
        vi. Bitwise operators.
6. **Input and Output Statements**
        i. Write a Python program to take user input and display output (e.g., calculate the area of a circle).
7. **Control Statements**
        i. Write a Python program to demonstrate:
        ii. Branching (if-else)
        iii. Looping (for, while)
        iv. Conditional statements (elif).
        v. Example: Check if a number is prime.
8. **Functions**
        i. Write a Python program to define and use functions with:
        ii. Default arguments
        iii. Return statements.
        iv. Example: Create a function to calculate the factorial of a number.
9. **Exception Handling**
        i. Write a Python program to handle exceptions (e.g., division by zero).
10. **Iterations**
        i. Write a Python program to demonstrate:
        ii. For loops
        iii. While loops.
        iv. Example: Print the Fibonacci series using a loop.
11. **Recursions**

i. Write a Python program to demonstrate recursion (e.g., calculate the factorial of a number using recursion).
ii. Draw a stack diagram for a recursive function.

12. **Multiple Assignments**
    i. Write a Python program to demonstrate multiple assignments (e.g., swap two numbers without a temporary variable).

13. **String Operations**
    i. Write a Python program to demonstrate:
    ii. String traversal using a for loop
    iii. String slicing
    iv. String comparison
    v. Finding a substring in a string.
    vi. Example: Count the occurrences of a character in a string.
14. **Factorial Calculation**
    Write a program to calculate the factorial of a number using both iteration and recursion.
15. **Fibonacci Series**
    Write a program to generate the Fibonacci series using both iteration and recursion.
16. **Prime Number Check**
    Write a program to check if a number is prime.
17. **String Palindrome**
    Write a program to check if a string is a palindrome.
18. **List Sorting**
    Write a program to sort a list of numbers without using built-in functions.
19. **Matrix Operations**
    Write a program to perform matrix addition and multiplication.

Suggested Readings:

1. Introduction to Computation and Programming Using Python by John V. Guttag, Publisher: MIT Press, Year of Publication: 2013 (1st Edition), 2016 (2nd Edition), 2021 (3rd Edition), Edition: 3rd Edition (Latest)
2. Think Python: How to Think Like a Computer Scientist by Allen Downey, Publisher: O'Reilly, Year of Publication: 2012 (1st Edition), 2015 (2nd Edition), Edition: 2nd Edition (Latest)
3. Learning Python, 5th Edition" by Mark Lutz, Publisher: O'Reilly, Year of Publication: 2013, Edition: 5th Edition (Latest)

INTERNSHIP/APPRENTICESHIP (INT)

Credit-04 Marks: 50 **(120 hours, 8 weeks)**

Guideline for internship/apprenticeship:

The internship program will commence at the beginning of the third semester and will be evaluated upon its completion at the end of the fourth semester.

1. A student may visit an industry for industry-related issues or a research institution, laboratory, or academic institute to engage in internship under the guidance of an industry official, scientist, or academician.
2. A student may work at a company's outlet or similar type of office, Professional bodies, etc. to develop programming / process etc. under the supervision of the respective official or a faculty of his/her own college teacher or a teacher from another college/university/industry person.
3. Interns may engage in advanced learning in topics beyond their course curriculum, under the guidance of their respective mentor.
4. Interns may be assigned a problem to solve using any programming language.
5. Interns may be assigned to design a webpage for college /department/ Entrepreneur/ start-up under the mentor's guidance.
6. Interns may be allowed to work as quantitative researchers in advance topics in Computer Science and applications from the reputed institute/ organization.

General instructions:

a) Each intern must maintain a daily logbook of activities.
b) At the end of the internship, a completion certificate must be obtained from the mentor, supervisor, or concerned authority.
c) Interns are expected to strictly adhere to the assigned tasks and deadlines.