DegreeLive

BCA Honours | Semester III | Calicut University

Data Structures using C

Course Code: BCA3CJ201 • Module 2 Notes

MODULE II: STACKS AND QUEUES

1. Stack and Queue Abstract Data Types

A **Stack** operates on the Last-In, First-Out (LIFO) protocol where insertions and deletions occur exclusively at the `top`. A **Queue** operates on the First-In, First-Out (FIFO) protocol where insertions occur at the `rear` and deletions occur at the `front`.

2. Illustrative Computational Working: Step-by-Step Numerical Example

Worked Example: Infix to Postfix Conversion & Postfix Evaluation

Problem Statement:

1. Convert the following infix expression into its equivalent **Postfix Expression** using the standard operator stack algorithm:

$$E = (A + B) * C - (D - E)/F$$

2. Evaluate the numerical Postfix expression using an operand stack:

$$P = 16, 4, /, 3, 5, *, +, 7, -$$

Solution & Step-by-Step Working:

Part 1: Infix to Postfix Conversion Trace

Operator Precedence: Parentheses () [highest], $*$, / [high, left-associative], $+$, $-$ [low, left-associative].

Symbol Scanned	Operator Stack	Postfix Output String
(	(	
A	(	A
+	(+	A
B	(+	A B
)	[empty]	A B +
*	*	A B +
C	*	A B + C
-	- (pops *)	A B + C *
(	- (	A B + C *
D	- (	A B + C * D
-	- (-	A B + C * D
E	- (-	A B + C * D E

DegreeLive

Symbol Scanned	Operator Stack	Postfix Output String
)	- (pops -)	`A B + C * D E -`
/	- /	`A B + C * D E -`
F	- /	`A B + C * D E - F`
End of String	[empty] (pops /, -)	`A B + C * D E - F / -`

Postfix Expression Result: `A B + C * D E - F / -`.

Part 2: Evaluation of Postfix Expression: $16, 4, /, 3, 5, *, +, 7, -$

Token	Action Taken	Operand Stack State
16	Push 16	[16]
4	Push 4	[16, 4]
/	Pop 4, Pop 16 → Compute $16/4 = 4$, Push 4	[4]
3	Push 3	[4, 3]
5	Push 5	[4, 3, 5]
*	Pop 5, Pop 3 → Compute $3 \times 5 = 15$, Push 15	[4, 15]
+	Pop 15, Pop 4 → Compute $4 + 15 = 19$, Push 19	[19]
7	Push 7	[19, 7]
-	Pop 7, Pop 19 → Compute $19 - 7 = 12$, Push 12	[12]

Final Numerical Result: 12.

Advanced Architectural Extensions & Industrial Systems Synthesis

In contemporary enterprise software engineering and distributed computational infrastructure, mastering Data Structures using C demands a rigorous integration of foundational algorithms with large-scale production architectures. Systems operate in high-availability, low-latency environments where computational efficiency, cache locality, and provable system reliability govern operational success.

1. Contemporary Engineering Dilemmas and Architectural Solutions

Engineering teams designing systems centered around Data Structures using C must systematically navigate fundamental computational trade-offs:

- **Time Complexity Optimization versus Space Consumption:** In high-throughput algorithmic pipelines related to Stacks and Parsing Engines, pre-computing intermediate lookup tables (memoization, spatial indices, hash tables) achieves near-instantaneous $O(1)$ query latency, but inflates

physical memory footprints. Conversely, in-place algorithms minimize RAM utilization at the expense of higher cycle counts. Modern architectures dynamically calibrate memory allocations using memory-mapped I/O and cache-aware data layouts.

- **Concurrency Scalability versus Thread Contention:** While multi-core architectures allow massive horizontal execution, concurrent modifications to shared state structures create severe lock contention, thread synchronization bottlenecks, and cache line invalidation storms. High-performance frameworks deploy lock-free algorithms, optimistic concurrency control, and atomic compare-and-swap (CAS) primitives.
- **Algorithmic Determinism versus Real-World Heuristic Scalability:** Exact mathematical algorithms frequently encounter NP-hard combinatorial explosions as problem dimensions scale. Contemporary industrial systems bridge this gap by deploying bounded approximation algorithms and meta-heuristics that produce near-optimal solutions within guaranteed execution time windows.

2. Empirical Comparative Framework

A structured comparative assessment clarifies operational distinctions across conventional and modern paradigms in Data Structures using C:

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Architecture Paradigm	Monolithic single-threaded architectures with tightly coupled state.	Distributed microservices with asynchronous event queues and thread-safe data structures.	Linear scalability and resilient fault isolation under peak transactional loads.
Data Processing Pipeline	Synchronous batch processing characterized by high latency and scheduled processing windows.	Real-time streaming pipelines (Apache Kafka, Flink) with sub-second event ingestion and processing.	Zero-latency decision intelligence and immediate anomaly response.
Resilience & Failover	Manual failover mechanisms requiring operator intervention and scheduled downtime.	Automated self-healing clusters with dynamic health checking and zero-downtime rolling deployments.	Continuous availability meeting strict 99.999% Service Level Agreements (SLAs).
System Telemetry & Auditing	Static text logging with periodic post-incident analysis.	Distributed tracing (OpenTelemetry), structured JSON metrics, and automated root-cause diagnostics.	Mean Time to Detection (MTTD) and Recovery (MTTR) reduced from hours to seconds.

3. Step-by-Step Implementation and Operational Governance Roadmap

Executing enterprise-grade implementations of Data Structures using C requires an organized four-phase execution roadmap:

- **Phase I: Architectural Assessment & Baseline Telemetry:** Conducting diagnostic audits of baseline throughput, memory access patterns, and I/O bottlenecks to establish empirical performance benchmarks.
- **Phase II: Algorithmic Architecture & Prototype Modeling:** Formulating mathematically sound algorithmic workflows, defining abstract interfaces, and constructing modular proof-of-concept prototypes with automated unit tests.

- **Phase III: High-Performance Production Deployment:** Optimizing cache locality, eliminating unnecessary memory allocations, and deploying through automated continuous deployment pipelines with canary releases.
- **Phase IV: Continuous Governance & Telemetry Auditing:** Implementing real-time telemetry dashboards tracking latency percentiles (P95, P99), thread contention rates, and memory saturation to drive continuous architectural refinement.

Industrial Risk Governance, Statutory Compliance & Failure Mode Analysis

Mission-critical systems must withstand rigorous scrutiny across operational, legal, and cyber-security dimensions. System architects deploy structured risk evaluation matrices to eliminate vulnerabilities across critical operational categories:

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
Algorithmic & Logic Risk	Infinite recursion, stack overflows, unhandled edge cases, pointer corruption.	Core dumps, segmentation faults, abrupt memory footprint spikes, unresponsive threads.	Formal loop invariant assertions, strict boundary validation, automated fuzz testing, static code analysis.
Concurrency & State Risk	Race conditions, deadlocks, livelocks, priority inversions in multi-threaded workflows.	Thread execution stalls, CPU starvation, lock wait time spikes in APM telemetry.	Deadlock detection algorithms, lock ordering hierarchies, lock-free concurrent collections.
Cybersecurity Risk	Buffer overflow exploits, unauthenticated API calls, injection vectors, man-in-the-middle attacks.	Abnormal network egress traffic, failed cryptographic handshake alerts, unauthorized memory writes.	Address Space Layout Randomization (ASLR), data execution prevention (DEP), mutual TLS (mTLS) encryption.
Statutory & Regulatory Risk	Non-compliance with national data protection regulations (DPDP Act, GDPR), audit trail gaps.	Unencrypted sensitive user data at rest, unlogged administrative data modifications.	Automated encryption (AES-256), immutable cryptographic audit logs, periodic external compliance audits.

Through systematic deployment of these engineering practices, rigorous mathematical modeling, and robust governance frameworks, modern computing professionals ensure the reliability, performance, and long-term resilience of critical digital infrastructure.

Advanced Architectural Extensions & Industrial Systems Synthesis

In contemporary enterprise software engineering and distributed computational infrastructure, mastering Data Structures using C demands a rigorous integration of foundational algorithms with large-scale production architectures. Systems operate in high-availability, low-latency environments where computational efficiency, cache locality, and provable system reliability govern operational success.

1. Contemporary Engineering Dilemmas and Architectural Solutions

Engineering teams designing systems centered around Data Structures using C must systematically navigate fundamental computational trade-offs:

- **Time Complexity Optimization versus Space Consumption:** In high-throughput algorithmic pipelines related to Data Structures using C, pre-computing intermediate lookup tables (memoization, spatial indices, hash tables) achieves near-instantaneous $O(1)$ query latency, but inflates physical memory footprints. Conversely, in-place algorithms minimize RAM utilization at the expense of higher cycle counts. Modern architectures dynamically calibrate memory allocations using memory-mapped I/O and cache-aware data layouts.
- **Concurrency Scalability versus Thread Contention:** While multi-core architectures allow massive horizontal execution, concurrent modifications to shared state structures create severe lock contention, thread synchronization bottlenecks, and cache line invalidation storms. High-performance frameworks deploy lock-free algorithms, optimistic concurrency control, and atomic compare-and-swap (CAS) primitives.
- **Algorithmic Determinism versus Real-World Heuristic Scalability:** Exact mathematical algorithms frequently encounter NP-hard combinatorial explosions as problem dimensions scale. Contemporary industrial systems bridge this gap by deploying bounded approximation algorithms and meta-heuristics that produce near-optimal solutions within guaranteed execution time windows.

2. Empirical Comparative Framework

A structured comparative assessment clarifies operational distinctions across conventional and modern paradigms in Data Structures using C:

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Architecture Paradigm	Monolithic single-threaded architectures with tightly coupled state.	Distributed microservices with asynchronous event queues and thread-safe data structures.	Linear scalability and resilient fault isolation under peak transactional loads.
Data Processing Pipeline	Synchronous batch processing characterized by high latency and scheduled processing windows.	Real-time streaming pipelines (Apache Kafka, Flink) with sub-second event ingestion and processing.	Zero-latency decision intelligence and immediate anomaly response.
Resilience & Failover	Manual failover mechanisms requiring operator intervention and scheduled downtime.	Automated self-healing clusters with dynamic health checking and zero-downtime rolling deployments.	Continuous availability meeting strict 99.999% Service Level Agreements (SLAs).
System Telemetry & Auditing	Static text logging with periodic post-incident analysis.	Distributed tracing (OpenTelemetry), structured JSON metrics, and automated root-cause diagnostics.	Mean Time to Detection (MTTD) and Recovery (MTTR) reduced from hours to seconds.

3. Step-by-Step Implementation and Operational Governance Roadmap

Executing enterprise-grade implementations of Data Structures using C requires an organized four-phase execution roadmap:

- **Phase I: Architectural Assessment & Baseline Telemetry:** Conducting diagnostic audits of baseline throughput, memory access patterns, and I/O bottlenecks to establish empirical performance benchmarks.
- **Phase II: Algorithmic Architecture & Prototype Modeling:** Formulating mathematically sound algorithmic workflows, defining abstract interfaces, and constructing modular proof-of-concept prototypes with automated unit tests.
- **Phase III: High-Performance Production Deployment:** Optimizing cache locality, eliminating unnecessary memory allocations, and deploying through automated continuous deployment pipelines with canary releases.
- **Phase IV: Continuous Governance & Telemetry Auditing:** Implementing real-time telemetry dashboards tracking latency percentiles (P95, P99), thread contention rates, and memory saturation to drive continuous architectural refinement.

Industrial Risk Governance, Statutory Compliance & Failure Mode Analysis

Mission-critical systems must withstand rigorous scrutiny across operational, legal, and cyber-security dimensions. System architects deploy structured risk evaluation matrices to eliminate vulnerabilities across critical operational categories:

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
Algorithmic & Logic Risk	Infinite recursion, stack overflows, unhandled edge cases, pointer corruption.	Core dumps, segmentation faults, abrupt memory footprint spikes, unresponsive threads.	Formal loop invariant assertions, strict boundary validation, automated fuzz testing, static code analysis.
Concurrency & State Risk	Race conditions, deadlocks, livelocks, priority inversions in multi-threaded workflows.	Thread execution stalls, CPU starvation, lock wait time spikes in APM telemetry.	Deadlock detection algorithms, lock ordering hierarchies, lock-free concurrent collections.
Cybersecurity Risk	Buffer overflow exploits, unauthenticated API calls, injection vectors, man-in-the-middle attacks.	Abnormal network egress traffic, failed cryptographic handshake alerts, unauthorized memory writes.	Address Space Layout Randomization (ASLR), data execution prevention (DEP), mutual TLS (mTLS) encryption.
Statutory & Regulatory Risk	Non-compliance with national data protection regulations (DPDP Act, GDPR), audit trail gaps.	Unencrypted sensitive user data at rest, unlogged administrative data modifications.	Automated encryption (AES-256), immutable cryptographic audit logs, periodic external compliance audits.

Through systematic deployment of these engineering practices, rigorous mathematical modeling, and robust governance frameworks, modern computing professionals ensure the reliability, performance, and long-term resilience of critical digital infrastructure.

Advanced Architectural Extensions & Industrial Systems Synthesis

In contemporary enterprise software engineering and distributed computational infrastructure, mastering Data Structures using C demands a rigorous integration of foundational algorithms with large-scale production architectures. Systems operate in high-availability, low-latency environments where computational efficiency, cache locality, and provable system reliability govern operational success.

1. Contemporary Engineering Dilemmas and Architectural Solutions

Engineering teams designing systems centered around Data Structures using C must systematically navigate fundamental computational trade-offs:

- **Time Complexity Optimization versus Space Consumption:** In high-throughput algorithmic pipelines related to Data Structures using C, pre-computing intermediate lookup tables (memoization, spatial indices, hash tables) achieves near-instantaneous $O(1)$ query latency, but inflates physical memory footprints. Conversely, in-place algorithms minimize RAM utilization at the expense of higher cycle counts. Modern architectures dynamically calibrate memory allocations using memory-mapped I/O and cache-aware data layouts.
- **Concurrency Scalability versus Thread Contention:** While multi-core architectures allow massive horizontal execution, concurrent modifications to shared state structures create severe lock contention, thread synchronization bottlenecks, and cache line invalidation storms. High-performance frameworks deploy lock-free algorithms, optimistic concurrency control, and atomic compare-and-swap (CAS) primitives.
- **Algorithmic Determinism versus Real-World Heuristic Scalability:** Exact mathematical algorithms frequently encounter NP-hard combinatorial explosions as problem dimensions scale. Contemporary industrial systems bridge this gap by deploying bounded approximation algorithms and meta-heuristics that produce near-optimal solutions within guaranteed execution time windows.

2. Empirical Comparative Framework

A structured comparative assessment clarifies operational distinctions across conventional and modern paradigms in Data Structures using C:

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Architecture Paradigm	Monolithic single-threaded architectures with tightly coupled state.	Distributed microservices with asynchronous event queues and thread-safe data structures.	Linear scalability and resilient fault isolation under peak transactional loads.
Data Processing Pipeline	Synchronous batch processing characterized by high latency and scheduled processing windows.	Real-time streaming pipelines (Apache Kafka, Flink) with sub-second event ingestion and processing.	Zero-latency decision intelligence and immediate anomaly response.
Resilience & Failover	Manual failover mechanisms requiring operator intervention and scheduled downtime.	Automated self-healing clusters with dynamic health checking and zero-downtime rolling deployments.	Continuous availability meeting strict 99.999% Service Level Agreements (SLAs).
System Telemetry &	Static text logging with periodic post-incident	Distributed tracing (OpenTelemetry), structured	Mean Time to Detection (MTTD) and Recovery

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Auditing	analysis.	JSON metrics, and automated root-cause diagnostics.	(MTTR) reduced from hours to seconds.

3. Step-by-Step Implementation and Operational Governance Roadmap

Executing enterprise-grade implementations of Data Structures using C requires an organized four-phase execution roadmap:

- **Phase I: Architectural Assessment & Baseline Telemetry:** Conducting diagnostic audits of baseline throughput, memory access patterns, and I/O bottlenecks to establish empirical performance benchmarks.
- **Phase II: Algorithmic Architecture & Prototype Modeling:** Formulating mathematically sound algorithmic workflows, defining abstract interfaces, and constructing modular proof-of-concept prototypes with automated unit tests.
- **Phase III: High-Performance Production Deployment:** Optimizing cache locality, eliminating unnecessary memory allocations, and deploying through automated continuous deployment pipelines with canary releases.
- **Phase IV: Continuous Governance & Telemetry Auditing:** Implementing real-time telemetry dashboards tracking latency percentiles (P95, P99), thread contention rates, and memory saturation to drive continuous architectural refinement.

Industrial Risk Governance, Statutory Compliance & Failure Mode Analysis

Mission-critical systems must withstand rigorous scrutiny across operational, legal, and cyber-security dimensions. System architects deploy structured risk evaluation matrices to eliminate vulnerabilities across critical operational categories:

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
Algorithmic & Logic Risk	Infinite recursion, stack overflows, unhandled edge cases, pointer corruption.	Core dumps, segmentation faults, abrupt memory footprint spikes, unresponsive threads.	Formal loop invariant assertions, strict boundary validation, automated fuzz testing, static code analysis.
Concurrency & State Risk	Race conditions, deadlocks, livelocks, priority inversions in multi-threaded workflows.	Thread execution stalls, CPU starvation, lock wait time spikes in APM telemetry.	Deadlock detection algorithms, lock ordering hierarchies, lock-free concurrent collections.
Cybersecurity Risk	Buffer overflow exploits, unauthenticated API calls, injection vectors, man-in-the-middle attacks.	Abnormal network egress traffic, failed cryptographic handshake alerts, unauthorized memory writes.	Address Space Layout Randomization (ASLR), data execution prevention (DEP), mutual TLS (mTLS) encryption.
Statutory & Regulatory Risk	Non-compliance with national data protection	Unencrypted sensitive user data at rest, unlogged	Automated encryption (AES-256), immutable cryptographic audit logs,

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
	regulations (DPDP Act, GDPR), audit trail gaps.	administrative data modifications.	periodic external compliance audits.

Through systematic deployment of these engineering practices, rigorous mathematical modeling, and robust governance frameworks, modern computing professionals ensure the reliability, performance, and long-term resilience of critical digital infrastructure.

Advanced Architectural Extensions & Industrial Systems Synthesis

In contemporary enterprise software engineering and distributed computational infrastructure, mastering Data Structures using C demands a rigorous integration of foundational algorithms with large-scale production architectures. Systems operate in high-availability, low-latency environments where computational efficiency, cache locality, and provable system reliability govern operational success.

1. Contemporary Engineering Dilemmas and Architectural Solutions

Engineering teams designing systems centered around Data Structures using C must systematically navigate fundamental computational trade-offs:

- **Time Complexity Optimization versus Space Consumption:** In high-throughput algorithmic pipelines related to Data Structures using C, pre-computing intermediate lookup tables (memoization, spatial indices, hash tables) achieves near-instantaneous $O(1)$ query latency, but inflates physical memory footprints. Conversely, in-place algorithms minimize RAM utilization at the expense of higher cycle counts. Modern architectures dynamically calibrate memory allocations using memory-mapped I/O and cache-aware data layouts.
- **Concurrency Scalability versus Thread Contention:** While multi-core architectures allow massive horizontal execution, concurrent modifications to shared state structures create severe lock contention, thread synchronization bottlenecks, and cache line invalidation storms. High-performance frameworks deploy lock-free algorithms, optimistic concurrency control, and atomic compare-and-swap (CAS) primitives.
- **Algorithmic Determinism versus Real-World Heuristic Scalability:** Exact mathematical algorithms frequently encounter NP-hard combinatorial explosions as problem dimensions scale. Contemporary industrial systems bridge this gap by deploying bounded approximation algorithms and meta-heuristics that produce near-optimal solutions within guaranteed execution time windows.

2. Empirical Comparative Framework

A structured comparative assessment clarifies operational distinctions across conventional and modern paradigms in Data Structures using C:

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Architecture Paradigm	Monolithic single-threaded architectures with tightly coupled state.	Distributed microservices with asynchronous event queues and thread-safe data structures.	Linear scalability and resilient fault isolation under peak transactional loads.

Evaluation Dimension	Conventional Traditional Approach	Contemporary Integrated Paradigm	Strategic Impact Metric
Data Processing Pipeline	Synchronous batch processing characterized by high latency and scheduled processing windows.	Real-time streaming pipelines (Apache Kafka, Flink) with sub-second event ingestion and processing.	Zero-latency decision intelligence and immediate anomaly response.
Resilience & Failover	Manual failover mechanisms requiring operator intervention and scheduled downtime.	Automated self-healing clusters with dynamic health checking and zero-downtime rolling deployments.	Continuous availability meeting strict 99.999% Service Level Agreements (SLAs).
System Telemetry & Auditing	Static text logging with periodic post-incident analysis.	Distributed tracing (OpenTelemetry), structured JSON metrics, and automated root-cause diagnostics.	Mean Time to Detection (MTTD) and Recovery (MTTR) reduced from hours to seconds.

3. Step-by-Step Implementation and Operational Governance Roadmap

Executing enterprise-grade implementations of Data Structures using C requires an organized four-phase execution roadmap:

- **Phase I: Architectural Assessment & Baseline Telemetry:** Conducting diagnostic audits of baseline throughput, memory access patterns, and I/O bottlenecks to establish empirical performance benchmarks.
- **Phase II: Algorithmic Architecture & Prototype Modeling:** Formulating mathematically sound algorithmic workflows, defining abstract interfaces, and constructing modular proof-of-concept prototypes with automated unit tests.
- **Phase III: High-Performance Production Deployment:** Optimizing cache locality, eliminating unnecessary memory allocations, and deploying through automated continuous deployment pipelines with canary releases.
- **Phase IV: Continuous Governance & Telemetry Auditing:** Implementing real-time telemetry dashboards tracking latency percentiles (P95, P99), thread contention rates, and memory saturation to drive continuous architectural refinement.

Industrial Risk Governance, Statutory Compliance & Failure Mode Analysis

Mission-critical systems must withstand rigorous scrutiny across operational, legal, and cyber-security dimensions. System architects deploy structured risk evaluation matrices to eliminate vulnerabilities across critical operational categories:

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
Algorithmic & Logic Risk	Infinite recursion, stack overflows, unhandled edge cases, pointer corruption.	Core dumps, segmentation faults, abrupt memory footprint spikes, unresponsive threads.	Formal loop invariant assertions, strict boundary validation, automated fuzz testing, static code analysis.

DegreeLive

Risk Category	Primary Sources of Vulnerability	Diagnostic Warning Indicators	Mitigation Protocol & Internal Controls
Concurrency & State Risk	Race conditions, deadlocks, livelocks, priority inversions in multi-threaded workflows.	Thread execution stalls, CPU starvation, lock wait time spikes in APM telemetry.	Deadlock detection algorithms, lock ordering hierarchies, lock-free concurrent collections.
Cybersecurity Risk	Buffer overflow exploits, unauthenticated API calls, injection vectors, man-in-the-middle attacks.	Abnormal network egress traffic, failed cryptographic handshake alerts, unauthorized memory writes.	Address Space Layout Randomization (ASLR), data execution prevention (DEP), mutual TLS (mTLS) encryption.
Statutory & Regulatory Risk	Non-compliance with national data protection regulations (DPDP Act, GDPR), audit trail gaps.	Unencrypted sensitive user data at rest, unlogged administrative data modifications.	Automated encryption (AES-256), immutable cryptographic audit logs, periodic external compliance audits.

Through systematic deployment of these engineering practices, rigorous mathematical modeling, and robust governance frameworks, modern computing professionals ensure the reliability, performance, and long-term resilience of critical digital infrastructure.

DegreeLive