Schedule of Practice

Practice Hours Per Week			Each Course: 4 + 2 = 6 Months
Theory	5 Days x 3 Hours (Evening) = 15 Hours	Average 20 Topics Each Days	4 Months / 4 Subjects
Lab Practice	5 Days x 3 Hours (Morning) = 15 Hours	Average 10 Topics Each Days	4 Months / C++ | Web | DevOps | UI-UX
Total Lesson Hours	5 Days x 6 Hours (Full Day) = 30 Hours		
Review and Test	1 Days x 6 Hours (Full Day) = 06 Hours	Re-Arrange All Topics	Basic Quiz, Interview, Real-life and Competitive Problem Solving
Industrial Project	1 Days x 6 Hours (Full Day) = 06 Hours	Project – Analysis | Clone | Develop	4 Months x 2 Projects = 8 Projects
Total Review Hours	2 Days x 6 Hours (Full Day) = 12 Hours		

Additional Resources:-

1. GPT
2. Websites and YouTube: Professional and Industrial Practice Example
3. Certification: MIT, Harvard, Oxford, Microsoft, Google, OpenAI, IBM, Etc

Practice (5 Days of Evening Shift):-

- **Using GPT** – (To Understand, Summarization Techniques) for Each Topics
- Understand Step by Step Each of the Concept by Applying on Lab
- **Summarize the Lessons in a Compacted Table with List**

Practice (5 Days of Morning Shift):-

- Review and Manipulate Every Day Lessons for – (Quiz, Interview, and Real-Live Problem)

Review (Off Days of Evening Shift):-

- Using GPT – Basic Quiz and Competitive Questions,
- Using GPT – Interview Question (Google, Microsoft)

Review (Off Days of Morning Shift):-

- Using GPT – Create Freelancing Project
- YouTube - Understanding Live Professional Client Project
- Build Community
- Market and Client Analysis

Year-01.1: Fundamental and Development

	Course-1: Fundamentals of Software Engineering		**Duration: (4 + 2) Months**		
01	Discrete Mathematics	T-H3	14	363	1118
02	Data Structure: Reference-1	T-H3 + S-H2	10	155	605
	Data Structure: Reference-2		17	178	734
	Algorithm: Reference-1		15	188	777
	Algorithm: Reference-2		08	156	1313
	Algorithm: Reference-3		12	93	480
03	Database Management System	T-H3 + S-H4	27	454 x 2	1373
04	Software Engineering: Reference-1	T-H3 + S-H4	15	241	712
	Software Engineering: Reference-2		33	510	930
	Theoretical Subjects				**8042**
01	Structured Programming	T-H3 + S-H4	21	184	564
02	Object-Oriented Concepts: Reference-1	T-H3 + S-H4	17	172	928
	Object-Oriented Concepts: Reference-2		15	165	347
	Lab Subjects				**1839**
05	Basic Mathematics: Reference-1	T-H3	19	117	322
	Basic Mathematics: Reference-2				353
	Elective Subjects				**675**

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental of Computer Science
- Fundamental of Software Engineering
- Fundamental of Language and Database
- Application of Data Structure and Algorithm

Year-01.2: Fundamental and Development

	Course-2: Development and Architecture		**Duration: (4 + 2) Months**		
01	Calculus and Analytical Geometry: Reference-1	T-H3	15	120	1281
	Calculus and Analytical Geometry: Reference-2		25	178	1283
02	Software Architecture and Design: Reference-1	T-H3 + S-H4	24	191	422
	Software Architecture and Design: Reference-2		12	69	357
	Software Architecture and Design: Reference-3		08	51	429
03	Unified Modeling Language (UML)	T-H3 + S-H2	07	124	288
04	Software Documentation	T-H3 + S-H2	11	149	241
	Theoretical Subjects				**4301**
01	Fundamental of Web Programming	T-H3 + S-H4		206	
	Front-End Development			540	
	Full-Stack Development			403	
	Fundamental Concepts: Reference-1		13	114	413
02	Database Design and Development: Reference-Web	T-H3 + S-H4		132	
	Fundamental Concepts: Reference-2		20	243	552
	Lab Subjects			**1455**	**2246**
05	Fundamental Statistics	T-H3	13	60	353
	Elective Subjects				**353**

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental of Web Technology
- Fundamental of Software Documentation
- Application of Web Design and Development
- Application of Database Design and Administration
- Application of Software Architecture and Design Pattern

Year-02.1: DevOps and Design

	Course-3: DevOps and Requirements		**Duration: (4 + 2) Months**		
01	Probability and Statistics for Engineers: Reference-1	T-H3	16	146	692
	Probability and Statistics for Engineers: Reference-2		15	189	710
02	Software Testing: Reference-1	T-H3 + S-H2	19	112	629
	Software Testing: Reference-2		21	285	573
03	Software Metrics: Reference-1	T-H3 + S-H2	15	390	651
	Software Metrics: Reference-2		14	169	276
04	Software Requirements	T-H3 + S-H2	11	126	399
	Theoretical Subjects				**3930**
01	DevOps Engineering: Reference-1	T-H3 + S-H4	28	165	435
02	DevOps Engineering: Reference-2		15	241	489
	DevOps Engineering: Reference-3		09	96	148
	Lab Subjects				**1072**
05	Software Maintenance	T-H3 + S-H2	16	160	420
06	Cloud Computing and Virtualization	T-H3 + S-H2	16	257	558
	Elective Subjects			**257**	**978**

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental Software Development Life-Cycle
- Fundamental of Development Process
- Application of DevOps Engineering
- Application of Software Testing
- Application of Software Maintenance

Year-02.2: DevOps and Design

	Course-4: System Design and Project Management		**Duration: (4 + 2) Months**		
01	Ordinary Differential Equation : Reference-1	T-H3	07	61	275
	Ordinary Differential Equation : Reference-1		12	177	819
02	System Analysis and Design: Reference-1	T-H3 + S-H4	19	364	529
	System Analysis and Design: Reference-1		14	269	677
04	Software Project Management: Reference-1	T-H3 + S-H4	12	290	718
	Software Project Management: Reference-2		23	364	498
02	Human-Computer Interaction	T-H3 + S-H2	21	516	861
	Theoretical Subjects				**4377**
01	Design Fundamental and Process	T-H2 + S-H6			
	UI Design				
02	UX Design				
	Lab Subjects				
05	Data Visualization	T-H3 + S-H4	29	87	389
06	Software Quality Assurance	T-H3 + S-H2	26	159	617
	Elective Subjects				**1006**

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental of Graphics Design
- Fundamental of UX
- Fundamental of Interaction
- Application of Design Tools and Techniques
- Application of UI Design and Data Visualization
- Application of System Analysis and Design
- Application of Project Management

Year-03.1: Artificial Intelligence and System

	Course-5: Intelligent System and Data Science		**Duration: (4 + 2) Months**		
01	Combinatorial Optimization	T-H3	21	125	545
02	Artificial Intelligence	T-H3 + S-H4	29	585	1151
03	Machine Learning: Reference 01	T-H3 + S-H4	13	140	379
	Machine Learning: Reference 02		14	226	510
	Deep Learning and Neural Network	T-H3 + S-H2	20	176	801
04	Data Science: Reference 01	T-H3 + S-H4	09	120	322
	Data Science: Reference 02		25	191	330
	Theoretical Subjects				**4038**
01	Python	T-H3 + S-H4	22	398	1155
02	Framework				
	Lab Subjects			**223**	**1155**
05	Data Mining	T-H3 + S-H2	13	245	740
06	Data Warehouse	T-H3 + S-H2	18	257	576
	Elective Subjects				**1316**

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental of Artificial Intelligence
- Fundamental of Data Science
- Application of Machine Learning
- Application of Deep Learning
- Application of Data Mining
- Application of Warehousing

Year-03.2: Artificial Intelligence and System

	Course-6: System and Communication		**Duration: (4 + 2) Months**		
01	Graph Theory	T-H3			
02	Theory of Computation	T-H3		116	
03	Operating System: Reference-1	T-H3 + S-H2		172	
	Operating System: Reference-2			388	
04	Data Communications and Networking	T-H3 + S-H2		549	
	Theoretical Subjects				
01	System Programming	T-H3 + S-H4			
02	Network Programming				
	Lab Subjects				
05	Distributed System	T-H3 + S-H2		84	
06	Parallel Computing	T-H3 + S-H2		255	
	Elective Subjects				

Course Outcomes:-

- Understanding all the Necessary Concept with Its –
  - Real-Life Professional Application and Problems
  - Comparison with Related Technology
  - Range, Domains and Classification
  - Invention and Evolution
- Fundamental of Operating System
- Fundamental of Data Communication
- Fundamental of Computational Theory
- Application of System Programming
- Application of Distributed System
- Application of Parallel Programming

Year-04.1: Business and Digital System

	Course-7: Industrial Training and Business		**Duration: (4 + 2) Months**		
01	Numerical Analysis of Engineers	T-H3			
02	Business Philosophy	T-H3			
03	Business Studies for Engineers	T-H3			
04	Business Communication	T-H3			
	Theoretical Subjects				
01	Industrial Training and Freelancing	S-H4			
02	Market Analysis and Building Community				
	Lab Subjects				
05	Ethics of Information System	T-H3		400	
06	Software Security	T-H3 + S-H4		319	
	Elective Subjects				

Year-04.2: Business and Digital System

	Course-8: Digital System and Interfacing		**Duration: (4 + 2) Months**		
01	Digital Electronics and Logic Design	T-H3 + S-H4			
02	Computer Organization	T-H3			
03	Embedded System	T-H3 + S-H2			
04	Computer Network and Wireless Communication	T-H3 + S-H2			
	Theoretical Subjects				
01	Assembly Language and Interfacing	T-H3 + S-H4			
02	Strategic Management	T-H3			
	Lab Subjects				
05	Computer Graphics and Multimedia	T-H3 + S-H4			
06	Pattern Recognizing and Image Processing	T-H3 + S-H4			
	Elective Subjects				

//

SL/NO	Part One	Engineering and Development (DSA & DB)
01	Structure Programming	Core Programming (C++ and JavaScript)
02	Data Structure	Core Programming (C++ and JavaScript)
03	Algorithm	Core Programming (C++ and JavaScript)
04	Database Management System	Database Design (MySQL and MongoDB)
05	Object-Oriented Programming	Core Programming (C++ and JavaScript)
	Part Two	**Engineering and Development (Architect)**
06	Software Engineering	
07	Software Design and Analysis	Micro-service, Scalability, Design Patterns – Data Intensive App
08	System Analysis and Design	
09	Software Security	
10	Professional Ethics for Information System	
	Part Three	**Engineering and Development (Web & AI)**
11	Web Technology and Frameworks	Web Development Basic (HTML, CSS, JavaScript)
	Backend Development	Node.JS and Express.JS
	Frontend Development	React.JS, State Management and Responsive Design
	Full Stack Development	API, Authentication (JWT|QAuth) and Advanced JavaScript
	Advanced Full Stack	Real-Time Apps (Web-socket) and Server-less Architecture
12	Artificial Intelligence and Machine Learning	
13	Applied Data Science and Engineering	
	Part Four	**Product Management**
14	Requirement Specification and Analysis	
15	Software Metrics	
16	Testing and Quality Assurance	Manual Testing and Testing Automation Tool (Selenium)
	QA Automation	Test Framework (Cypress|Appium), Perofrmance Testing
17	Project Management	Agile Methodology, Scrum, Stakeholder Management
	Part Five	**Product Management (DevOps Engineering)**
18	Software Maintenance	
19	Virtualization and Cloud Computing	
20	DevOps Fundamental	Linux Command Line, Version Control, CI/CD Fundamentals
	DevOps Advance	Docker, Kubernetes, Infrastructure as Code
21	Development Process	
	Part Six	**Design and User Experience**
22	Human-Computer Interaction	
23	UI/UX Design Fundamental	Design Principle, Figma and Prototyping
	UI/UX Design Advanced	Advance Prototyping, Usability Testing, Motion Design
24	Technical Writing and UML	

SL/NO	Part Six	Computer science (Mathematics)
24	Discrete Mathematics	
25	Numerical Analysis	
26	Probability and Statistics	
27	Calculus, Deferential Equation and Analytical Geometry	
28	Combinational Optimization	
	Part Seven	**Computer Science Part-1**
29	Theory of Computation	
30	Operating System and System Programming	
31	Computer Network	
32	Distributed System and Parallel Computing	
	Optional Group One	**Computer Science Part-2**
33	Computer Organization	
34	Computer Graphics and Multimedia	
35	Mobile and Wireless Computing	
36	Embedded System	
37	Pattern Recognizing and Image Processing	
	Optional Group Two	**Business Computing and Customer Support**
38	Numerical Computation for Financial Modeling	
39	Information Retrieval	
40	Enterprise Information System	
41	Data Mining and Warehouse	
42	Business Psychology	
43	Business Studies for Engineers	
44	Business Communication	CRM, Communication Strategies, Handling User Feedback
45	Strategic Management	

Here's a Bachelor of Science (BSc) degree-style curriculum table structured to cover Full Stack Development, Design, QA, DevOps, Project Management, Software Architecture, and Customer Support as Specialized areas. This schedule spans 8 semesters (4 years) and includes core courses, electives, projects, and industry-ready skills for expertise.

Semester	Subject Area	Topics	Learning Resources
01	Core Programming	Programming Fundamentals (Python, JavaScript), Algorithms, and Data Structures	- Introduction to the Theory of Computation by Michael Sipser - CS50's Introduction to Computer Science (Harvard)
02	Web Development Basics	HTML, CSS, JavaScript Basics	- HTML and CSS: Design and Build Websites by Jon Duckett - FreeCodeCamp Web Dev Guide
03	Backend Development	Node.JS, Express.JS, Database (SQL, MongoDB)	- Eloquent JavaScript by Marijn Haverbeke - MDN Backend Docs
04	UI/UX Design Basics	Design Principles, Figma, Prototyping	- The Elements of User Experience by Jesse James Garrett - Interaction Design Foundation
05	Frontend Development	React/Angular, State Management, Responsive Design	- Learning React by Kirupa Chinnathambi - Frontend Mastery by Codecademy
06	QA Testing Basics	Manual Testing, Introduction to Automation Testing Tools (Selenium)	- Testing Computer Software by Cem Kaner - Test Automation University
07	Full Stack Development	APIs, Authentication (JWT, OAuth), Advanced JavaScript	- The Odin Project Full Stack Path
08	UI/UX Advanced	Advanced Prototyping, Usability Testing, Motion Design	- Don't Make Me Think by Steve Krug - Design + Code Tutorials
09	Software Architecture	Micro-services, Design Patterns, Scalability	- Designing Data-Intensive Applications by Martin Kleppmann
10	Advanced Full Stack	Real-Time App (WebSocket), Server-less Architecture	- Node.JS in Action
11	QA Automation	Test Frameworks (Cypress, Appium), Performance Testing	- Continuous Testing for DevOps Professionals by Katrina Clokie
12	DevOps Basics	Linux Command Line, Git, CI/CD Fundamentals	- DevOps Full Course by Simplilearn
13	Project Management	Agile Methodology, Scrum, Stakeholder Management	- Scrum: The Art of Doing Twice the Work in Half the Time Agile M.
14	DevOps Advanced	Docker, Kubernetes, Infrastructure as Code	- The Phoenix Project by Gene Kim - Docker Documentation
15	Customer Support	CRM, Communication Strategies, Handling User Feedback	- Zendesk Customer Support Guide
16	Capstone Project	Build a Full-Scale Application Incorporating All Sills	- Mentorship Programs (linkedIn Learnig) - Personal GitHub Projects

Software Development

1. Self-Taught (Part-Time)

If you're learning in your spare time (e.g., evenings or weekends):

- **Time Frame:** 6-12 months
- **Details:** You'll need to go through the basics of HTML, CSS, JavaScript, and then move on to more advanced topics such as frameworks (React, Angular, Vue), back-end development (Node.js, databases), and version control (Git).
    - Expect to build small projects in the first few months.
    - The last few months will involve more complex projects and learning real-world tools like APIs, deployment, and debugging.
    - A portfolio of personal or freelance projects is key to landing a job.

2. Bootcamp (Full-Time)

If you enroll in an intensive coding bootcamp:

- **Time Frame:** 3-6 months
- **Details:** Coding bootcamps are designed to get you job-ready in a short time. The curriculum is intense, covering full-stack development (front-end and back-end) in a structured format.
    - You'll learn HTML, CSS, JavaScript, frameworks (React, Angular), server-side technologies (Node.js, databases), and Git.
    - Bootcamps often include job search support and real-world projects.

3. Traditional Computer Science Degree (Part-Time or Full-Time)

If you pursue a degree in Computer Science (CS):

- **Time Frame:** 3-4 years (for a bachelor's degree)
- **Details:** This is the most comprehensive path, covering both web development and broader computer science concepts such as algorithms, data structures, and system design.
    - While it will take longer, a CS degree provides deeper knowledge, which can help you in technical interviews and advanced development roles.
    - This path may also open opportunities in software engineering, system architecture, and more.

4. Freelancing or Internships

If you learn while doing practical work:

- **Time Frame:** 6-12 months (depends on how quickly you can build a portfolio)
- **Details:** Gaining experience by working as a freelancer or intern can accelerate your learning process.
    - You will likely need to start with small projects and gradually build a portfolio that demonstrates your skills.
    - Real-world experience will help you quickly understand what works and what doesn't in web development.

Key Milestones to Becoming a Professional:

1. **Mastering Core Skills:**
    - HTML, CSS, JavaScript
    - Version control (Git)
    - Frameworks (React, Vue, or Angular for front-end)
    - Back-end development (Node.js, Express, databases)
    - Deployment (Netlify, Heroku, AWS)
2. **Building a Portfolio:**
    - Have a portfolio website showcasing your projects.
    - Include both personal projects and any freelance or internship work.
3. **Gaining Practical Experience:**
    - Contribute to open-source projects.
    - Work on freelance projects or internships to apply what you've learned.
    - Continuously update your portfolio with new projects.
4. **Staying Up-to-Date:**
    - Web development is an ever-evolving field, so continuous learning is necessary. Stay updated with new technologies, best practices, and industry trends.

Conclusion:

- **If full-time:** You could become a professional web developer in about 3-6 months with intensive learning (via bootcamp or self-study).
- **If part-time:** Expect 6-12 months, depending on your dedication.
- **For a deeper understanding (CS degree):** It would take about 3-4 years.

Ultimately, the timeline can vary depending on your learning speed and how much time you can commit to studying. The key is consistent practice, building real-world projects, and continuously improving your skills.

Web Development:

Here is a detailed table for the part-time self-taught web development course, including a suggested practice schedule over the course of 6-12 months:

Month	Topics to Learn	Key Activities/Practice	Milestones
1	**HTML & CSS Basics**	- Learn basic HTML (structure, elements, forms) - Learn CSS (selectors, styles, layouts) - Introduction to responsive design (media queries)	- Build a basic static webpage - Create a personal portfolio webpage
2	**HTML5, CSS3, Responsive Design**	- Learn advanced HTML5 elements (audio, video, canvas) - Dive deeper into CSS3 (animations, transitions) - Mobile-first design and Flexbox	- Enhance personal portfolio with responsiveness - Build a simple static multi-page website
3	**JavaScript Basics**	- Learn JavaScript syntax, variables, operators - Functions, loops, and conditionals - Introduction to the DOM (Document Object Model)	- Add interactivity to previous projects (form validation, simple animations)
4	**JavaScript Deep Dive**	- Learn objects, arrays, and data structures - Asynchronous JavaScript (callbacks, promises, async/await) - Introduction to basic JavaScript libraries (e.g., Lodash)	- Build a simple interactive app (e.g., to-do list) - Complete JavaScript practice exercises on platforms like FreeCodeCamp or Codecademy
5	**Advanced JavaScript (ES6+), Web APIs**	- Learn ES6+ features (arrow functions, destructuring, modules) - Learn to use Web APIs (local storage, fetch API)	- Refactor existing projects with ES6+ features - Build a project that interacts with a Web API (e.g., weather app using OpenWeather API)
6	**Introduction to Front-End Frameworks**	- Learn the basics of React (components, props, state) - Learn about the component lifecycle	- Build a basic React app (e.g., a blog or weather app)
7	**Advanced Front-End Frameworks (React/Angular/Vue)**	- Dive deeper into React (hooks, context, routing) - Learn about React libraries (React Router, Axios)	- Build a multi-page React app with routing and API interaction
8	**Back-End Development (Node.js & Express)**	- Learn Node.js basics (server setup, HTTP requests, routing) - Introduction to Express (middleware, routing)	- Build a simple back-end with Node.js and Express - Create a REST API (e.g., user authentication system)
9	**Databases & Server Deployment**	- Learn about databases (SQL, MongoDB) - Learn to connect back-end with databases - Introduction to server deployment (Heroku, Netlify)	- Build a full-stack app with a database (e.g., task manager with user authentication) - Deploy your app to a platform like Heroku or Netlify
10	**Version Control (Git & GitHub)**	- Learn Git basics (commits, branches, merges, clone, push) - Use GitHub for hosting code and collaboration	- Upload all projects to GitHub - Collaborate on a small open-source project or contribute to a repo

Month	Topics to Learn	Key Activities/Practice	Milestones
11	**Deployment, Debugging, and Optimization**	- Learn about deployment tools and CI/CD (continuous integration/continuous deployment) - Debugging techniques and performance optimization	- Debug and optimize your full-stack app - Deploy a project on a live server
12	**Capstone Project**	- Develop a complex full-stack web application (combine front-end and back-end skills) - Incorporate features such as user authentication, API integration, and deployment	- Complete and deploy a capstone project - Build and update portfolio with all projects

Ongoing Activities (Throughout the Course):

- **Build Portfolio**: Continuously add projects to your portfolio as you learn new skills.
- **Practice Consistently**: Dedicate around 10-15 hours per week to learning and coding, adjusting the time as you get more comfortable with the concepts.
- **Contribute to Open-Source**: Contribute to small open-source projects on GitHub to gain practical experience and collaborate with others.
- **Stay Up-to-Date**: Follow web development blogs, attend meetups, or watch tutorials on new tools and technologies.

This schedule is flexible and can be adjusted based on your personal pace and availability, with a focus on building projects and gaining practical experience throughout the learning process.

At BUET (Bangladesh University of Engineering and Technology), the number of classes required to complete a course generally depends on the specific course and its credit hours.

For a typical **"Structured Programming"** course in the Computer Science and Engineering (CSE) department, it is likely to be a **3-credit hour** course. The structure of such a course could typically be:

- **Lecture Classes:** These are usually held 2-3 times a week, with each class lasting around 1.5 hours.
- **Total Duration:** For a semester, there are usually about **14-15 weeks** of classes, which would mean roughly **28-45 classes** for the entire course.

In addition to lectures, there might also be **laboratory sessions**, **tutorials**, and **assignments**, which would add to the total workload.

Thus, for a typical **structured programming course**, you can expect around **30 to 45 classes** over the course of a semester.

In a typical "Structured Programming" course at BUET (Bangladesh University of Engineering and Technology), the content and structure of the laboratory sessions, tutorials, and assignments will vary, but they generally focus on reinforcing the theoretical concepts taught in lectures and providing hands-on experience. Here's a breakdown:

1. Laboratory Sessions

Content & Purpose:

- **Objective**: Laboratory sessions allow students to practice coding and problem-solving based on the concepts learned in the lectures.
- **Programming Tasks**: Students will write, debug, and test programs that implement structured programming techniques (e.g., loops, conditionals, functions, arrays, etc.).
- **Concepts Covered**: These may include topics such as:
    - Basic data types, variables, operators
    - Functions and procedures
    - Control structures (if-else, switch, loops)
    - Arrays, strings, and pointers
    - File handling
- **Tools**: Typically, students use an Integrated Development Environment (IDE) like Code::Blocks, Visual Studio, or any other relevant programming tools provided by the department.

Duration:

- Each laboratory session typically lasts around **2-3 hours**.
- Lab sessions might be conducted once a week (or sometimes twice), depending on the course schedule.

What to Do:

- Attend the session with a specific task or assignment related to the topic of the week.
- Work individually or in groups to complete the tasks, write code, and solve the problems.
- Demonstrate your working program to the lab instructor for feedback and debugging assistance.

2. Tutorials

Content & Purpose:

- **Objective**: Tutorials are designed to clarify doubts, explain concepts in more detail, and provide additional practice problems.
- **Topics Covered**: Tutors may revisit or expand on lecture topics, such as:
  - Problem-solving strategies
  - Algorithms and data structures (like sorting, searching)
  - Code optimization and debugging
  - Software design principles
- **Interactive Learning**: During tutorials, students often work on problems in small groups or individually, and the tutor offers guidance on how to approach different problems.

Duration:

- Typically, tutorials last **1-2 hours** per session.
- Like lab sessions, tutorials are usually held once a week.

What to Do:

- Engage in solving theoretical and practical problems posed by the tutor.
- Work on algorithm design and logical thinking.
- Participate in discussions, ask questions, and collaborate with peers.

3. Assignments

Content & Purpose:

- **Objective**: Assignments are designed to test students' understanding of programming concepts and their ability to apply them to solve more complex problems. These can include both theoretical questions and practical coding tasks.
- **Topics Covered**:
  - Algorithm development and analysis
  - Code optimization techniques
  - Real-world problem-solving scenarios requiring structured programming approaches.
- **Complexity**: Assignments could range from writing short programs to implementing larger projects that require multiple steps, like designing a program from scratch to solve a specific problem.

Duration:

- Assignments are usually assigned **weekly or bi-weekly**.
- They may require 5-10 hours of work each, depending on the difficulty.

What to Do:

- Complete coding tasks and submit them by the given deadline.
- Often, assignments require you to write clean, efficient, and well-documented code.
- Some assignments may involve both writing code and providing a written explanation of the approach used, challenges faced, and how the solution works.

In Summary:

- **Laboratory Sessions**: Hands-on coding (2-3 hours per week).
- **Tutorials**: Problem-solving, concept clarification, coding practice (1-2 hours per week).
- **Assignments**: Independent work on programming tasks (5-10 hours per assignment, usually due weekly or bi-weekly).

These activities, when combined with lectures, ensure that students not only understand theoretical concepts but also gain practical experience in solving real-world programming problems using structured programming techniques.

For a typical "Structured Programming" course in the Computer Science and Engineering (CSE) department at BUET, considering the details you've provided, the total number of classes and hours would be calculated as follows:

1. Lecture Classes

- **Frequency**: 2-3 times a week.
- **Duration of each class**: 1.5 hours.
- **Weeks per semester**: 14-15 weeks.

Total Lecture Classes:

- If 2 classes per week:
  2 classes/week × 14 weeks = 28 classes
  2 classes/week × 15 weeks = 30 classes
- If 3 classes per week:
  3 classes/week × 14 weeks = 42 classes
  3 classes/week × 15 weeks = 45 classes

Total Lecture Hours:

- For 1.5 hours per class:
  1.5 hours × 28-45 classes = 42-67.5 hours.

2. Laboratory Sessions

- Typically held once a week, with each session lasting around 2-3 hours.
- **Frequency**: 1 lab session per week.
- **Duration**: 2-3 hours per session.
- **Weeks per semester**: 14-15 weeks.

Total Laboratory Sessions:

- 1 lab session per week for 14-15 weeks = 14-15 sessions.

Total Laboratory Hours:

- For 2-3 hours per lab session:
  2 hours × 14-15 sessions = 28-30 hours
  3 hours × 14-15 sessions = 42-45 hours.

3. Tutorials and Assignments

- Tutorials may occur 1-2 times a week, with 1 hour per session.
- Assignments are usually to be completed independently and will require additional time outside of class hours.
- **Tutorial Frequency**: 1-2 times per week.
- **Duration per session**: 1 hour.
- **Weeks per semester**: 14-15 weeks.

Total Tutorial Sessions:

- 1-2 tutorial sessions per week for 14-15 weeks = 14-30 sessions.

Total Tutorial Hours:

- 1 hour × 14-30 sessions = 14-30 hours.

4. Assignments (Outside of Class Hours)

- Typically, assignments will require students to spend additional hours beyond regular class and lab time, often 2-3 hours per assignment, depending on the complexity.
- Assume 3-4 assignments in the course.

Total Assignment Hours:

- 3 assignments × 3 hours each = 9 hours
- 4 assignments × 3 hours each = 12 hours.

Summary of Total Classes and Hours

1. Lecture Classes:

- Total Classes: 28-45
- Total Hours: 42-67.5 hours

2. Laboratory Sessions:

- Total Sessions: 14-15
- Total Hours: 28-45 hours

3. Tutorials:

- Total Sessions: 14-30
- Total Hours: 14-30 hours

4. Assignments (Outside Class):

- Total Hours: 9-12 hours (depends on the number and complexity of assignments)

Overall Total Hours for the Course:

- Total Hours (Lecture + Lab + Tutorial + Assignments):
  - **Minimum**: 42 (Lecture) + 28 (Lab) + 14 (Tutorial) + 9 (Assignments) = 93 hours
  - **Maximum**: 67.5 (Lecture) + 45 (Lab) + 30 (Tutorial) + 12 (Assignments) = 154.5 hours

Therefore, the total workload for the "Structured Programming" course at BUET could range from **93 to 154.5 hours** over the course of the semester, depending on the specific setup and number of sessions.

Here’s the updated table of courses:

Course	Lecture Classes	Lecture Hours	Laboratory Sessions	Laboratory Hours	Tutorial Sessions	Tutorial Hours	Assignments Hours	Total Hours (Min)	Total Hours (Max)
Structured Programming	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5
Data Structures	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5
Algorithms	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5
DBMS	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5
Software Engineering	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5
Object-Oriented Programming	28-45	42-67.5	14-15	28-45	14-30	14-30	9-12	93	154.5

Explanation:

- **Lecture Classes**: Number of classes in total (2-3 times per week for 14-15 weeks).
- **Lecture Hours**: Total hours for lecture classes, each class lasting 1.5 hours.
- **Laboratory Sessions**: Number of lab sessions in total (1 per week for 14-15 weeks).
- **Laboratory Hours**: Total hours for laboratory sessions, each lasting 2-3 hours.
- **Tutorial Sessions**: Number of tutorial sessions in total (1-2 times per week for 14-15 weeks).
- **Tutorial Hours**: Total hours for tutorial sessions, each lasting 1 hour.
- **Assignments Hours**: Total hours for assignments outside class (3-4 assignments with 3 hours each).
- **Total Hours (Min)**: Minimum total hours considering the lower bound of each range.
- **Total Hours (Max)**: Maximum total hours considering the upper bound of each range.

This table now includes **Structured Programming** and **Object-Oriented Programming**, alongside the other courses. The total workload for each course still ranges from **93 to 154.5 hours**.

Sure! Here’s a table summarizing the likely contents of the first class in a "Structured Programming" course for Computer Science and Engineering (CSE) at BUET:

Topic	Details
Introduction to Structured Programming	Overview of structured programming concepts, importance of structured programming in software development.
What is Programming?	Defining programming, purpose of programming languages, and understanding how software is created to solve problems.
History of Programming	A brief history of programming languages, evolution of structured programming from earlier paradigms like unstructured programming.
Key Concepts of Structured Programming	Explanation of basic structured programming concepts: sequence, selection (if-else), iteration (loops), and the importance of these structures for readability and maintainability.
Programming Paradigms	Overview of different programming paradigms (procedural, object-oriented, etc.), with a focus on structured programming.
Syntax and Semantics of C	Introduction to the C programming language, including the syntax and semantics of basic constructs (variables, operators, data types).
Importance of Algorithms	The role of algorithms in structured programming, basic principles of designing algorithms for solving problems.
Introduction to Development Environment	Overview of the tools and environment needed for programming, such as IDEs, compilers, and basic debugging techniques.
Example of Structured Program	Walkthrough of a simple structured program written in C (such as a Hello World program or basic input/output operations).
Homework/Assignments	Assignment on writing a simple program to practice concepts like input/output, variables, and basic control structures.

This structure might vary depending on the instructor, but these topics typically form the foundation for a structured programming course at BUET.

Certainly! Here’s a table summarizing the likely contents of the **second class** in a "Structured Programming" course for CSE at BUET:

Topic	Details
Review of Previous Class	Quick recap of the first class, including the basics of structured programming and an overview of C programming.
Introduction to C Programming	Detailed introduction to the C language, including its origin, importance, and basic syntax.
Basic Structure of a C Program	Explanation of the components of a C program: `#include` statements, `main()` function, curly braces `{}`, and `return` statement.
Variables and Data Types	Introduction to different data types in C (int, float, char, etc.), how to declare variables, and the importance of type safety.
Constants in C	Explanation of constants in C, how to declare and use them using `#define` and `const`.
Input and Output in C	Introduction to basic input/output functions in C such as `printf()`, `scanf()`, and the use of format specifiers (e.g., `%d`, `%f`, `%s`).
Operators in C	Introduction to different operators in C: arithmetic, relational, logical, and assignment operators.
Basic Program Example	Walkthrough of a simple program that includes input/output, variable declaration, and operators (e.g., a program to add two numbers).
Understanding Program Flow	Discussion on how the program flows from top to bottom, the role of the `main()` function, and how control is passed between statements.
Homework/Assignments	Assignment involving writing simple programs using basic variables, operators, and I/O functions.

The second class would typically provide foundational understanding and hands-on experience with writing and running basic C programs, which would be essential for the rest of the course.

Here’s a table summarizing the likely contents of the **third class** in a "Structured Programming" course for CSE at BUET:

Topic	Details
Review of Previous Class	Quick recap of the previous class, focusing on C syntax, variables, data types, input/output, and operators.
Control Structures – Introduction	Introduction to control structures in C, including conditional statements and loops.
Conditional Statements (if, if-else, nested if)	Explanation of `if` and `if-else` statements for decision-making in programs, with examples demonstrating their use in decision-making logic.
Relational Operators	Introduction to relational operators (e.g., `==`, `!=`, `<`, `>`, `<=`, `>=`) and their use in conditional statements.
Logical Operators	Explanation of logical operators (e.g., `&&`, `
Switch-Case Statement	Introduction to the `switch` statement, its syntax, and how it can be used as an alternative to multiple `if-else` statements for decision-making.
Examples of Control Flow	Demonstration of simple programs using `if`, `if-else`, and `switch-case` to control program flow based on conditions (e.g., checking for even/odd numbers).
Introduction to Loops (for, while, do-while)	Explanation of different types of loops in C: `for`, `while`, and `do-while`.
Looping Techniques	How and when to use different loops, differences between `for`, `while`, and `do-while` loops, and example programs that use loops for repetitive tasks.
Nested Loops	Introduction to nested loops, where one loop is inside another, and example use cases such as printing patterns or processing 2D arrays.
Homework/Assignments	Assignment on writing programs involving control structures and loops, such as finding prime numbers, calculating factorials, or creating number patterns.

The third class would build upon basic programming skills by introducing decision-making and repetition (loops) structures, which are essential for implementing more complex logic in structured programs.

Here’s a table summarizing the likely contents of the **fourth class** in a "Structured Programming" course for CSE at BUET:

Topic	Details
Review of Previous Class	Quick recap of the previous class, covering control structures (`if`, `switch`, loops) and their application in problem-solving.
Functions in C - Introduction	Introduction to functions in C: their purpose, definition, and how they help modularize code for better readability and reusability.
Defining Functions	Detailed explanation of how to define functions: function prototype, function definition, and function call.
Function Parameters and Return Values	Explanation of how to pass arguments to functions (parameters) and how to return values from functions using `return`.
Types of Functions	Introduction to different types of functions: *void functions* (functions that don't return a value) and functions with return types (e.g., `int`, `float`).
Function Scope and Lifetime	Discussion on scope (local vs global variables) and lifetime of variables, explaining how variables declared inside a function are not accessible outside it.
Recursion	Introduction to recursion: concept of a function calling itself, how to implement a recursive function, and when recursion is useful.
Examples of Functions and Recursion	Examples demonstrating functions, including recursive functions like calculating factorial or Fibonacci numbers.
Function Overloading	(Optional) Brief introduction to the concept of function overloading, if applicable (though C doesn't support function overloading directly).
Function Declarations and Prototypes	Detailed explanation of function declarations and prototypes, and why they are important for the compiler to recognize functions before they are used.
Homework/Assignments	Assignment involving writing functions: creating simple programs that use functions for tasks like sum of numbers, factorials, or recursive functions like Fibonacci.

The fourth class would focus on teaching functions, which are critical in structured programming for organizing and reusing code. Recursion would also be introduced as a key concept for solving problems that can be broken into smaller subproblems.

Here’s a table summarizing the likely contents of the **fifth class** in a "Structured Programming" course for CSE at BUET:

Topic	Details
Review of Previous Class	Quick recap of the previous class, covering functions, recursion, scope, and function prototypes.
Arrays in C - Introduction	Introduction to arrays in C: definition, purpose, and why they are useful for storing multiple values of the same type.
Declaring and Initializing Arrays	Explanation of how to declare and initialize arrays in C, including one-dimensional arrays (e.g., `int arr[5];`) and initializing at the time of declaration.
Accessing Array Elements	How to access and manipulate elements in an array using indices, with examples demonstrating array indexing.
Multidimensional Arrays	Introduction to multidimensional arrays (e.g., 2D arrays). Explanation of how to declare, initialize, and access elements in a 2D array (e.g., `arr[2][3]`).
Array Traversal	Techniques for traversing arrays using loops (e.g., `for` loop) to access and manipulate each element in the array.
Array Operations	Performing common operations on arrays, such as finding the maximum/minimum value, calculating the sum of elements, and sorting an array.
Passing Arrays to Functions	Explanation of how arrays are passed to functions in C: passing by reference (array name is a pointer to the first element). Examples of modifying array elements in functions.
Array and Pointer Relationship	Introduction to the relationship between arrays and pointers in C, explaining how arrays can be treated as pointers and the consequences of this.
Examples of Array Programs	Walkthrough of sample programs that use arrays, such as calculating the average of numbers, searching for an element, and sorting an array.
Homework/Assignments	Assignment on writing programs involving arrays, such as matrix multiplication, finding the largest element in an array, or reversing an array.

The fifth class would introduce arrays, a crucial data structure in programming, and help students learn how to handle collections of data. Arrays will be linked with concepts like pointers, which are important for understanding memory management in C.